An Exploration into Image Classification and Object Detection

Anonymous Authors[1]

Abstract

Image Classification and Object Detection are core areas of work in Computer Vision. Much work is devoted to attempting to increase accuracy without having to build a model that is too large to be of practical use. In this paper we study EfficientNet (Tan & Le, 2019). EfficientNet is an image classification paper that explores the best ways to scale convolution nets with respect to width, depth, and resolution. EfficientNet proposes **compound scaling**. The compound scaling coefficient provides the best way to scale a model in all three of width, depth, and resolution at the same time. EfficientNet then builds a baseline model and tests the effects of compound scaling on the baseline model. In this paper, we attempt to reproduce the results of the baseline b0 EfficientNet as well as test the effects of compound scaling on a custom model.

1. Introduction

The idea of object detection is to take a target from one frame and then accurately, quickly and efficiently identify the object in the next frame. The object can be hard to track if it is moving quickly, distorts from frame to frame (humans rapidly changing poses or orientation relative to the camera angle), or becomes hidden behind distractors (e.g. shade covering the object, other people or objects moving in front of your target, etc.). The motivation for this report was centered around discovering the state of the art in Object Detection, specifically for edge devices. The papers initially reviewed all brought something novel to the field. ATOM (Danelljan et al., 2019) beats state-of-the-art (in 2019) by breaking down the problem into two subproblems: target estimation and target classification. AutoTrack (Li et al., 2020) takes the idea of discriminative correlation filters further by establishing two new parameters that can be tuned online for spatial and local regularization (dubbed Automatic Spatio-Temporal Regularization). Finally, SiamMOT (Shuai et al., 2021) makes two independent components, an implicit motion tracker and an explicit motion tracker that can be used with their generalized Siamese tracker.

After reviewing these papers more closely, the objective became to attempt to reimplement EfficientDet (Tan et al., 2020). The way to achieve this was to first implement FasterRCNN (Ren et al., 2015), which consists of a Region Proposal Network, then passes region proposals to an image classification system that would provide a single final bounding box per object and a classification of the object in the bounding box. EfficientDet has a similar workflow as FasterRCNN, in that there is a region proposal network that proposes bounding boxes, but differs in the methodology of those bounding box proposals by using a system they call BiFPN. EfficientDet uses EfficientNet to take the bounding boxes and perform image classification. After going through the implementation of the Region Proposal Network in FasterRCNN, I needed more discovery into the inner workings of image classification systems. Therefore, the focus of the reimplementation was spent on EfficientNet and image classification.

2. Related Works: Review and Critique

The related works will start at a high level with the most recent developments in the overarching goal of object detection. Then, we will get more detailed in the exploration of the core of this paper, EfficientNet.

2.1. Atom: Accurate tracking by overlap maximization

Review: ATOM aims to score well by breaking down and optimizing object tracking in two areas: target estimation and target classification. The first part is target estimation. This is achieved by extensive offline training of the target estimation system. The second component is an online trained target classification system. Both systems use ResNet-18 pretrained on ImageNet as their backbone network to provide feature maps. The goal of the target estimation system is to produce a predicted bounding box from the current frame and a reference frame by using gradient ascent to maximize an IoU (Intersect over Union) score determined from changing the bounding box. IoU is a commonly used metric in object detection. The bounding box is is then sent to the target estimation system to further refine the box. This system is trained online. They claim that Stochastic Gradient Descent would not be fast enough for their use, so they propose a new method based on Conjugate Gradient and Gauss-Newton (Danelljan et al., 2019). Their idea is to approximate their loss function using Gauss-Newton approximation. This approximation results in a positive definite quadratic function, which are special types of functions that have a closed-form solution. In their eventual implementation, they rely heavily on BackProp (Danelljan et al., 2019) to train the learned target classification parameters, which helps in refining the final bounding box output. ATOM acheives state-of-the-art, or close to it, while operating at a rate of 30 fps, which is about the expected rate in the field of object detection with UAVs.

