Using MobileNet to Tackle Handwritten Chinese Character Recognition Task

Lu CHEN
Taishan College
Shandong University
chenlu.scien@gmail.com

Yulong HUANG
Taishan College
Shandong University
huangyl@mail.sdu.edu.cn

Abstract

We trained a light Convolutional Neural Network to tackle the Handwritten Chinese Character Recognition task under complex conditions. After studying and comparing different popular CNN architectures, we chose MobileNet as our network for its relatively small training cost and real-time application value. In order to solve the problem of insufficient data in the training set, we used Data Augmentation method to generate various writing forms of Chinese characters through simple processing of the input images. As a result, the model after training could be robust to any image transformations such as rotation and flip. This paper introduces the history and current progress of HCCR problem and basic elements of Convolutional Neural Network, emphatically, it describes the architecture of MobileNet and its reduction theory on computational cost. Furthermore, it presents the details of our Data Augmentation and hyperparameters adjustment method. In the end, the analysis and discussion of our final experiment results are included as well.

1 Introduction

1.1 Handwritten Chinese Character Recognition

The 21st century is often known as "The Age of Information", with the amazing development of science and technology, people's expectations for intelligent and automated life are getting higher and higher. And building a bridge between the real physical world and the virtual digital world to provide connections and conversions for eveyone is, without doubt, an extremely important step towards bringing these expectations into reality.

It has always been a vital work for researchers in computer science to seek methods to convert hand-made works (such as handwritten manuscripts and hand-drawn paintings) to digital information, which can support convenient re-editing. Ever since the last century, due to its various challenges and great application value, Optical Character Recognition (OCR) has been a hot topic for researchers, and thus, a lot of algorithms were proposed to deal with this task.

In early stages of Character Recognition research, the problems and difficulties were vastly underestimated by most scientists. At that time, lots of people thought that recognizing the shapes of ten Arabic numbers, zero to nine, were very simple and correctly identifying them would not be difficult. However, in practice, although there are only ten characters and their glyphs are not complicated, it is really hard to ensure a 100% correct recognition rate because of so many different writing styles. And Handwritten Chinese Character Recognition (HCCR) also suffers similar problems.

Despite the fact that the HCCR problem has been studied for more than 4 decades, it remains a partially unsolved problem as a result of its huge vocabulary scale, diverse writing styles, high similarities between certain characters, and so on. Traditional approaches of HCCR, such as the methods based on modified quadratic discriminant function (MQDF) or discriminative learning quadratic discriminant function (DLQDF) achieved quite satisfying performances with no more than 93% accuracy on CASIA-HWDB, an offline HCCR database constructed by Institute of Automation, Chinese Academy of Science, leaving a relatively big margin compared with human performance.

Nowadays, with the rapid growth of deep learning, especially the Convolutional Neural Networks, new breakthrough technology was brought about for HCCR with unprecedented success, greatly narrowed the margin between algorithms and human performance.

1.2 Convolutional Neural Network

Convolutional Neural Network (ConvNet/CNN) is one of the most popular Deep Learning algorithms which takes in an input image, assigns importance (learnable weights and biases) to various aspects in the image and finally is able to differentiate one iamge from all the others. Usually, an RGB image which has been separated by its three colour planes — Red, Green, and Blue is used as input. And the role of ConvNet is to reduce such images into a form which is easier to process, without losing features which are critical for getting a good prediction. With enough training, ConvNets can possess the ability to learn numerous characteristics hidden behind the images.

A ConvNet is made up of Layers. Every Layer has a simple API: It transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters. There are three main types of layers for building ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. These layers are stacked together to form a complete ConvNet architecture.

Convolutional Layer The Convolutional layer is the core component of a ConvNet, which bears most of the computation. Its parameters consist of a set of learnable filters. Every filter is spatially small, but will extend through the full depth of the input volume. For example, a typical filter on the first layer of a ConvNet might be 5x5x3 (width x height x depth) in size. During the forward pass, each filter slides and convolves along the width and height of the input volume and computes dot products between the entries of the filter and the input at any position. Meanwhile, a 2-dimensional activation map is gradually produced, indicating the responses of that filter at every spatial position.

Intuitively, the network will learn filters that only activate when they see certain type of visual features. After obtaining an entire set of filters in each Convolutional layer (each of them will produce an independent activation map), stack these activation maps along the depth wise and produce the output volume.

