GreenEdge Contest: Energy-Aware Image Classification

Andrea Scanu, Luca Vergolani
University of Padova, Padova (Italy)
scanuandrea1997@gmail.com, luca.vergolani@gmail.com
GreeNet Group
(Dated: June 15, 2024)

DATASET

The dataset used in this competitions and during the training of the neural networks is named TinyImageNet Dataset. This dataset is a subset of the ImageNet Dataset which contains more than a million images, labeled by hand in 10000 categories. TinyImageNet contains just a small portion of the total dataset, in fact, it is composed by 100000 images divided equally in 500 categories, 200 images for each one. In particular the dataset has been divided in three different chunks: train, validation and test dataset as in the dataset present in the deeplake python library. Each image is composed of 3 channel (RGB) and for each of them a total of 64x64 pixels.

ARCHITECTURES

In science literature and image recognition history there are plenty of networks that were used to accomplish different levels of accuracy on the ImageNet dataset. The first attempt in the analysis done for this report was the training of different networks built for the image classification task, here a list of the networks used for the challenge:

- AlexNet: this network was created to classify ImageNet, it is composed of 5 convolutional layers, 3 fully connected layers and Relu activation functions. It is a pioneered CNNs in large-scale image recognition.
- VGGNet: one of the first deep networks with 16 or 19 layers.
- ResNet: this is a family of neural networks, it introduced residual connections to prevent degradation in deep networks and enabled very deep (50-100 layers) networks to be trained effectively.
- EfficientNet: uses a compound scaling method to balance network depth, width, and resolution.

The training method that has been used is the same for all the networks and here the characteristics are summarized:

- Batch size = 256
- learning rate = 0.001
- Optimizer = Adam
- loss = cross entropy

These networks were used to have a benchmark for the accuracy and energy (measured in Gflops) of a heterogeneous set of neural networks from a very shallow architecture such as ALexNet to a very deep and complex like EfficientNet.

All the results are presented in the RESULTS section.

AUGMENTATION

Once we tested the architectures presented in the previous section, we moved our focus on the data augmentation techniques to enhance the classification performances. Indeed, data augmentation well suits to the aim of this challenge since it doesn't change the final energy consumption of the network inference due to the fact that is present only during the training process. Finally, this compensate the reduced size of the dataset and small size of the images.

The first approach to data augmentation was focused on the standard augmentation techniques, in particular:

- Horizontal and Vertical Flipping: Flipping images horizontally or vertically.
- Rotation: Rotating images by a certain angle.
- Scaling: Zooming in or out of the image.
- Translation: Shifting the image along the x or y axis.
- Shearing: Applying a shearing transformation.
- Cropping: Randomly cropping a part of the image.
- Brightness Adjustment: Modifying the brightness of the image.
- Gaussian Noise: Adding random noise to the image.
- Blur: Applying blurring techniques like Gaussian blur.
- Cutout: Randomly masking out square regions of the image.
- Channel Shuffling: Randomly shuffle RGB channels.

Moreover, there exists more sophisticated data augmentation techinuques, among these we tried the following:

- Mixup: Combining two images by blending them. NB: In every graph related to cutmix/mixup the training accuracy value appears to be worse that the test since the definition of accuracy for a model that has to predict two different labals is not suitable. We choose to only refer to one of the two labels to compute the training accuracy
- CutMix: Combining two images by cutting and pasting patches from each other.
- Local Augmentation: A combination of rotation and shuffling of patches from the same image [9]

We tried different combination of data augmentation techniques, trying to isolate the ones that improve the model generalization. We observe that the best results are reached by using cutmix and a simple augmentation sequence of random horizontal and vertical flip with 0.25 probability.

PRE-TRAINED NETWORKS

Training a model from scratch is computational demanding and time consuming, since model studied for image classification are very complex and the number of classes is relatively high. Since training different network for hundreds of epochs and optimising hyper-parameters for each of them is not feasible on a personal computer we tried the knowledge distillation technique.

Pre-trained networks, also known as transfer learning or knowledge distillation, leverage models that have been previously trained on large datasets to solve new but related tasks. This technique is especially popular in deep learning.

Our approach was to take pre-trained networks from ImageNet1k dataset and to adapt them on the TinyImageNet dataset since the latter is a subset (with low resolution images and less categories) of the former. The idea behind this attempt was that the pre-trained networks are able to catch features that are relevant for the TinyImageNet classification task. However, these networks need to be modified to be adapted to the new classification task, and also they need to be retrained to adjust their weights.

