COMP 551 MiniProject3

Group 30: Xinchen Hou, Xinran Xiong, Ruoyu Wang

Abstract

In this project, we investigated the performance of different variations of Convolutional Neural Network (CNN) models[1] on recognizing hand-written digit-letter combinations from the MNIST dataset. In the data preprocessing phase, labelled image data is divided into training and validation set, where we used training datasets to train different models including basic CNN, VGG-16, ResNet-34 and DenseNet-121, and used validation dataset to calculate validation accuracy. Additional techniques such as data augmentation, data transformation and image denoising are also tested on models in order to speed up the learning process and achieve better accuracy. In the end, the VGG16 with rotation achieved validation accuracy of 96.5% and test accuracy of 95.8%.

1 Introduction

The goal of the task is to predict the labels of a given test dataset, where each image data consists of exactly one digit and one letter written by hand. To produce the result with the highest accuracy, we trained different CNN models that differ in constructions of hidden layers, namely convolutional layers, batch normalization layers, max-pooling layers and down-sample layers[4]. After comparing the training and validation accuracy among the 4 models, we conclude that the VGG16 performs the best in classifying data from the MNIST dataset. We deployed hyperparameter tuning for variables like learning rate and the number of layers. To further improve our model and prevent overfitting, we performed data augmentation by adding rotation and introducing Gaussian noise. We also explored the usage of unlabelled data. Combining all the techniques we explored, we realized a training accuracy of 98% and validation accuracy of 96.5% with a batch size of 30 with 120 training epochs.

2 Data Augmentation

Data augmentation, by adding slightly modified copies of already existing data, will help with the problem of overfitting and eventually increase the performance of the model. Through our trials, it is observed that overfitting could happen as the training accuracy is much higher than the validation accuracy. We explored various ways of data augmentation: rotation, Gaussian noise, zoom-in, and so on. Our results showed that rotation worked best for our model. Figure. 2 and 3 are examples of rotation graphs in our dataset. We also try to use the built-in denoise tool to smooth our dataset by removing some noise pixels. However, after our images are denoised, the validation accuracy drops during the training process. Model overfitting happens when we use denoised images. To avoid this issue, we remove denoising from our data preprocessing section. This experience also shows that the noise will help with the model performance.

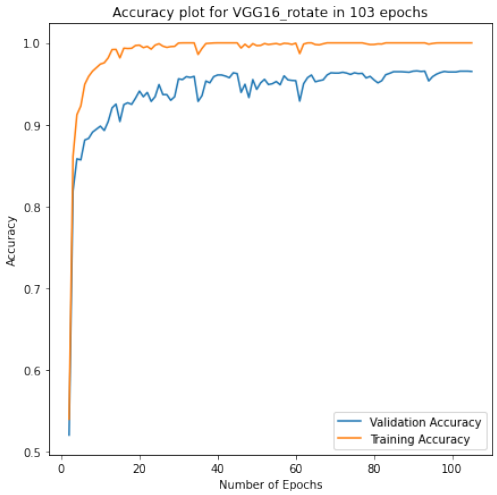
3 Experiments

3.1 basic model

We started our model with a simple CNN consisting of 2 convolutional layers with ReLU activation function, followed by max-pooling layers to keep the dimensions relatively small. However, the model only reached a validation accuracy of 30%, which indicates that the model is too simple for our image classification task and the underfitting issue raises in the training process.

3.2 VGG16

For the next step, we trained our dataset on VGG16, which is one of the best performing architectures to use in many deep learning image classification problems. We read through several papers and found that VGG16, built as a deep convolutional neural network, has been widely used in many deep learning image classification techniques.[2] VGG16 has been widely used. The feature map stacks 5 groups of convolutional and max-pooling layers with increasing dimensions, and batch normalization is followed after each layer to stabilize output and improve learning speed. We believe the VGG16 model will be suitable in our mnist image classification, as it has good performance and is not too complicated to train. It turned out that VGG16 showed the best performance amongst all the models we've tried (e.g., Densenet, Resnet[3]). Figure 1. shows the changes in training accuracy and validation accuracy with epochs. Figure 7. in the Appendix shows the changes of the cross-entropy loss.



