Improving Fire Detection with Efficient Deep Learning Training Techniques

Ricardo Gonzalez, 2003297

Abstract—Deep Learning and Convolutional Neural Networks are currently a popular topic of research, leading to works that attempt to build models that can be used in the world, instead of just being used in research. Nevertheless, there are works that focus on this but don't achieve a good performance, due to the use of inefficient architectures or not applying techniques to improve the training and reduce the resources used. In this work we will increase the accuracy of our baseline model, with the use of newer architectures and by the use of new techniques such as Mixed-Precision training, Mixup and Transfer Learning. The result of this work were four new models that not only trained in fewer amount of epochs, but also achieve higher accuracy than our baseline.

Index Terms—CNN; Deep Learning; Image Classification; Efficient Training

I. Introduction

Nowadays due to the advancements in Artificial Intelligence (AI) and the increasing amount of ecological disasters many researchers have focused their attention in not only making new advancements in the theory of the field, but also by proposing ways to solve, prevent or detect those disasters to mitigate the impact that they have [1].

A natural disaster is researched this way are wildfires. According to [2], wildfires produce emissions which are highly contaminant, leading to an increase in air pollution not only in the area affected by the wildfires, but also close areas near to it. Furthermore, the natural fauna and flora are also threatened, by fires caused from human intervention. This is explained by [3], who uses as an example the Amazon Rainforest, where the human caused wildfires alongside droughts and have threatened the Rainforest to a possible tipping point, where it could become unsustainable unless there is some kind of intervention.

To tackle this problem, researchers have decided to work in the detection of wildfires in their early stages such as in [4] and [5], whose work have been either proposing CNN models using their own datasets or creating a new dataset containing images to create models that can be used to solve the problem.

Nevertheless, the models normally are not very accurate, an example of this is the one proposed by [4] whose model achieved 76%, which shows that it is still possible and worth to achieve a higher score. Specially considering that the architecture used could be seen as old, as new and better ones have been published since then. In addition to the prior, the training process was a very simple training process that was primarily based on training through a lot of epochs. Thus, new techniques can be used to not only increase the accuracy but also reduce the amount of epochs needed to train it.

That is why the objective of this research is to present, using new architectures and better training techniques, models that can achieve higher accuracy over the same test dataset as the one used in [4], while needing less amount of epochs to be trained.

II. Background / Literature Review

A. Mixed-precision training

One of the current problems that people are facing nowadays with Deep Learning (DL) is the amount of resources that it takes to train a model, either because the architecture has a lot of parameters and takes a lot of memory of the GPU. Or because it takes a lot of time to be trained due to the computational power it needs. To solve this problem, [6] proposed what is known as mixed-precision training, where instead of using the full-precision number of 8 bytes, it would use the 4 byte format. As showed in [7] using this technique led to a reduction of the amount of memory it took to train the model, in addition to a speedup in the time that the model took to be trained.

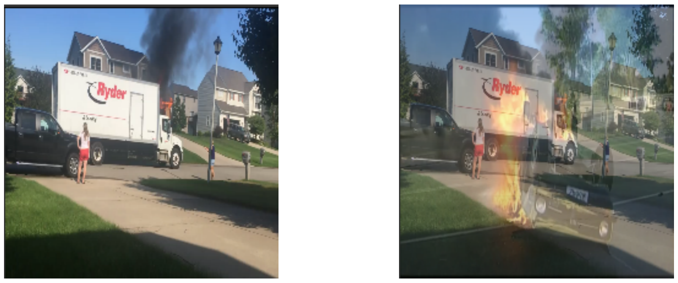
B. Data Augmentation

Data Augmentation is a technique that improves the generalization of the network. This is done by performing manipulation of the data, in the case of images, this mean applying techniques such as: shearing, rotation, saturation and others [8]. This technique is effective because it generates new data each time a new epoch is run; thus, resulting quite effective when learning from an overall small dataset. In addition to the prior, it also has been proven to be effective as it adds randomness to the training and it reduces the changes that the same batch have the same data each epoch as was shown in [9].

