UNDER REVIEW

RESCIENCE C

Reproducibility report

Patches are All We Need?

Reviewed by
(Henok Birru)
(Husnul Abid)
(June Rose Manuzon)

Published
25th October

Reproducibility Summary

The in the first page, no exception will be allowed. When submitting your report in, copy the entire summary and paste it in the abstract input field, where the sections must be separated with a blank line.

Scope of Reproducibility — State the main claim(s) of the original paper you are trying to reproduce (typically the main claim(s) of the paper). This is meant to place the work in context, and to tell a reader the objective of the reproduction.
Transformer based models - best performers most especially vision transformers
Although convolutional networks have long dominated the architecture for vision tasks, current research indicates that Transformer-based models, most notably the Vision Transformer (ViT), may perform better in some circumstances. <add something here> In this paper, we investigate whether the high performance of vision transformers is due more to patch-based representation than to the Transformer architecture itself.

Methodology — Briefly describe what you did and which resources you used. For example, did you use author's code? Did you re-implement parts of the pipeline? You can use this space to list the hardware and total budget (e.g. GPU hours) for the experiments.

Results — Start with your overall conclusion — where did your results reproduce the original paper, and where did your results differ? Be specific and use precise language, e.g. "we reproduced the accuracy to within 1% of reported value, which supports the paper's conclusion that it outperforms the baselines".

What was easy — Describe which parts of your reproduction study were easy. For example, was it easy to run the author's code, or easy to re-implement their method based on the description in the paper.

What was difficult — Describe which parts of your reproduction study were difficult or took much more time than you expected. Perhaps the data was not available and you couldn't verify some experiments, or the author's code was broken and had to be debugged first.

Communication with original authors — Briefly describe how much contact you had with the original authors (if any).

The authors have declared that no competing interests exists.
Code is available at Google Colab.

1 Introduction

Convolutional Neural Networks (CNN) have been dominant in the computer vision field for a long time. However, architectures based on Transformers like Vision Transformer are also showing great promise. At times, they even managed to outperform classical CNNs like ResNet especially for large datasets. It is therefore acceptable to theorize that Transformer architectures could potentially outperform CNNs in the field of computer vision.

However, applying transformers to images is computationally expensive. Thus, the concept of patches were introduced in Vision Transformers. Patches are created from further breaking down images into tinier splits and creating linear embeddings of them. Transformers are then used on these patches instead of the individual pixels in an image.

Because patches were introduced as a workaround for the use of Transformers, the question of how much impact patches have individually on a network arises. To answer this, the researchers developed ConvMixer. Its architecture is similar to the Vision Transformer and MLP-Mixer, which ConvMixer was named after. The commonality between the three is their use of patches. However, what sets apart ConvMixer is that it only makes use of standard convolutions in its architecture.

2 Scope of reproducibility

- Despite the simplicity of ConvMixer architecture, it can outperform variants of Vision Transformer, MLP-Mixer, and standard computer vision models like ResNet
- Increasing kernel size improves accuracy of ConvMixer
- Increasing patches will decrease accuracy of ConvMixer

3 Methodology

We referred the authors' publicly available code from github to replicate their work. ConvMixer, their architecture, is implemented in PyTorch using the timm framework[1]. We used Tensorflow Keras to reimplement their code by following the guide created by Sayak Paul[2]. Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. Our code is available in google colab[3]. Tensor Processing Units(TPUs) are the selected runtime environment to run the code in the google colab notebook. They are Google's custom-developed application-specific integrated circuits(ASICs) used to accelerate machine learning workloads.

3.1 Model descriptions

Three versions of the ConvMixer model which areConvMixer-1536/20 with kernel size 9 and patch size 7, ConvMixer-768/32 with both kernel size and patch size 7 and ConvMixer-1024/20 with kernel size 9 and patch size 14 are provided in the documentation of the original authors' github readme, however to run the code with the resource we have, ConvMixer-256/8 is applied with kernel size 5 and patch size 2. The name of the models contain two numbers, ConvMixer-h/d, where h is the number of filters or hidden layers and d is the depth.