Critique: This system requires extensive offline training for the first target estimation system. This is common in many object tracking papers, though there are ways to make your system more flexible in practice. For this system, they use a tactic called ”Hard Negative Mining” (Danelljan et al., 2019), which doubles the learning rate if a distractor peak is detected. I would worry about this in practice if used on a system where the target was different than what the target estimation system was trained on. In this case, the target might move differently than how the estimator predicts, which more frequently produces high distractor scores, which continuously raises the learning rate.

Another criticism of this paper is the arbitrary selection of certain parameters and outputs without proper explanation. Some examples: 1) Part of the training process for the target estimation system is to ”generate 16 candidate bounding boxes by adding Gaussian noise to the ground truth coordinate” for each input image pair (Danelljan et al., 2019). This number is never explained and no supplementary material is mentioned. 2) Also during offline training of the target estimation system, they select their starting learning rate, learning rate decay, and number of epochs without mention of how they came to those numbers.

They mention that SGD is not suitable for their online learning of the target classification system. During ablation study, they analyze the effectiveness of their proposed online learning system using Gauss-Newton approximation and Conjugate Gradients against SGD. The results do show that their algorithm gives a very slight bump in overlap precision and Area under Curve score for the overlap precision, the speed is the same for their algorithm and GD. They created a new categorization called GD++ that performed 5x more iterations as the standard GD categorization and commented on that category as performing slower than their algorithm. Their method is slightly better, but the results are overblown and potentially misleading.

2.2. AutoTrack: Towards high-performance visual tracking for uav with automatic spatio-temporal regularization

Review: AutoTrack takes a common concept in Object Tracking, Discriminative Correlation Filters (DCF), and improves it. At the time of this paper, DCFs use hard-to-calculate regularization terms that still end up being too rigid to adapt to new scenarios. In AutoTrack they online learn two regularization parameters for spatial and temporal regularization. These can also be thought of as local response variation and global response variation respectively. The derivation of the learning function starts with STRCF from (Li et al., 2018). There are some redefinitions of terms from STRCF to change from paramenters to references of the parameters (Li et al., 2020) to come up with a slightly different objective function as STRCF. Then, through rewriting the equation via Lagrangian Augmented form, they optimize their parameters through alternating direction method of multipliers (ADMM). AutoTrack is built for UAV application, which is common for many object detection systems, and uses a CPU. AutoTrack outperforms state-of-the-art CPU based trackers, and even outperforms deep-learning-based, GPU-based trackers that have been deployed in UAV systems.

Critique: My main question to the authors would be around the selection of the GPU based systems that they tested against. They claim their system is built for generality and can be used widely in robotics, but then do not test the system against the state of the art GPU based trackers, at least there is no mention of this. The only thing that is said is that they evaluate against ”some” GPU, deep learning based trackers. It is impressive that a CPU based tracker still beats these, but there is no reference point to how well this system performs generally against state of the art. Similarly, seeing this system performed on a dataset not built for UAVs would be interesting to see, to test its generality.

2.3. SiamMOT: Siamese multi-object tracking

Review: SiamMOT is a multiple object tracker, obviously different than the first two examined trackers, which can only track one object at a time. SiamMOT is built on a class of trackers referred to as Siamese trackers, not to be confused with Siamese Neural Networks. Siamese Neural Networks are essentially the same neural net with two different input vectors, and the results are compared. One common use case is in fingerprint analysis (Alrashidi et al., 2021). In Siamese trackers, two subsequent frames are used to train how targets move. SiamMOT is based on a multi-object tracker called SORT (Bewley et al., 2016), which uses geometric features of tracks in an object's motion to predict how the target will move across frames (Shuai et al., 2021). The authors of SiamMOT made two models of their system for evaluation: an Implicit Motion Model, and an Explicit Motion Model. In the implicit motion model, the objects being tracked are treated as points on a feature map. The Siamese Tracker then estimates where those points will be next. In the explicit motion model, the objects are treated as regions on the feature map. The motion is then explicitly predicted for the next frame. SiamMOT uses Faster-RCNN (image classification) to develop a feature map for images. The implicit motion model is trained on the loss between the guessed location change, scale change, and visibility confidence and the actual for all three between successive frames. The explicit motion model performs an IoU loss regression (similar to some tactics in earlier papers), as well as a similar loss function as the Implicit model, but performs these on a channel independent operator, which generates a pixel-level response map as opposed to just a feature level map. This pixel level map grabs all of the pixels that the supposed object is in and then learns which pixels to look for in the next frame. In ablation analysis, the writers implemented a previously implemented system that uses SORT but without either of their models as the baseline. They then test that system with the implicit model and then that system with the explicit model. The results show that the explicit model performs the best in all metrics measured for all datasets used. While the initial network is trained offline using two separate datasets, this system has shown that it is adaptive in online settings and can be a multi-use -multi-object tracker in many scenarios.

