

Delft University of Technology

Deep Learning

CS4240

**Reproduction of paper ‘Invariant Information
Clustering for Unsupervised Image Classification
and Segmentation’**

Authors:

Dina Chen (4730712)

Zhiyi Wang (5310938)

April 16, 2021



1. Introduction

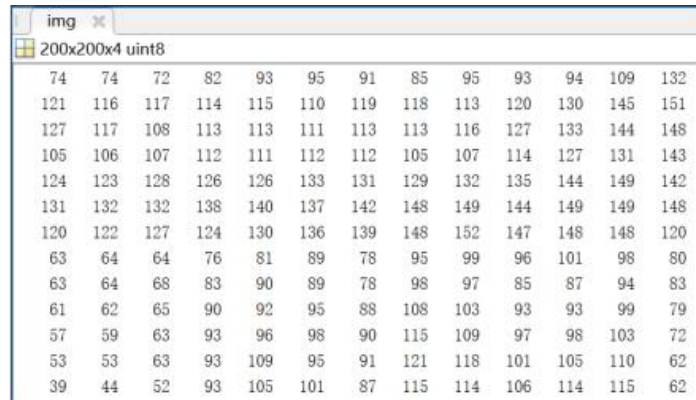
In the paper, the authors introduce a new unsupervised deep learning method--Invariant Information Clustering (IIC).[1] IIC directly trains a randomly initialized neural network into a classification function, end-to-end and without any labels. It involves a simple objective function, which is the mutual information between the function's classifications for paired data samples.

IIC can be applied to image clustering as well as image segmentation. In this project, we reproduce the segmentation part. In the image segmentation process, we do some transformations(e.g. flipping, colorjitting) to each image and add displacements to each pixel.

2. Prepare data

In the reproduction project, we use Potsdam dataset[2] in training. The dataset consists of 5400 different satellite landscape images. There are two main reasons for us to choose this dataset. One is that the size of the Potsdam dataset is relatively small, which means less computation required. Another reason is that there's no need to select images out of the dataset by the percentage of "stuff pixel" like Ccostuff. Before applying the data to the model, we do some preparation work.

The Potsdam dataset is in the format of .mat files. It is divided into a training set and ground truth set. For training set, each picture file is a 200*200*4 matrix, which represents 200*200 pixels and R, G, B, IR value for each pixel. The ground truth set consists of 200*200*1-dimension .mat files, which represents 200*200 pixels with label 1-6 on each pixel.



74	74	72	82	93	95	91	85	95	93	94	109	132
121	116	117	114	115	110	119	118	113	120	130	145	151
127	117	108	113	113	111	113	113	116	127	133	144	148
105	106	107	112	111	112	112	105	107	114	127	131	143
124	123	128	126	126	133	131	129	132	135	144	149	142
131	132	132	138	140	137	142	148	149	144	149	149	148
120	122	127	124	130	136	139	148	152	147	148	148	120
63	64	64	76	81	89	78	95	99	96	101	98	80
63	64	68	83	90	89	78	98	97	85	87	94	83
61	62	65	90	92	95	88	108	103	93	93	99	79
57	59	63	93	96	98	90	115	109	97	98	103	72
53	53	63	93	109	95	91	121	118	101	105	110	62
39	44	52	93	105	101	87	115	114	106	114	115	62

Figure 1 Part of data in a single img file

	1	2	3	4	5	6
1	2	2	2	2	2	2
2	2	2	2	2	2	2
3	0	0	2	2	2	2
4	0	0	0	2	2	2
5	0	0	0	0	2	2
6	0	0	0	0	0	2
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0
11	0	0	0	0	0	0
12	0	0	0	0	0	0

Figure 2 Part of data in a single gt file

For data augmentation, we apply horizontal flip to each picture in the training set by flipping along its symmetry axis. Also, we apply the colorjitter function to the training set. All four parameters (brightness, contrast, saturation and hue) are set to 0.1 according to their implementation.

3. Model

A	B	C
1×Conv@64	1× Conv@64	1× Conv@64
3×BasicBlock@64	1× MaxPool	1× Conv@128
4×BasicBlock@128	1× Conv@128	1× MaxPool
6×BasicBlock@256	1× MaxPool	2× Conv@256
3×BasicBlock@512	1× Conv@256	2× Conv@512
1×AvgPool	1× MaxPool	
	1× Conv@512	

Figure 3 Model selection

Network architectures given in supplement material. By segmentation, VGG network C, which is shown in Figure 3 is used.

