

Homework #2

Deep Learning for Computer Vision

NTU, Fall 2021

110/11/2

110/11/23 (Tue.) 3:00 AM due

Outline

- Problems & Grading
- Dataset
- Rules
- Training Tips

Problems – Overview

- **Image Generation and Feature Disentanglement**
 - Problem 1: GAN (35%) [Face dataset]
 - Problem 2: ACGAN (30%) [Digits dataset]
- **Unsupervised Domain Adaptation (UDA)**
 - Problem 3: DANN (35%) [Digits dataset]
 - Bonus: Improved UDA Model (6%) [Digits dataset]

Please refer to “Dataset” section for more details about face/digits datasets.

Problem 1: GAN (35%)

A generative adversarial network (GAN) is a deep learning method in which two neural networks (Generator and Discriminator) **compete** with each other and improve themselves together.

In this problem, you should implement a GAN model **from scratch** and train it on the **face dataset**. Besides, you will need to compute some scores to analyze your GAN model. Please follow the problems below:

1. Build your generator and discriminator **from scratch** and show your model architecture in your report (You can use “`print(model)`” in PyTorch directly). Then, train your model on the face dataset and describe implementation details. (**Include** but not limited to training epochs, learning rate schedule, data augmentation and optimizer) (5%)

 **You should not load pretrained model for your own implementation.**

Problem 1: GAN (35%) (cont'd)

2. Now, we can use the Generator to randomly generate images. Please sample 1000 noise vectors from Normal distribution and input them into your Generator. Save the **1000** generated images in the assigned folder path for evaluation, and show **the first 32** images in your report. **[see the below example]** (5%)

⚠ You should **fix the random seed** in your program such that the generated images are always the same.



Example for the first 32 generated images in your report

Problem 1: GAN (35%) (cont'd)

3. Evaluate your 1000 generated images by implementing two metrics:

(1) Fréchet inception distance (FID) : use all “test” images in the face dataset as reference

(2) Inception score (IS)

- Suggested reference for the metric usage:

(1) FID: <https://github.com/mseitzer/pytorch-fid>

(2) IS: <https://github.com/sbarratt/inception-score-pytorch>

4. We will calculate FID and IS to evaluate your generated images. (20% total)

Metric	Simple Baseline	Strong Baseline
FID ↓	35.0 (5%)	30.0 (5%)
IS ↑	2.00 (5%)	2.15 (5%)

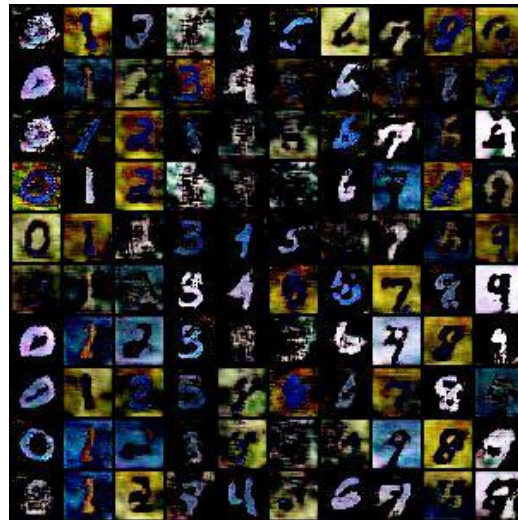
5. Discuss what you’ve observed and learned from implementing GAN. (5%)

- You can compare different architecture or describe some difficulties during training.

Problem 2: ACGAN (30%)

Auxiliary Classifier GAN (ACGAN) is a conditional GAN method applying auxiliary classifiers for conditional image synthesis and feature disentanglement.

In this problem, we aim to generate images for the corresponding digit inputs. You should implement an ACGAN model **from scratch** and train it on the **mnistm dataset (inside the digit dataset)**. Besides, you will need to conduct some experiments to analyze your model. Please follow the problems in the next pages:



Generated 10 images for each digits

Problem 2: ACGAN (30%)

1. Build your ACGAN model **from scratch** and show your model architecture in your report (You can use “print(model)” in PyTorch directly). Then, train your model on the **mnistm dataset** and describe implementation details. (e.g. How do you input the class labels into the model?) (10%)
2. Sample random noise and generate 100 conditional images **for each digit (0-9)**. Save total 1000 outputs in the assigned folder path for further evaluation. We will provide a **digit classifier** to evaluate your output images.
3. You should name your output digit images as the following format:
(The first number of your filename indicates the corresponding digit label)

Output_folder/

0_001.png

0_002.png


...

0_099.png

0_100.png

1_001.png

...

 You should **fix the random seed** in your program such that the the generated images are always the same.

