The Methodologies Behind Computationally Efficient Convolutional Neural Networks

Quah Johnnie

School of Computing

Singapore Polytechnic

Singapore

johnnie.20@ichat.sp.edu.sg

***Abstract*—Over the years many technological advances have been made to convolution neural networks (CNNs), and accuracy of models have improved dramatically along with their complexity. However, what is not commonly noted is the computing power that comes along with more robust models. This paper presents the methods behind making computationally efficient CNNs by setting a computational limit when training models in this technical paper. A 100 Tera Floating Pointing Operations (TFLOPs) bottleneck is introduced to train all models in this technical paper. FLOPs is a measurement of computational power used to train a particular model. An in depth analysis will be presented on the models and architectures that provide the best results with this bottleneck. Model architectures benchmarked consist of VGGNets, EfficientNets, SqueezeNets, MobileNets, ShuffleNets and MnasNets.**

Keywords—Convolutional Neural Networks, CNN, Computationally Efficient Neural Network, Benchmark

I. Introduction

A. The Hidden Cost Behind Machine Learning

Building a machine learning model for corporate, research and industry uses come at many hidden costs and debt according to Google's machine learning research team. One of the main contributors to technical costs and debt is Model Complexity Erode Boundaries [1], where due to the complexity of the model costs cascade and accumulate exponentially when a minor correction is needed for a model.

B. What are FLOPs and FLOPS?

FLOPs and FLOPS are both commonly misunderstood. FLOPs is a short form term of Floating Point Operations, which is the number of Floating Point Operations done by device(s). FLOPS is a short form term of Floating Point Operations per second, which is the number of Floating Point Operations per second of device(s) and is commonly used as a benchmark for GPU and CPU speed. In this paper I explored and experimented with different efficient CNN models with the limit of 100 TFLOPs, which is equivalent to 100,000,000,000,000 Floating Point Operations.

C. Computational Trends Behind The ILSVRC Winners

ImageNet Large Scale Visual Recognition Challenge, ILSVRC, evaluates algorithms for image classification and object detection on a wide scale. The ILSVRC-2011, which is notably known as the last year before the rise of CNN, used an hybrid algorithm called XRCE, which employs high-dimensional image signatures with compression using product quantization, and one-vs-all linear SVMs, this produced the winning classification state-of-the-art entry in 2011 [2]. This implementation was computationally effective, the team spent most of their time refining its XRCE algorithms, while only needing to train their model on a single GPU.

Before long, AlexNet came, it was a defining moment for large-scale object recognition. AlexNet team, also known as SuperVision Team, was the clear victor of the classification and localization tasks in 2012. They used a clever hidden-unit dropout technique and an effective and efficient GPU implementation to train a sizable, deep convolutional neural network with 60 million parameters on RGB data [3]. Though it was computationally efficient even in today's standard, the amount of computational power needed to achieve this result was drastic compared to the ILSVRC winners in 2011. AlexNet was trained on 2x GTX580, a flagship NVIDIA graphics card during their time.

The trend of using more complex models with higher demand of computational power continues among ILSVRC winners, with exception of ResNet152 led by the team called the Great Learning Team. The team employed connection skippings with residuals in their CNN to overcome vanishing gradient problems in image classification, which is a defining moment in upscaling CNN. ResNet152 was presented and achieved state-of-the-art performances [4]. Although the number of layers were 8x more compared to previous state-of-the-art VGG16 used in ILSVRC-2014 [5], ResNet152 has a lower number of parameters and FLOPs compared to VGG16. From then on, the complexity and computational power across ILSVRC winners generally continue to increase with a few models that hover around similar parameters until EfficientNet is presented.

D. EfficientNets - First state-of-the-art but Efficient CNN

EfficientNet presents a new way of scaling called compound scaling. EfficientNets combine the ideas of width, depth and resolution scaling to form compound scaling. EfficientNet-B7, which is 8.4 times smaller and 6.1 times faster to train than the best existing CNN, delivers state-of-the-art 84.3% accuracy on ImageNet. Their EfficientNets model family also performs well, reaching state-of-the-art accuracy on the CIFAR-100 (91.7%) and Flowers (98.8%) in 2019 [6].

This state-of-the-art performance proves that getting an outstanding model does not always mean having a complex and computationally heavy model.

II. Related Works

Over the past few years, there has been a lot of research being done on how to make CNN models more resource-efficient.

