EfficientNet: A faster and effective image feature descriptor

E4040.2020Fall.EFNN.report
Valentine d'Hauteville vd2256, Vishnusai Yoganand vy2163

***Abstract*—Neural Networks are shown to effectively represent image features. However, most neural networks are over-parameterized. Most often, to increase the accuracy of the neural network towards a particular dataset, they are scaled only depth-wise. EfficientNet [1] was introduced by Google Brain is a neural network leveraging scaling across the dimensions of width, depth and image resolution. In this project, we discuss and demonstrate the capabilities of EfficientNet. More specifically, we implement EfficientNet-B0 and test its performance on a subset of the ImageNet dataset. Then, we conduct transfer learning experiments by choosing three publicly available datasets and compare EfficientNet-B0's performance with MobileNetV2. The results demonstrate that EfficientNet-B0 has comparable performance with MobileNetV2 but with significantly fewer FLOPS (floating point operations per second).**

***Index Terms*—EfficientNet, MobileNet, CNN, ImageNet**

I. Introduction

The advent of deep learning [2] has put forward a lot of applications in the space of computer vision. Examples include multi-class image classification [2]–[5], object detection [6], [7] semantic segmentation [8], etc. This has been possible because of the availability of large-scale datasets [9], [10] and high-speed computing [11].

Deep Learning is achieved using a huge set of tunable parameters organized in layers, where each layer is connected to another layer to establish effective information flow. Such an arrangement is a called Neural Network. The input data flows through each layer; at each layer the parameters extract features from the data; the final layer uses these features to perform meaningful predictions. A specific class of neural networks called Convolutional Neural Networks (CNN) is frequently used to perform predictions on image data (2D or 3D). Each layer of the CNN applies a 2D or 3D kernel on the input-feature set. The values of the kernel are then fine-tuned to the input data distribution through the process of back-propagation.

The advantages of CNNs over traditional neural networks are that they require significantly fewer parameters and take advantage of feature sharing over the entire spatial content of the image without compromising on performance. Ever since the introduction of AlexNet [2], neural network architectures have significantly improved, and we have seen a boost in output classification accuracy [3]–[5].

In this project, we explore a specific family of neural networks developed at Google Brain called EfficientNets [1]. The base network, EfficientNet-B0 was built by conducting a neural network search (NAS) [12] with MobileNet-v2 [13] as the base model. The other networks in the EfficientNet family were build by scaling up EfficientNet-B0's depth, width and resolution.

For our final project, the key objectives we focused on were to:

- Show that EfficientNets achieve SOA accuracy on the ImageNet dataset.
- Demonstrate that EfficientNet-B0's performance is comparable to MobileNetV2's but that the former model has lower latency.
- Corroborate that an ImageNet pre-trained EfficientNet-B0 model performs well on transfer learning tasks.

The rest of the report is organised as follows: Section II briefly discusses the recent work in the field of exploring different neural network architectures and delves into the key results presented by the official EfficientNet paper. Section III goes over our own objectives and roadblocks in implementing some of the paper's key insights. Section IV goes over our methodology and implementation to execute our project's tasks. Section V discusses the results obtained and Section VI concludes the paper.

II. Summary of the original paper

A. Literature survey and prior work

Because scaling up a ConvNet generally leads to better accuracy results, the field of Neural Network was for a while solely focused on building deeper networks and optimizing gradient flow. However, with deeper networks came an increasing number of parameters, computational costs and diminishing accuracy returns. More recently, more work was done to study how to reduce the number of parameters without incurring significant drops in accuracy. For instance, Han et al. [14] introduced deep-compression through a three stage pipeline i.e. pruning, quantization and huffman-coding. This technique reduces the parameters by 45x amount without reduction in accuracy on the ImageNet dataset.

Another line of works focused on building neural networks that were mobile deployable. MobileNets [15] used depthwise separable convolutions to reduce the number of parameters of a CNN. An improvement to this work was MobileNetV2 [13] which included inverted residuals and linear bottlenecks. Shortly after, ShuffleNet [16] included pointwise group convolution and channel shuffle, and managed to achieve the same accuracy as AlexNet with 13x fewer parameters. SqueezeNet [17] also achieved an accuracy comparable to AlexNet but with 50x fewer parameters.

