Pneumonia Detection with Deep Learning

Vinh-Thuyen Nguyen-Truong
Advanced Program in Computer Science
Faculty of Information Technology
University of Science, VNU-HCM
Email: ntvthuyen@apcs.vn

Xuan-Quang Nguyen-Diep
Advanced Program in Computer Science
Faculty of Information Technology
University of Science, VNU-HCM
Email: ndxquang@apcs.vn

Phuc Tran
Advanced Program in Computer Science
Faculty of Information Technology
University of Science, VNU-HCM
Email: tbphuc@apcs.vn

Trong Tran-Cong-Hoang
Advanced Program in Computer Science
Faculty of Information Technology
University of Science, VNU-HCM
Email: tchtrong@apcs.vn

***Abstract*—Pneumonia kills approximately 2400 children worldwide every day despite having simple and cheap cures. It can be detected by radiologists through chest x-ray photos, but access to experienced radiologists is not always available. Many approaches using neural networks to detect pneumonia from chest x-rays are proposed. However, these networks take a long time to train and are not very adaptive to new datasets. To improves this, the authors adopted Yuan Tian's approach and do experiment to train and measure the networks performance on a dataset on Kaggle. The authors also adjust several hyper-parameters, try to train the network with a smaller portion of the dataset, and extend the experiment with others network architectures. The authors achieve the best accuracy of 93.75% within approximately 40 minutes of training time. The reduction in training time allows the neural network to learn faster and use less computational resources.**

I. Introduction

Pneumonia is an infection that inflame lungs, causing patients to suffer from cough, fever, chills and trouble breathing. [1]. According to the UNICEF, pneumonia is one of the deadliest diseases among children [2]. In 2016, pneumonia is responsible for the death of 16% of children under the age of five worldwifacfade (approximately 880,000 children) [2]. This is because despite having easy, existing cures available, detecting pneumonia can be a problem. The inflammation can be detected by radiologists through chest x-ray photos. However, even when equipment is available, there is a shortage in experts who could interpret chest X-rays [3]. This raise the social needs of a system to automatically classify whether a patient is suffering from pneumonia given their chest X-ray, assist radiologists in giving accurate diagnosis.

There are several deep learning algorithms [3], [4] proposed to classify pneumonia from chest X-ray images. Rajpurkar et al. propose CheXNet [3], a convolutional neural network trained on ChestX-ray14 dataset, and achieved state-of-the-art result (76.8%) on that dataset. Abiyev et al. implement and compare the training time and performance of three types of neural networks [4] - Back-propagation neural networks (BPNNs) , Competitive neural networks (CpNNs) , Convolutional neural networks (CNNs) - on the same dataset, and confirm that CNNs has the best performance among them, despite the longer training time.

Despite the high accuracy of existing methods, they tend to take a lot of time and computer resource to learn new data. The authors main goal of the experiment is trying to reduce training time with the accuracy kept at an acceptable rate.

In order to achieve this, the authors made some modifications to Yuan Tian's implementation [5] (which is based on fast.ai [6] framework). The authors experiment with changing hyper-parameters (learning rate, algorithms for data augmentation, input image sizes, number of training cycles). The authors also experiment using a smaller portion of training datasets and with different network architectures.

The experiment is performed on Kaggle's "Chest X-Ray Images (Pneumonia)" dataset [7]. The best accuracy the authors achieved is up to 93.75% within approximately 40 minutes of training on the above dataset.

With such modifications, the authors' method can learn faster, take less computational resources, and is able to assist radiologists in giving accurate result.

In this paper, section 2 presents related work about applying deep learning in chest X-ray diagnostic. The explanation in detail of the authors' approach can be found in chapter 3. Chapter 4 describe the authors' process of fi. Chapter 5 describe the experiment conducted in the hyper-parameters, the portion of used training dataset and the network architecture. The conclusion is given in chapter 6.

II. Related Work

Chest x-ray8[8] and chest x-ray14[9] is released by Wang et al. NAIN's work uses tranfer learning with VGG16[10]. Rohit Verma uses InceptionV3 architeture for tranfer learning[11]. CheXNet, Classification and visualization using DenseNet-121-layers with chest x-ray14 dataset[3].

