Rubika Interview Exam Report

Hamid Mohammadi - 09395191963

Question 1:

My initial impression was to use a pre-trained backbone and fine-tune it on the CIFAR10 dataset. I have evaluated some backbones from **Keras Applications,** such as ResNet50, MobineNet v1, and v2, DenseNet 121, and EfficientDet B0. **The best results were achieved using the DenseNet 121 with an accuracy of 95% and an f1-score of 88% on average.**

However, implementing a CNN from scratch proved to be more efficient for the following reasons:

1. Firstly, pre-trained models are trained on much larger images than the ones in the CIFAR10 dataset. The CIFAR10 dataset images are 32x32x3 which is much smaller than recommended input sizes for the Keras Applications models which are larger than 192x192x3. This fact is the main reason why the frozen pre-trained models have lower accuracy and f1-score in our case (92% accuracy and 85% f1-score). In contrast, un-freezing the backbone, allows the model to gain the aforementioned results.
2. Secondly, using a pre-trained model requires more processing power as the pre-trained models are at least 4 million parameters in size. A custom CNN would have around 300 thousand parameters. The much smaller model is preferred here.

Therefore, a custom CNN model was designed. The model has 3 convolutional cells with two 3 by 3 convolutional layers, a 2 by 2 max-pooling layer, and a batch normalization layer. Extracted features are then classified using a simple MLP with 2 outputs, the same as the number of required classes. The MLP is using a standard relu activation function for its hidden layers and softmax for the output layer.