With regards to model-scaling, prior work was done which focused on scaling either the width or the depth of the CNN. For instance, ResNet-50 [4] can be scaled down to ResNet-18 by decreasing the depth (#layers) of the network, or scaled up to ResNet-200 by increasing the depth. ResNet can be scaled in the width dimension [18] (WideResNet) and MobileNet can also be scaled using the same technique.

More recently, there has been a line of work making use of reinforcement learning strategies. He et. al [19] used DDPG algorithm to come up with a compressed neural network based out of VGG-16. Zoph et al [20] came up with NASNet where each convolutional layer is designed to perform well on CIFAR-10 [21] dataset.

B. Methodology of the original paper

The original paper introduces two keys results: a new family of models/architecture called EfficientNets that perform as well as much larger SOA models with fewer parameters, as well as compound scaling, a method to effectively scale a model in depth,width and resolution.

As the paper demonstrates, scaling a network in one dimension quickly leads to saturation in performance. For instance, the authors noted that ResNet-101 and ResNet-1000 have a very similar accuracy despite a huge difference in number of layers. On the other hand, they measured that scaling along depth, width (number of layers) as well as resolution led to less saturation and better accuracy for a given FLOPS budget compared to 1D scaling. Intuitively, it makes sense to scale both resolution and width along with depth, as increased resolution will enlarge the receptive field, while larger width entails more channels to extract features out of the input images will be used. In particular, increased width would enable deeper networks to better capture higher level features. To this end, the authors proposed a new scaling policy which scaled width, resolution and depth according to a constant "compound" ratio. In concrete terms, giving 2^N more computational resources to a network (measured in number of FLOPS) would entails scaling its depth by α^N, width by β^N, and size by γ^N. The figure below is a helpful visual to understand compound scaling.

More importantly, the authors supplemented these findings with the introduction of a new family of models called EfficientNets, which performed equally well as SOA models of the time but with much fewer parameters. First, they used the policy search procedure of NASNets to come up with the base neural network architecture called EfficientNet-B0. They then scaled this novel CNN architecture across the depth, width and height dimensions and formed 7 larger CNN architectures, EfficientNet-B1 to B7.

The code for their implementation can be found in this github repo.

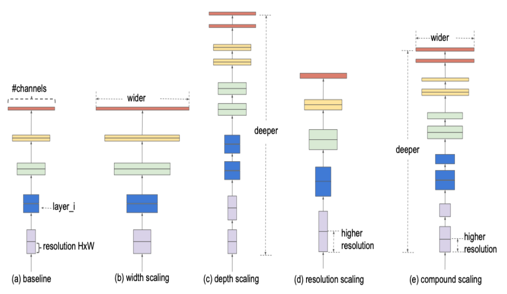
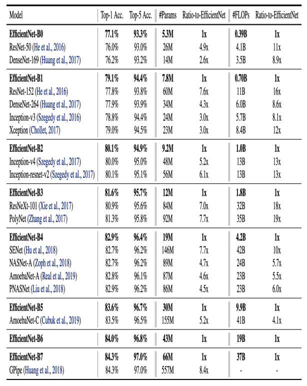


Fig. 1. Model Scaling.

C. Key Results of the original Paper

The original paper presents three main lines of results:

- State of the art performance of the EfficientNet models with fewer FLOPS and parameters compared to models with a similar accuracy. For example, EfficientNet-B7 achieved 84.3% top-1 accuracy on ImageNet, while being 8.4x smaller and 6.1x faster than GPipe the best existing ConvNet at the time. The accuracy findings are summarized in the following table (taken from [1])



Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.6%**	**95.7%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.9%**	**96.4%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.6%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.3%**	**97.0%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

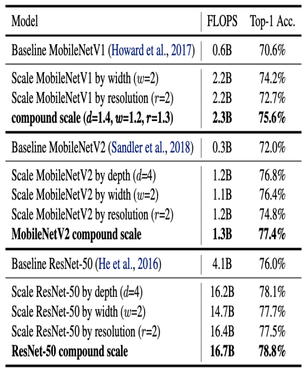
Fig. 2. EfficentNet Performance Results on ImageNet.