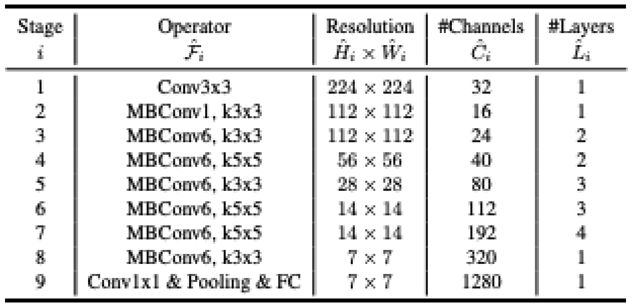
Critique: In the mathematical derivation of the objective functions, the writers continuously use t and $t+\delta$ to describe subsequent frames. I would ask them why δ and not just a generic ”1”. I would assume it has something to do with leaving it open to interpretation so that it does not assume frame rate, but that is one question I would have to the authors.

In the explicit motion model, the authors write that they calculate the likelihood of each pixel to contain the object. In videos with a high definition value, and over the course of time, I would wonder if there are any computational efficiency gains that could be attained by either reducing the pixel map to only areas that you care about (you likely know which areas the object will definitely not be).

Computational gains could be helpful for this system, as it only operates at a rate of 17 FPS which is far below average for UAV systems, where these systems are often used. The UAV accepted rate is supposed to be at 30 FPS (Danelljan et al., 2019). So for real-time practical use, this system still has a way to go.

2.4. EfficientNet

Review: EfficientNet (Tan & Le, 2019) is the image classification network used in EfficientDet (Tan et al., 2020) and is the focus of this paper. In EfficientNet, they propose a scaling coefficient to be used when scaling Convolutional Neural Nets up or down. Usually, in order to increase accuracy, the tradeoff is execution time and number of parameters. The model needs to be scaled in either the depth of the convolutional stages, the width of the channels per layer, or the resolution of the image. EfficientNet shows that you can scale in all three at the same time in order to increase the accuracy but keep the number of parameters to a minimal value. The key equation they propose is as follows:



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

Figure 1. EfficientNet-B0-baseline network

$$\begin{aligned} \text{depth: } & d = \alpha^{\phi} \\ \text{width: } & w = \beta^{\phi} \\ \text{resolution: } & r = \gamma^{\phi} \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{1}$$

where α, β, and γ are scaling coefficients for depth, width and resolution respectively. The coefficients are first found by a grid search with the baseline network. ϕ is the coefficient you want to scale your network with. The first constraint is a constraint on the number of FLOPS that will increase when you scale up your network. According to (Tan & Le, 2019), it is known that if you double the depth, d of your network, then you double the FLOPS. If you double the width or resolution, then you increase the FLOPS by a factor of 4. Therefore, when scaling your network by a factor of ϕ, then you will increase the FLOPS by $(\alpha \cdot \beta^2 \cdot \gamma^2)^{\phi}$. The constraint of ≈ 2 means that as you increase ϕ, then you are increasing the FLOPS by a factor of 2^{ϕ}

The baseline net remains to be an important factor when performing image classification. EfficientNet tests the compound scaling coefficient on a new architecture inspired by MnasNet (Tan et al., 2018). The baseline architecture is shown in Figure 1. EfficientNet then tests the compound scaling coefficient on their own network by scaling the network up seven times. Testing these networks on various datasets such as CIFAR10 and ImageNet produces state of the art results for accuracy, while also keeping the number of parameters to as low as 20x lower than the previous state of the art.

