DL Assignment-3 [700 Level]

DCGANs, Few-Shot Learning and DArtS

By:- Kwanit Gupta
B19EE046

Question-1

Train a DCGAN to generate images from noise. Use the MNIST database to learn the GAN Network.

[Discriminator in DCGAN:-
i. if roll no. % 2 == 0: use VGG16 as a discriminator.
ii. if roll no. % 2 == 1: use Resnet-18 as a discriminator.]

Perform the following tasks: [35 marks for training GAN]

a. Uniformly generate ten noise vectors that act as latent representation vectors, and generate the images for these noise vectors, and visualize them at [5 + 5 + 5 marks]

i. After the first epoch.
ii. After n/2 th epoch.
iii. After your last epoch. (say n epochs in total)

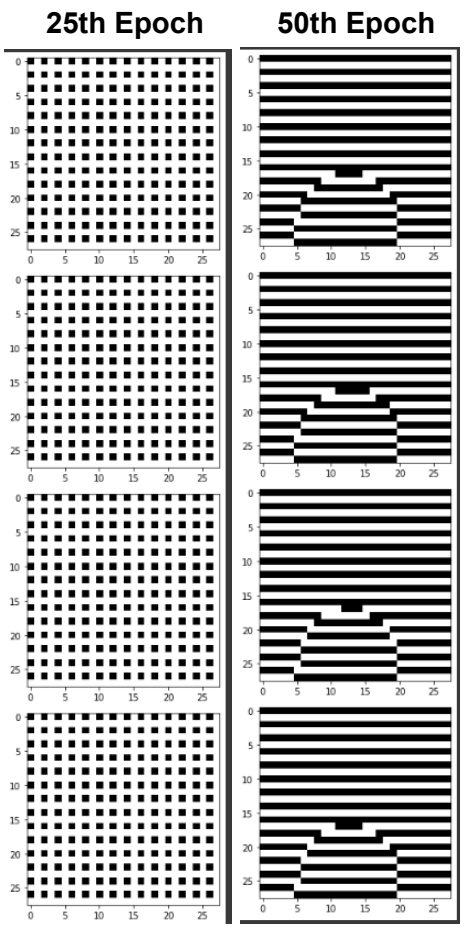
and comment on the image interpretation at (i), (ii) and (iii) and can you identify the images? [5 marks]

b. Plot generator and discriminator losses for all the iterations. [One iteration = forward pass of a mini-batch] [10 marks]

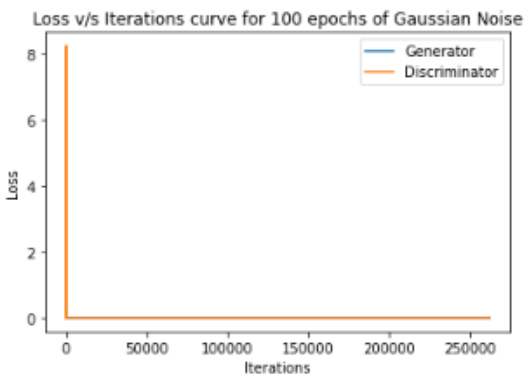
c. Do we have control of what class image the generator will generate, given a noise vector? Suppose, we are interested in generating only “4” images, will the GAN you trained in (a) can do that? Explain why. If not, modify the GAN trained above to do this. [10 + 40 Marks]

