[Spring 2024] CS172 Assignment 1
Basics in computer vision

#NAME #StudentID

April 3, 2024

Acknowledgements

1. Deadline: **2024/4/30 23:59:00**. Late Policy please refer to the course slides.
2. Please submit your assignment in **Gradescope** in PDF format. Registration code: 4GNJPE
   - Giving your report in English, a report in Chinese is not accepted.
   - Handwritten homework is not accepted and we highly recommend using LaTeX. The LaTeX template has been uploaded to Blackboard.
   - Note that you **MUST** select pages for each question on Gradescope, and TA will **NEVER** select pages for you. Otherwise, you will lose **ALL** the points.
3. Please upload your code zip to the **ShanghaiTech cloud** disk.
   - https://epan.shanghaitech.edu.cn/l/CFKM20
   - All source files and *readme* should be included but remember to remove the datasets.
   - Your zip should be named as CS172_NAME_ID_hw1.zip.
4. Plagiarism or cheating is strictly prohibited.
   - DO NOT share your assignment or code!
   - NO fake solution is allowed!
   - Make sure your codes can run and are consistent with your solutions.

1 [25 points] Bag-of-Words model

BoW can be applied to image classification, image retrieval, etc. The general process is to first extract image features, then cluster the features, build a codebook based on the clustering results, and obtain the vector corresponding to each image through feature quantization and pooling. We require using **SIFT** to extract image features, and for clustering, we recommend **K-means**, but you are free to try any other cluster algorithm. Use **animals(dog, cat, pandas)** dataset that we offered in the compressed package. Here are some tutorials you can refer to when processing data.

1. Write a function that can read an image from the dataset and display it with the corresponding label.
2. Uniform image size. Select an image and apply at least two image augmentation methods (like flip, rotation, blur, mixup, etc.) to it. Show the results.
3. Compute SIFT for all the images and select five pictures to visualize the results after SIFT. We suggest saving the features on disks after each iteration to avoid memory issues.
4. For all the features, apply with the K-Means algorithm. Discuss the influence of different values of k on the results.
5. Use the BoW model to quantize features and represent by frequencies.

2 [25 points] SVM classification

A Support Vector Machine (SVM) is a machine learning algorithm that can be used for classification or regression tasks. In the context of classification, the SVM algorithm **finds a hyperplane** that separates the data points in different classes with the maximum margin. The hyperplane is defined by a set of weights and biases that are learned from the training data. **In this section, you need to use the results obtained in the previous section, and you are required to:**

1. Implement an SVM classifier with hinge loss.
2. Train your model with the extracted BoW features and try to improve the performance.
3. Explain the backward pass in the training of the SVM classifier.

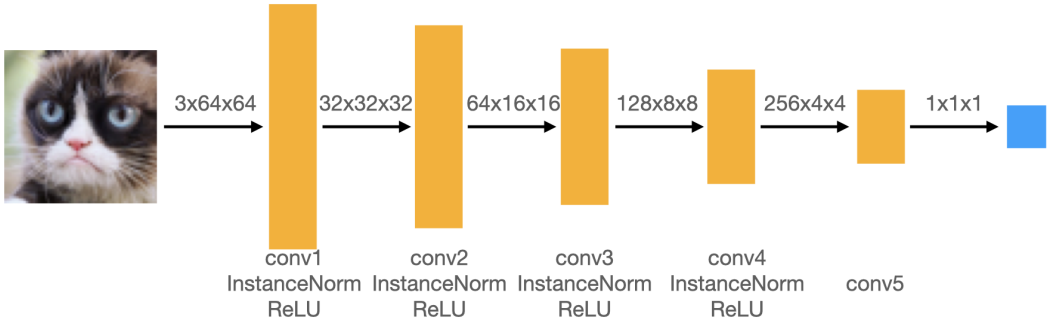
Hint: Principal component analysis (PCA) is useful to reduce feature dimension and make features condensed. Try to add the PCA function and compare these two versions.