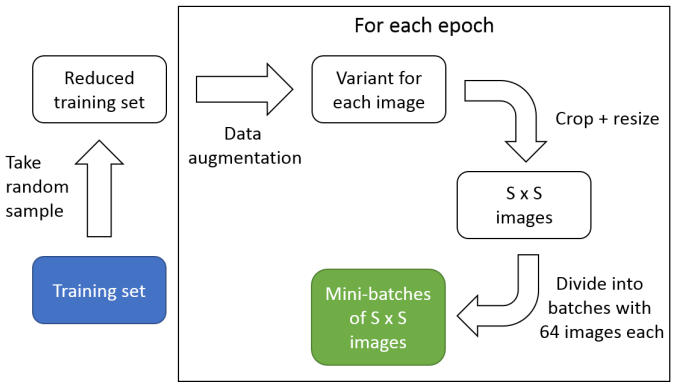


Fig. 1: Data preprocessing

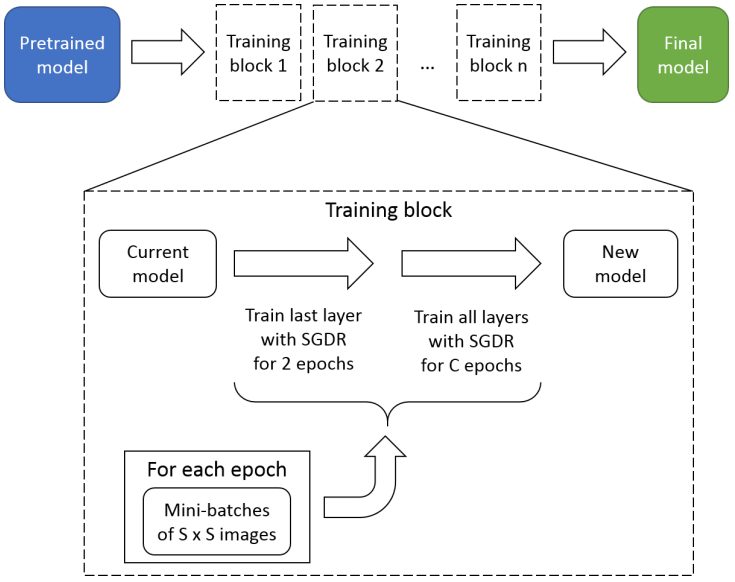


Fig. 2: Training process and detail of the training block

III. Method

A. Overview

The method consists of three main parts:

1) Data preprocessing: Create the reduced training set by taking a random sample from the training set, and perform data augmentation for each epoch in the training process (see section III-B for more detail).
2) Network training: Continue training a pre-trained model with the reduced training set to get the final model (see section III-C for more detail).
3) Giving prediction: Give prediction for an chest X-ray image with test-time augmentation (TTA) (see section III-D for more detail).

B. Data preprocessing

The data preprocessing is summarized with the diagram in Figure 1.

For each label, the authors consider all images having that label and take a random uniform sample with $X\%$ capacity (in V-C1, we experimented with various value for X from 50 to 100). All samples are then combined to form the reduced training set, which has $X\%$ capacity of the original training set. The ratio between the number of images of the two labels in the reduced training set is the same with that ratio in the original training set.

For each training block in the training process, the reduced training set is preprocessed. More specifically, to avoid overfitting, for each epoch, the authors use data augmentation algorithm A (an algorithm that change the perspective of the image by performing several random transformations) to get a slightly different variant of each image in the reduced training set. In V-C4, the authors experimented with two augmentation algorithms provided by the fast.ai framework:

- `transform_basic`: Slightly rotate the image; slightly change the constrast and balance of the image (used in Yuan Tian's implementation [5])
- `transform_side_on`: include the same modification with `transform_basic`, but also randomly flip the image horizontally.

These variants is then resized to S x S (where S is a parameter belong to the training block). These resized images is then divided into multiple mini-batches, each consist of 64 images. These mini-batches in used to train the model for that epoch.

C. Training process

The diagram in figure 2 give a overall view of the training process. The authors continue training a model with architecture N that is pretrained on the ImageNet dataset [12] (in V-C1, the authors experimented with two ResNet [13] architecture: ResNet34 - which is used in the Yuan Tian's implementation [5] - and ResNet18). The training process is a series of training blocks T consist of several training blocks, each having a parameter S (which decide the size of the images used in the training block).