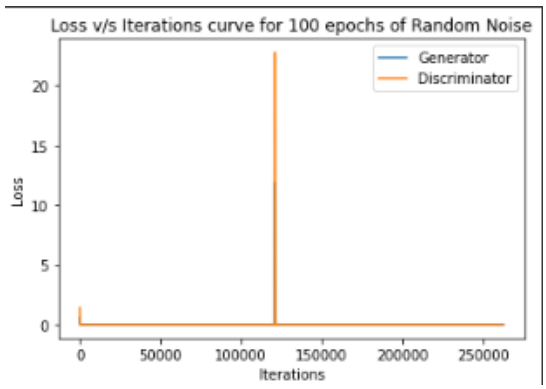
Solution-1

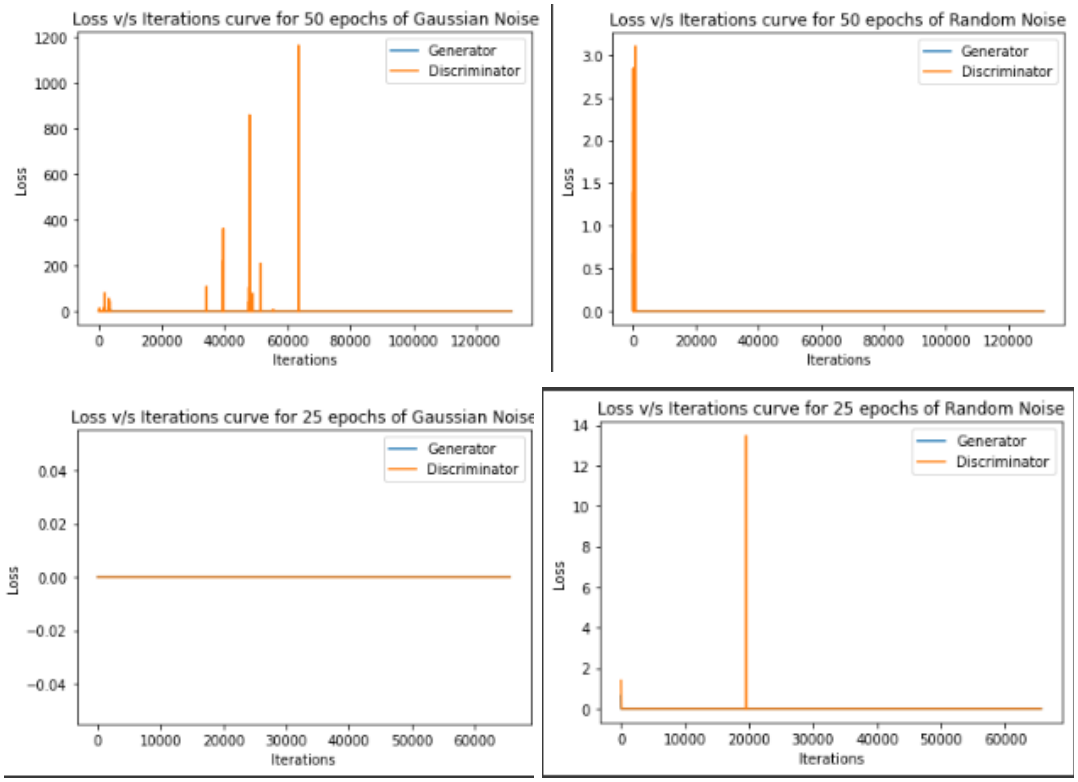
(a) Since my roll number is B19EE046, thereby the ending number signifies an even quantity leading to VGG-16 Model features for the discriminator. Since the Generator wasn’t robust enough to generate the images, even after doing variations in Noises (Gaussian and Random) and increasing epochs (25,50,100), the images produced were as following:-



(b) Following are the graphs corresponding to Loss v/s Iterations for variations in number of epochs invested:-

