電腦視覺與深度學習
(Computer Vision and Deep Learning)

Homework 2

TA:

Gmail: nckubot65904@gmail.com

Office Hour: 14:00~16:00, Mon.

10:00~12:00, Thu.

At CSIE 9F Robotics Lab.

Notice (1/2)

- Copying homework is strictly prohibited!! Penalty: Both individuals will receive a score of 0!!
- Due date ➔ 09:00:00, 2024/12/27 (Fri.)

  Do not submit late, or the following points will be deducted:
  - Submit within seven days after the deadline, and your score will be reduced by half.
  - If you submit after this period, you will receive a score of 0.
- You must attend the demonstration, otherwise your score will be 0. The demonstration schedule will be announced on NCKU Moodle.
- You must create GUI, otherwise your point will be deducted.
- Upload to ➔ 140.116.154.28 → Upload/Homework/Hw2
  - User ID: cvdl2024 Password: RL2024cvdl
- Format
  - Filename: Hw2_StudentID_Name_Version.zip
    - Ex: Hw2_F71234567_林小明_V1.zip
    - If you want to update your file, you should update your version to be V2,
    - Ex: Hw2_F71234567_林小明_V2.zip
  - Content: Project folder *(Excluding the pictures)

    *Note: Remove your “Debug” folder to reduce file size.

Notice (2/2)

- Python (recommended):
  - Python 3.10
  - Matplotlib 3.8.0
  - UI framework: pyqt5 (5.15.11)
  - Torch 2.5.1
  - Torchvision 0.20.1
  - Torchsummary 1.5.1

Assignment Scoring (Total: 100%)

1. (50%) Training a CIFAR10 Classifier Using VGG19 with BN (出題：Jun, Tim)
   - 1.1 (10%) Load CIFAR10 and show 9 augmented images with labels.
   - 1.2 (10%) Load model and show model structure.
   - 1.3 (15%) Show training/validating accuracy and loss.
   - 1.4 (15%) Use the model with highest validation accuracy to run inference, show the predicted distribution and class label.
2. (50%) Training a MNIST Generator Using DcGAN (出題：Neil, Alan)
   - 2.1 (10%) Load MNIST and show training images.
   - 2.2 (10%) Load Model and show model structure.
   - 2.3 (15%) Show training loss.
   - 2.4 (15%) Show the real images and fake images using Generator.

1. Training a CIFAR10 Classifier Using VGG19 with BN (50%)

1.1 Load CIFAR10 and show 9 augmented images with labels. (10%) (出題： James, Tim)

1.2 Load model and show model structure. (10%)

1.3 Show training/validating accuracy and loss. (15%)

1.4 Use the model with highest validation accuracy to run inference, show the predicted distribution and class label. (15%)

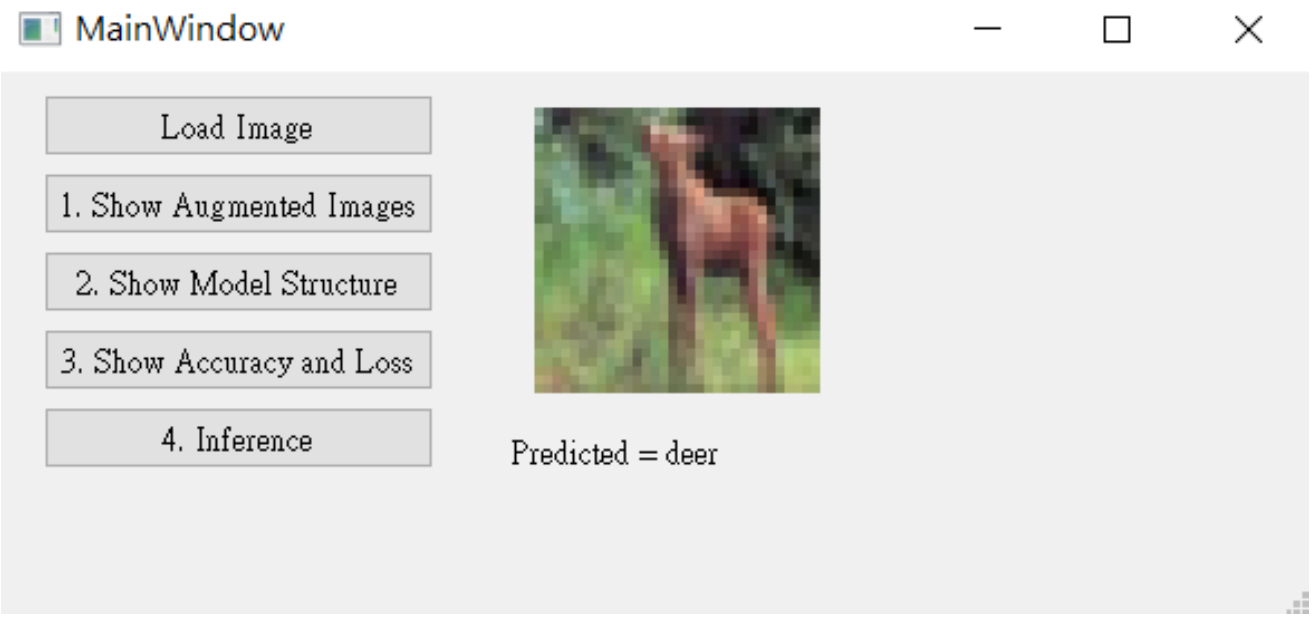


Figure: GUI example

1. Training a CIFAR10 Classifier Using VGG19 with BN (50%)

(出題： James, Tim)

1. Objective

1) Learn how to train a VGG19 with BN (Batch Normalization) model to classify 10 different classes images of CIFAR10.

2. VGG19 with BN

1) VGG19: A convolutional neural network that is 19 layers deep.
2) BN (Batch Normalization): used to make training of artificial neural networks faster and more stable.

3. CIFAR10

1) A collection of 60,000 32x32 color images in 10 different classes that is commonly used to train machine learning and computer vision algorithms.
2) 10 classes:
   airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck
3) Datasets
   (1) Training dataset: 50000 images in total.
   (2) Validation dataset: 10000 images in total.
   (3) Testing dataset: 10 images in total. (Generating from validation dataset.)

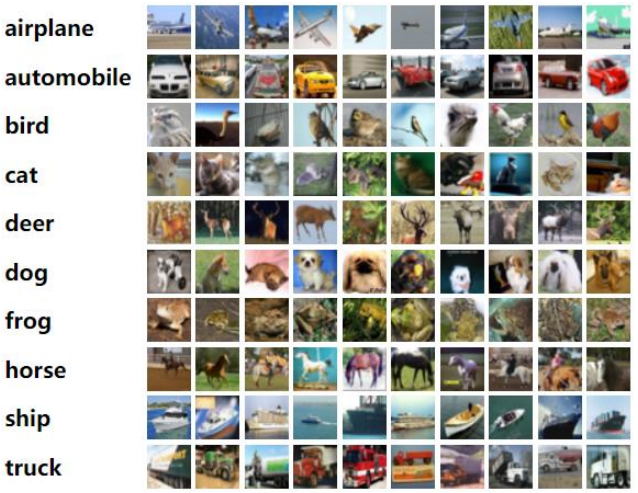


*Figure*1: CIFAR10

R. Reference

1) VGG19
2) Batch Normalization

1. Training a CIFAR10 Classifier Using VGG19 with BN (50%)

4. Requirements

(出題： James, Tim)

1) Train VGG model with batch normalization (BN) using PyTorch.
2) In the submitted file, you need to include
   A. Weight file for VGG19 with BN in .pth format. (File size is approximately 540MB)
   B. Figure of training/validating loss and accuracy in .jpg or .png format.
   C. Code for your GUI program
   D. Code for model training.
3) Please do not include image data in the submitted file.