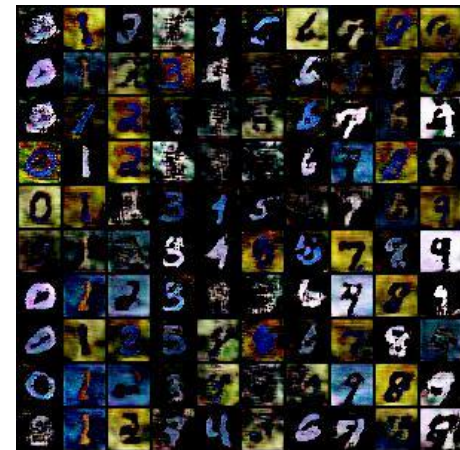
Problem 2: ACGAN (30%)

4. We will evaluate your generated output by the classification accuracy with a pretrained **digit classifier**, and we have provided the model architecture [**digit_classifier.py**] and the weight [**Classifier.pth**] for you to test. (15%)

Metric	Simple Baseline (10%)	Strong Baseline (5%)
Accuracy	70%	80%

5. Show 10 images for each digit (0-9) in your report. You can put all 100 outputs in one image with columns indicating different digits and rows indicating different noise inputs. **[see the below example]** (5%)

⚠ You should **fix the random seed** in your program such that the the generated images are always the same.



Example for Prob. 2-5

Problem 3: DANN (35%)

In this problem, you need to implement DANN on **digits datasets (USPS, MNIST-M and SVHN)** and consider the following 3 scenarios:
(Source domain → Target domain)

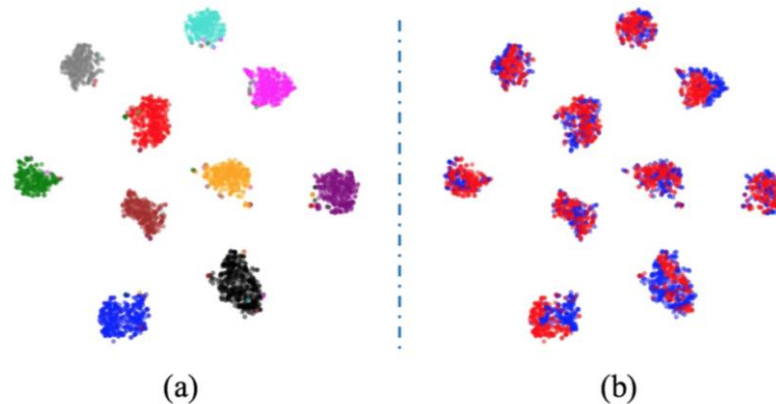
SVHN → MNIST-M, MNIST-M → USPS, USPS → SVHN

Note that during training DANN, we utilize the **images and labels** of **source** domain, and **only images (without labels)** of **target** domain.

1. Compute the **accuracy** on **target** domain, while the model is trained on **source** domain only. (lower bound) (3%)
 - Use source images and labels in the training folder for training, target images and labels in the testing folder to compute the accuracy
2. Compute the **accuracy** on **target** domain, while the model is trained on **source and target** domain. (domain adaptation) (4+9%)
 - Use source images and labels in the training folder + target images in the training folder for training, target images and labels in the testing folder to compute the accuracy
3. Compute the **accuracy** on **target** domain, while the model is trained on **target** domain only. (upper bound) (3%)
 - Use target images and labels in the training folder for training, target images and labels in the testing folder to compute the accuracy

Problem 3: DANN (35%) (cont'd)

4. Visualize the latent space by mapping the *testing* images to 2D space **with t-SNE** and use different colors to indicate data of (a) different digit classes 0-9 and (b) different domains (**source**/**target**). (9%)
- Note that you need to plot the figures of the three scenarios, so you would need to plot **6 figures** in total in this sub-problem. (1.5% for each figure)



5. Describe the implementation details of your model and discuss what you've observed and learned from implementing DANN. (7%)

Bonus: Improved UDA model (6%)

For the bonus problem, you need to modify your model in Problem 3 and redo the three scenarios in Problem 3-2. Note that your modified model should perform better on all three scenarios.