Two 3 by 3 convolutional layers are used instead of a 5 by 5 convolutional layer to reduce the number of parameters in the model. And batch normalization is used as a means to apply regularization on the network.

Sparse categorical cross-entropy is used as the loss function, as the labels are integers. The alternative method could be to convert the integer labels to one-hot vectors. The use of sparse categorical cross-entropy could be useful for memory preservation.

Custom models summary is displayed below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 14, 32)        128
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 10, 10, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
batch_normalization_4 (Batch (None, 5, 5, 64)          256
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 3, 3, 128)         73856
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 1, 1, 128)         147584
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)               512
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 288,162
Trainable params: 287,714
Non-trainable params: 448
```

Question 2:

Useful evaluation metrics for this application are precision, recall, and f1-score. The use of an accuracy metric is not recommended here as classes are imbalanced and subpar results are expected at the class level. The confusion matrix could also be interesting here. The classification results are presented in the table below:

Type	Accuracy	Class	Precision	Recall	F1-score	Confusion matrix
0 vs others	95	0	78	73	76	[[733 267] [201 8799]]
		Others	97	98	97	
		Macro	88	86	87	
1 vs others	98	1	92	88	90	[[880 120] [73 8927]]
		Others	99	99	99	
		Macro	96	94	95	
2 vs others	94	2	71	59	65	[[593 407] [239 8761]]
		Others	96	97	96	
		Macro	83	78	81	
3 vs others	91	3	57	61	59	[[612 388] [465 8535]]
		Others	96	95	95	
		Macro	76	78	77	
4 vs others	95	4	79	68	73	[[682 318] [184 8816]]
		Others	97	98	97	
		Macro	88	83	85	
5 vs others	94	5	73	65	69	[[666 334] [264 8736]]
		Others	96	97	97	
		Macro	85	81	83	
6 vs others	96	6	84	76	80	[[768 232] [135 8865]]
		Others	97	98	98	
		Macro	91	87	89	
7 vs others	96	7	78	81	78	[[757 243] [151 8849]]
		Others	98	98	97	
		Macro	90	87	89	
8 vs others	97	8	87	82	85	[[815 185]

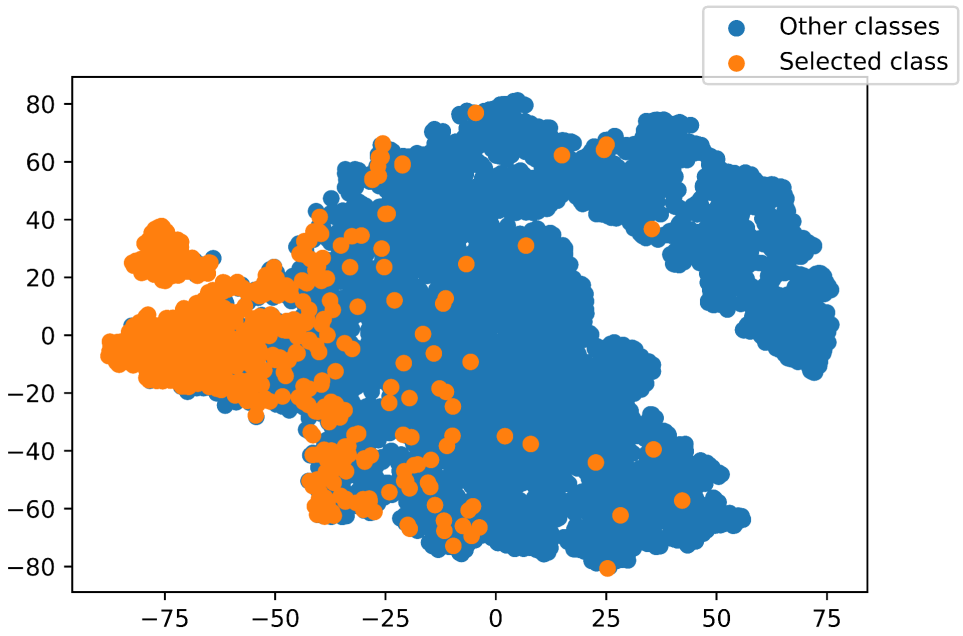
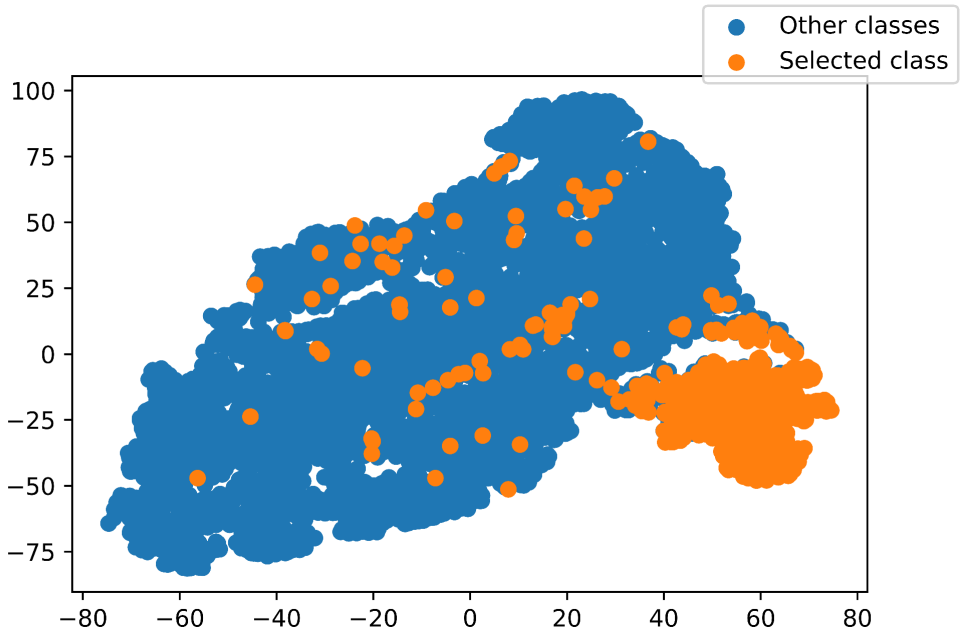
		Others	98	99	98	[88 8912]]
		Macro	96	87	92	
9 vs others	97	9	92	80	85	[[797 203] [70 8930]]
		Others	98	99	98	
		Macro	95	89	92	

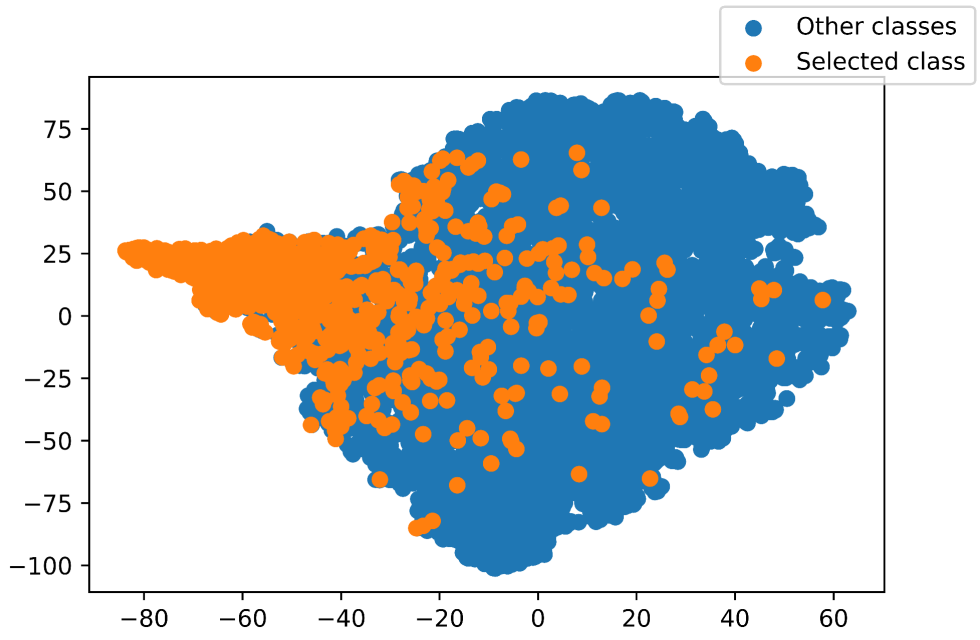
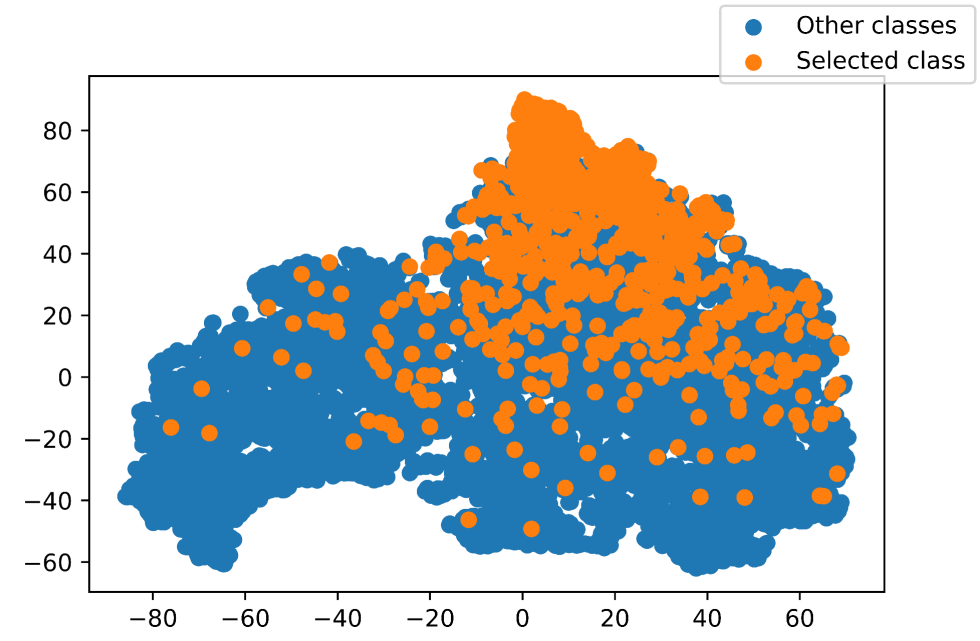
The above results are achieved by setting the loss function to **sparse categorical focal loss.** The imbalance in data has caused the model to have a low f1-score in the smaller class. The use of the focal loss slightly (around 2 percent) increases the overall f1-score. The class weight method is also used, with results similar to the focal loss results. Yet, I gave up this method as its convergence rate was much slower than the focal loss method.

