CNN, GAN and Transfer Learning on Digits Classification

Dec. 2020, Machine Learning

Chenkuan Liu

1 Introduction

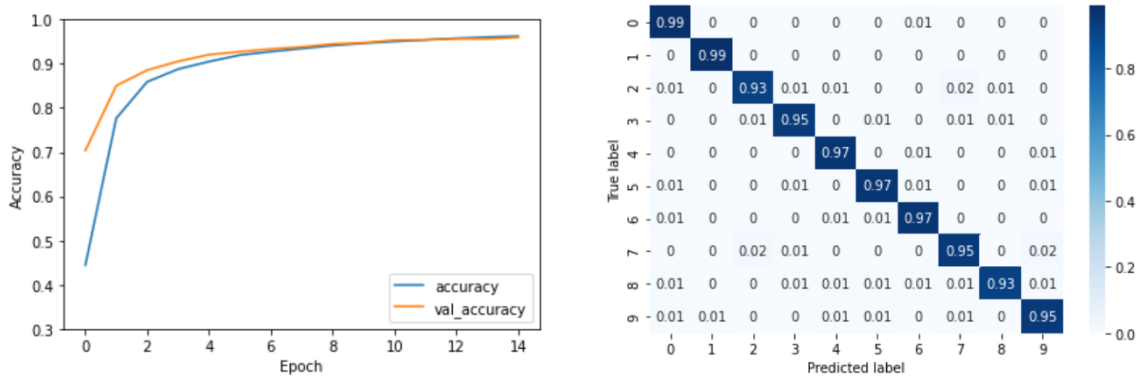
In this project we are using convolutional neural network and generative adversarial network to perform classifications on MNIST hand-written digits dataset. The convolutional neural network is a well-established neural network that has achieved decent performance in image classification in general. At here, we will use it as a baseline to classify MNIST digits. Generative adversarial network, on the other hand, is used to generate images that look like real. We will use the existing models of GAN and apply modifications on them to train and generate new "hand-written digits" images.

The close relation between digits classification and digits generation and their same usage of the dataset also allow the possibility to transfer GAN to the task of image classification. We will use techniques of transfer learning and fine-tuning to modify our trained GAN model to perform image classification, and compare the accuracy results and misclassification cases between our baseline CNN model and the transferred GAN model. The package we use here is `keras` from `tensorflow`.

2 Baseline: CNN

Since our task is to compare these two models rather than absolute accuracy, we do not use the entire dataset. Of the 60000 total training images in MNIST, I randomly selected 10000 of them as the training data. The random seed is set to 42 throughout the project for both models. In order to see how well the performance might be with only 10000 training images, I use the entire 10000 testing images in MNIST as the testing data.

The CNN model here is simple. It only has 2 convolutional layers, both with Relu activation function and followed by max pooling. After that, there are 3 fully connected layers, in which the first one is a flattening layer, the second one is a dense layer with 128 units, and the last one is a dense layer with 10 units as outputs and softmax as activation function. The pixels of training images are subtracted and then divided by 127.5 so that each pixel is within the range of $[-1, 1]$. Both traning and testing labels are transformed through one-hot encoding. The optimizer is stochastic gradient descent, loss function is categorical cross-entropy, and the metric is accuracy. The batch size is 100 and epoch number is 15. We perform test validation after each epoch. After fitting, the accuracy curves and confusion matrix[1] are as follows:



(a) accuracy over epoch

(b) confusion matrix heatmap

Figure 1: CNN Performance; around 0.96 accuracy

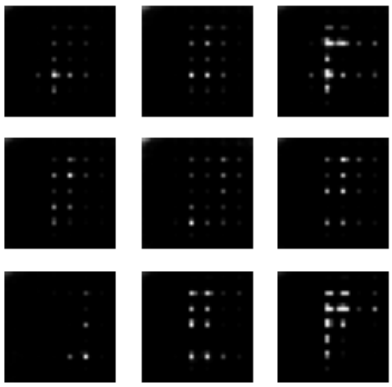
Both training accuracy and validation accuracy reach around 0.96 after 15 epochs. After that, we compute the confusion matrix to examine the classification performance, which is also demonstrated above.

