AI Report

Group 20

Contents

1. Convert csv to image

- The dataset of this task is "The Quick, Draw! Dataset", the csv file corresponding to this task contains multiple lines, one data per line, including drawings and labels two parts of information, where drawings are stroke information when drawing the image, and labels are the corresponding category information. The specific drawings are three-dimensional matrices,

```
[
  [   // First stroke
    [x0, x1, x2, x3, ...],
    [y0, y1, y2, y3, ...],
    [t0, t1, t2, t3, ...]
  ],
  [   // Second stroke
    [x0, x1, x2, x3, ...],
    [y0, y1, y2, y3, ...],
    [t0, t1, t2, t3, ...]
  ],
  ... // Additional strokes
]
```

  where x and y are the pixel coordinates, and t is the time in milliseconds since the first point. x and y are real-valued while t is an integer. The raw drawings can have vastly different bounding boxes and number of points due to the different devices used for display and input.

- Our classification and generation tasks are image-based, so the first step is to convert the drawing arrays to the corresponding image. The conversion function we used is based on simulating the drawing process of the image. (https://github.com/googlecreativelab/quickdraw-dataset/an/19) We generate the original size of the image as 256 and then resize to the desired size according to the task, especially for the classification size of 28 x 28, the size of the

generation is 60 x 60. The resulting image dataset information is as follows:

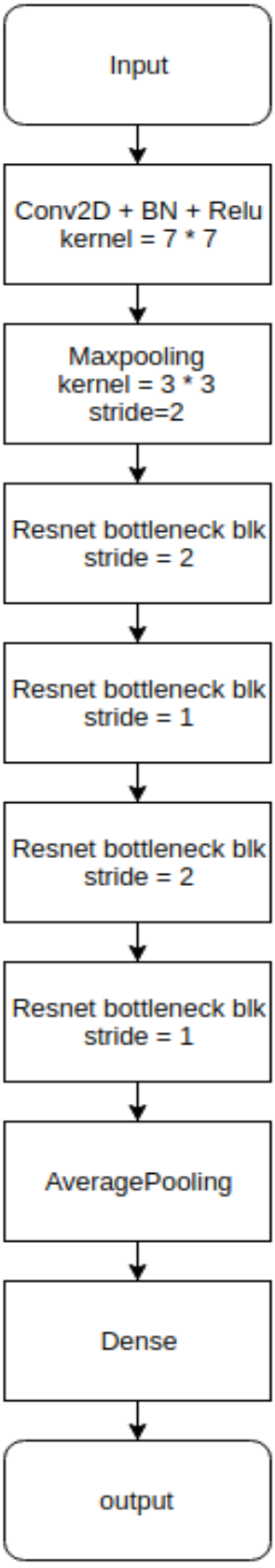
train images shape: (3943968, 28, 28, 1)
valid images shape: (438086, 28, 28, 1)
test images shape: (41, 28, 28, 1)

2. Classifier

2.1 Methodology

- First, the stored stroke in CSV is drawn into a picture, and then use the deep neural network to classify the images.

- Model settings
    - The model is a modified resNet because the task is relatively simple and can be implemented with fewer layers, which can reduce the complexity of the model and reduce training time. The model schematic is as follows, a total of 10 layers [two layers per stat (two layers per head, globalAverage2D not one layer, MLP one)
      conv2D (7*7 kernel) -> bottleneck (stride=2) -> bottleneck (stride=1) -> bottleneck (stride=2) -> bottleneck (stride=1) -> GlobalAverage2D -> MLP


Input
Conv2D + BN + Relu
kernel = 7 * 7
Maxpooling
kernel = 3 * 3
stride=2
Resnet bottleneck blk
stride = 2
Resnet bottleneck blk
stride = 1
Resnet bottleneck blk
stride = 2
Resnet bottleneck blk
stride = 1
AveragePooling
Dense
output

2.2 Parameter Settings

- batch_size = 128
- epoch = 12 (In fact, due to the Early Stop, iteration stops at the 8^{th} epoch.)
- The loss function is cross-entropy.
- Data Augmentation
    - We adopt the ImageDataGenerator generator given by Keras, to enhance including normalizing uint8 to [0, 1], with a maximum random distortion (shear) ratio of 0.2, and a random scale ratio of 0.2, which allows random horizontal flipping.

