Artificial Intelligence for Embedded Systems

Final Report

Fractures detection on X-Ray images using Machine Learning

Prepared by

Yu Zhu

Technical University of Munich

Winter 2018

Approved by

Yu Zhu

On

Jan. 31, 2019

1. Objectives and Expected Significance

1.1. Introduction: Assuming that a child is injured when playing, it is meaningful to detect immediately whether he/she gets a fracture. And sometimes, only those professional doctors could do right diagnosis. In this project, I try to run a machine learning model in Raspberry Pi and classify X-ray images to be normal or abnormal. So, with a well-trained model, every doctor could reach a high accuracy to diagnose with a portable device.

1.2. Objectives: Considering the limited hardware resources in Raspberry Pi, it requires to design a small but complete neural network. To realize this purpose, traditional structure is not feasible any more.

1.3. Significance: Normally, we expect to see professional doctors in the hospital. But those doctors in the remote areas are not professional enough and they could not afford expensive devices either. With the machine learning model in portable devices, they could diagnose accurately and efficiently.

2. Rationale and Related Background

2.1. Rationale: Classification of X-ray images by machine is possible. Besides, the hardware resources in Raspberry Pi is now enough to test or even train machine learning model.

2.2. MURA[1]: Nowadays, with the rapid development of neural network, more and more data sets are collected. MURA (musculoskeletal radiographs) is a large data set of bone X-rays. Algorithms are tasked with determining whether an X-ray study is normal or abnormal.

2.3. MobileNet[2]: Different structures of neural networks are put forward. Normally, we would repeat several layers to design a network, including convolution layer, B-N layer, activation layer. Pooling layer would be used either. Then, fully-connected layer would be used as the top layer to make classification with the function of 'softmax'. It is easy to see that the size of network is decided by the the number of parameters. To reduce the size of network, one simple way is to adjust the shape of convolution layer, using n*1*1 filter to replace n*n filter. Another way is to use global-average-pooling to replace fully-connected layer. So Google put forward two versions of MobileNet to be used in portable devices.

2.4. PlaidML+OpenCL: Commonly, using GPU could accelerate the process of training a machine learning model. Based on OpenCL, PlaidML provides the chance for the owner of AMD or Intel GPU to accelerate their models.

3. Approach

3.1. Comparison: To determine whether the result has been the best, it is meaningful to make comparisons. So, I use MobileNet[2], MobileNetV2[3] and MobileNet_Self_Design, three models in total and compare with the result shown in the paper[1].

	Radiologist 1	Radiologist 2	Radiologist 3	Model
Elbow	0.850 (0.830, 0.871)	0.710 (0.674, 0.745)	0.719 (0.685, 0.752)	0.710 (0.674, 0.745)
Finger	0.304 (0.249, 0.358)	0.403 (0.339, 0.467)	0.410 (0.358, 0.463)	0.389 (0.332, 0.446)
Forearm	0.796 (0.772, 0.821)	0.802 (0.779, 0.825)	0.798 (0.774, 0.822)	0.737 (0.707, 0.766)
Hand	0.661 (0.623, 0.698)	0.927 (0.917, 0.937)	0.789 (0.762, 0.815)	[illegible]
Humerus	0.867 (0.850, 0.883)	0.733 (0.703, 0.764)	0.933 (0.925, 0.942)	0.600 (0.558, 0.642)
Shoulder	0.864 (0.847, 0.881)	0.791 (0.765, 0.816)	0.864 (0.847, 0.881)	[illegible]
Wrist	0.791 (0.766, 0.817)	0.931 (0.922, 0.940)	0.931 (0.922, 0.940)	0.931 (0.922, 0.940)
Overall	0.731 (0.726, 0.735)	0.763 (0.759, 0.767)	0.778 (0.774, 0.782)	0.705 (0.700, 0.710)

3.2. Data Augmentation: Overfitting is a serious problem in the process of training neural network. With the function of 'flow_from_directory'[4], infinite images after augmentation could be generated.