5. Homework Images

1) There are 2 different folders in 'Q1_image'.
2) In the subfolder 'Q1_image/Q1_1,' there are 9 different images used in Q1-1. When demoing, use the same images.
3) In the subfolder 'Q1_image/Q1_4,' there are 9 different images used in Q1-4. These images are used for testing your program. When demoing, we will use different images for the demonstration.

1.1 Show 9 Augmented Images with Labels (10%)

(出題： James, Tim)

Q1.1

1) At home:

(1) Use PIL.Image.open() form Pillow package to load 9 images in /Q1_image/Q1_1/ folder.

(2) Apply at least 3 different type of data augmentation (tutorial).

A. transforms.RandomHorizontalFlip()

B. transforms.RandomVerticalFlip()

C. transforms.RandomRotation(30)

Notice: This is an example; you can use different data augmentation techniques

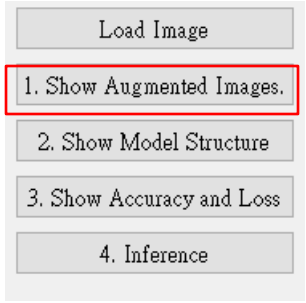
2) When the demo:

(1) Click the button "1. Show Augmentation Images"

(2) Load 9 images in /Q1_image/Q1_1/ folder

(3) Apply data augmentation on 9 images.

(4) Show 9 augmented images with label in a new window



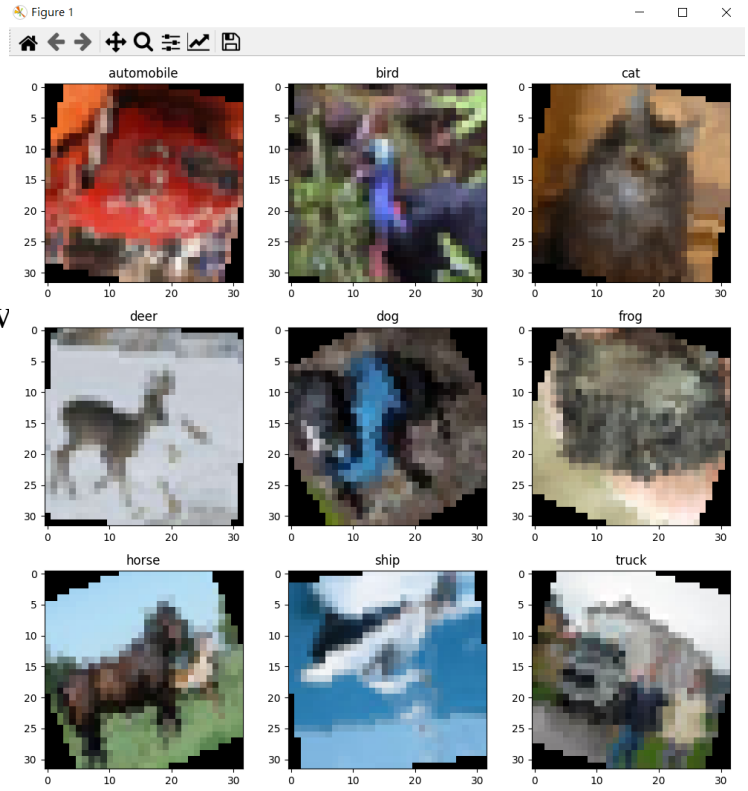


Figure1: 9 Augmented images

Notice: this is an example, the images might differ

1.2 Show the Structure of VGG19 with BN (10%)

(出題：Jun, Tim)

Q1.2

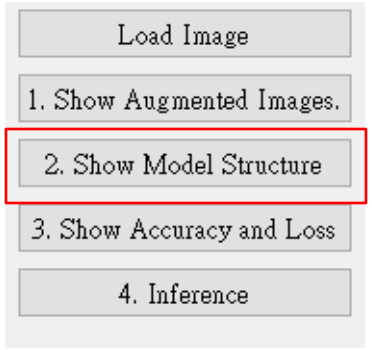
1) **At home:**
   (1) Use torchvision.models.vgg19_bn(num_classes=10) to build a VGG19 with batch normalization (BN) model.
   (2) Use torchsummary.summary to show the structure in the terminal.
2) **When the demo:**
   (1) Click the button "2. Show Model Structure"
   (2) Run the function to show the structure in the terminal.

The -1 indicates that the actual size of batch size can vary.

Feature map shape (Batch, Channels, Height, Width)

Layer (type)

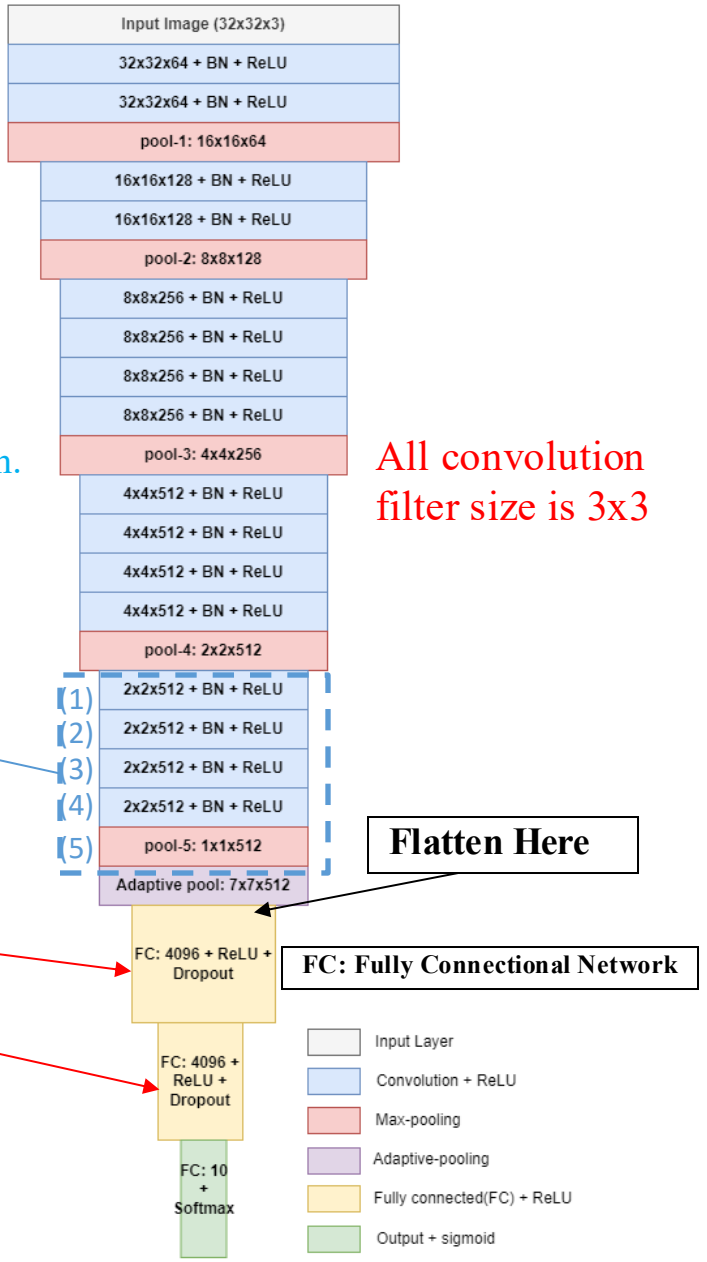
Num. of param.




```
       BatchNorm2d-38          [-1, 512, 4, 4]           1,024
              ReLU-39          [-1, 512, 4, 4]               0
         MaxPool2d-40          [-1, 512, 2, 2]               0
            Conv2d-41          [-1, 512, 2, 2]       2,359,808
       BatchNorm2d-42          [-1, 512, 2, 2]           1,024
              ReLU-43          [-1, 512, 2, 2]               0
            Conv2d-44          [-1, 512, 2, 2]       2,359,808
       BatchNorm2d-45          [-1, 512, 2, 2]           1,024
              ReLU-46          [-1, 512, 2, 2]               0
            Conv2d-47          [-1, 512, 2, 2]       2,359,808
       BatchNorm2d-48          [-1, 512, 2, 2]           1,024
              ReLU-49          [-1, 512, 2, 2]               0
            Conv2d-50          [-1, 512, 2, 2]       2,359,808
       BatchNorm2d-51          [-1, 512, 2, 2]           1,024
              ReLU-52          [-1, 512, 2, 2]               0
         MaxPool2d-53          [-1, 512, 1, 1]               0
AdaptiveAvgPool2d-54          [-1, 512, 7, 7]               0
            Linear-55                [-1, 4096]     102,764,544
              ReLU-56                [-1, 4096]               0
           Dropout-57                [-1, 4096]               0
            Linear-58                [-1, 4096]      16,781,312
              ReLU-59                [-1, 4096]               0
           Dropout-60                [-1, 4096]               0
            Linear-61                  [-1, 10]          40,970
================================================================
Total params: 139,622,218
Trainable params: 139,622,218
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 7.55
Params size (MB): 532.62
Estimated Total Size (MB): 540.18
----------------------------------------------------------------
```

