Chiranjeev
P21CS007
chiranjeev.1@iitj.ac.in

Programming Assignment - 3

Question1:

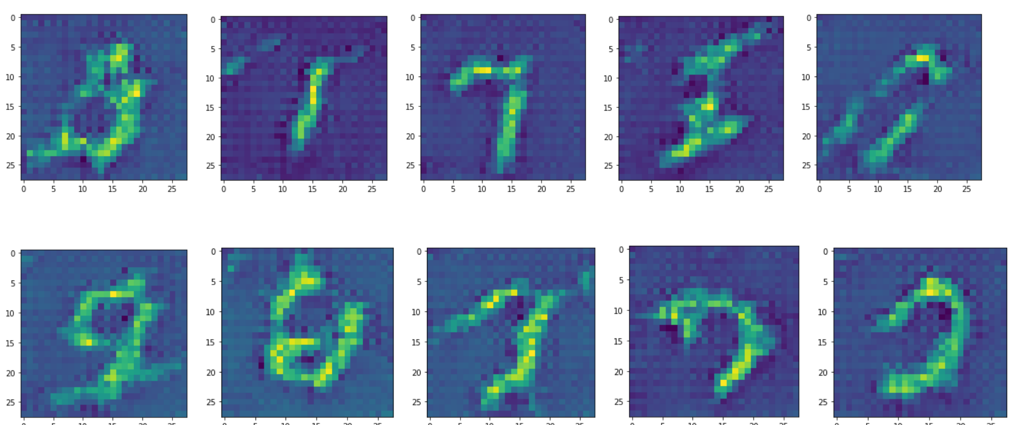
Dataset Used: MNIST

(A) Implementation of DCGAN

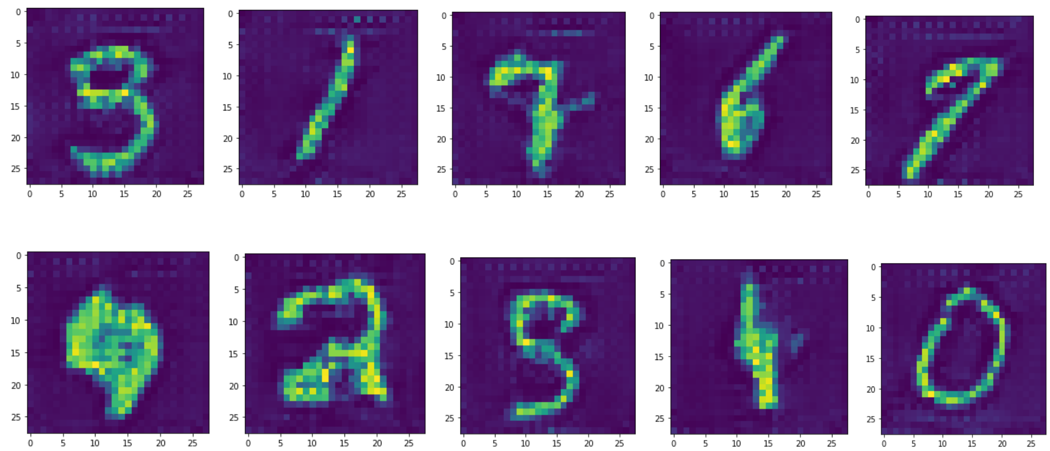
Discriminator Used: Resnet18

Results:

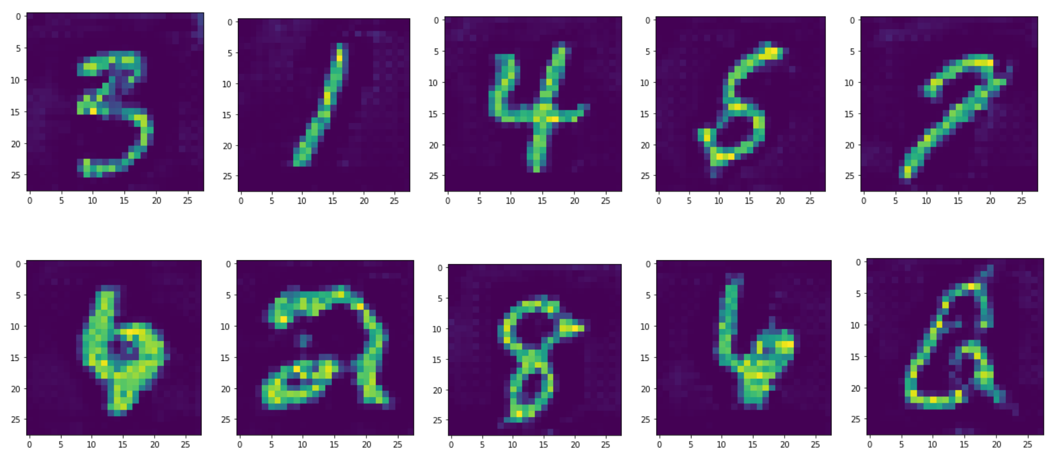
i. Image Generated after 1st epoch



ii. Image Generated after(n/2) 10th epoch



iii. Image Generated after last 20th epoch

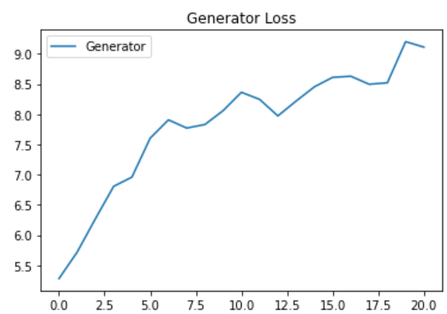


Observation:

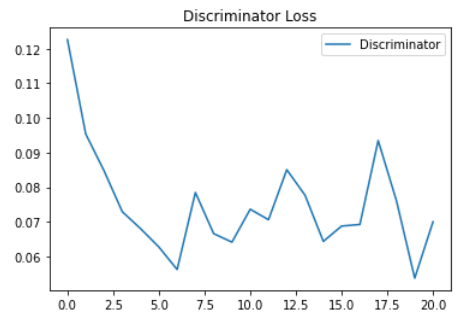
We can observe that not clear image is generated after 1st epoch the image has many artificats as in 1st epoch Generator has not learnt much. But as we can observe that as epoch continue Generator tries to learn greatly and produce good images and trying to produce images which can fool the discriminator by looking real. After 20th epoch Generator has generated improved quality images some of which may fake the discriminator.

(B) **Loss graphs**

Generator Loss



Discriminator Loss

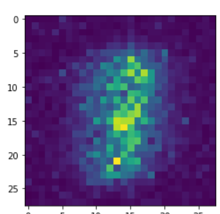
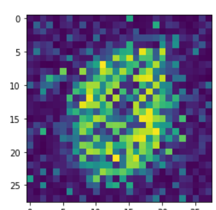
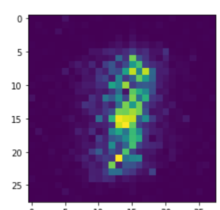
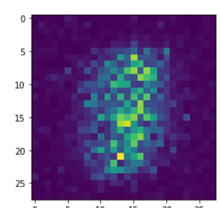
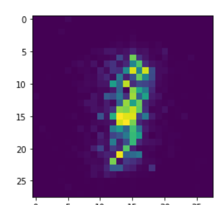


(C) In above model we can’t control that which class the generator can produce images. But we can make control on that by using Conditional GANs. The Conditional GANs make the generator to produce class images which it has been has asked to by giving particular input as label along with the input vector.

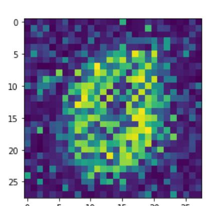
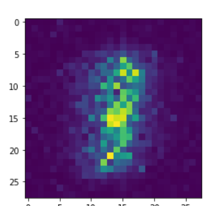
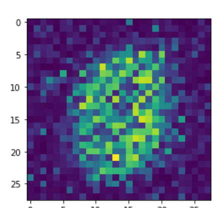
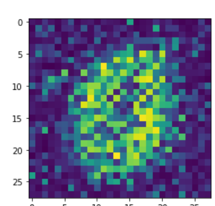
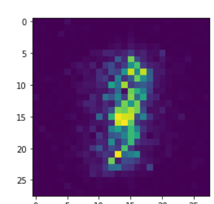
Results:

i. Image Generated after 1st epoch

Label: 9 3 1 4 7

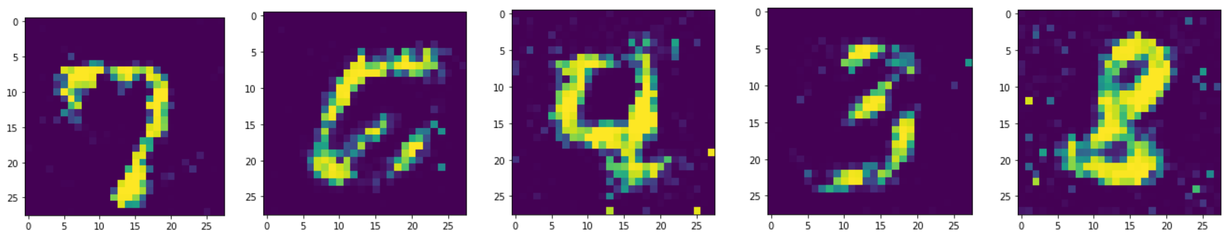






Label: 8 5 8 8 1

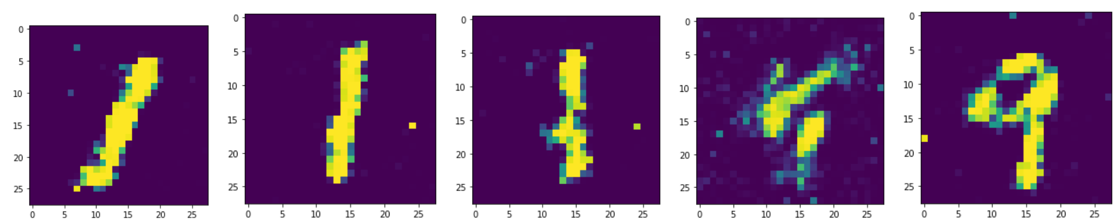






ii. Image Generated after(n/2) 10th epoch

Label: 7 5 4 3 8

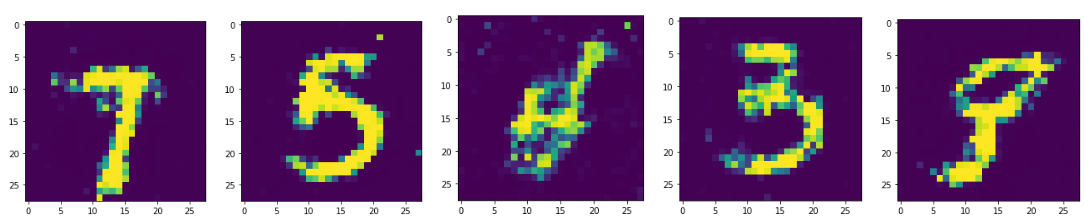


Label: 1 1 1 4 9

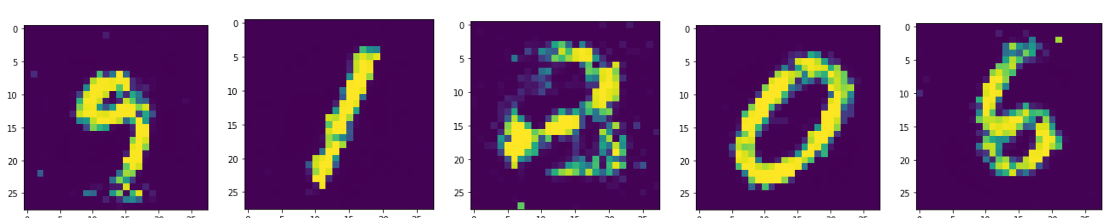


iii. Image Generated after last 20th epoch

Label: 7 5 4 3 8



Label: 9 1 2 0 6



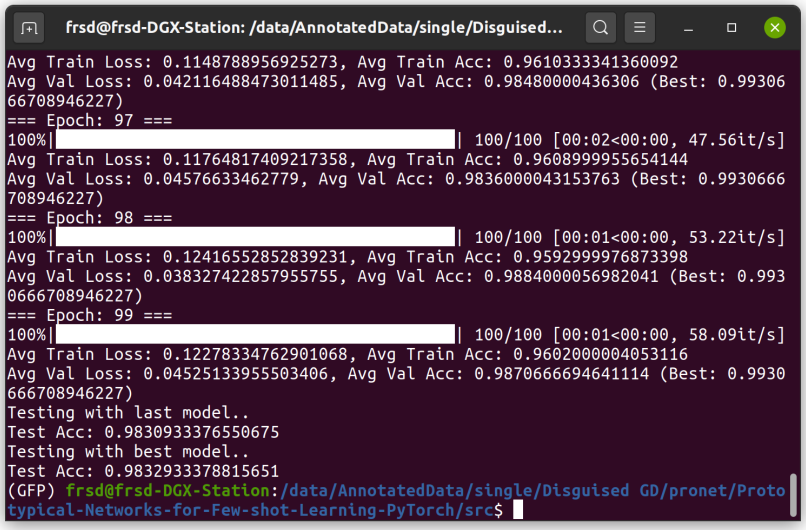
Question2:

“Prototypical Networks for Few-shot Learning”

Reproduced results of paper on Dataset: **Omniglot**

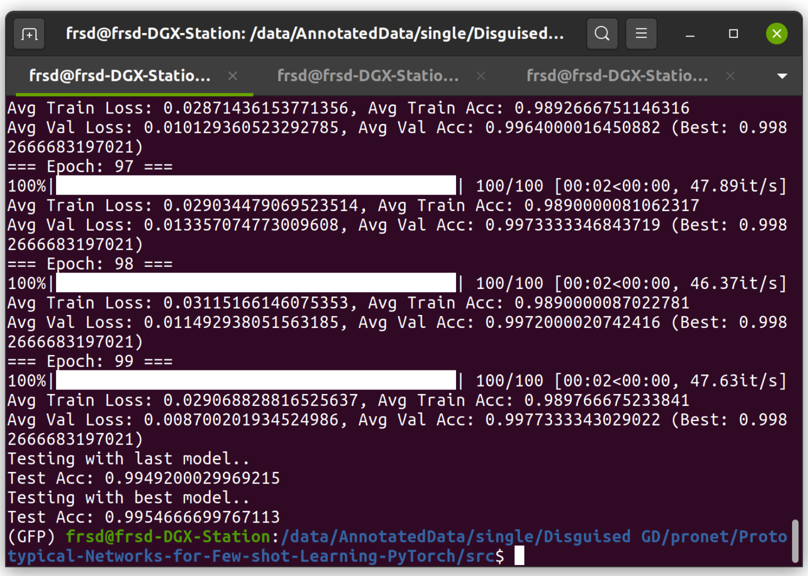
Results:

1-shot (5-way)



