Deep Learning
Assignment 4

Deep Convolutional GAN (DCGAN)

Deadline: 20th June 2022, 11:59 pm

Submission Guidelines:

Submission:

Submit all of your codes and results in a single zip file with the name FirstName_RollNumber_04.zip

- Submit a single zip file containing
  (a) codes (b) report (c) Saved Models (d) Readme.txt
- There should be a Report.pdf detailing your experience and highlighting any interesting results. Kindly don't explain your code in the report, just explain the results. Your report should include your comments on the results of all the steps, with images, for example, what happened when you changed the learning rate, etc.
- Readme.txt should explain how to run your code, preferably it should accept the command line arguments e.g dataset path used for training the model.
- The assignment is accepted in both .py and Jupyter notebooks. You can use Google Collab for GPU.
- In the root directory, there should be **1** python/notebook file, a report, and a folder containing saved models.
- The root directory should be named **FirstName_RollNumber_04**
- Your main code file should be named **rollNumber_04.py**
- Follow all the naming conventions.
- For each convention, there is a 3% penalty if you don't follow it.
- Email the instructor or TA if there are any questions. You cannot look at others' code or use others' code, however, you can discuss it with each other. Plagiarism will lead to a straight zero with additional consequences as well.
- 10% (of obtained marks) deduction per day for a late submission.
- The submissions will only be accepted till Wednesday(22nd of june).
- DON'T RESUBMIT THE DATASETS PROVIDED IN YOUR SUBMISSION.
- Report without code/experiment will be rewarded zero
- **Use the provided dataset for this assignment.**

Due Date: 20/06/2022

Note: For this assignment (and for others in general) you are not allowed to **search online for any kind of implementation.** Do not share code or look at the other's code. You should not have any implementation related to the assignment, other than your own.

In case of any confusion please reach out to the TAs (email them or visit them during their office hours).

Objectives:

- Understand how GAN works
- Understand what discriminator and Generator
- Understand how to implement the training loop for GAN

What is a GAN? (source https://lilianweng.github.io/posts/2017-08-20-gan/)

In 2014, Goodfellow et al. presented a method for training generative models called Generative Adversarial Networks (GANs for short).

GAN consists of two models:

A discriminator D estimates the probability of a given sample coming from the real dataset. It works as a critic and is optimized to tell the fake samples from the real ones.

A generator G outputs synthetic samples given a noise variable input (brings in potential output diversity). It is trained to capture the real data distribution so that its generative samples can be as real as possible, or in other words, can trick the discriminator to offer a high probability.

These two models compete against each other during the training process: the generator G is trying hard to trick the discriminator, while the discriminator model D is trying hard not to be cheated. This interesting zero-sum game between two models motivates both to improve their functionalities.

$$\min_G \max_D \mathbb{E}_{x \sim q_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))],$$

where Z ~ p(z) are the random noise samples, G(z) are the generated images using the neural network generator G, and D is the output of the discriminator, specifying the probability of an input being real.

Introduction

In this assignment, you will implement and train GANs from scratch. We will implement a specific type of GAN designed to process images, called a Deep Convolutional GAN (DCGAN).

A DCGAN is simply a GAN that uses a convolutional neural network as the discriminator, and a network composed of transposed convolutions as the generator. To implement the DCGAN, we need to specify three things:

1) the generator,

2) the discriminator, and

3) the training procedure.

We will develop each of these three components in the following subsections.

Discriminator

The discriminator in this DCGAN is a convolutional neural network that has the following architecture:

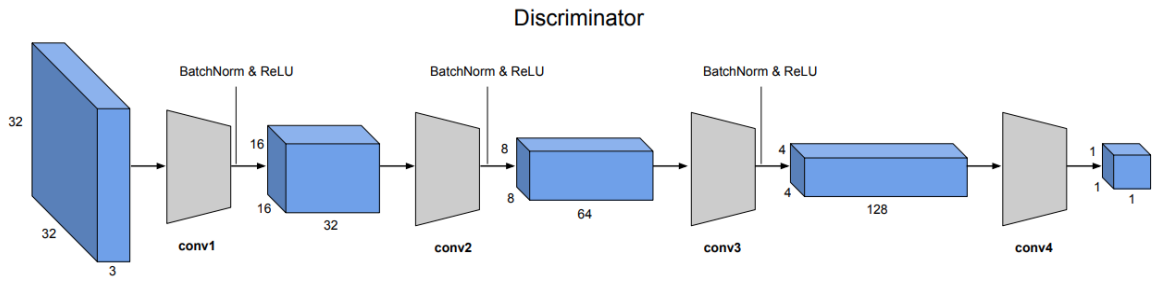


Figure 1 Discriminator Architecture

In each of the convolutional layers shown above, we downsample the spatial dimension of the input volume by a factor of 2. You need to implement the `Discriminator` class, shown below according to the provided architecture.