Critique: The critique of this paper revolves around two items: training time required for these neural nets, and the method in finding the scaling parameters for depth, width,

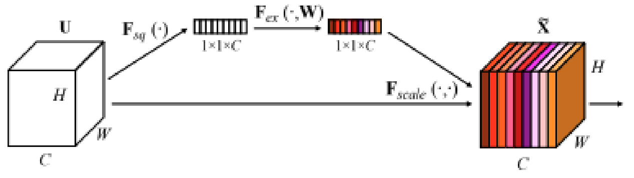


Figure 2. From (Hu et al., 2018), this is the idea of Squeeze and Excite. A convolution is performed on each channel and the image is scaled back up to the original size to learn which channels should hold the most weight.

and resolution. First, the number of epochs is not provided in the paper. They did eventually come back to this issue in the evolution of EfficientNet, called EfficientNetV2 (Tan et al., 2019). It is revealed that they used 350 epochs to train, and in the case of EfficientNetB7, training time took over 7 days of TPU time for ImageNet. The second criticism is the method of finding the baseline scaling coefficients. The method provided is not comprehensive. GridSearch is the method provided, but the only constraint given is the $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ constraint. This search space is still unbounded, and therefore needs to be clarified more.

3. Implementation

3.1. EfficientNet Reimplementation

EfficientNet was inspired by MNasNet (Tan et al., 2019), which in turn used building blocks from MobileNetV2 (Sandler et al., 2018). The building block of MobileNetV2 is a type of convolutional layer called MBConv. MBConv layers use Squeeze and Excitation (Hu et al., 2018), stochastic depth (Huang et al., 2016), and a new form of Dropout.

MBConv layers: The process flow of an MBConv layer expands the number of channels by a given factor, then performs the squeeze and excitation, and then reduces the number of channels back to the given number of output channels. This concept of expansion then reduction is a common methodology of convolution. Intuitively, Squeeze and Excite wants to assign more weight to certain channels that it learns to be important and learns the interdependence between channels. A depiction of squeeze and excite is given in Figure 2. For EfficientNet, to decrease training time, they include Stochastic Depth (Huang et al., 2016). This is a way to skip connections between layers when backpropogating and avoid the effects of vanishing gradients. This uses the input of the block as residual and adds the residual at the end of the block so that the gradient persists between layers. Another tool used for easier training in EfficientNet is the use of a new kind of form of Dropout. Traditional Dropout zeros out specific features to increase the generality of training and reduce overfitting. EfficientNet drops entire layers during training for some samples. Instead of zeroing out certain features of certain samples, with a given probability, an entire sample will be dropped out from that layer. This helps again with vanishing gradients and generalization. Putting this all together, the architecture of EfficientNet consists of a pre-processing convolutional layer to change the channel input to the expected channels of the first MBConv layer, then 7 MBConv layers, and finally a convolutional and linear head layer to perform the prediction. Within the 7 MBConv layers, most are MBConv6, which means that the channels are expanded by a factor of 6. The first MBConv1 layer expands by a factor of 1, which means that it essentially does not expand the sample, and simply performs the stochastic depth and squeeze and excite on the incoming sample.

Incorporating expansion The baseline model of EfficientNet has a predetermined number of layers per stage, number of channels per layer per stage, and an expected resolution of the input image. The EfficientNet model can be initialized with given expansion factors, α, β, γ. The depth expansion factor, α will expand the number of layers per stage by the given factor. Similarly for the resolution factor as well as the depth factor. These expansions need to be incorporated at initialization time in order to incorporate the scaling into the architecture of the resulting network.

Training EfficientNet The training methods are an important part of EfficientNet. Without the residual layers, the model does not train due to vanishing gradients. Without setting the proper learning rate and learning rate decay, the model ends up converging to a non-optimum local minimum very easily. To train the reimplementation of EfficientNet, I use the ADAM optimizer with an initial learning rate of 0.256, learning rate decay of 0.9 every 5 epochs and weight decay of 1e-5 for 300 epochs. The models are all trained on CIFAR10 on the Google Colab High Performance GPU with High-RAM runtime shape.

