Mini Project3 Report

Chicheng, Zheng
260874447
chicheng.zheng@mail.mcgill.ca

Cheng, Li
260706615
cheng.li2@mail.mcgill.ca

Muhang, Li
260736135
muhang.li@mail.mcgill.ca

***Abstract*—In this project, we considered the problem of identifying the digit with the highest numeric value in a 3-digit image by applying methods such as Convolutional Neural Network, and transfer learning, with the Keras library run on TensorFlow backend. We investigated how changing the parameters and using a technique called data augmentation can affect the performance of our model. In the end, we achieved an accuracy of 0.97866 on the Kaggle competition.**

I. Introduction

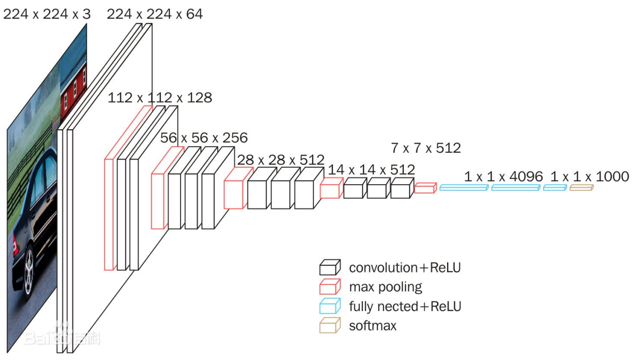
As image recognition has developed rapidly in recent years, it has many practical applications, and one of the most critical areas is handwriting recognition. It can be instrumental when converting images into text files. Handwritten digit recognition is an essential part of handwriting recognition.

This paper presents the work and results based on the supervised learning methodology to find the largest one in three handwritten numbers from 0 to 9 using the Convolutional Neural Network.

Convolutional Neural Network is a kind of feedforward neural network containing convolution calculation, which is one of the representative algorithms of deep learning. The input layer of the convolutional neural network can process multidimensional data. Here we input three-dimensional input data, that is, pixels and RGB channels on a two-dimensional plane.

In this case, we will use VGG16, which is one of the convolutional neural network algorithms developed by the Visual Geometry Group (VGG) of Oxford University [2]. The hidden layer of VGG-16 consists of 13 convolutional layers, 3 fully connected layers, and 5 pooled layers. Among them, the convolutional layer and the fully connected layer have weight coefficients, so they are also called weight layers, and the total number is 13 + 3 = 16, which is the reason for 16 in VGG16.

VGGNet only uses 3×3 convolution kernels and keeps the output feature map size in the convolutional layer unchanged. Double the number of channels and half the size of the feature map output in the pooled layer also simplifies the topology of the neural network and achieves good results.



(Figure 1. The structure of VGGNet [1])

II. Related Work

The digit recognition problem on the MNIST dataset is well studied. Before deciding what kind of classifiers to use for the problem, we want to investigate the previous works by others in the domain of digit recognition and build up our solution from there.

From the original VGG paper, we understood that adding deeper depth to a CNN is beneficial for its classification accuracy and that the conclusion generalizes well to datasets other than the dataset used in the paper[4]. We would start from the VGG-16 model and verify the statement in the paper.

We also understood that, according to Shorten and Khoshgoftaar, that "Data Augmentation can improve the performance of their models and expand limited datasets to take advantage of the capabilities of big data"[5]. This method expands the dataset by performing various changes to an image, such as flipping, shearing, shifting ,and rotation. We want to implement this method in our classifier. It would be interesting to see how the Data Augmentation method can affect our accuracy result.

III. Tool

PyCharm 2019.2.4 (Professional Edition)
Anaconda 2019.10 with Python 3.7
tensorflow-gpu 1.14.0
keras-gpu 2.24
NVIDIA CUDA Toolkit 10.1 Update 2
NVIDIA cuDNN 7.6(v100)

IV. Dataset and Setup

The modified MNIST competition on Kaggle provides a labeled dataset of 50,000 images saved as (128*128) arrays in which the labels indicate the most significant numbers are recognized. And a dataset of 10,000 images for testing purposes. We use pandas.read_pickle(), The method in the example provided on Kaggle to read the training dataset and test dataset. Moreover, pandas.read_csv() for the labels of the training dataset.