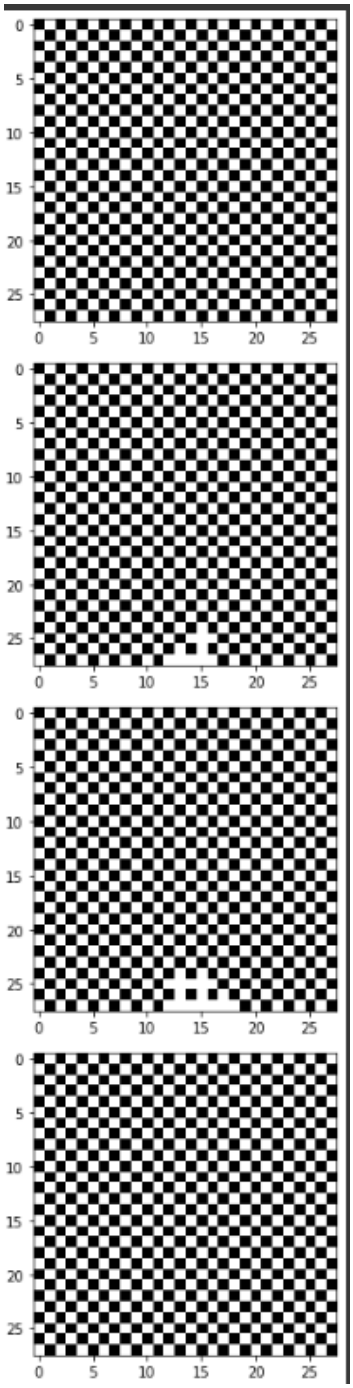


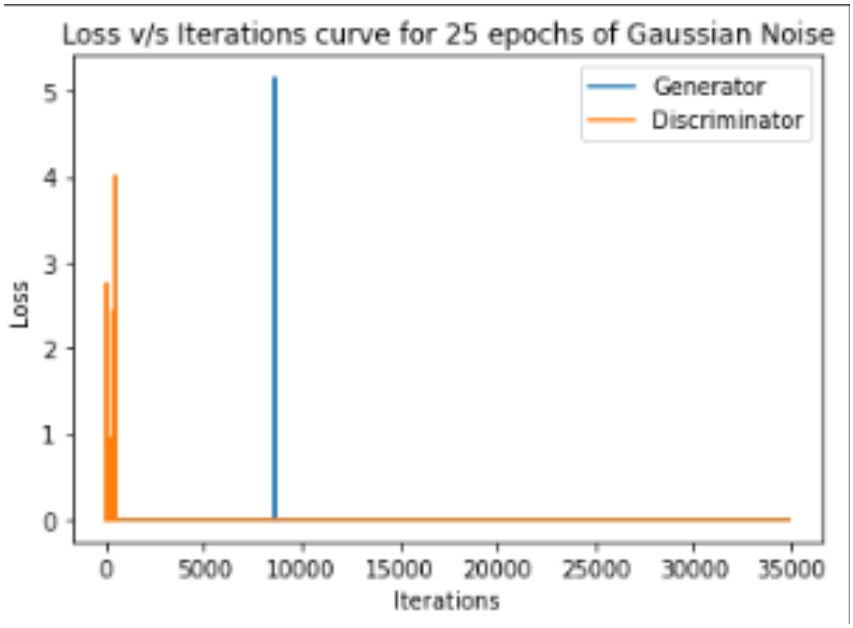




(c) No, Because the output is purely dependent on random noise, we have no control over what is created in the initial GAN. We can add a conditional input c to the random noise z. In most circumstances, the conditional input vector c is concatenated with the noise vector z and fed into the generator in the same way as the initial GAN. We may also do other data upgrades on c and z. The picture class, object characteristics, or an embedding of text descriptions for the image we want to make, for example, are all possibilities for conditional input c.

Although I tried to make it according to a single class, for example 9, the results came out as following:-





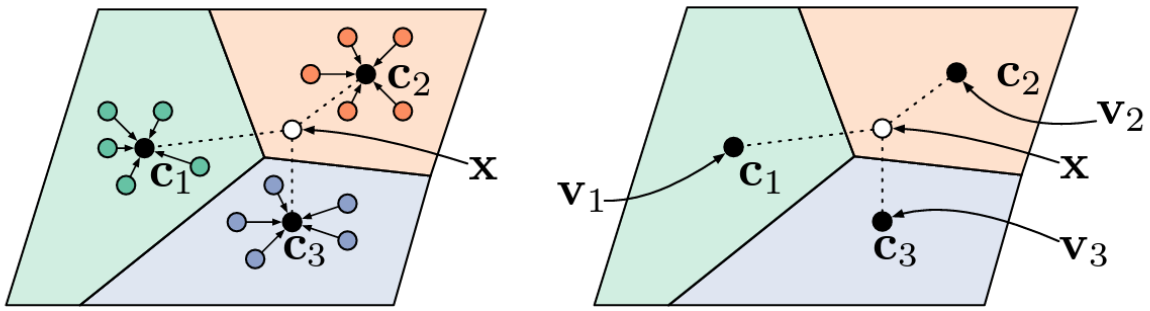
Question-2

Reproduce the results of the paper “Prototypical Networks for Few-shot Learning”.

You can use the authors’ code (mostly provided via GitHub) or any other reimplementation available on the internet. [Please cite the source]. [30 marks]

You should be able to understand and explain the code that you are using. Write a 2 page report explaining the algorithm and your analysis based on your results. [20+10 marks]

Solution-2



(a) Few-shot (b) Zero-shot

In this paper, the following components are important on algorithmic basis and their corresponding explanations would be present as well:-

1. Surface-level overview of Prototypical Networks:-

For the sake of dataset, a small support set of N labelled instances S = (x1,y1),...,(xN,yN), was provided where each xi, is the D-dimensional feature vector of each example and yi is the associated label. Group of instances belonging to class k, is denoted by Sk.

Starting by knowing about an embedding function fφ, which can be pre-trained or scratch-built network, with fine-tunable/learnable set of parameters φ. These are defined so as to generate an M-dimensional representation point ck, or prototype, of each class (Somehow looking analogous to clustering, where cluster centres are crucial guiding points).

Each prototype represents the average of the embedded support points in its class (As seen in Clustering Algorithms too).

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_{\phi}(\mathbf{x}_i)$$

2. Choice of Distance Metric:-

Given a distance function d, prototypical networks produce a probabilistic distribution over classes for a given query point x, based on a softmax over distances to the prototypes, as following:-

$$p_{\boldsymbol{\phi}}(y = k \mid \mathbf{x}) = \frac{\exp(-d(f_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{c}_{k'}))}$$

Depending upon the nature of the Distance Metric (Whether Statistical-based, Matrix-based or Vector-based), there is a concept defined as Bergman Divergence (some examples being Squarred Euclidean Distance, Mahalnobis Distance), which can be mathematically written as:-

$$d_{\varphi}(\mathbf{z}, \mathbf{z}') = \varphi(\mathbf{z}) - \varphi(\mathbf{z}') - (\mathbf{z} - \mathbf{z}')^T \nabla\varphi(\mathbf{z}'),$$

By minimizing the negative log-probability J(φ) = −logpφ(y = k|x) of the true class k, learning starts (even by using SGD). Training episodes are formed by randomly selecting a subset of classes for training, then choosing a subset of examples within each class for support set of the remainder as query points.

