CA2 - Generative Adversarial Networks Analysis: CIFAR10

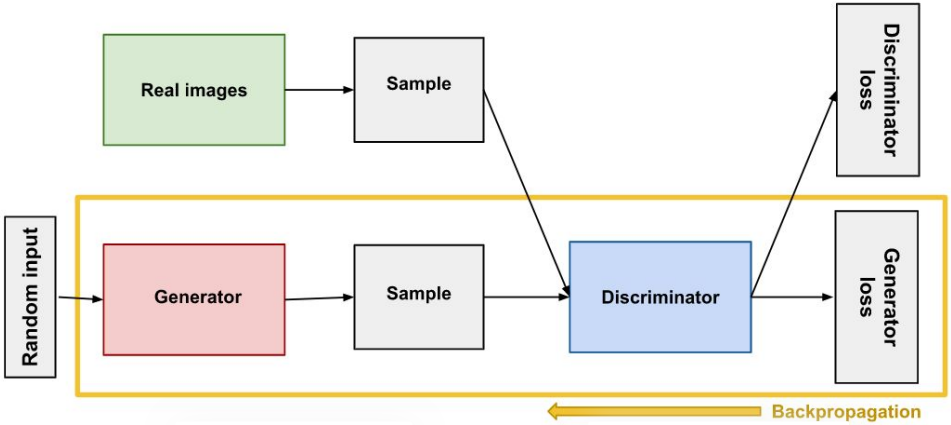
Name: **Quah Johnnie**
Admin No: **2007476**

Contents (Slides)

1.0: Intro GAN & CIFAR10	Overview of CIFAR10 | Introduction to GAN & Ideology
2.0: Data Engineering	Preparing data for GAN training
3.0: Metrics	Available metrics, what do the metrics they stand for?
4.0: Training & Evaluation	Simple DCGAN & cDCGAN | Evaluation & Results
5.0: Training & Evaluation (Advanced Architectures)	BigGANs & Complex Architectures | Evaluation & Results
6.0: Image Generation	Truncation trick | Observing Final Images
7.0: Conclusion	-

https://thisxdoesnotexist.com/
https://www.twitch.tv/watchmeforever

1.0) GAN? What's GAN



- Two competing neural networks: Generator + Discriminator:
  - Generator's Ultimate Goal: Produce fake data that is indistinguishable from real data
  - Discriminator's Ultimate Goal: Correctly identify whether a given sample is real or fake

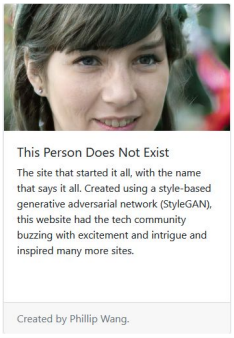
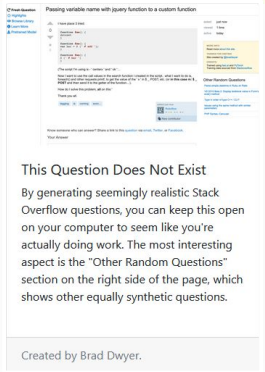
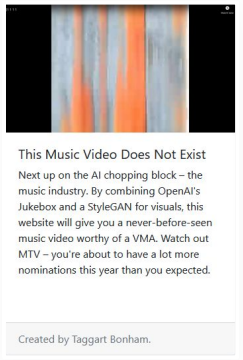