3 GAN

Both the generator and discriminator of GAN here use deep convolutional neural network. Their structures and training process are based on the template provided in Tensorflow's

[1]The confusion matrix heatmap here, as well as those in Figure 4,5,6 later, is only for the 10000 testing digits.

official website. In particular, in order to keep the same standard as the CNN above and accelerate the training progress, I used the exact same 10000 images for training and modified the output number so that 9 example images are generated instead of 16 after each epoch. The number of total epoch is 50. The evolution of example digits and the generator and discriminator's training losses change over time are demonstrated below. We use checkpoints to store the GAN model so that it can be used later for the classification task.



(a) 5th epoch

(b) 20th epoch

(c) 40th epoch

(d) 50th epoch

Figure 2: Generated digits evolution

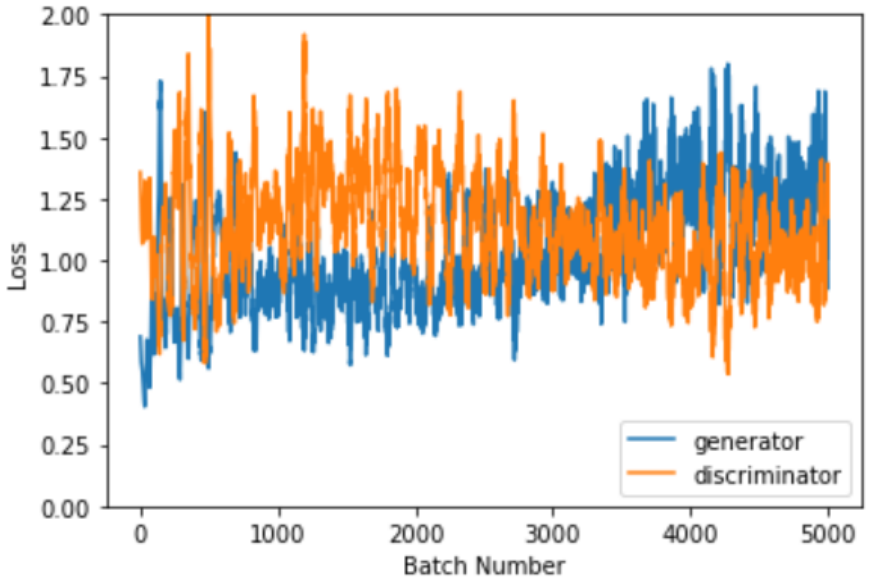


Figure 3: Training loss curves

From Figure 2, we see that, at the 5th epoch, the images are still random noises. At the 20th epoch, they begin to form shape and start to look like hand-written figures. As epoch number continues increasing, the digits become more and more well-constructed. Figure 3 shows that when batch number is below 3000 (namely before the 30th epoch since each epoch has 100 batches), the discriminator's losses are generally above those of the generator. After that, the discriminator's losses get smaller while the generator's losses increase. Around 5000 batch number, their losses become closer to each other.[2]

4 Transfer Learning and Fine-Tuning on Discriminator

At here, we will modify the discriminator in our GAN model to a 10-way classifier. We first remove the last dense layer, which only contains one unit for fake/real image detection, and then keep all previous layers frozen. Note that now we have multiple choices regarding how

[2]Note that we cannot conclude here that the losses of generator and discriminator become stabilized around the 5000th batch. To make more valid conclusion, more epochs are needed to observe the change of losses through time.

to add the dense layers. The first choice is to only add one dense layer with 10 units and softmax activation function; the second choice is to add a dense layer with many units, and then add another one with 10 units and softmax activation. We will experiment them both. In addition, during model fitting, we can also decide whether to unfreeze all layers after certain number of epochs. We will experiment it too and demonstrate all the results below. To make the comparisons meaningful, we will fix the total epoch number to be 15.

4.1 Trial 1: Transfer Learning only

In our first trial, we only replace the last dense layer and make it the only trainable one during classification. After 15 epochs, the training accuracy is around 0.89 and the validation accuracy is around 0.90.

4.2 Trial 2: Transfer Learning & Fine-Tuning

Based on trial 1 above, we add fine-tuning to our training process. After experimenting with different choices, I freeze all layers but the last one during the first 5 epochs, and then unfreeze them for the following 10 epochs. The resulting training accuracy is around 0.91 and validation accuracy is around 0.92. We can see there are slight increases in both accuracy values after adding fine-tuning to the training process.