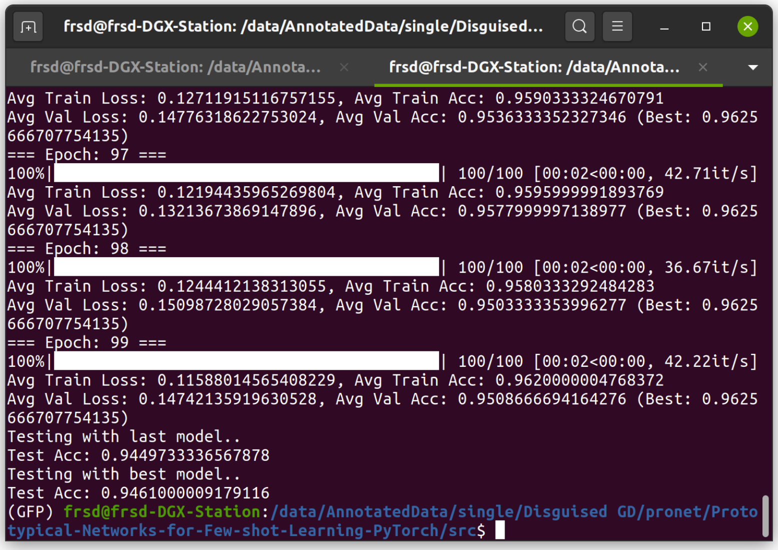
```
frsd@frsd-DGX-Station: /data/AnnotatedData/single/Disguised...
Avg Train Loss: 0.1148788956925273, Avg Train Acc: 0.9610333341360092
Avg Val Loss: 0.042116488473011485, Avg Val Acc: 0.98480000436306 (Best: 0.99306
66708946227)
=== Epoch: 97 ===
100%|                                        | 100/100 [00:02<00:00, 47.56it/s]
Avg Train Loss: 0.11764817409217358, Avg Train Acc: 0.9608999955654144
Avg Val Loss: 0.04576633462779, Avg Val Acc: 0.9836000043153763 (Best: 0.9930666
708946227)
=== Epoch: 98 ===
100%|                                        | 100/100 [00:01<00:00, 53.22it/s]
Avg Train Loss: 0.12416552852839231, Avg Train Acc: 0.9592999976873398
Avg Val Loss: 0.038327422857955755, Avg Val Acc: 0.9884000056982041 (Best: 0.993
0666708946227)
=== Epoch: 99 ===
100%|                                        | 100/100 [00:01<00:00, 58.09it/s]
Avg Train Loss: 0.12278334762901068, Avg Train Acc: 0.9602000004053116
Avg Val Loss: 0.04525133955503406, Avg Val Acc: 0.9870666694641114 (Best: 0.9930
666708946227)
Testing with last model..
Test Acc: 0.9830933376550675
Testing with best model..
Test Acc: 0.9832933378815651
(GFP) frsd@frsd-DGX-Station:/data/AnnotatedData/single/Disguised GD/pronet/Proto
typical-Networks-for-Few-shot-Learning-PyTorch/src$
```

5-shot (5-way)



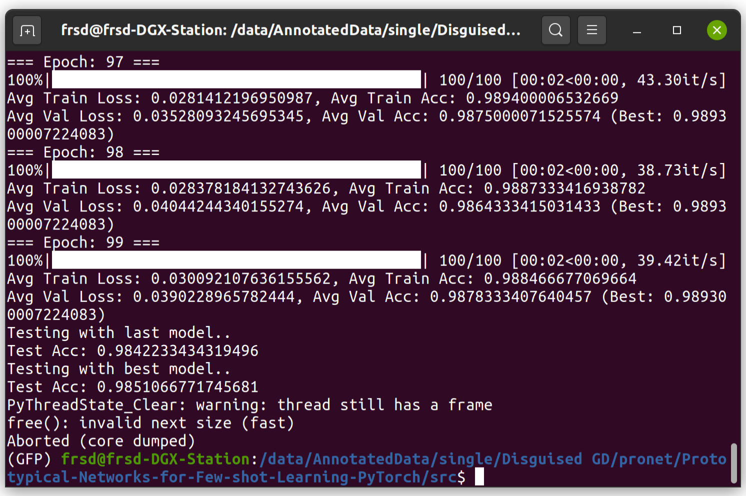
```
frsd@frsd-DGX-Station: /data/AnnotatedData/single/Disguised...
frsd@frsd-DGX-Statio...   frsd@frsd-DGX-Statio...   frsd@frsd-DGX-Statio...
Avg Train Loss: 0.02871436153771356, Avg Train Acc: 0.9892666751146316
Avg Val Loss: 0.010129360523292785, Avg Val Acc: 0.9964000016450882 (Best: 0.998
2666683197021)
=== Epoch: 97 ===
100%|                                        | 100/100 [00:02<00:00, 47.89it/s]
Avg Train Loss: 0.02903447906952351 4, Avg Train Acc: 0.9890000081062317
Avg Val Loss: 0.013357074773009608, Avg Val Acc: 0.9973333346843719 (Best: 0.998
2666683197021)
=== Epoch: 98 ===
100%|                                        | 100/100 [00:02<00:00, 46.37it/s]
Avg Train Loss: 0.03115166146075353, Avg Train Acc: 0.9890000087022781
Avg Val Loss: 0.011492938051563185, Avg Val Acc: 0.9972000020742416 (Best: 0.998
2666683197021)
=== Epoch: 99 ===
100%|                                        | 100/100 [00:02<00:00, 47.63it/s]
Avg Train Loss: 0.029068828816525637, Avg Train Acc: 0.989766675233841
Avg Val Loss: 0.008700201934524986, Avg Val Acc: 0.9977333343029022 (Best: 0.998
2666683197021)
Testing with last model..
Test Acc: 0.9949200029969215
Testing with best model..
Test Acc: 0.9954666699767113
(GFP) frsd@frsd-DGX-Station:/data/AnnotatedData/single/Disguised GD/pronet/Proto
typical-Networks-for-Few-shot-Learning-PyTorch/src$
```