Image Generation



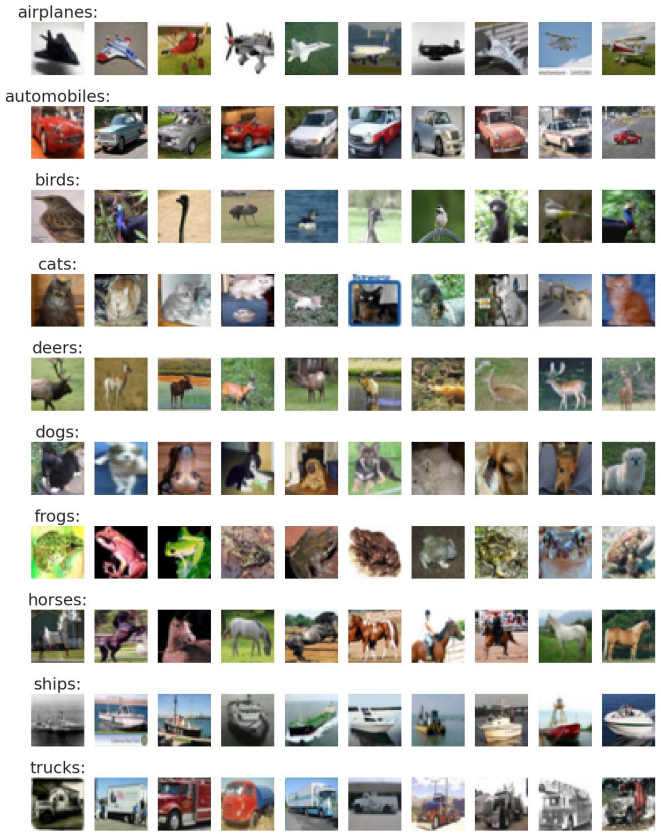
Text Synthesis

Meme Generation



Audio + Video Generation

1.0) CIFAR10 Dataset



- 10 Classes
- 60,000 Images
- Uniformly distributed
- 32 x 32 Pixel size

The 32x32 pixel size of the images in the CIFAR10 dataset is relatively small, and this may limit the capacity of the GAN to capture the high-level features and patterns in the images.

Image Averages:

- Animals have a green-ish background
- Objects have a blue or brown-ish background

2.0) Data Engineering

Flow:

Download Raw CIFAR10 Data (API) -> Load Into Tensor -> Convert To Desire Value Range -> Data Concat -> Separate Data and Labels -> Ready To Load Into Dataloader -> Train!

```
In [2]: training_data = datasets.CIFAR10(
            root="data",
            train=True,
            download=True,
            transform=transforms.ToTensor()
        )
        test_data = datasets.CIFAR10(
            root="data",
            train=False,
            download=True,
            transform=transforms.ToTensor()
        )

        Files already downloaded and verified
        Files already downloaded and verified

In [3]: class_labels = training_data.classes

        train_loader = DataLoader(training_data, batch_size=len(training_data))
        test_loader = DataLoader(test_data, batch_size=len(test_data))

        train_data = torch.Tensor(next(iter(train_loader))[0].numpy())
        test_data = torch.Tensor(next(iter(test_loader))[0].numpy())

        train_label = torch.Tensor(next(iter(train_loader))[1].numpy())
        test_label = torch.Tensor(next(iter(test_loader))[1].numpy())

        del train_loader, test_loader, training_data

        def img4np(tensor):
            tensor = np.swapaxes(tensor.numpy(),1,-1)
            return np.swapaxes(tensor,1,2)

        train_data_np = img4np(train_data)
        train_data_np.shape

Out[3]: (50000, 32, 32, 3)
```

Download Data and Load to Tensor

```
print('Data min:',train_data.min())
print('Data max:',train_data.max())
print(train_data.shape,'\n\n')

Data min: tensor(0.)
Data max: tensor(1.)
torch.Size([50000, 3, 32, 32])
```

0 to 1 Normalized Data (Default)

```
#Multiply data by 2 and substract by 1
train_data_2 = torch.sub(torch.mul(train_data, 2), 1)
test_data_2 = torch.sub(torch.mul(test_data, 2), 1)

print('Data min:',train_data_2.min())
print('Data max:',train_data_2.max())

Data min: tensor(-1.)
Data max: tensor(1.)
```

Looks like its in the right scale

Data Concatenation (Combining train and test)

```
cifar10 = torch.cat((train_data_2, test_data_2),0)
cifar10_og = torch.cat((train_data, test_data),0)
cifar10_labels = torch.cat((train_label, test_label),0)

print('CIFAR10 Training Images (real) shape:', cifar10.shape)
print('CIFAR10 Class Labels shape:', cifar10_labels.shape)

del train_data, train_label, train_data_2, test_data_2

CIFAR10 Training Images (real) shape: torch.Size([60000, 3, 32, 32])
CIFAR10 Class Labels shape: torch.Size([60000])
```

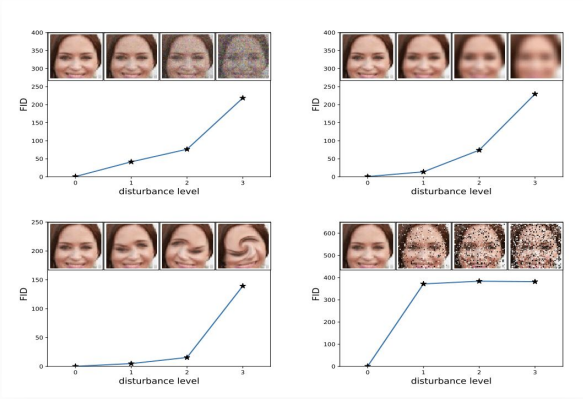
-1 to 1 Normalized, Data Concat and Seperate Labels

