Name: Parashara Ramesh	**Student Id**: A0285647M	**NUSNET Id**: E1216292

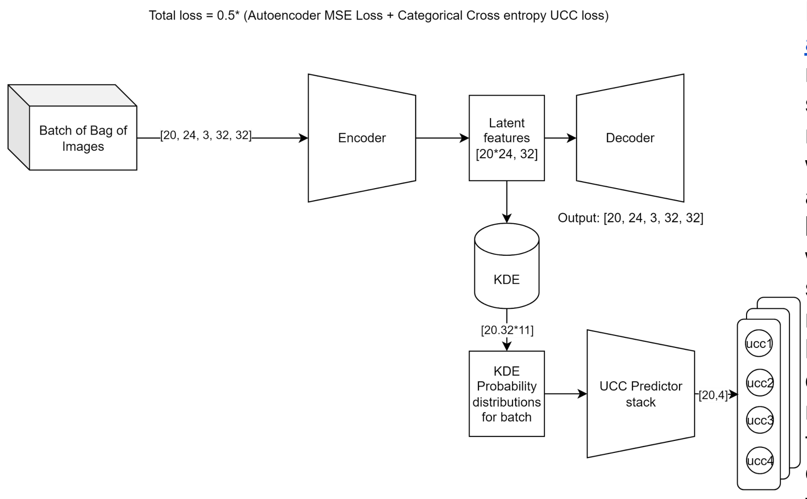
Introduction

The assignment involves reproducing results from a paper titled "**Weakly Supervised Clustering by Exploiting Unique Class Count.**" The paper proposes improving clustering accuracy by leveraging latent features from an autoencoder while jointly predicting the unique class count of each data bag; the model offers a comprehensive approach to enhance clustering precision.

The paper aims to train a model for labeling individual instances in a group of images, even when only bag level labels are provided. It's particularly useful for tasks like identifying cancer cells in images, treating the image as a bag and pixels as individual instances. The authors of the paper demonstrated mathematically that employing a Unique Class Count (UCC) setup improves clustering and is comparable to supervised methods. This showcases that even with weak supervision, the model is capable of effective classification.

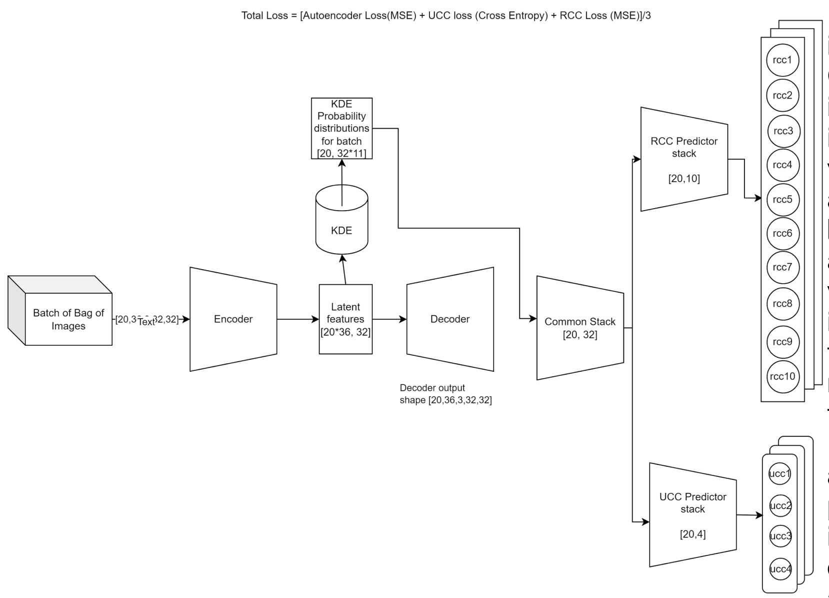
In a whimsical analogy, the paper's approach mirrors Superman using X-ray vision to classify the contents of a bag as opposed to individually checking each item, highlighting the model's capacity to discern and categorize individual instances even within a collection/bag of images.