- An equally good performance of EfficientNets on transfer learning tasks, making them good feature extractors. The table snippet below (from [1]) shows that EfficientNet models obtain similar performance to other SOA models on many dataset with much fewer parameters.

	Comparison to best public-available results					
	Model	Acc.	#Param	Our Model	Acc.	#Param(ratio)
CIFAR-10	NASNet-A	98.0%	85M	EfficientNet-B0	98.1%	4M (21x)
CIFAR-100	NASNet-A	87.5%	85M	EfficientNet-B0	88.1%	4M (21x)
Birdsnap	Inception-v4	81.8%	41M	EfficientNet-B5	82.0%	28M (1.5x)
Stanford Cars	Inception-v4	93.4%	41M	EfficientNet-B3	93.6%	10M (4.1x)
Flowers	Inception-v4	98.5%	41M	EfficientNet-B5	98.5%	28M (1.5x)
FGVC Aircraft	Inception-v4	90.9%	41M	EfficientNet-B3	90.7%	10M (4.1x)
Oxford-IIIT Pets	ResNet-152	94.5%	58M	EfficientNet-B4	94.8%	17M (5.6x)
Food-101	Inception-v4	90.8%	41M	EfficientNet-B4	91.5%	17M (2.4x)
Geo-Mean						**(4.7x)**

Fig. 3. EfficientNet Performance Results on Transfer Learning Datasets.

- The effectiveness of the compound scaling method. The authors show that scaling a given model using the compound method systematically achieves higher top-1 accuracy on ImageNet. For instance the figure below shows some results they obtained for MobileNet, MobileNetV2 and ResNet-50.



Model	FLOPS	Top-1 Acc.
Baseline MobileNetV1 (Howard et al., 2017)	0.6B	70.6%
Scale MobileNetV1 by width (w=2)	2.2B	74.2%
Scale MobileNetV1 by resolution (r=2)	2.2B	72.7%
compound scale (d=1.4, w=1.2, r=1.3)	**2.3B**	**75.6%**
Baseline MobileNetV2 (Sandler et al., 2018)	0.3B	72.0%
Scale MobileNetV2 by depth (d=4)	1.2B	76.8%
Scale MobileNetV2 by width (w=2)	1.1B	76.4%
Scale MobileNetV2 by resolution (r=2)	1.2B	74.8%
MobileNetV2 compound scale	**1.3B**	**77.4%**
Baseline ResNet-50 (He et al., 2016)	4.1B	76.0%
Scale ResNet-50 by depth (d=4)	16.2B	78.1%
Scale ResNet-50 by width (w=2)	14.7B	77.7%
Scale ResNet-50 by resolution (r=2)	16.4B	77.5%
ResNet-50 compound scale	**16.7B**	**78.8%**

Fig. 4. Scaling Up MobileNets and ResNet.

III. Methodology (of the Student's Project)

A. *Objectives and Technical Challenges*

In a world without constraints, our objective for this project would have been to reproduce some findings for each line of results mentioned above. However, this was not possible given our computing and time constraints, as well as some roadblocks we encountered. Below we detail some of the obstacles we faced.

- Most of the accuracy results of the paper are for the ImageNet dataset. However, training a model such as EfficientNet on the whole ImageNet dataset would have required many GPUs running in parallel for a number of days and was thus out of the question given our current computing budget.
- We attempted to train EfficientNet from scratch on smaller datasets (Stanford Dogs, CIFAR-10) but got poor results and over-fitted the training set. Indeed, EfficientNet is actually overparametrized compared to such small datasets. Even this official keras documentation page cautions against training EfficientNet from scratch on a small dataset.
- Implementing compound scaling or at least confirming that 2D/3D scaling is better than 1D scaling proved to be too complicated. Initially, we thought that loading a Keras pre-trained model, increasing its resolution (by changing the input layer) and depth (by adding a couple layers) and then fine-tuning the obtained model would do the trick. In doing so we hoped to show that scaling by depth and resolution was better than scaling just by depth. However, we realized that scaling by depth actually required scaling each block of layers (where each block is a sequence of layers that repeats), as opposed to adding layers at the end. Thus, properly scaling by depth (or width) entailed training a new model from scratch. Given our time constraints, we decided to not go through with it.