1) AutoAugment: Nevertheless, one limitation this technique has is that its effectiveness depends on the augmentations done over the data, as well as choosing the correct parameters, mainly their magnitude and probability. A solution of this limitation is proposed by [10] who created a procedure called AutoAugment that created a search space consisting of a policy which consisted of sub-policies that decide which augmentation to do and which are their parameters. This resulted in an improvement of the previous state-of-the-art models.

2) Test Time Augmentation: Even though, data augmentation is commonly used only for the training phase it also has a purpose during the testing phase. This technique is called Test Time Augmentation (TTA), in which the input is augmented and passed as an input for the model n times to result in a total of n outputs. With these outputs, a merge operation is then performed, which normally is a mean function. This merge result is then used to obtain the desired test metrics [11].

C. Mixup



(a) Before Mixup (b) After Mixup

Fig. 1: Example of transformation of an image when using mixup

Mixup is a technique that was proposed by [12], that aims to help in the stabilization of adversarial networks in generative models; nevertheless, it has also found success in classification tasks as was demostrated in [13]. The technique consists of mixing both the data and labels of elements in the batch, resulting in an overall generalization of how the distribution of the data would look for two different elements of the same or different class. An example of the result of mixup can be seen in figure 1.

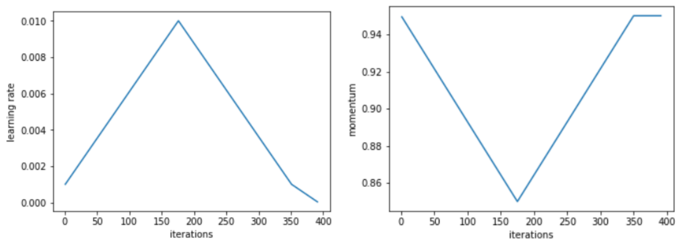
D. Transfer learning

As said by [14] “Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned”. This technique helps in reducing the amount of computational time to achieve the same or better accuracy. By using the weights of a model with the same architecture trained for another task, the model will use that prior knowledge to learn this new task faster. This has been proved by the examples of [15], [16] and [17].

E. OneCycleLR

Proposed by [19], OneCycleLR is a scheduler that was created to achieve what [19] called as “Super-Convergence”, where the model achieves an improvement of the accuracy in less epochs.

The scheduler, as can be seen at figure 2, doesn’t only consist of modifying the learning rate as most schedulers do, but it also modifies the momentum. In the case of the learning rate, it starts with an initial value, which will increase by each step until it reaches a max value;



(a) Learning rate behavior (b) Momentum behavior

Fig. 2: Behavior of momentum and LR when using OneCycleLR. Plots done by [18]

conventionally this is at the middle of the training, but can be at any other point. Then, the learning rate will decrease until reaching a value lower than the initial value. In the case of the momentum, the behavior is the opposite of the learning rate, starting with a very high value, then descending and finally going back again to the max value.

The scheduler has proven its usefulness in reducing the amount of iterations need, while keeping or increasing the accuracy in the works of [20] and [21]. In the case of the former it was possible to reduce the amount of epochs needed by a 68% compared to their baseline, while also improving the model’s accuracy.

III. Methodology

A. Dataset

The dataset used was a merge between the datasets done by [22], [5] and [23]. The code to perform the same preprocessing is fully available on Github [1].

Dataset	Fire	No Fire	Total
Train/Val	25018 (63.54%)	14357 (36.46%)	39375 (100.00%)
Test	5137 (59.61%)	3480 (40.39%)	8617 (100.00%)

TABLE I: FLAME dataset distribution

1) FLAME dataset: The FLAME dataset consists of 47,992 images that are labeled as having fire or not. 39,375 of the total amount of images are for training/validation. As can be seen at table I, the training/validation set, the labels are skewed towards the class with fire. These images were obtained by the researchers by extracting the frames of videos recorded by drones of forest areas [22].

2) FIRE’s dataset: This dataset was created for a NASA challenge in 2018, the authors collected a total of 1,000 images, all labeled for training data. These images, contrary to the previous dataset are from a wide range of environments, from urban to rural areas. Nevertheless, the dataset is skewed, containing 755 images labeled as fire and the rest as no-fire [5].