(1) (2) (3) (4) (5)

Figure: the Structure of VGG19 with BN



All convolution filter size is 3x3

Figure: VGG19 with BN model structure

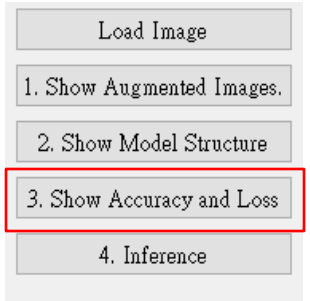
1.3 Show Training/Validating Accuracy and Loss (15%)

Q1.3

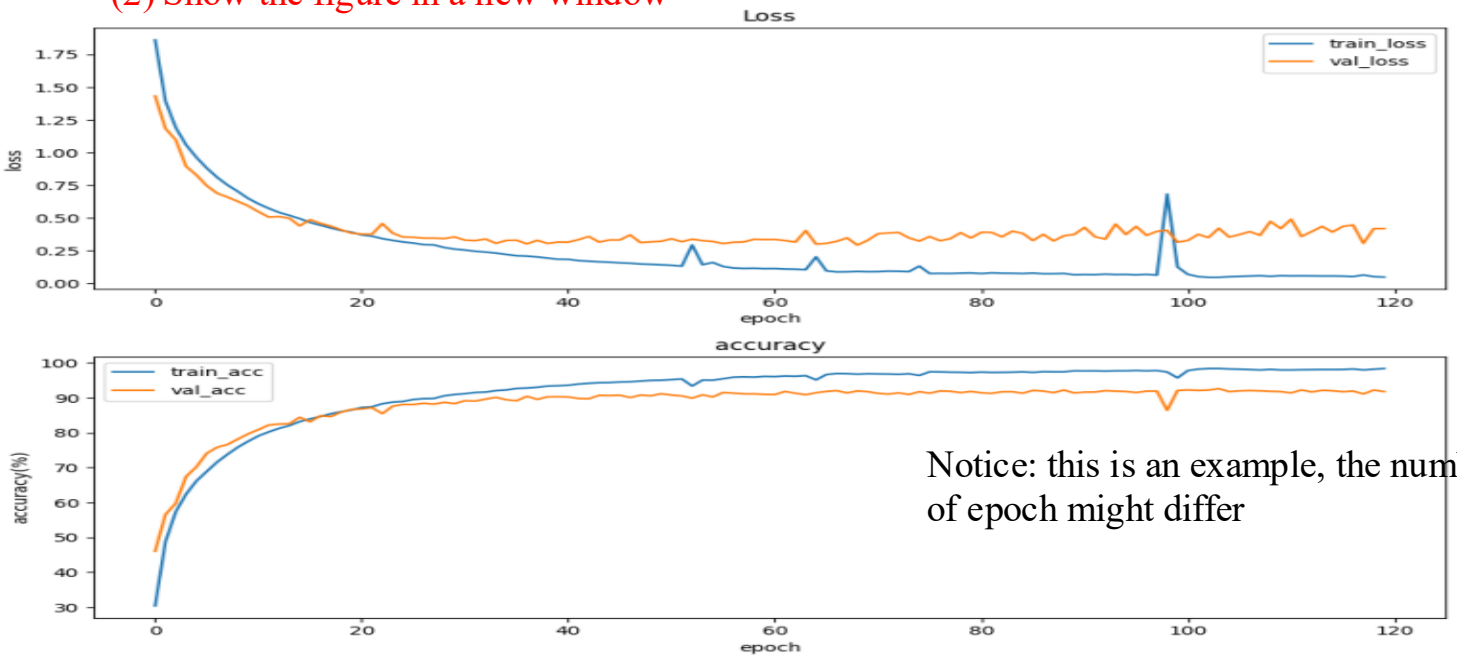
(出題： James, Tim)

1) **At home:**
   (1) Use torchvision.datasets.CIFAR10 to load the training and validation datasets. (tutorial)
   (2) Training and validating VGG19 with BN at least 40 epochs at home (tutorial) and record the training/validating accuracy and loss in each epoch (tutorial).
   (3) Notice: If your validation accuracy is low, you can try
      A. Adjust the learning rate of the optimizer.
      B. Change the data augmentation techniques used.
   (4) Save weight file with highest validation accuracy .
   (5) Use matplotlib.pyplot.plot() to create a line chart for the training and validating loss and accuracy values.
   (6) Save the figure in .jpg or .png format.
2) **When the demo:**
   (1) Click the button "3. Show Accuracy and Loss"
   (2) Show the saved figure of Training/Validating loss and accuracy in a new window

(1) Click the button.