Pooling Layer The main idea of pooling is down-sampling in order to gradually reduce the spatial scale of the image representation, lessen the number of parameters and the amount of computation in the network, so as to avoid overfitting to a certain extent. The Pooling Layer operates independently on depth slices of the input and resizes them spatially, and max-pooling is one of the most widely used pooling methods. It is common to periodically insert a Pooling layer in-between successive Convolutional layers in a ConvNet architecture.

Fully-Connected Layer Neurons in a Fully-Connected Layer (FC layer) have full connections to all activations in the previous layer. That is to say, each node in a FC layer is directly connected to every node in both the previous and the next layer.

The most evident drawback of a FC layer is that it contains a large number of parameters that require complex computations in training process. Therefore, a lot of measures have been proposed to decrease the number of neurons and connections, one of which is by using the Dropout technique.

2 Case Study

There are several popular and efficient architectures in the field of ConvNets, below are some examples:

2.1 LeNet

LeNet5 [1], one of the earliest ConvNets, was developed by Yann LeCun in 1990's. Through ingenious designing and engineering, it extracts features by Convolution, Parameter Sharing, Pooling and other operations, which successfully reduced computational costs. And uses the FC layer for identification and classification tasks. It can be seen as the start of a large number of contemporary neural network architectures.

The LeNet-5 architecture consists of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully-connected layers and finally a softmax classifier.

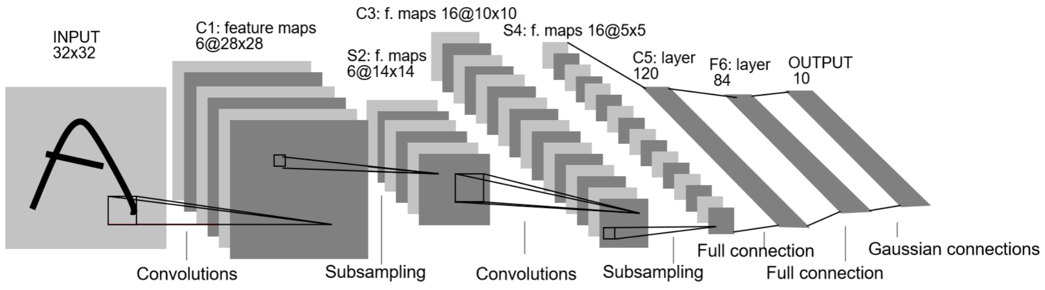

Figure 1: Architecture of LeNet-5, a convolutional neuron network

2.2 AlexNet

Although LeNet achieved good results in image classification, due to the limitations of computers by then, it did not gain much attention. It was not until 2012 that the AlexNet [2] proposed by Alex Krizhevsk et al. won the title of ImageNet Contest with 10.9% lower error rate than the second place, caused widespread attention of ConvNet and even Deep Learning.

AlexNet deepens the network on the basis of LeNet, making it possible to learn richer and higher dimensional image features. Besides deeper network, there are also some other important promotions. First, Replacing the previous Sigmoid with Relu as the activation function, so that the superposition of multiple network layers is no longer limited by simple linear transformation, but has stronger capability to do nonlinear operations. Second, training with multiple GPUs. Last but not least, using Data Augmentation, Overlapping Pooling and Dropout technique to avoid overfitting.

Data Augmentation By employing two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, AlexNet artificially enlarges the dataset to reduce overfitting on image data

Overlapping Pooling Pooling operations do not overlap in LeNet, because the length of pooling window is equal to that of every moving step of pooling window. However, in AlexNet, pooling operations are overlapped. With nothing changed on output dimensions, overfitting is suppressed to some extent.

Dropout technique The Dropout technique operates by setting the output of each hidden neuron to zero with probability 0.5. The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in backpropagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

AlexNet consists of 5 Convolutional Layers and 3 Fully Connected Layers. The first two Convolutional layers are followed by the Overlapping Max Pooling layers that we describe next. The third, fourth and fifth convolutional layers are connected directly. The fifth convolutional layer is followed by an Overlapping Max Pooling layer, the output of which goes into a series of two fully connected layers. The second fully connected layer feeds into a softmax classifier with 1000 class labels.

