Introduction to Artificial Intelligence Final Project Report

Team 18

106061146 陳兆廷 106062173 吳東弦 106062171 姚柏佑

Purpose:

Train a doodle classifier and a doodle generator for 30 objects.

Classifier:

A. Methodology:

Model selection:

Since I have learned some basic concept on CNN before, I decided to train the doodles on a basic classifier based on ResNet34. The model came from Deep Residual Learning for Image Recognition, a paper on CVPR 2016. The paper introduced residual learning and added them between layers of deep convolutional models. Residual layers make deep models converge more easily when performing SGD and backpropagation. A convolutional neural network is trained, for reference.

Data preprocessing:

The training data and target test data we received from TA is stroke-representation of graphs. Therefore, to train the data, I need to draw the graphs by myself. I wrote a strokes-to-graph script to transform stroke-represented graphs into 2D graphs.

Training:

I selected 10000 graphs from each category, 80% of them as training dataset and 20% of them as testing dataset.

Loss calculation:

Cross-entropy loss.

$$H = \sum_{c=1}^{C}\sum_{i=1}^{n} -y_{c,i} log_2(p_{c,i})$$

B. Parameter setting:

Model	Epoch	Data size	Learning rate	Batch size
ResNet34	30	10k each	1e-1~1e-9	2048

C. Optimizer:

I choose SGD (Stochastic Gradient Descent) as my optimizer, as the paper suggested, with 5e-4 weight-decay, 0.9 momentum and adaptive learning rate. As the learning rate decays, SGD would find the saddle point and converge more easily.

D. Evaluation:

The evaluation of model performance is the scoring for test dataset. ResNet34 achieves 96% accuracy on test dataset and CNN achieves 95%. Also, I plotted the 200 test-data given by TA and labeled them. The result was acceptable to me.

E. Performance:

In every epoch, the accuracies increased, and the losses decreased (Fig 1.). Finally, the test accuracy went to 96% (Fig 2.).

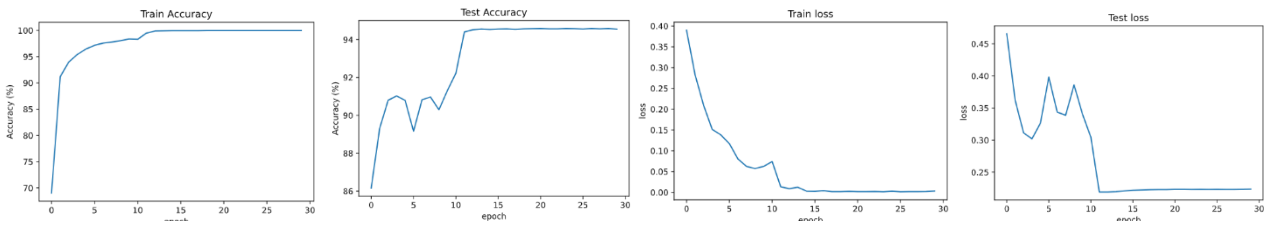

Fig 1. Train Accuracies, Test Accuracies, Train Loss and Test Loss every epoch. (left to right)

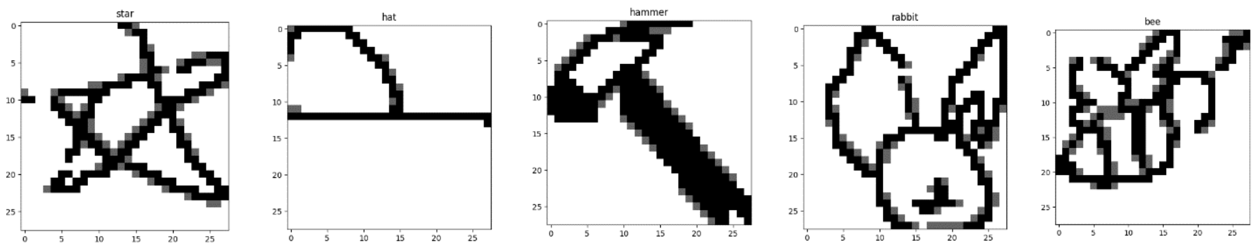

Fig 2. Classified result: star, hat, hammer, rabbit and bee. (left to right)

Generator:

A. Methodology:

Model Selection:

I did some research on GANs. Finally, I picked Deep Convolutional GAN(DCGAN) as my model and Conditional GAN(CGAN) as my model. DCGAN provides a basic drawing generator that generates drawing from random noise, and CGAN provides a generator that generates drawings according to labels. I combined two of them as Conditional Deep Convolutional GAN(CDCGAN). Here are my models (DCGAN, CGAN, CDCGAN).

