Benchmark of neural network models with reduced number of parameters for image classification

Rares Folea
Department of Computer Science
Politehnic University of Bucharest
Bucharest, Romania
rares.folea@stud.acs.upb.ro

Abstract

This paper analyze the performance of various **deep neural networks**, such as **residual networks**, **VGGs** or **Vision Transformers** against the problem of image classification problems. The analyzed networks are reduced versions of well-known deep neural network architectures that have under 1M trainable parameters. We propose architectural design changes to the networks, as well as looking into pruning techniques on the well-established models. As of 2022, networks under 1M trainable parameters are considered to be composed relatively of a number of **reduced parameters**. We analyze, as well, the influence of various optimizers based on **Stochastic Gradient Descent** algorithms. Most of the evaluations for this project are being performed on the **CIFAR10** dataset, but we present some additional results for the **CIFAR100** dataset.

1 Introduction

Image recognition is a common method for allowing computers to recognize items inside a picture. Image labeling, image search, medical condition recognition, and self-driving cars are just a few of the jobs that use such technologies.[7] The **CIFAR-10** dataset is a set of photographs used to train machine learning and computer vision techniques. This is one of the most used databases for deep neural network studies.

State-of-the-Art models reach phenomenal classification performances on the **CIFAR-10** dataset, with even over 99% accuracy on the task of prediction. However, as outlined in **Table 1**, these models can have tens or hundreds of million parameters. As of Late 2021, the highest percentage of correct classifications is achieved by the **ViT-H/14** model, reaching **99.5%** accuracy. Yet, this model has **632M parameters**. Also other top performing architecture (over 95% accuracy), such as models based on **VGG-19** or **ResNet50** have *20M*, respectively *25M parameters*.

In this paper, we investigate neural network models with a reduced number of parameters, against the task of image classification of on the **CIFAR-10** and **CIFAR-100** dataset.

By *reduced number of parameters*, we want to investigate neural networks with **at most one million number of parameters**. While **1M** parameters can sound completely not a small value for parameters, we will see that state-of-the-art networks can have up to **632M** parameters.

As stated in the **lottery ticket hypothesis**, it is believed that for any given *sufficiently* dense neural network, there exists a random sub-network, within that network, that has *one tenth of parameter*s, but would **converge to the same performance in the same amount of training time** [4].

2021-2022: NN and ATAI project submission. AI master @ ACS, UPB, Bucharest, Romania

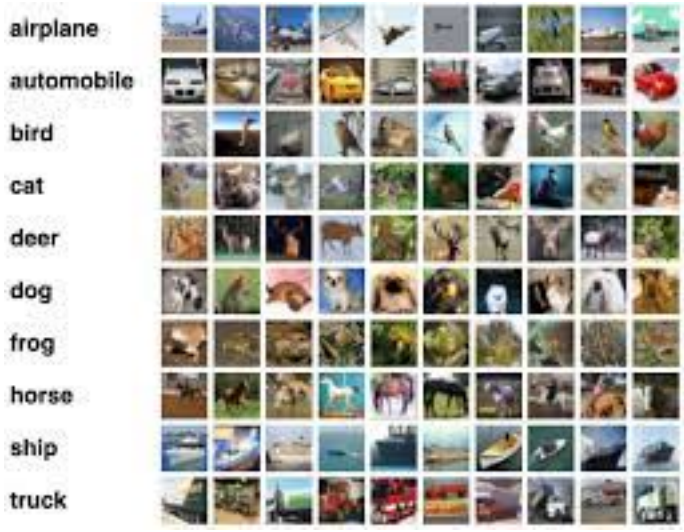


Figure 1: The 10 classes in the CIFAR-10 dataset, with some sample 32x32 images from each class

We wanted to analyze the networks are reduced versions of well known deep neural network architectures that have under 1M trainable parameters, by either proposing architectural design changes to the networks, as well as looking into pruning techniques on the well-established models.

2 Dataset

CIFAR-10 is a computer-vision dataset used for benchmarks for the task of object recognition. The **CIFAR-10** dataset consists of 60000 32x32 colour images in **10 classes**, with 6000 images per class. There are *50000 training images and 10000 test images*. [8]

Throughout our research, we will benchmark all the neural networks against the **CIFAR-10** dataset. This ensures an objective comparison between the models and optimizers.

The **CIFAR-10** dataset contains *60 thousand* **32x32** color images in *10 different classes*. The 10 different classes represent **airplanes**, **cars**, **birds**, **cats**, **deer**, **dogs**, **frogs**, **horses**, **ships**, and **trucks**. There are **6 thousand** images of each class. All the classes in this dataset are mutually exclusive, with no overlap between the classes. The total size of the dataset is **163 MB**.

For architecture that perform similar, we will use the **CIFAR-100** dataset as a tie-break. Introduced by the Canadian Institute for Advanced Research, this 100 classes dataset is a subset of the Tiny Images dataset and consists of 60000 32x32 color images. The 100 classes in the **CIFAR-100** are grouped into 20 super classes, with 600 images per class. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the super-class to which it belongs). There are 500 training images and 100 testing images per class [8].

3 Related Work

3.1 State-of-the-art Models for image classification on CIFAR-10

Over the time, the **CIFAR-10** dataset become a benchmark for the task of image classification for various models.

As of Late 2021, the highest percentage of correct classifications is achieved by the **ViT-H/14** model. Over 10000 training images, the model has achieved a 99.5% accuracy on predicting the class of the image, meaning less than 50 incorrect prediction over the 10k testing dataset. This has been initially introduced in 2020 [3], in the paper *"An Image is Worth 16x16 Words: Transformers for*

Model	Number of parameters	Percentage correct
ViT-H/14 (2020)	**632M**	**99.50**
LaNet (2019)	**44.1M**	**99.03**
DenseNet-BC-190 + Mixup (2017)	**25.6M**	**97.3**
VGG-19 with GradInit (2021)	**20.03M**	**94.71**

Table 1: The above table presents some important models developed over time for the Image Classification challenge on the CIFAR-10 dataset.

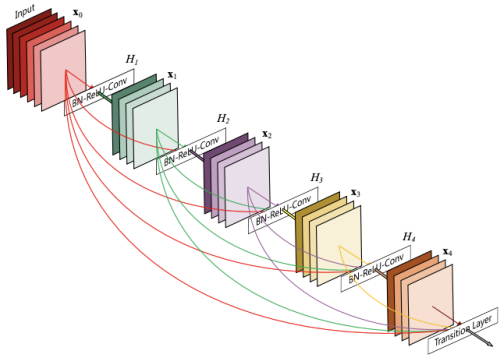


Figure 2: A sample DenseNet architecture, a model that utilises dense connections between layers, through *"Dense Blocks"*, layers where all layers are connected directly with each other.

Image Recognition at Scale". This introduced a Transformer architecture for solving the task of image classification. The model uses attention in conjunction with convolutional networks and to replace certain components of convolutional networks. [3] shows that Convolutional Neural Networks are not required for small image classification tasks, and that a pure transformer applied straight to sequences of picture patches can perform well.

Another notable model from 2019, **LaNet** [13] was introduced by the *"Sample-Efficient Neural Architecture Search by Learning Action Space for Monte Carlo Tree Search"* paper. Such a architecture learns to partition the search space, so that solvers can focus on a smaller region to find better solutions with fewer samples.

The **DenseNet-BC-190 + Mixup** model [14] introduced in 2017. The model uses at core a **DenseNet** architecture, a type of convolutional neural network that utilises dense connections between layers.

The paper [14] presents the drawbacks of Large Deep neural networks such as memory and sensitivity to hostile samples. Mixup is a learning principle, which is proposed in this paper as a solution to these problems. What Mixup does essentially is that it trains a neural network using convex combinations of pairs of instances and their labels to train a neural network, allowing it to favor simple linear behavior between training examples.

Finally, **VGG-19 with GradInit** [15] is another model that performs well on the CIFAR-10 dataset.