3.3. Self Design: Keras provides two ways to build a model. One is ' import model' , the other is building with the function of ' Sequential'. To better understand the structure of MobileNet, I design the model according to the paper[3].

3.4. Update: To get a good result, adjustment of network parameters is necessary.

3.4.1. Normally, there would be some hyper parameters. Take 'MobileNet' for example. In the entry function, I could adjust the value of 'alpha' to control the width of network.

3.4.2. Besides, it is particularly important to adjust the value of learning rate if I use 'SGD' as the optimizer. 'Adam' is another more general optimizer. It would automatically adjust the value of learning rate.

3.4.3. Last but not least, even with the same structure, different values of epochs, steps and batch_size would result in different outcomes.

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

Table 2: MobileNetV2 : 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 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

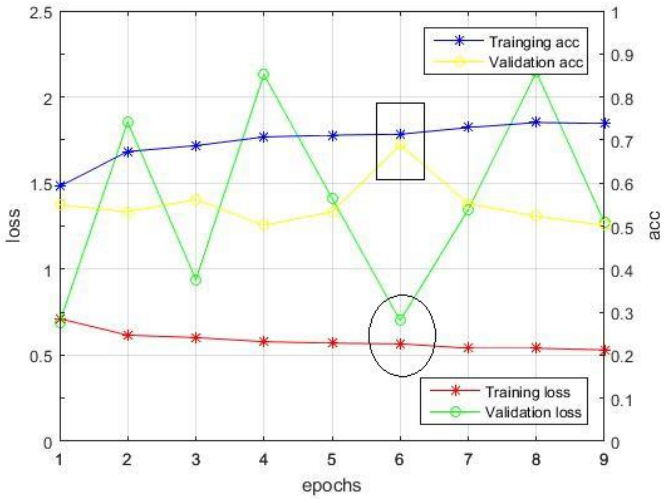
4. Project Outcome

4.1. Parameters Selected:

Model	MobileNet	Layers	157
Parameters	2, 227	Weights	26.2M
Training Time	2:27:19	Accuracy	68.9%

4.2. Performance Analysis:

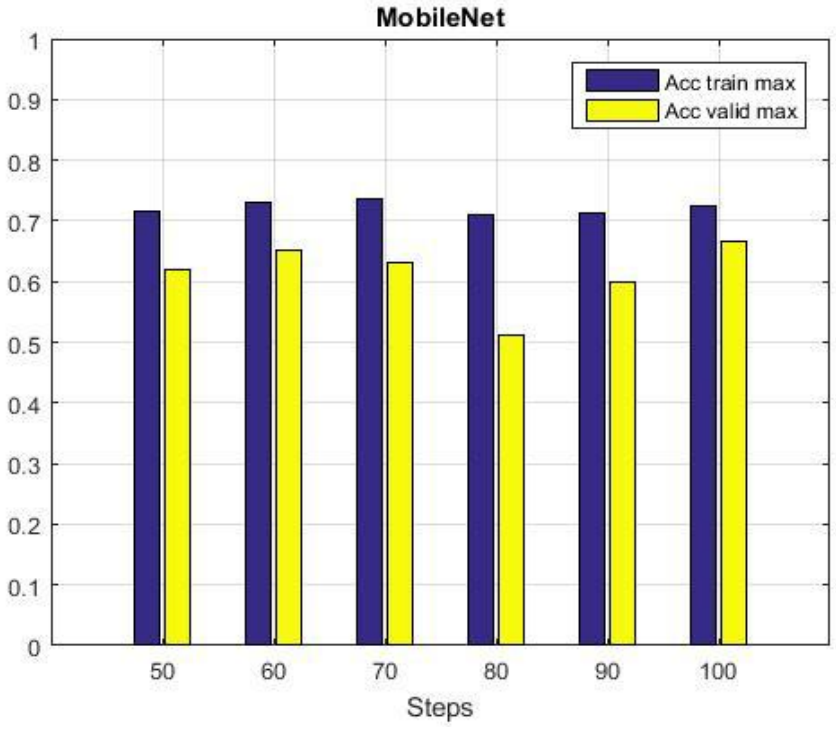
4.2.1. **Checkpoint**: It is shown on the right side that in the 6th epoch, the accuracy for the validation set reaches the best. As to why only 9 epochs are trained, the reason is that 'Keras' provides the parameter of 'Checkpoint' during training. For example, when the loss of validation do not improve for three times, the training process would end. It could greatly reduce the training time and get a good performance at the same time.