Generator

Now, we will implement the generator of the DCGAN, which consists of a sequence of `upconv` (*transpose convolutional) layers* (see `torch.nn.ConvTranspose2d` documentations here) that progressively upsample the input noise sample to generate a fake image. You need to implement the `Generator` class using the following architecture:

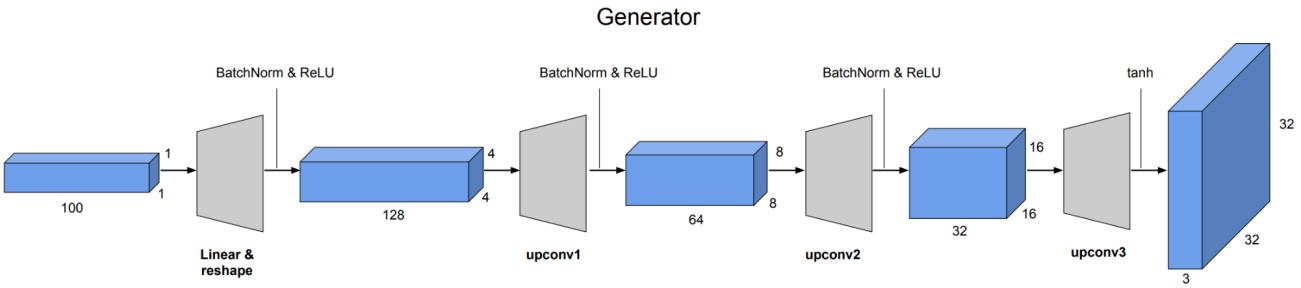


Figure 2 Generator Architecture

Note: Please zoom this image if you are unable to read the text on architecture

You will take 100-d z randomly sampled as a starting point followed by a linear layer. You need to set the parameters of linear such that when you reshape we get 128x4x4.

Note:

The original DCGAN generator uses the deconv function to expand the spatial dimension. Odena et al. later found the deconv creates checkerboard artifacts in the generated samples.

Training Loop

Next, you will implement the training loop for the DCGAN.

We train it in exactly the same way as a standard GAN. The pseudo-code for the training procedure is shown below. The actual implementation is simpler than it may seem from the pseudo-code: this will give you practice in translating math to code.

HINTS:

You can use the function binary cross entropy loss function (`torch.nn.functional.binary_cross_entropy_with_logits`)to compute these losses of Generator and Discriminator.

You will also need to compute labels corresponding to real or fake. You can assign 1 to real images label and 0 to generated images. Batch size can help you to create labels in the training loop.

```
real_labels = torch.ones(size, device=device)
```

Note: You can use a single optimizer to train both G and D but in this assignment, you will use a separate optimizer for both.

Algorithm 1 GAN Training Loop Pseudocode

1: **procedure** TRAINGAN
2: Draw m training examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from the data distribution p_{data}
3: **Draw m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from the noise distribution p_z**
4: **Generate fake images from the noise: $G(z^{(i)})$ for $i \in \{1, \ldots .m\}$**
5: **Compute the (least-squares) discriminator loss:**

$$J^{(D)} = \frac{1}{2m}\sum_{i=1}^{m}\left[\left(D(x^{(i)}) - 1\right)^2\right] + \frac{1}{2m}\sum_{i=1}^{m}\left[\left(D(G(z^{(i)}))\right)^2\right]$$

6: Update the parameters of the discriminator
7: **Draw m new noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from the noise distribution p_z**
8: **Generate fake images from the noise: $G(z^{(i)})$ for $i \in \{1, \ldots .m\}$**
9: **Compute the (least-squares) generator loss:**

$$J^{(G)} = \frac{1}{m}\sum_{i=1}^{m}\left[\left(D(G(z^{(i)})) - 1\right)^2\right]$$