- Optimizer Setting
    - The optimizer adopts Keras default Adadelta optimizer. (Reference: https://keras.io/zh/optimizers/introduction)
    - Regularizer is not set since this task has a validation dataset that can determine whether it is overfitting by cross-validation, and also have a stop early policy to prevent network overfitting.

- Other Setting
    - Early stop:
      With the EarlyStop hook provided by Keras, when we accumulate five epoch validation set, and Loss does not drop, then stop training can overcome the overfitting.
    - Save the best model:
      With the ModelCheckpoint hook provided by Keras, the overfitting can be overcome by saving the smallest model of the validation set Loss during training as a subsequent model for inference.

2.3 Evaluation

- The evaluation test set has only been normalized to [0,1].

- We first import the best model, and then evaluate it.

2.4 Test Result

- The best performance model achieved by this task is saved in output/best_model.hdf5.

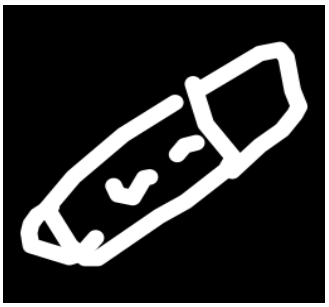
  Valid accuracy: 0.9269800906671293
  Test loss: 0.2298217517573659
  Test accuracy: 0.9756097575513328
  Test_ans loss: 0.2298217517573659
  Test_ans accuracy: 0.9756097575513328

- For 41 samples on the test set, its ground-truth and prediction are shown below, with only one classification error. ('marker' -> 'bandage')

- Test prediction:
  ['key', 'swan', 'light bulb', 'bed', 'roller coaster', 'door', 'key', 'The Great Wall of China', 'paintbrush', 'light bulb', 'bee', 'coffee cup', 'popsicle', 'bandage', 'whale', 'bandage', 'hand', 'toaster', 'toaster', 'banana', 'banana', 'wine bottle', 'banana', 'popsicle', 'spoon', 'spoon', 'bear', 'hand', 'cake', 'snail', 'rain', 'whale', 'giraffe', 'train', 'The Great Wall of China', 'fork', 'cactus', 'bandage', 'raccoon', 'wine bottle', 'raccoon']

- Test label:
  ['key', 'swan', 'light bulb', 'bed', 'roller coaster', 'door', 'key', 'The Great Wall of China', 'paintbrush', 'light bulb', 'bee', 'coffee cup', 'popsicle', 'bandage', 'whale', 'bandage', 'hand', 'toaster', 'toaster', 'banana', 'banana', 'wine bottle', 'banana', 'popsicle', 'spoon', 'spoon', 'bear', 'hand', 'cake', 'snail', 'rain', 'whale', 'giraffe', 'train', 'The Great Wall of China', 'fork', 'cactus', 'marker', 'raccoon', 'wine bottle', 'raccoon']

- The image of the misclassification error in the test set is visualized below, and the shape is close to the bandage after 45 degree of rotation.

