Image Processing at ISCAS

PPHP: Project Hand Pose

An Exploration to Develop Neural Networks for Personal Devices

Man Fung Leung, Ka Chung Wong, Chenfeng Han

Supervised by Dr. Libo Zhang

1 Introduction

Before the 2010s, computer vision researchers and engineers relied on hand-crafted features and statistical methods to design vision systems (Krizhevsky, Sutskever & Hinton, 2017). As a result, image recognition was a difficult task that required detailed understanding or professional knowledge. With the emergence of the convolutional neural network (CNN) in 2012, machine learning becomes the de facto approach to image recognition (Krizhevsky, Sutskever & Hinton, 2017). Nevertheless, one common problem with the existing deep learning based methods is that a considerable amount of computational resources, as well as data, are required before the model can achieve satisfactory performance (Goodfellow, Bengio & Courville, 2016). This limits the practicality of these methods in real life on personal devices.

On the other hand, there is a potential demand for deep learning on personal devices. Taking web application as an example, it can enhance the concept of Software as a Service (SaaS). The ability to use the neural network on the client-side can provide users with confidence in their privacy (e.g. grammar checker for confidential documents). Individual developers with limited computational resources in their servers are benefited in terms of trust and flexibility. Other applications include Federated Learning (McMahan & Ramage, 2017) and mobile development. With the incoming era of the Internet of Things, such demand is expected to be increasing.

Therefore, in this project, we aim to explore and share some practical methodologies in developing neural networks for personal devices. We illustrate our results with a web game based on hand pose classification. The remainder of this paper is organized as follows: Section 2 reviews some related work in the literature. Section 3 discusses some practical concerns arose from exploratory data analysis, image pre-processing and application development. Section 4 details the methodology and network architecture. Section 5 reports our experimental results. Section 6 analyzes the web game and model's online performance briefly. Section 7 concludes.

2 Related work

2.1 Image classification

In 2012, AlexNet won the ImageNet competition championship with excellent results. For the first time, AlexNet used ReLU, Dropout and LRN in a CNN, and accelerated the operation of the GPU (Krizhevsky, Sutskever & Hinton, 2017), making AlexNet perform much better than traditional machine learning technology. After that, a considerable number of deep learning models have been developed. We will review three of them, namely VGG16, ResNet and MobileNet.

VGG16 is a deep learning network model proposed by the Visual Geometry Group of Oxford University. One of its improvement over AlexNet is the use of successive 3 by 3 convolution kernels (Simonyan & Zisserman, 2014). The use of stacked small convolution kernels is superior because multiple nonlinear layers can increase network depth to ensure more complex patterns are learned with fewer parameters (Simonyan & Zisserman, 2014).

ResNet (Residual Neural Network) is proposed by four Chinese researchers at Microsoft Research. The main feature of ResNet is to add the idea of Highway Network to the model, allowing the original input information to be passed directly to the later layers (He et. al., 2015). Such residual learning mechanism can tackle the problem of degradation/gradient explosion in a very deep model (He et. al., 2015). Using ResNet, they successfully trained a 152-layer neural network and won the championship in ILSVRC2015.

MobileNet is a deep learning model designed for mobile application by researchers from Google. It transforms the traditional convolution structure into a two-layer convolutional operation, which greatly reduces the computation time without affecting the accuracy a lot (Howard et. al., 2017). Theoretically, it can reduce the time by 8 to 9 times for a CNN using 3 by 3 kernels (Howard et. al., 2017).

2.2 Object detection

The emergence of AlexNet, as well as deep learning, also revolutionize object detection. Currently, object detection algorithms are mainly divided into two categories. The first one is based on regional proposals, such as R-CNN, Fast R-CNN and Faster R-CNN. The second one is based on end-to-end learning, such as YOLO and SSD.

The R-CNN algorithm mainly includes regional proposal, normalization processing, feature extraction, classification and regression (Girshick et. al., 2016). Based on R-CNN, Fast-RCNN optimizes the whole network by adaptive scale pooling, which avoids the redundant feature extraction operation in R-CNN and improves the accuracy and speed of network recognition (Girshick, 2015). For Faster R-CNN, it extracts the candidate frames by constructing the Region Proposal Network (RPN) instead of the selective search method with large time overhead, which further improves the speed (Ren et. al., 2015).