A. *EfficientNet - Comping Scaling*

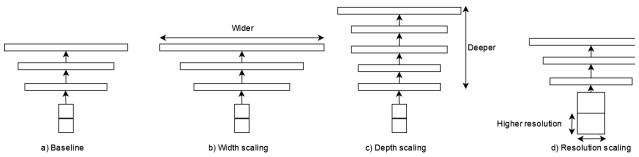

Fig. 1. Illustration of scaling methods

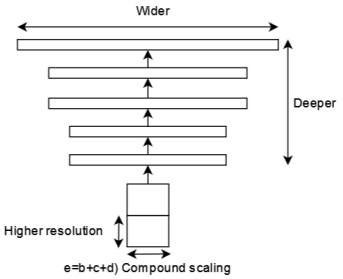

Fig. 2. Illustration of compound scaling

The EfficientNet original researchers show that different scaling dimensions are not independent [6]. Intuitively, we should increase network depth for higher resolution photos so that the larger receptive fields can aid in capturing similar features that comprise more pixels in larger images. To catch more fine-grained patterns with more pixels in high resolution photos, we should additionally widen the network when resolution is higher. These intuitions imply that rather than using traditional single-dimension scaling, we should coordinate and balance various scaling dimensions. Thus, forming compound scaling as shown in Fig 2, which is composed of scaling methods illustrated in Fig 1. Doing proper compound scaling can achieve similar results as a much more complex model with unbalanced dimensions.

B. *SqueezeNet - Efficient Model Design*

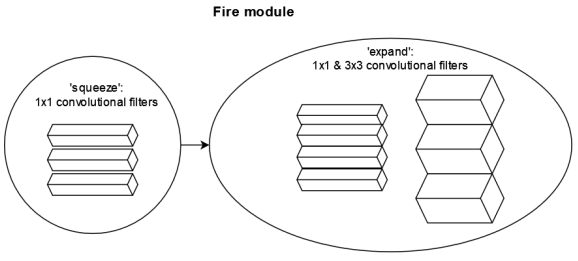

Fig. 3. Illustration of Fire module used in SqueezeNet (Only convolution filters are illustrated)

SqueezeNet was presented in 2016 [7], it was able to achieve results similar to AlexNet while being 50x smaller in parameters. SqueezeNet's architecture involves clever design strategies between accuracy and resource trade-offs by replacing 3x3 filters with 1x1 filters, 1x1 filters have 9x less parameters, they also decrease the number of input channels going into 3x3 filters, and they downsample later into the network instead of earlier, so that convolutional layers have larger activation maps. Fire modules are introduced which consist of 'squeeze' convolution layers, which comprises only 1x1 convolution filters, feeds into an 'expand' layer, which combines 1x1 and 3x3 convolution filters, as shown in Fig. 1. By using Fire modules, the trade-offs between using smaller filter size and accuracy becomes smaller. Hence, enabling them to obtain similar results to larger architectures while being extremely small.

C. *MobileNet - Depthwise Separable Convolution*

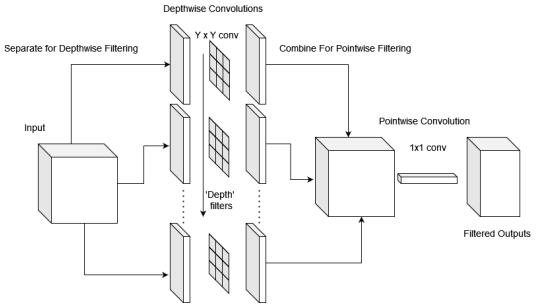

Fig. 4. Illustration of one depthwise separable convolution module

MobileNet was developed by a Google research team. The foundations of the MobileNet model are made up of efficient depthwise separable convolution [8] as shown in Fig 4, which is also used in previous state-of-the-art models, such as Xception [9] and ResNeXt [10]. Each input channel for MobileNets receives a single filter through depthwise convolution. The outputs of the depthwise convolution are combined using a 1x1 convolution after the pointwise convolution. A conventional convolution will filter and combine inputs to create a new set of outputs, in one step. However, in MobileNets, the depthwise separable convolution is divided into two layers by the depthwise separable convolution module: a layer for filtering and a layer for combing as shown in Fig 4. The computation and model size are significantly decreased as a result of this factorization. This configuration provides an excellent trade-off between accuracy and computational power.