VGG-19 is a Convolutional Neural Network. There are 19 stands for the number of layers with trainable weights, 16 Convolutional layers and 3 Fully Connected layers.

GradInit [15], an automated and architecture agnostic method for initializing neural networks. GradInit is based on a simple heuristic; the norm of each network layer is adjusted so that a single step of *SGD* or *Adam* with prescribed hyper parameters results in the smallest possible loss value.

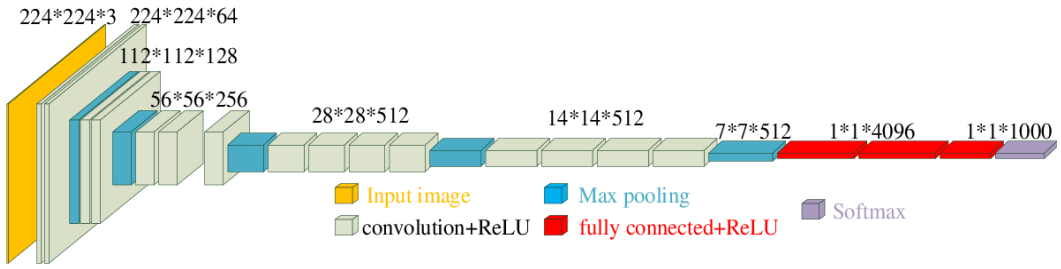


Figure 3: An overall architecture for a VGG-19 network.

However, as outlined in **Table 1**, these models have a high numbers of parameter, making the process of training a significant inconvenient, requiring dedicated hardware and resulting in high energy commitments.

3.2 Architectures of Deep Neural Networks with a reduced number of parameters

The objective of this paper is to benchmark the image classification of neural network models with a reduced number of parameters. In the next subsection, we will analyze the state-of-the-art models for the classification problem on the **CIFAR-10** dataset. For the architectures that perform really well (over 90% accuracy) we will also provide some results on the **CIFAR-100** dataset.

3.2.1 Reduced-ResNet architecture

A residual neural network (known as ResNet [5]) is a type of neural network. Skip connections, or shortcuts, are used by residual neural networks to jump past some layers. The majority of ResNet models use double or triple layer skips with non-linearities (such as Rectified Linear Unit) and batch normalization in between.

Residual networks are highly configurable and usually the more deeper the ResNet is, the higher accuracy is obtained. Certain ResNet architectures have been wildly studied and proved to provide good results: **ResNet18**, **ResNet34**, **ResNet50**, **ResNet101**, **ResNet152**. ResNet18 has 18 convolutional layers, while ResNet152 has 152 convolutional layers.

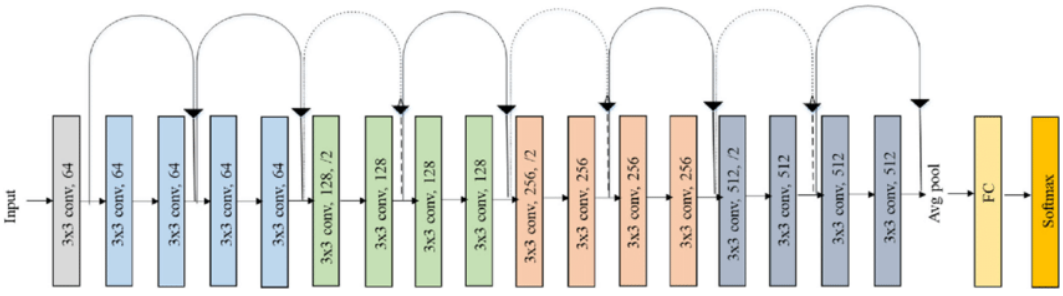


Figure 4: The classical architecture for the Deep Convolutional Network for ResNet18.

However, the number of parameters of these networks is quite high. While ResNet18 has around 11 million trainable parameters, ResNet34 has 21M. More deep models such as ResNet50, ResNet101 or ResNet152 have 23M, 42M, respectively 58M parameters.

In order to fit the limit of 1M parameters, we have decided to use in benchmarking a different ResNet architecture, based on ResNet18 but with less output channels for the first 2D convolution. Therefore,

instead of outputting the 64 channels, our **ResNet18_Small** (also referred as **ResNet18_S**) neural network will only output on 16 channels.

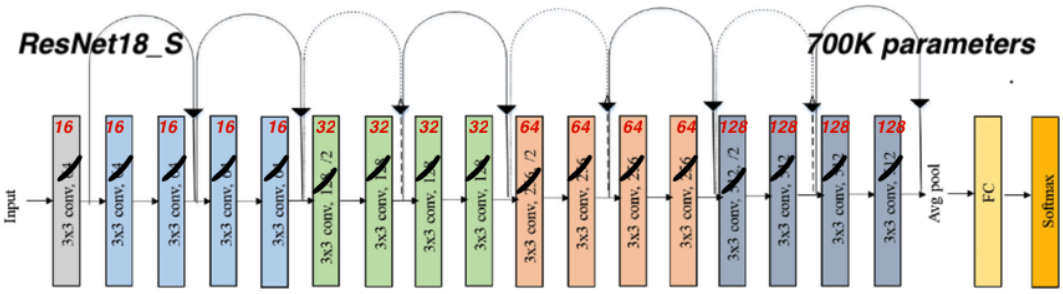


Figure 5: The benchmark-ed ResNet18_S, a slightly modified ResNet18 with less than 1M parameters.

The resulting network ResNet18_S has a total of 700K trainable parameters, as opposed to the 11M trainable parameters of ResNet18, making it a good candidate for our research.

3.2.2 Reduced VGG-16 architecture

VGG are a type of Very Deep Convolutional Networks, initially introduced in [12] are networks of increasing depth using an architecture with very small (3x3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16-19 weight layers.

In particular, **VGG16** (16 depth-layers) is a convolutional neural network architecture which instead of having a large number of hyper-parameter, it is focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and max-pool layer of a 2x2 filter with the stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 fully-connected layers followed by a soft-max layer for output. [12]. A classic **VGG16** architecture for **CIFAR-10** has **15245130** trainable parameters.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 6: Sample ResNet [5] architectures. For each network (**ResNet18**, **ResNet34**, **ResNet50**, **ResNet101**, **ResNet152**) the main convolutions blocks are similar, but replicated a different number of times, therefore a different in the depth of the convolutional network.

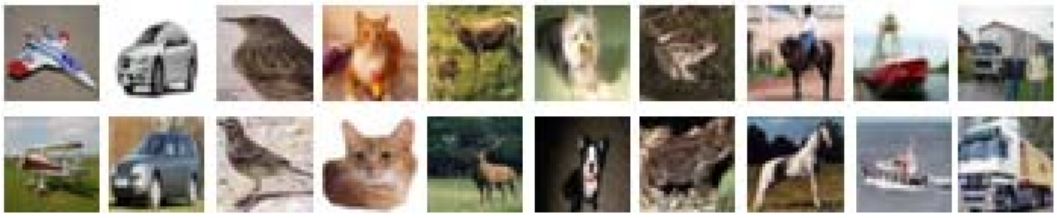


Figure 7: Sample images from the CIFAR-10 dataset.

3.2.3 Reduced Visual Transformers(ViT) architecture

A Vision Transformer (ViT) is a transformer designed for image recognition and other vision-processing applications. Usually, transformers have way more parameters than already classical convolutional deep neural network architectures such as ResNet18 or VGG-16. The SotA ViT-H/14 transformer that obtained 99.5% accuracy on the CIFAR-10 dataset had over 600M trainable parameters.

3.3 Optimizers

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.

We are aiming to benchmark the following suite of algorithms for Stochastic Gradient Descent Optimization Algorithms for the image classification problems based on **CIFAR10** dataset.