For the other stream, YOLO combines object detection and recognition, which greatly improves the speed (Redmon et. al., 2015). Its background false detection rate is lower than R-CNN and supports the detection of unnatural images (Redmon et. al., 2015). However, one disadvantage is that the object positioning error can be large due to the rough division of the S×S grid (Redmon et. al., 2015). The SSD improves the idea of the regional proposal and uses an RPN

network similar to that of Faster R-CNN (Liu et. al., 2016). The difference is that the SSD uses RPN on multiple feature layers of CNN for classification and Bounding-Box regression (Liu et. al., 2016). Therefore, the detection of small objects on the image becomes more accurate.

3 Practical concerns

3.1 Datasets

The first difficulty of using deep learning based method encountered by individual developers is probably insufficient quality data. This is one of our practical concerns as well. As we try to develop a web game with hand pose classification, we need to collect different hands' images. Nonetheless, taking photos of our hands under different background might lead to a network that only works with our hands, while taking photos of different hands under the same background might lack generalization ability to the surrounding. In light of this, we blend two datasets downloaded from Kaggle (alish_manandhar, 2019; Bruère-Terreault, 2019) as we find that using a single type of data would lead to poor performance in reality.

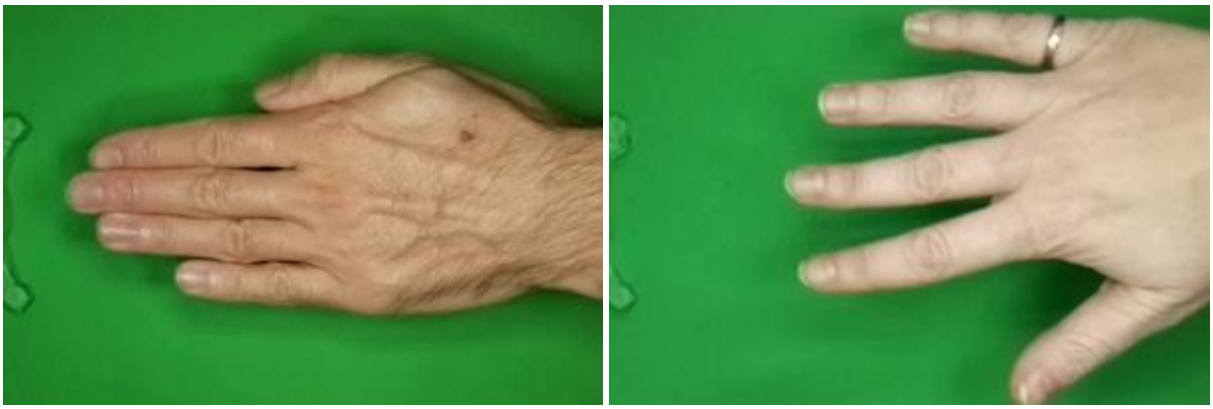

Figure 1: different hands under the same background (Bruère-Terreault, 2019)

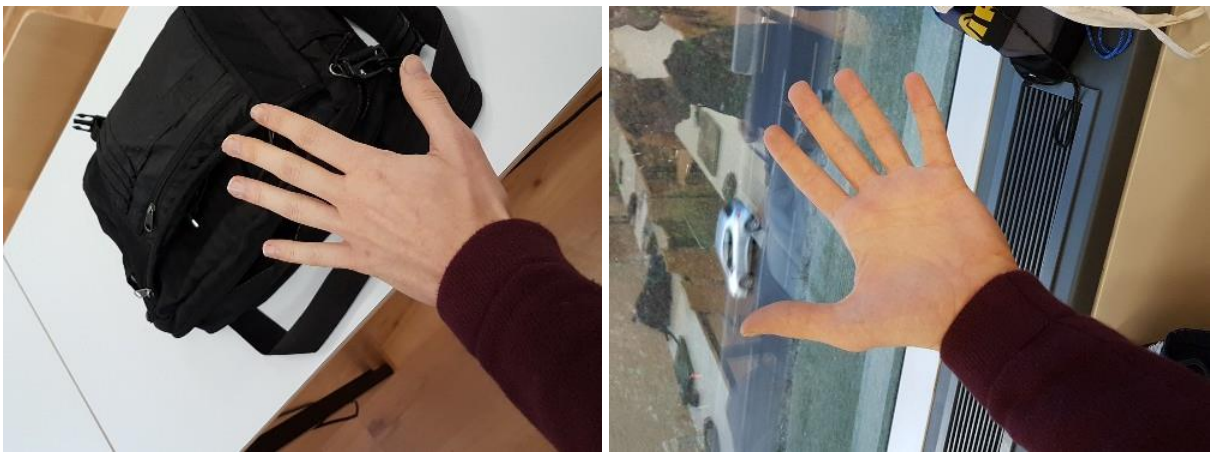

Figure 2: the same hand under different backgrounds (alish_manandhar, 2019)