.4

Figure 1: Changes of accuracy with epochs, VGG-16, with data augmentation

3.3 special trials

We also had a diversity of interesting trials. First, we tried separate predictions. That is, we trained two models separately, one for the numbers and the other for the letters. After training, we used the two models to predict the number and alphabetic letter and concatenated the results as the final label. Moreover, through feeding the unlabelled data to our best-performed model and combining them with the predicted labels, we gain a new dataset that could represent the general distribution of the data. Unluckily, these two didn't make it to increase the validation accuracy. We will discuss the potential improvement in using the unlabelled data in the discussion section.

4 Results

Table 1 shows that VGG16 achieves the highest validation accuracy(96.5%) among all the competitors after around 100 epochs. Figure 1 gives us a visualization of how the training and validation accuracy changes as we increase the training iterations. There is a huge increase in accuracy from 1 to 40 epochs. After the validation accuracy exceeds 90%, there exist some small fluctuations(around 96.5%) for validation accuracy.

	base CNN	VGG16	Resnet	Densenet
Training accuracy	0.46	0.99	0.97	0.95
Validation accuracy	0.30	0.965	0.92	0.94

Table 1: The performance of different CNN Models.

5 Discussion

Our in-depth experiment results show that among all the CNN models we choose, VGG16 achieves the highest validation accuracy around 96.5%. Future optimization could be achieved by conducting model parameter tuning, implementing more sophisticated semi-supervised learning algorithms with unlabeled images and training on more advanced machine learning models.

6 References

1. Myeongsuk Pak, Sanghoon Kim, A review of deep learning in image recognition, 2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT). Available: https://ieeexplore.ieee.org/abstract/document/8320684

2. Jianguo Shi, Yuanyuan Zhao, Image classification optimization and results analysis based on VGG network, 2020 IEEE International Conference on Information Technology,Big Data and Artificial Intelligence (ICIBA). Available: https://ieeexplore.ieee.org/document/9277329

3. Arpana Mahajan, Sanjay Chaudhary, Categorical Image Classification Based On Representational Deep Network (RESNET), 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA). Available: https://ieeexplore.ieee.org/document/8822133

4. Karen Simonyan, Andrew Zisserman, VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION Visual Geometry Group, Department of Engineering Science, University of Oxford. Available: https://arxiv.org/abs/1409.1556