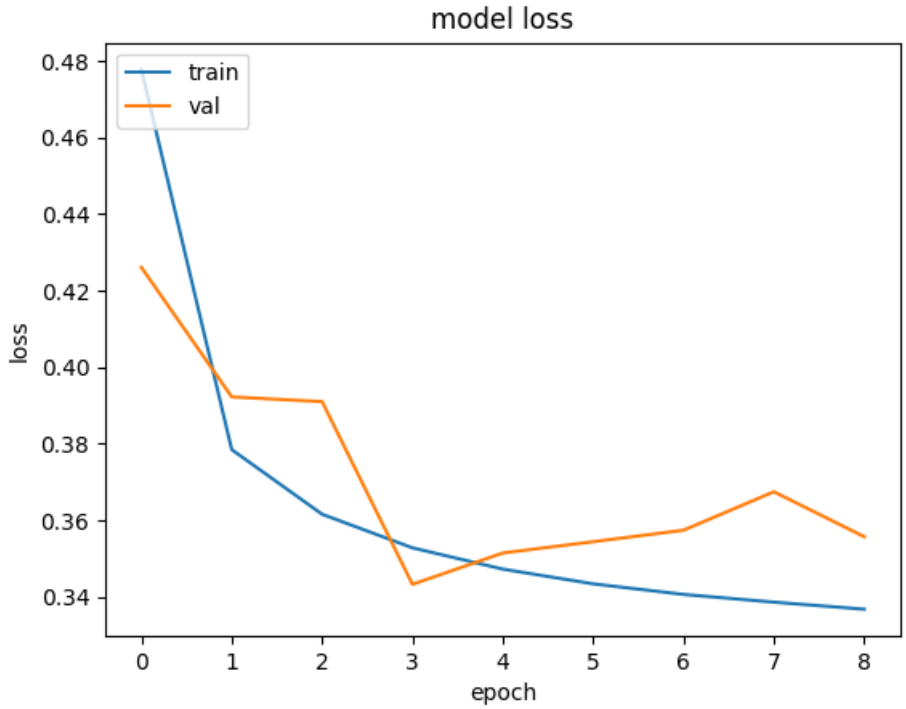


2.5 Demo Result

- By pointing the test CSV path of evaluate.py to demo.csv, we can get the following results that there are no classification errors on the demo set.
- Test prediction: ['bee', 'cactus', 'banana', 'fork', 'wine bottle']
- Test label: ['bee', 'cactus', 'banana', 'fork', 'wine bottle']

2.6 Discussion

- The train_loss and val_loss of this classification task are shown in the figure below, and we can see that the model val_loss to a minimum at epoch = 3. After the stop early hooks, this classification task achieves the stop early after five further pieces of training and saves the models of the best performing while epoch = 3.



2.7 Conclusion

- We got a pretty precise result as we test our model, with the ratio of 97% correctness.