In each training block, the model go under two training sub-processes:

1) Train only the last layer with Stochastic Gradient Descent with Restarts (SGDR) [14]) for 2 cycles, each cycle is 1 epoch long.
2) Train the entire network with SGDR for C cycles. The first cycle is 1 epoch long, and the length of the next cycle is two time longer than the previous one. Yuan Tian's implementation [5] used $C = 2$. In V-C3, the authors also experimented with $C = 3$).

In V-C2, the authors experimented with the following training blocks series:

- [64, 256]: 2 training blocks with S parameters of 64 and 256 respectively
- [128, 256]: 2 training blocks with S parameters of 128 and 256 respectively
- [64, 128, 256]: 3 training blocks with S parameters of 64, 128 and 256 respectively (this is used in Yuan Tian's implementation [5])

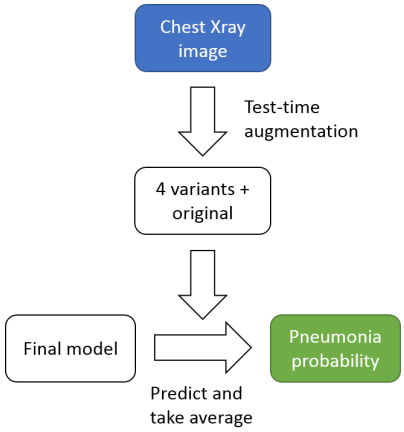


Fig. 3: Giving prediction with a completed model

D. Giving prediction

The final model is then used to classify pneumonia from chest X-ray. This process is summarized with the diagram in Figure 3.

To give prediction for an image, the authors perform test-time augmentation (TTA), which produce 4 different variants of the image (using the same algorithm with the one for data augmentation). Then, the model gives prediction for all of the images (including 4 variants and the original one), and then output their average as the final prediction.

IV. Process

The authors' process is summarized in Figure 4. The authors notice that there are methods that already have good accuracies. So we inherit the method from Yuan Tian [5] then focus on improving training time. First of all, the authors reduce the number of layers in ResNet architecture from 34 to 18. Following this, we reduce size of training set by $10\%, 20\%, 30\%, 40\%$ and 50% randomly from the original training set, then use them to train with both ResNet34 and ResNet18 architetures and observe the changes of training time.

V. Dataset and Experiments

A. Dataset

The authors use the "Chest X-Ray Images (Pneumonia)" dataset [7] on Kaggle for all experiments. The dataset consists of chest X-ray images with the following properties:

- Type of chest X-ray: anterior-posterior (AP)
- Patients' ages: from 1 to 5 years old
- Location: Guangzhou Women and Children's Medical Center, Guangzhou
- All the images are quality controlled (low quality images and unreadable scans are removed)
- For each image, the label is assigned by two experts physicians, and checked by a third one.

The number of images of each label (Normal or Pneumonia) in the dataset [7] are given in Table I.

TABLE I: Number of images in the data set

Label	Normal	Pneumonia
Training Set	1341	3875
Validation Set	8	8
Testing Set	234	390

B. Experiments Overview

1) Hardware specification: All experiments are conducted on Google Colab, which have the following hardware specifications:

- GPU: 1xTesla K80, compute 3.7, having 2496 CUDA cores, 12GB GDDR5 VRAM
- CPU: 1xsingle core hyper threaded Xeon Processors @2.3Ghz i.e (1 core, 2 threads)

2) Note on data preprocessing: In the data preprocessing step, for all value of hyper-parameter X that is considered in experiments, the reduced training set correspond to that value is prepared beforehand. In other word, all configuration with the same hyper-parameter X will use the same reduced training set.

3) Experiments measurement: Since the authors' method involve randomness (due to the algorithm for data augmentation and test-time augmentation), for each hyper-parameters configuration, the author run the method three times and record the running times and the accuracy for each time. The running time is measured as the time required to execute step 2 of the authors' method (i.e. the time elapsed from the beginning to the end of the training process). Both the average and range of running time and accuracy are then reported.