I have prepared a (-1 to 1) CIFAR Dataset as well as (0 to 1) CIFAR10 Dataset

3.0) Metrics Used: FID & IS

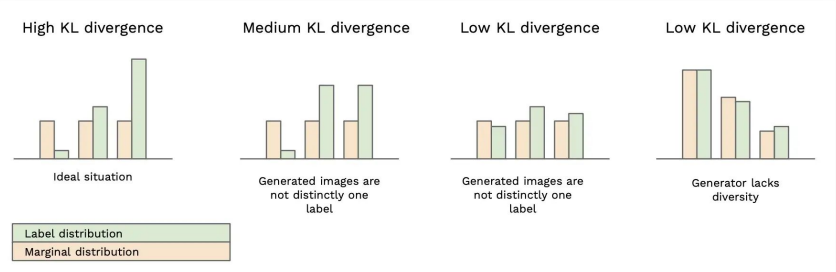
Frechet Inception Distance (FID):

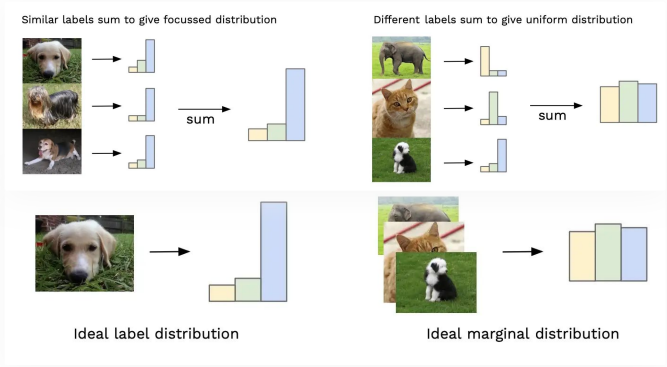
- It measures the distance between the activations of a pre-trained Inception network when applied to real and generated images. Lower the better (Less disturbance between real vs fake images).



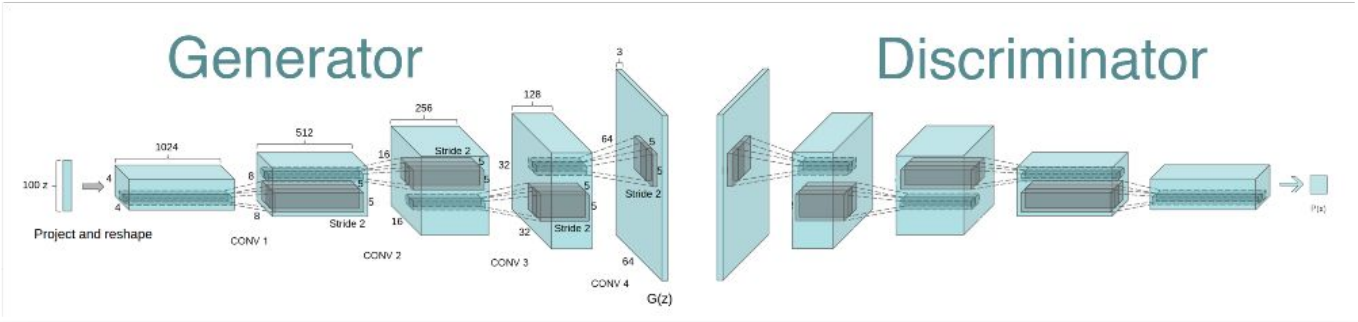
Inception Score (IS):

- Measures quality and diversity of generated images. It's based on the KL-divergence between the conditional class distributions in real and generated images.





4.0) Simple DCGAN & cDCGAN

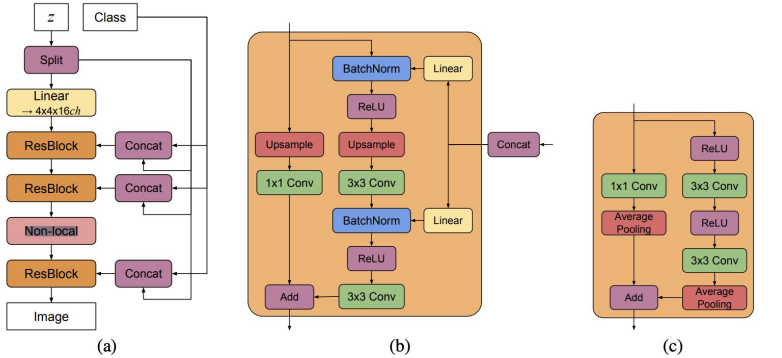


The image shows the idea of the architecture I implemented. However, it is not the same as the one I implemented.