We use the model created in the original IIC, which can be found in SegmentationNet10a class in the original code. As some parameters such as the kernel size and stride are already predefined and hardcoded in the code, what we need to do is correctly define the input channels (4)and output channels (3 or 6).

In the paper and supplement material they mentioned the benefit of overclustering, we follow this instruction and use SegmentationNet10aTwoHead class, it is a network with two output layer, one with the output size equals the amount of class label (3 for Potsdam 3 and 6 for Potsdam), the other with 3 times larger output size. In this case

we can calculate 2 losses, we average the two losses.

4. Our original implementation

In the beginning, we tried to follow the IIC paper’s step and reproduced the code strictly by the equations presented in the paper. However, it is too slow to run on the full Potsdam data set. But anyway, as a part of our project, it is still worth explaining our original implementation.

In the process, we performed pixel-by-pixel comparison by computing the outer product of two vectors into a matrix, and then took the average. After that, we computed such a matrix for each image in one batch and took the average. Since each batch consisted of 2 transformation--flipping and colorjitting, so the same process should be done for each transformation. Then we calculated the loss for the matrix we have so far. This procedure applied to function 5 in the IIC paper, and the way of computing mutual information is described in function 3 of the paper.

$$I(z, z') = I(\mathbf{P}) = \sum_{c=1}^C \sum_{c'=1}^C \mathbf{P}_{cc'} \cdot \ln \frac{\mathbf{P}_{cc'}}{\mathbf{P}_c \cdot \mathbf{P}_{c'}}. \quad (3)$$

$$\max_{\Phi} \frac{1}{|T|} \sum_{t \in T} I(\mathbf{P}_t), \quad (5)$$

$$\mathbf{P}_t = \frac{1}{n|G||\Omega|} \sum_{i=1}^n \sum_{g \in G} \sum_{u \in \Omega} \overbrace{\Phi_u(\mathbf{x}_i) \cdot [g^{-1}\Phi(g\mathbf{x}_i)]_{u+t}^\top}^{\text{Convolution}}.$$

Figure 4 Equations we used originally to calculate loss function[1]

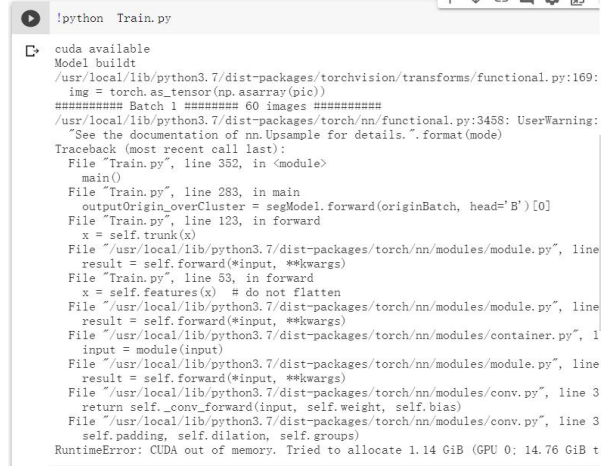
The goal is to maximize the information between each pixel $\Phi_u(\mathbf{x}_i)$ and the patch label $[g^{-1}\Phi(g\mathbf{x}_i)]_{u+t}$ of its transformed neighbour patch, in expectation over images $i = 1, \dots, n$, patches $u \in \Omega$ within each image, and perturbations $g \in G$. Information is in turn averaged over all neighbour displacements $t \in T$. We assumed the displacement process was moving one pixel to 8 directions (E, W, S, N, NE, NW, SE, SW), so loss calculation needs to be done for each of them. [1]

While training, we first implemented a small demo with 16 images and ran it on Google Colab, and then ran the whole dataset. However, due to the large amount of computation and storage, The RAM got full immediately, which led to a decrease of computing speed. It is approximated that training the whole dataset may take a few weeks.



Figure 5 Full RAM

Even worse, the cuda failed to work properly when we ran the model on the whole dataset. Normally, the batch size should be 60. However, the cuda ran out of memory when the batch size was larger than 10. So we finally had to change the computing method and applied the loss function in the authors' code.



```
python Train.py
cuda available
Model build:
/usr/local/lib/python3.7/dist-packages/torchvision/transforms/functional.py:169:
img = torch.as_tensor(np.asarray(pic))
##### Batch 1 ##### 60 images #####
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:3458: UserWarning:
"See the documentation of nn.Upsample for details.".format(mode)
Traceback (most recent call last):
  File "Train.py", line 352, in <module>
    main()
  File "Train.py", line 283, in main
    outputOrigin_overCluster = segModel.forward(originBatch, head='B')[0]
  File "Train.py", line 123, in forward
    x = self.trunk(x)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line
    result = self.forward(*input, **kwargs)
  File "Train.py", line 53, in forward
    x = self.features(x) # do not flatten
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line
    result = self.forward(*input, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/container.py", line
    input = module(input)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line
    result = self.forward(*input, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py", line 3
    return self._conv_forward(input, self.weight, self.bias)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py", line 3
    self.padding, self.dilation, self.groups)
RuntimeError: CUDA out of memory. Tried to allocate 1.14 GiB (GPU 0: 14.76 GiB t
```

