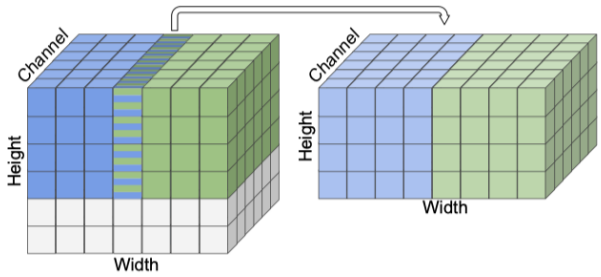
CPSC 440 (Jan-Apr 2024) Project Proposal

Jordan Bourak and Conor Brennan

March 27, 2024

Recently, Nasirigerdeh et. al [1] proposed KernelNorm, a normalization method that serves as a potential alternative to BatchNorm [2], GroupNorm [3], LayerNorm [4], and LocalContextNorm [5]. KernelNorm can be viewed as a pooling layer that normalizes each of the regions specified by the kernel, stride, and padding sizes during the pooling algorithm instead of summarizing them by max or mean. The mean and variance is calculated for a particular region, but taking into consideration *all channels at once*, and then each element is normalized accordingly. This means that depending on the kernel size, stride, and padding, the output of KernelNorm can be smaller or larger than the input. Furthermore, depending on these hyperparameters, some elements from the input may have multiple normalized counterparts in the output, and others may be left out entirely. A toy example from the original paper is shown below:



The authors found that using KernelNorm within the ResNet architecture over other normalization methods lead to higher or equal performance and faster convergence (in iterations) for semantic segmentation and classification across several experiments. KernelNorm has also been tested in privacy-preserving settings [6] and shown similar improvements. These benefits come at the cost of longer training and test times, however: up to a factor of 1.66 for the largest model tested, as well as higher memory usage.

The performance benefits of KernelNorm within many other well-known architectures remains mostly unknown. Hence, a natural progression of this finding is to experiment further with the KernelNorm counterparts for other architectures, such as DenseNet, as suggested by the authors. Specifically, we have identified 5 different architectures that we think are worthwhile to experiment with the use of KernelNorm over the commonly used normalization methods within each architecture:

- DenseNet: although DenseNets with KernelNorm have been tested, it was only in privacy-preserving settings [6] and not in a more "regular" setting. The implementations of KernelNorm-DenseNet from the paper are public on GitHub, but we will write our own non-DP non-FL version.
- ConvNeXt: the original authors briefly tested a Tiny variant of ConvNeXt in the original paper, but minimal time was spent attempting to optimize the architecture for KernelNorm specifically due to time constraints and resource availability (see author's OpenReview reply). As their implementation is publicly available, we will use it as the basis for our optimizations.

- ConvMixer: the original ConvMixer architecture uses BatchNorm instead of LayerNorm, as the authors found that using LayerNorm had a decrease in performance of 1% as well as slower convergence, but admit there may be situations in which other normalization methods are preferrable.
- Vision Transformer (ViT): since "LayerNorm can be considered as a special case of KernelNorm" (see author's OpenReview reply), ViTs could potentially be improved by incorporating KernelNorm over the commonly used LayerNorm.
- Conformer: because Conformers have a ResNet block and a ViT block, performance could be potentially improved by replacing BatchNorm and/or LayerNorm with KernelNorm

A PyTorch implementation of KernelNorm is available on GitHub, as well as a KNConv2d layer which is a combination of the normalization and then a 2d convolution. So, our plan is to modify the DenseNet, ConvMixer, ViT, and Conformer architectures to use KernelNorm as their normalization methods. If all goes well, our secondary goal is to optimize these architectures for KernelNorm to improve performance. Our contribution will be the comparison of these KernelNorm-modified architectures against their traditionally normalized counterparts to determine if KernelNorm is useful beyond the scope of the initial paper.

References

[1] Reza Nasirigerdeh, Reihaneh Torkzadehmahani, Daniel Rueckert, and Georgios Kaissis. Kernel normalized convolutional networks, 2024.

[2] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[3] Yuxin Wu and Kaiming He. Group normalization, 2018.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[5] Anthony Ortiz, Caleb Robinson, Dan Morris, Olac Fuentes, Christopher Kiekintveld, Md Mahmudulla Hassan, and Nebojsa Jojic. Local context normalization: Revisiting local normalization, 2020.

[6] Reza Nasirigerdeh, Javad Torkzadehmahani, Daniel Rueckert, and Georgios Kaissis. Kernel normalized convolutional networks for privacy-preserving machine learning. In *First IEEE Conference on Secure and Trustworthy Machine Learning*, 2023.