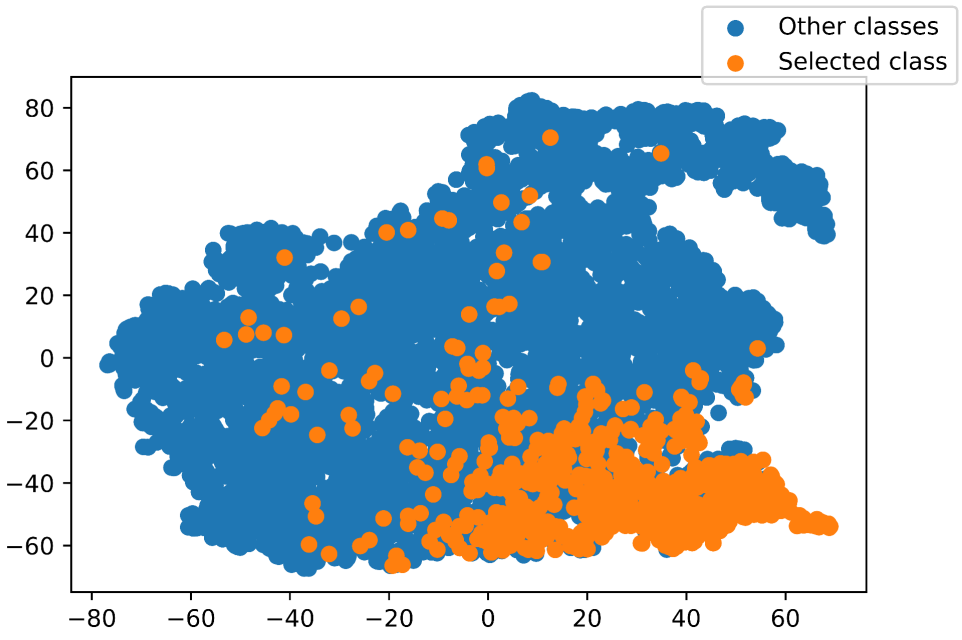
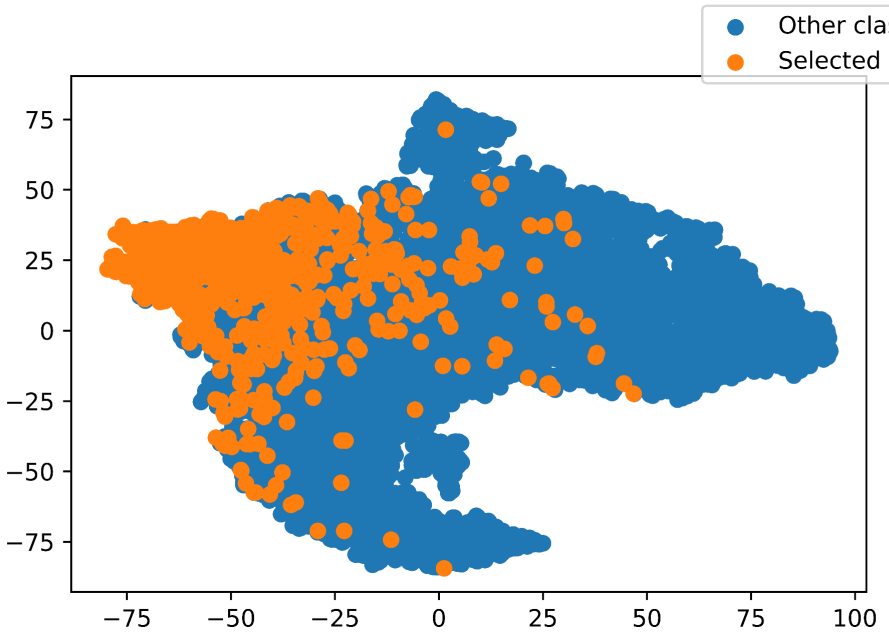
Question 3:

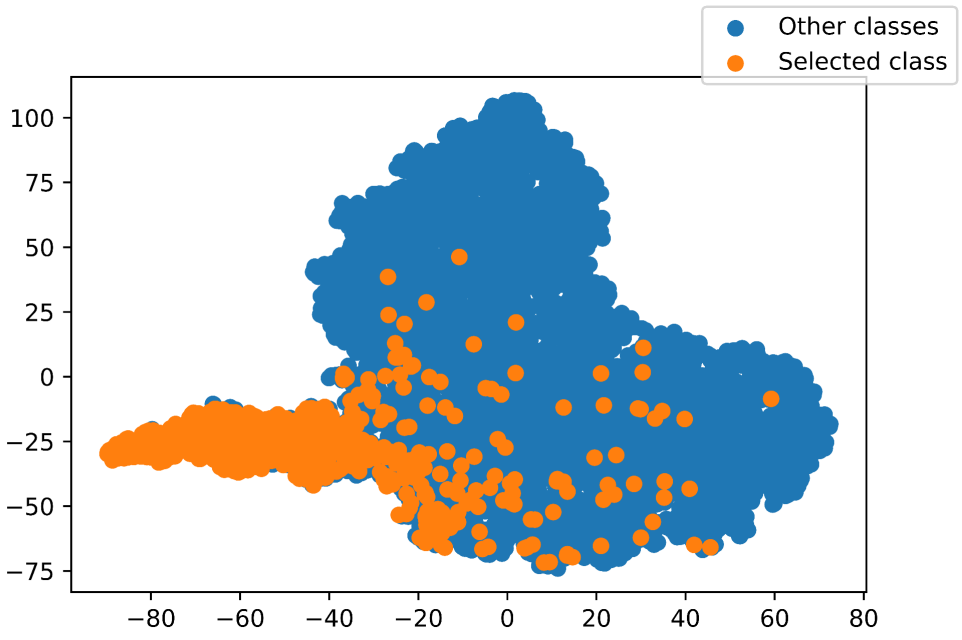
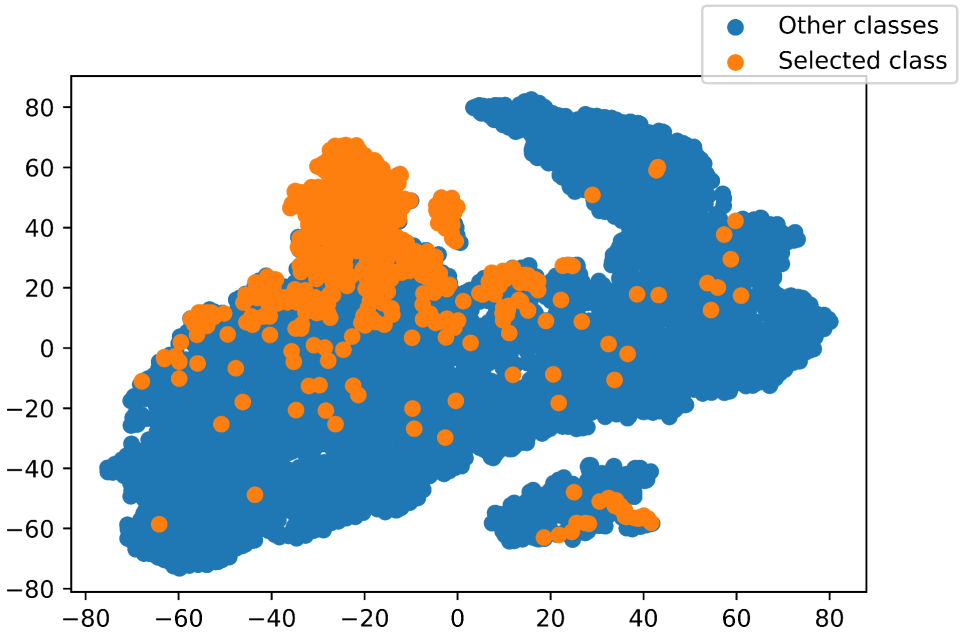
The features are extracted from the last hidden layer in the custom-trained model. I have used TSN-e for dimensionality