3.2. Compound Scaling Implementation

MyNet In order to examine the effects of compound scaling, I made a new, simple convolution net inspired by VGG (Simonyan & Zisserman, 2014) and ResNet (He et al., 2016). The base block of this architecture is a simple convolutional block with a residual connection. The convolutional blocks are generalized to allow for scaling, and the skip connection happens if the input channels are the same as the output channels to allow for scaling the depth of each layer. The number of layers per stage and channel width per layer are arbitrarily chosen just to test effects of scaling. Each stage has a 2-dimensional max pool, similar to VGG (Simonyan & Zisserman, 2014).

Baseline scaling factors The baseline coefficients for MyNet were found by GridSearch and RayLib Tuning. RayLib Tuning is a library that can be used when tuning

Scaling Coefficients for MyNet				
	ϕ	α	β	γ
MyNet-B0	0	1	1	1
MyNet-B1	0.5	1.3	1.1	1.01
MyNet-B2	1	1.6	1.1	1.02
MyNet-B3	2	2.5	1.2	1.04
MyNet-B4	3	4.0	1.3	1.06
MyNet-B5	4	6.5	1.4	1.08
MyNet-B6	5	10.4	1.9	1.1
MyNet-B7	6	16.7	2.1	1.13

Table 1. The realized scaling coefficients for MyNet. ϕ starts with zero for the baseline network. ϕ controls the number of resources necessary to scale up, so scaling is up to the user. MyNet-B1 uses ϕ=0.5 so that the highest scaled network would be computationally feasible to train and test.

hyperparameters for high dimensional and computationally expensive deep convolutional networks. You can specify the search space (which may be very large), and RayLib will sample from the search space, cancel trials quickly if it finds the result to be obviously worse than a previous trial, and run trials in parallel. The search space for compound scaling has three parameters: α, β, γ. α can be uniform between 1 and 2 with step size of 0.05.β and γ can be between 1 and 1.5 with step size of 0.05. These values are chosen because of the constraints on equation 1. With these constraints, the search space of the parameters is greatly reduced, but still impractical to search comprehensively. For the tuning of MyNet, I had RayLib collect 60 samples and established a general correlation to find that this network has a higher tendency to scale depth as compared to width and resolution. The baseline scaling factors were found to be α=1.6, β=1.1 and γ=1.02.

4. Experiments

4.1. Ef-feaux-cientNet vs EfficientNet

To evaluate the reimplementation of EfficientNet-B0, two metrics were used. It was important to get the architecture right, so to evaluate the architecture, the number of parameters for input size of 64x3x224x224 was compared. The number of parameters matched for B0-B7. The next metric to evaluate on was the Top-1 accuracy achieved. Top-1 accuracy is a common metric used to evaluate image classification models. It is the metric saying whether the top probability class matches the class of the target. This is in contrast to the Top-5 accuracy, which is correct if the target label is anywhere in the Top 5 of the probabilities of the output. Top-5 accuracy is not used in this paper. The result for B0 is shown in Table2. While the architecture matches exactly, the accuracy differs greatly. For my reimplementation, some training techniques from the EfficientNet paper were

Comparison of EfficientNet and Reimplementation		
	Params	Top-1 Acc.
Ef-feaux-cientNet	4.8M	75.4%
EfficientNet	4.8M	98.1%

Table 2. Comparison between the actual EfficientNet on CIFAR10 and my reimplementation (named Ef-feaux-cientNet)

MyNet Evaluation		
	Params	Top-1 Acc
MyNet-B0	1.7M	66.7%
MyNet-B1	2.9M	70.92%
MyNet-B2	3.9M	64.22%
MyNet-B3	6M	73.19%
MyNet-B4	11.5M	67.41%

Table 3. Results of training MyNetB0-B4. Parameters increase as scaling increases as excpected, though accuracy is more erratic.

not available to me. Google Colab (Pro version purchased) with the High-RAM structure GPU would only hold a batch size of 200 for the CIFAR10 images resized up to 224x224. More discussion on this in the Discussion section.

4.2. Evaluation of Compound Scaling

Compound scaling is proposed to be effective for any deep convolutional net. To test these effects, I made a simple net with the intention of having relatively low parameter count in the baseline model to quickly scale up and test. After finding the baseline scaling factors, each scaled model was trained in a similar fashion as Ef-feaux-cientNet with CIFAR10, for 13 epochs and a batch size of 100. The results are shown in Table 3 and Figure 3. Evaluation of MyNet is only acheived up to model B4 because the Google Colab High performance GPU could not hold all of the MyNetB5 model and a batch size of 100.