Figure 6 Cuda out of memory

We adapted the author's loss function called `IID_segmentation_loss`, which takes the output (class probability distribution) of two batches (one for the original images and the other for the transformed images) as input and calculates the information as output. We can not understand how this function works exactly since it is implemented in a smart and complex way, with this function we don't suffer from the full RAM problem above anymore. Besides, the loop through all the transformations and displacements seems to be done in this one function, but we do not understand how exactly and question whether it implements equation 5 strictly. These confusions were also the reason why we were uncomfortable to use their loss function directly at the beginning, and only adapt to it in the end.

Although the RAM problem has been solved, the problem with batch size still exists. We can not run size 75 for Potsdam-6 (or 60 for Potsdam- 3), instead, the maximum batch size we can run without getting "Cuda out of memory error" is 20. In their experiments, they run thousands of epochs, which is also what we are unable to, seeing the time we have left.

Finally, we decided to limit the batch size to 20. However, something unexpected happened. The GPU of Google colab always stop working in the midway of our validation process, which directly led to failure to calculate the final accuracy. The most probable reason might be system failure.

5. Result

Unfortunately, we are not able to fully reproduce the paper. It is partly due to hardware issues like failure of cuda and Google colab, and also because the original code provided by the authors of IIC paper is quite messy. However, the good thing is that our loss function goes on the right way towards optimization. Figure7 shows the trend of loss function. Since the loss is based on mutual information, the larger absolute value this loss reaches, the better effect it could be.

```

Done batch 2
0##### Batch 3 ##### 20 images #####
tensor(-0.4380, device='cuda:0', grad_fn=<DivBackward0>)
Done batch 3
0##### Batch 4 ##### 20 images #####
tensor(-0.7115, device='cuda:0', grad_fn=<DivBackward0>)
Done batch 4
0##### Batch 5 ##### 20 images #####
tensor(-0.6834, device='cuda:0', grad_fn=<DivBackward0>)
Done batch 5
0##### Batch 6 ##### 20 images #####
tensor(-0.7606, device='cuda:0', grad_fn=<DivBackward0>)
Done batch 6
0##### Batch 7 ##### 20 images #####
tensor(-0.7867, device='cuda:0', grad_fn=<DivBackward0>)
Done batch 7

0##### Batch 424 ##### 20 images #####
tensor(-1.3300, device='cuda:0', grad_fn=<DivBackward0>)
Done batch 424
0##### Batch 425 ##### 20 images #####
tensor(-1.3614, device='cuda:0', grad_fn=<DivBackward0>)
Done batch 425
0##### Batch 426 ##### 20 images #####
tensor(-1.3947, device='cuda:0', grad_fn=<DivBackward0>)
Done batch 426
0##### Batch 427 ##### 20 images #####
tensor(-1.3140, device='cuda:0', grad_fn=<DivBackward0>)
Done batch 427
0##### Batch 428 ##### 20 images #####
tensor(-1.3790, device='cuda:0', grad_fn=<DivBackward0>)
Done batch 428

```

Figure7 Trend of loss function

In future days, we plan to find out the problems and try to fully reproduce the paper if time is available.

6. Task distribution

Dina Chen

1. Getting Potsdam data ready as input to the model
2. Data transformation: flip and color jitter
3. Reproduce equation 5 and equation 3 from the paper (calculate the loss from output of model)
4. Adapt model and loss function from the original code
5. Run experiments

Zhiyi Wang

1. Study and pre-process the datasets (both CocoStuff and Potsdam)
2. Run our model on Potsdam3 dataset
3. Write validation function to calculate accuracy based on ground truth and the trained result

References

[1] Ji X, Henriques J F, Vedaldi A. Invariant information clustering for unsupervised image classification and segmentation[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019: 9865-9874.

[2] Potsdam Dataset

<https://www.robots.ox.ac.uk/~xuji/datasets/Potsdam.tar.gz>

Appendix

Github link of our reproduction code:

<https://github.com/DinaChen/IIC-segmentation>

Github link of the original authors' code:

<https://github.com/xu-ji/IIC>