[1]https://github.com/ricglz/CE888_activities/blob/main/assignment/scripts/data_preprocessing.py

3) Dunning's dataset: The dataset was created by Dunning et al. consisting of 23,408 images for training. This dataset was created by merging other datasets and material from public videos [23]. This dataset also has a skew over the fire images.

4) Merging datasets: All the images of the FIRE's and Dunning's dataset were merged into the training/validation dataset of flame.

5) Balancing the datasets: After merging the datasets, the next part of the preprocessing was to balance the dataset; because, as mentioned in the prior sections, all the datasets are skewed towards the label with fire. To balance the dataset, we over-sample the no fire class label by performing Data Augmentation over random samples of the label. The augmentations done to the dataset were brightness, contrast, rotation, horizontal and vertical flip. This resulted in a dataset containing 76,726 images with a perfect balance between the 2 classes.

6) Dividing Training/Validation: The next step would be to split the training/validation dataset into their own predefined folders, this will help by always using the same images for training and validation, instead of random ones. Therefore the dataset [2] was split into 80% training and 20% validation, keeping the balanced ratios between the labels.

Dataset	Fire	No Fire	Total
Train	15341 (50%)	15341 (50%)	30682 (100.00%)
Validation	7671 (50%)	7671 (50%)	15342 (100.00%)
Test	5137 (59.61%)	3480 (40.39%)	8617 (100.00%)

TABLE II: Dataset distribution after preprocessing

7) Reducing the amount of data in training: With a total of 61,378 images, there was a lot of data to process. If we want that the training to be as efficient as possible we need to reduce the amount of data used for training. As there are a lot of images that are very similar between each other, due to being frames extracted from videos. Then it was decided to cut the amount of training data into half, while keeping the ratio of classes as before. This was the last step for the creation of the dataset [3] and resulted in a distribution as it shows in table II

B. Model

For this paper we will experiment with different architectures as the backbone of our model. Taking in consideration that [4] used the Xception [24] architecture as the backbone of their model, which could be considered an old architecture. Therefore it is important to look for newer models which are more efficient in terms of time of inference, training and amount of parameters.

These conditions reduced the experimentation to the next architectures: EfficientNet [25], ReXNet [26], GENet [27] and RepVGG [28].

With these architectures we will perform transfer learning to be able to learn from the current task. It must be mentioned that all of these architectures have already been trained and have been shared by [29] who have also shared the code to use the models.

C. Training

The model will be trained for 5 epochs using Mixed Precision and OneCycleLR as a learning scheduler. In addition to augmenting the data, the images will be resized to a ratio of 224x224 and normalize the values based on the mean and standard deviation of the Imagenet dataset [4]. As this was the one used in which the models were pre-trained.

Also we used a special data loader which will create batches containing the same amount of random elements of each class. This was possible due to the previous work of [30], who developed a similar sampler for their use case. Also depending of the model, it will be trained using mixup.

The optimizer for the training will be either SGD, Adam or RMSProp, depending of how the tuning of the hyper-parameters turns out. Meanwhile, the loss function will depend if the model is trained by using mixup or not. When using Mixup it will be trained by using CrossEntropyLoss, meanwhile if it's not trained by it, the model will be trained using BCEWithLogitsLoss.

When the training has ended we will restore the weights of the model to the epoch in which it had the best score, which we consider to be the average between the accuracy and F1 score in the test dataset.

In addition the complete code can be found on Github [5], where it can be found the scripts used to perform the experiments.

1) TTA: For the test dataset we will use TTA. The merge function that will be used is the mean function. Meanwhile, the amount of augmentations to perform will depend on the backbone of the model. The augmentations to perform will be determined by AutoAugment policies.

2) Data Augmentation: For training the model either custom AutoAugment policies will be used or random vertical and horizontal flips, rotations of 45º and modifications in the brightness and contrast of the photos.