4.2.2. **Saturation**: It could be seen that the accuracy for training seems to be saturated at nearly 70% after the 4th epoch. That means the best result for validation set should be 70%. And it could be seen that the accuracy reaches 68.9% for validation set in 6th epoch. I could say it is a good result.

```
Conv_1 (Conv2D)                 (None, 7, 7, 1280)   409600      block_16_project_BN[0][0]
Conv_1_bn (BatchNormalization)  (None, 7, 7, 1280)   5120        Conv_1[0][0]
out_relu (ReLU)                 (None, 7, 7, 1280)   0           Conv_1_bn[0][0]
global_average_pooling2d_1 (Glo (None, 1280)         0           out_relu[0][0]
Logits (Dense)                  (None, 2)            2562        global_average_pooling2d_1[0][0]
==================================================================================================
Total params: 2,259,970
Trainable params: 2,225,858
Non-trainable params: 34,112
__________________________________________________________________________________________________
Layers: 157, parameters: 2259970
2019-01-28 10:22:25.613136: W tensorflow/core/framework/allocator.cc:113] Allocation of 154140672 exceeds 10% of system memory.
2019-01-28 10:22:26.037192: W tensorflow/core/framework/allocator.cc:113] Allocation of 154140672 exceeds 10% of system memory.
2019-01-28 10:22:26.037192: W tensorflow/core/framework/allocator.cc:113] Allocation of 154140672 exceeds 10% of system memory.
2019-01-28 10:22:27.537028: W tensorflow/core/framework/allocator.cc:113] Allocation of 156905472 exceeds 10% of system memory.
9/9 [==============================] - 180s 20s/step-01-28 10:22:44.960046: W tensorflow/core/framework/allocator.cc:113] Allocation
Accuracy:0.6890459361851426
(keras2-4) pi@raspberrypi:~/MobileNet/Raspberry_Pi $
```

4.2.3. **Comparison of Different Steps**: When the value of step to be 100, the result turns to be the best.



4.2.4. Comparison of Different Models: MobileNetV2 performs the best.

	Layers	Parameters	Weights
MobileNet	93	3230338	37.1M
MobileNetV2	157	2259970	26.2M
MobileNet_Self_Design	159	2278210	26.4M

4.2.5. Performance Limit: Nearly the problem of overfitting is solved, but the accuracy is not so satisfying. The scaling of pictures would account for this because some

features may be lost. (But whether to use 'rgb' or 'grayscale' seems to have little impact.) Of course, limited numbers of images is the core reason.

(512, 420, 3)

(224, 224, 1)

5. Unexpected Events

5.1. Network Structure: For the network designed by myself, its performance is not good enough. It does have a similar structure with the original 'MobileNetV2'. I guess one reason may lie in the different versions of 'Keras' I use to train. I need to get some intrinsic functions from the old library. And there should be some optimization for the latest version.

5.2. Hardware Limit: It is important for a model to be small enough to be applied in portable devices. For general case, it is preferable to train the model in a server and then use the same weights in portable devices. Because I train the model in my laptop and limited by its performance, I could not try as much optimization as possible.

5.3. Argument in Team: Now I think it could have been avoided.

6. Lessons Learned

6.1. Criterion: I run the model and I would get a result. How I could judge whether it has been the best result I could get? For one thing, I hope the accuracy could be as high as possible. So I would train the model with more epochs, hoping more features would be extracted. For another, with limited data set, it is easy to encounter the problem of overfitting. More epochs would result in a larger gap for the accuracy of training and validation. The ideal condition is that the loss is small enough and it would be nearly the same for both training and validation. So is the accuracy. For example, in the best result I get, the accuracy for training is about 70%, and for validation is 68.9%. It has reached a quite good condition, though I do want the accuracy to be higher.