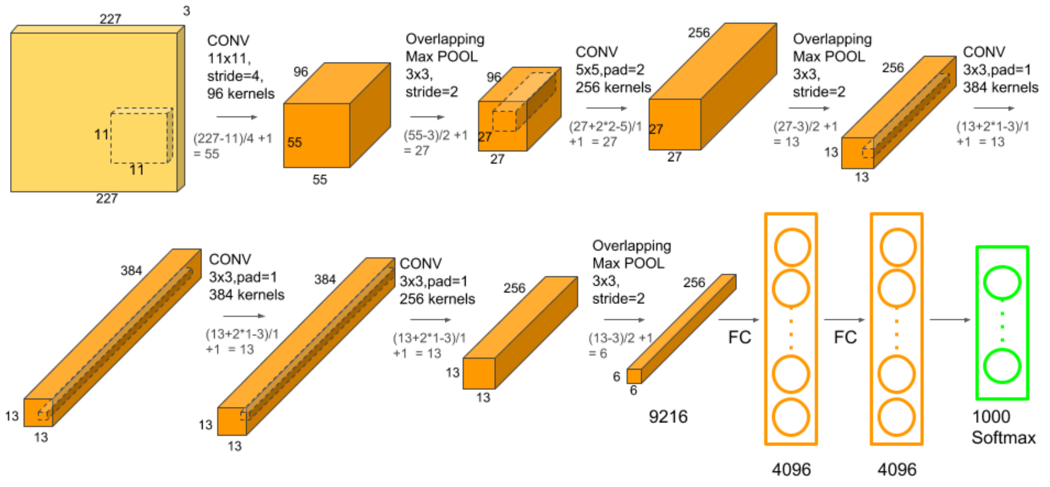

Figure 2: Architecture of AlexNet, it contains eight learned layers — five convolutional and three fully-connected

2.3 ResNet

Residual Network [3] developed by Kaiming He et al. was the winner of ILSVRC 2015. It features special skip connections and a heavy use of Batch Normalization (Batch Norm). The architecture is also missing fully connected layers at the end of the network. ResNet makes it possible to train up to hundreds or even thousands of layers and still achieves compelling performance.

The core idea of ResNet is introducing a so-called "identity shortcut connection" that skips one or more layers, as shown in the following figure:

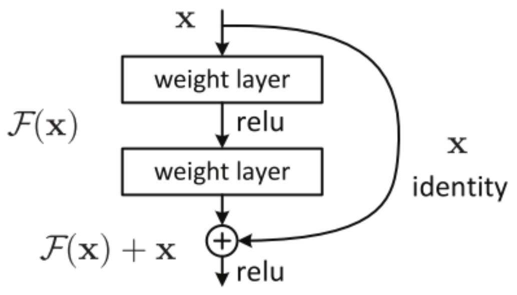

Figure 3: Residual learning: A residual block

Instead of learning a direct mapping of 'x' to 'y' with a function H(x) (A few stacked non-linear layers), define the residual function using F(x) = H(x) - x, which can be reframed into H(x) = F(x)+x, where F(x) and x represents the stacked non-linear layers and the identity function (input = output) respectively. The author's hypothesis is that it is easy to optimize the residual mapping function F(x) than to optimize the original, unreferenced mapping H(x). If the identity mapping is optimal, we can easily push the residuals to zero (F(x) = 0) than to fit an identity mapping (x, input=output) by a stack of non-linear layers. So, this function F(x) is what the authors called Residual function.

3 MobileNet

ConvNets are becoming more and more popular in the field of deep learning, especially computer vision, ever since AlexNet won ILSVRC 2012 (an ImageNet Challenge). For most of the related research, the general trend is to design deeper and more complex neuron networks to reach higher accuracy. However, when it comes to efficiency, since these advances hugely increased the network size and computational cost, the performance is not so satisfying. In many real world applications such as robotics, self-driving car and augmented reality, where the algorithm needs to perform on a computationally limited platform in a real-time fashion, the huge network and large computation severely harm user experience, and could even cause safety issues.

Since smart mobile terminals have been widely used in recent years, more and more tasks are launched through these devices. And the HCCR task certainly has wider application value in real scenes than other tasks carried out by ConvNets, we chose MobileNet as our network architecture in order to achieve real-time processing. In this section, we will first explain the core layers (i.e. depthwise separable filters) which MobileNet is built on. We will then describe the network structure of MobileNet Version 1 and 2, and analyze the reason why they are faster.