(2) Show the figure in a new window



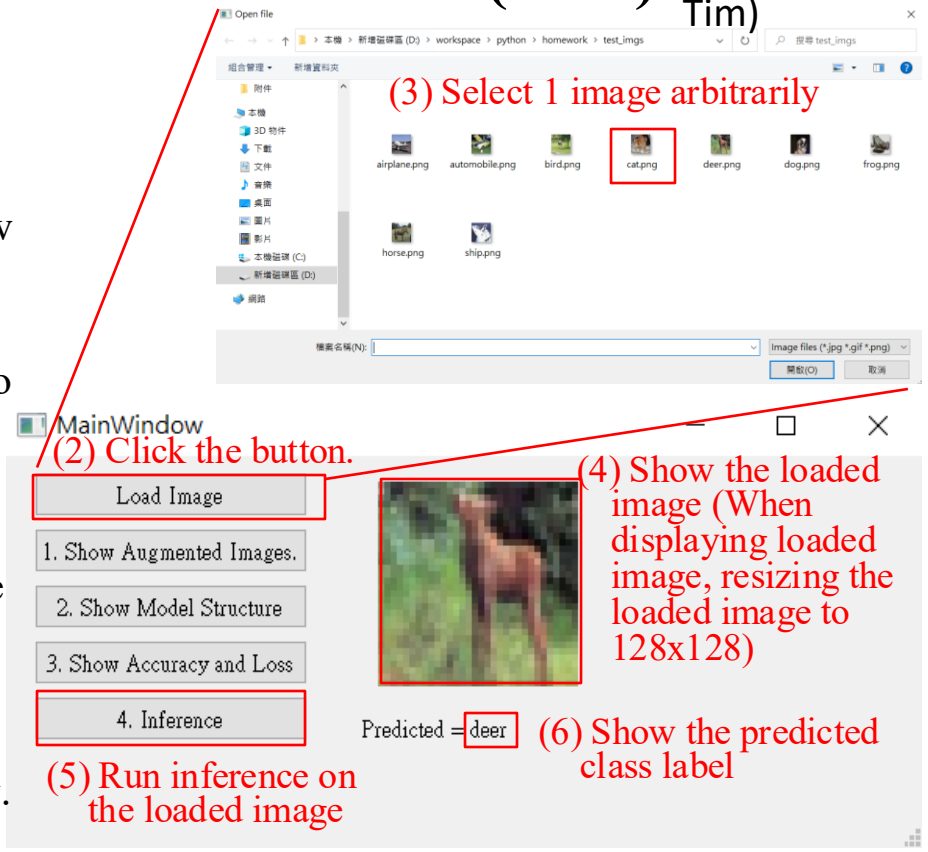
Notice: this is an example, the number of epoch might differ

1.4 Use the Model with Highest Validation Accuracy to Run Inference, Show the Predicted Distribution and Class Label. (15%)

(出題： James, Tim)

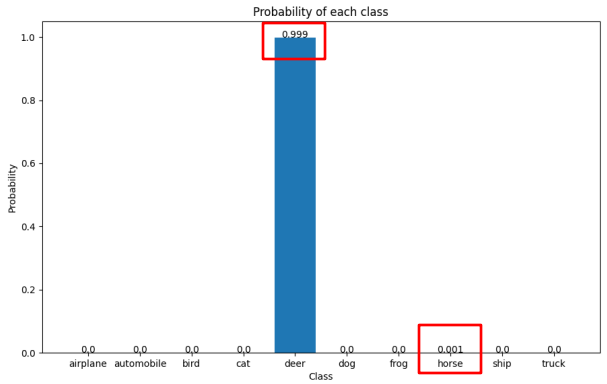
Q1.4

1) **At home:**
   (1) Load the model which trained at home
   (2) Click the button "Load Image" to display a new file selection dialog
   (3) Select 1 image arbitrarily.
   (4) Show the loaded image on the GUI. (In order to make it visually clear on the UI, use QtGui.Qpixmap.scaled to scale the image to 128x128 when displaying it.)
   (5) Click the button "4. Inference" to run inference on the image. (use softmax function) (tutorial)
   (6) Show the predicted class label on the GUI.
   (7) Show the probability distribution of model predictions using a histogram in a new window.
2) **When the demo:** repeat the process



(7) Show the probability distribution of the model predictions in a new window.

(Adding the probability value to each bar of the plot is necessary.)

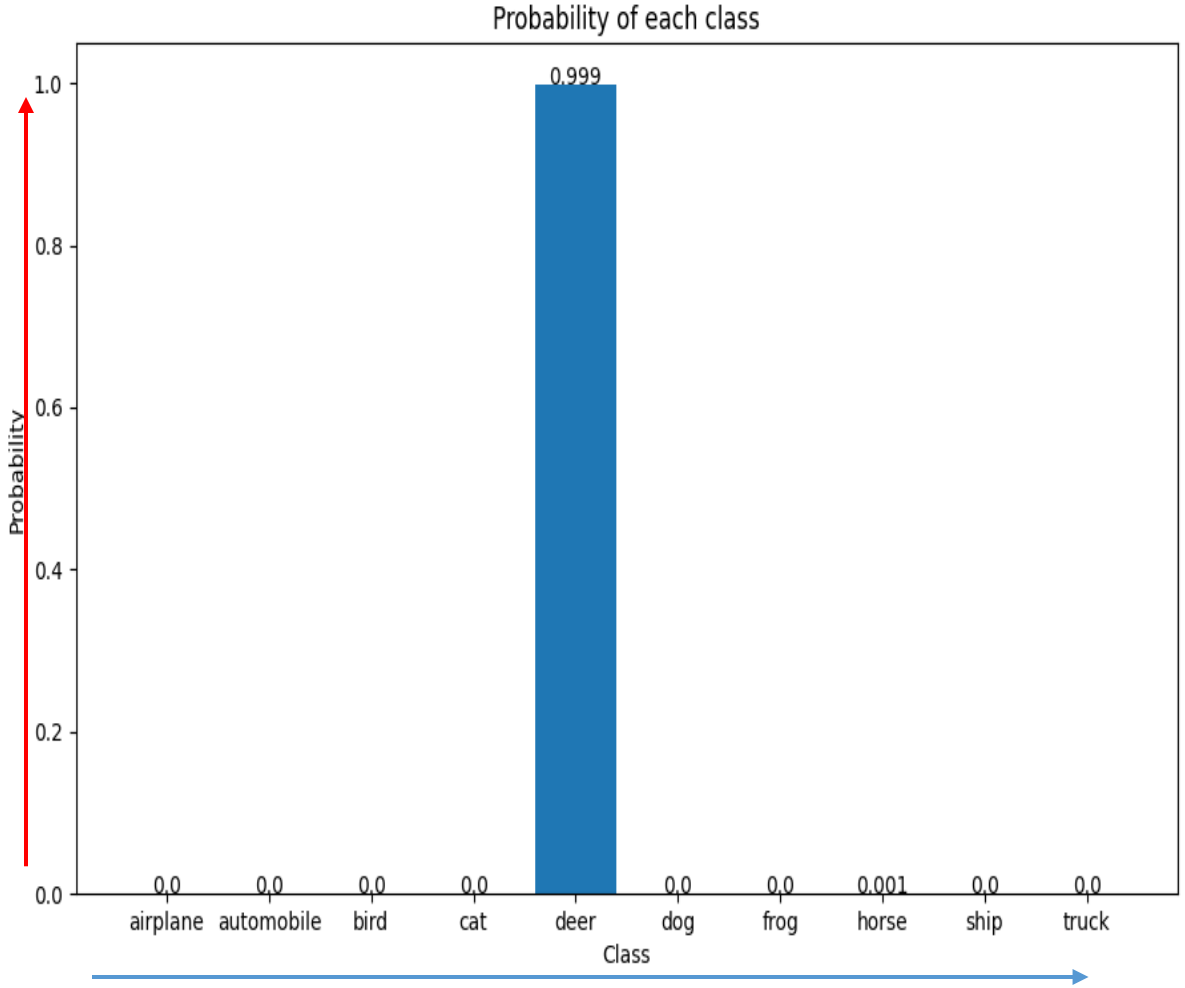


1.4 Use the Model with Highest Validation Accuracy to Run Inference, Show the Predicted Distribution and Class Label. (15%)

(出題： James, Tim)

- The probability distribution of model prediction using a histogram.