We have noticed that in this data set, the images are grayscale. In order to use ImageNet's pretrained weight, we convert the grayscale images into RGB images, which is to change [a] to [a, a, a] without affecting the properties (colors) of the image itself.

Similar to other neural network algorithms, the input features of the convolutional neural network need to be normalized due to learning using the gradient descent algorithm. The input data are pixels whose values distributed at [0, 255] can be normalized to the interval [0, 1]. The standardization of input features is beneficial to improve the learning efficiency and performance of convolutional neural networks.

Then we split the training dataset and the corresponding labels into two parts: the training set and the validation set with the split ratio 9:1, which is 45,000 samples for training and 5,000 samples for validation.

In order to achieve more accurate results, we will try to expand the size of the training data by rotating the original images within a specific range of random angles, shear the original images along an axis, and so on. Also, data Augmentation prevents overfitting by modifying limited datasets to possess the characteristics of big data.

Finally, we use the preprocess_input() method provided by the vgg16 model to get our normalized training dataset to be vgg16 ready.

V. Proposed Approach and results

For the instantiation and compilation of the vgg16 model, there are some parameters we do not have many choices, and we list them here as follows:

1. Instantiation:
   weights='imagenet'
   include_top=False
   input_shape=(128, 128, 3)
2. Compilation:
   loss='sparse_categorical_crossentropy'
   metrics=['accuracy']

To find the best combination of the other hyperparameters, we designed two sets of controlled trials. For the first experiment, we did not do data augmentation, only change the batch size, the number of epochs, optimizer, and learning rate. For the selection of optimizer, adam and SGD are two kinds of optimizers that are widely used in convolutional neural networks, so we choose these two to compare. In the second experiment, we have the same controlled variables, but we do use data augmentation.

batch size	epochs	optimizer	learning rate	validation /test accuracy
32	12	adam	0.0001	99.17% / 97.87%
32	20	adam	0.0001	99.56% / 97.43%
128	12	adam	0.0001	99.52% / 96.67%
32	12	adam	0.001	26.78% / 24.80%
32	20	SGD	0.0001	93.16% / 88.30%

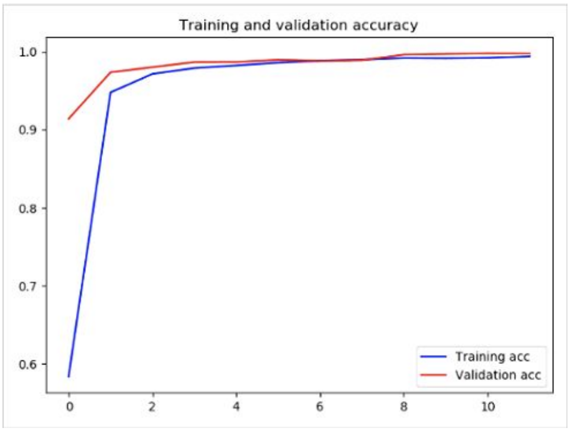
(Table 1: VGG16)

batch size	epochs	optimizer	learning rate	validation /test accuracy
32	12	adam	0.0001	98.59% / 97.77%
32	20	adam	0.0001	98.31% / 97.27%
128	12	adam	0.0001	95.77% / 94.70%
32	12	adam	0.001	21.24% / 21.73%
32	20	SGD	0.0001	81.00% / 79.87%

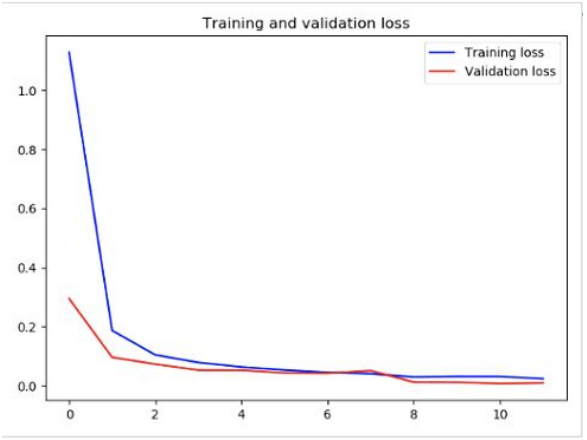
(Table 2: VGG16 with data augmentation)

(Note: Validation accuracy represents the accuracy we tested using a portion of the training dataset. And test accuracy is the accuracy of the system scored after we submitted the prediction.csv to the Kaggle competition.)