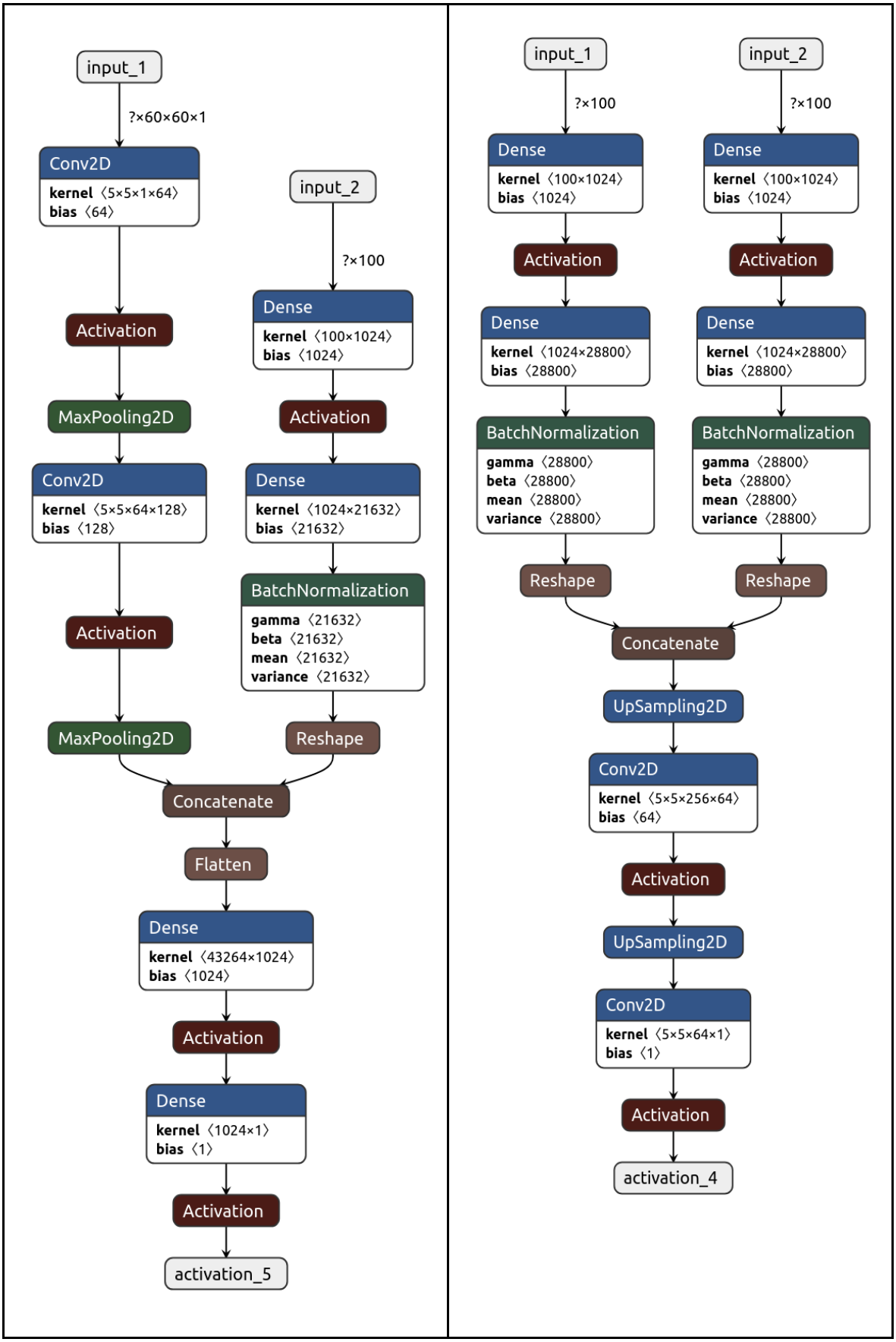
3. Generator

3.1 Methodology

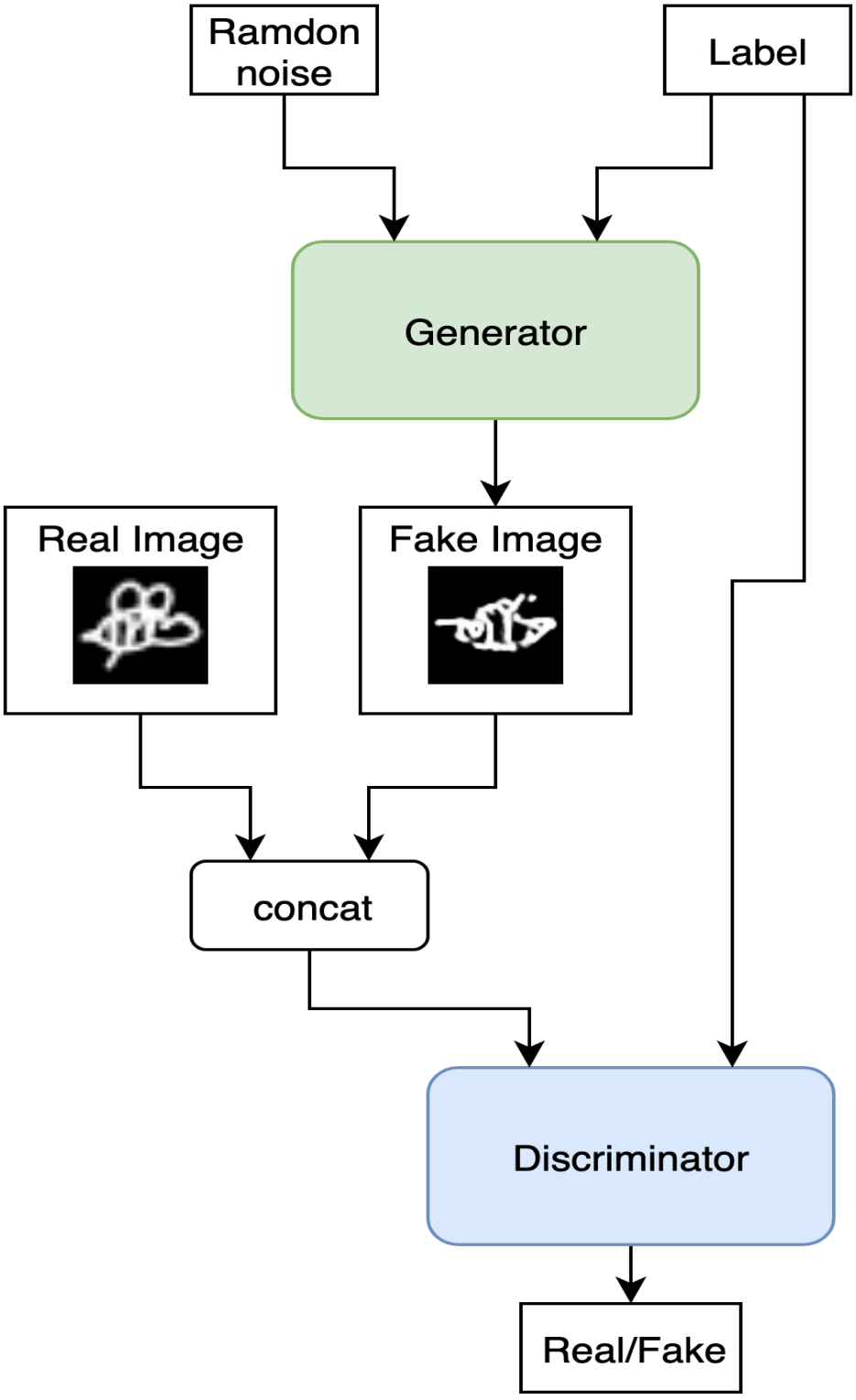
- Since thus task of generating is closely related to the label value, we adopt the conditional GAN.
  (Reference: https://zhuanlan.zhihu.com/p/23648795, https://medium.com/@sam.maddrellmander/conditional-dcgan-in-tensorflow-336f8b03b7b6)