10: Update the parameters of the generator

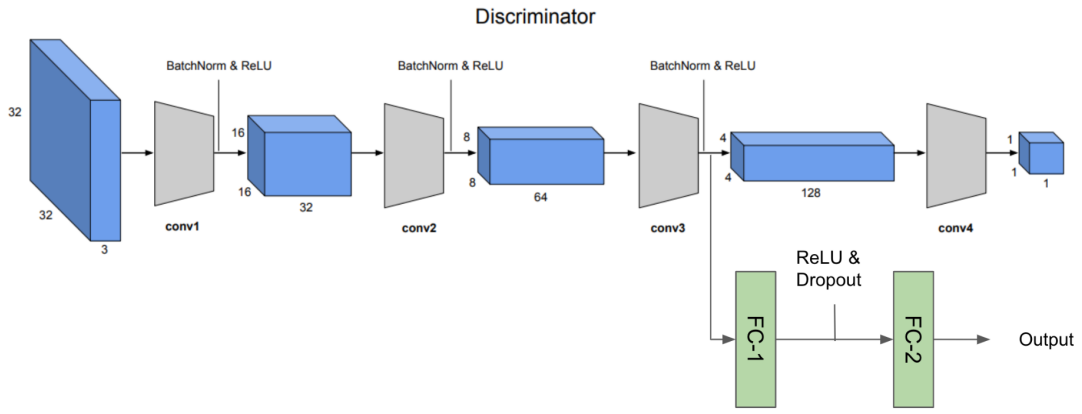
Task 2: Multi-task discriminator

In this task, you need to utilize the discriminator as an image classifier along with the fake/real discriminator. Specifically, you will add a classification head after conv3 of the original discriminator which will classify input image labels. This classification head consists of 2 fully connected layers.

The total loss of discriminator will be :

$$J^{D\_total} = J^{(D)} + J^{(C)}$$

Where $J^{(D)}$ is your original discriminator loss and $J^{(C)}$ your new classification loss. Following is the architecture of new Discriminator.



Dataset Details

CIFAR-10 is an established computer-vision dataset used for image classification. It is a subset of the 80 million tiny images dataset and consists of 60000 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. It has the classes: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'. The images are of size 3x32x32, i.e. 3-channel color images of 32x32 pixels in size. You will use this dataset to generate real-world images using GANs.

You can download the dataset from here:

https://drive.google.com/drive/folders/1Cr84QOtrxeBF-BDVIDEc06Um90R3mjlA?usp=sharing

This dataset is divided into training and testing folders with 10 subfolders in each indicating the class labels. For GANs you don't need the class labels so you can put all folders into one.

Note: We will ask you to implement another module later and for that, you will need class labels.

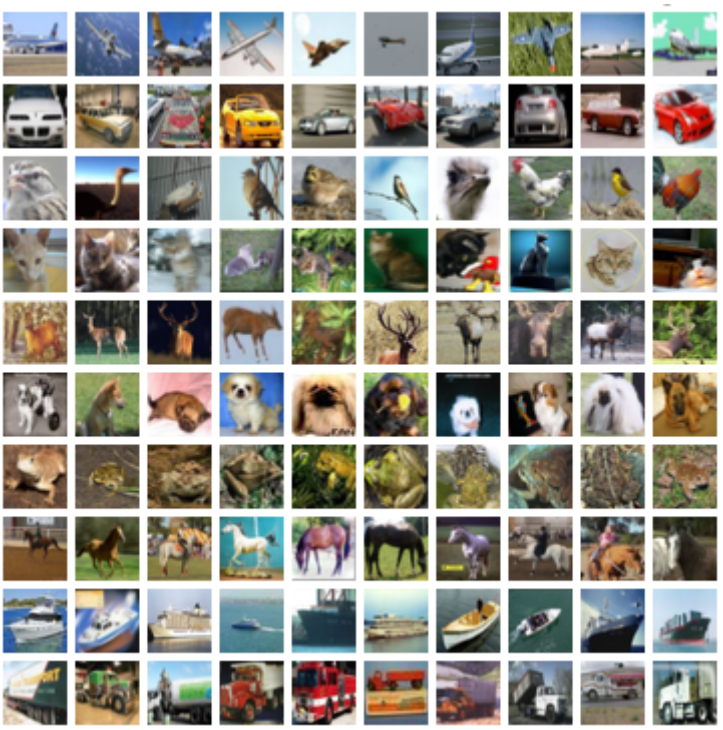


Figure 3 Dataset examples

Summary:

1. Prepare data loader
2. Implement `Discriminator` and `Generator` classes
3. Define Loss functions and optimizers
4. Implement training loop
5. Save and load model

Marks Distribution

- **Code** **50%**
  - Dataloader 5
  - Discriminator 10
  - Generator 15
  - Training Loop 15
  - Loss Functions 5
- **Report** **40%**
  - Plot the losses(generator and discriminator) separately over training iterations.
  - Visualize and save generated images every epoch (No screenshots)
  - Try over different batch sizes, epochs, and optimizers and report results with loses plot
  - Plot and save images of weights of discriminator and generator (No screenshots)

    - Loss and Accuracy Curves for classification head-on validation and training dataset.
    - Plot examples of images generated after using the new discriminator.
    - Discuss the behavior of the discriminator after adding the classification head? Does the new head affect the generator?
- **Viva** **10%**

***Note**: Deliverables will be updated in case there are additional tasks in the assignment.*

🤓 Good Luck 🤓