D. *ShuffleNet - Channel Shuffle for Group Convolutions*

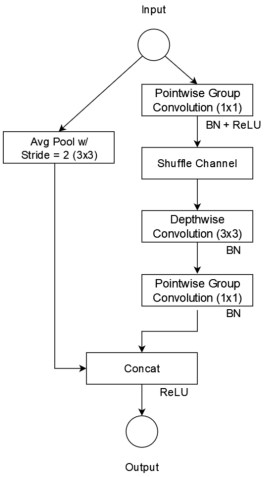

Fig. 5. Main ShuffleNet block

ShuffleNet also leverages off efficient depthwise separable convolution similarly to MobileNet. However, ShuffleNet is more efficient and different in a few ways [11]. ShuffleNet uses a different block configurations and utilizes channel sparse connections, such as group

convolutions, on 1x1 layers unlike MobileNet, but the main optimizer to ShuffleNet's efficiency is its used of channel shuffle operation, which is the equivalent to a module consisting a group convolutional layer of 1x1 followed by a random channel shuffle. Channel shuffle operation makes it possible to produce more effective and efficient structures when combining multiple group convolutional layers.

E. MnasNet - Squeeze-and-Excitation Blocks

MnasNet was developed by the same Google team that developed MobileNet. MnasNet shows improvement over ShuffleNet [12] with better efficiency and accuracy by achieving 3% higher Top-1 accuracy than ShuffleNet in ImageNet when using the same number of parameters. MnasNets make use of SE blocks, Squeeze-and-Excitate blocks, which improves the learning of convolutional features, by explicitly modeling channel interdependencies, this means the network can become more sensitive to informative features while suppressing less informative features [13]. Thus, giving rise to attention. The allocation of available computational resources towards a signal's most informative elements can be viewed as being biased by attention. MnasNet is also very efficient due to the carefully planned out architecture in terms of the number of convolution blocks, kernel size, layers to skip, layer per block, filter size and SE ratio to use and they achieve that by using what the team called Factorized Hiearchical Search Space.

F. Choice of Optimizer - Adam vs SGD

Adam is an efficient stochastic optimization technique that just needs first-order gradients and uses little memory. Using estimations of the first and second moments of the gradients, the approach calculates unique adaptive learning rates for various parameters. Adam is made to combine the benefits of two popular optimizers: RMSProp and AdaGrad, which perform well with sparse gradients [14]. Although Adam has presented itself as efficient, a few researchers and even the original implementer has pointed out that SGD generally lead to better generalization on testing data and a few method were used to close this gap, mainly the use of LAWN which uses a 3-phase learning rate schedule, compared to 2-phase on the original [15]. The difference between Adam and SGD can be summarized into 2 points. Adam provides solutions that generalize less well than those produced by SGD. Adam's test performance is inferior even when it achieves a training loss equal to or lower than SGD. Adam often displays faster initial progress on the training loss, but its performance quickly plateaus on the test error. Is there a limit to where the trade-off of using Adam is better than SGD, vice versa, especially when computational power is a bottleneck.

III. Discussion

A. Flow of Experiment

3 datasets of differing complexity, Kuzushiji-49 [16], Stanford-cars [17] and Food-101 [18], respectively in increasing complexity order, will be tested with 5 model architectures consisting of VGGNet, SqueezeNet, MobileNet, ShuffleNet and MnasNet. The goal is to find out the limitations and benchmark of each model architecture by setting a 100 TFLOPs limit for each model.