3 [50 points] Deep Convolutional Generative Adversarial Networks

A generative adversarial network (GAN) is a class of machine learning frameworks and a prominent framework for approaching generative AI. In this section, we will train the DCGAN to generate grumpy cats from samples of random noise. 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. **The code and data required for this section are in the compressed package.**

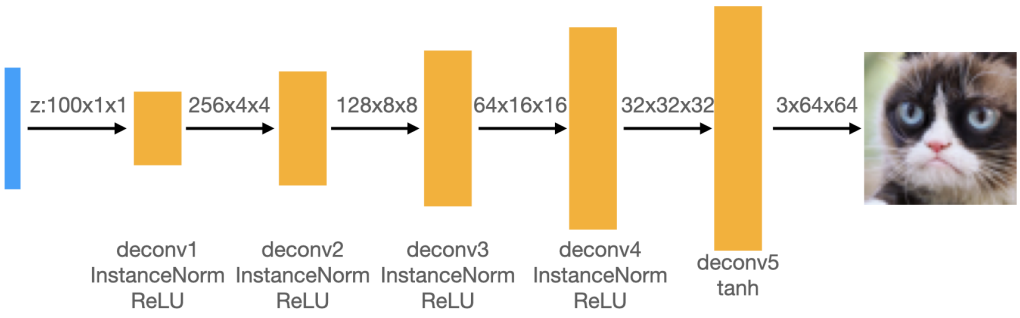
1. Data Augmentation. DCGAN will perform poorly without data augmentation on a small-sized dataset because the discriminator can easily overfit to a real dataset. To rescue, we need to add some data augmentation such as random crop and random horizontal flip. You need to fill in the deluxe version of data augmentation in **data_loader.py**. We provide some scripts for you to begin with. You need to compose them into a transform object which is passed to **CustomDataset**.

2. Discriminator. The discriminator in this DCGAN is a convolutional neural network with the following architecture:



   - Padding. In each of the convolutional layers shown above, we downsample the spatial dimension of the input volume by a factor of 2. Given that we use kernel size K = 4 and stride S = 2, what should the padding be? Write your answer on your website, and show your work (e.g., the formula you used to derive the padding).
   - Implementation. Implement this architecture by filling in the **__init__** and forward method of the **DCDiscriminator** class in **models.py**, shown below. The **conv_dim** argument does not need to be changed unless you are using larger images, as it should specify the initial image size. Note the function **conv** is defined in **models.py**.

3. Generator. Now, we will implement the generator of the DCGAN, which consists of a sequence of transposed convolutional layers that progressively upsample the input noise sample to generate a fake image. The generator in this DCGAN has the following architecture:



   - Implementation. Implement this architecture by filling in the **__init__** and forward method of the **DCGenerator** class in **models.py**. Note: Use the **deconv** function (analogous to the **conv** function used for the discriminator above) in your generator implementation.

4. Training Loop. Next, you will implement the training loop for the DCGAN. A DCGAN is simply a GAN with a specific type of generator and discriminator; thus, 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.

Algorithm 1 GAN Training Loop Pseudocode

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

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

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

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

  Update the parameters of the generator

- Implementation. Open up the file **vanilla_gan.py** and fill in the indicated parts of the **training_loop** function. There are 5 numbered bullets in the code to fill in for the discriminator and 3 bullets for the generator. Each of these can be done in a single line of code, although you will not lose marks for using multiple lines.

5. Experiment. Train the DCGAN with the command:

```
python vanilla_gan.py —num_epochs=100
```

The script saves the output of the generator for a fixed noise sample every 200 iterations throughout training; this allows you to see how the generator improves over time. More command line parameters can be found in **vanilla_gan.py**.

- Screenshots of discriminator and generator training loss with both **data_aug=basic/deluxe** – 4 curves in total. Briefly explain what the curves should look like if GAN manages to train.
- Set data augmentation to deluxe and then show one of the samples from early in training (e.g., iteration 200) and one of the samples from later in training, and give the iteration number for those samples. Briefly comment on the quality of the samples, and in what way they improve through training. (It typically requires more than 3,000 iterations to attain a visually appealing effect. Pictures in the early stages of training will be blurry.)

6. Discussion.

- DCGAN is more stable than traditional GAN in training, but there are still certain problems. What problems did you find during this training?
- How do you think it should be improved? Do other generative models (VAE, Diffusion) also have such problems?