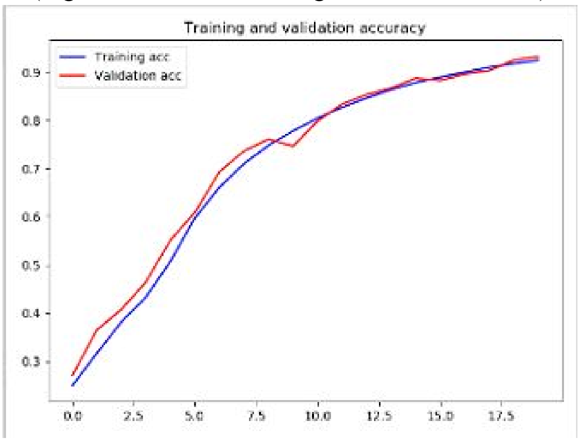
We can easily observe excellent performance when the learning rate is 0.0001, and the optimizer is adam (accuracies are all higher than 95%). However, when the learning rate is 0.001, it is too large for the model to converge, the loss never drops, and the accuracy never rises from the first epoch to the end(always around 26%).



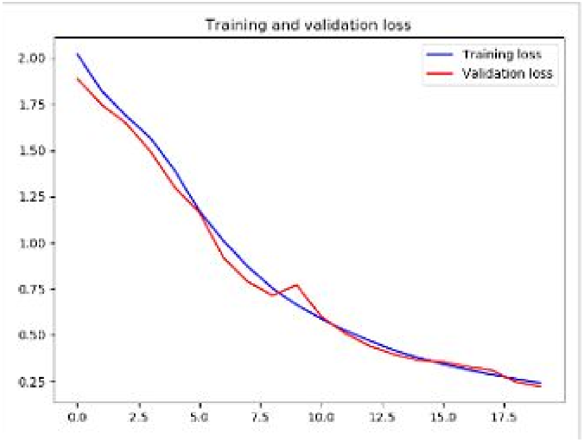
(Figure 2. VGG16 Training and Validation accuracy)



(Figure 3. VGG16 Training and Validation loss)

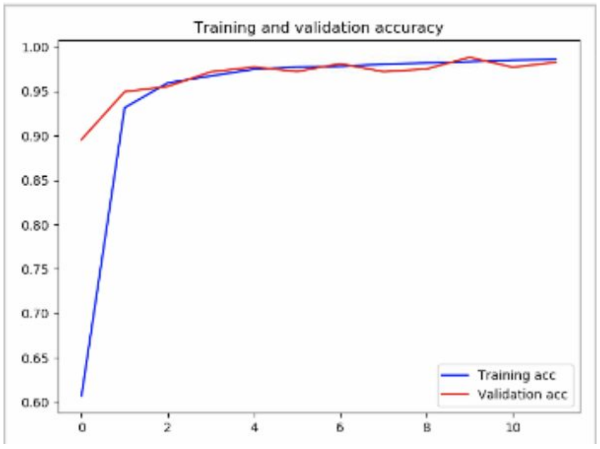


(Figure 4. SGD Training and Validation accuracy)

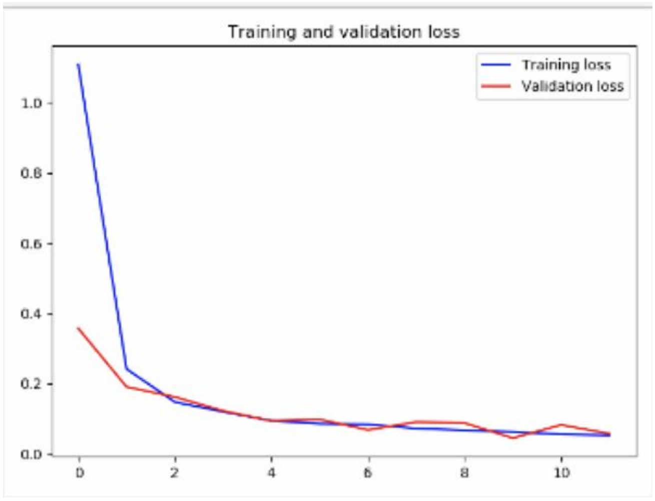


(Figure 5. SGD Training and Validation loss)