3.1 Depthwise Separable Convolution

The basic component of MobileNet model is called Depthwise Separable Convolution. It is a form of factorized convolutions which factorize a standard convolution into a depthwise convolution and a pointwise convolution (i.e. 1x1 convolution) [8].

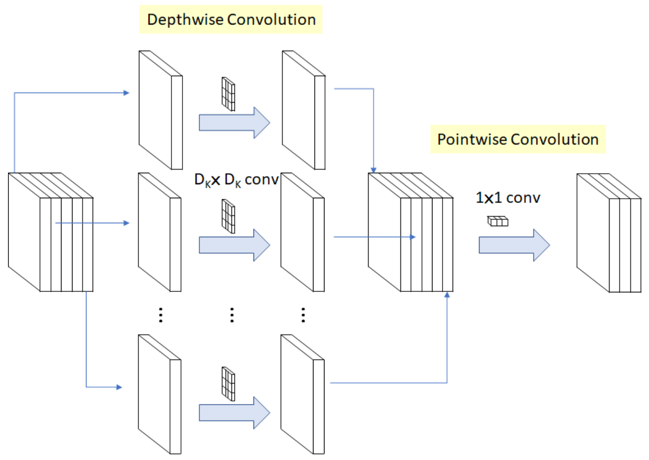

Figure 4: Depthwise separable convolution

As seen in Fig 4, the depthwise convolution applies a single filter to each input channel and the pointwise convolution then applies a pointwise convolution to combine the outputs of the depthwise convolution. For a standard convolution, both the filtering and combining operation are done in one step. However, the depthwise separable convolution makes one small factorization, which splits standard convolution into two separate layers, one for filtering and one for combining. Through this alteration, the computation cost and model size are drastically reduced while the performance of trained model remains perfect.

3.2 Computation Comparison

Here is a rough estimate for standard convolution, the computational cost of building blocks can be calculated as follow: let H x W denotes the spatial size of output feature map, N denotes the number of input channels, K x K denotes the size of convolutional kernel, and M denotes the number of output channels, the computational cost of standard convolution becomes HWK^2NM.

Below is an intuitive illustration about how convolution in both spatial and channel domain is done for standard convolution. The lines between input and output is to visualize dependency between input and output. The number of lines roughly indicate the computational cost of convolution in spatial and channel domain respectively.

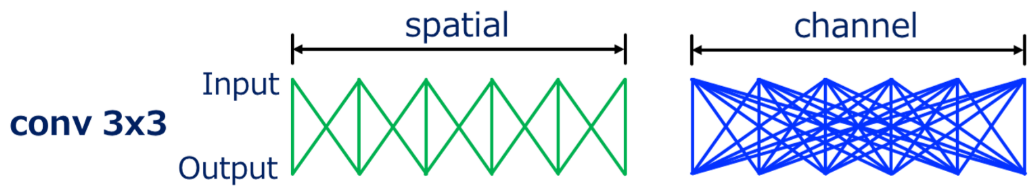

Figure 5: An intuitive illustration on how convolution 3x3 operates in spatial and channel domain

For instance, conv 3x3, the most commonly used convolution, can be visualized as shown above. We can see that the input and output are locally connected in spatial domain while in channel domain, they are fully connected.

In depthwise convolution, convolution is performed independently for each of input channels. It significantly reduces the computational cost by omitting convolution in channel domain, which ends up to HWK^2N.

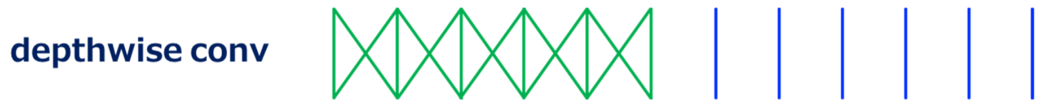

Figure 6: An intuitive illustration on how depthwise convolution operates in spatial and channel domain

As for pointwise convolution (conv 1x1), which is used to change the size of channels. The computational cost is HWNM because the size of kernel is 1x1, resulting in $\frac{1}{9}$ reduction in computational cost compared with conv 3x3. This convolution is used in order to “blend” information among channels.