4.3 Trial 3: Transfer Learning & Fine-Tuning with Two Dense Layers

In this trial, after deleting the last layer of the discriminator model, I add two more dense layers: the first one has 128 units and the second one has 10 units with softmax activation. Now the last two layers are identical to those in our CNN model in Section 2. After experimenting, I freeze all layers but the last two added ones during the first 7 epochs, and unfreeze them for the following 8 epochs. Both the training accuracy and validation accuracy are around 0.94. At here, we get further improvements compared to the second trial and the results are very close to the baseline CNN model.

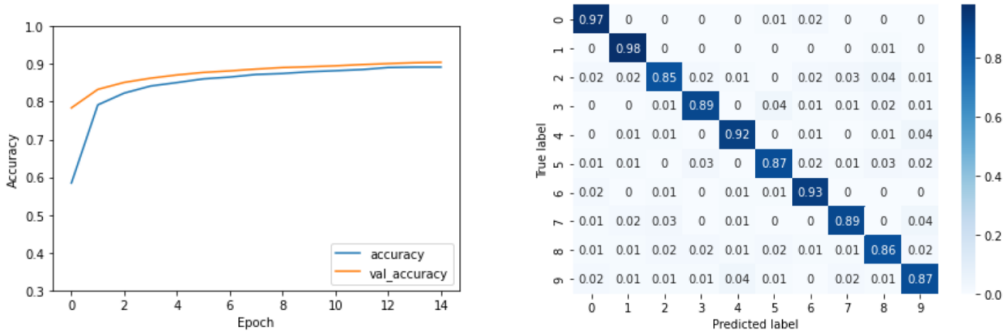


Figure 4: Trial 1 Performance; training and validation accuracy are 0.89 and 0.90 respectively

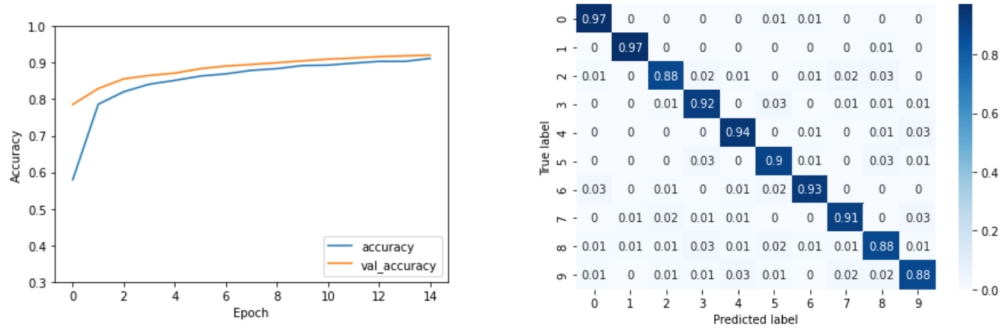


Figure 5: Trial 2 Performance; training and validation accuracy are 0.91 and 0.92 respectively