3) Fine-tuning: As part of the training the model is not completely trainable from the start. At the start, only the first N layers will be trainable while the others will be frozen. At the end of each epoch the next N layers will be unfrozen, until either all the layers are now trainable or the training has been completed. N will depend of the

[2]Dataset without halving training: https://drive.google.com/file/d/1uv9vAl55IinuEMXHocnJQUhPbMikuSIX

[3]Dataset after halving training: https://drive.google.com/file/d/1RrO4boe9jHUsCY1l9Z55iG1sfydJzubs/view

[4]$mean = [0.485, 0.456, 0.406]$ and $std = [0.229, 0.224, 0.225]$

[5]https://github.com/ricglz/CE888_activities/tree/main/assignment/scripts

	Training		Validation		Test	
Model	Accuracy (%)	Loss	Accuracy (%)	Loss	Accuracy (%)	Loss
FLAME	96.79	0.0857	94.31	0.1506	76.23	0.7414
GENet	73.05	0.3925	99.15	0.0749	83.17	0.3722
EfficientNet	97.94	0.006275	99.62	0.0293	83.18	0.4198
RepVGG	98.58	0.2313	98.74	0.1525	0.8463	0.5854
ReXNet	99.23	0.0855	99.07	0.02632	90.68	0.2259

TABLE III: Models accuracies and losses in all datasets

model, as well as if the BatchNormalizer layers will also be trainable or not.

D. Hyper-parameters tuning

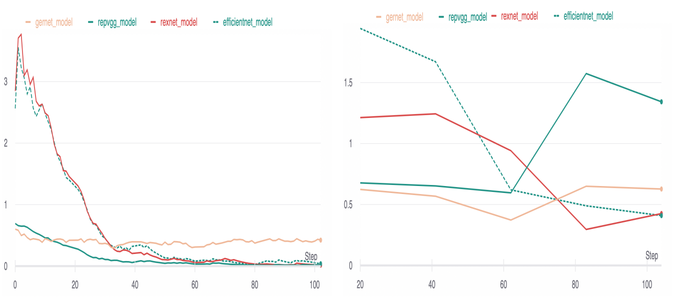
As we have architectures as backbone that have an inference time small enough to be able to iteratively tune the hyper-parameters of the model, it was decided to do it. This was done by using Weight and Biases (wandb) [31], which is a tool that allows to log models, their metrics and, more importantly for our case, to create “sweeps”. These “sweeps” work by defining a search space that allows training models with different hyper-parameters with one or many machines. In these “sweeps” it allows the user to use either grid, random or bayes search to generate the hyper-parameters to experiment with. In our case most of the time random search was enough, but there were occasions where a bayes or grid search were used instead. The hyper-parameters that were tuned were the following:

- Amount of augmentations to do for TTA
- Drop rate
- Optimizer
- Optimizer’s weight decay
- Mixup’s alpha
- If BatchNormalizer layers will be trained or not
- AutoAugment hyper-parameters:
  - If AutoAugment or pre-defined augmentations will used
  - Amount of augmentations per policy
  - Magnitude of augmentations per policy
  - Probability of augmentations
- OneCycleLR hyper-parameters:
  - Anneal strategy to use (linear or cos)
  - Min momentum value
  - Max momentum value
  - Max learning rate value
  - Div factor to determine the initial learning rate
  - Div factor to determine the minimum learning rate
  - When the learning rate will start decreasing
- Fine-tuning hyper-parameters:
  - How many layers to unfreeze per epoch

IV. Results

The following results were obtained by using NVidia RTX 2080 gpu for the ReXnet model and a GTX 1080 TI for the other.

A. Results compared to baseline



(a) Training loss through steps (b) Validation loss through steps

Fig. 3: Progression of the loss in training and validation datasets

As we can see in figure 3a, the reduction of the training loss seems normal among most of the models, the GENet model being the exception. This is because the model was the only one that was trained by using mixup. Thus, even though it has overall a high loss through all the training, the loss is also very stable without a lot of peaks, just like the ones that ReXNet and EfficientNet have at the start of the training.

We will consider the model done in [4] as our baseline to determine if the tuning and new techniques, actually improve the model performance over what is the current state-of-the-art. As it can be seen in table III, all the models perform better than the FLAME model in the test dataset, as both the accuracy and the loss in the dataset improve. Nevertheless, we have a special case that is the GENet model, in which the performance in the training dataset is considerably lower than the other models, even though it achieved a higher accuracy than the FLAME model in the test dataset. This is because GENet is the only model that used mixup as part of the training, as a result of the tuning of the hyper-parameters.