1. **Stochastic Gradient Descent with Momentum** [9] (SGDM)
2. **Adam**: A method for stochastic optimization [6] (ADAM)
3. **AdaBound**: Adaptive Gradient Methods with Dynamic Bound of Learning Rate [10]

At each iteration, *stochastic gradient descent* samples a selection of summand functions in order to reduce the computational cost. In the case of large-scale machine learning problems, this is particularly effective.

Stochastic gradient descent with momentum remembers the update of deltas at each iteration, and determines the next update as a linear combination of the gradient and the previous update. **Adam** [6] is also an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.

[11] presents **AdaBound** as an optimizer that behaves like Adam at the beginning of training, and gradually transforms to SGD at the end, therefore considered by the author an optimizer that trains as fast as Adam and as good as SGD.

3.4 Pruning techniques

Deep learning approaches that are now in use rely on over-parametrized models that are difficult to implement. Identifying the best strategies for compressing models by minimizing the number of parameters in them is critical in order to save memory, power, and hardware without sacrificing accuracy, deploy lightweight models on devices, and ensure privacy using private on-device computing. Pruning is used to research the function of fortunate sparse subnetworks and initializations (known as lottery tickets) as a destructive neural architecture search tool, as well as to analyze the variations in learning dynamics between over-parametrized and under-parametrized networks [2].

Model pruning is a strategy for shrinking the size of a deep learning model by discovering and removing tiny weights from the model. Model pruning may significantly reduce model size while also speeding up inference time.

Weights are removed on a case-by-case basis in unstructured pruning methods. Weights are removed in groups using structured pruning methods, such as removing entire channels at a time. Structured pruning offers better runtime speed (due to the compact computation on fewer channels), but it has a greater influence on model correctness (due to the fact that it is less selective) [1].

In the global unstructured pruning, instead of specifying individual pruning parameters for individual layers, we apply to the model all at once [2].

Pruning can be added as an extra step in between training epochs (*iterative pruning*) or done all at once once model training is complete (*one-shot pruning*).

4 Experiment Setup

The implementation on which this benchmark of neural network models with reduced number of parameters the task of image classification has been done is available open-source:

`https://github.com/raresraf/benchmark-dnn-small`

We used pyTorch, an open source machine learning framework for the python programming language to implement and test our models. Also, we have used WandB, a central dashboard to keep track of your hyper parameters, system metrics, and predictions of our models.

5 Results

This section will present the results of our research, for the ResNet, VGG and ViT architectures.

5.1 ResNet spotlight

We want to dedicate one entire subsection for analysing some experimental results where we tested a classic **ResNet18** architecture, as well as a lightweight version, as we have described it as **ResNet18_S** in **Appendix A**, with different optimizers.

ResNet18 obtained **94.13** % accuracy while testing on the CIFAR-10 dataset when mixed with AdaBound optimizer. Interesting, the second best result was obtained by the **ResNet18_S** network with AdaBound optimized, outclassing the **ResNet18** architecture performance when combined with SGDM or ADAM optimizers. Also, it is worth noticing that the **ResNet18_S** network with ADAM optimizer performed almost as good as the **ResNet18** SGDM.

Architecture	Optimizer	Test accuracy
ResNet18	*SGDM*	89.4
	ADAM	89.97
	AdaBound	**94.13**
ResNet18_S	*SGDM*	86.85
	ADAM	89.27
	AdaBound	91.72

Table 2: The test accuracy, on the **CIFAR-10** dataset, for multiple configurations of the ResNet networks architectures and used optimizers.

For the ResNet18 and ResNet18_S architectures confusion matrix results on the CIFAR-10 dataset please refer to Appendix D.

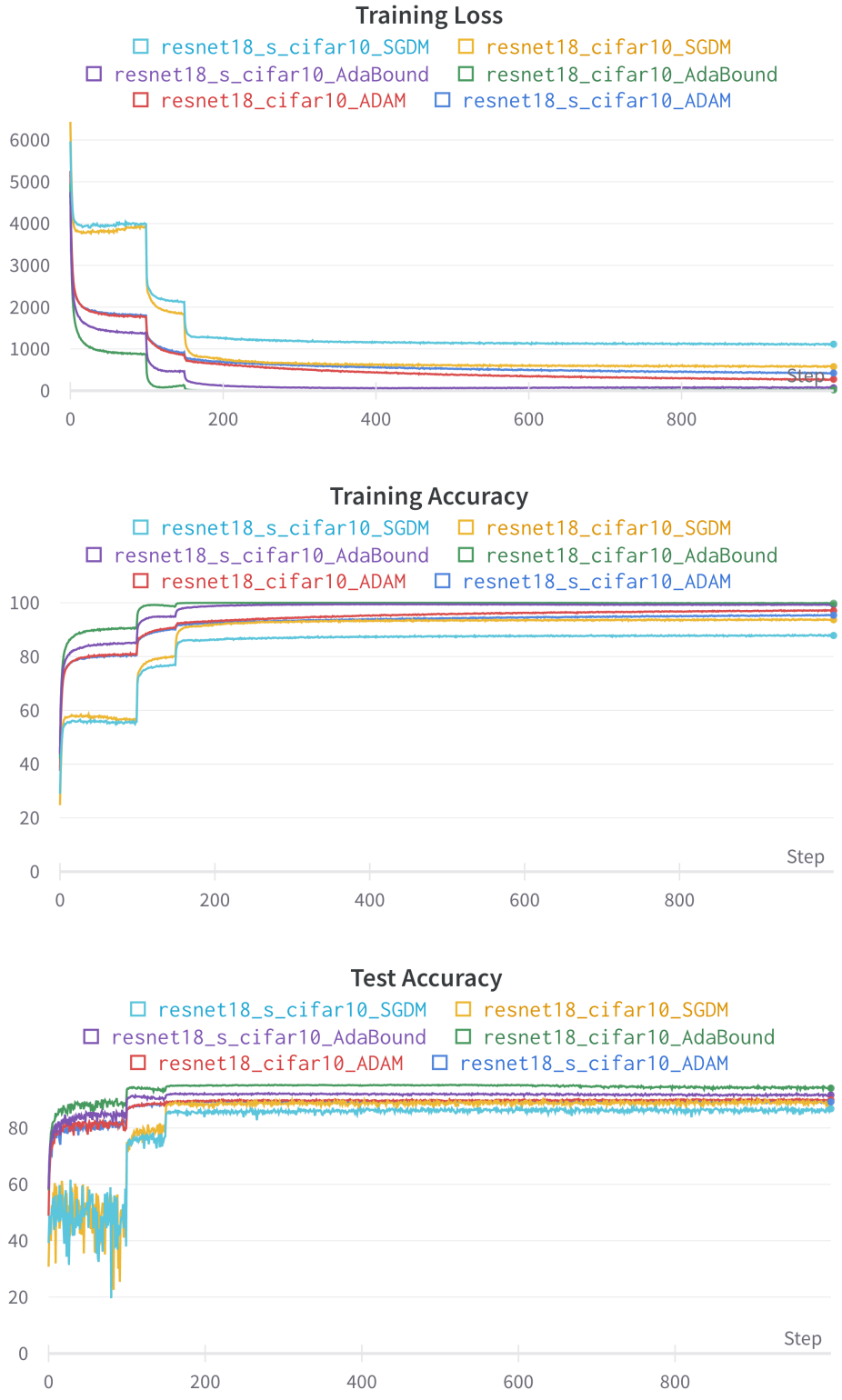
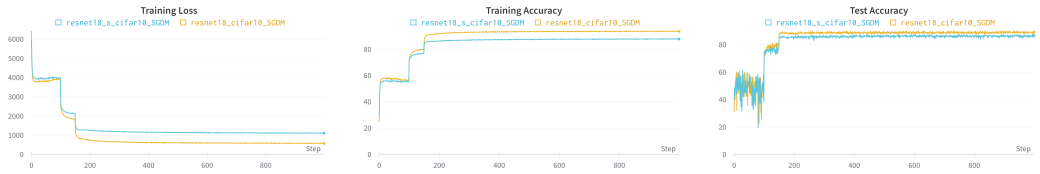
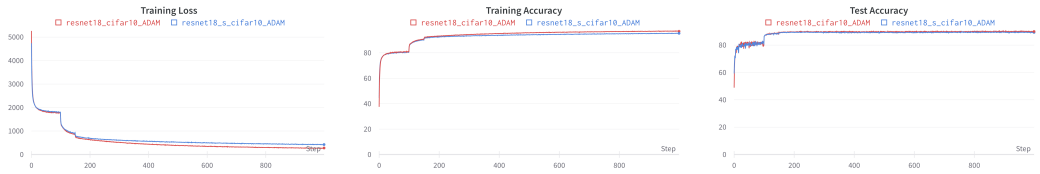


Figure 8: Overall view on the training loss error, training accuracy and testing accuracy obtained on the CIFAR-10 dataset obtained when using different optimizers, such as **SGDM**, **ADAM** and **AdaBound**, during 1000-epoch training of ResNet18/ResNet18_S networks.