Compared to the scaling of EfficientNet, the results are not that far off. The accuracy of EfficientNet does not monotonically increase as the scaling increases. For CIFAR10, EfficientNetB2 has lower accuracy than EfficientNetB1, which

Scaling Factor Effects		
Model	Params	Top-1 Acc.
Baseline model (MyNet-B0)	1.7M	66.7%
Scale Model by depth (d=1.6)	3.3M	73.1%
Scale Model by width (w=1.2)	2.1M	69.29%
Scale Model by resolution (r=1.02)	1.7M	69.2%
Comp Scale (d=1.3,w=1.1,r=1.01)	2.9M	70.92%

Table 4. Comparison of different scaling methods. Number of parameters and Top-1 accuracy for scaling the custom baseline model by width, depth, resolution, and then compound scaling. Baseline parameters are on an assumed input size of 64x3x224x224

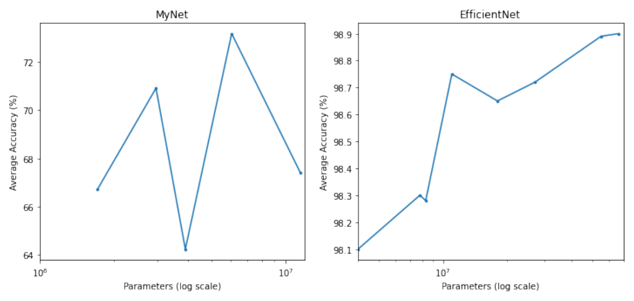


Figure 3. Compound scaling of MyNet vs the scaling of EfficientNet. The x-axis is shown with a log scale of number of parameters in the model and the y-axis is Top-1 accuracy of the model tested against CIFAR10. EfficientNet is shown on a different graph because of the higher accuracy as well as the order of ten higher number of parameters.

follows a similar arc as my scaled up model. It appears that the scaling of the base model generally increases as scaling increases, but not monotonically.

One interesting finding is that the MyNetB3 model acheives 73% accuracy in just 13 epochs and batch size of 100, which ended up taking 50 minutes. This is good relative to EfficientNetB0 trained with a batch size of 2000 for 350 epochs that ended up taking 12 hours of TPU time.

The compound scaling factor also needs to be compared to baseline models and scaling by one dimension only. To evaluate this, I compare the results of the MyNet-B2 model with the baseline B0 model as well as to scaling by only depth, only width, or only resolution. The singular scaling was done to a factor greater than any single factor of the compound scaling. For example, compound scaling was done with a depth of 1.3, while the depth only scaling was done with d=1.6. This is to show that you can get as good of results while not having to scale one dimension as much. The results, shown in Table 4 do not show a clear pattern. In general, it does appear that compound scaling is good for finding an efficient way to scale. But, atleast in a small scaling factor, it would be worth finding what is the most important dimension to scale in, and then just do that method. The compound scaling method keeps parameters relatively low and accuracy high, but simple depth scaling achieves better results with comparable parameters.

5. Discussion

5.1. Training Findings

Residual Blocks and Vanishing Gradients For networks that get very large such as these, the issue of vanishing gradients becomes very apparent. There was a point where my implementation of residual blocks was not implemented correctly. This caused the model to stay at the same loss value/accuracy value forever. Once the residual block was implemented correctly, training happened right away. In the implementation of MyNet, residual blocks were not originally intended. After starting training of these nets, the same vanishing gradient issue was found even in a model 4x smaller than EfficientNetB0. Residual blocks fixed this problem right away.