1 -shot (20-way)



```
frsd@frsd-DGX-Station: /data/AnnotatedData/single/Disguised...
frsd@frsd-DGX-Station: /data/Annota...    frsd@frsd-DGX-Station: /data/Annota...
Avg Train Loss: 0.12711915116757155, Avg Train Acc: 0.9590333324670791
Avg Val Loss: 0.14776318622753024, Avg Val Acc: 0.9536333352327346 (Best: 0.9625
666707754135)
=== Epoch: 97 ===
100%|                                        | 100/100 [00:02<00:00, 42.71it/s]
Avg Train Loss: 0.12194435965269804, Avg Train Acc: 0.9595999991893769
Avg Val Loss: 0.13213673869147896, Avg Val Acc: 0.9577999997138977 (Best: 0.9625
666707754135)
=== Epoch: 98 ===
100%|                                        | 100/100 [00:02<00:00, 36.67it/s]
Avg Train Loss: 0.1244412138313055, Avg Train Acc: 0.9580333292484283
Avg Val Loss: 0.15098728029057384, Avg Val Acc: 0.9503333353996277 (Best: 0.9625
666707754135)
=== Epoch: 99 ===
100%|                                        | 100/100 [00:02<00:00, 42.22it/s]
Avg Train Loss: 0.11588014565408229, Avg Train Acc: 0.9620000004768372
Avg Val Loss: 0.14742135919630528, Avg Val Acc: 0.9508666694164276 (Best: 0.9625
666707754135)
Testing with last model..
Test Acc: 0.9449733336567878
Testing with best model..
Test Acc: 0.9461000009179116
(GFP) frsd@frsd-DGX-Station:/data/AnnotatedData/single/Disguised GD/pronet/Proto
typical-Networks-for-Few-shot-Learning-PyTorch/src$
```

5-shot (20-way)



```
frsd@frsd-DGX-Station: /data/AnnotatedData/single/Disguised...
=== Epoch: 97 ===
100%|                                        | 100/100 [00:02<00:00, 43.30it/s]
Avg Train Loss: 0.0281412196950987, Avg Train Acc: 0.989400006532669
Avg Val Loss: 0.03528093245695345, Avg Val Acc: 0.9875000071525574 (Best: 0.9893
00007224083)
=== Epoch: 98 ===
100%|                                        | 100/100 [00:02<00:00, 38.73it/s]
Avg Train Loss: 0.028378184132743626, Avg Train Acc: 0.9887333416938782
Avg Val Loss: 0.04044244340155274, Avg Val Acc: 0.9864333415031433 (Best: 0.9893
00007224083)
=== Epoch: 99 ===
100%|                                        | 100/100 [00:02<00:00, 39.42it/s]
Avg Train Loss: 0.030092107636155562, Avg Train Acc: 0.988466677069664
Avg Val Loss: 0.0390228965782444, Avg Val Acc: 0.9878333407640457 (Best: 0.98930
0007224083)
Testing with last model..
Test Acc: 0.9842233434319496
Testing with best model..
Test Acc: 0.9851066771745681
PyThreadState_Clear: warning: thread still has a frame
free(): invalid next size (fast)
Aborted (core dumped)
(GFP) frsd@frsd-DGX-Station:/data/AnnotatedData/single/Disguised GD/pronet/Proto
typical-Networks-for-Few-shot-Learning-PyTorch/src$
```

Model	1-shot (5-way Acc.)	5-shot (5-way Acc.)	1 -shot (20-way Acc.)	5-shot (20-way Acc.)
Values Reported in paper	98.8%	99.7%	96.0%	98.9%
Values after reproducing paper	98.32%	99.54%	94.61%	98.51%

Analysis:

Above table provides reproduced results on the given dataset “Omlicon”. The model is trained for 100 epochs, which gives an accuracy similar to the results reported in the paper for different types N-way K-shot classification. Also, best model is returned after training for all epochs. Authors uses prototypical network to classify Omniglot data samples by calculating distance between prototype representations with others inorder to exibit Inductive-bias technique.

Explaination of Algorithm:

As we know few shot classification is type of learning technique whenever there is less training data available. Few shot learning is a kind pof meta-learning in which the model is trained on several related tasks, during the training phase. It is trained in such a way that it can perform well on non-seen classes.

This algorithm represents that there exist an embedding in which points cluster around a single prototype representation for every class. It has k-way n-shot type of training. Here we have k classes and for each class we have 2 set : support set and query set.

Algorithm steps:

1. Flatten the images to be transformd into 1-D vectors.
2. Then the class prototypes are computed by forming the clusters and each cluster is represented by a centroid.
3. The embeddings of the support set images are averaged to form a class prototype.
4. Now distance is computed between the queries and prototypes. In this the metric choice is very important, the authors have specified there own choice of distance metric.
5. After computing the distance softmax is performed over distances to the prototypes inorder to get probabilities.