Methods	Architecture Description	Methodology Description	Loss	FID	IS
Vanilla DCGAN	-	-	Binary Cross-entropy Loss	36.1	6.24
cDCGAN	-	cBN	Binary Cross-entropy Loss	32.3	6.72

Images displayed are as seen in the demo/Jupyter notebook or HTML

5.0) BigGANs & More



(a) (b) (c)

Figure 15: (a) A typical architectural layout for BigGAN's **G**; details are in the following tables. (b) A Residual Block (*ResBlock up*) in BigGAN's **G**. (c) A Residual Block (*ResBlock down*) in BigGAN's **D**.

Tested BigGAN Models and Architectures During This Assignment

Methods	Architecture Description	Methodology Description	Loss	FID	IS
cBigGAN	ATT, RES, PD	cBN, SN, O-SLS	Hinge Loss	14.55	8.32 ± 0.088
cBigGAN-LeCam	ATT, RES, PD, LeCam, EMA	cBN, SN, O-SLS	Hinge Loss + Regularization Loss (LeCam)	13.63	8.28 ± 0.081
cBigGAN-LeCam-DiffAug	**ATT, RES, PD, LeCam, EMA**	**cBN, SN, O-SLS, DiffA**	**Hinge Loss + Regularization Loss (LeCam)**	**7.98**	**8.79 ± 0.080**

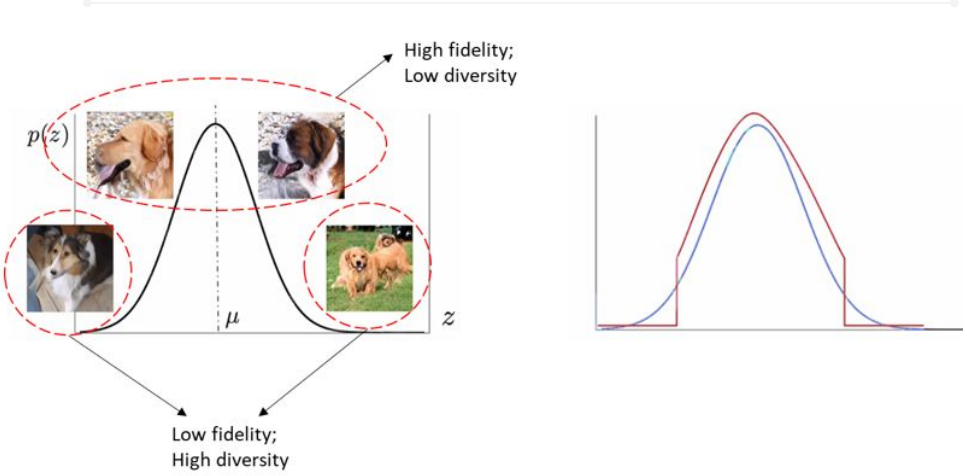
EMA: Exponential Moving Average Used (Generator). **cBN** : conditional Batch Normalization. **SN**: Spectral Normalization. **ATT**: Attention Module Used. **RES** : ResNet Modules Used. **PD** : Projection Discriminator. **O-SLS**: One-sided label smoothing. **LeCam**: LeCam EMA/Regularization Used. **DiffA**: Differentiable Augmentation Used. **EMA**: Exponential Moving Average Used For Generator

*The BigGAN architecture is built upon multiple architectures, notably its ResNet backbone for both the generator and discriminator

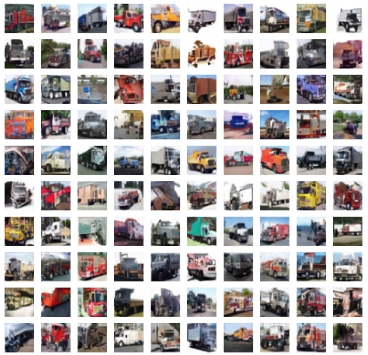
Images displayed are as seen in the demo/Jupyter notebook or HTML

6.0) Image Generation /w Truncation Trick

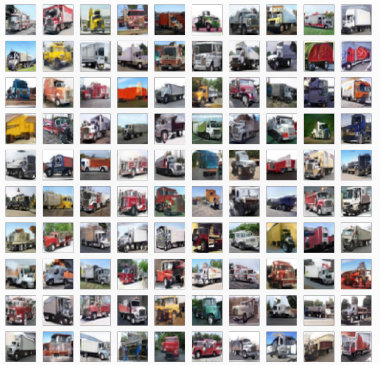
Truncation trick can decrease FID scores and improve the quality of generated images by discarding those with high FID. However, it often comes at the cost of increased computation power and decreased image diversity.