- Example of modifying model: change loss function, change model architecture, contrastive learning, ...
 - Note that finetuning hyper-parameters (learning rate, weight decay, batch size, optimizer) is not counted as a modification.
1. Compute the **accuracy** on **target** domain, while the model is trained on **source and target** domain. (domain adaptation) (3%)
 - Use source images and labels in the training folder + target images in the training folder for training, target images and labels in the testing folder to compute the accuracy.
 2. Briefly describe implementation details of your model and discuss what you've observed and learned from implementing your improved UDA model. (3%)

Bonus: Improved UDA model (6%)

(cont'd)

Related UDA papers

- [Adversarial Discriminative Domain Adaptation](#). CVPR 2017
- [Generate To Adapt: Aligning Domains using Generative Adversarial Networks](#). CVPR 2018
- [From source to target and back: Symmetric Bi-Directional Adaptive GAN](#). CVPR 2018
- [Sliced Wasserstein Discrepancy for Unsupervised Domain Adaptation](#). CVPR 2019
- [Light-weight Calibrator: a Separable Component for Unsupervised Domain Adaptation](#). CVPR 2019
- [DRANet: Disentangling Representation and Adaptation Networks for Unsupervised Cross-Domain Adaptation](#). CVPR 2021
- You may select any one of the papers above, or search for other related papers, or design a method by yourself to implement your improved model.

Grading – Problem 3 & Bonus

- Baseline score for DANN (Problem 3-2)

	MNIST-M → USPS	SVHN → MNIST-M	USPS → SVHN
Baseline	72%	42%	28%

- If your accuracy of Problem 3-2 does not pass any baseline accuracy, you only get 4 points in this sub-problem.
 - Pass the baseline in one scenario: 4+3 points, two: 4+6 points, three: 4+9 points
- Your accuracy of the Bonus problem should **surpass** the accuracy reported in Problem 3-2.

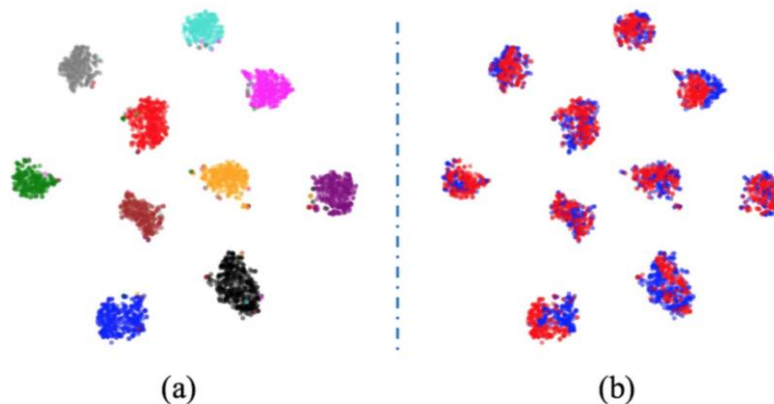
Grading – Problem 3

- Expected results on report:

Accuracy: e.g.,

	MNIST-M \rightarrow USPS	SVHN \rightarrow MNIST-M	USPS \rightarrow SVHN
Trained on source			
Adaptation (DANN/Improved)			
Trained on target			

t-SNE: e.g.,



Grading – Bonus

- Expected results on report:

Accuracy: e.g.,

	MNIST-M \rightarrow USPS	SVHN \rightarrow MNIST-M	USPS \rightarrow SVHN
Original model			
Improved model			

Outline

- Problems & Grading
- **Dataset**
- Rules
- Training Tips

Dataset – Face

A subset of human face dataset CelebA

- Images are cropped and downscaled to 64×64
- 40000 training samples (about 21% of complete CelebA)

Dataset – Face

Format

data/

└─ face/

└─ train/

└─ test/

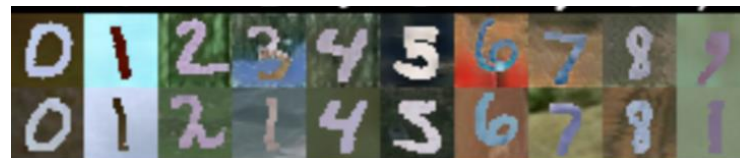
40000 images for training (00000.png ~ 39999.png)

2621 images for testing (40000.png ~ 42620.png)

└─ digits/

Dataset – Digits

- USPS Dataset
 - # of data: 7,291 / 2,007 (testing)
 - # of classes: **10** (0~9)
 - Image size: **28 * 28 * 1**
- MNIST-M Dataset
 - # of data: 60,000 / 10,000 (testing)
 - # of classes: **10** (0~9)
 - Generated from MNIST
 - A subset of MNIST. The digit images are normalized (and centered) in size **28 * 28 * 3** pixels.