The choice of the networks that we have done was made keeping in mind the energy score that we have obtained with the architectures analysed before and their generalization capability.

EARLY EXIT

The previous approaches, such as data augmentation, knowledge distillation and working with small size images were useful to guarantee that models kept the inference energy cost low while trying to get higher accuracy levels. We then proceeded to reduce the energy cost by adopting a specific method: Early Exit [5].

Early Exit consists in adding some exit branches to the original model. The model evaluates to use an exit considering the distribution entropy of the corresponding exit prediction. This allows the model to reduce inference energy cost since for some data is not needed to be processed through the entire network.

We then tried different models changing the entropy threshold to find a good trade-off between energy and accuracy.

RESULTS

In this section we briefly explain the results we got. In TABLE I we show how the trained from scratch models we tested (AlexNet, VggNet, ResNet, ResNeXt) behave with different techniques appied during training. From this results we notice that the best overall model is ResNeXt because it reaches higher accuracy levels but with the same energy consumption.

In FIGURE 3, 4, 5 we repoerted the training graphs for the best results among non pre-trained network and with different techniques.

Whereas in TABLE II we show how the pre-trained models behave applying the respective training techniques. Since the best models were ResNet and ResNext we choose to train only those two models. In addition we compare the results we got using 64x64 and 128x128 image size. The choice of resizing the images was made observing from TABLE I that increasing the image size can indeed improve the accuracy performance but at the cost of higher energy consumption and higher computational complexity.

Finally in FIGURE 1 we compare the accuracy/energy consumption of both the pretrained ResNet and ResNeXt (trained on 64x64 and 128x128 image size) with respect to the entropy threshold T that controls the exits output. It can be possible to observe that the inference energy cost is quite similar between the different models of ResNet and ResNeXt but ResNeXt performs a bit better on accuracy performance. For this reason for the aim of the challenge we choose to evaluate only ResNeXt models.

In FIGURE 2 we show the chosen models compared with respect to accuracy, inference energy consumption and entropy threshold. We also show the corresponding area computed with the provided method in the challenge python code, which is 31.78.