In addition to multiple data source, we also try to label the region of hand with a bounding box in these images and delete some which were wrongly labeled or of bad quality (like multiple hands with different poses) in alish_manandhar (2019). This may be useful if we try to build another head for hand detection in our model. It also adds value to future users of this combined dataset.

Finally, we balance the composition of the two sources in the validation set to avoid potential bias. Below is a summary of the final dataset after our labeling and amendment:

Category	No. in training set (~70%)	No. in validation set (~30%)	Total size/MB
Paper	1,075	400	136
Rock	1,144	400	134
Scissors	1,155	400	131

Table 1: Summary statistics of the final dataset

3.2 Model

Due to the limited computing power of personal devices, our model has to strike a balance between performance and size. There are several reasons:

Firstly, the model size has a huge impact on the loading time of application. Since privacy protection is the edge of deep learning on personal devices, we would consider loading the model on client-side first to verify its feasibility. This may also shed light on other areas like Federated Learning (McMahan & Ramage, 2017) with millions of personal devices.

Secondly, the network's depth has a positive relationship with processing time. In general, a deeper neural network has more parameters in higher-dimensional space. Therefore, the deeper a network is, the longer it takes for the model to make a prediction. This may hinder our users' experience, especially given the computational limitation of personal devices.

Last but not least, the network's size may affect training efficiency as well. When there is insufficient data, developers cannot train a large neural network under a reasonable number of epochs as the optimizer cannot escape local optima in high-dimensional parameter space. This is true even if we use a pre-trained network, which we will discuss further in Section 5.

In light of these, we compare the statistics of several popular image classification network, assuming an input size of (224, 224, 3). Base on the size statistics, we have preliminarily chosen

MobileNet as the backbone of our hand pose classification network. We will discuss its architecture further in the next section.

Model	No. of parameters	Depth/No. of layers	Size/MB
MobileNet	3,228,864	88	39
ResNet50	23,587,712	52	283
VGG16	14,714,688	23	177

Table 2: Size statistics of MobileNet, ResNet50 and VGG16 as Keras application (keras-team, 2019)

3.3 Application

To deploy the trained model on the client-side, we need to perform model conversion and preferably compression as well. Since the model framework is Keras and the application is web-based, we need a tool to convert Keras model into JavaScript. There are two sets of these tools available: Keras.js (transcranial, 2018) and TensorFlow.js (TensorFlow, 2019). However, Keras.js is no longer active after 2018 (transcranial, 2018). As a result, we will use TensorFlow.js under Keras v.2.0.9 for the application.

Another practical concern, which arises from the nature of web languages, is that the logic/code of the game is visible to players. We can partially tackle this problem by compressing our JavaScript files as well as encrypting the code. However, for educational purpose, we did not do so to the source code on GitHub. Future developer should be aware of this problem when they develop client-side application.

4 Methodology

To address the aforementioned modeling concerns, we try to reduce the complexity of convolution, which is a major mathematical operation in image classification networks. This is inspired by Howard et. al. (2017), who propose Depthwise Separable Convolution in their MobileNet. Compare with the standard convolution, Depthwise Separable Convolution factorizes a standard convolution into a depthwise convolution and a 1×1 pointwise convolution, which uses 8 to 9 times less computation under a 3×3 kernel while maintaining a good accuracy (Howard et. al., 2017).

On the other hand, we apply transfer learning to reduce training time and boost model performance. According to Goodfellow, Bengio and Courville (2016), it is common for computer

vision researchers to use the features from a convolutional network trained on ImageNet and finetune to complete different tasks. For instance, Girshick et al. (2015) use a pre-trained image classification network to solve the problem of object detection.