B. Training Procedure

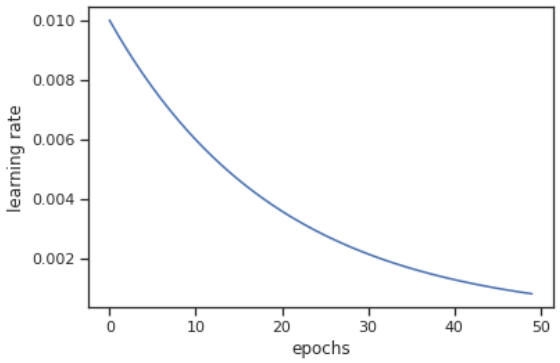

Fig. 6. Learning rate decay (scheduler) used for SGD optimizer

The datasets will be divided into train-dev-split and the best validation model, based on Top-1 accuracy, will be checkpointed and tested on the testing data, note that this means not all the training data will be used before evaluation on testing data, since the validation data is not being used during the training. 3 generic learning rates will be used 0.001, 0.0001, 0.00001, and 3 generic batch sizes of 32, 64 and 128 will be used, lastly 2 different optimizers SGD and Adam will be used as well. Different complexity of the model architecture will be evaluated. The main goal of this experiment is to find out the limitations of different model architectures and complexity by giving myself a 100 TFLOPs bottleneck. All datasets except Kuzushiji-49 will use AutoAugment with ImageNet policy as it is just a generic augmentation used for 1000 classes. The main metrics used to evaluate our model would be its test accuracy. A generic scheduler of 0.95^{epoch} , this is a rather higher learning rate decay. However, note that I am only limited to 10 TFLOPs, so it might make sense not to use a high learning rate decay in my experiment.

IV. Methodology

A. Dataset 1: Kuzushiji-49

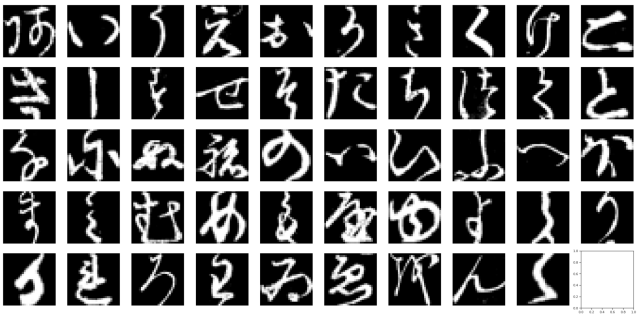

Fig. 7. All 49 classes of Kuzushiji

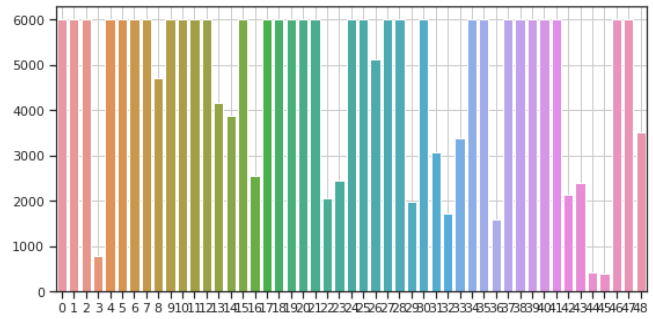

Fig. 8. Distribution of all Kuzushiji-49 classes from training data

Kuzushiji-49 consists of 232,365 training and 38,547 testing images. It contains 28x28 single-channel images of Japanese Hiragana characters and is an extension from the more notable 10 class Kuzushiji-MNIST, a replacement for the MNIST handwritten dataset. Kuzushiji-49 is an imbalanced dataset where 19/49 of its classes do not contain 6,000 data points; nonetheless, this should not affect the main metrics used. A good way to check how well a model generalizes well is to check the f1-scores of labels 3, 44 and 45 as these are characters with the least number of datapoints, sampling is often done to increase their f1-scores. However, this will not be done as it is not the main focus of this paper. This dataset will be our 'simple' complexity dataset used for my experiment.

B. *Dataset 2: Stanford Cars*

Fig. 9. All 196 classes of cars in stanford cars

Stanford cars was put together by an ex-stanford student in 2012, it is a balanced dataset that contains 8,144 training and 8,041 testing images with 196 classes consisting of cars [17]. Even though it has a lot more classes but less images compared to other datasets, the features of different cars are rather more distinct, making it a data set with decent complexity. However, due to the low number of training images and some cars appearing similar it is still a relatively difficult problem as well. This dataset will be our 'intermediate' complexity dataset used for my experiment.

C. *Dataset 3: Food101*

Fig. 10. 100/101 classes of Food101

Food101 is a balanced dataset containing 75,750 training and 25,250 testing images with 101 classes consisting of cars [18]. However, this is known to be complex as the different foods can look very different even if they are of the same classes. It is a great dataset to test covariate shift of a model, since some classes may contain very different looking food. This dataset will be our 'complex' complexity dataset used for my experiment.