$J \leftarrow 0$
for k in $\{1, \ldots, N_C\}$ **do**
  for $(\mathbf{x}, y)$ in Q_k **do**

$$J \leftarrow J + \frac{1}{N_C N_Q} \left[d(f_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{c}_k)) + \log \sum_{k'} \exp(-d(f_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{c}_k)) \right]$$

  end for
end for

3. Choice of Network Architecture:-

The authors made use of customized Convolutional Neural Network with their own set of configurational components like Pooling Layers, Convolutional Kernels, Normalizations, dropouts, etc.

4. Analysis based on Results (Till 20 epochs):-

Configuration	Accuracy (%)
Cosine, k-test=10, n-test=5	65.4%
Cosine, k-test=5, n-test=5	64.2%
Cosine, k-test=5, n-test=1	64.6%
Euclidean with L2 Norm, k-test=10, n-test=5	78.8%
Euclidean with L2 Norm, k-test=10, n-test=1	79.4%
Euclidean with L2 Norm, k-test=20, n-test=1	78%
Euclidean with L2 Norm, k-test=5, n-test=5	78.2%
Euclidean with L2 Norm, k-test=5, n-test=1	79%

Cosine Similarity (Not a Bergman Divergence) failed to converge the loss and thereby the accuracy dipped more than that of Euclidean Distance.

Also, with decrease in k-test (keeping n-test same), accuracy dipped slightly and with decrease in n-test (keeping k-test same), accuracy increased.

Question-3

Your task is to train a GAN that generates maps given satellite images. [Dataset Link]. [40 marks for training GAN]

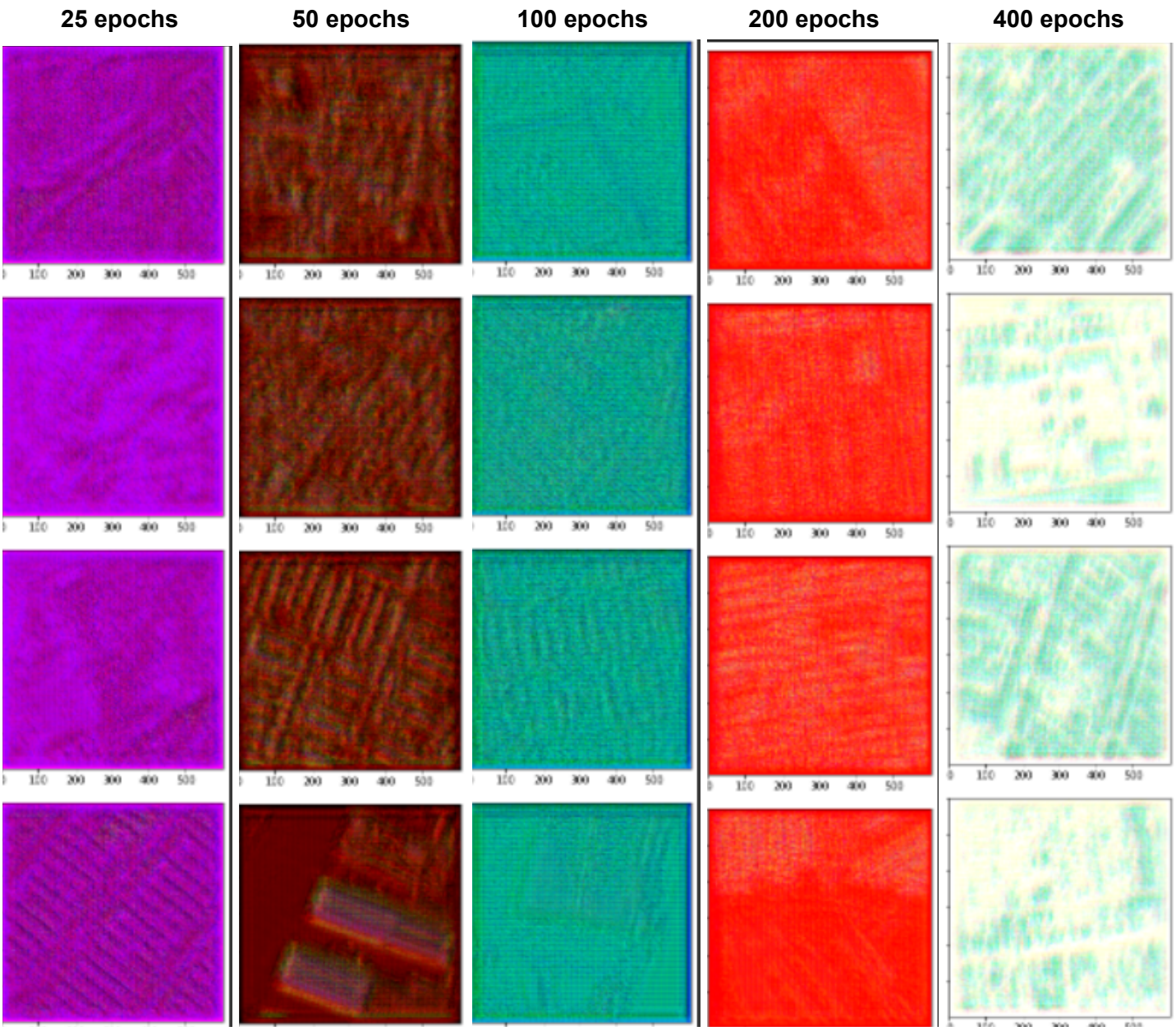
a. Report visualized image translations [satellite image -ground truth map - map generated]. [at least 10 examples] [10 marks]