As a result, we construct our network based on the following architecture (the complete architecture of MobileNet can be found in appendix A1):

Layer/type	Input size	Output size	No of parameters
MobileNet (backbone)	224× 224 × 3	7 × 7 × 1024	3228864
Global average pooling	7 × 7 × 1024	1 × 1 × 1024	0
Dropout	1024	1024	0
Softmax	1024	3	3075

Table 3: Architecture of our hand pose classification network

5 Experiment

The experiments are conducted on an NVIDIA Tesla V100 SXM2 32 GB. For a comprehensive evaluation, we try four underlying networks: a vanilla 5-block (32-64-128-256-512) CNN without pre-training, MobileNet (Howard et. al., 2017), ResNet50 (He et. al., 2015) and VGG16 (Simonyan & Zisserman, 2014) pre-trained with ImageNet data. Training is implemented with stochastic gradient descent using the ADAM scheme with 25 epochs. The learning rate and other hyper-parameters follow the original paper (Kingma & Ba, 2014). For image preprocessing, we resize the images into (224,224,3) and add random flipping to the training set such that the model will be more resilient to rotated hands (Liu et. al., 2019).

Testing with our validation set, which contains 400×3 hand images (equally split between the two sources), the following results are obtained:

Underlying network	Accuracy	Speed/fps	Size/MB	Environment
Vanilla 5-block CNN	66.67%	124.15	334	NVIDIA Tesla V100 SXM2 32 GB
MobileNet	98.56%	131.36	39	
ResNet50	89.22%	123.06	283	
VGG16	66.67%	111.61	177	

Table 4: Performance statistics of our model with different backbones

Compare with the other networks, MobileNet achieves a great balance between performance and size. Besides, it avoids most of the practical concerns regarding deep learning on personal devices, such as local optima trap and lack of computational resources. As we observe from the training process, MobileNet converges much quicker and better than other networks.

On the other hand, we notice that although ResNet has a large model size and plenty of parameters, it is still able to converge unlike VGG and vanilla CNN. This is probably due to its Residual Learning mechanism, which attempts to tackle the degradation problem in deep neural network (He et. al., 2015). Noticing this fact, individual developers may also add Residual Learning in addition to Depthwise Separable Convolution in the future for more complex problems.

6 Online Performance

To illustrate our results, a web-based game is designed and published at https://pphp2019.do.am/index.html, which the interface is shown in Figure 3. The player can start a round by pressing the button in the middle, which will be followed by a countdown gif showing the remaining time on the top. When the countdown is finished, the camera will capture a photo (as shown on the left-hand side) and the result will appear in the bottom. However, we identify some problematic issues during the implementation of the game. These issues will be discussed in the following subsections.