4) Hyper-parameters summary: In the experiments, the following hyper-parameters are concerned:

- N: Architecture of the network model
- X: Proportion of training set that is used in the reduced training set
- T: Training block series in the training process
- C: Number of training cycle in the second sub-process of each training block
- A: Algorithm for data augmentation and test-time augmentation

C. Conducted Experiments

1) Experiment 1: Network architecture and proportion of used training data

In this experiment, the author evaluate the running time and performance of the method on different network architectures with various setting of hyper-parameter X.

The authors experimented with two ResNet [13] architecture: ResNet34 (which is used in the Yuan Tian's implementation [5]) and ResNet18. For each architecture, the authors experimented with 6 different values for hyper-parameter X (50, 60, 70, 80, 90 and 100). All other hyper-parameters are kept the same as in Yuan Tian's implementation [5] ($C = 2$, $T = [64, 128, 256]$, $A =$ transform_side_on).

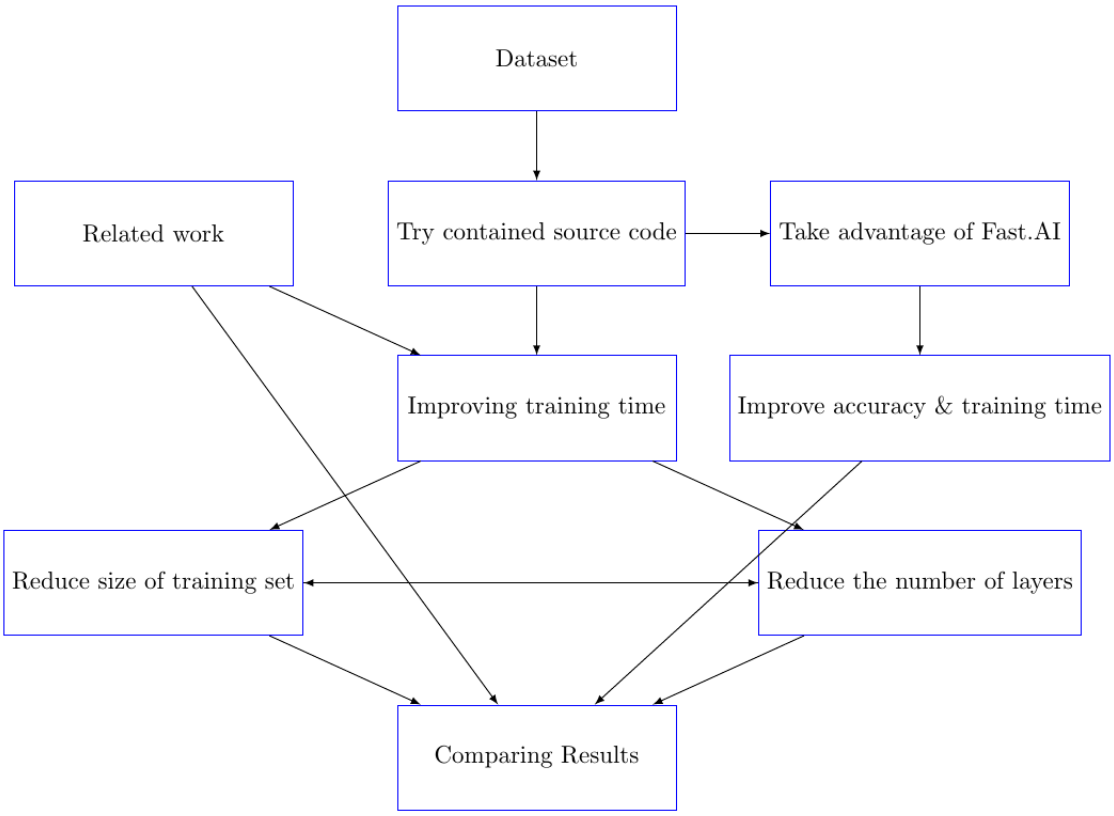


Fig. 4: Overview of the authors' process

TABLE II: Running time and accuracy with different architectures and proportion of used training set