D. *Estimating and Limiting FLOPs*

The estimation of FLOPs will be done by using pytorch 3rd party module *pthflops* [19], it is done by passing a random input of size *MxM* and batch size *N* into a CNN and the number of Floating Point Operations are calculated on every layer, and the number of FLOPs is returned for the input of size *MxM* and batch size *N* including an estimated FLOPs for back propagation. The number of GFLOPs for each epoch will be shown for every model I am evaluating in Table 1, while the maximum estimated number of epochs to train to reach the 100 TFLOPs limit will be shown in Table 2. EfficientNet was not designed to integrate single-channel input into compound scaling. Hence, EfficientNet is not evaluated on the Kuzushiji49 dataset.

TABLE I. NUMBER OF GFLOPS MADE PER EPOCH FOR EACH DATASET

Models	***Parameters***	***GFLOPs per epoch (Kuzushiji49/SF Cars/Food101)***
VGGNet5	3,852,740	4,710 / 10,445 / 110,074
EfficientNet_b0	5,288,548	NA / 3,311 / 30,797
SqueezeNet	823,044	474 / 2,359 / 21,943
MobileNetV2	2,474,948	1090 / 2,553 / 23,742
MobileNetV3_S	1,527,818	397 / 491 / 4,569
MobileNetV3_M	4,214,842	1,329 / 1,886 / 17,540
ShuffleNetV2	1,366,792	418 / 1329 / 12,340
MnasNet_S	950,322	391 / 936 / 8,711
MnasNet_M	3,114,546	1,150 / 2,722 / 25,317

TABLE II. THE NUMBER OF EPOCHS TO TRAIN FOR DIFFERENT MODELS.

Models To Evaluate	**Parameters**	**Epochs Able To Train *(Kuzushiji49/SF Cars/Food101)***
VGGNet5	3,852,740	21.2 / 9.6 / 0.9
EfficientNet_b0	5,288,548	NA / 30.2 / 3.2
SqueezeNet	823,044	210 / 42.4 / 4.5
MobileNetV2	2,474,948	91.5 / 39.1 / 4.2
MobileNetV3_S	1,527,818	252 / 203.6 / 21.9
MobileNetV3_M	4,214,842	75.6 / 53 / 5.7
ShuffleNetV2	1,366,792	238.9 / 75.2 / 8.1
MnasNet_S	950,322	255 / 106.8 / 11.5
MnasNet_M	3,114,546	86.9 / 36.7 / 3.9

Fig. 11. The number of epochs to train for different architecture and their respective dataset and model complexity.

E. *Hypothesis*

Since there are 3 datasets of differing complexity, Kuzushiji-49, Stanford-cars and Food101, I will be giving 3 hypotheses. For the Kuzushiji-49 dataset, I expect MnasNet_M to do the best, since most models do not have a bottleneck when they are limited to 100 TFLOPs, in fact, most models seem to have more than enough epochs to train on the Kuzushiji-49 dataset. Hence, the most complex model should observe better test accuracy. For the Stanford-cars dataset I expect MnasNet_S to perform the best as it is able to perform about 106 epochs, and is a relatively more complex model than MobileNetV3_S despite it having more parameters and being able to perform more epochs, as MnasNet_S is an attention base model. For the last dataset, Food101, I expect MobileNetV3_S to have the best test accuracy as it is able to perform significantly more epochs compared to other architectures and models.

F. Experiment Results

Legend for 'Best Parms' column: *LR = Learning Rate, BS= Batch size, Epochs = Highest Epoch of Best Validation Accuracy, Opt = Optimizer used*

TABLE III. TEST ACCURACIES FOR KUZUSHIJI49

Models To Evaluate	Best Parms *(LR/BS/Epochs/Opt)*	Test Accuracies (%)
VGGNet5	0.001/32/21/Adam	88.2%
EfficientNet_b0	NA	NA
SqueezeNet	0.0001/32/87/SGD	88.9%
MobileNetV2	0.0001/32/79/Adam	90.9%
MobileNetV3_S	0.00001/32/184/SGD	91.7%
MobileNetV3_M	0.0001/64/75/Adam	91.5%
ShuffleNetV2	**0.00001/32/173/SGD**	**93.2%**
MnasNet_S	0.0001/128/126/SGD	92.7%
MnasNet_M	0.0001/64/85/Adam	92.9%