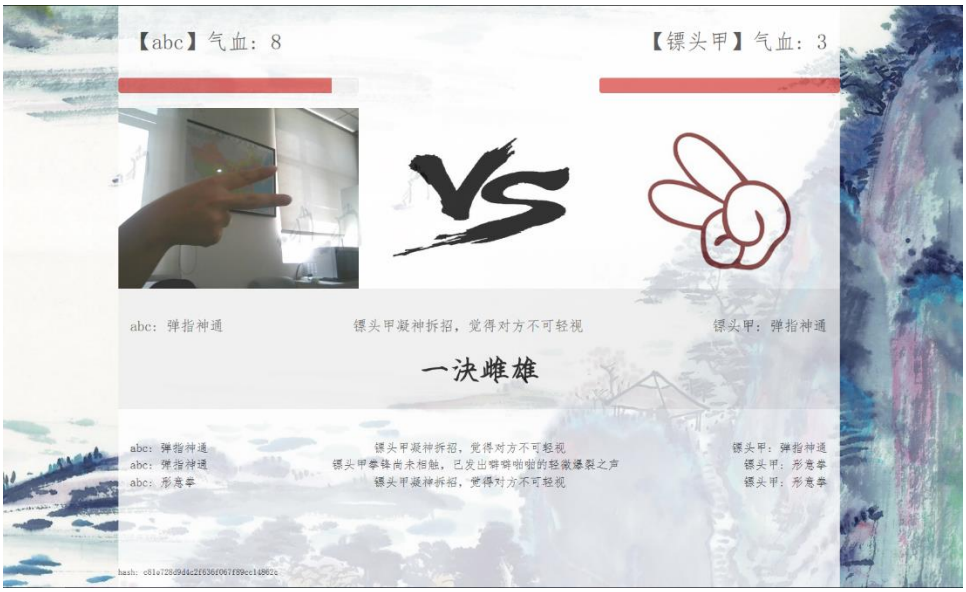

Figure 3: The UI design for the game

6.1 Loading Time

Though we foresee the model's loading time as a problem during the backbone selection process in Section 3 and make improvement through considering model size and complexity, this problem still exists depending on the user's bandwidth. To enhance user's experience, we design a loading gif which briefly explains the game's mechanism. After the model is downloaded for the first time, it will be cached in the player's PC and the loading time will be greatly reduced for the later access.

On the other hand, there is an obvious delay in the waiting time for the first prediction. After a code review, we hypothesize that such delay is due to the initialization process in the called function predict() in tensorflow.js. To reduce this undesirable effect, individual developers may try calling this function once before the game is started.

6.2 Integrity

Benefiting from the high computational power, the computer can compute the countermeasure to defeat a player in a time which human can hardly sense. In a PvE (player versus environment) situation, it is important to verify the existence of integrity, i.e. the computer does not change its original choice after knowing the choice made by the player in its flavor. To deal with this issue, we have applied two methods:

Firstly, we program the computer's decision to be purely generated by a random function from the JavaScript Math library. As we did not compress or encrypt the code for educational purpose, the player can verify that the variable storing the computer's decision is independent of the player's choice. In addition, the variable assignment is done at the beginning of each round and no reassignment is performed.

Secondly, we hash the computer's decision and show the hash value before the player makes their move. Since the hash value must be the same for the same message, the player can encode the received message (computer's decision after a round) again and compare the two hash values. If it aligns with the old hash value, then integrity is ensured. In this case, we import an md5 javascript to assist the hashing computation. When a round start, the computer's choice is encoded and the hash value will be displayed. After the round finish, the player can encode the computer's result with the same hashing algorithm to verify the validity of the result.

7 Conclusion

We compare the performance of different network architectures and, impressed by the high accuracy and low complexity of Depthwise Separable Convolution, we use MobileNet as the backbone. Experimental results demonstrate that MobileNet has a significant performance of 98.56% accuracy, which is considerable compared with the other network such as VGG16 and ResNet. For future development, we may add the residual learning mechanism and introduce a hand detection head to the network. We believe the accuracy can be further enhanced since the hand is located and the background is neglected during the prediction.

Reference

alish_manandhar. (2019). Rock Scissor Paper. Available: https://www.kaggle.com/alishmanandhar/rock-scissor-paper

Bruère-Terreault, J. (2019). Rock-Paper-Scissors Images. Available: https://www.kaggle.com/drgfreeman/rockpaperscissors

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2016). Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 38*(1), 142-158.

Girshick, R.B. (2015). Fast R-CNN. 2015 IEEE International Conference on Computer Vision (ICCV), 1440-1448.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. Available: https://www.deeplearningbook.org/

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., & Weyand, T., … Adam, H. (2017). Mobilenets: efficient convolutional neural networks for mobile vision applications.

keras-team. (2019). Keras Documentation. Available: https://keras.io/applications/

Kingma, D. P., & Ba, J. (2014). Adam: a method for stochastic optimization.

Krizhevsky, A., Sutskever, I., & Hinton, G. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM, 60*(6), 84-90.

Liu, D., Du, D., Zhang, L., Luo, T., Wu, Y., Huang, F., & Lyu, S. (2019). Scale Invariant Fully Convolutional Network: Detecting Hands Efficiently. Presented at AAAI Conference on Artificial Intelligence, Hawaii, 2019.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., & Berg, A.C. (2016). SSD: Single Shot MultiBox Detector. ECCV.

McMahan, B., & Ramage, D. (2017). Federated Learning: Collaborative Machine Learning without Centralized Training Data. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

Redmon, J., Divvala, S.K., Girshick, R.B., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788.

Ren, S., He, K., Girshick, R.B., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 39, 1137-1149.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

TensorFlow. (2019). TensorFlow.js. Available: https://www.tensorflow.org/js

transcranial. (2018). Keras.js. Available: https://github.com/transcranial/keras-js

Appendix

A1 MobileNet architecture (Howard et. al., 2017)

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$