Dataset – Digits

- SVHN Dataset
 - # of data: 73,257 / 26,032 (testing)
 - # of classes: **10** (0~9)
 - Real-world image dataset for developing machine learning.
 - MNIST-like (size: **28 * 28 * 3**) images centered around a single character

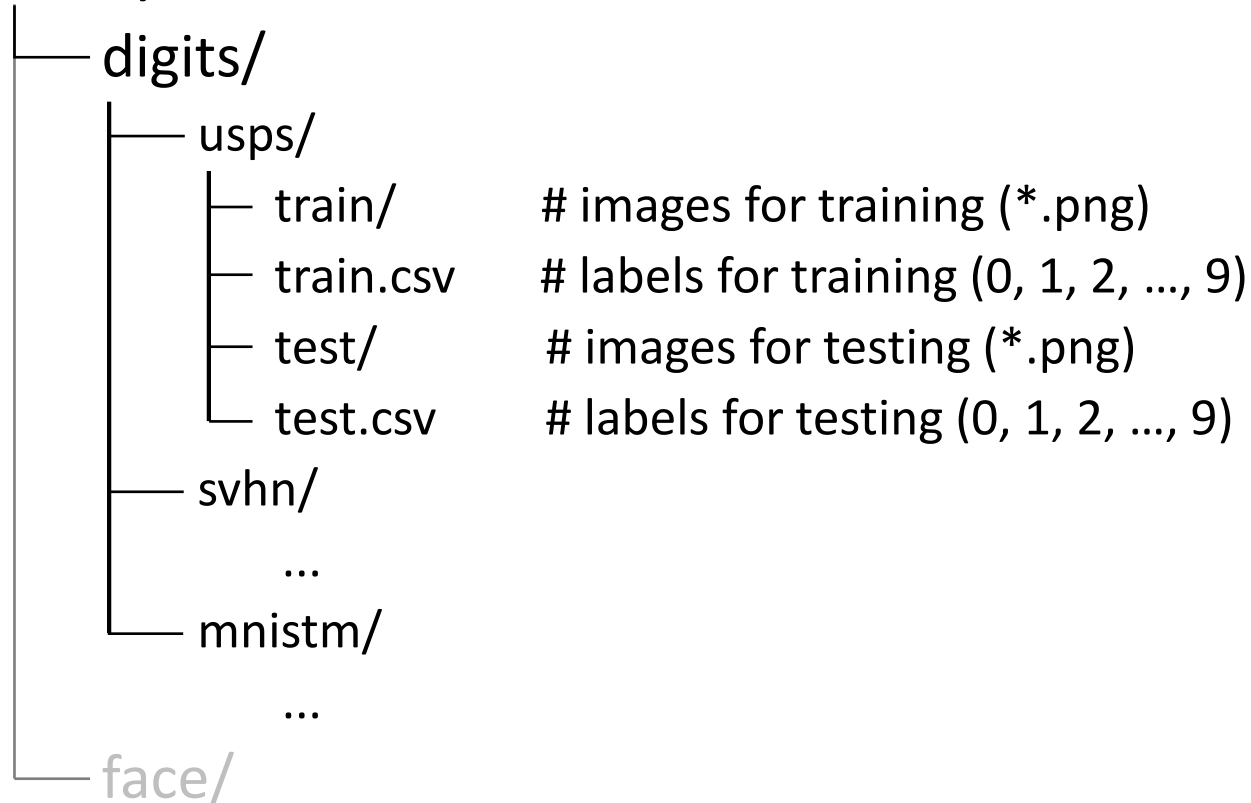
You need to resize the images of the above three datasets to **28 * 28 * 3 pixels!**



Dataset – Digits

Format

data/



Outline

- Problems & Grading
- Dataset
- Rules
- Training Tips

Rules – Deadline & Policy

- Report and source code deadline: **110/11/23 (Tue.) 03:00 AM (GMT+8)**
- Late policy : Up to 3 free late days in a semester (depends on your hw0 result). After that, late homework will be deducted 30% each day
- Taking any unfair advantages over other class members (or letting anyone do so) is strictly prohibited. Violating university policy would result in **F** for this course.
- Students are encouraged to discuss the homework assignments, but you must complete the assignment by yourself. TA will compare the similarity of everyone's homework. Any form of cheating or plagiarism will not be tolerated, which will also result in F for students with such misconduct.

Rules – Deadline & Policy

- Searching for online materials or discussing with fellow classmates are highly encouraged. However, you must provide the code or solution by yourself.
- Please specify, if any, the references for any parts of your HW solution in your report (e.g., the name and student ID of your collaborators and/or the Internet URL you consult with). If you complete the assignment all by yourself, you must also specify “no collaborators”.

Rules – Deadline & Policy

- For Problems 3 & Bonus, **do not use the testing data to select your models**. You can split the training data into training / validation sets and use the validation set to select models.
- Using external dataset is **forbidden** for this homework.