reduction. I am aware of PCA or LDA, yet I think TSN-e is better for visualization purposes. The features are converted from a 128D space to a 2D space using TSN-e. The 2D features are visualized below:

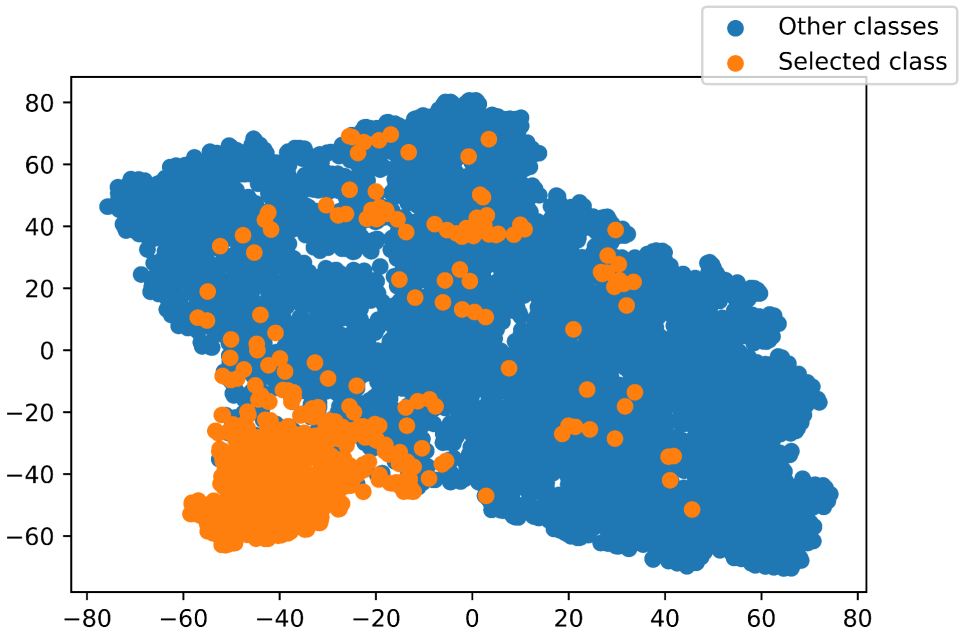
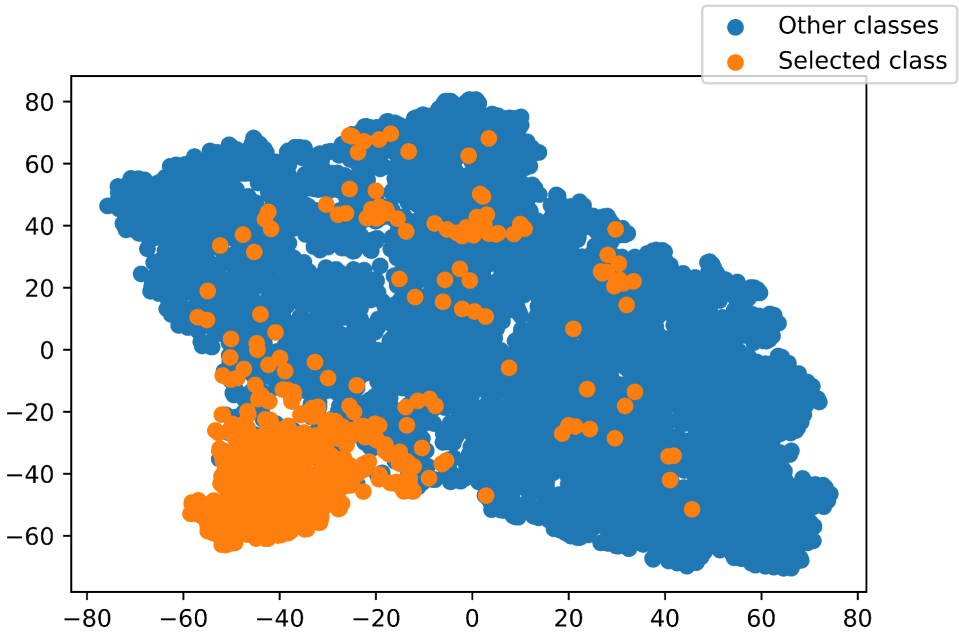
Type	**Image**

0 vs others	
1 vs others	

2 vs others	
3 vs others	

4 vs others	
5 vs others	

6 vs others	
7 vs others	

8 vs others	
9 vs others	

Visualization with TSN-e clearly shows the lack of clear discrimination between classes which has caused the “not perfect” classification results.