Model	Time Taken (minutes)
GENet	9.8
EfficientNet	26.1
RepVGG	9.89
ReXNet	8.22

TABLE IV: Time that each model takes to train

In addition another metric that we consider worthy to analyze is the amount of time it took to train the model. And as it can be seen in table IV, it is very impressive that most of the models used can be trained in less than 10 minutes. A comparison with the FLAME model is not possible as [4] only shared the amount of epochs needed to train, which were 40. But taking into consideration that in average we took around 3 minutes per epoch to train, then we can assume that probably their model was trained in around 120 minutes, but the real number may be higher.

B. Hyper-Parameter tuning insights

After tuning the hyper-parameters an insight concluded is that even though a lot of papers decide to use SGD as their optimizer, in our case none of the models ended up using it. Instead, the most popular optimizer for our models was Adam, which was the one used for most of the models, with the exception of RepVGG which used RMSProp.

In addition to the prior, it was interesting to see that mixup was almost non-effective, as it was only used in the GENet. The reason could be because there were only two classes; thus, reducing the effectiveness of the technique. Also it can be concluded based on the models, that if the OneCycleLR is used, it's possible that the best anneal strategy to use is a linear behavior instead of a cosine. Additionally, also for the OneCycleLR, the minimum value of the momentum should be between 0.75 and 0.82, meanwhile the max value should be 0.86 and 0.95

Finally, for this dataset the models require a drop rate between 0.45 and 0.55, as all the models have the models their best drop rate in this range.

V. Conclusion

This paper developed four models that showed the necessity that exists of using efficient techniques and architectures to not only obtain a good model performance, while using less resources during the training. This allows researchers or other developers to implement these models in IoT devices, as the response will be faster than with models that demand more resources to operate. It could even allow the developers to add the software directly to the devices themselves, because based on the information in table IV we can assume that the inference time of most of the models, with the exception of EfficientNet, is faster than needed.

Now that we know that these techniques help in the creation of better models, a next step could be to replicate the procedure, we should replicate the procedure for architectures that were designed for mobile devices or those devices with limited resources. Architectures such as: MobileNet [32], FBNet [33] and pruned versions of the less complex versions of EfficientNet [25].

Nevertheless, there are still factors that could improve the performance of the models, starting with having a more diverse dataset. As we consider that one problem with the datasets is the repetition of the same frames, caused due to being extracted from videos. This procedure may cause that the model can't learn to generalize quickly enough to classify one label with the other, or maybe every if the dataset is of very poor quality. In addition that basically this create the illusion that our dataset may be big enough, able to generalize the information, while the reality may be that there are not enough samples of all the environments. As a proposal to solve this issue it would be to create a generative model, which creates images of environments with and without fire. With this technique there would be a dataset with a broad spectrum of environments that could lead better performance and generalization for the classification models.

As mentioned before, Adam was the most used optimizer in our models, despite SGD can be seen in some cases as the favorite for training. This opens a possible new research to answer the following question: In which circumstances SGD can be a better optimizer than Adam or RMSProp? As there are works such as [34] in which it shows how SGD performs better than Adam. Could it be related to the dataset used or to the amount of epochs in which the run is performed?

References

[1] E. Weber, N. Marzo, D. P. Papadopoulos, A. Biswas, A. Lapedriza, F. Ofli, M. Imran, and A. Torralba, "Detecting natural disasters, damage, and incidents in the wild," in European Conference on Computer Vision. Springer, 2020, pp. 331–350.

[2] NOAA, "The impact of wildfires on climate and air quality," Online, 2020. [Online]. Available: https://csl.noaa.gov/factsheets/csdWildfiresFIREX.pdf

[3] Y. Malhi, L. E. Aragão, D. Galbraith, C. Huntingford, R. Fisher, P. Zelazowski, S. Sitch, C. McSweeney, and P. Meir, "Exploring the likelihood and mechanism of a climate-change-induced dieback of the amazon rainforest," Proceedings of the National Academy of Sciences, vol. 106, no. 49, pp. 20 610–20 615, 2009.