UCC model


Total loss = 0.5* (Autoencoder MSE Loss + Categorical Cross entropy UCC loss)
Batch of Bag of Images
[20, 24, 3, 32, 32]
Encoder
Latent features [20*24, 32]
Decoder
Output: [20, 24, 3, 32, 32]
KDE
[20.32*11]
KDE Probability distributions for batch
UCC Predictor stack
[20,4]
ucc1
ucc2
ucc3
ucc4

My base model, inspired by the paper, has the following architecture. Unlike the paper, I chose a bag size of 24, a multiple of 1, 2, 3, and 4,(to ensure even splits) and a batch size of 20. The architecture utilizes a pretrained ResNext model as the encoder for robust latent features, a decoder with Wide Residual blocks & batch norm layers, along with a simple linear stack with LeakyReLU activation for UCC logits prediction. Training involves Adam optimizer with weight decay, gradient clipping, and a learning rate scheduler. Data augmentation techniques, such as random rotation and flips, ensure training data variance. After 8 hours and 80k steps, the model achieved a UCC accuracy of approximately 63%, clustering accuracy of 22%, and a minimum JS divergence of 0.00095. Despite challenges in training, a more extended duration on better hardware could potentially yield similar accuracy scores, as jointly training both objectives displayed a slow-gradual increase in accuracy over time.