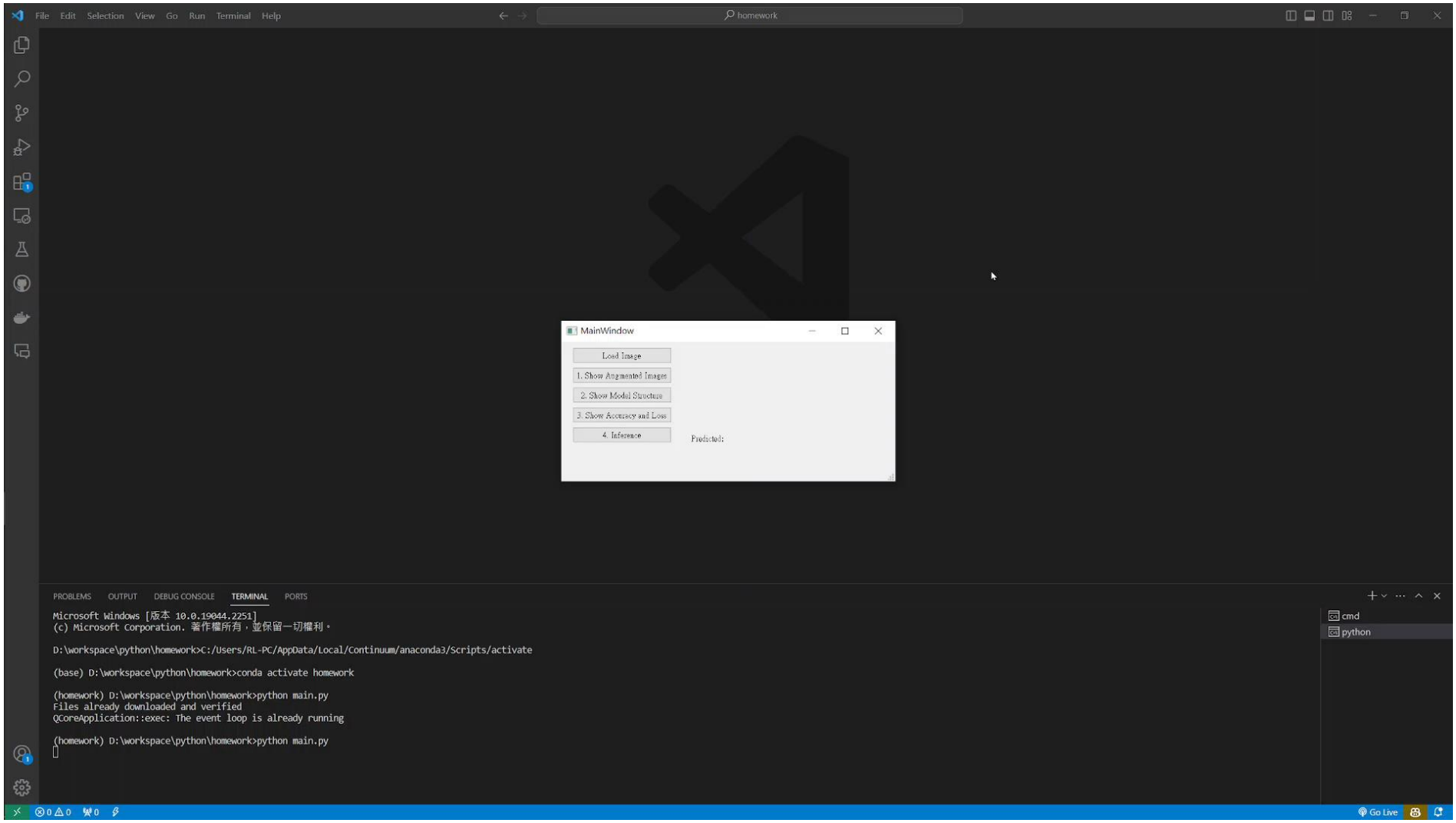
Y-axis: probability



X-axis: class name

1. Training a CIFAR10 Classifier Using VGG19 – Example Video

- This is an example illustrating the objectives from 1.1 ~ 1.4. (出題： James, Tim)



2. Training a MNIST Generator Using DcGAN (50%) (出題：Neil, Alan)

2.1 Load MNIST to show **training images** and **augmented training images**. (10%)

2.2 Load model and show **model structure**. (10%)

2.3 Show **training loss**. (15%)

2.4 Show the **real images and fake images** using Generator. (15%)

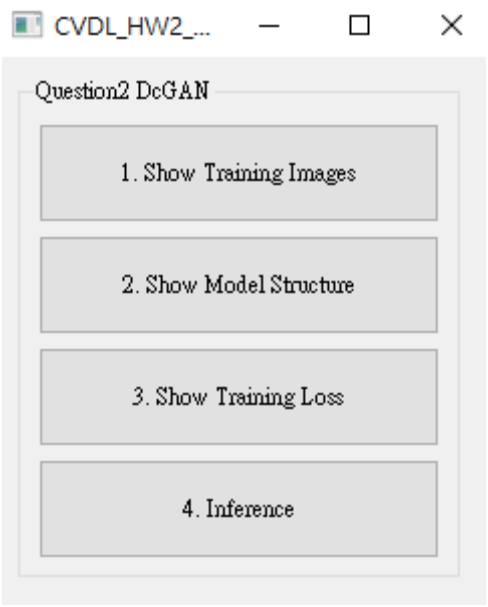


Fig.1 GUI example

2. Training a MNIST Generator Using DcGAN (50%)

(出題：Neil, Alan)

1) Objective

(1) Learn how to train a DcGAN (Deep Convolution Generative Adversarial Network) model to generate handwritten digits images of MNIST.

2) DcGAN

(1) A convolutional neural network that is made of a Generator and a Discriminator.
(2) Generator: Aim to spawn 'fake' images that look like the training images.
(3) Discriminator: Aim to look at an image and output whether or not it is a real training image or a fake image from the generator.

3) MNIST

(1) A collection of 70,000 handwritten digits (0-9), with each image being 28x28 pixels, each pixel value (0-255) represents the grayscale intensity of the corresponding pixel in the image.
- Training dataset: 60,000 images in total.
- Testing dataset: 10,000 images in total.

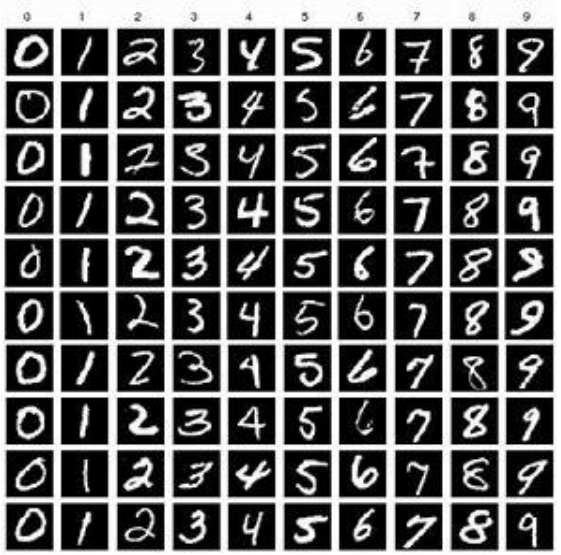


Fig.2 MNIST datasets

R. Reference: DcGAN

2. Training a MNIST Generator Using DcGAN (50%) (出題：Neil, Alan)

4) Requirements

(1) Train DcGAN model using PyTorch.