- GAN is divided into generator and discriminator, and the network structure of both in this task is shown below:
    - A different layer from regular DNN, which requires labels 0 to 29, embedding to a specific dimension of the one-hot form. For simpler calculation and scalability, we select a 100-dimensional one-hot embedding vector as the input.

Discriminator	**Generator**
input_1: image, resolution=60 * 60 input_2: label, embedding to 100-d output (activation_5): The input image is real or fake.	input_1: noise, embedding to 100-d input_2: label, embedding to 100-d output (activation_4): generated image, resolution=60*60


input_1
?×60×60×1
Conv2D
kernel 〈5×5×1×64〉
bias 〈64〉
Activation
MaxPooling2D
Conv2D
kernel 〈5×5×64×128〉
bias 〈128〉
Activation
MaxPooling2D
input_2
?×100
Dense
kernel 〈100×1024〉
bias 〈1024〉
Activation
Dense
kernel 〈1024×21632〉
bias 〈21632〉
BatchNormalization
gamma 〈21632〉
beta 〈21632〉
mean 〈21632〉
variance 〈21632〉
Reshape
Concatenate
Flatten
Dense
kernel 〈43264×1024〉
bias 〈1024〉
Activation
Dense
kernel 〈1024×1〉
bias 〈1〉
Activation
activation_5
input_1
?×100
Dense
kernel 〈100×1024〉
bias 〈1024〉
Activation
Dense
kernel 〈1024×28800〉
bias 〈28800〉
BatchNormalization
gamma 〈28800〉
beta 〈28800〉
mean 〈28800〉
variance 〈28800〉
Reshape
input_2
?×100
Dense
kernel 〈100×1024〉
bias 〈1024〉
Activation
Dense
kernel 〈1024×28800〉
bias 〈28800〉
BatchNormalization
gamma 〈28800〉
beta 〈28800〉
mean 〈28800〉
variance 〈28800〉
Reshape
Concatenate
UpSampling2D
Conv2D
kernel 〈5×5×256×64〉
bias 〈64〉
Activation
UpSampling2D
Conv2D
kernel 〈5×5×64×1〉
bias 〈1〉
Activation
activation_4

Furthermore, the overall design is shown as follow:



3.2 Parameter Settings

- batch_size = 512
  (Bigger batch_size is convenient for batch normalization and the image will become more realistic.)

- epoch = 1
  (After a few experiments, there are a lot of images in theis task, so the more realistic image could be generated when epoch = 1.)

- Loss Function: Generator and discriminator alternate training
  - First when we train the discriminator, we fix the generator parameter and make the loss function binary cross-entropy.
  - Then when we train the generator, we fix the parameter of discriminator, and the loss function is also binary cross-entropy.

- Data Augmentation
  - We do not augment the data as the number of images in each category is quite a lot, which has already satisfied the training requirements. (We only normalize the image to [-1,0] to meet the requirement within the range for the generator to activate the function tanh.)

- Optimizer
  - We adopt adam in this task, and the learning rate is 0.002, beta_1 = 0.5， beta_2 = 0.999.
  - After the experiment, we proved that the result for adam to be as the optimizer is up to snuff.