We can see from the above figure (Figure 2, 3, 4, 5) that different optimizers have different learning curves. Adam can learn a lot in the first 1~3 epoch to achieve high accuracy and low loss. Nevertheless, the progress in the latter epochs is relatively little. SGD, although the final result is not as good as adam, its learning curve is smoother and closer to linear, so if we run more epochs with VGG16 with SGD optimizer, we may be able to achieve a good result.



（Figure 6. VGG16 with Data Augmentation Training and Validation accuracy)



(Figure 7. VGG16 with Data Augmentation Training and Validation loss)

We can see from Figure 2, 3, 6, 7 that the learning curve for both cases are approximately the same, with one significant difference that the VGG16 model with Data Augmentation presents "zig-zag" patterns in both validation accuracy and validation loss. We think the reason behind the "zig-zag" pattern is that the Data Augmentation process adds noise to the dataset. The added noise would help prevent our model from overfitting on the given training set, according to Chris M. Bishop, that "It is well known that the addition of noise to the input data of a neural network during training can, in some circumstances, lead to significant improvements in generalization performance"[6].

VI. Discussion and Conclusion

1. Conclusion.

   i. According to Table 1, compared to the first and third samples, it can be concluded that a larger batch size will not have a better performance. Besides, by comparing 12 epochs and 20 epochs' accuracies, it can be shown that the improvement of the epoch does not necessarily lead to better performance. The reason for the above two situations may be overfitting. By viewing the second row and the last row, it is evident that adam will result in significantly higher accuracy than SGD. Moreover, VGG with adam optimizer is a 'quick learner' comparing to SGD since it always reaches a good accuracy/loss performance within three epochs. Therefore, adam is a better choice when choosing the type of optimizer.

   ii. Data augmentation does not help with improving the final accuracy comparing to each row in Table 2 and Table 1, respectively.

2. Future work.

   i. Try to run more epochs with the SGD optimizer because from the perspective of Figures 4 and 5, SGD has not yet reached its bottleneck.

   ii. We have just discussed the VGG model so far. Stacking different models may be a possible approach to improve the accuracy.

   iii. We can also try early stopping via a callback called 'EarlyStopping' provided by Keras.

VII. Appendix

Introduction to Adam Optimizer and SGD Optimizer:

1. SGD optimizer:

   The algorithm immediately calculates the gradient of the loss function to update the parameters every time one data is read.

   $\theta = \theta - \eta \cdot \nabla_\theta J(\theta;x(i);y(i))$

2. Adam Optimizer:

   The Adam optimization algorithm is intuitively a combination of RMSprop and AdaGrad. It comprehensively considers the first Moment Estimation of the gradient and the Second Moment Estimation, which is the uncentralized variance of the gradient and calculates the update step size.

VIII. Statement of Contributions

Cheng Li: Data processing, Implementation of VGG and other models, Report editing
Muhang Li: Data processing, Implementation VGG with data augmentation, Report editing.
Chicheng Zheng: Data collection and analysis, Making charts and tables, Report editing.

IX. References

[1] Ng, A., Kian, K. and Younes, *B. Convolutional Neural Networks, Deep learning.* . Coursera and deeplearning.ai. 2018

[2] Simonyan, K. and Zisserman, A., 2014. *Very deep convolutional networks for large-scale image recognition.* arXiv preprint arXiv:1409.1556.

[3] LeCun, Y. and Bengio, Y., 1995. *Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks*, 3361(10), 1995.

[4] Deng, J. and Dong, W. and Socher, R. and Li, L.-J. and Li, K. and Fei-Fei, L. 2009. *ImageNet: A Large-Scale Hierarchical Image Database* CVPR09

[5] Shorten, C. & Khoshgoftaar, *T.M. J Big Data* (2019) 6: 60. https://doi.org/10.1186/s40537-019-0197-

[6] Bishop, C. M. (1995). Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 7(1), 108–116. doi: 10.1162/neco.1995.7.1.108