Rules – Submission

- Click the following link and sign in to your GitHub account to get your submission repository:

https://classroom.github.com/a/AsTh_ZLM

- After getting your GitHub repository (which should be named "hw2-<username>"), be sure to **read the README carefully** before starting your work.
- **By default, we will only grade your last submission before the deadline (NOT your last submission).** Please e-mail the TAs if you'd like to submit another version of your repository and let us know which commit to grade.
- **We will clone the `main` branch of your repository.**

Rules – Submission

- Your GitHub repository should include the following files:
 - hw2_<studentID>.pdf
 - hw2_p1.sh (for Problem 1)
 - hw2_p2.sh (for Problem 2)
 - hw2_p3.sh (for Problem 3)
 - hw2_bonus.sh (for Bonus)
 - your python files (e.g., training code & testing code)
 - your model files (can be loaded by your python file)
- **For more information, please check the README.md in your repository.**
- **Don't upload your dataset.**
- If any of the file format is wrong, you will get zero point.

Rules – Submission

- If your model is larger than GitHub's maximum capacity (100MB), you can upload your model to another cloud service (e.g., Dropbox). However, your script file should be able to download the model **automatically**.
 - Dropbox tutorial:
<https://drive.google.com/file/d/1XOz69Mgxo67IZNQWnRSjT2eZAZtpUAgZ/view>
- Do not delete your trained model before the TAs disclose your homework score and before you make sure that your score is correct.
- Use the **wget** command in your script to download you model files. Do not use the curl command.

Rules – Bash Script

- TA will run your code as shown below:
 - `bash ./hw2_p1.sh $1`
 - **\$1**: path to the folder for your 1000 generated images (Problem 1) (e.g. *~/hw2/GAN/output_images*).
 - `bash ./hw2_p2.sh $1`
 - **\$1**: path to the folder for your 1000 generated images (Problem 2) (e.g. *~/hw2/ACGAN/output_images*).
 - After running your code, the 1000 generated images are saved in \$1.

 You should **follow the filename format** for different digit images in Problem 2

Rules – Bash Script (cont'd)

- TA will run your code as shown below:
 - `bash ./hw2_p3.sh $1 $2 $3`
 - `bash ./hw2_bonus.sh $1 $2 $3`
 - **\$1**: path to testing images in the target domain (e.g. *~/hw2_data/digits/mnistm/test*).
 - **\$2**: a string that indicates the name of the target domain, which will be either *mnistm*, *usps* or *svhn*.
 - **\$3**: path to your output prediction file (e.g. *~/test_pred.csv*).
 - After running your code, the predicted labels for each testing images in target domain are saved in test_pred.csv
 - **The format inside test_pred.csv should be same as test.csv.**

Rules – Bash Script (cont'd)

- Note that you should **NOT** hard code any path in your file or script.
- Your testing code have to be finished in **10 mins**.
- You must **not** use commands such as **rm**, **sudo**, **CUDA_VISIBLE_DEVICES**, **cp**, **mv**, **mkdir**, **cd**, **pip** or other commands to change the Linux environment.
- In your submitted script, please use the command **python3** to execute your testing python files.
 - For example: `python3 test.py < --img_dir $1> < --save_dir $2>`

Rules – Testing Environment

- Environment
 - Ubuntu 20.04 LTS
 - GPU: 2080 Ti
 - GPU memory: 11 GB
 - GPU driver version: 450.119.03
 - CUDA version: 11.0
- Although our GPU has 11GB, please utilize only **8GB** for testing.
- We will execute your code on **Linux** system, so try to make sure your code can be executed on Linux system before submitting your homework.

Rules - Packages

- python: 3.8
 - numpy: 1.21.2
 - pytorch: 1.10.0
 - torchvision: 0.11.1
 - cudatoolkit: 11.3.1
 - scikit-learn: 0.23.2
 - and other standard python packages
 - **E-mail or ask TA first if you want to import other packages.**
- matplotlib: 3.4.3
 - pillow: 8.4.0
 - imageio: 2.9.0
 - scipy: 1.6.2
 - scikit-image: 0.18.1
 - pandas: 1.1.3

Rules - Packages

- Do not use `imshow()` or `show()` in your code or your code will crash.
- Use **`os.path.join`** to deal with path as often as possible.

Rules - Penalty

For **Problem 1, 2, 3-2 and Bonus**:

- If we can not reproduce your generated image/accuracy on the testing set, you will receive a 50% penalty in the sub-problem.
- If we can not execute your code, we will give you a chance to make minor modifications to your code. After you modify your code,
 - If we can execute your code and reproduce your results on the testing set, you will still receive a 30% penalty in the sub-problem.
 - If we can execute your code but cannot reproduce your results on the testing set, you will receive a 50% penalty in the sub-problem.
 - If we still cannot execute your code, you will get 0 in the sub-problem.