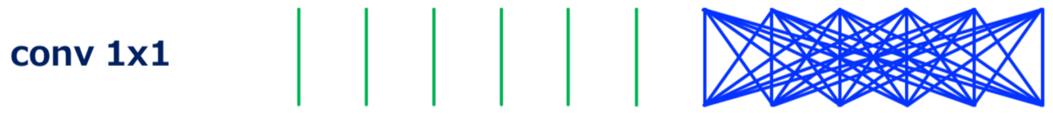

Figure 7: An intuitive illustration on how pointwise convolution (convolution 1x1) operates in spatial and channel domain

MobileNet is a stack of the separable convolution modules which are composed of depthwise conv and conv1x1 (pointwise conv). The separable conv independently performs convolution in spatial and channel domains. This factorization of convolution significantly reduces the computational cost from HWK^2NM to HWK^2N (depthwise) + HWNM (conv1x1), which equals to $HWN(K^2 + M)$. In general, M»K^2 (e.g. K=3 and $M \geq$ 32), the reduction rate is roughly $\frac{1}{8} - \frac{1}{9}$.

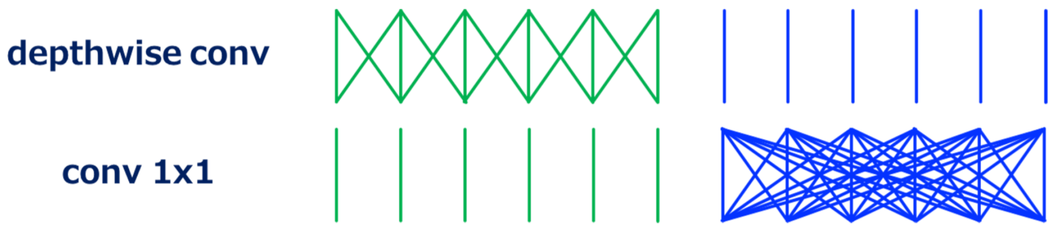

Figure 8: an intuitive illustration on how the separable convolution modules of MobileNet V1 operates in spatial and channel domain

4 Model Architecture

4.1 Network Structure V1

The first version of MobileNet is basically built upon depthwise separable convolutions mentioned above, except that the very first layer is a full/standard convolution. The architecture of MobileNet Version 1 is defined in Table 1.

Type / Stride	Filter Shape	Input Size
Conv / s2	3 x 3 x 3 x 32	224 x 224 x 3
Conv dw / s1	3 x 3 x 32 dw	112 x 112 x 32
Conv / s1	1 x 1 x 32 x 64	112 x 112 x 32
Conv dw / s2	3 x 3 x 64 dw	112 x 112 x 64
Conv / s1	1 x 1 x 64 x 128	56 x 56 x 64
Conv dw / s1	3x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 256	56 x 56 x 128
Conv dw / s2	3 x 3 x 256 dw	56 x 56 x 128
Conv / s1	1 x 1 x 256 x 256	28 x 28 x 128
Conv dw / s1	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 512	28 x 28 x 256
Conv dw / s2	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 512	14 x 14 x 256
Conv dw / s1 Conv / s1	3 x 3 x 512 dw 1 x 1 x 512 x 512	14 x 14 x 512 14 x 14 x 512
Conv dw / s2	3 x 3 x 512 dw	14 x 14 x 512
Conv / s1	1 x 1 x 512 x 1024	7 x 7 x 512
Conv dw / s2	3 x 3 x 1024 dw	7 x 7 x 1024
Conv / s1	1 x 1 x 1024 x 1024	7 x 7 x 1024
Avg Pool / s1	Pool 7 x 7	7 x 7 x 1024
FC / s1	1024 x 1000	1 x 1 x 1024
Softmax / s1	Classifier	1 x 1 x 1000

Table 1: MobileNet V1 body architecture

Counting depthwise and pointwise convolutions as separate layers, MobileNet V1 has 28 layers in total. Apart from the last one, all other layers are followed by a batchnorm and ReLU nonlinearity. The final fully connected layer has no nonlinearity and feeds into a softmax layer for classification. Figure 9 contrasts a layer with regular convolutions, batchnorm and ReLU nonlinearity to the factorized layer with depthwise convolution, 1x1 pointwise convolution as well as batchnorm and ReLU after each convolutional layer. Down sampling is handled with strided convolution in the depthwise convolutions as well as in the first layer. A final average pooling reduces the spatial resolution to 1 before the fully connected layer.