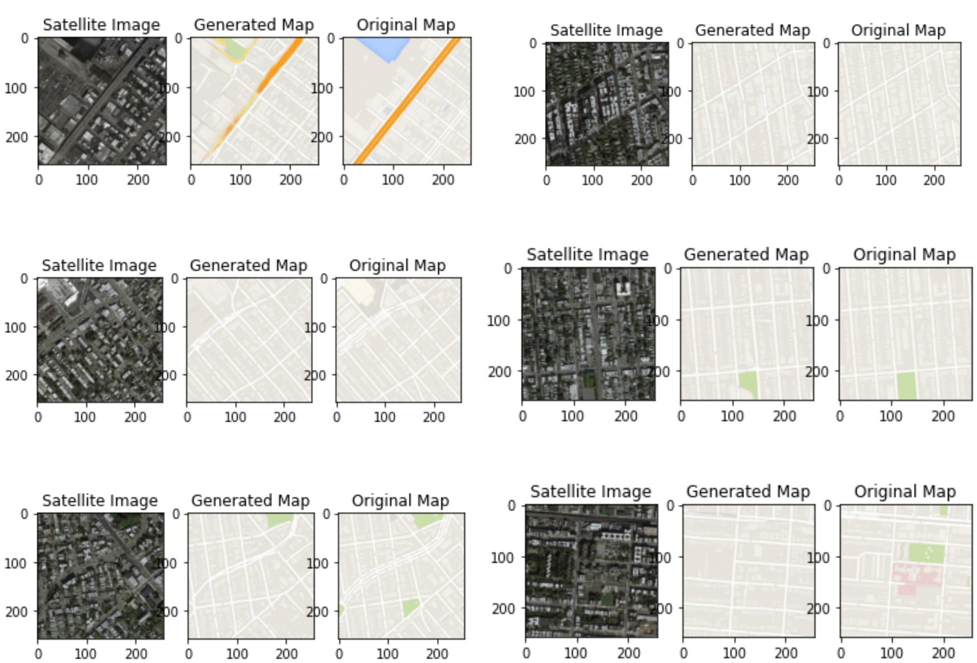
6. Now the classes are classified on the basis of probability scores. The higher the score more the chance of the query sample belonging to that class.
7. Finally the loss is calculated and backpropagated.

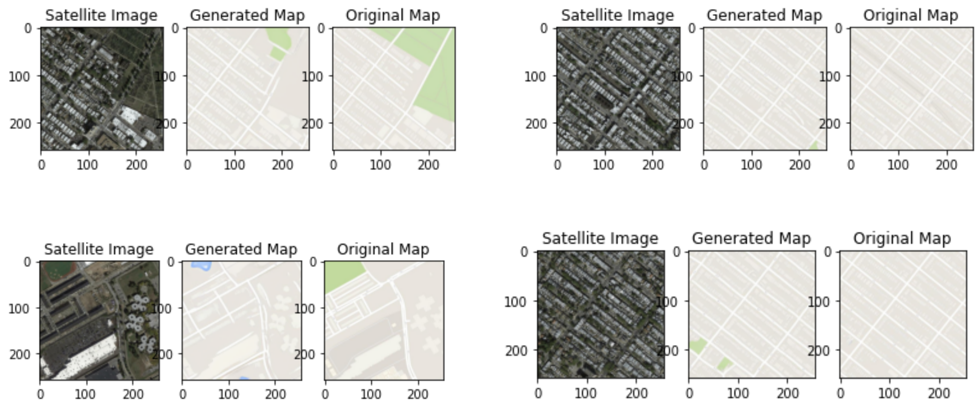
Pseudo code for Reproducing the results of this paper

- The embedding part takes a (28x28x3) image and returns a column vector of length 64.
- The image2vector function is composed of 4 modules.
- Each module consists of various layers: conv layer, batch norm layer, ReLU activation function and 2x2 max pooling layer.
- **Optimizer:** Adam
- **learning rate:** 0.0001
- The model is trained for 2000 episodes for every 5th epoch. training is performed by randomly picking new sample in the training set at each episode.
- It is tested on 1000 episodes.

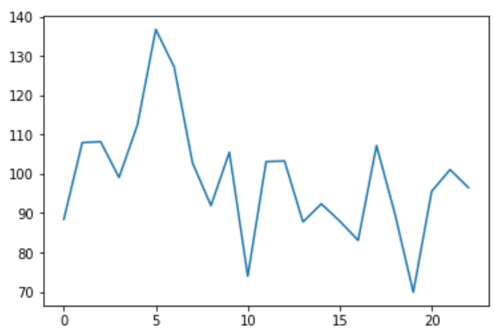
Question3:

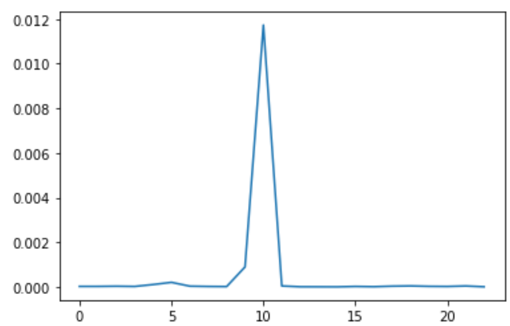
(a) Trained GAN generated results are as follows