(2) In the submitted file, you need to include

A. Weight file for DcGAN in .pth format. (File size is approximately 13.6MB for Generator & 10.5MB for Discriminator)

B. Figure of training loss in .jpg or .png format.

C. Code for your GUI program

D. Code for model training.

(3) Please do not include image data in the submitted file.

5) Homework Images

(1) There is one folder in 'Q2_image'.

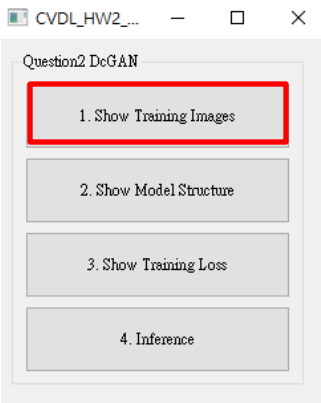
(2) In the subfolder 'Q2_image/mnist,' there should have 70,000 images (18.3 MB) in the folder. When do the training, use those images.

(3) The folder structure shown as below:

```
Q2_image/
└── mnist/
    ├── test_0_60003.png
    ├── test_0_60010.png
    ...
    └── train_9_59992.png
```

2.1 Load MNIST to Show Training Images and Augmented Training Images (10%)

(出題：Neil, Alan)



Q 2.1

1) At home:

(1) Use dest.ImageFolder(root=dataroot, transform=transforms.Compose([…])) at torchvision to load MNIST dataset /Q2_image/mnist/ folder. (tutorial)

Dataset's path

Data preprocess (includes resize, augmentation, …)

(2) Apply at least 1 different type ofdata augmentation (tutorial).

A. transforms.RandomRotation(60)

degree

Notice: this is an example. You can use different data augmentation techniques.

2) When the demo:

(1) Click the button "1. Show Training Images"

(2) Load MNIST dataset /Q2_image/mnist/ folder

(3) Apply data augmentation on training images.

(4) Show 64 original training data and 64 augmented training data in a new window.

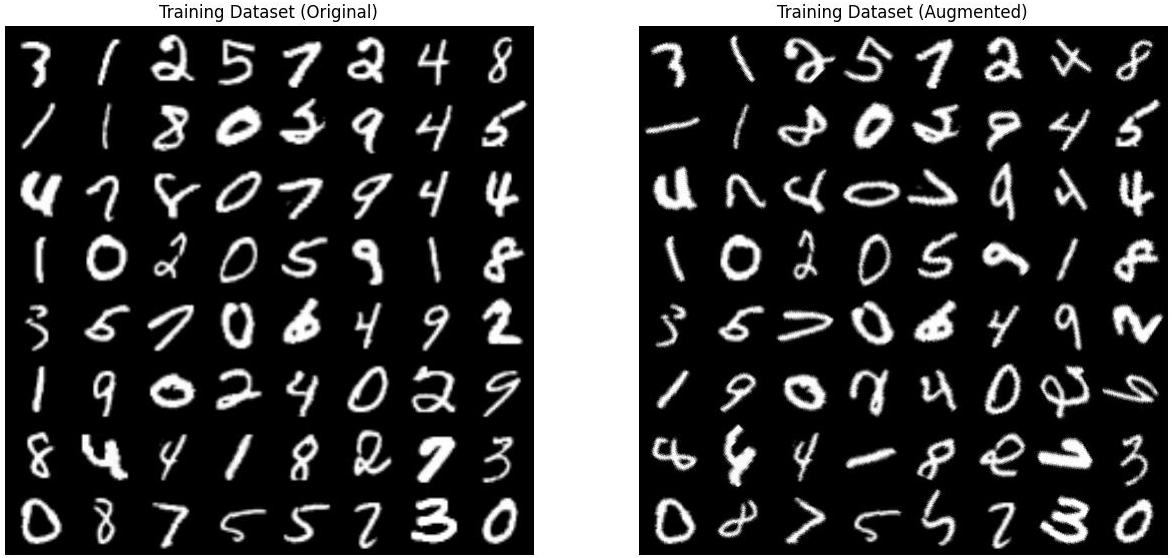


Fig.3 Training images from mnist dataset

Notice: this is an example, the images might be different

2.2 Load Model and Show Model Structure (10%) (1/2)

(出題：Neil, Alan)

Q 2.2 – Generator (tutorial) (5%)

1) At home:

(1) Use netG = Generator(ngpu).to(device) to build the Generator model into gpu or cpu.

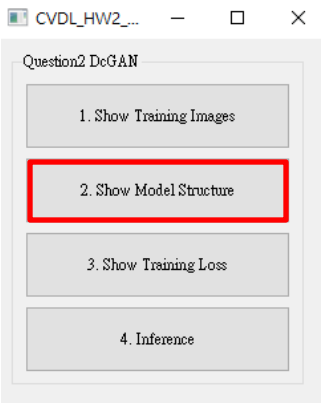
(2) Use netG.apply(weights_init) to apply initial weight into Generator.

(3) Use print(netG) to print the Generator structure. (Fig.4.2)

2) When the demo:

(1) Click the button "2. Show Model Structure"

(2) Run the function to show the 2 structures in the terminal.




```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

i/p channels

o/p channels

Bias in weights

Prevent small values that divide by zero

Computing the moving average of mean and variance

Learnable affine parameters (scale and shift)

Tracks the running mean and variance during training

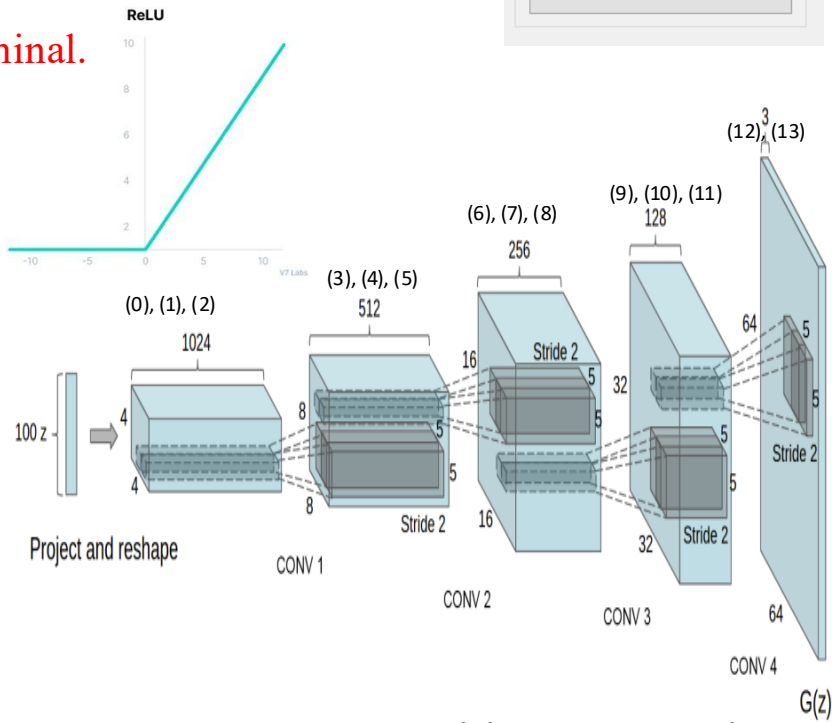


Fig.4.1 DcGAN Generator model structure visualize

Fig.4.2 DcGAN Generator model structure

2.2 Load Model and Show Model Structure (10%) (2/2)

(出題：Neil, Alan)

Q 2.2 – Discriminator (tutorial) (5%)

1) At home:

(1) Use netD = Discriminator(ngpu).to(device) to build the Discriminator model into gpu or cpu.

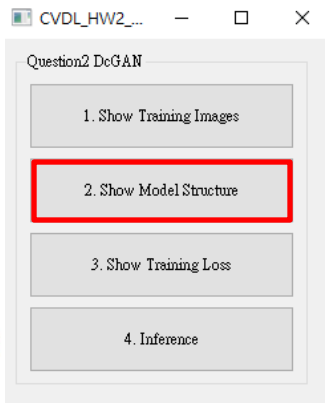
(2) Use netD.apply(weights_init) to apply initial weight into Discriminator.

(3) Use print(netD) to print the Discriminator structure. (Fig.5.2)

2) When the demo:

(1) Click the button "2. Show Model Structure"

(2) Run the function to show the 2 structures in the terminal.