b. Plot generator and discriminator losses for all the iterations. [One iteration = forward pass of a mini-batch]. [10 marks]

c. Use the pre-trained sat2map generator modeland generate images for the same ten examples in (a). Compare and comment on the images generated with a pre-trained generator with your trained generator in (a). Support your claim with SSIM. [15 + 5 marks]

Solution-3

(a) For easier understanding (Otherwise the 10 examples for Image translations were already made in the Colab Notebook), the differences in the visual representation for map can be seen as following:-

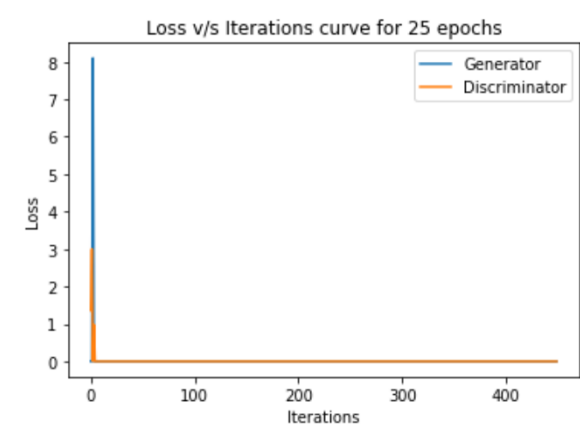


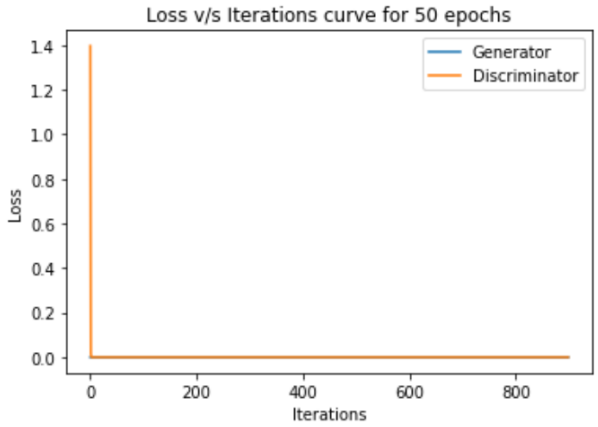
It can be pretty much seen that the custom-made GAN Archtecture is somehow starting to capture the structural details of Map, but there isn't clarity made upon the colour palette until certain epochs were made to reach (In this case, it is more than 400, which themselves consumed over 3 hours of training).

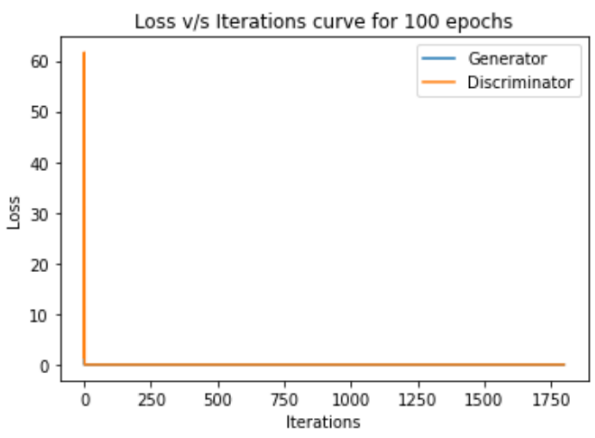
It was surprising to know that such dynamicity of colour changes observed in this experimentation (Starting from Reddish-Blue Spectrum, then Greenish Spectrum and after certain epochs, then stabilising).

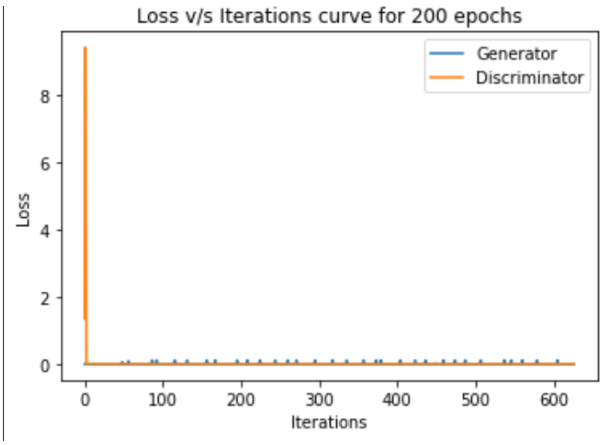
(b)