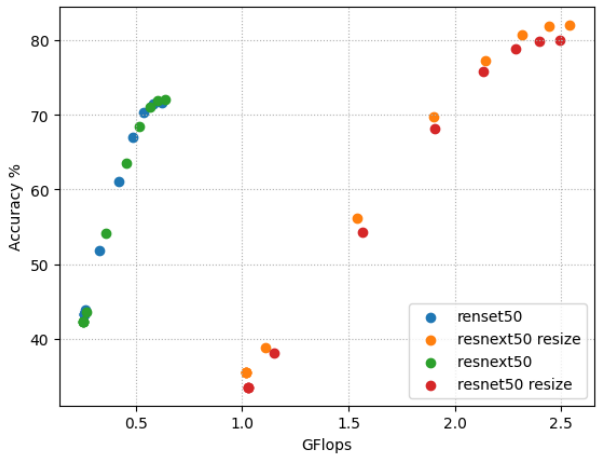

Figure 1: Early Exit Models accuracy and energy wrt entropy threshold

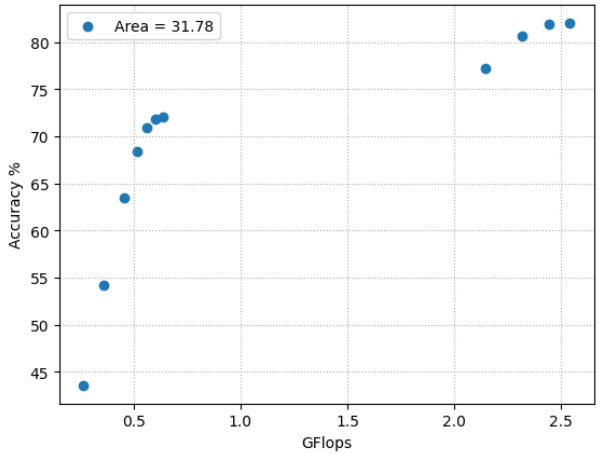

Figure 2: Chosen Early Exit Models accuracy and energy wrt entropy threshold

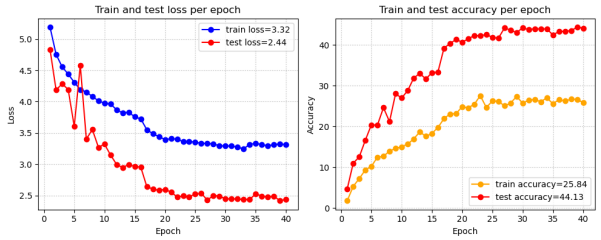

Figure 3: From scratch ResNet training with cutmix + augmentation

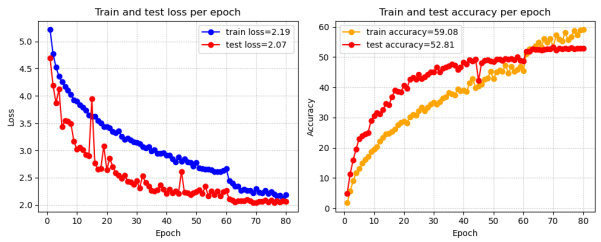

Figure 4: From scratch ResNeXt training with cutmix + augmentation

Table I: Results obtained training the models from scratch

Model	Augmentation	Cutmix	Resize	top-accuracy
Trained	No	No	No	39%
Trained	Yes	No	No	43%
Trained	Yes	Yes	No	45%
ResNeXt50	Yes	Yes	No	52.8%
ResNeXt50	Yes	Yes	Yes	55.6%

Table II: Results obtained training the pre-trained models

Model	Augmentation	Cutmix	Resize	Early Exit	top-accuracy
ResNet50	Yes	Yes	No	No	64.1%
ResNet50	Yes	Yes	No	Yes	71.6%
ResNet50	Yes	Yes	Yes	Yes	79.9%
ResNeXt50	Yes	Yes	No	Yes	72.0%
ResNeXt50	Yes	Yes	Yes	Yes	82.0%

SUBMISSION AND TECHNICAL NOTES

How to properly test.

From the beginnig of our work we faced different problems using deeplake library. The first problem was the one mentioned in the github issue: even trying to change the parameters we couldn't solve the problem so we downloaded the entire dataset and we loaded it by using a custom Dataset class which can be found in the utils.py file. The second problem we faced was the labels mismatch between tiny-imagenet and imagenet dataset mentioned in issue that was later fixed by deeplake staff. Our work was based entirely on the dataset we downloaded at the beginning so it is not updated as the current deeplkake verion. To work with the old dataset we created a custom map that relates the dataset labels to the interval [0-199] needed by pytorch to create the hot-vectors and to train the models. In this moment we noticed that deeplake deleted the validation dataset so even for validating out models we used the old validation dataset we downloaded. To make possible to test our models on a different test dataset, we created another custom map that relates the imagenet classes labels to our custom labels. We provide both maps in "labels_map.pkl" and "deeplake_labels_map.pkl" files. **If you want to test our models notice that our models will output values in a range [0-199], and to convert this values to the deeplake corresponding labels you will need the "deeplake_labels_map.pkl" which consist in a python dictionary where the keys are our output labels and the values are the corresponding deeplake labels. We save the map as a pickle file so to load it you need to run the following lines of code:**

```
import pickle

with open('dict.pkl', 'rb') as handle:
```

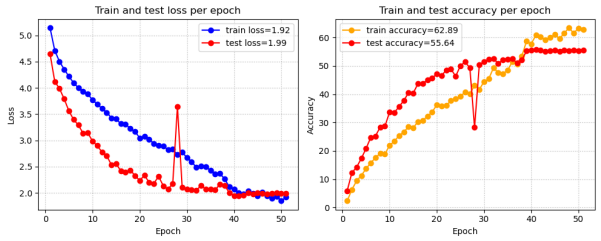

Figure 5: From scratch ResNeXt training with cutmix + augmentationc + resize

```
dict = pickle.load(handle)
```

where dict is the "deeplake_labels_map".

Also if you want to test our model that work with 128x128 image size you need to resize the dataset to match this image size to work properly. In pytorch you just need to add the following code to your transformation:

```
t=transforms.InterpolationMode.BICUBIC
transforms.Resize(128,interpolation=t)
```

Drive folder description

In the drive folder are present 4 folders and 5 files. The "Papers", ".ipynb_checkpoints", "Other attempts" folders are not of interest for the present discussion. The 5 files present are just the datasets and some maps that are used by our notebook. The folder where you can find our models is the "Pretrained" folder. Inside of this folder there are other 2 folders and 5 files.

- The two folders are "Resize" and "Models" where we saved the weights of our models.
- Then there is a "utils.py" file where we keep classes that are used in the notebooks.
- The "deeplake_labels_map.pkl" file contains the map to convert our labels to the deeplake one.
- The "train" notebooks are the ones that we used to train our models.
- The "test" notebook contains a code that test all of our models on the validation dataset and then compute the area for the models that we have chosen to submit to this challenge.