As our final objective, we thus decided to implement the following:

- Build EfficientNet from scratch and confirm its performance on the ImageNet validation dataset.
- Show that EfficientNet outperforms other models in transfer learning tasks

B. *Problem Formulation and Design Description*

We organized our github repository as follows:

- A utils library, EfficientNet_model.py containing the code to build the EfficientNet model from scratch. While writing up the architecture, we referred to the official tensorflow implementation The utils folder also contains *efficientnet_weight_update_util.py*, a script we copied from the the official Keras source code and which we use to convert a checkpoint file into an h5 file.
- A notebook which instantiates an EfficientNet-B0 using the library mentionned above, loads it with checkpoint ImageNet weights taken from the internet and evaluates it on a test ImageNet dataset.
- A series of notebooks comparing the performance of EfficientNet-B0 and MobileNetV2 on other datasets: stanford dogs, Cats versus Dogs and CIFAR-10. We compare accuracy, as well as FLOPS (see this notebook). We used MobileNetV2 as a comparator because it is built on inverted bottleneck convolutions, which EfficientNet-B0 also uses as building blocks. EfficientNet-B0 is a bit smaller in size than MobileNetV2 but was designed to perform as least as well.

To implement the notebooks mentioned above, we also had to source ImageNet weights for our EfficientNet model, source the ImageNet test dataset used to evaluate our model, and decide which datasets to use for our transfer learning task.

- We identified three sources from which to download ImageNet weights: the official repository for the paper, the Keras source code and a link put up by a (well-followed) individual contributor in this repository. We tried weights from all three sources and found that they yielded similar performance.
- The official method to obtain the ImageNet test dataset we would have been through the ImageNet website. However, Keras also provides links to repositories containing more accessible test sets created with subsets of ImageNet. We ended up settling for this dataset called ImageNetV2. According to Keras ”ImageNet-v2 is an ImageNet test set [...] collected by closely following the original labelling protocol..”
- For the transfer learning task, we used Cats vs Dogs [22], Stanford Dogs [23] and CIFAR-10 [21] because they as

easily loadable, relatively small and straightforward to train, with a resolution $\leq$ 160 x 160 (where 160 x 160 is the default resolution of MobileNetV2).

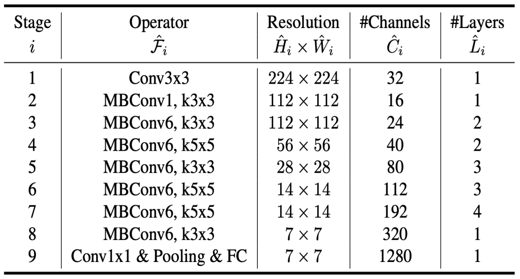
IV. Implementation

A. Deep Learning Network

Below we give the details of the EfficientNet-B0 architecture. The fundamental building block of EfficientNet-B0 is the mobile inverted bottleneck architecture [13] to which the squeeze-and-excitation optimization is added. The inverted residual structure expands the representation of the feature space by increasing the number of channels before compressing it down again. The residual structure is comprised of an input convolution layer which expands the feature representation space, followed by a depth-wise convolution layer. The last layer, which is a regular convolution layer compresses the feature representation space. The residual output of the mobile inverted convolution layer is fed through the squeeze and excitation optimization layer to assign weights to each channels. The activation function of all the layers is ReLU except for the output of the squeeze and excitation layer which uses the Swish activation function defined by the equation 1

$$Swish(x) = x * sigmoid(x) \tag{1}$$

The architecture is summarized in this figure from [1]



Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

Fig. 5. Model Scaling.

B. Software design

The transfer learning task involves fine-tuning a pre-trained EfficientNet model to a dataset of our interest. [1] stated that the imagenet pre-trained EfficientNet can be fine-tuned and effectively used for smaller datasets. To demonstrate this, we train EfficientNet-B0 on three publicly available datasets:

- Cats vs Dogs [22]
- Stanford Dogs [23]
- CIFAR-10 [21]

The dataset was made available, thanks to Tensorflow [24]. The dataset was rescaled to 160 x 160 and mean-normalised. The procedure we followed for transfer learning is:

- Freeze the pre-trained weights and train only on the top layers for the initial 10 epochs.
- Un-freeze the pre-trained weights except for the first 100 layers and train again for 10 epochs.