(a) SGDM



(b) ADAM

(c) AdaBound

Figure 9: The training loss error, training accuracy and testing accuracy obtained on the CIFAR-10 dataset obtained when using the optimizer: (a) **SGDM** (b) **ADAM** (c) **AdaBound**, during 1000-epoch training of ResNet18/ResNet18_S networks.

We ran a similar scenario for the CIFAR-100 dataset, training the same networks on a 100-epoch scenario with various optimizers. The results showed that best accuracy (58%) has been obtained while using a ResNet18 with AdaBound optimizer, followed with a 4% decrease in quality by the ResNet18_S architecture, with the same optimizer. Interesting enough, the latter performed better than the traditional ResNet18 architecture while using ADAM optimizer, with more than 4.2%.

Architecture	**Optimizer**	**Test accuracy**
ResNet18	*SGDM*	1
	ADAM	39.04
	AdaBound	**58**
ResNet18_S	*SGDM*	1
	ADAM	43.31
	AdaBound	54.05

Table 3: The test accuracy, on the **CIFAR-100** dataset, for multiple configurations of the ResNet networks architectures and used optimizers.

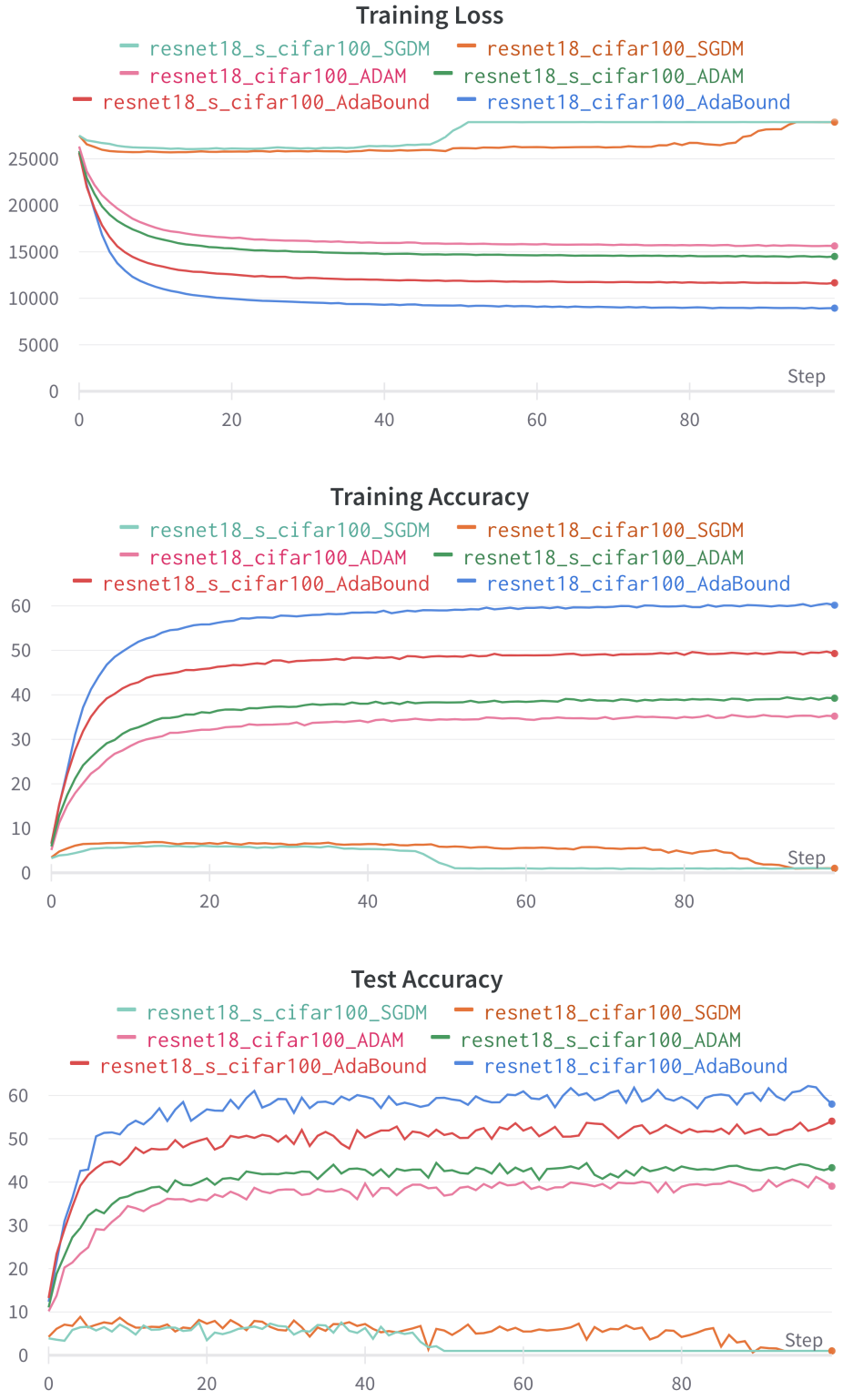
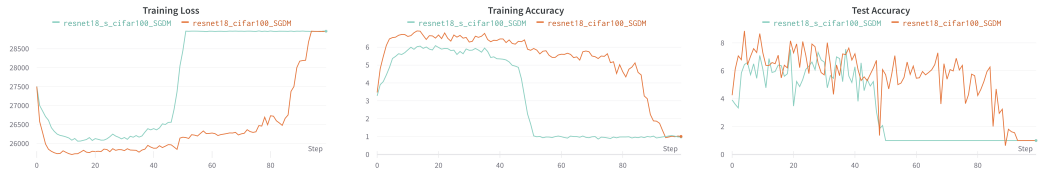
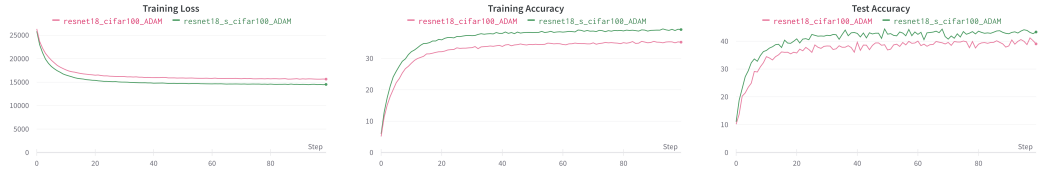


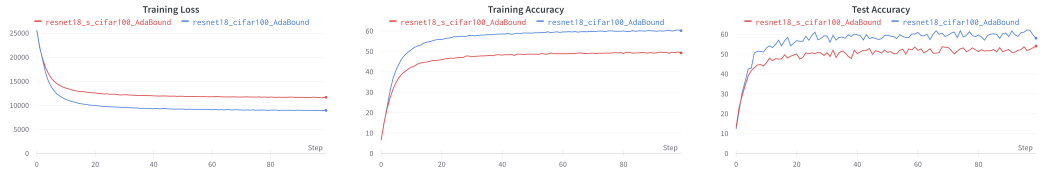
Figure 10: Overall view on the training loss error, training accuracy and testing accuracy obtained on the CIFAR-100 dataset obtained when using different optimizers, such as **SGDM**, **ADAM** and **AdaBound**, during 100-epoch training of ResNet18/ResNet18_S networks.