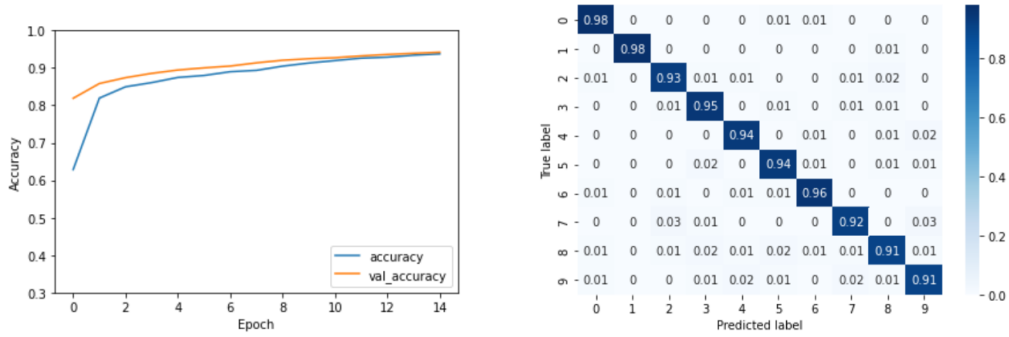


Figure 6: Trial 3 Performance; training and validation accuracy are both 0.94

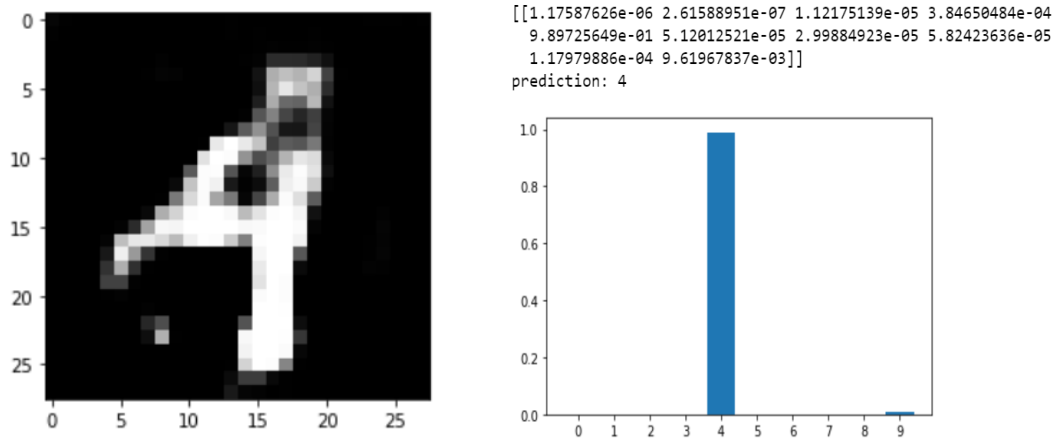
From the three figures above, we see that the performance gets better from trial 1 to trial 3. In particular, the correct classification percentage for all digits gets increased from trial 2 to trial 3. This implies that adding a second layer allows better adaptation of the discriminator towards classification task. In addition, if we observe all of the four confusion matrix heatmaps together (including the CNN one in Figure 1(b)), we can see that the most misclassified digits are 2,7,8 and 9. Some typical cases are misclassifying 7 as 2 and 2 as 7, misclassifying 9 as 7 and 7 as 9. These misclassification cases make sense, since the similarities between the strokes of 2 and 7, 7 and 9 make them harder to classify than other digits even for human.

5 Summary

We can see that, with only 10000 training images and 50 epochs, the GAN model's discriminator, after modification, can achieve decent performance during digits classification. The accuracy is only 2 percent below the original CNN model. This means that the GAN model is suitable for both the task of digits generation and the task of digits classification. It also implies that convolutional neural network can achieve great versatility through techniques of transfer learning and fine-tuning.

There are multiple ways to extend this project. In future work, we can increase the epoch number when fitting the GAN model and then examine whether its discriminator can surpass the accuracy of the original CNN model. However, we need to note that the increase of epoch number also means the increase of computation time. In addition, we can increase the epoch number and training images during digits classification for both CNN and GAN's modified discriminator, and experiment with different optimizer and learning rate. In this way, we are aiming towards the best performance of both models on the MNIST dataset. Last but not least, after sufficient training of the GAN model, we can expand our original MNIST dataset by including newly generated images inside and performing training and testing on both the original and generated digits. We can then observe the classification performance on each of them and make more detailed comparisons.

Figure 7 below demonstrates this idea: with only 10000 training digits and 50 epochs, we are able to generate such image with a definite digit shape; we can create its label manually, store its numpy array, and use it as our additional training image. Furthermore, we can also apply additional image processing techniques here to remove the small dot at the lower-left



(a) A well-generated digit after repeated running

(b) Prediction of (a) and its probability distribution using the model in Trial 3

Figure 7: One way of extension

of Fig. 7(a) or sharpen the image in general to make it more accurate. In this way, if the dataset is small (but it should not be too small), this approach allows us to generate new reasonable data that can be used to further improve our performance.

6 References

- `numpy, matplotlib` libraries
- `tensorflow, keras` libraries for convolutional neural network and generative adversarial network. The models are modified from https://www.tensorflow.org/tutorials/images/cnn and https://www.tensorflow.org/tutorials/generative/dcgan. Trasfer learning and fine-tuning are based on https://keras.io/guides/transfer_learning/
- `glob, os, PIL, time, IPython` libraries for saving and restoring checkpoint, displaying running time and reopening stored images.
- `pandas, seaborn` libraries for displaying confusion matrix heatmaps