CMPUT 328 Fall 2020
Assignment 5
Semi-Supervised Learning with GAN

Goal: Implement a semi-supervised learning algorithm using features from the discriminator of a Generative Adversarial Network (GAN) by adding a classification branch to it.

Introduction: In this assignment, you need to design GAN with a classification branch attached to the discriminator network so that the two will share some layers.

The new loss will be a combination of the binary cross entropy loss of the discriminator itself and the cross-entropy loss of the classification branch. There is one catch, though. In the semi-supervised learning problem, only a part of the data has classification labels on it. For example, in this assignment, **only 20% of images have labels** for training. classification component of the loss will be zero for unlabeled images.

Task: You are provided with three files: **A5_main.py, A5_utils.py** and **A5_submission.py**. Your task is to complete the classes SharedNet, Discriminator, Classifier, Generator and CompositeLoss in **A5_submission.py**
You might also want to modify the respective parameter classes for these modules as well as TrainParams and OptimizerParams and SaveCriteria to customize the training process.
You can also add any other classes or functions you need but **make sure that A5_submission.py is the only file you modify and submit.**
SharedNet contains the shared layers between Discriminator and Classifier and its output is used as input to both.
CompositeLoss takes the classifier and discriminator losses as input and combines them to produce the overall loss that will be used to train them. In the simplest case, it can be a weighted sum of the two, but you are free to design it anyway you want.
SaveCriteria can be used to customize when to save new checkpoints using training and validation metrics.

You will be training and testing your model on the **CIFAR-10** dataset.
The training set of 50K images is split into 38K for training and 12K for validation. **20%** of the training images (or 7600 images) have labels for the classification loss. The test set of 10K images will be used for evaluation.
The training, validation, and testing code along with the splitting of labeled/unlabeled data is provided in **A5_main.py and A5_utils.py**.

It is up to you to decide how the discriminator-classifier sharing is designed except that **at least 50% of the parameters must be shared**, i.e. the number of trainable parameters in SharedNet must **not be less** than that in either Discriminator or Classifier. **A5_main.py w**ill raise an error if this condition is not satisfied.
There are no other constraints on the architecture of the GAN.

An interesting case is where the discriminator and classifier share *all* their parameters so that the discriminator not only learns to distinguish between real and fake samples but also to classify each into a corresponding class.
To allow this, **A5_main.py** assigns a distinct class label to each fake image as well depending on the class of the corresponding real image.
As a result, there are 20 classes out of which 10 are for real images (0 – 9) and the remaining (10 – 19) are for fake images. For instance, classes 12 and 15 correspond to fake images of objects in classes 2 and 5.
In addition, class labels -1 and -2 are used to indicate unlabelled real and fake images respectively.

The training regime itself is mostly borrowed from this tutorial except that the discriminator training part is replaced with discriminator + classifier training.
You are advised to avoid copying architectural details from this tutorial but if you do, make sure to document it in sufficient detail to make your understanding of the same clear to the marking TA.
You are expected to experiment with many different configurations of how the discriminator and classifier share weights and report your findings in the documentation.

GANs are known to be slow and difficult to train so it might be better to keep the architectures simple. It might also help to disable the classifier first and get only the GAN part working and then add the classifier as well. There are also some debugging parameters in A5_Params that might help to speed up the process.

These resources might help you choose better training settings:
Improved Techniques for Training GANs
Tips for Training Stable Generative Adversarial Networks

Marking: 1/3 of the marks are for documentation and the remaining 2/3 are for the code which will be awarded based on relative performance in terms of the following metrics:

1. Classification accuracy on the test set (higher is better)
2. Inception Score of the generator images (higher is better)
3. Fréchet Inception Distance of the generator images compared with the test set (lower is better)
4. Test runtime (lower is better)

The evaluation is designed like a competition where all the valid submissions will be ranked by each metric and a linear scale will be used to award marks ranging from 50% for the worst performing submission to 100% for the best.
The overall marks will be obtained by averaging over the four metrics.
More information about the generator evaluation metrics can be found in these papers:
Pros and Cons of GAN Evaluation Measures
An empirical study on evaluation metrics of generative adversarial networks

Note that the **test runtime should not exceed 30 minutes** using any GPU on Colab. There is no limit on training time.

What to Submit: You need to submit a single **zip file** with the modified **A5_submission.py** along with the **checkpoints** folder containing trained weights that will be loaded for testing.

All submissions will be used in an automated evaluation system and any failure to run, for whatever reason, will be heavily penalized and might lead to the submission being disqualified from ranking and getting a zero.

Even though most submissions will not be trained as part of the evaluation, all parameter settings must be correct to be able to reproduce the submitted checkpoint if needed. Any other information about the training process must likewise be included in the documentation. Any submission whose training process fails to produce a checkpoint matching the submitted one will be disqualified from ranking and get a zero for the code section.

Check before submitting:

- A5_main.py can save multiple checkpoints in each run but **only the one to be loaded for testing** should be included for each dataset.
- The checkpoint file must be named model.pt.* where the last suffix (indicating the epoch) is optional.
- Add all import statements in A5_submission.py needed to make it work as an imported module from A5_main.py.
- Remove any Colab specific lines (e.g. mounting Google drive) or any other code that can produce an error when it is imported.
- Set TrainParams.weights_path to the checkpoint path relative to A5_submission.py. It is absolutely essential that the name and location of the saved checkpoint exactly match this parameter.
- Do NOT include any file other than A5_submission.py and checkpoints folder and do not rename the former in any way. Also, both must be in the root of the zip file (instead of being contained inside another folder).
- Submit a zip file instead of any other archive format (e.g. rar, tar or tar.gz)
- Eclass has a maximum upload size limit of 400 MB so your network must be small enough to fit in this size. Google Drive (or any cloud service) links for the checkpoint will not be accepted.

To reiterate, there will be heavy penalty for failing to follow any of these instructions.

Submission deadline is **November 15, 11:59 PM**.