```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

i/p o/p

The slope of the activation function for the negative input values

Modifies the input tensor in-place to save memory

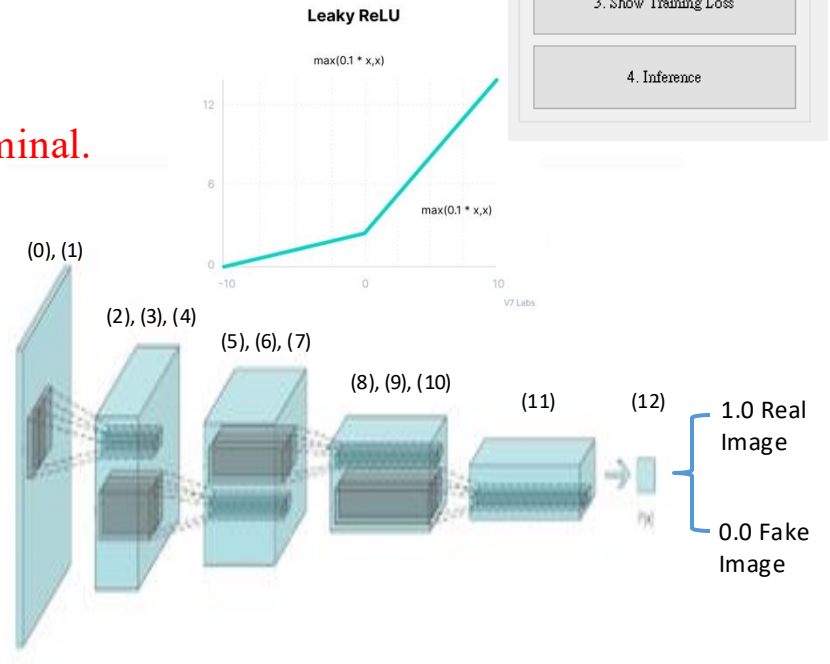


Fig.5.1 DcGAN Discriminator model structure

Fig.5.2 DcGAN Discriminator model structure visualize

2.3 Show Training Loss (15%) (1/2)

(出題：Neil, Alan)

Q 2.3 (reference as DcGAN tutorial)

1) At home:

(1) Load the training dataset as Q 2.1.

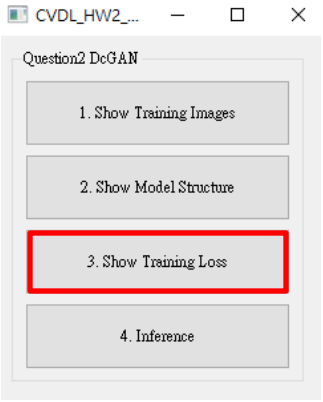
(2) Training DcGAN at least 10 epochs at home (tutorial) and record the training loss in each epoch (tutorial).

(3) Notice: If your loss is too high, you can try
   A. Adjust the learning rate of the optimizer.
   B. Change the data augmentation techniques used.

(4) Save weight file (.pth) with lowest Generator loss. (how to save and load model in pytorch?)

(5) Use plt.plot(G_losses, label="G") & plt.plot(D_losses, label="D") to create a line chart for the Generator and Discriminator loss during training values (tutorial).

(6) Save the figure.



2) When the demo:

(1) Click the button "3. Show Training Loss".

(2) Show the saved figure of Generator and Discriminator loss during training in a new window.

Notice: this is an example, the number of iterations might differ

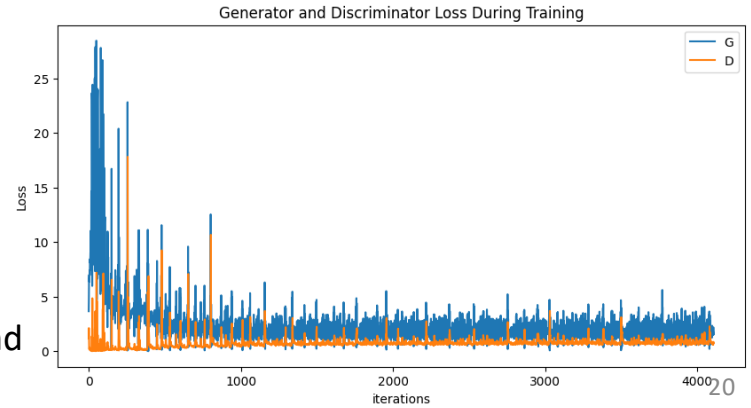


Fig.6 images for the Generator and Discriminator loss during training

2.3 Show Training Loss (15%) (2/2)

(出題：Neil, Alan)