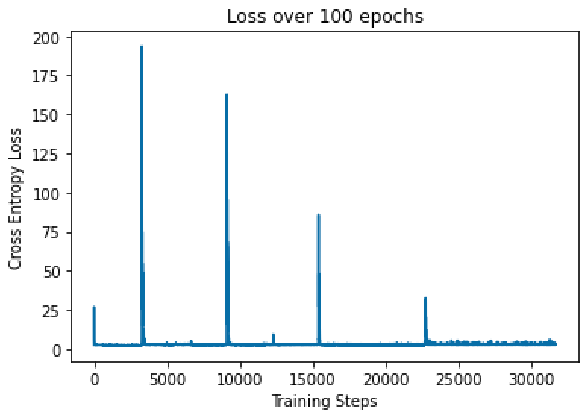


Figure 4. Spikes in CIFAR10. CIFAR10 is a dataset that is susceptible to spikes in training loss. This problem is a known problem in neural networks, and is apparent in CIFAR10. The best way to fix this issue is to increase batch size so that you have less chance of getting a random batch of abnormal images.

5.2. Scaling Findings

The scaling findings do not match the EfficientNet paper exactly. Compared to EfficientNet for CIFAR10, the scaling does match the general trend for scale factor vs accuracy. Though, compound scaling for MyNet performs similarly to scaling in the depth dimension. Due to time and resource constraints, these models were all only trained with 13 epochs and the scaling factors were closely related (width only scaling of 1.2 is not far from the compound scaling width of 1.1). In order to confirm these results, I would like to increase the scaling factors, train for greater epoch count, and be able to hold a larger batch size on the GPU. The initial results do show a trend, but cannot be confirmed.

5.3. Spikes in training and impact of batch size

Figure 4 shows one attempt at training the reimplementation of EfficientNetB0. The occasional spike is an issue that can arrise when training deep nets, and has been seen with CIFAR10. The way to address this issue is to increase the batch size so that the variance between batches can be reduced. With the constraints of the device I was training on, the batch size could not get over 200 even for smaller networks. Therefore this issue of spikes during training could not exactly be avoided with the EfficientNet architecture.

Another downside of this small batch size was the inability to attain the accuracy attained by the EfficientNet paper. I believe that a higher batch size would have gotten the accuracy closer to the results in the paper because you can see each sample more times, and you don't see this spiking issue during training. For comparison, the EfficientNet paper batch size was set to 2000; 20x what the Google Colab High performance GPU could handle.

5.4. Edge Device Application

The original intention of this report was to explore object detection and image classification, specifically for resource constrained devices such as UAVs. EfficientNet is a good start down that path. When deciding what image classification/object detection network to use, it is helpful to know that you could scale up or down based on the resource constraints of your device. The next steps will be to further explore resource efficient object detection methods.

References

Alrashidi, A., Alotaibi, A., Hussain, M., AlShehri, H., AboAlSamh, H. A., and Bebis, G. Cross-sensor fingerprint matching using siamese network and adversarial learning. *Sensors*, 21(11):3657, 2021.

Bewley, A., Ge, Z., Ott, L., Ramos, F., and Upcroft, B. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*, pp. 3464–3468. IEEE, 2016.

Danelljan, M., Bhat, G., Khan, F. S., and Felsberg, M. Atom: Accurate tracking by overlap maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. URL https://openaccess.thecvf.com/content_CVPR_2019/papers/Danelljan_ATOM_Accurate_Tracking_by_Overlap_Maximization_CVPR_2019_paper.pdf.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Li, F., Tian, C., Zuo, W., Zhang, L., and Yang, M.-H. Learning spatial-temporal regularized correlation filters for visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Li, Y., Fu, C., Ding, F., Huang, Z., and Lu, G. Autotrack: Towards high-performance visual tracking for uav with automatic spatio-temporal regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. URL https://openaccess.thecvf.com/content_CVPR_2020/papers/Li_AutoTrack_Towards_High-Performance_Visual_Tracking_for_UAV_With_Automatic_Spatio-Temporal_CVPR_2020_paper.pdf.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Shuai, B., Berneshawi, A., Li, X., Modolo, D., and Tighe, J. Siammot: Siamese multi-object tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12372–12382, June 2021. URL https://openaccess.thecvf.com/content/CVPR2021/papers/Shuai_SiamMOT_Siamese_Multi-Object_Tracking_CVPR_2021_paper.pdf.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Tan, M. and Le, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/tan19a.html.

Tan, M., Chen, B., Pang, R., Vasudevan, V., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018. URL http://arxiv.org/abs/1807.11626.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Tan, M., Pang, R., and Le, Q. V. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10781–10790, 2020.