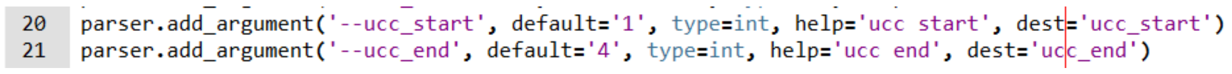
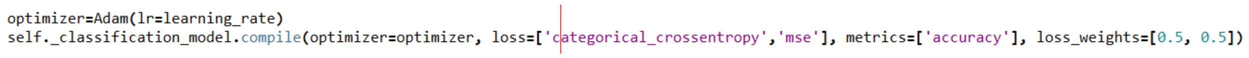
BS6207

For this question, I used the UCC code to recreate Table 1 from the paper by modifying the corresponding code in model.py to train the different models. As we know from the article, while UCC and UCC^{2+} models had autoencoder branch in their architecture; they were optimized jointly over autoencoder loss and ucc loss, UCC$^{2+}_{\alpha=1}$and UCC$_{\alpha=1}$models did not have an autoencoder branch in their architecture and were optimized over ucc loss only, and UCC and UCC$_{\alpha=1}$models were trained on bags with labels of ucc1 to ucc4, UCC^{2+} and UCC$^{2+}_{\alpha=1}$models were trained on bags with labels ucc2 to ucc4. Thus for UCC^{2+} and UCC$^{2+}_{\alpha=1}$I modified the train label count from ucc1 to ucc2 in the train.py file.



```
parser.add_argument('--ucc_start', default='1', type=int, help='ucc start', dest='ucc_start')
parser.add_argument('--ucc_end', default='4', type=int, help='ucc end', dest='ucc_end')
```

As for UCC$^{2+}_{\alpha=1}$and UCC$_{\alpha=1}$, we know that 'ucc loss' is cross-entropy loss and 'autoencoder loss' is mean square error loss. Thus, I deleted the 'mse' loss function from the model in the model.py file.



```
optimizer=Adam(lr=learning_rate)
self._classification_model.compile(optimizer=optimizer, loss=['categorical_crossentropy','mse'], metrics=['accuracy'], loss_weights=[0.5, 0.5])
```

Due to the fact that epochs = 100000 takes a very long time to run, I changed the number of epochs to 1500 and got the following result.

	Mnist min JS divergence	**Mnist Ucc acc.**	**Mnist Clustering acc**
UCC	0.557	0.85	0.834
UCC^{2+}	0.618	0.718	0.796
UCC$_{\alpha=1}$	0.591	0.739	0.84
UCC$^{2+}_{\alpha=1}$	0.195	0.684	0.573

Since we decreased the number of epochs dramatically, the result accuracy also dropped a lot compared to the result from the article. However, it did follow a similar trend as in the article. Compare UCC and UCC^{2+}If we decrease the number of the label and start from a label forward, the UCC accuracy decreases from 0.85 to 0.739, but the cluster accuracy increases from 0.834 to 0.84. The reason why UCC accuracy may decrease is very likely because of an insufficient amount of the number of epochs. However, we can conclude from the increase in clustering accuracy that the higher the label, the more accurate the model is. Compare UCC and $\text{UCC}_{\alpha=1}$ we can see that both accuracies dropped significantly from 0.8+ to 0.7+, which suggests that the lack of one loss function has a significant impact on the model performance, which means that the mean squared error function is tightly related to the model.

In addition to the four suggested models, I also tried to use different training set sizes to train the model and observe the result. I recorded the result of training size equal to 20000, 5000 and 500 respectively.

Different Training Size:

	Mnist min JS divergence	**Mnist Ucc acc.**	**Mnist Clustering acc**
60000	0.557	0.85	0.834
20000	0.548	0.665	0.74
5000	0.693	0.512	0.499
500	Unable to get a result due to cannot to proceed the training procedure		

From the above result, we can see a general trend such that as the number of training sizes decreases, the accuracy decreases, and it becomes harder to converge, as, in the end, the model was unable to provide a result for training size equal to 500 due to the size is too small.

However, one phenomenon that needs to be highlighted is that when training size is equal to 20000, the clustering accuracy is actually higher than the UCC accuracy, which suggests that the model is quite overfitting at that point.

What's more, I also tried to train the model with different activation functions; I tried to use ReLU and sigmoid for model training and observed the different results.

Different Activation Function:

	Mnist min JS divergence	**Mnist Ucc acc.**	**Mnist Clustering acc**
ReLU	0.557	0.85	0.834
Sigmoid	0.693	0.545	0.512

From the above result, we can see that compared to ReLU, Sigmoid produce a much worse result. By defination, ReLU can overcome the gradient vanishing problem. It is more computationally efficient to compute than Sigmoid functions since Relu needs to pick max(0, x) and not perform expensive exponential operations as in Sigmoids.

As a result, the original model from the paper is able to deliver the best result at the most efficient computational speed.