(a) SGDM



(b) ADAM

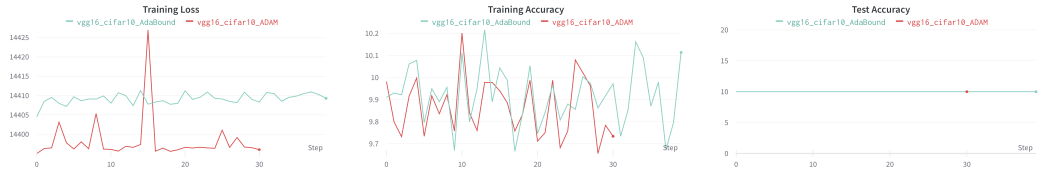


(c) AdaBound

Figure 11: The training loss error, training accuracy and testing accuracy obtained on the CIFAR-100 dataset obtained when using the optimizer: (a) **SGDM** (b) **ADAM** (c) **AdaBound**, during 100-epoch training of ResNet18/ResNet18_S networks.

5.2 VGG

The results for the VGG architectures were inconclusive. So far, we were not able to induce the VGG-16 architecture into a training mode that would make meaningful learning progress.



(a) VGG-16

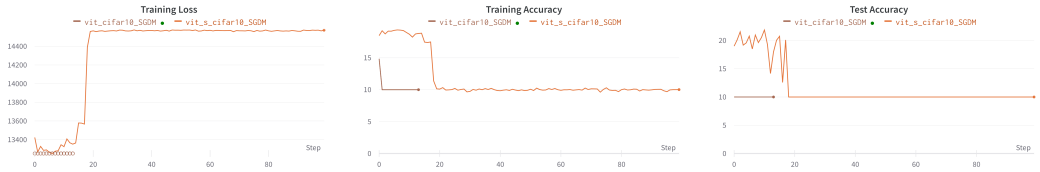
Figure 12: The training loss error, training accuracy and testing accuracy obtained on the CIFAR-10 dataset for the VGG16 architecture during the preliminary training.

5.3 Vision transformers

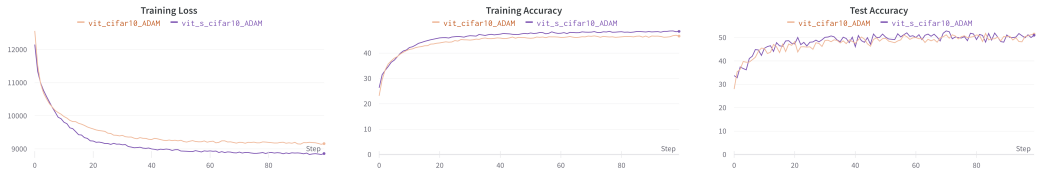
The Vision Transformer is a model for image classification that employs a Transformer-like architecture over patches of the image.

Architecture	Optimizer	Test accuracy
ViT	*SGDM*	10
	ADAM	**51.49**
	AdaBound	48.38
ViT_S	*SGDM*	10
	ADAM	50.99
	AdaBound	49.44

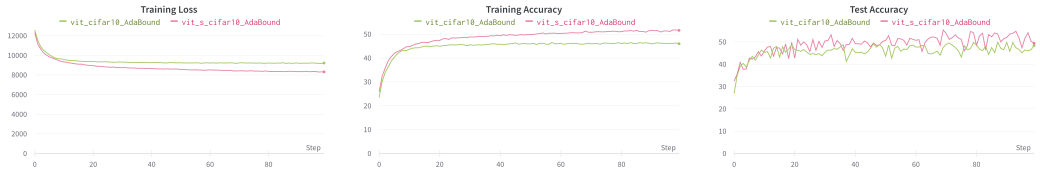
Table 4: The test accuracy on the **CIFAR-10** dataset, for multiple configurations of network architectures and used optimizers for the ViT architecture. The SotA ViT model reaches over 99% accuracy on the **CIFAR-10** dataset.



(a) SGDM



(b) ADAM



(c) AdaBound

Figure 13: The training loss error, training accuracy and testing accuracy obtained on the CIFAR-10 dataset obtained when using the optimizer: (a) **SGDM** (b) **ADAM** (c) **AdaBound**, during 1000-epoch training of ResNet18/ResNet18_S networks.

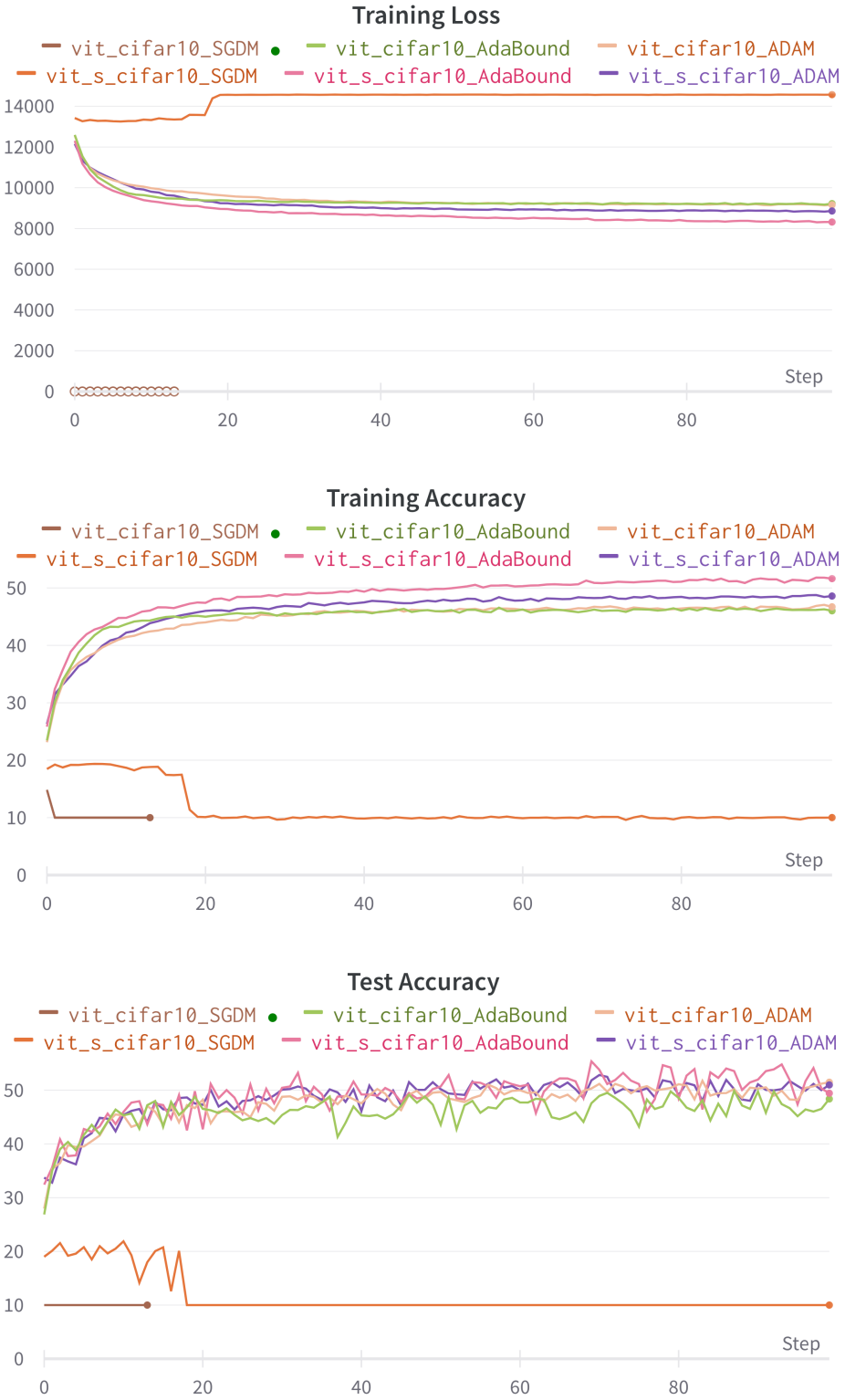
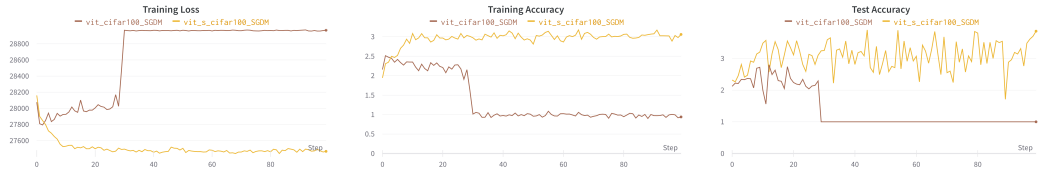