TABLE IV. TEST ACCURACIES FOR STANFORD-CARS

Models To Evaluate	Best Parms *(LR/BS/Epochs/Opt)*	Test Accuracies (%)
VGGNet5	0.001/32/9/Adam	31.6%
EfficientNet_b0	0.001/64/29/Adam	64.8%
SqueezeNet	0.0001/64/42/Adam	54.2%
MobileNetV2	0.001/32/39/Adam	66.5%
MobileNetV3_S	**0.0001/64/184/SGD**	**71.8%**
MobileNetV3_M	0.001/64/51/Adam	68.6%
ShuffleNetV2	0.001/128/75/Adam	69.5%
MnasNet_S	0.0001/64/103/SGD	70.2%
MnasNet_M	0.001/64/36/Adam	66.7%

TABLE V. TEST ACCURACIES FOR FOOD101

Models To Evaluate	Best Parms *(LR/BS/Epochs/Opt)*	Test Accuracies (%)
VGGNet5	NA	NA
EfficientNet_b0	0.001/32/3/Adam	18.4%
SqueezeNet	0.001/32/4/Adam	22.5%
MobileNetV2	0.001/32/4/Adam	24.4%
MobileNetV3_S	**0.001/32/22/Adam**	**46.3%**
MobileNetV3_M	0.001/32/5/Adam	31.8%
ShuffleNetV2	0.001/32/8/Adam	35.6%
MnasNet_S	0.001/32/11/Adam	40.7%
MnasNet_M	0.001/32/4/Adam	28.5%

V. DISCUSSION

A. Revisiting Hypothesis

Seems the only hypothesis that was correct was MobileNetV3_S have the highest test accuracy for Food101.

For the Kuzushiji49 dataset, ShuffleNetV2 was able to leverage off a higher epochs level and lower learning rate that it managed to top the other models at 93.2%.

For the more complex stanford-cars dataset, MobileNetV3_S comes out on top as it is also able to leverage off a much higher epochs level than the rest at 184 and obtain an accuracy of 71.8%.

For the even more complex Food101 dataset, MobileNetV3_S comes out on top again as it is able to leverage off a much higher epochs level than the rest at 22 and obtain a test accuracy of 46.3%.

B. Bottleneck Trends

Bottleneck is heavily seen with VGGNet5, VGGNet was not designed for efficiency at all. e.g. On the simple Kuzushiji49 dataset VGGNet5 was able to still achieve a decent 88.2% with 22 epochs. However, this quickly falls off as the dataset becomes more complicated, in fact, the VGGNet5 cannot even perform a single epoch on the Food101 dataset. Models that are more efficient are able to perform more epochs with the 100 TFLOPs limit and generally do better.

C. Optimizer Trends

It seems that the simpler our dataset are, the more SGD optimizers are seen as their best optimizer to train with. This is likely because the SGD optimizer is able to generalize better than Adam at higher epoch levels. Most architectures that use SGD optimizer to obtain their highest accuracy are models that are able to train for a longer epoch due to its efficiency.

For my most complex dataset Food101. Only the learning rate 0.001, batch size 32 and Adam optimizer are used to obtain the models' best accuracies. This is due to the limited number of epochs each model can make, especially with the 100 TFLOPs limitation. Adam is seen as a more efficient optimizer than SGD here, and the highest learning rate of 0.001 and lowest batch size of 32 is seen as the best hyperparameters as it is the fastest the model can move towards the local minima for my experiment.

VI. CONCLUSION

A. Summary

To summarize, I have explored the architectures of 6 models VGGNet, EfficientNet, SqueezeNet, MobileNet, ShuffleNet and MnasNet. Much research has been done on all models and architecture. SqueezeNets, MobileNets, ShuffleNets and MnasNets are made to be computationally efficient architectures for mobile devices [7,8,11,12], and MnasNets' architecture has the best Top-1 ImageNet accuracy among these 4 architectures. However, when a bottleneck of 100 TFLOPs is introduced only MobileNet comes out on top, it is very likely that MnasNet is better than MobileNet but maybe only with a more complex dataset and higher bottleneck. Although EfficientNet has the highest Top-1 ImageNet accuracy among all the models, EfficientNet is not 'advertised' as a computationally efficient architecture, in fact, the 'Efficient' in EfficientNet actually means 'efficient scaling', as known as, compound scaling [6].

B. Rooms For Improvement

There is some room for improvement. However, due to the time limitation some analysis and experimentation is cut down.