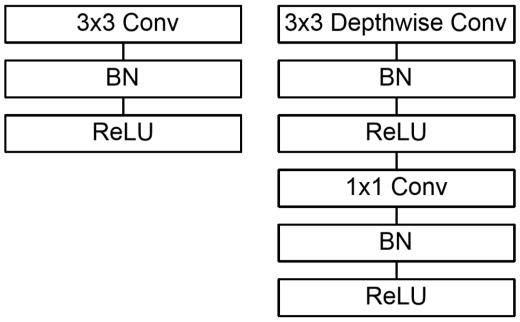

Figure 9: Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU

4.2 Network Structure V2

The basic building block of MobileNet V2 [9] is a bottleneck depth-separable convolution with residuals, which is similar to the residual unit with bottleneck architecture of ResNet. The modified version of the residual unit where conv3x3 is replaced by depthwise convolution is roughly illustrated as follows. The detailed structure of this block is shown in Table 2. Letting T denote an expansion factor of channel dimension, the computational cost of two conv1x1s is $\frac{2HWN^2}{T}$ while that of conv1x1 in separable conv is HWN^2.

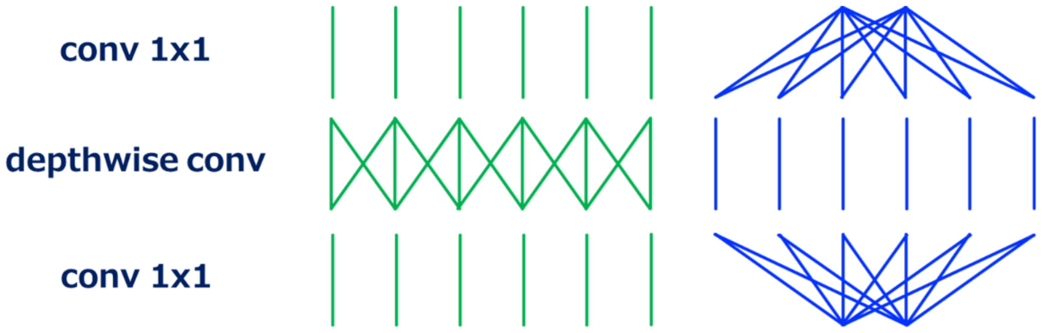

Figure 10: An intuitive illustration on how the separable convolution modules of MobileNet V2 operates in spatial and channel domain.

Input	Operator	Output
h x *w* x *k*	1 x 1 conv 2d, ReLU6	*h* x *w* x *(tk)*
h x *w* x *tk*	3 x 3 dp s = s, ReLU6	*h/s* x *w/s* x *(tk)*
h/s x w/s x tk	linear 1 x 1 conv 2d	*h/s* x *w/s* x *k'*

Table 2: Bottleneck residual block transforming from k to k' channels, with stride s, and expansion factor t

The architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers described in the Table 3. We use ReLU6 as the non-linearity because of its robustness when used with low-precision computation. We always use kernel size 3x3 as is standard for modern networks, and utilize dropout and batch normalization during training.

Input	Operator	t	c	n	s
224 x 224 x 3	conv2d	-	32	1	2
112 x 112 x 32	bottleneck	1	16	1	1
112 x 112 x 16	bottleneck	6	24	2	2
56 x 56 x 24	bottleneck	6	32	3	2
28 x 28 x 32	bottleneck	6	64	4	2
14 x 14 x 64	bottleneck	6	96	3	1
14 x 14 x 96	bottleneck	6	160	3	2
7 x 7 x 160	bottleneck	6	320	1	1
7 x 7 x 320	conv2d 1x1	-	1280	1	1
7 x 7 x 1280	avgpool 7x7	-	-	1	-
1 x 1 x 1280	conv2d 1x1	-	k	-	

Table 3: MobileNet V2 Body Architecture: Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3 x 3 kernels. The expansion factor t is always applied to the input size as described in Table 2.

5 Data Augmentation

The easiest way to help deep learning model perform better is to use large dataset. However, a truly qualified dataset that contains every variation of the feature is very hard to construct and thus almost impossible to find. One efficient way to expand the dataset and make better use of the current data is data augmentation.

Common augmentation methods/operations for images include Flip, Rotation, Transposition, Scale, Crop, Colour Adjustment and adding Gaussian Noise. At first, we randomly select one operation from all the methods mentioned above for each image in the training dataset, then save them to the disk for training. But this caused the repetition for every image in each epoch, which means the training process just takes in different subsets of one certain universal set, which has a significant risk of overfitting. And our experiment verified this theory for showing an extremely low accuracy in the test dataset.