7 Appendices

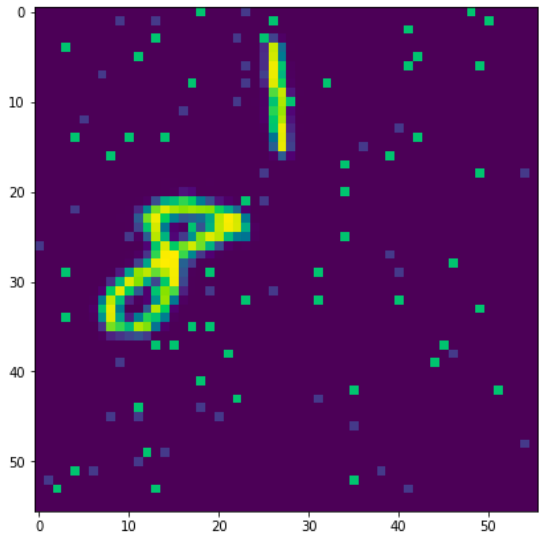


Figure 2: example graph before data augmentation

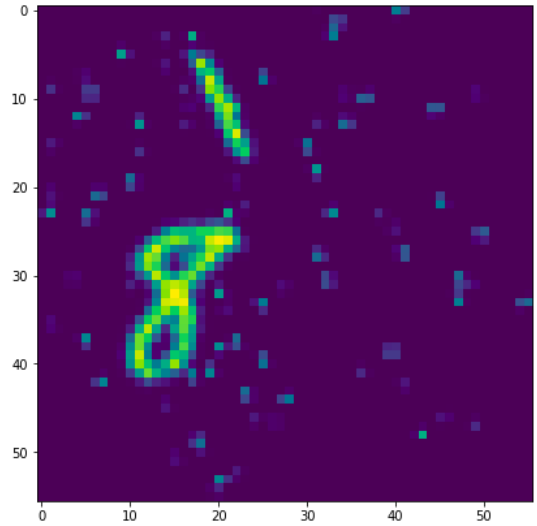


Figure 3: example graph after data augmentation

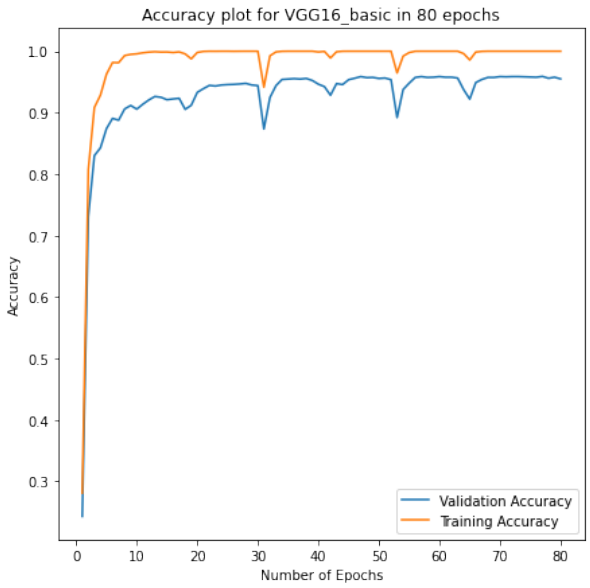


Figure 4: Changes in accuracy of vgg16 model

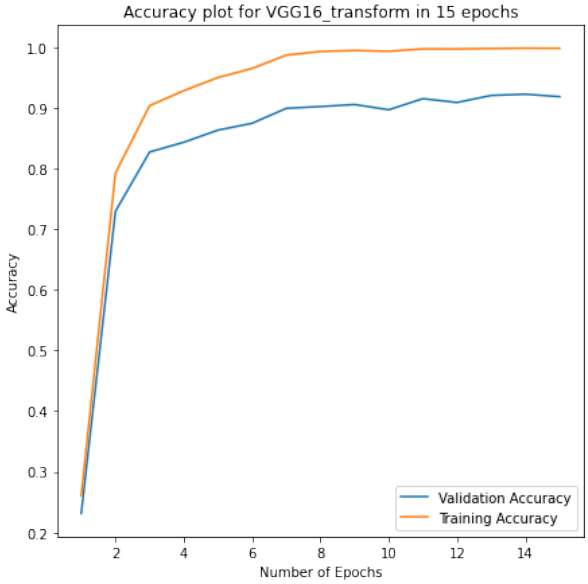


Figure 5: Changes in accuracy of vgg16 model with data transform

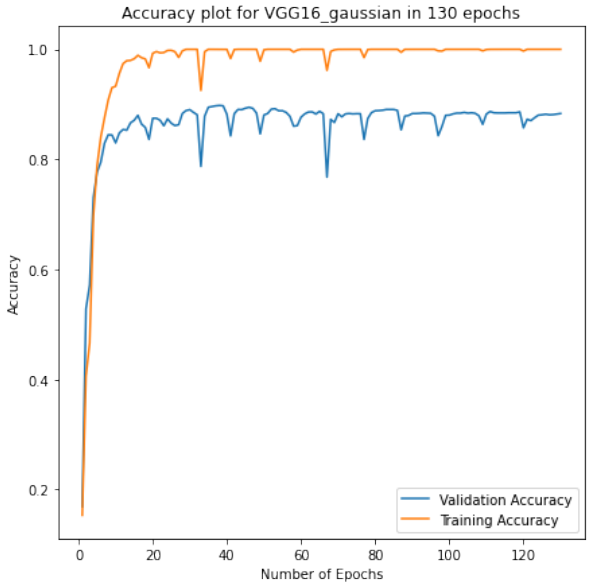


Figure 6: Changes in accuracy of the selected vgg16 model with gaussian noise

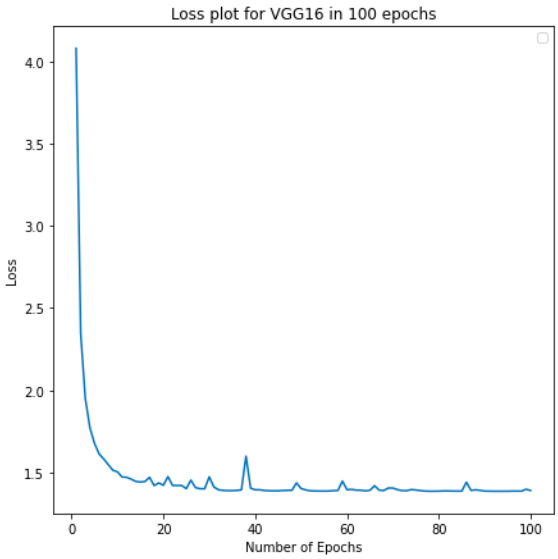


Figure 7: Changes in loss of our selected vgg16 model

```
VGG(
  (features): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): ReLU(inplace=True)
    (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): ReLU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): ReLU(inplace=True)
    (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (19): ReLU(inplace=True)