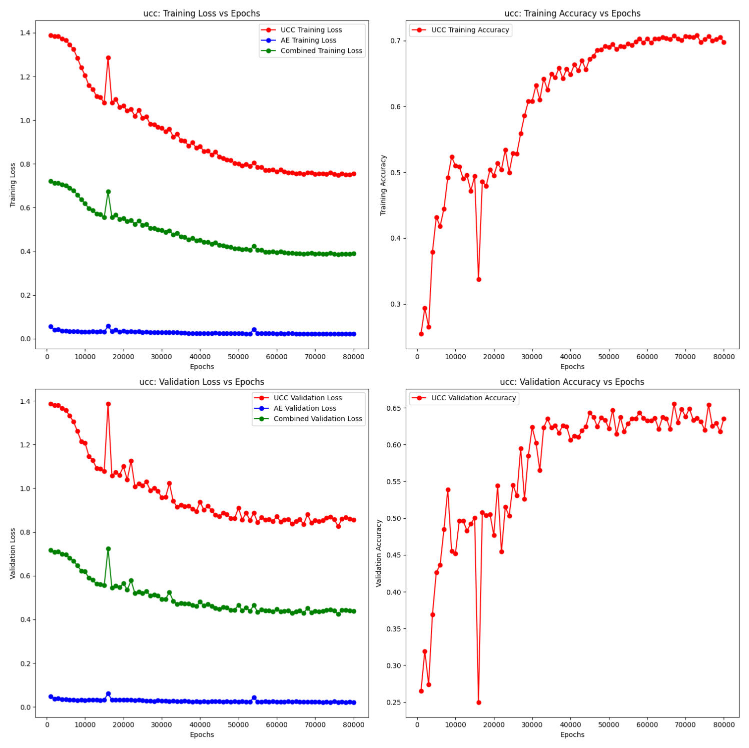
RCC model

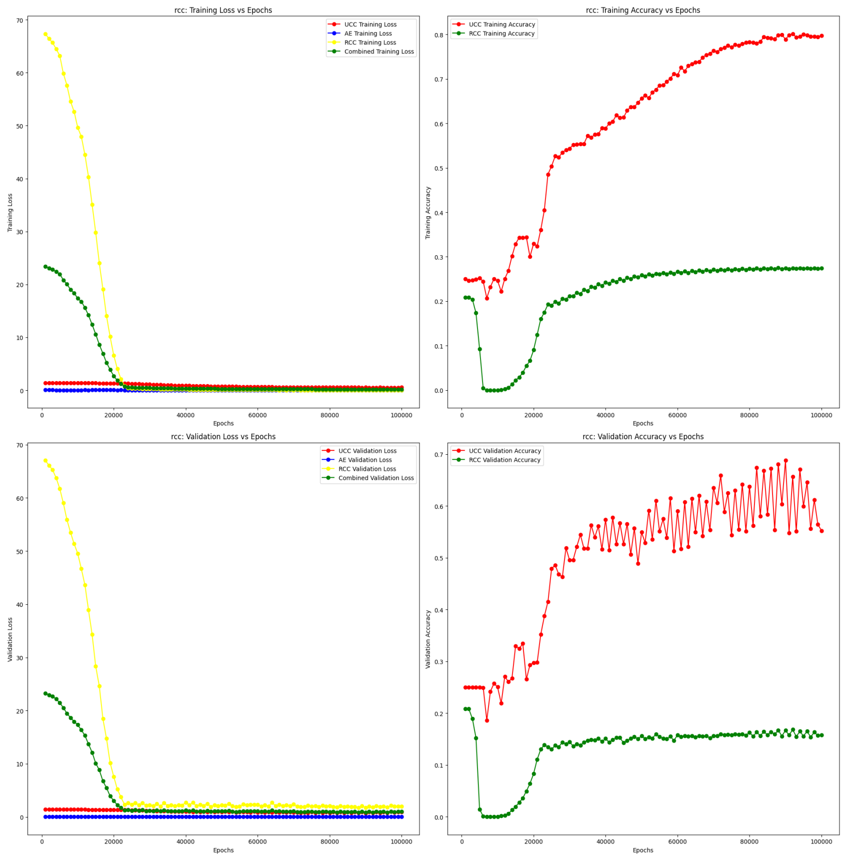

Total Loss = [Autoencoder Loss(MSE) + UCC loss (Cross Entropy) + RCC Loss (MSE)]/3
KDE Probability distributions for batch [20, 32*11]
KDE
Batch of Bag of Images
[20,36,3,32,32]
Text
Encoder
Latent features [20*36, 32]
Decoder
Decoder output shape [20,36,3,32,32]
Common Stack [20, 32]
RCC Predictor stack
[20,10]
rcc1
rcc2
rcc3
rcc4
rcc5
rcc6
rcc7
rcc8
rcc9
rcc10
UCC Predictor stack
[20,4]
ucc1
ucc2
ucc3
ucc4

In my enhanced model, I expand the UCC model by introducing an additional path to predict RCC logits (Real Class Counts for each class label within a bag). The goal is to jointly optimize all three tasks, assigning equal importance (0.33) to each loss function, akin to the alpha value. This modification maintains the same model architecture as before but concurrently predicts RCC logits. However I chose a different bag size of 36. For e.g a bag with a UCC count of 3 with classes 4, 6, 9; the RCC vector (size 10) would have values at the 4th, 6th, and 9th indices equal to bagsize/3, while the rest are 0. Same training techniques like gradient clipping and a learning rate scheduler are applied, introducing a new loss term, the RCC loss—a mean squared error between predicted RCC logits and the ground truth RCC vector. While an additional loss enforcing the same UCC count in predicted RCC logits could further enhance performance, it hasn't been implemented. Despite a UCC accuracy of only 56%, and an RCC accuracy of 27% this approach achieved the best clustering accuracy of 89% after 100k steps and 8 hours of training. This novel concept could theoretically outperform the original author's implementation when trained on superior computing resources for longer durations, serving as a proof of concept. The model also yielded the best minimum JS divergence of 0.2309. Another improvement approach which was discarded due to lack of time was using SSIM loss in the autoencoder as it is known to be a great loss function for learning latent features.

Results

Model	Clustering Accuracy	Min JS Divergence	Combined Training Loss	UCC Training Accuracy	RCC Training Accuracy	Combined Test Loss	UCC Test Accuracy	RCC Test Accuracy
UCC (Base)	22.115%	0.00095	**AE**: 0.0222 **UCC**: 0.755 **Total**:0.3389 =(ae+ucc)/2	69.775%	-	**AE**: 0.0219 **UCC**: 0.8479 **Total**:0.4349 =(ae+ucc)/2	63.81%	-
RCC(Improvement)	89.335%	0.2309	**AE**: 0.0307 **UCC**: 0.5410 **RCC**: 0.0326 **Total**: 0.1997 =(ae+ucc+rcc)/3	79.76%	27.44%	**AE**: 0.0337 **UCC**: 0.9598 **RCC**: 2.004 **Total**:1.00927 =(ae+ucc+rcc)/3	56.215%	15.68%





Left : UCC training plot(See my github for full plot) **Right**: RCC Training plot(See my github for full plot)

Conclusion

This project attempts to implement the original paper by reproducing the original results using my base model (UCC) along with an improvement model (RCC). However, the exact results could not be reproduced primarily due to lack of good computing resources and lack of training time needed to actually achieve the expected results as training the model for 2 whole days was not feasible from my end. That said, there were a lot of problems I encountered while attempting to implement this paper as I tried more than 15 to 16 different architecture changes all of which still failed to reproduce the same results.While the original idea of the paper to learn bag level instances for improving clustering is novel, the general consensus amongst the majority of my classmates is that the exact results are almost impossible to reproduce. Despite the lack of compute resources and time needed for training & slow convergence (see graphs above), my RCC model yielded a surprisingly high clustering accuracy indicating the potential of this approach as a suitable improvement by taking this idea of weakly supervised clustering one step further. The last section lists the majority of the problems I personally faced while implementing this assignment.