Figure 14: Overall view on the training loss error, training accuracy and testing accuracy obtained on the CIFAR-10 dataset when using different optimizers, such as **SGDM**, **ADAM** and **AdaBound**, during 100-epoch training of ViT/ViT_S networks.

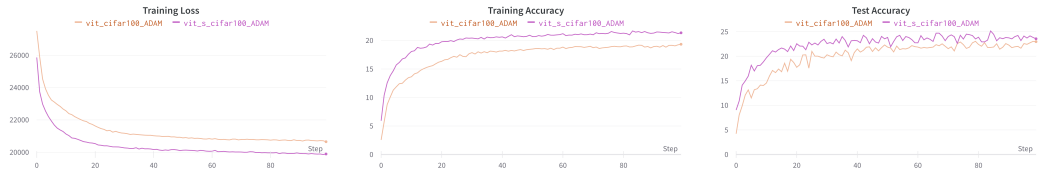
We ran a similar scenario for the CIFAR-100 dataset, training the same vision transformers on a 100-epoch scenario with various optimizers.

Architecture	Optimizer	Test accuracy
ViT	*SGDM*	1
	ADAM	22.97
	AdaBound	**25.05**
ViT_S	*SGDM*	3.86
	ADAM	23.56
	AdaBound	21.56

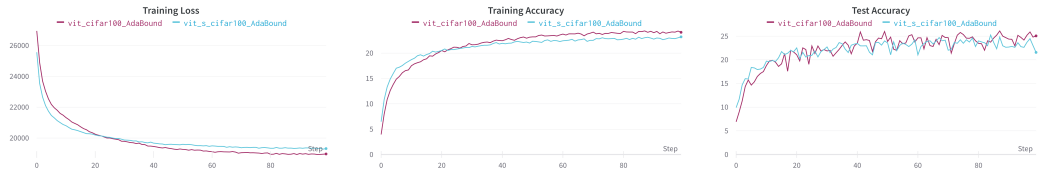
Table 5: The test accuracy, on the **CIFAR-100** dataset, for multiple configurations of the ResNet networks architectures and used optimizers.



(a) SGDM



(b) ADAM



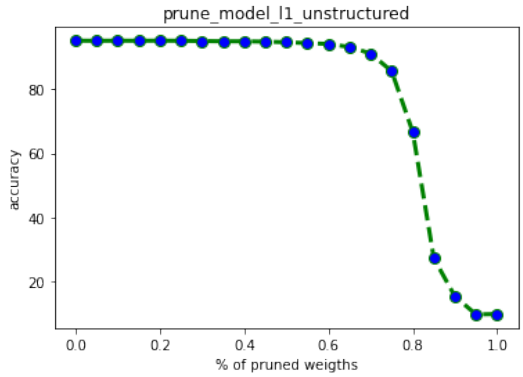
(c) AdaBound

Figure 16: The training loss error, training accuracy and testing accuracy obtained on the CIFAR-100 dataset obtained when using the optimizer: (a) **SGDM** (b) **ADAM** (c) **AdaBound**, during 100-epoch training of ResNet18/ResNet18_S networks.

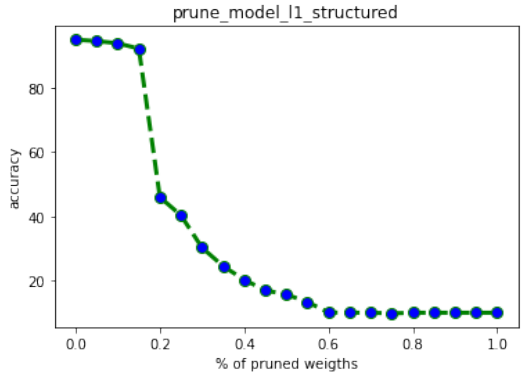
Figure 15: Overall view on the training loss error, training accuracy and testing accuracy obtained on the CIFAR-100 dataset obtained when using different optimizers, such as **SGDM**, **ADAM** and **AdaBound**, during 100-epoch training of the visual transformers networks.

5.4 Experimenting with pruning techniques for the ResNet architecture

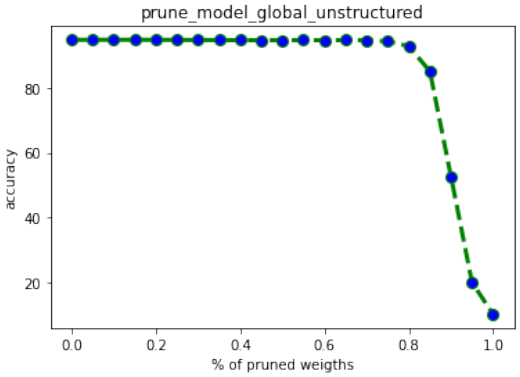
Using the trained ResNet18 network on the CIFAR-10 dataset, we applied various pruning techniques, such as random pruning, l1 structured and unstructured pruning and global pruning. The detailed results can be referred in **Appendix E**.



(a) L1 unstructured



(b) L1 structured



(c) Global unstructured

Figure 17: Overall view on the testing accuracy obtained on the CIFAR-10 dataset obtained when using different pruning techniques on the ResNet18 network.

6 Conclusion

Our research has investigated whether we can use traditional deep neural architectures and achieve good performance with a reduced number of trainable parameters. For this, we have used both architectural changes and pruning techniques.

The optimizers play a key role in obtaining the desired high-accuracy for the deep neural networks analyzed. A network with tens of millions of trainable parameters may not perform any better than a smaller replica, with fewer trainable parameters but a different optimizer.

ResNet18 obtained **94.13** % accuracy on the CIFAR-10 dataset when mixed with AdaBound. Interesting, the second best result was obtained by the **ResNet18_S** network with AdaBound optimized, outclassing the **ResNet18** architecture performance when combined with SGDM or ADAM optimizers. With less than 3% accuracy decrease, the ResNet18_S model managed to classify correctly the class of the image in over than 91% with the Adabound optimizer, while only using 700k parameters (down from over 10M, as the original ResNet18 architecture was initially proposed with).

We ran a similar scenario for the CIFAR-100 dataset, training the same networks on a 100-epoch scenario with various optimizers. The results showed that best accuracy (58%) has been obtained while using a ResNet18 with AdaBound optimizer, followed with a 4% decrease in quality by the ResNet18_S architecture, with the same optimizer. Interesting enough, the latter performed better than the traditional ResNet18 architecture while using ADAM optimizer, with more than 4.2%.

The vision transformers that we have analyzed were relatively small wrt. the total number of parameters as opposed to the SotA vision transformers that obtain benchmark results on the two datasets. As a result, the top performance for these transformers on the CIFAR-10 dataset was just shy over 50%, effectively being outclassed by the traditional ResNet architectures.