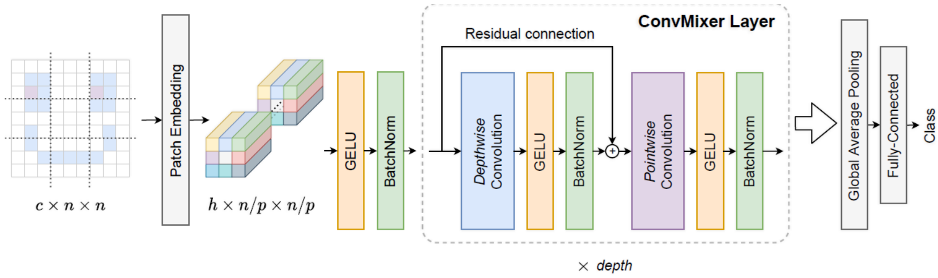

Figure 1. ConvMixer model

The ConvMixer model combines methods from other recent architectures, ViT and MLP-Mixer. Both of these architectures operate on patches, while ViT uses the state-of-art Transformer architecture, MLP-Mixer uses another concept called multi-layer perceptrons.

A patch embedding layer is followed by several applications of a simple fully-convolutional block in the ConvMixer architecture. The implementation of the model has an advantage of simplicity since it takes small amount of code to write it. The model uses standard convolutional layers. The first convolutional layer is the depthwise convolution and it is followed by pointwise convolution. The depthwise convolution is used to mix the spacial locations while pointwise convolution is used to mix channel locations.

ConvMixer's instantiation is dependent on four parameters: (1) the "width" or hidden dimension h (i.e., the dimension of the patch embeddings) , (2) the depth d, or the number of repetitions of the ConvMixer layer, (3) the patch size p which controls the internal resolution of the model, (4) the kernel size k of the depthwise convolutional layer[4].

3.2 Datasets

The CIFAR-10 dataset is used to train the model. The dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. Out of the 50000 training examples 10% is used for validation data samples. The original paper describes experiments on both ImageNet-1k and CIFAR-10 datasets.

Figure 2. CIFAR-10 dataset

3.3 Hyperparameters

Although several kernel sizes and patch sizes were used throughout the experiment, the authors of the original study noted that they did not look for excellent settings because their experiment was not intended to enhance accuracy or speed. In our experiment, we trained the model across 5 and 40 iterations using various kernel and patch sizes. We run the model with 256 hidden layers and 8 depth because of our restricted computing power. The results section discusses the values for the various parameters.

3.4 Experimental setup and code

To build up our experiment code, we adhere to Sayak Paul's ConvMixer keras demonstration's instructions[2]. Google colab is enough to write and run the code. To utilize the AdamW optimizer Tensorflow-addons package is loaded as well as tensorflow and tensorflow keras modules are imported. The experiments are evaluated based on accuracy. Code is available at Google Colab.

3.5 Computational requirements

write some thing heeereeeeeeeeeeeeeeeeeeeeee

4 Results

write some thing heeereeeeeeeeeeeeeeeeeeeeeeeeeeee
dfd
dfd
dfd

Width (h)	Depth (d)	Patch size (p)	Kernel size (k)	Params (10^3)	Original Accuracy (200 epochs)	Reproduction Accuracy (40 epochs)
256	8	1	3	559	93.61	82.8
256	8	2	9	709	95	80

Figure 3. Original and reproduction result 40 epochs

Width (h)	Depth (d)	Patch size (p)	Kernel size (k)	Params (10^3)	Original Accuracy (200 epochs)	Reproduction Accuracy (5 epochs)
256	8	1	5	592	95.19	82.48
256	8	1	9	707	95.88	81.24
256	8	4	9	718	92.61	70.62