We present the results in the next section. We also perform comparisons by performing similar experiments with MobileNetV2.

As mentioned in Section III, we used an ImageNet test dataset of 20,000 images taken from the internet to test the performance of our EfficientNet-B0 implementation. Loading the dataset using the method suggested by the tensorflow/datasets documentation returned a *NonMatchingCheckSumError* (see documentation) so we had to load the dataset manually using tensorflow's *utils.get_file()* and *image_dataset_from_directory()* functions. This returned a generator of test images which we used as argument in our call to *model.evaluate()*.
We also loaded two EfficientNet-B0 models from the Keras library to use a controls. We instantiated the first model with the argument *weights= "imagenet"*. Thus in the ideal case, we expected this model to return a prediction accuracy similar to the one reported in the official [1] paper. Furthermore, we did not instantiate the weights of the second model (ie: we set *weights=None* in the arguments to the constructor). Instead, we provided it with the same checkpoint weights as the ones used in our implementation. We expected this model's performance to match our implementation's.

V. Results

A. Project Results

As discussed in the previous section, we trained EfficientNet-B0 and MobileNet-V2 on three datasets. We also made use of the following data-augmentation techniques: horizontal random flip and rotation. The obtained accuracy results are shown in Table I.

TABLE I
Performance comparison between EfficientNet-B0 and MobileNetV2

Dataset	EfficientNet-B0	MobileNetV2
Cats vs Dogs	98.39%	97.52%
Stanford Dogs	70.7%	71.4%
CIFAR-10	83.03%	85.19%

The accuracy values obtained are on the test dataset of the individual dataset. We also conducted an experiment to calculate the FLOPS of the two models. For an image size of 160 x 160, we obtained

- MobileNetV2: 589,844,034
- EfficientNet-B0: 411,886,635

The FLOPS difference between MobileNetV2 and EfficientNet-B0 is 17.7957399 * 10e6. The results in Table I suggest that the performance between MobileNetV2 and EfficientNet-B0 is comparable, even though EfficientNet-B0 has lesser FLOPS than MobileNetV2.

Figure 6 shows an example of the training history upon implementing transfer learning technique on the Cats vs Dogs dataset.

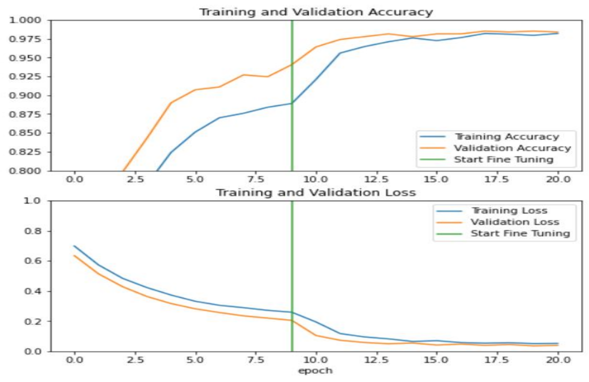


Fig. 6. Training history for Cats vs Dogs on MobileNetV2

The results obtained from building EfficientNet-B0 from scratch and evaluating against the ImageNetV2 test dataset are summarized in the Table II below. The reference Keras model with pre-trained ImageNet weights only achieved a top-1 accuracy of 59.7%, which is 17.4% points below the results reported by the official paper. Our EfficientNet-B0 model achieved results similar to the Keras Model with checkpoint weights, indicating that our implementation of the model architecture was correct. However, the accuracy reported for these two models was about 20% points under the performance of the reference Keras model.

Note: During development, we also tried loading checkpoint weights into an EfficientNet-B1 model created using our library but got terrible accuracy results (ie: results corresponding to random weights). Moreover, the Keras EfficientNet-B1 model loaded with the same checkpoint weights gave equally terrible accuracy values, so we hypothesize that there might have been some underlying discrepancy between the models weights and the checkpoint weights. Because of these bad results, we did not include the EfficentNet-B1 code in our final notebook.