The pruning techniques proved also to be useful, and by using L1 local and global unstructured pruning, we obtained high accuracy models with less than 50% of the parameters, with only 2% loss in accuracy by only using 20% of the parameters. However, the target of pruning a ResNet18 network to obtain a good accuracy with less than 1M parameters did not succeed.

The target of making architectural changes to a ResNet18 network to obtain a good accuracy with less than 1M parameters did succeed, as the ResNet18_S model obtained an accuracy almost as good as the well known ResNet18.

References

[1] A developer-friendly guide to model pruning in pytorch, Dec 2020.

[2] Pruning tutorial, 2020.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[4] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Kamil Klosowski. Image recognition on cifar10 dataset using resnet18 and keras. 2018.

[8] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[9] Yanli Liu, Yuan Gao, and Wotao Yin. An improved analysis of stochastic gradient descent with momentum. *arXiv preprint arXiv:2007.07989*, 2020.

[10] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

[11] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. In *Proceedings of the 7th International Conference on Learning Representations*, New Orleans, Louisiana, May 2019.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[13] Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning actions for monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[14] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[15] Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W Ronny Huang, and Tom Goldstein. Gradinit: Learning to initialize neural networks for stable and efficient training. *arXiv preprint arXiv:2102.08098*, 2021.

A ResNet18_S architecture

The architecture of the **ResNet18_S** deep neural network used for the benchmark is as follows:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 32, 32]             432
       BatchNorm2d-2           [-1, 16, 32, 32]              32
            Conv2d-3           [-1, 16, 32, 32]           2,304
       BatchNorm2d-4           [-1, 16, 32, 32]              32
            Conv2d-5           [-1, 16, 32, 32]           2,304
       BatchNorm2d-6           [-1, 16, 32, 32]              32
        BasicBlock-7           [-1, 16, 32, 32]               0
            Conv2d-8           [-1, 16, 32, 32]           2,304
       BatchNorm2d-9           [-1, 16, 32, 32]              32
           Conv2d-10           [-1, 16, 32, 32]           2,304
      BatchNorm2d-11           [-1, 16, 32, 32]              32
       BasicBlock-12           [-1, 16, 32, 32]               0
           Conv2d-13           [-1, 32, 16, 16]           4,608
      BatchNorm2d-14           [-1, 32, 16, 16]              64
           Conv2d-15           [-1, 32, 16, 16]           9,216
      BatchNorm2d-16           [-1, 32, 16, 16]              64
           Conv2d-17           [-1, 32, 16, 16]             512
      BatchNorm2d-18           [-1, 32, 16, 16]              64
       BasicBlock-19           [-1, 32, 16, 16]               0
           Conv2d-20           [-1, 32, 16, 16]           9,216
      BatchNorm2d-21           [-1, 32, 16, 16]              64
           Conv2d-22           [-1, 32, 16, 16]           9,216
      BatchNorm2d-23           [-1, 32, 16, 16]              64
       BasicBlock-24           [-1, 32, 16, 16]               0
           Conv2d-25            [-1, 64, 8, 8]          18,432
      BatchNorm2d-26            [-1, 64, 8, 8]             128
           Conv2d-27            [-1, 64, 8, 8]          36,864
      BatchNorm2d-28            [-1, 64, 8, 8]             128
           Conv2d-29            [-1, 64, 8, 8]           2,048
      BatchNorm2d-30            [-1, 64, 8, 8]             128
       BasicBlock-31            [-1, 64, 8, 8]               0
           Conv2d-32            [-1, 64, 8, 8]          36,864
      BatchNorm2d-33            [-1, 64, 8, 8]             128
           Conv2d-34            [-1, 64, 8, 8]          36,864
      BatchNorm2d-35            [-1, 64, 8, 8]             128
       BasicBlock-36            [-1, 64, 8, 8]               0
           Conv2d-37           [-1, 128, 4, 4]          73,728
      BatchNorm2d-38           [-1, 128, 4, 4]             256
           Conv2d-39           [-1, 128, 4, 4]         147,456
      BatchNorm2d-40           [-1, 128, 4, 4]             256
           Conv2d-41           [-1, 128, 4, 4]           8,192
      BatchNorm2d-42           [-1, 128, 4, 4]             256
       BasicBlock-43           [-1, 128, 4, 4]               0
           Conv2d-44           [-1, 128, 4, 4]         147,456
      BatchNorm2d-45           [-1, 128, 4, 4]             256
           Conv2d-46           [-1, 128, 4, 4]         147,456
      BatchNorm2d-47           [-1, 128, 4, 4]             256
```

```
        BasicBlock-48            [-1, 128, 4, 4]               0
            Linear-49                   [-1, 10]           1,290
================================================================
Total params: 701,466
Trainable params: 701,466
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.81
Params size (MB): 2.68
Estimated Total Size (MB): 5.50
----------------------------------------------------------------
```

B VGG16_S architecture

The architecture of the **VGG16_S** deep neural network used for the benchmark is as follows:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 32, 32]             448
              ReLU-2          [-1, 16, 32, 32]               0
            Conv2d-3          [-1, 16, 32, 32]           2,320
              ReLU-4          [-1, 16, 32, 32]               0
         MaxPool2d-5          [-1, 16, 16, 16]               0
            Conv2d-6          [-1, 32, 16, 16]           4,640
              ReLU-7          [-1, 32, 16, 16]               0
            Conv2d-8          [-1, 32, 16, 16]           9,248
              ReLU-9          [-1, 32, 16, 16]               0
        MaxPool2d-10            [-1, 32, 8, 8]               0
           Conv2d-11            [-1, 64, 8, 8]          18,496
             ReLU-12            [-1, 64, 8, 8]               0
           Conv2d-13            [-1, 64, 8, 8]          36,928
             ReLU-14            [-1, 64, 8, 8]               0
           Conv2d-15            [-1, 64, 8, 8]          36,928
             ReLU-16            [-1, 64, 8, 8]               0
        MaxPool2d-17            [-1, 64, 4, 4]               0
           Conv2d-18           [-1, 128, 4, 4]          73,856
             ReLU-19           [-1, 128, 4, 4]               0
           Conv2d-20           [-1, 128, 4, 4]         147,584
             ReLU-21           [-1, 128, 4, 4]               0
           Conv2d-22           [-1, 128, 4, 4]         147,584
             ReLU-23           [-1, 128, 4, 4]               0
        MaxPool2d-24           [-1, 128, 2, 2]               0
           Conv2d-25           [-1, 128, 2, 2]         147,584
             ReLU-26           [-1, 128, 2, 2]               0
           Conv2d-27           [-1, 128, 2, 2]         147,584
             ReLU-28           [-1, 128, 2, 2]               0
           Conv2d-29           [-1, 128, 2, 2]         147,584
             ReLU-30           [-1, 128, 2, 2]               0
        MaxPool2d-31           [-1, 128, 1, 1]               0
          Dropout-32                 [-1, 128]               0
           Linear-33                 [-1, 128]          16,512
             ReLU-34                 [-1, 128]               0
          Dropout-35                 [-1, 128]               0
           Linear-36                 [-1, 128]          16,512
             ReLU-37                 [-1, 128]               0
           Linear-38                  [-1, 10]           1,290
================================================================
Total params: 955,098
Trainable params: 955,098
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.12
Params size (MB): 3.64
Estimated Total Size (MB): 4.78
```

```
377 ------------------------------------------------------------------
```

C ViT_S architecture

The architecture of the **ViT_S** transformer used for the benchmark is as follows:

```
ViT(
    image_size = 32,
    patch_size = 4,
    num_classes = 10,
    dim = 192,
    depth = 4,
    heads = 8,
    mlp_dim = 128,
    dropout = 0.1,
    emb_dropout = 0.1
)
```

