Improving Fire Detection with Efficient Training Techniques

Ricardo Gonzalez, 2003297

Abstract—Pending abstract

Index Terms—CNN; Deep Learning; Image Classification

1 Introduction

Nowadays due to the advancements in AI and the increasing amount of ecological disasters have led that many researchers focused their attention in not only make new advancements in the theory of the field, but also propose 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 consequence that the fire was caused due to human intervention. This is explained by [3], who uses as an example the Amazon Rainforest where the human caused wildfires alongside droughts have threatened the Rainforest to reach a possible tipping point, where it would become unsustainable unless there is some kind of intervention.

To be able to tackle this problem, researchers have decided to attempt the detection of wildfires in their early stages having the examples of [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 proposed model achieved 76%, which shows that is still possible and worth to achieve a higher score, specially considering that the architecture used is considered an old model, and 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 propose using new architectures, models that can achieve higher accuracy over the same test dataset as the one used at [4], while needing less amount of epochs to be trained.

2 Background / Literature Review

2.1 Mixed-precision training

One of the current problems that people are facing nowadays with 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 use the full-precision number of 8 bytes, it would use the 4 byte format. This 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.

2.2 Data Augmentation

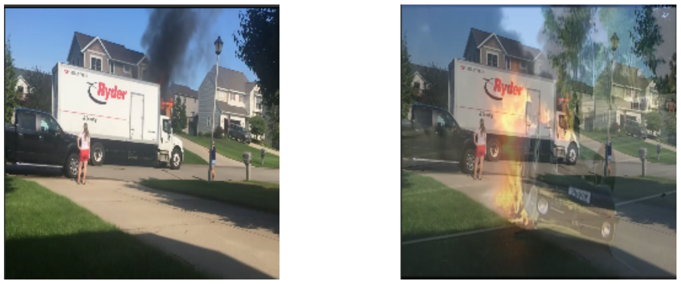
Data Augmentation is a technique that improves the generalization of the network, this is done by performing manipulation of the data, being in the case of images techniques such as: shearing, rotating, saturate and others [7]. This technique is effective because it generates new data each time a new epoch is ran. Thus, resulting quite effective when learning from an overall small dataset. In addition to the prior, it also has been proven effective as it adds randomness to the training as it reduces the changes that the same batch have the same data each epoch [8].

2.2.1 AutoAugment

Nevertheless, one limitation this technique has its effectiveness depends on the augmentations done over the data and which will be its correct parameters, mainly its magnitude and probability. A solution of this limitation is proposed by [9] who created a procedure called AutoAugment that created a search space consisting of policy which consisted of sub-policies that decide which augmentation to do and which are its parameters. This resulted in an improvement of the previous state-of-the-art models.

2.2.2 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, then is performed a merge operation which normally is to perform a mean between all the outputs obtained. This merge result is then used to obtain the desired test metrics [10].



(a) Before Mixup (b) After Mixup

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

2.3 Mixup

Is a technique that was proposed by [11] that was aimed to help in the stabilization of adversarial networks in generative model, nevertheless it has found success also in classification tasks. The technique consists of mixing both the data and labels of elements in the batch, resulting in an overall generalization of how it would look the distribution of the data of two different elements of the same or different class. An example of the result of mixup can be seen in figure 1.

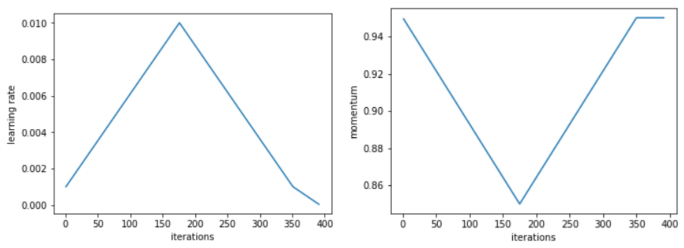
2.4 Transfer learning

As said by [12] “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 [13]

2.5 OneCycleLR

Proposed by [15], it’s a scheduler that was created to be able to achieve what [15] 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, conventionally this is at the middle of the training, but can be at any other point.



(a) Learning rate behavior (b) Momentum behavior

Fig. 2: Behavior of momentum and LR when using OneCycleLR. Plots done by [14]

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 1: FLAME dataset distribution

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 go back again to the max value.

3 Methodology

3.1 Dataset

The dataset used was a merge between the datasets done by [16], [5] and [17]. The code to perform the same preprocessing is fully available on Github [1].

3.1.1 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 1 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 [16].

3.1.2 Kaggle’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].

3.1.3 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 [17]. This dataset also has a skew over the fire images.

3.1.4 Merging datasets

All the images of the Kaggle’s and Dunning’s dataset were merged into the training/validation dataset of flame.

3.1.5 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.

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

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 2: Dataset distribution after preprocessing

3.1.6 Dividing Training/Validation

The next step would be to split the training/validation dataset into its own predefined folders, this would help for 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, will keeping the balanced ratios between the labels.

3.1.7 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 would be as efficient as possible it was a lot of data to handle, in addition that most of images were very similar between each other, as these were frames extracted of 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 2

3.2 Model

For this paper we will experiment with different architectures as the backbone of our model. Taking in consideration that [4] used the Xception [18] architecture as the backbone of their model, which could be considered as an old architecture 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 [19], ReXNet [20], GENet [21] and RepVGG [22].