It would be good to analyze the learning curves of the top 3 or 4 models of each dataset, this way we can intuitively see the rate at which the models' validation accuracies are increasing and where it cuts off, this can intuitively tell us which models are limited by its epoch level. e.g. EfficientNet may be learning at a faster rate than MobileNet. However, since it is limited by the number of epochs it could do, it fails to outperform MobileNet.

A wider range of model complexity, learning rate and optimizers could be used. Only 3 different learning rates, 3

different batch sizes and 2 different optimizers are used in this experiment, while weight decay or any form of regularization is not used. Perhaps, a wider range of these hyperparameters and model complexity can be used, this way more insights can surface from this experiment. Different numbers of FLOPs limitation could also be used along with more dataset examples as well.

An original implementation could also be done and compared to the other models. Perhaps, combining the idea of compound scaling from EfficientNet and the architecture of MnasNet, mainly its Squeeze-and-Excite blocks and depthwise convolutions. However, testing, tuning and finding the right number of SE blocks along with the 3 other scaling dimensions to scale will take much time with no guarantee that it is better than the current solutions we have. Furthermore, I am heavily penalized by the amount of available time left.

C. *Ending Note*

This technical paper is free of copyright, open-source and reusable. All diagrams are original by the author, Quah Johnnie, and made with draw.io [20]. However, this paper is **NOT** fact checked nor submitted by any institution or a legitimate academic personnel, this is purely for an assignment. Do **NOT** quote me on any figures or analysis done in this technical paper as it may appear unintentionally false. Thank you.

REFERENCES

[1] S. D. , G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, and D. Dennison, *Hidden Technical Debt in Machine Learning Systems*, vol. 2, pp. 2–3, Dec. 2015.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, vol. 3, pp. 18–20, 2014.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, pp. 1–2, 2012.

[4] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, vol. 1, pp. 1–4, 2015.

[5] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, vol. 1, pp. 1–2, Sep. 2014.

[6] M. Tan and Q. V. Le, *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*, vol. 1, pp. 1–2, May 2019.

[7] F. N. Iandola, K. Keutzer, W. J. Dally, K. Ashraf, M. W. Moskewicz, and S. Han, *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*, vol. 1, no. 4, pp. 3–4, Feb. 2016.

[8] A. G. Howard, H. Adam, M. Andreetto, T. Weyand, W. Wang, D. Kalenichenko, B. Chen, and M. Zhu, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, vol. 1, pp. 2–2, Apr. 2017.

[9] F. Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, vol. 1, no. 3, pp. 1–2, Oct. 2017.

[10] S. Xie, R. G. Girshick, P. Dollár, Z. Tu, and K. He, *Aggregated Residual Transformations for Deep Neural Networks*, vol. 1, no. 2, pp. 1–3, Nov. 2016.

[11] X. Zhang, X. Zhou, M. Lin, and J. Sun, *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*, vol. 1, no. 2, pp. 2–3, Jul. 2017.

[12] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, *MnasNet: Platform-Aware Neural Architecture Search for Mobile*, vol. 1, no. 3, pp. 3–6, Jul. 2018.

[13] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, *Squeeze-and-Excitation Networks*, vol. 1, no. 4, pp. 1–4, Sep. 2017.

[14] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, vol. 1, no. 9, pp. 2–4, Dec. 2014.

[15] A. Gupta , R. Ramanath, J. Shi, and S. S. Keerthi, *Adam vs. SGD: Closing the generalization gap on image classification*, vol. 1, no. 1, pp. 1–4, 2021.

[16] ROIS-CODH, "KMNIST Dataset," *Center for Open Data in the Humanities*. [Online]. Available: http://codh.rois.ac.jp/kmnist/index.html.en. [Accessed: 25-Nov-2022].

[17] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, *3D Object Representations for Fine-Grained Categorization*, vol. 1, no. 1, pp. 1–1, Dec. 2013.

[18] L. Bossard, M. Guillaumin, and L. V. Gool, *Food-101 – Mining Discriminative Components with Random Forests*, vol. 1, no. 1, pp. 1–1, 2014.

[19] A. Bulat, J. Kossaifi, and M. Monashev, *pytorch-estimate-flops*. [Online]. Available: https://github.com/1adrianb/pytorch-estimate-flops. [Accessed: 25-Nov-2022].

[20] Draw.io, "Draw.io Diagramming WebApp," Draw.io. [Online]. Available: https://app.diagrams.net/. [Accessed: 25-Nov-2022].