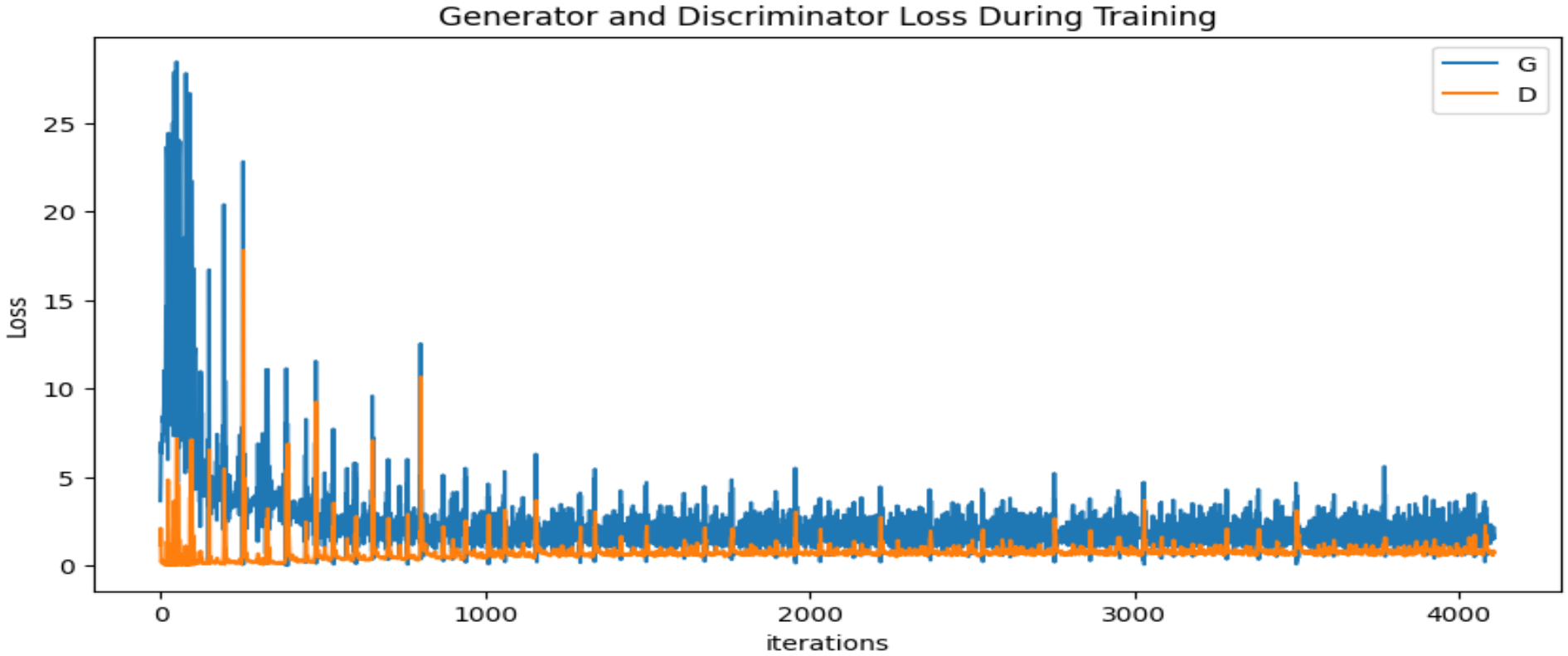
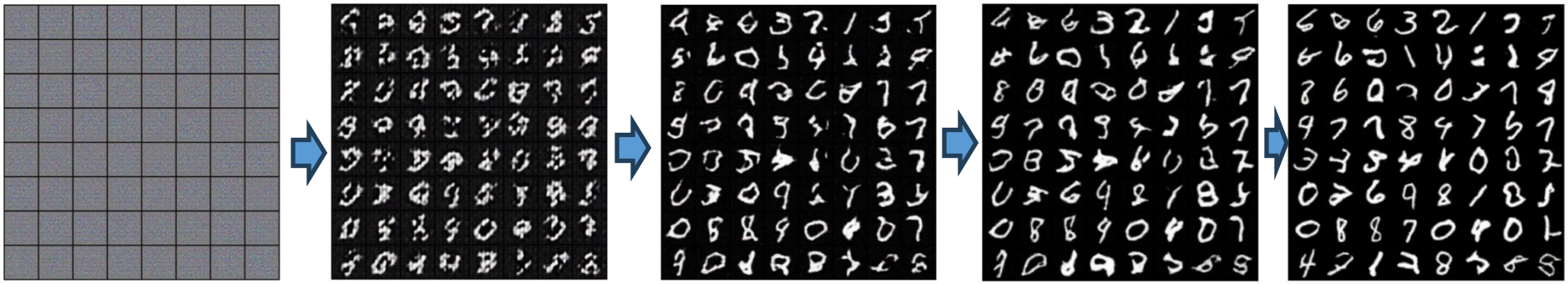


Fig.6 images for the Generator and Discriminator loss during training



2.4 Show the Real Images and Fake Images Using Generator (15%)

(出題：Neil, Alan)

Q 2.4

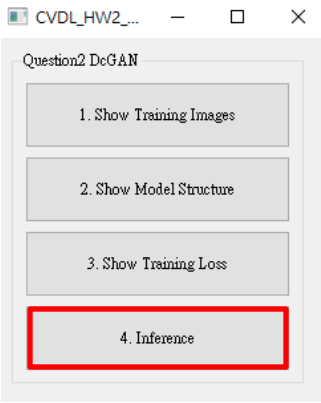
1) At home:

(1) Load the .pth model which trained at home. (how to save and load model in pytorch?)

(2) Click the button "4. Inference" to run inference on the image.

- Remember you need to load dataset like Q 2.1, load model structure like Q 2.2.

(3) Show the loaded image on the GUI. (Fig.6)



2) When the demo: repeat the process

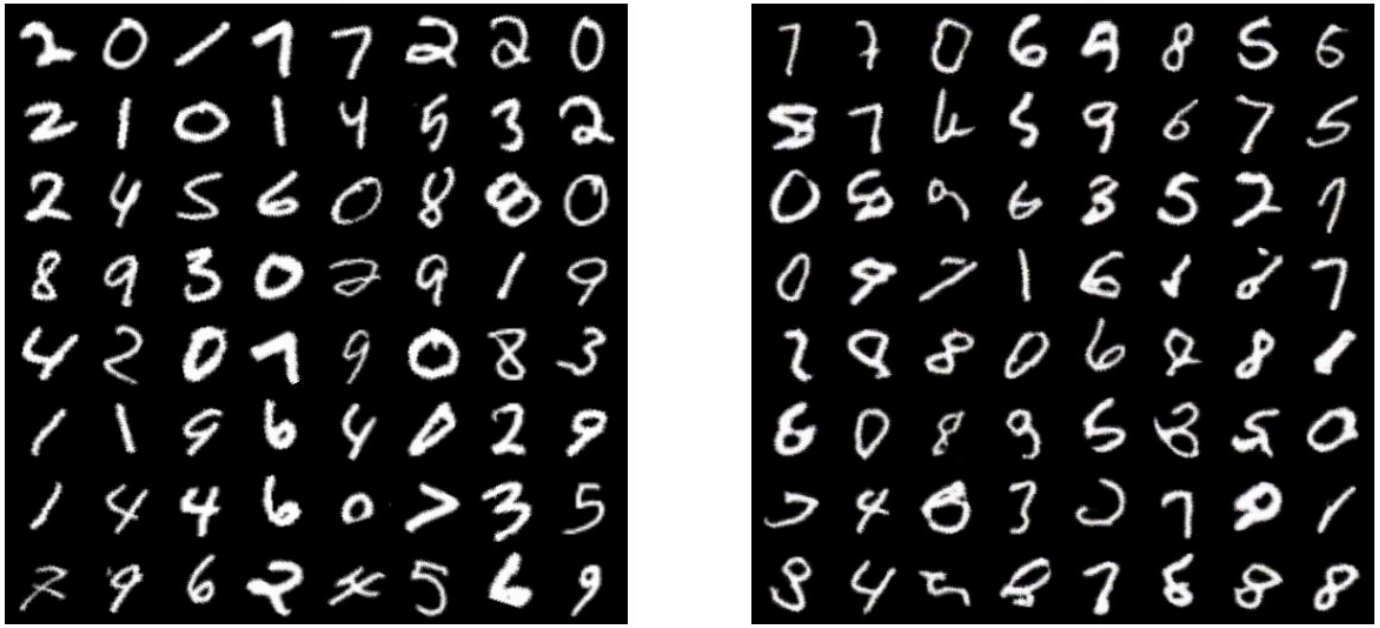


Fig.7 Show Real and Fake images via Generator

2. Training a MNIST Generator Using DcGAN - Example Video

(出題：Neil, Alan)

- This is an example illustrating the objectives from Q 2.1 ~ 2.4.