TABLE II
EFFICIENTNET PERFORMANCE RESULTS ON IMAGENETV2 VALIDATION SET

Model	**Top-1 Acc**	**Top-5 Acc**
EfficientNet-B0 official paper	77.1%	93.3%
EfficientNet-B0 Keras reference	59.7%	82.1%
EfficientNet-B0 Keras ckpt weights	39.9%	64.4%
EfficientNet-B0 self ckpt weights	39.6%	63.5%

B. Comparison of the Results between the Original Paper and the Student's Project

The transfer learning of EfficientNet-B0 on CIFAR-10 in the original paper [1] yielded an accuracy of 98.1%. However, we were only able to produce an accuracy of 83.03%.

With regards to evaluating our EfficientNet model on the ImageNetV2 test dataset, we were also not able to reproduce the results of the paper. Indeed, as noted earlier, the reference Keras model with pre-trained ImageNet weights only achieved a top-1 accuracy of 59.7%, that is 17.4% points under the 77.1% accuracy results reported by the official paper. Furthermore, our model with checkpoints weights, as well as Keras' model both gave results around 39.7%, which is 20% points lower than the reference Keras model and 37.4% lower than the results of the paper.

C. Discussion of Insights gained

This huge difference in performance observed in the transfer learning task can be attributed to factors like hyper-parameter tuning, test-time augmentation, etc. Also we were limited by compute capability and time, prohibiting us from conducting hyper-parameter search and testing on bigger EfficientNet models.

We do not really know the reasons behind the lower ImageNet validation accuracy values we measured, and time constraints prevented us from fully investigating. Even the reference model used (straight from keras, with pre-loaded imagenet weights) gave results much lower than the official paper's, which was surprising. However, we can hypothesis the following:

- The test set we used was slightly different compared to the true ImageNet validation set and also smaller thus prone to more noise. According this source the ImageSet validation set contains 50,000 images, while our own test set only contains 20,000.
- The authors of the paper could have gone through further hyper-parameter tuning or other test time augmentation techniques to improve their results.

VI. Conclusion

The project explored a specific convolutional neural network architecture called EfficientNet. We discussed the advantages of EfficientNet when compared to other standard neural networks. We implemented EfficientNet-B0 and tested it against a subset of the ImageNet dataset. The reasons for obtaining low-accuracy as compared to results posted in the paper has been reasoned out. We also conducted transfer learning experiments on three datasets and demonstrated the performance of the pre-trained EfficientNet-B0, also comparing with MobileNetV2. The project culminated by demonstrating that EfficientNet-B0 was comparable to MobileNetV2 but had significantly lower FLOPS. Future work in this direction could be pointed towards conducting hyper-parameter search to increase the performance accuracy and towards implementing bigger EfficientNet models.

VII. Acknowledgement

Thanks to TAs Desmond Yao and Vedant Dave for answering our questions and suggesting ideas as to how to go about implementing this paper and getting around our GPU and time constraints. In particular, Desmond suggested the approach of loading checkpoints weights in order to evaluate our own

implementation of EfficientNet. Vedant gave some hints about how to go about finding an alternative ImageNet validation set.

A. *Individual student contributions in Fractions*

	vd2256	vy2163
Last Name	d'Hauteville	Yoganand
Fraction of (useful) total contribution	1/2	1/2
What I did	EfficientNet implementation and evaluation against the ImageNet test dataset	Transfer Learning for EfficientNet-B0 and MobileNet-V2

References

[1] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[6] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *NIPS*, pages 91–99, 2015.

[7] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[10] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[11] David Kirk. Nvidia cuda software and gpu parallel computing architecture. In *Proceedings of the 6th International Symposium on Memory Management*, ISMM '07, page 103–104, New York, NY, USA, 2007. Association for Computing Machinery.

[12] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.

[13] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[14] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[16] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices, 2017.

[17] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$1mb model size. *CoRR*, abs/1602.07360, 2016.

[18] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.

[19] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices, 2019.

[20] Xu Qin and Zhilin Wang. Nasnet: A neuron attention stage-by-stage net for single image deraining, 2020.

[21] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

[22] Jeremy Elson, John (JD) Douceur, Jon Howell, and Jared Saul. Asirra: A captcha that exploits interest-aligned manual image categorization. In *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, Inc., October 2007.

[23] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.

[24] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.