Architecture	Used data (%)	Running time		Accuracy	
		Mean	Range	Mean	Range
ResNet18	50	1582	38	0.865	0.025
	60	1915	149	0.857	0.024
	70	2176	64	0.833	0.114
	80	2495	71	0.863	0.008
	90	2822	266	0.870	0.025
	100	3391	165	0.885	0.005
ResNet34	50	1629	119	0.858	0.029
	60	1910	208	0.862	0.023
	70	2431	316	0.841	0.063
	80	2725	129	0.871	0.019
	90	3003	255	0.874	0.029
	100	3476	549	0.887	0.027

The result is reported in table II and is visualized in Figure 5. For both network architecture, the average running time increase when X become larger (i.e when more data is used for training). However, the average accuracy does not always increase (e.g for ResNet18, the average accuracy drop from $X = 50$ to $X = 70$). When $X = 70$, the average accuracy is much lower compared to other X, and much more unstable (the range is 0.114 for ResNet18 and 0.063 for ResNet34) for both architecture. For each X, the difference in average running time and accuracy between the two architecture are not significant.

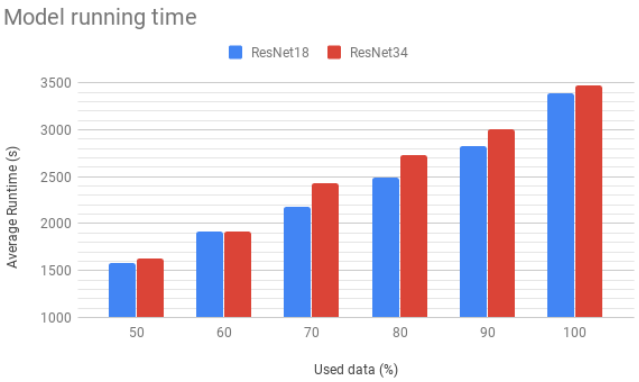
2) Experiment 2: Training blocks series

In this experiment, the authors evaluate the running time and performance of the method on Resnet34 architecture, with the following setting for training blocks series:

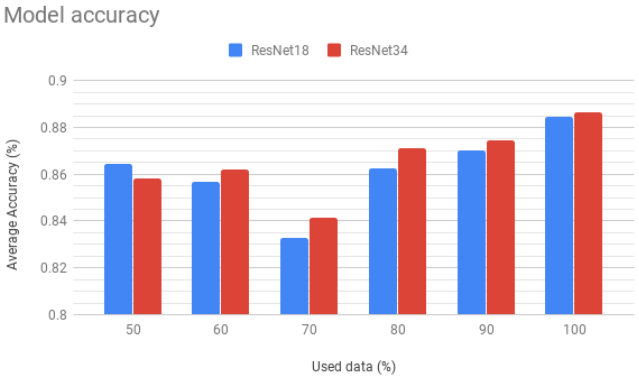
- [64, 256]: 2 training blocks with S parameters of 64 and 256 respectively
- [128, 256]: 2 training blocks with S parameters of 128 and 256 respectively
- [64, 128, 256]: 3 training blocks with S parameters of 64, 128 and 256 respectively (this is used in Yuan Tian's implementation [5])

For each training blocks series, the authors also do the experiment with hyper-parameter X of 50 and 100. All other hyper-parameters are kept the same with Yuan Tian's implementation [5] (N = ResNet34, C = 2, A = transform_side_on).

The result of the experiment is shown in Table III and is visualized in Figure 6. For both data proportion ($X = 50$ and $x = 100$), compared to $T = [64, 256]$, the average running time of $T = [128, 256]$ is slight longer, but the average accuracy is much higher. The average accuracy of

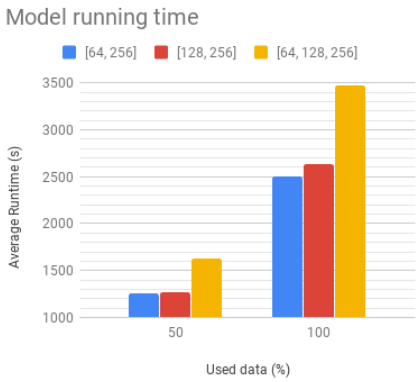


(a) Running time

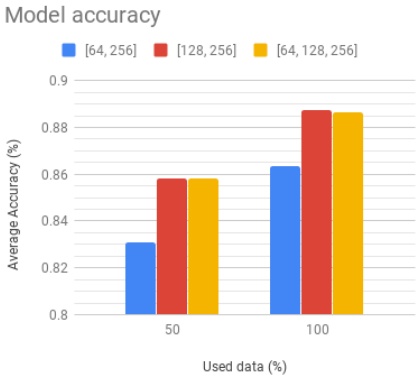


(b) Accuracy

Fig. 5: Average running time and accuracy with different different architectures and proportion of used training set