Reminder

- Please start working on this homework as early as possible.
- The training may take a few hours on a GPU or days on CPUs.
- Please read and follow the HW rules carefully.
- If not sure, please ask your TAs!

How to find help

- Google!
- Use TA hours (please check [course website](#) for time/location)
- Post your question under HW2 Q&A section in FB group
- Contact TAs by e-mail: ntudlcv@gmail.com

DOs and DONTs for the TAs (& Instructor)

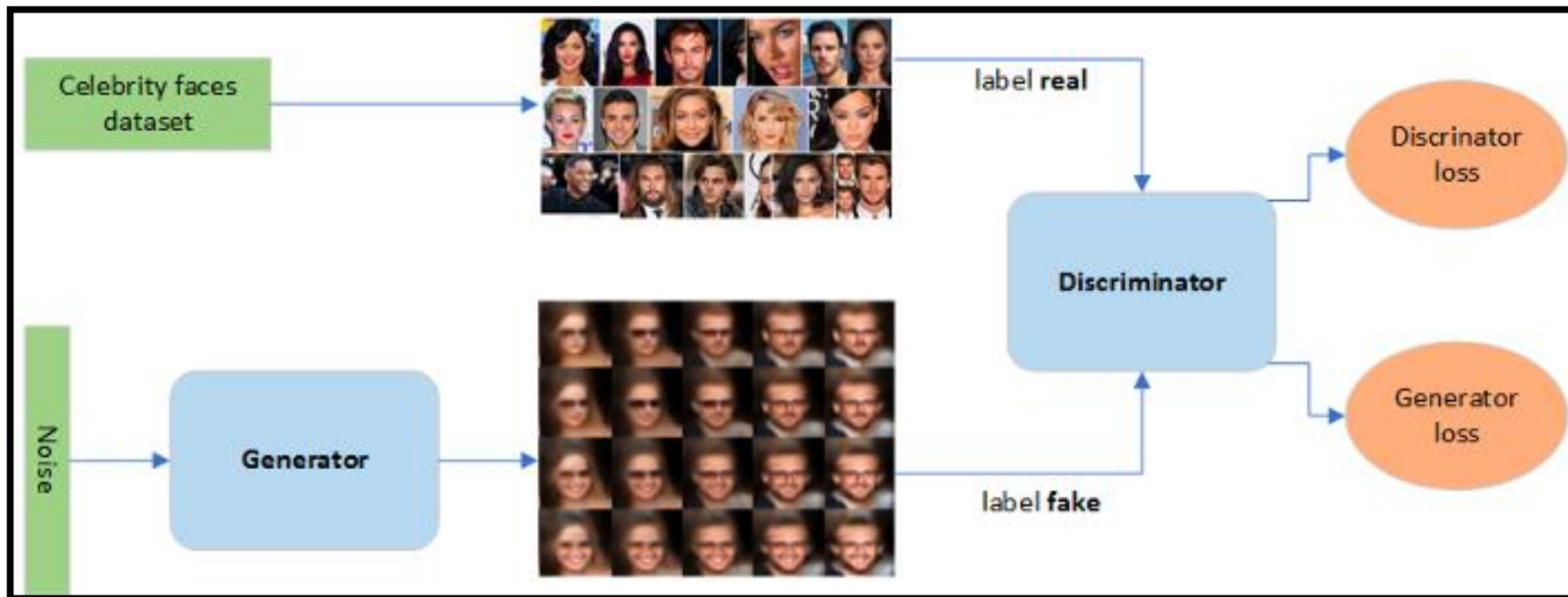
- Do NOT send private messages to TAs via Facebook.
 - TAs are happy to help, but they are not your tutors 24/7.
- TAs will NOT debug for you, including addressing coding, environmental, library dependency problems.
- TAs do NOT answer questions not related to the course.
- If you cannot make the TA hours, please email the TAs to schedule an appointment instead of stopping by the lab directly.