Normal Generation



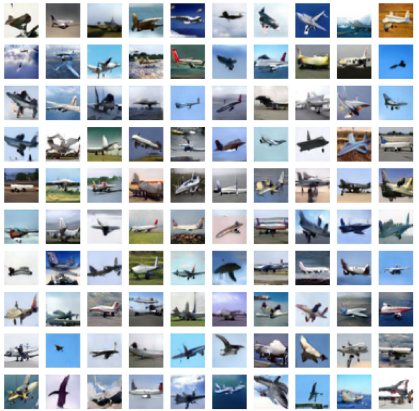
Truncation Trick



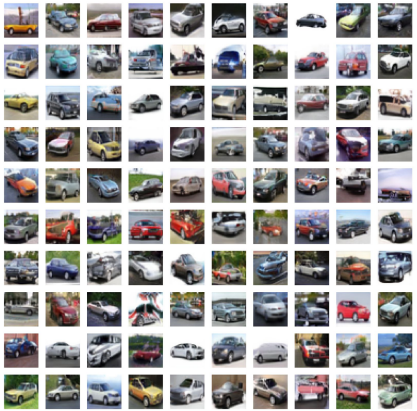
-> ./images/submission/GAN_100_Generated_{class}

6.0) Images Generated

airplane (Generated)



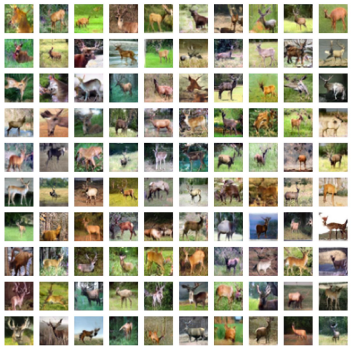
automobile (Generated)



bird (Generated)

cat (Generated)

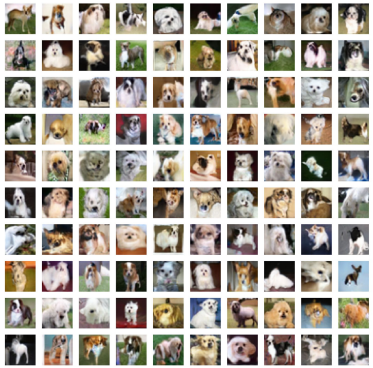
deer (Generated)



-> ./images/submission/GAN_100_Generated_{class}

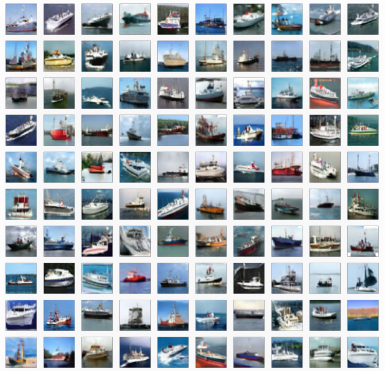
6.0) Images Generated

dog (Generated)

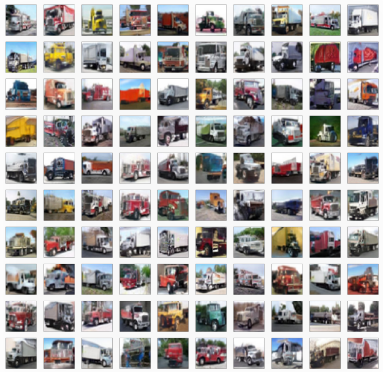


frog (Generated)

ship (Generated)



truck (Generated)



6.0) Image Generation

Go to `./images/submission/GAN_100_Generated_{class}` to see a higher quality of the images shown..

Conclusion

In the assignment, I made progress by building different GAN models, starting from simple ones and gradually increasing their complexity. This allowed me to improve the quality of my generated images from poor to good. To achieve this improvement, I used advanced architectures and methodologies, which is referred to as a ”horizontal” improvement, rather than fine-tuning hyperparameters, known as a ”vertical” improvement. I tried different architectures and loss functions to get better results. I also used evaluation metrics like inception score and FID to measure the quality of the generated images and guide further improvement. I employed the truncation trick to select high-quality images. Overall, the assignment was a learning experience with GANs and focused on achieving better image generation results.

AI Generation memes

Thanks!

Fin. 完。