(a) Running time



(b) Accuracy

Fig. 6: Average running time and accuracy with different training blocks series

TABLE III: Running time and accuracy with different training blocks series

Training Blocks	Used data (%)	Running time		Accuracy	
		Mean	Range	Mean	Range
[64, 256]	50	1260	206	0.831	0.026
	100	2498	234	0.864	0.019
[128, 256]	50	1271	149	0.858	0.016
	100	2635	41	0.888	0.015
[64, 128, 256]	50	1629	119	0.858	0.029
	100	3476	549	0.887	0.027

$T = [128, 256]$ and $T = [64, 128, 256]$ are approximately the same, but the running time of the former is much shorter.

3) Experiment 3: Number of training cycles in the second sub-process for each training block

In this experiment, for each network architecture, the authors evaluate the running time and performance of the method with two different setting for C and X:

- $C = 3$, $X = 50$
- $C = 2$, $X = 100$ (this is used in Yuan Tian's implementation [5])

For other parameters, the authors used $T = [128, 256]$ (since it gives better result in Experiment 2). The augmentation algorithm A are kept the same as in Yuan Tian's implementation [5] (which use $A =$ transform_side_on).

TABLE IV: Running time and accuracy with different settings of X and C on two network architectures

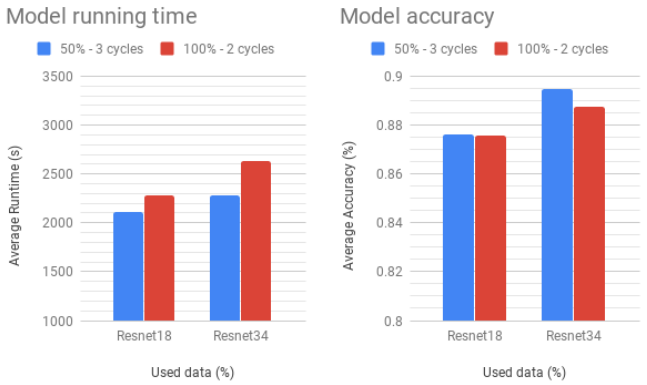
Used data (%) - Num of Cycle	Architecture	Running time		Accuracy	
		Mean	Range	Mean	Range
50% - 3 cycles	ResNet18	2114	183	0.876	0.005
	ResNet34	2276	295	0.895	0.002
100% - 2 cycles	ResNet18	2281	189	0.876	0.022
	ResNet34	2635	41	0.888	0.015

The result of the experiment is shown in Table IV and is visualized in Figure 7. (Comment on the result will be added later)

4) Experiment 4: Augmentation algorithm

The authors experimented with two augmentation algorithms provided by the fast.ai framework:

- `transform_basic`: Slightly rotate the image; slightly change the constrast and balance of the image (this is used in Yuan Tian's implementation [5])
- `transform_side_on`: include the same modification

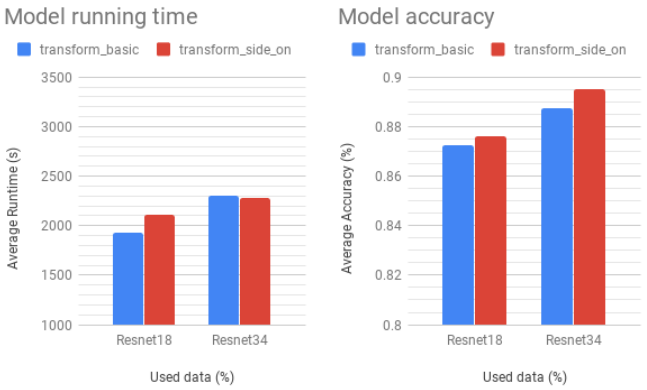


(a) Running time (b) Accuracy

Fig. 7: Average running time and accuracy with different settings of X and C on two network architectures