Outline

- Problems & Grading
- Dataset
- Rules
- Training Tips

Training Tips - GAN

- Architecture of GAN

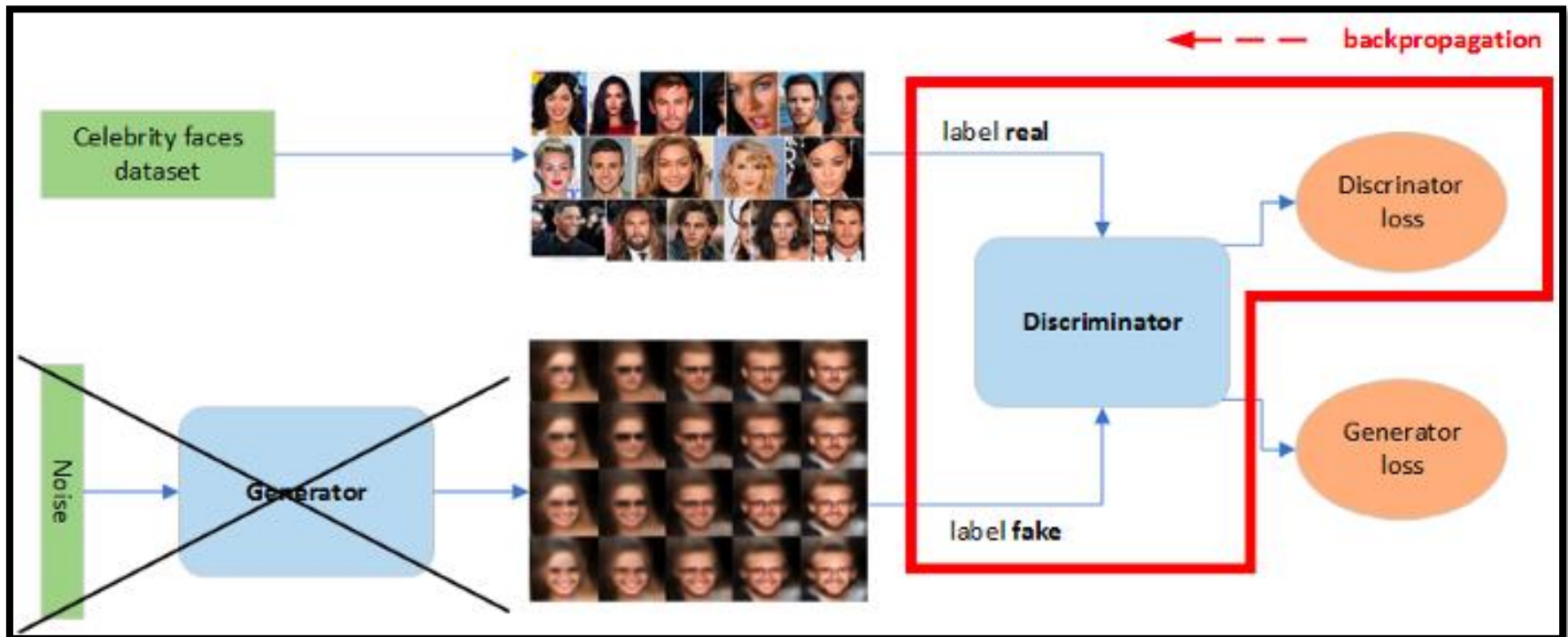


Training Tips - GAN (cont'd)

- Training

- (1) Discriminator

- The generator's weights are fixed. Only the discriminator's weights are updated. Both real and fake image data are observed during training. The discriminator learns to detect fake image inputs.



Click the link below for more details:

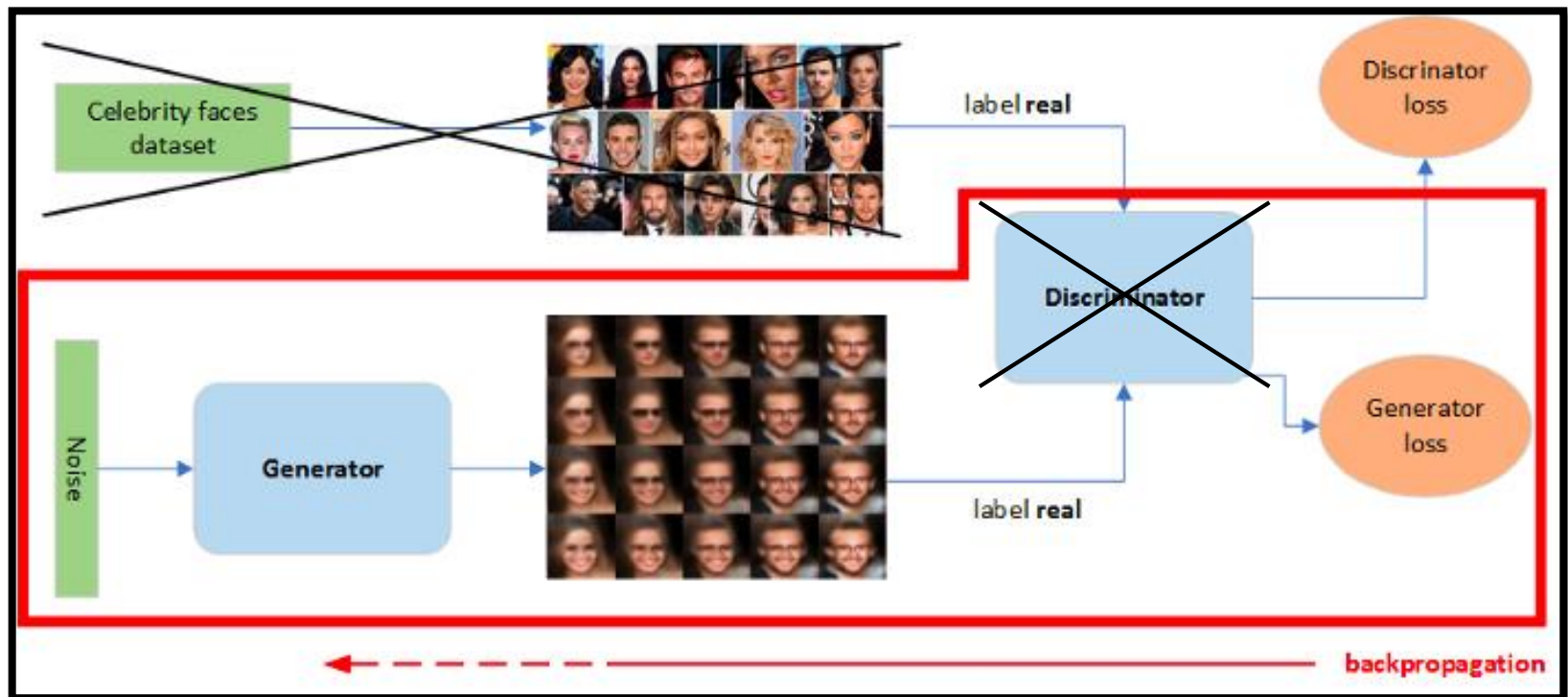
Reference: [GAN for dummies](#)

Training Tips - GAN (cont'd)

- Training

- (2) Generator

- The discriminator's weights are fixed. Only the generator's weights are updated. The generator learns to fool the discriminator.



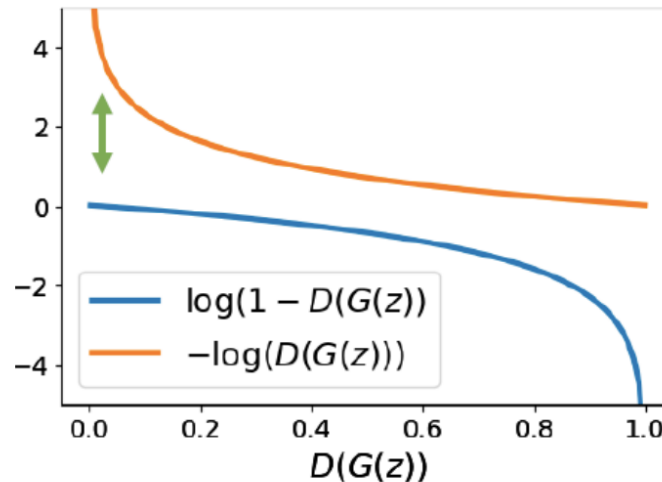
Click the link below for more details:
Reference: [GAN for dummies](#)

Training Tips - GAN Loss function (cont'd)

Potential Problem

$$\min_{\mathbf{G}} \max_{\mathbf{D}} \left(E_{x \sim p_{data}} [\log \mathbf{D}(x)] + E_{z \sim p(z)} \left[\log \left(1 - \mathbf{D}(\mathbf{G}(z)) \right) \right] \right)$$

- At start of training, G is not OK and D easily tells apart real/fake data (i.e., $D(G(z))$ close to 0)
- Solution:
 - Instead of training G to minimize $\log(1-D(z))$ in the beginning, we train G to minimize $-\log(D(G(z)))$.
 - With strong gradients from G, we start the training of the above min-max game.



Training Tips - GAN (cont'd)

- Architecture Guideline proposed by Radford et al. in [DCGAN](#)

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Well known [tips and tricks](#) published on GitHub.

It is suggested that you take a look before you start training.

TA's experience and hints

- Surveying related papers and using similar architectures may help.
- Trace the accuracy of discriminator network to see if G_{net} and D_{net} performance matches.
- Improved GAN algorithm is harder to implement but easier to train (e.g. WGAN, WGAN-GP)
- Use your VAE's decoder/encoder as the generator/discriminator prototype.

GAN is often difficult to train and tune, starting this part early may help a lot.