Here a link to our drive: link.

To use the shared folder on colab you need to click on the shared folder and click on "Add shortcut to Drive", in this way the shared folder will appear on the personal drive and can be seen in colab drive directory. We set the path variables in the notebooks to work properly.

Description of the submitted models.

Since the model that we are submitting are all created using the early exit approach we have to specify the entropy threshold that we used in the models definition in order to allow you to replicate the models.

We are delivering 11 models, 4 with ResNeXt50 trained on resized images with entropy thresholds T = [0, 0.63, 1.26, 2] , and 7 with ResNeXt50 trained on TinyImageNet non resized with entropy thresholds T = [0, 0.63, 1.26, 2, 2.63, 3.26, 3.9]. To replicate our model we have built two classes: EarlyExitCustomNet64 and EarlyExitCustomNet128, by default the classes load the ResNeXt50 model and if you give a set of weights, the threshold and the device (cpu or gpu) you get a model ready to predict on a dataset.

OTHER CONSIDERATIONS AND FUTURE IMPROVEMENTS

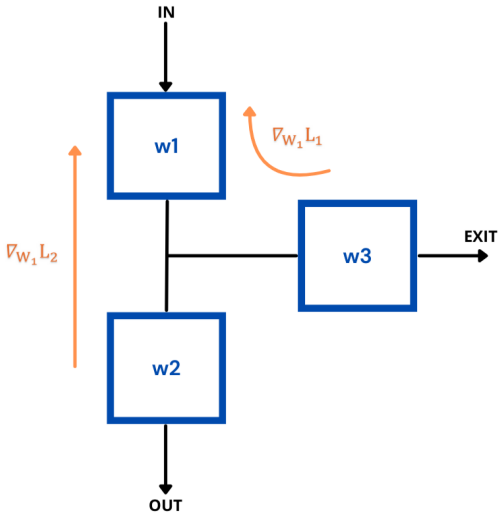

Figure 6: Early Exit example with two exits

Early Exit (EE) improvement

The main focus of this section is to highlight the effect of the Early exit on the model training, we supposed that this technique was just a energy-improving method but we noticed that the effect of the early exit was positive also for the accuracy score. For instance from TABLE II it can be noticed a huge improvement of ResNet50 which scored a 64.1% without EE and 71.6% using the EE technique. The idea behind this improvement is that some layers receive more gradient signal during the training, one for every exit, pushing the first layers to produce more meaningful features. Mathematically, if we think to a model with just one

extra exit, as shown in FIGURE 6, the loss can be written as:

$$L(w_1, w_2) = L_2(w_1, w_2) + L_1(w_1, w_{ex})$$

so the loss is the sum of two term, L_1 is the loss for the first exit, L_2 is the loss for the standard exit of the network, w_{ex} are the weights for the exit branch, w_1 are the weights of the first layers and w_2 the weights of the layers after the exit. This means that computing the gradient just for the first layers:

$$\nabla_{w_1} L = \nabla_{w_1} L_2 + \nabla_{w_1} L_1$$

it is clear that the term $\nabla_{w_1} L_1$ is an extra signal that in a normal training would not be present. It would be interesting to study the difference between the first layers of EE network and the first layers of a classical network and how the position of the exit affects the accuracy.

Early Exit on complex networks

A possible challenge with the early exit can be the formal implementation for an algorithm that decides the position and the number of exit in particular with more complex networks like EfficientNet that is provided with stochastic layers that can be randomly used or not. In these cases the positioning of an exit can be more challenging. We decided not to try EfficientNet in this work because the best EfficienNet models required high computational and time cost to train and test and we did not have enough ram and gpu memory.

CONCLUSIONS

The best accuracy that we obtained is 82% with a consumption of 2.55 Gflops and the set of models submitted scores an area of 31.78 on the validation dataset.

[1] *Paper: AutoMix: Unveiling the Power of Mixup for Stronger Classifiers*,
[2] *Paper: Boosting Discriminative Visual Representation Learning with Scenario-Agnostic Mixup*,
[3] *Paper: ResNet on Tiny ImageNet*,
[4] *Paper: mixup: BEYOND EMPIRICAL RISK MINIMIZATION*,
[5] *Paper: BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks*,
[6] *Paper: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*,
[7] *Paper: Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*,
[8] *Paper: AutoMix: Unveiling the Power of Mixup for Stronger Classifiers*,
[9] *Paper: Local Augment: Utilizing Local Bias Property of Convolutional Neural Networks for Data Augmentation*,