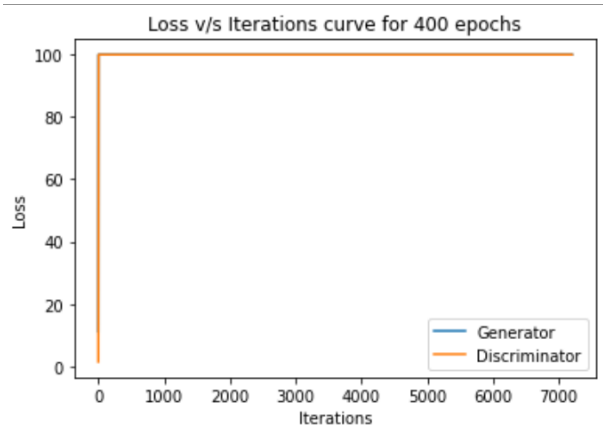
Following were the Loss v/s Iteration Curves for varied amounts of Epochs (25, 50, 100, 200 and 400 epochs):-





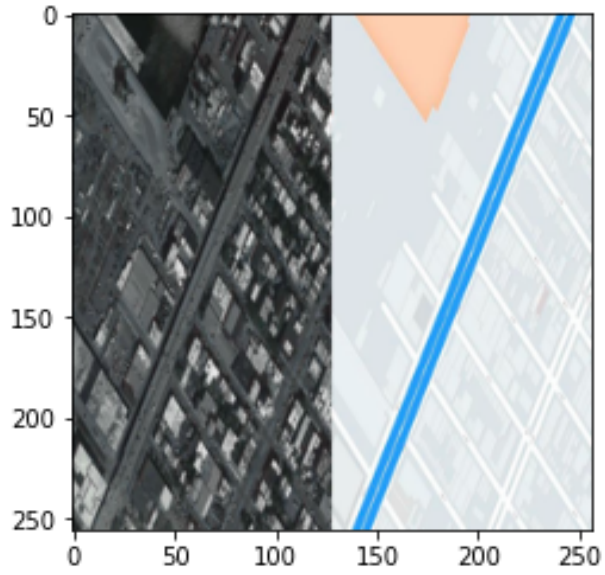




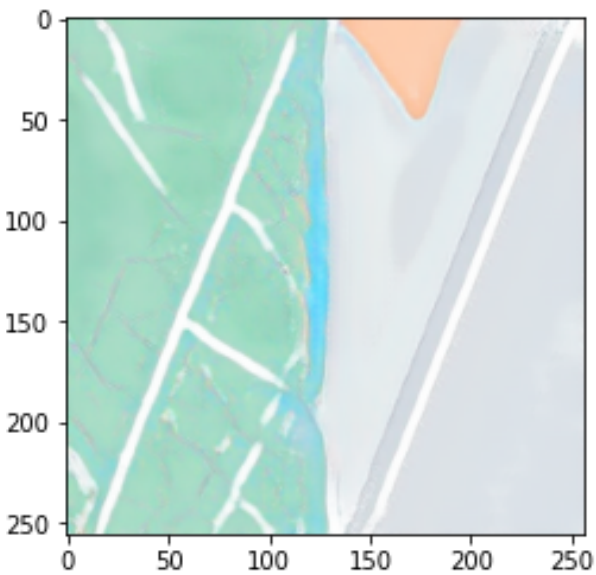


The noticeable trend hereby was that within initial sets of iterations, the loss would be able to start optimizing upto a significant extent (even 100x as well).

(c) Following was the output produced after passing the input via Sat2Map based CycleGAN implementation:-



Actual Input



Faked Output

With finer structural details and colour palette, Sat2Map proved to be more powerful than that of the simplistic GAN architecture that was solely made for an hands-on-experience in this assignment.

The SSIM readings for Sat2Map was coming out to be around 0.4 on average. On the other hand, the SSIM readings for out implementation came out to be around 0.2 on average.

Question-4

Reproduce the results of the paper “DARTS: Differentiable Architecture Search” only on CIFAR-10 dataset.

You can use the authors’ code (mostly provided via GitHub) or any other reimplementation available on the internet. [Please cite the source]. [30 marks]

You should be able to understand and explain the code that you are using. Write a 2 page report explaining the algorithm and your analysis based on your results. [20+10 marks]

Solution-4

Depending on the use case of the dataset/application area, the techniques associated with Neural Architecture Search can vary. For the sake of this question, the respective authors discussed and came up with the following variation:-

1. Instead of manually handling the building blocks of a Deep Learning Model (Whether belonging to Vision or Time-Series Application), people used to think about using Reinforcement Learning and deploying an agent to grab the necessary components for the Neural Network in order to receive rewards as lower computational expense and higher evaluation metrics.