Layer (type)	Output Shape	Param #
Linear-1	[-1, 64, 192]	9,408
Dropout-2	[-1, 65, 192]	0
LayerNorm-3	[-1, 65, 192]	384
Linear-4	[-1, 65, 576]	110,592
Linear-5	[-1, 65, 192]	37,056
Dropout-6	[-1, 65, 192]	0
Attention-7	[-1, 65, 192]	0
PreNorm-8	[-1, 65, 192]	0
Residual-9	[-1, 65, 192]	0
LayerNorm-10	[-1, 65, 192]	384
Linear-11	[-1, 65, 128]	24,704
GELU-12	[-1, 65, 128]	0
Dropout-13	[-1, 65, 128]	0
Linear-14	[-1, 65, 192]	24,768
Dropout-15	[-1, 65, 192]	0
FeedForward-16	[-1, 65, 192]	0
PreNorm-17	[-1, 65, 192]	0
Residual-18	[-1, 65, 192]	0
LayerNorm-19	[-1, 65, 192]	384
Linear-20	[-1, 65, 576]	110,592
Linear-21	[-1, 65, 192]	37,056
Dropout-22	[-1, 65, 192]	0
Attention-23	[-1, 65, 192]	0
PreNorm-24	[-1, 65, 192]	0
Residual-25	[-1, 65, 192]	0
LayerNorm-26	[-1, 65, 192]	384
Linear-27	[-1, 65, 128]	24,704
GELU-28	[-1, 65, 128]	0
Dropout-29	[-1, 65, 128]	0
Linear-30	[-1, 65, 192]	24,768
Dropout-31	[-1, 65, 192]	0
FeedForward-32	[-1, 65, 192]	0
PreNorm-33	[-1, 65, 192]	0
Residual-34	[-1, 65, 192]	0
LayerNorm-35	[-1, 65, 192]	384
Linear-36	[-1, 65, 576]	110,592

```
            Linear-37              [-1, 65, 192]          37,056
           Dropout-38              [-1, 65, 192]               0
         Attention-39              [-1, 65, 192]               0
           PreNorm-40              [-1, 65, 192]               0
          Residual-41              [-1, 65, 192]               0
         LayerNorm-42              [-1, 65, 192]             384
            Linear-43              [-1, 65, 128]          24,704
              GELU-44              [-1, 65, 128]               0
           Dropout-45              [-1, 65, 128]               0
            Linear-46              [-1, 65, 192]          24,768
           Dropout-47              [-1, 65, 192]               0
       FeedForward-48              [-1, 65, 192]               0
           PreNorm-49              [-1, 65, 192]               0
          Residual-50              [-1, 65, 192]               0
         LayerNorm-51              [-1, 65, 192]             384
            Linear-52              [-1, 65, 576]         110,592
            Linear-53              [-1, 65, 192]          37,056
           Dropout-54              [-1, 65, 192]               0
         Attention-55              [-1, 65, 192]               0
           PreNorm-56              [-1, 65, 192]               0
          Residual-57              [-1, 65, 192]               0
         LayerNorm-58              [-1, 65, 192]             384
            Linear-59              [-1, 65, 128]          24,704
              GELU-60              [-1, 65, 128]               0
           Dropout-61              [-1, 65, 128]               0
            Linear-62              [-1, 65, 192]          24,768
           Dropout-63              [-1, 65, 192]               0
       FeedForward-64              [-1, 65, 192]               0
           PreNorm-65              [-1, 65, 192]               0
          Residual-66              [-1, 65, 192]               0
       Transformer-67              [-1, 65, 192]               0
          Identity-68                  [-1, 192]               0
         LayerNorm-69                  [-1, 192]             384
            Linear-70                  [-1, 128]          24,704
              GELU-71                  [-1, 128]               0
           Dropout-72                  [-1, 128]               0
            Linear-73                   [-1, 10]           1,290
================================================================
Total params: 827,338
Trainable params: 827,338
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 6.76
Params size (MB): 3.16
Estimated Total Size (MB): 9.93
----------------------------------------------------------------
```

D ResNet18 and ResNet18_S architectures confusion matrix results on CIFAR-10 dataset

D.1 Resnet18_S with Adabound

	precision	recall	f1-score	support
plane	0.93	0.91	0.92	1000
car	0.96	0.96	0.96	1000
bird	0.89	0.90	0.89	1000
cat	0.81	0.83	0.82	1000
deer	0.94	0.91	0.93	1000
dog	0.87	0.86	0.87	1000
frog	0.95	0.94	0.95	1000
horse	0.95	0.94	0.94	1000
ship	0.94	0.95	0.94	1000
truck	0.95	0.95	0.95	1000
accuracy			0.92	10000
macro avg	0.92	0.92	0.92	10000
weighted avg	0.92	0.92	0.92	10000

Test acc 91.720

D.2 ResNet18 with Adabound

	precision	recall	f1-score	support
plane	0.95	0.96	0.95	1000
car	0.97	0.98	0.97	1000
bird	0.95	0.92	0.94	1000
cat	0.87	0.89	0.88	1000
deer	0.97	0.94	0.95	1000
dog	0.90	0.92	0.91	1000
frog	0.97	0.97	0.97	1000
horse	0.98	0.97	0.97	1000
ship	0.97	0.96	0.96	1000
truck	0.97	0.97	0.97	1000
accuracy			0.95	10000
macro avg	0.95	0.95	0.95	10000
weighted avg	0.95	0.95	0.95	10000

D.3 ResNet18_S with ADAM

	precision	recall	f1-score	support
plane	0.88	0.91	0.90	1000
car	0.93	0.96	0.94	1000
bird	0.85	0.84	0.85	1000
cat	0.81	0.79	0.80	1000
deer	0.89	0.89	0.89	1000
dog	0.84	0.86	0.85	1000
frog	0.90	0.94	0.92	1000
horse	0.94	0.91	0.92	1000

```
          ship       0.94      0.94      0.94      1000
         truck       0.94      0.93      0.93      1000

      accuracy                           0.89     10000
     macro avg       0.89      0.89      0.89     10000
  weighted avg       0.89      0.89      0.89     10000

Test acc 89.270
```

D.4 ResNet18 with ADAM

```
              precision    recall  f1-score   support

         plane       0.89      0.92      0.91      1000
           car       0.94      0.97      0.95      1000
          bird       0.88      0.84      0.86      1000
           cat       0.81      0.79      0.80      1000
          deer       0.90      0.89      0.90      1000
           dog       0.84      0.86      0.85      1000
          frog       0.90      0.94      0.92      1000
         horse       0.94      0.91      0.92      1000
          ship       0.94      0.94      0.94      1000
         truck       0.95      0.94      0.94      1000

      accuracy                           0.90     10000
     macro avg       0.90      0.90      0.90     10000
  weighted avg       0.90      0.90      0.90     10000

Test acc 89.970
```

E Detailed pruning results

Pruning technique / proportion of pruned connections	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45
L1 unstructured	95.0	94.97	95.0	94.98	95.01	94.97	94.89	94.82	94.81	94.71
L1 structured	95.0	94.56	93.87	92.22	45.96	40.3	30.24	24.51	20.25	16.99
Global unstructured	95.0	95.0	95.0	95.0	94.99	94.96	94.94	94.9	94.89	94.86

Table 6: The influence in accuracy based on the proportion of pruned connections in the ResNet18 network, against the CIFAR-10 dataset.

Pruning technique / proportion of pruned connections	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1
L1 unstructured	94.52	94.25	93.98	93.05	91.02	85.48	66.59	27.31	15.48	9.89	10.0
L1 structured	15.67	13.35	10.06	10.01	10.0	9.79	10.0	10.0	10.0	10.0	10.0
Global unstructured	94.87	94.9	94.86	94.91	94.87	94.55	93.01	85.24	52.73	19.85	10.0

Table 7: The influence in accuracy based on the proportion of pruned connections in the ResNet18 network, against the CIFAR-10 dataset. (contd.)