Problems Faced in this assignment

Here are some of the many problems I personally encountered during this assignment. On discussing with my classmates, I found that many other students were also struggling to reproduce the same results either due to lack of computing resources (or) due to lack of clarity in the original paper & code implementation. Therefore, it is my recommendation that such an assignment should not be given in the future as I personally had to spend > $50 SGD just to get computing resources needed to train the two models and still fell short of reproducing the exact results. If such an assignment is to be given , I would urge the professor to also give the required computing resources needed for training the models.

Problem	**Comments**
Lack of powerful computing resources	Training on my laptop's GPU resulted in Out of memory errors due to the size of the training data and training on the free tier of Colab resulted in my runtime being disconnected after training for a few hours. Even after saving the model after every 1k steps, the model training when restarting from that same point would worsen its loss and accuracy as it is a common observation in Pytorch
NAN loss	When creating a UCC bag with randomized images, it very quickly resulted in ucc loss going to NaN values. In order to fix this, I had to restructure my bag so that each of the labels in the ucc bag are in the same proportion and in order. The fact that this fixed my loss issue indicates that the original implementation of how a bag should be is in essence flawed and not generic enough.
UCC Accuracy stuck at 25% + very slow convergence	After implementing a multitude of models, I found that the UCC accuracy was always stuck at 25% . Initially this was due to a bug in my loss calculation, but despite fixing this , the same problem emerged yet again and the accuracy was stuck at this 25% mark for nearly 4 hours after which it improved. On closer analysis, I found that the model was only rapidly learning the autoencoder loss while the ucc loss was stuck at 1.39. This did not change despite my trying of different alpha values such as 0.2 for autoencoder and 0.8 for ucc loss.
KDE not working if latent feature size is more	The kernel density estimator had very low gradients in all of my experiments because of which the ucc stack was not learnable at all. Another issue I noticed was that the latent feature input size fo KDE had to be really small(e.g. 10 or 32) for the KDE gradients to be decent enough to backpropogate the losses. This obviously impacted the autoencoder loss as the CIFAR-10 dataset is a complex dataset which cannot purely be reconstructed from such small latent features. Taking an effort to improve the autoencoder by changing the latent feature size ensured that UCC loss was not learnable because of which there is a tradeoff needed with respect to the latent feature size.
Pretraining only the autoencoder separately	In my efforts to make the ucc model learn. I also tried training an autoencoder separately and experimented with both freezing & unfreezing the autoencoder weights in the combined model while also experimenting with the alpha values as I wanted to give a higher importance to the ucc predictor instead of my autoencoder in an effort to make it learn. All these experiments also failed and showed no real improvement as my ucc loss was still stuck for a long time. It was only after I let it train for nearly 8 hours that I noticed that the ucc loss changed around the 4 hour mark indicating that the original paper approach is clearly not reproducible nor powerful enough.
UNet Autoencoder	In my efforts to improve my autoencoder I implemented a Unet Autoencoder. This approach learnt a near perfect autoencoder with a loss as low as 0.0008 but resulted in the UCC model still being stuck at 25% accuracy even after 6 hours of training. This observation could be because of the fact that my gradients were getting short-circuited due to the unet skip connections because of which there was no gradient flow in the ucc predictor stack resulting in no changes in the ucc loss while training the combined model.
Trying out other pretrained vision models as my encoder	I additionally also tried other pretrained models in my encoder like Vision Transformer. But ViT was discarded due to incredibly slow training time (80+hours!). Therefore other pretrained models like Resnet , VGG were also explored each of which still gave similar problems due to the incredibly slow convergence of the model
Training for > 10 hours still results in only gradual improvement	Based on all of my experiments, my only conclusion is that reproducing the original results is only possible if the model is trained continuously for many days on very powerful computing hardware as the graph of loss vs steps indicates a very gradual & very slow convergence.