    (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (26): ReLU(inplace=True)
    (27): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (29): ReLU(inplace=True)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (32): ReLU(inplace=True)
    (33): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (36): ReLU(inplace=True)
    (37): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (39): ReLU(inplace=True)
    (40): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (42): ReLU(inplace=True)
    (43): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (44): AvgPool2d(kernel_size=1, stride=1, padding=0)
  )
  (classifier): Linear(in_features=512, out_features=36, bias=True)
)
```

Figure 8: Layers of our vgg model

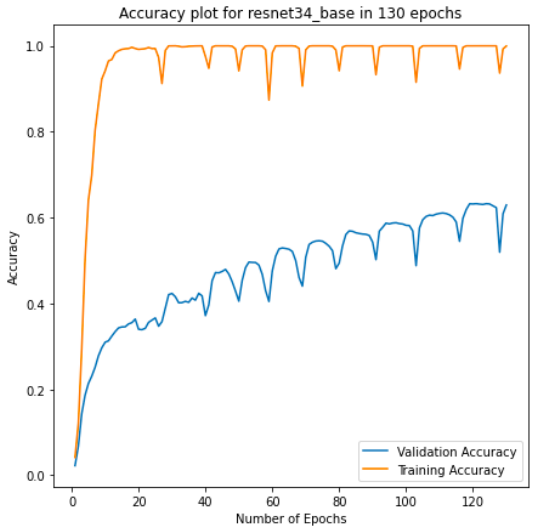


Figure 9: FAILED trail on using resnet model.

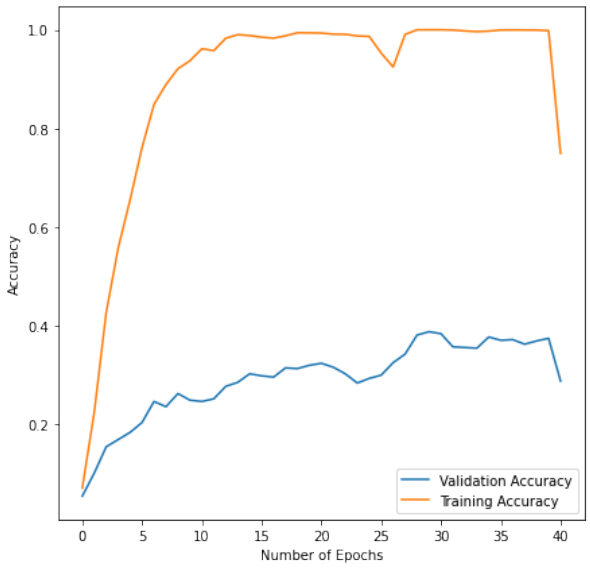


Figure 10: FAILED trail on using resnet model with data transform