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 [23] who have also shared the code to be able to used the models.

3.3 Training

The model will be trained for 5 epochs using Mixed Precision, in addition of using as learning scheduler the OneCycleLR. 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 [24], who developed a similar sampler for their use case. Also depending of the model, it will be trained using mixup. And finally, it must be mentioned that the model that will be obtained at the end will be when the model achieves the highest average between accuracy and f1 score in the validation dataset.

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 be BCEWithLogitsLoss.

In addition the prior the complete code where it shows can be found on Github [4]

3.3.1 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 AutoAugment policies.

3.3.2 Data Augmentation

For training the model will either custom AutoAugment policies or will be random vertical and horizontal flips, rotations of 45^{o} and modifications in the brightness and contrast of the photos.

3.3.3 Fine-tuning

As part of the training the model is not completely trainable since 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 model, as well as if the BatchNormalizer layers will also be trainable or not.

3.4 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) [25], which is a tool that allows to log models, its metrics and more important for our case to create "sweeps". These "sweeps" work by defining a search space it allows to train models with different hyper-parameters with one or many machines, as the wandb backend was the one that decided the hyper-parameters to use from the search space, allowing to parallelize the search between several GPUs more easily, leading to faster results. It also allowed to use either grid, random or bayes search, in our case most of the cases random search was enough, but there were cases where it was used a bayes or grid search instead.

The hyper-parameters to be tuned are 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 will use AutoAugment or pre-defined augmentations
  - 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

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

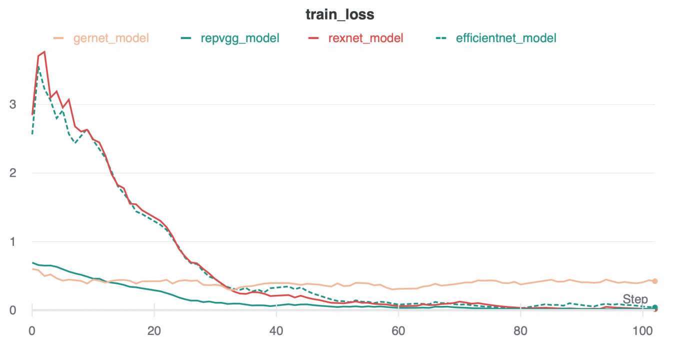
4. https://github.com/ricglz/CE888_activities/tree/main/assignment/scripts

	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

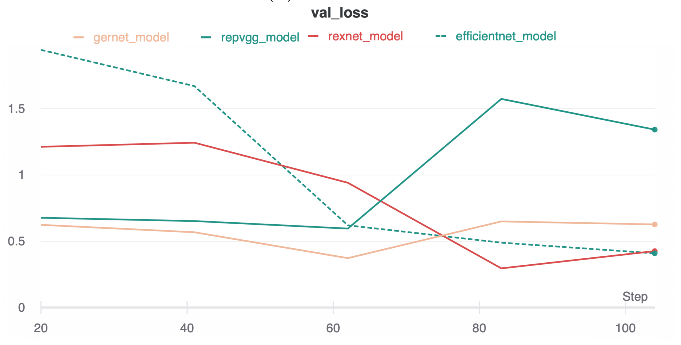
TABLE 3: Models accuracies and losses in all datasets

  - 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

4 Results



(a) Train loss



(b) Val loss

Fig. 3: Losses progression in training and validation datasets

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

As we can see in figure 3a, the reduction of the training loss seems normal among must of the models, being the exception the GENet model. 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 improves the model performance over what is the current state of the art. As it can be seen in table 3, all the models performs better than the FLAME model in the test dataset, being this that 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 it was decided based on the tuning of hyper-parameters that it will use Mixup as part of the training.

In addition another metric that we consider worthy to analyse is the amount of time it took to train the model. And as it can be seen at table 4 is very impressive that must 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, which were 40, needed to trained the epochs. But taking in consideration than in average we took around 3 minutes per epoch to be trained, then we can assume that probably their model was trained in around 120 minutes, but the real number may be higher.

5 Conclusion

This paper developed four models that showed the necessity that exists of using efficient techniques and architectures to be able to not only obtain a model good a good performance, but also to be able to achieve them with less resources. By doing this it allows the 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 4 we can assume that the inference time of must of the models, with the exception of EfficientNet, is faster than needed.

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 that the repetition of the same frames due to being extracted from videos, may cause that the model can't learn to generalize quickly enough to recognize when there is and

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

TABLE 4: Time that each model takes to train

when there's not fire in the image. 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 another next step would be, that now that we know that these techniques help in the creation of better models, we should replicate the procedure for architectures that were designed for mobile devices or those devices with limited resources. Architectures such as: MobileNet [26], FBNet [27] and pruned versions of the less complex versions of EfficientNet [19].

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] 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.

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

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

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

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

[12] 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.

[13] 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.

[14] S. Gugger, "The 1cycle policy," 2018.

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

[16] 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

[17] 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.

[18] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2017.

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

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

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

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

[23] R. Wightman, "Pytorch image models," https://github.com/rwightman/pytorch-image-models, 2019.

[24] F. Galato, "pytorch-balanced-batch," https://github.com/galatolofederico/pytorch-balanced-batch, 2019.

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

[26] 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.

[27] 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.