[4] A. Shamsoshoara, F. Afghah, A. Razi, L. Zheng, P. Fulé, and E. Blasch, "Aerial imagery pile burn detection using deep learning: the flame dataset," 2020.

[5] Ahmed Saied, "Fire dataset," Online, 2020. [Online]. Available: https://www.kaggle.com/phylake1337/fire-dataset

[6] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2018.

[7] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 558–567.

[8] D. A. Van Dyk and X.-L. Meng, "The art of data augmentation," Journal of Computational and Graphical Statistics, vol. 10, no. 1, pp. 1–50, 2001.

[9] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," 2017.

[10] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation policies from data," 2019.

[11] I. Kim, Y. Kim, and S. Kim, "Learning loss for test-time augmentation," arXiv preprint arXiv:2010.11422, 2020.

[12] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," 2018.

[13] C. Gong, T. Ren, M. Ye, and Q. Liu, “Maxup: A simple way to improve generalization of neural network training,” arXiv preprint arXiv:2002.09024, 2020.

[14] L. Torrey and J. Shavlik, “Transfer learning,” in Handbook of research on machine learning applications and trends: algorithms, methods, and techniques. IGI global, 2010, pp. 242–264.

[15] S. Shao, S. McAleer, R. Yan, and P. Baldi, “Highly accurate machine fault diagnosis using deep transfer learning,” IEEE Transactions on Industrial Informatics, vol. 15, no. 4, pp. 2446–2455, 2018.

[16] B. Zoph, D. Yuret, J. May, and K. Knight, “Transfer learning for low-resource neural machine translation,” arXiv preprint arXiv:1604.02201, 2016.

[17] C.-Y. Chiang, C. Barnes, P. Angelov, and R. Jiang, “Deep learning-based automated forest health diagnosis from aerial images,” IEEE Access, vol. 8, pp. 144 064–144 076, 2020.

[18] S. Gugger, “The 1cycle policy,” 2018.

[19] L. N. Smith and N. Topin, “Super-convergence: Very fast training of neural networks using large learning rates,” 2018.

[20] C.-W. Kuo, C.-Y. Ma, J.-B. Huang, Z. Kira, and G. Tech, “A state-of-the-art results with other ssl techniques.”

[21] B.-H. Kim, J. Song, J. C. Ye, and J. Baek, “Pynet-ca: enhanced pynet with channel attention for end-to-end mobile image signal processing,” in European Conference on Computer Vision. Springer, 2020, pp. 202–212.

[22] A. Shamsoshoara, F. Afghah, A. Razi, L. Zheng, P. Fulé, and E. Blasch, “The flame dataset: Aerial imagery pile burn detection using drones (uavs),” 2020. [Online]. Available: https://dx.doi.org/10.21227/qad6-r683

[23] A. Dunnings and T. Breckon, “Experimentally defined convolutional nerual network architecture variants for non-temporal real-time fire detection,” in Proc. International Conference on Image Processing. IEEE, September 2018, pp. 1558–1562.

[24] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” 2017.

[25] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” 2020.

[26] D. Han, S. Yun, B. Heo, and Y. Yoo, “Rexnet: Diminishing representational bottleneck on convolutional neural network,” 2020.

[27] M. Lin, H. Chen, X. Sun, Q. Qian, H. Li, and R. Jin, “Neural architecture design for gpu-efficient networks,” 2020.

[28] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, “Repvgg: Making vgg-style convnets great again,” 2021.

[29] R. Wightman, “Pytorch image models,” https://github.com/rwightman/pytorch-image-models, 2019.

[30] F. Galato, “pytorch-balanced-batch,” https://github.com/galatolofederico/pytorch-balanced-batch, 2019.

[31] L. Biewald, “Experiment tracking with weights and biases,” 2020, software available from wandb.com. [Online]. Available: https://www.wandb.com/

[32] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, “Searching for mobilenetv3,” 2019.

[33] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search,” 2019.

[34] N. S. Keskar and R. Socher, “Improving generalization performance by switching from adam to sgd,” arXiv preprint arXiv:1712.07628, 2017.