To solve this problem, instead of storing all images after augmentation, we randomly augment every image before they were put into the file queue. Thus, input images with the same original character in all epochs will be different from each other. And since the transformed images do not need to be stored on the disk, not only did this method save the storage space, but also speeded up the data reading process.

However, with operations introduced above, the accuracy we got is still very low. We concluded by conducting many experiments that due to the unique font structure of Chinese character, the transpose and rotation operation can make characters very different from the original ones, even make them totally look like another character. Because these operations can cause the data distributions vary greatly, the model cannot capture the right features among so many variations, as a result, the accuracy remains on a low level.

In addition, since the input images are all black and white, the colour adjustments such as contrast, saturation, hue and brightness would not do much help to enrich the writing styles in the dataset. And cutting operations like scale and crop will break the structure of the character, we abandon these augmentation methods as well.

At last, we choose to use three kinds of operations, which are flip up and down, flip left and right, rotate in the range of -30 degrees to 30 degrees to do data augmentation. For each input image, possible operations include doing these transformations randomly and doing not. Following this pipeline, the enhanced dataset will provide more writing styles of each character, and can even robust to images which have been rotated or flipped. It should be stressed out that after processing the images, we resize them to 64x64 for neural network training needs.

6 Experiment

6.1 Dataset

We chose CASIA-HWDB1.1, an offline HCCR database constructed by Institute of Automation, Chinese Academy of Science, as our dataset. The CASIA-HWDB1.1 set contains 3755 classes of level-1 Chinese characters in GB2312-80 (a Coded Chinese Character Set for Information Communication), and for each individual character, there are about 240 images of different handwritten versions sampled from 300 writers for training and 60 for testing. Totally, 897758 images are included in training dataset, and 223991 in testing dataset.

6.2 Adjusting parameters

6.2.1 Learning rate

During the training, the first parameter we focus on is Learning Rate (LR). LR is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. The lower the value, the slower we travel along the downward slope. While this might be a good idea (using a low learning rate) in terms of making sure that we do not miss any local minima, it could also mean that we'll be taking a long time to converge — especially if we get stuck on a plateau region. Bad learning rate can lead to the situation where loss rises instead of reducing, eventually bring negative effect on accuracy.

At first, we set LR as 0.01, but it turns out that convergence only occur on the dataset which has not been augmented. After lots of experiments, we found out that setting LR as 0.001 and decay rate as 0.94 exports the best performance.

6.2.2 Dropout

Dropout is another parameter we pay attention to, it represent the probability of a feature detector been shut down during training, through shut down some feature detector, we can prevent overfitting to some extent and help our network achieve a better performance.

At the very beginning, the dropout was set as 0.97, which is apparently too high. And it led to serious overfitting in the end. Despite the accuracy on training set is above 90%, the accuracy on testing set is only a litter higher than 70%. So we tried to decrease dropout to 0.75 and 0.55 respectively, but still did not solve the overfitting problem. But we found that higher dropout can make accuracy grow faster than the lower one. Finally we changed dropout to 0.5, and got one satisfying result in a 160000 steps training which is shown in the next section.

6.3 Result

With the batch size as 256 and 160000 training steps, the accuracy on one batch while training reached 95%, and the accuracy on the testing set reached 90.4%. But when training steps grew to 170000, overfitting occurred. Although the accuracy on training set is still above 90%, the testing set is only around 25%. We then tried to reduce LR to 0.0005, but the finally accuracy for training set just maintained at 80% or so, and testing set 81% with same training step.