6.2. Hardware: For neural network, there are some important factors, including the size of data set, the acceleration of hardware, the design of neural network. As I have said as above, acceleration of hardware could contribute a lot to build a better network.

6.3. Team Work: It is hard to judge because the criterion is quite different in university and in company. Normally, I hope the ideal condition that everyone could contribute something to the whole team.

7. Project Performance

7.1. On Time

7.1.1. Most milestones as follows are achieved on time.

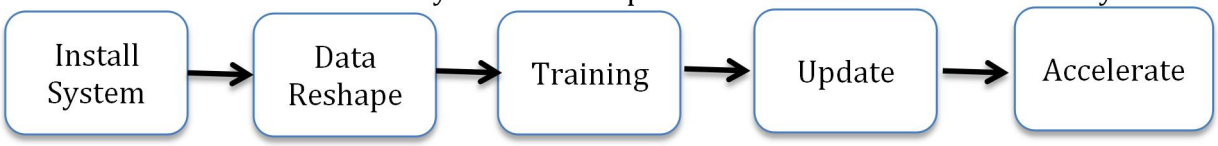
7.1.2. Install System: One day's work as expected**.** Install Raspbian for Raspberry Pi is easy. And with the latest version of Tensorflow, it is easy to install it in Raspberry Pi, with this command-- 'Pip3 install tensorflow'. There might be some problems of dependency, well solved in technical websites.

7.1.3. Data Reshape: One day's work as expected. The original structure of MURA is like this-- 'MURA/train/XR_HUMERUS/patient00051/study1_positive/image1.png'. What I hope is that all the images of XR_HUMERUS could be divided into two subdirectories, such as 'MURA/train/XR_HUMERUS/positive/...'. So all the images imported from the subdirectory 'positive' or 'negative' could be regarded as the same class.

7.1.4. Training: Work for a week as expected. During this process, I successfully train the model and verify in Raspberry Pi. I also design my original network structure and make comparisons.

7.1.5. Update: Work for a week as expected. Different data sets of the same network tend to have different best parameters. During this process, I update some external parameters to get better results.

7.1.6. Accelerate: Work beyond my expectation and poor performance. To use PlaidML, I need to install OpenCL. To install OpenCL, I need to install drivers for my GPU. During the process to install drivers for AMD, my system is broken. After my efforts, PlaidML run based on my GPU but the performance is even worse than my CPU.



7.2. On Budget:

7.2.1. **No extra device.** The equipment I use is my laptop. And it does work. When I verify the accuracy in Raspberry Pi, it is about 1 second per image. I think with an Intel Compute Stick to add memory, it could perform better. And the cloud platform -- Colab help me a lot to understand parallel computing.

7.3. Meeting Original Expectations:

7.3.1. **Realized:** I have got the best result for current network structure while it is still a bit low. To get a higher accuracy, a more flexible network is needed. Besides, more data is requested for a better performance. Limited by the hardware resources, I

choose XR_HUMERUS(~1,500images) to be my data set. Compared with XR_WRIST(~10,000images), this data set is a bit small.

7.3.2. **Not Realized:**In the original proposal, I hope to get original X-ray images from local hospitals to verify whether this model works well. I regret not being able to do this.

Reference:

[1]. Rajpurkar P, Irvin J, Bagul A, et al. Mura dataset: Towards radiologist-level abnormality detection in musculoskeletal radiographs[J]. arXiv preprint arXiv:1712.06957, 2017.
[2]. Howard A G, Zhu M, Chen B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861, 2017.
[3]. Sandler M, Howard A, Zhu M, et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks[J]. arXiv preprint arXiv:1801.04381, 2018.
[4]. Keras. https://keras.io/preprocessing/image/
[5]. PlaidML. https://github.com/plaidml/plaidml
[6]. MobileNetV2. https://github.com/xiaochus/MobileNetV2
[7]. MobileNet_Self_Design. https://github.com/Arith2/yuzhu_MURA_MobileNet