Conditional Variational Autoencoder Implementation in Python

Tham Yik Foong (20200786)

December 2019

1 Introduction

The goal of this project is to create a Conditional Variational Autoencoder (CVAE) with Python to generate handwritten characters and inspect the performance of the model, computational time, and usability of different Python modules. This report will introduce the background of Conditional Variational Autoencoder, methodology of implementing the model, and choices of coding. Followed by that, the remaining section will demonstrate testing of the project, result yield at the end of the project, future implementation, and improvement.

2 Background of CVAE

CVAE (Figure 1) is a generative model composed of neural network. It has the ability to form an empirical distribution by learning from the distribution of a given training data set, it then generate an output by sampling from the empirical distribution. CVAE is an extension of variational autoencoder (VAE), what distinct them is CVAE allow additional condition to be passed in into the model to specify an expected outcome, unlike VAE, which only can generate outcome by manually manipulating its latent variables.

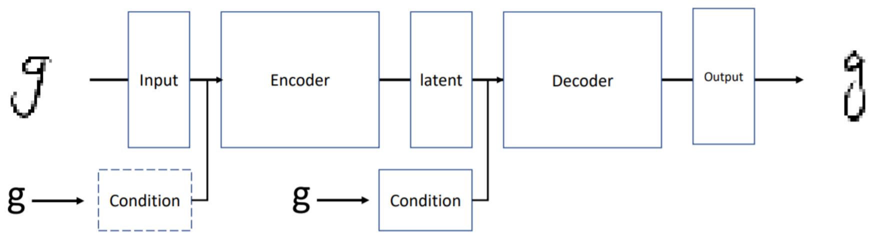

Figure 1: Structure of CVAE model

3 NIST Dataset

Before starting the project, a data set need to be determined first. Because the goal of this project is to create a CVAE model to generate hand written characters. Hence, I have decided to use NIST special database. It is a data set consist of 810,000 hand written character images, each of them are 128 * 128 pixel wide and tall. NIST data set contain 62 different classes of hand written character labelled by hexadecimal (0 to 9, a to z, A to Z). A complete user guide are provided which clearly explain the usage and details of the data set.

4 Implementation of project

4.1 Data prepossessing

before building the model itself, I created a Python script *data_processing.py* to handle data prepossessing. the objectives of performing data prepossessing include, firstly, to scale down the size of data images. re-scaling images help to scale down the model, which makes training of the model faster and easier. With the help of OpenCV library, script is able to read, resize, and save resized images into a target path*. Furthermore, because NIST data set are labeled by hexadecimal. Therefore, I have to convert labels to ASCII and turning them into a one-hot encoded labels storing in a dictionary. Lastly, the script will save both images and labels into a separate Numpy arrays, in order to be fed into the model.

4.2 Building CVAE model

To create a CVAE model, I created a separate Python class *model.py* to construct the model. Although the model can be implemented directly into a main script, and one can directly run and work on one script. But storing model in separate modules provide a "Plug and play" interface for other projects that also make use of a CVAE model. in addition, it also makes code easier to visualize in my own preferences. By utilizing Tensorflow module, I am able to create a CVAE model by defining initializer, optimizer, structure of model, and a loss function. The reason of choosing Tensorflow as model framework is it provides better graph and model visualization with TensorBoard, which make troubleshooting of the model easier.

4.3 Training the model

After the model and data is ready, I created a main script *main.py* to put everything together and start training and testing the CVAE model. First, initiate hyper-parameters of models. Moving forward, load the Numpy arrays created from *data_processing.py* as data. Because all data are currently ordered by class, it is necessary to shuffle the data, else it will decrease the performance of model or leads to slow convergence. Data can be shuffle with the help of Numpy function, a seed can be provided to ensure persistence permutation of data in each shuffle, but is not required for this project. Perform cross-validation by splitting data and label into training and testing data set. After that, initiate and use the CVAE model constructed in *model.py.* Start training the model by running a Tensorflow session, iterate the training process until it runs all required epochs. In the end, after the training is completed, plot the generated result with Matplotlib and save in a directory specified. Moreover, utilize Scipy module to draw the normal distribution of latent distribution and plot the latent distribution with Matplotlib.

5 Test and Performance

Hyper-parameters are first initialized using standard configuration and are determined after a certain trail and error. The model is not working as expected at the beginning, the model failed to train due to loss function returning Nan value. This is due to output value fed into a log operation in loss function are returning infinity as value, even the output value is clipped by a certain minimum value. After some troubleshooting, increasing minimum value of clipped output help to resolve the issue. Another issue I encounter is loss get incredibly big due to mismatch of input and output. This is caused by a mistake which I forgot to normalize the input before feeding into the model. Other then that, there are a few minor issues fixed by quick debugging.

6 Result

the image of the handwritten character (Figure 2) generated by the model is fuzzy and blur, this is due to how the loss function works in CVAE or VAE model, but the outline of the character is still visible. On the other hand, the model converges quickly in the first few epochs, which indicate that the hyper-parameters such as learning rate are set up correctly. Moreover, the model is easy to interpret as latent distribution can be plotted out for inspection. Last but not least, NIST dataset is not very compatible with CVAE model due to how loss are calculated.

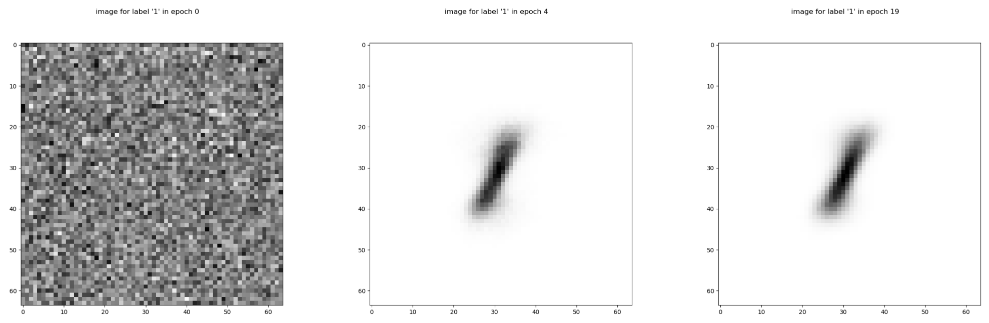

Figure 2: Generated Output

7 Future Implementation and Improvement

There are a few improvements that can be done for this project. Firstly, the model can be further improved by applying more machine learning techniques and trying out different parameters. Also, the latest version of Tensorflow 2.0 enables Eager Execution by default. Hence, it is considerable to enable it instead of running with a Tensorflow Session. For future implementation, this project can be further enhanced by enabling it to generate handwritten sentences or paragraphs with a given document. The model can be reused to generate handwritten character and joining them together to form a sentence, but the project needs to use other data set as NIST data set does not provide punctuation.

8 Conclusion

In this project, A CVAE model is created to generate handwritten character and its result is being well studied. I have learned to utilize some Python modules, such as Scipy, OpenCV, Metplotlib, Numpy, Tensorflow, and et cetera to create a scientific related Python project. At the same time, I am more familiar with the syntax of Python. both aspects are definitely beneficial for future Python program development.