Data Preprocessing:

I prepared 2 sets of 2D graph training data with different thickness of lines.

Training:

In every epoch, I generated batches of images according to the true labels using generator and fool the discriminator with generated images (calculate generator loss). Then, I calculate the real-loss and fake-loss of discriminator using true images with true labels and fake images with fake labels.

Loss calculation:

Binary cross entropy loss.

$$\mathrm{BCELoss}(\mathrm{O},\mathrm{T}) = -\frac{1}{n}\sum_i \big((T[i] * \log(O[i])) + (1 - T[i]) * \log(1 - O[i])\big)$$

B. Parameter setting:

Model	Epoch	Data size	Learning rate	Batch size
DCGAN	40 each	100k+ each	2e-5~2e-7	1024
CDCGAN	20	30k each	2e-5~2e-7	1024

C. Optimizer:

I choose Adam optimizer as my optimizer for generator and discriminator. Adam is a combination of adaptive learning rate and momentum SGD optimizer. By changing the momentum, the optimizer is smoother when converging.

D. Evaluation:

The way I evaluate the model is by monitoring the loss of discriminator and generator, also the output of the generated result after training. Furthermore, I generated some images every epoch for me to evaluate the model by my eyes.

E. Performance:

Since the complexity of each object varies, I found out that some simple objects (star/clock) are easier to train, while other complex objects (lion/bee) must be carefully monitored and tuned. Sadly, I did not have much time, therefore complex objects such as bee, lion or rabbit could be disasters. Furthermore, training data that has thinner strokes could generate drawings more clearly, but it took more time using thinner strokes (Fig 4.). Lastly, CDCGAN and DCGAN both generated similar results (Fig 5.).

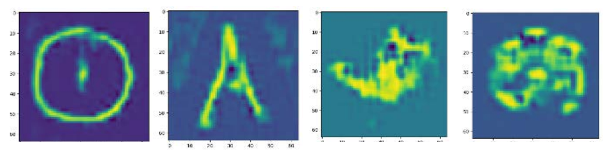

Fig 3. Simple v.s. Complex objects (clock/tower/bee/lion)

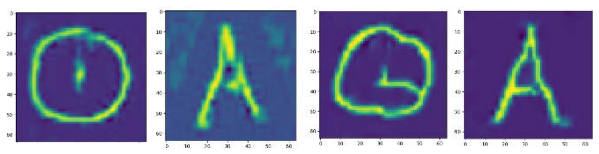

Fig 5. DCGAN v.s. CDCGAN (clock/tower)

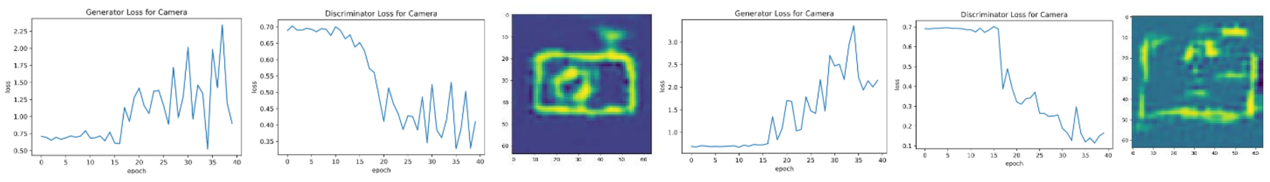

Fig 4. Thick strokes v.s. Thin strokes: Generator/Discriminator loss, result for camera.

Demo Result:

Discussion:

RNN GANs: It is quite interesting since I did not know anything about RNN GAN. I think that the space complexity of training an RNN should be lesser than a CNN because there are multiple empty pixels. However, the difficulty should be way harder since the input contains the orders of the strokes. Stroke-represented graphs contains more information comparing to 2D graphs.

Style-transferring GANs: When I did my research, I found out that there are multiple style-transferring GANs. Although I choose DCGAN and GANs that can be trained on MINST dataset, I wonder if others can also be used on this project. The difference is the input image and how the noise are generated.

Conclusion:

Classifier:

ResNet did a wonderful job. The accuracy achieves 96%. The errors could be some drawings that could not be identified. Increase training data size and train for longer time can improve the performance and accuracy.

Generator:

Generator did well on some simple drawings such as clock, airplane, and camera. However, difficult drawings such as bee, lion, and bird did not perform well. Different object requires different parameters setting (epoch/learning rate/decay …).