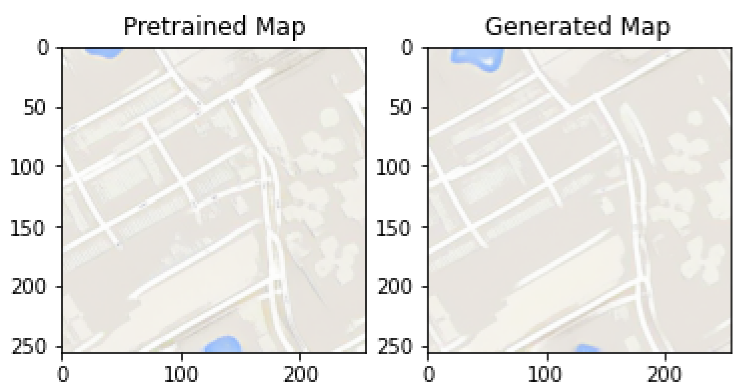
(b) **Generator loss graph** **Discriminator loss graph**



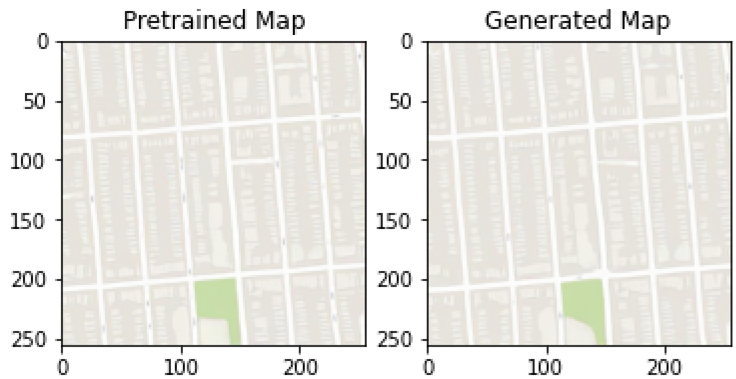


(c) **Pretrained GAN and Trained GAN results**

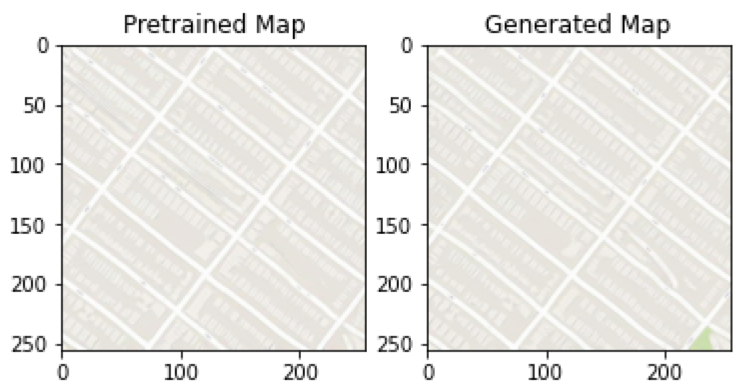
SSIM = 0.686



SSIM = 0.701

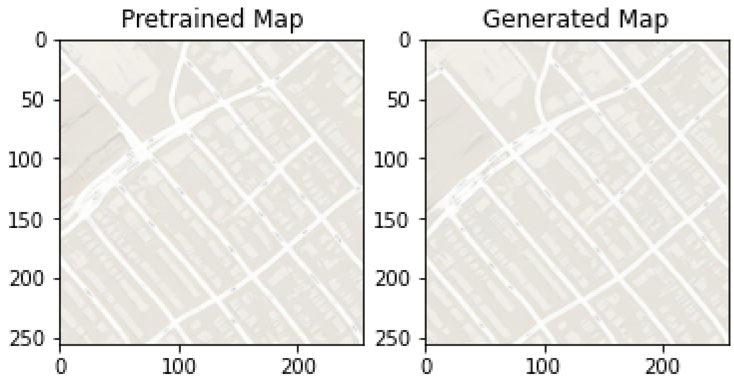


SSIM = 0.657



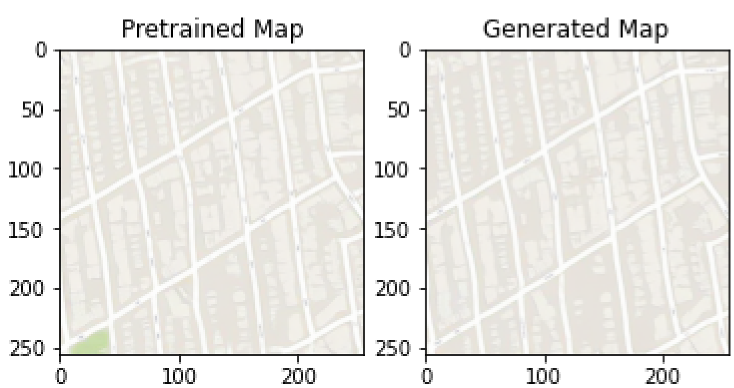
SSIM = 0.688

SSIM = 0.668



SSIM = 0.718

SSIM = 0.683



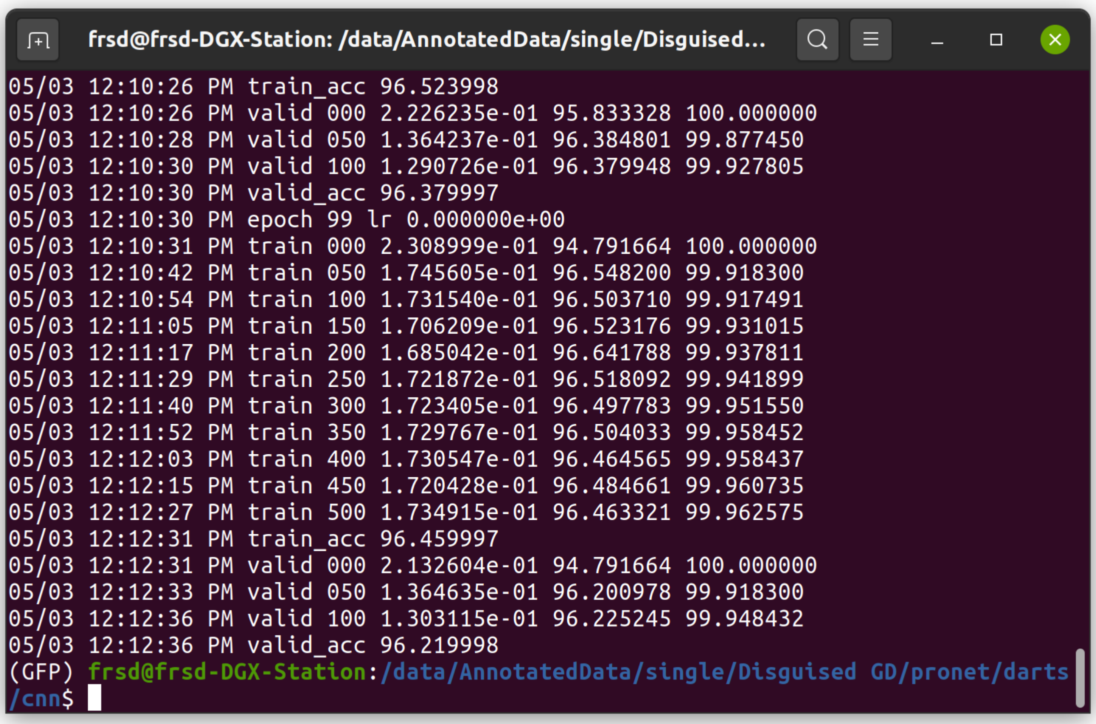
Observations:

Pretrained model used is CycleGAN As we can see from the above SSIM (Structural Similarity Index Measrue) pretraind models generates better visual maps of satellite images. Images generated by pretrained GAN are verysimilar to the ground truth maps as compared to images generated by trained model.

Question4:

"DARTS: Differentiable Architecture Search"

Reproduced results of paper on Dataset: **CIFAR-10**



DARTS: Differentiable Architecture Search

- This paper takes a different approach to the problem and propose DARTS, an efficient architecture search method" (Differentiable Architecture Search). Instead of searching over a discrete set of candidate architectures, we broaden the search space to be continuous, allowing the architecture to be optimised by gradient descent with respect to its validation set performance.
- DARTS achieves competitive performance with the state of the art using orders of magnitude fewer computation resources due to the data efficiency of gradient-based optimization as opposed to inefficient black-box search.
- Authors present a novel bilevel optimization-based algorithm for differentiable network architecture search.

Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)
while *not converged* **do**
  1. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$
     ($\xi = 0$ if using first-order approximation)
  2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
Derive the final architecture based on the learned α.

Explaination for Above Algorithm:

1. The problem with searching over a discrete set of candidate operations is that the model has to be trained on specific configuration before moving onto the next configuration. This obviously is time-consuming. Authors found a way of relaxing the discrete set of candidate operations.
2. This is the model's core structure. There are one or more nodes in the cell. These nodes are also referred to as states.
3. The input to a cell is the output of the previous two cells, just like ResNets. There are nodes in this cell. Assume we create a cell with three states/nodes. As a result, the first node will have two inputs, i.e. outputs from the previous two cells.
4. The second state will receive inputs from the first state as well as outputs from the last two cells, for a total of three inputs.

5. The third state will receive inputs from the second and first states, as well as outputs from the last.
6. They used Bilevel Optimization

$$\begin{array}{ll} \min_{\alpha} & \mathcal{L}_{\text{val}}\left(w^{*}(\alpha), \alpha\right) \\ \text{s.t.} & w^{*}(\alpha) = \operatorname{argmin}_{w} \mathcal{L}_{\text{train}}(w, \alpha) \end{array}$$

Since we have weight available that have been optimised on the training set, the optimisation problem can be framed as finding the alphas to minimise validation loss.

$$\nabla_{\alpha} L_{\text{val}}\left(w^{*}(\alpha), \alpha\right)$$

α : parameter for operation weight $w^{*}(\alpha)$: optimal conv weights for specific α

7. Obtaining the optimal weight for each configuration of alpha in equation above requires two optimization loops, hence the authors suggested approximating in such a way that does not need to be optimised until convergence.

Steps to run code:

- **For training**

  %cd cnn

  !python train.py —auxiliary —cutout

- **For Testing**

  %cd cnn

  !python test.py —auxiliary —model_path weights.pt

****Note: Question2 and Question 3 have run the github so not providing the code in the submission.**