Figure 4. Original and reproduction result with 5 epochs

write more heeeereeeeee
dfd
dfd
dfd

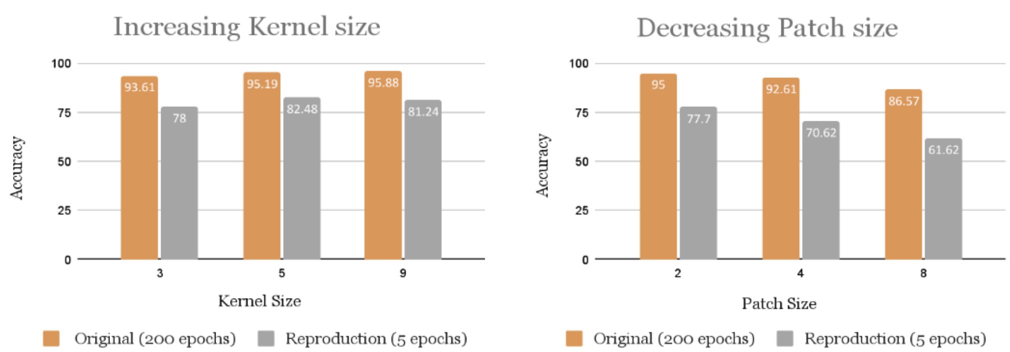

Figure 5. Effects of hyperparameter tuning

5 Discussion

In this paper, the key point authors claimed is that using patch embeddings is a powerful and important takeaway besides the architecture design. The described architecture is quite straightforward and consists of patching the input images before using a combination of depth-wise and point-wise convolutions. Authors classified the images to evaluate the performance of this model. They only used the ImageNet-1k dataset to train the architecture in their primary experiment. On that study, they have demonstrated how their straightforward design is competitive with cutting-edge ones.

According to our experiment on the CIFAR-10 dataset with fewer epochs, we found almost the similar result and pattern. The authors' investigations and findings are fully supported by ours replication results. In our opinion, ConvMixer„ might become even more competitive with further improvements, such as increasing the number of epochs, a more expressive classifier or refinement on the bottlenecks.

We have implemented the code in Google Colab which enables us to use resources from Google and utilize GPU to train the model. The implemented code is very simple and easy to understand due to the architecture of the model. One of the strength of our approach is experimenting on simultaneous google colab which allows us to shorten the time to get result. We have used Tensorflow to implement the model instead of Pytorch. TensorFlow offers the high-level Keras API which makes it simple to build and train models. we were ablel to experiment on only one dataset because of the lack of computational resource. However, we found very similar pattern like the authors while changing the Kernel and Patch size.

5.1 What was easy

The paper was well written with clear motivation and this helps us to understand the paper in details. The comparison with other research and experiments were described properly which helps us to analyze more on this topics. Additionally, implementation was smooth because of the simplicity of the model. Another positive aspect of this research is that they used ImageNet-1k and CIFAR-10 dataset. Both dataset is well known, extremely popular and available. The code was scalable and maintainable. Also it was well documented by the authors.

All the hyperparameters were easy to tune because of their explicit definition. Moreover, authors discussed about the changing of the hyperparameters in details. This helps us to understand the model in depth. We have also ran the model with similar parameters and able to see similar patterns with authors.

5.2 What was difficult

In the journey to reproduce, we face some difficulties that was hard to overcome. In some scenario, we tried decrease the needs of computational resource and time using hyperparameter tuning. There were no pre-trained model that required us to train the model everytime which is very time consuming. We have successfully implemented ConvMixer with the 256 filters. However, we were unable to train the model with larger filters with limited amount of resource. The training was extremely slow due to the low throughput. Throughput is the quantity of data units processed in a given amount of time. Also we couldn't produce result for 200 epochs because of the limited time.

5.3 Communication with original authors

The research work has done on January 2022 and since then this paper got a lot of attention to the community. In the paper with code, they have 13000 stars on the implementation. Due to our growing interest on this topic, we have emailed the main author asking about any further study has been done by them or the research community. This will help us to understand the possible areas to improve for future work.