2. But for this paper, the Authors made a mathematicall-differentiable bounded optimization function, in order to configure the cell (An acyclic graph of ordered sequence of N nodes, where each node represent the latent representation/feature map and the directed edge between nodes represent the mathematical operation). Assumption is made that a cell consist of 2 input nodes and an output node.

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$$

3. For Convolutional Neural Network, the input node can be the output of previous cells and for Recurrent Neural Network, the input node can be the output of the current cell and the state for the previous node. In order to extract the output of the cell, all the intermediate nodes are concatenated together.

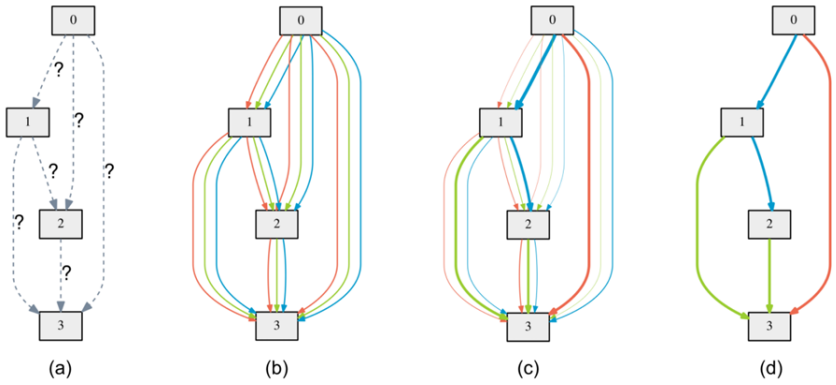


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

4. A continuous search space is established with the set of network operations defined as O and following probabilistic function is defined with softmax support:-

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

5. The task just remains is to replace the discrete o-(i,j)(x) with the continuous maximum likely function, defined via argmax o(x). It is done via optimizing the validation loss with any of the optimization function (even SGD as well).

6. A Bilevel optimization task emerges where the training as well as testing losses would be required to lower simultaneously with the given choice of architecture and the weights associated with it.\

When it comes to the analysis part of the implementation, it was done on CIFAR-10 as well as PTB. Following were the results obtained:-

1. Reproduced Results on CIFAR-10

```
!python test.py --auxiliary --model_path '/content/drive/MyDrive/B19EE046_Q4/darts/cnn/cifar10_model.pt'

04/28 06:21:23 PM gpu device = 0
04/28 06:21:23 PM args = Namespace(arch='DARTS', auxiliary=True, batch_size=96, cutout=False, cutout_length
108 108 36
108 144 36
144 144 36
144 144 36
144 144 36
144 144 36
144 144 72
144 288 72
288 288 72
288 288 72
288 288 72
288 288 72
288 288 72
288 288 144
288 576 144
576 576 144
576 576 144
576 576 144
576 576 144
576 576 144
04/28 06:21:27 PM param size = 3.349342MB
Files already downloaded and verified
test.py:84: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  input = Variable(input, volatile=True).cuda()
test.py:85: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  target = Variable(target, volatile=True).cuda()
04/28 06:21:29 PM test 000 1.233735e-01 96.875000 100.000000
04/28 06:21:34 PM test 050 1.105460e-01 97.120095 99.959150
04/28 06:21:39 PM test 100 1.074740e-01 97.359733 99.948432
04/28 06:21:39 PM test_acc 97.369997
```

2. Reproduced Results on PTB

```
!python test.py --data '/content/drive/MyDrive/B19EE046_Q4/awd-lstm-lm/data/penn/' --model_path '/content/drive/MyDrive/B19EE046_Q4/darts/rnn/ptb_model.pt'
80570 82429
80605 82429
80640 82429
80675 82429
80710 82429
80745 82429
80780 82429
80815 82429
80850 82429
80885 82429
80920 82429
80955 82429
80990 82429
81025 82429
81060 82429
81095 82429
81130 82429
81165 82429
81200 82429
81235 82429
81270 82429
81305 82429
81340 82429
81375 82429
81410 82429
81445 82429
81480 82429
81515 82429
81550 82429
81585 82429
81620 82429
81655 82429
81690 82429
81725 82429
81760 82429
81795 82429
81830 82429
81865 82429
81900 82429
81935 82429
81970 82429
82005 82429
82040 82429
82075 82429
82110 82429
82145 82429
82180 82429
82215 82429
82250 82429
82285 82429
82320 82429
82355 82429
82390 82429
82425 82429
=========================================================================================
| End of training | test loss  4.02 | test ppl    55.68
=========================================================================================
```