TABLE V: Training time and accuracy with different algorithm for data augmentation and TTA

Algorithm	Architecture	Running time		Accuracy	
		Mean	Range	Mean	Range
transform_basic	ResNet18	1931	36	0.872	0.009
	ResNet34	2299	194	0.888	0.024
transform_side_on	ResNet18	2114	183	0.876	0.005
	ResNet34	2276	295	0.895	0.002



(a) Running time (b) Accuracy

Fig. 8: Average running time and accuracy with different algorithms for data augmentation and TTA

with `transform_basic`, but also randomly flip the image horizontally.

For other parameters, the authors used $T = [128, 256]$ (the same with Experiment 2 and 3), $X = 50$ and $C = 3$ (which gives better result in Experiment 3).

The result of the experiment is shown in Table V and is visualized in Figure 7. (Comment on the result will be added later)

TABLE VI: Hyper-parameters setting of Yuan Tian's implementation and the author's implementation

Implementation	**Yuan Tian's**	**The authors'**
N	ResNet34	
X	100%	50%
T	T = [64, 128, 256]	T = [128, 256]
C	C = 2	C = 3
A	transform_side_on	

TABLE VII: Hyper-parameters setting of Yuan Tian's implementation and the author's implementation

	Yuan Tian's	**The authors'**
Average running time	2276	3391
Average accuracy	0.887	0.895

VI. Conclusion

Table VI compare the hyper-parameter setting, and and Table VII compare the running time and accuracy between the author's implementation and Yuan Tian's implementation. (Comment will be added later)

Acknowledgment

The authors would like to thank Dr. Yuan Tian for his original approach on the subject.

References

[1] A. L. Association, "Pneumonia symptoms and diagnosis," accessed: February 27, 2019. [Online]. Available: https://www.lung.org/lung-health-and-diseases/lung-disease-lookup/pneumonia/symptoms-and-diagnosis.html

[2] UNICEF, "Pneumonia claims the lives of the worlds most vulnerable children." [Online]. Available: https://data.unicef.org/topic/child-health/pneumonia

[3] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya, M. P. Lungren, and A. Y. Ng, "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning," *CoRR*, vol. abs/1711.05225, 2017. [Online]. Available: http://arxiv.org/abs/1711.05225

[4] R. H. Abiyev and M. K. S. Ma'aitahcorresponding, "Deep convolutional neural networks for chest diseases detection," *Journal of Healthcare Engineering*, vol. Volume 2018; 2018, 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6093039

[5] Y. Tian, "Detecting pneumonia with deep learning." [Online]. Available: https://medium.com/@yuan_tian/detecting-pneumonia-with-deep-learning-3cf49b640c14?fbclid=IwAR1W7PQ-vFS-Y40HyclH9AN9gWle_qFwCP5QF9-UdlDwekbLmiPIoc0ifnk

[6] J. Howard, R. Thomas, and S. Gugger, "Fast.ai." [Online]. Available: https://www.fast.ai/

[7] P. Mooney, "Chest x-ray images (pneumonia)." [Online]. Available: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

[8] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," *CoRR*, vol. abs/1705.02315, 2017. [Online]. Available: http://arxiv.org/abs/1705.02315

[9] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *2017 IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, 2017, pp. 3462–3471.

[10] N. D. D. I. NAIN, Deep Learning Engineer at OLA, "Beating everything with depthwise convolution." [Online]. Available: https://www.kaggle.com/aakashnain/beating-everything-with-depthwise-convolution

[11] S. a. I. I. o. I. T. Rohit Verma and M. P. I. Management, Gwalior, "Beating everything with depthwise convolution." [Online]. Available: https://github.com/deadskull7/Pneumonia-Diagnosis-using-XRays-96-percent-Recall

[12] P. U. Stanford Vision Lab, Stanford University, "Imagenet," accessed: March 09, 2019. [Online]. Available: http://www.image-net.org/

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[14] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with restarts," *CoRR*, vol. abs/1608.03983, 2016. [Online]. Available: http://arxiv.org/abs/1608.03983