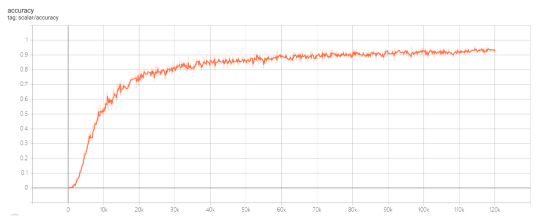

Figure 11: Accuracy curve during training

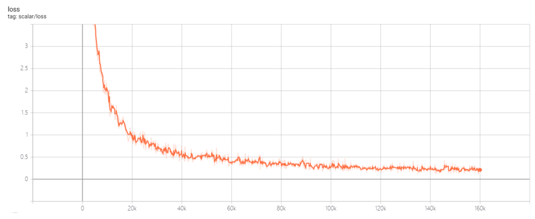

Figure 12: Loss curve during training

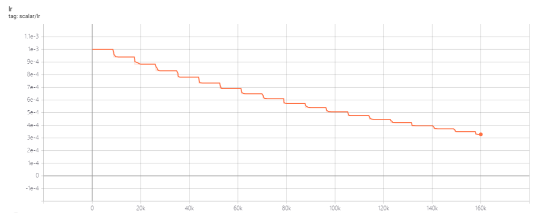

Figure 13: Learning rate curve during training

In the final test, our model predicted on a dataset which contains 9000 images in total, with 3000 images testing on robustness on rotation, flip and resize respectively. (The rotation angle is between -25 degrees and 25 degrees, the scale of resizing width and height is between 0.5 to 2.5.) The accuracy of our model on the whole dataset is 80.57%, and 82.13% on rotation, 93.60% on flip and only 65.97% on the resized image set.

6.4 Analysis and Improvements

The main reason of the low accuracy on resized data set is lack of training data, which trace back to the data augmentation process. We thought by resizing all input images to one uniformed scale (64x64) should train our model robust to all resized images. However, it turns out that for those resized images which the inside character is seriously distorted, our model performed really bad and fail to recognize almost half of the testing images. In addition, we found that some hand written characters in the dataset was written in the wrong way, and some even has cross lines may indicate that the writer scored them out, which may contaminate the training data.

To improve our model's performance, we added a resize operation into our augmentation pipeline, which will randomly shrink or enlarge the width and length of the image to 0.5 to 2.5 times. We presume it will make our model capable of recognizing different input images with various sizes with a relatively high accuracy. And the experiment showed that the accuracy increased to 85.03% on the whole test data set, and the accuracy of rotation, flip, and resize were 81.13%, 91.60%, and 82.37% respectively. Due to the time constraints, the accuracy may be even further improved.

7 Conclusion

In this project we trained a model based on MobileNet, a light Convolutional Neuron Network, to tackle the HCCR problem. With 160,000 training steps which equals to 45 epochs, the accuracy reached over 90% on training set, and 90.4% on the testing set. Using 897758 images from HWDB1.1 dataset and data augmentation method to expand the dataset, our model is not only capable of recognizing normal handwritten characters, but can also support recognizing images which have been flipped horizontally or vertically and rotated between -30 degree to 30 degree. For the future work, we plan to do further research in detecting and recognizing multiple words in the input images to make this project more useful. We also intend to explore methods to train networks robust to ghost (wrongly written) words.

Acknowledgement

This is the designated project of the Machine Learning And Visual Computing course in SDU. The authors would like to thank Prof. Chandrajit Bajaj from University of Texas at Austin for his excellent teaching and patient explaining, and Prof. Jianlong Wu from Shandong University for all the suggestions and guidance he offered in this project. Without their supervision and dedication, we would not have done such amazing work as beginners of Deep Learning.

References

[1] Y LeCun L Bottou Y Bengio & P Haffner (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 1998.*

[2] Alex, K., Elya, S. & Geoffrey, E.H. (2012) ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NIPS 2012).*

[3] Kaiming He Xiangyu Zhang Shaoqing Ren & Jian Sun (2015) Deep Residual Learning for Image Recognition. *arXiv preprint arXiv: 1512.03385v1 .*

[4] Andrew, G.H. Menglong Zhu & Bo Chen et al. (2017) MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv: 1704.04861v1 .*

[5] L. Sifre (2014) Rigid-motion Scattering for Image Classification. *Ph.D. thesis, 2014.*

[6] L. Sifre & S. Mallat (2013) Rotation, Scaling and Deformation Invariant Scattering for Texture Discrimination. *Proc. of CVPR, 2013.*

[7] F. Chollet (2017) Xception: Deep Learning with Depthwise Separable Convolutions. *Proc. of CVPR, 2017.*

[8] M. Lin Q. Chen & S. Yan (2013) Network in Network. *Proc. of ICLR, 2014.*

[9] Mark, S. Andrew, H. Menglong Zhu Andrey Z. & Liang-Chieh Chen (2019) MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv preprint arXiv: 1801.04381v4 .*