Due to computational restraints in the form of GPU Power and RAM, I wasn't able to perform the actual Architecture search in the training perspective. But, I somehow manage to find the right genotypes for the PTB Data and results can be shown as following:-

```
!python train_search.py --data '/content/drive/MyDrive/B19EE046_Q4/awd-lstm-lm/data/penn/' --unrolled
        [0.2095, 0.2128, 0.1888, 0.1826, 0.2063],
        [0.2041, 0.2010, 0.1894, 0.1878, 0.2176],
        [0.1713, 0.2083, 0.1982, 0.1965, 0.2257],
        [0.2078, 0.2075, 0.1782, 0.2001, 0.2064],
        [0.1809, 0.2079, 0.1844, 0.2125, 0.2143],
        [0.1998, 0.2126, 0.1850, 0.1878, 0.2148],
        [0.1858, 0.2219, 0.1857, 0.1903, 0.2164],
        [0.2254, 0.1999, 0.1806, 0.1950, 0.1991],
        [0.2097, 0.2052, 0.1910, 0.1943, 0.1998],
        [0.1773, 0.2248, 0.1906, 0.1930, 0.2144],
        [0.2321, 0.1906, 0.1824, 0.1898, 0.2051],
        [0.1961, 0.2095, 0.1888, 0.1955, 0.2101],
        [0.1859, 0.2254, 0.1846, 0.1920, 0.2122],
        [0.1901, 0.2164, 0.1822, 0.1991, 0.2122]], device='cuda:0',
       grad_fn=<SoftmaxBackward0>)
04/28 10:08:44 PM | epoch  15 |    50/  103 batches | lr 20.00 | ms/batch 3271.02 | loss  6.71 | ppl   818.22
04/28 10:11:23 PM Genotype(recurrent=[('sigmoid', 0), ('tanh', 0), ('tanh', 0), ('sigmoid', 3), ('identity', 3), ('identity', 3), ('identity', 3), ('tanh', 6)], concat=range(1, 9))
tensor([[[0.2048, 0.2021, 0.1934, 0.2113, 0.1884],
        [0.1812, 0.2328, 0.1857, 0.1907, 0.2096],
        [0.2155, 0.2229, 0.1712, 0.1889, 0.2015],
        [0.1668, 0.2338, 0.1792, 0.2206, 0.1996],
        [0.2185, 0.2154, 0.1716, 0.1932, 0.2013],
        [0.2147, 0.2176, 0.1707, 0.1972, 0.1997],
        [0.1809, 0.2037, 0.2104, 0.1921, 0.2129],
        [0.2220, 0.1980, 0.1949, 0.2084, 0.1767],
        [0.2169, 0.1801, 0.1964, 0.1927, 0.2139],
        [0.1711, 0.2059, 0.2124, 0.2142, 0.1964],
        [0.1866, 0.2085, 0.1860, 0.2094, 0.2095],
        [0.2060, 0.1975, 0.1876, 0.2067, 0.2022],
        [0.2154, 0.1944, 0.1779, 0.1968, 0.2155],
        [0.1746, 0.2038, 0.1998, 0.1988, 0.2230],
        [0.2066, 0.1885, 0.1898, 0.1974, 0.2175],
        [0.2010, 0.1917, 0.1928, 0.1881, 0.2263],
        [0.2188, 0.1832, 0.1926, 0.1783, 0.2271],
        [0.2021, 0.1841, 0.2010, 0.1808, 0.2320],
        [0.1846, 0.1879, 0.2048, 0.1820, 0.2407],
        [0.2277, 0.1799, 0.1872, 0.1822, 0.2231],
        [0.1995, 0.1864, 0.1992, 0.1863, 0.2286],
        [0.1892, 0.2069, 0.1847, 0.2032, 0.2159],
        [0.2095, 0.2129, 0.1888, 0.1826, 0.2062],
        [0.2037, 0.2010, 0.1896, 0.1878, 0.2180],
        [0.1713, 0.2083, 0.1984, 0.1963, 0.2258],
        [0.2086, 0.2078, 0.1782, 0.2005, 0.2049],
        [0.1807, 0.2077, 0.1842, 0.2132, 0.2141],
        [0.1991, 0.2116, 0.1842, 0.1871, 0.2180],
        [0.1856, 0.2219, 0.1857, 0.1903, 0.2165],
        [0.2250, 0.1998, 0.1807, 0.1949, 0.1995],
        [0.2097, 0.2052, 0.1909, 0.1943, 0.1999],
        [0.1774, 0.2249, 0.1905, 0.1930, 0.2142],
        [0.2318, 0.1918, 0.1820, 0.1898, 0.2046],
        [0.1962, 0.2095, 0.1888, 0.1953, 0.2102],
        [0.1855, 0.2251, 0.1848, 0.1918, 0.2128],
        [0.1899, 0.2163, 0.1811, 0.2023, 0.2105]], device='cuda:0',
       grad_fn=<SoftmaxBackward0>)
04/28 10:11:23 PM | epoch  15 |   100/  103 batches | lr 20.00 | ms/batch 3190.78 | loss  6.46 | ppl   638.11
04/28 10:12:39 PM -----------------------------------------------------------------------------------------
04/28 10:12:39 PM | end of epoch  15 | time: 398.70s | valid loss  6.63 | valid ppl   759.27
```