3.3 Evaluation

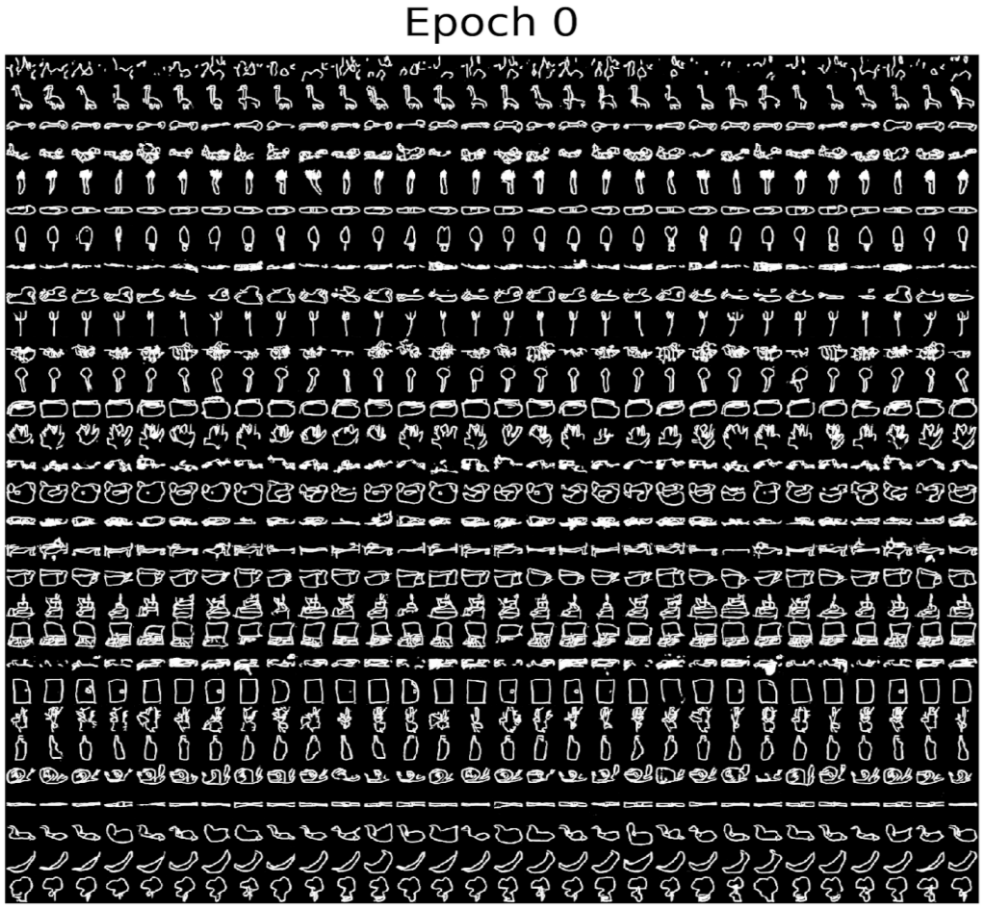
- During the training process, we save a collection of generated pictures every 20 steps, so that we can filter out the number of iteration steps that produce better results for early stop.
- Iteration images are saved in generation/images/iterations.

3.4 Test results

- The code is in generation/evaluate.py.
- Evaluation only need the model of generator, according to the assigned label, generate the images from the noise.

3.5 Demo results:

- The last iteration to generate the images is shown below:



- However, during the demo day in class, we only got 0/100, 0/100, 44/100 in each category, with a vast difference between our results.

3.6 Discussion

- We adopt the conditional deep convolution GAN method to generate the required picture in this task to enhance the quality, and also make us able to achieve the requirements of condition control.

- During training, we optimize the discriminator and generator alternately, so the overall Loss does not iterate with the number of training. Instead, it is in equilibrium, which indicates that the discriminator and generator are balanced.

- Training GAN has some skills:
    - At first, we set too many layers in discriminator that the generator is not able to learn, which is a form of mode collapse that the images generated being single, and each time very different.

    - As the number of layers of the discriminator decreases, the generator can influence and supplement each other with the discriminator, making the picture with higher quality.

    - For the conditional GAN, we first embed into a high-dimension vector and then use several layers of a full-connection layer to extract the features. Finally, we combine with the features of the images to generate or discriminate.
      The advantage of this is that through several layers of pre-connecting layers, We can be better handle the features and extract the more appropriate ones. Otherwise, it is easy to produce an image with poor quality.

- This task uses the conditional deep convolution GAN method to generate the required images which have good quality and can achieve the requirements of condition control.

4. Conclusion

- A better idea of how to process data, including loading CSV, loading .png, and the operations about python as well as numpy.

- Learn how to use Keras (Tensorflow) to train classifier and generator, and to be familiar with the concepts of machine learning such as overfitting or loss function.

- There is still a lot to be improved. We only understand some basic concepts in machine learning. We could perform better if we can adjust how we set the parameter, use better algorism, or have a more accurate dataset.

5. Work Assignment

Team Member	閻紹寧	羅睿君
Contribution	1. Write the whole project. 2. Design the presentation slides. 3. Produce the report.	1. Double check if the model runs well. 2. Present on stage.