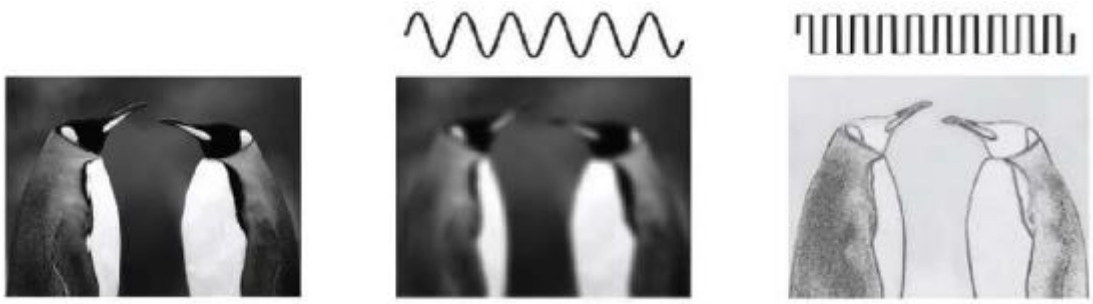
Proof of concept: Octave Convolution

1. Introduction

Convolutional Neural Networks (CNNs) are among the best techniques when it comes to computer vision tasks. And their efficiency keeps increasing with recent research efforts. Some of the state-of-the-art models may use techniques to reduce the inherent redundancy in dense model parameters or in the channel dimension of feature maps. However, none takes advantage of the redundancy in the spatial dimension of the feature maps produced by CNNs.

In general, an image can be decomposed in two parts: a low-frequency part that represents the smoothly changing global structure and a high-frequency part that represents the rapidly changing details. We can argue that this is also the case of the feature maps of a CNN.



The Octave Convolution studied in this project takes advantage of those multi-frequency components by providing a generic way to replace the vanilla convolution layer by a new Octave Convolution layer, called “OctConv”. This new operator stores the high- and low-frequency parts in different groups. Then, the spatial resolution of the low-frequency part can be safely reduced one octave lower without losing information.

The primary goal of this new operator was to reduce substantially memory consumption and computational resources. But it turns out in certain cases it can also improve performance.

In the original paper, experimental work was done on the ImageNet dataset and with a bunch of state-of-the-art network architectures. The goal here is to see if the performance improvements can be extended to other datasets and/or architectures. In particular, we will first try the new operator on the CIFAR-10 dataset and then on the Stanford Dogs Dataset - which was used on the last project to classify dog breeds.

The paper on which this project is based upon can be found here:

https://arxiv.org/pdf/1904.05049v2.pdf

2. Related work

2.1. Improving the efficiency of CNNs

Since the earlier CNNs like AlexNet (2012) or VGG (2014), based on stacking a set of convolutional layers, showed impressive results, a lot of work has been done to improve the efficiency of CNNs.

Some of the more impactful streams of work include:

- Improving the topology of the network by making shorter connections with early layers – ResNet (2015), DenseNet (2016).
- Using sparsely connected group convolutions or blocks to reduce redundancy between channels, making it feasible to go deeper or wider but with the same computational cost – ResNeXt (2016), ShuffleNet (2017).
- Using depth-wise convolutions to reduce the computational cost of classical convolutions by using a two-step convolution to limit the number of matrix multiplications – Xception (2016), MobileNet (2017).
- Automatically finding the best network architecture for the dataset at hand, capable of outperforming manually-designed networks – NAS (2017), PNAS (2017), AmoebaNet (2018).

2.2. Multi-scale feature representation

The idea of using a multi-scale representation is not new. Long before deep learning the SIFT algorithm was already using multi-scale representation for feature extraction. But even in deep learning the idea is not recent, some of the work done in this area include:

- Merging convolution features from different depths for object detection – FPN (2016), PSP (2016).
- Having multiple branches where each branch has its own spatial resolution – MSDNet (2017), HR-Nets (2019).
- Using the same idea but as a replacement of residual block for ResNet, which make it more flexible and easier to use – bL-Net (2018), ELASTIC-Net(2018).

Finally, the closest work to OctConv is Multi-grid CNNs (2016) (or MG-Conv), which propose an operator that can be integrated in the network and using multi-scale features. But, essentially the difference is a less efficient design to exchange inter-frequency information.

3. Method

In this section, we first describe what we mean by the octave feature representation before describing in details the new convolution operator OctConv. Finally, we talk about some implementation details.

3.1. Octave Feature Representation

A natural image can be factorized into a low-frequency signal that captures the global layout, and a high-frequency part that detects fine details. The idea behind the Octave Convolution is to say that this is also the case for the feature maps of convolutional layers.

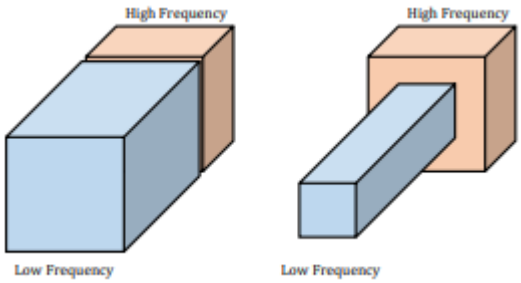
Usually for a convolution, all input and output feature maps have the same spatial dimensions. But here we will separate the feature maps into two groups (high- and low-frequencies), and treat the low-frequency part one octave lower – meaning its spatial dimensions are divided by 2. We can do that because the low-frequency evolves slowly throughout the image so there is a lot of spatial redundancy.

Formally, let $X \in \mathbb{R}^{c \times h \times w}$ denote the input feature tensor of a convolutional layer, where h and w are the spatial dimensions and c the number of feature maps (or channels).

Here, we say we can factorize X along the channel dimension into $X = \{X^H; X^L\}$, where the high-frequency feature maps are represented by $X^H \in \mathbb{R}^{(1-\alpha)c \times h \times w}$, and the low-frequency feature maps are represented by $X^L \in \mathbb{R}^{\alpha c \times \frac{h}{2} \times \frac{w}{2}}$.

We can see the apparition of a parameter $\alpha \in [0,1]$ that represents the proportion of low-frequency channels in the new convolution. These feature maps are represented one octave lower.

As we can see on this figure, the new reprensentation is more compact than the original one, by a factor of 4 on the low-frequency part.

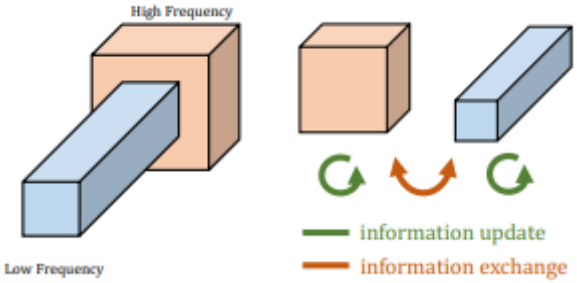


3.2. Octave Convolution

We now have feature maps factorized into two groups but the vanilla convolution cannot directly operate on such a representation, because of the different dimensions of the two groups.

One way we could make it work is by up-sampling the low-frequency part to the original dimensions, concatenate it with the high-frequency part and then convolve, but this would lead to extra costs in computation and memory and diminish all the savings done by the compression.

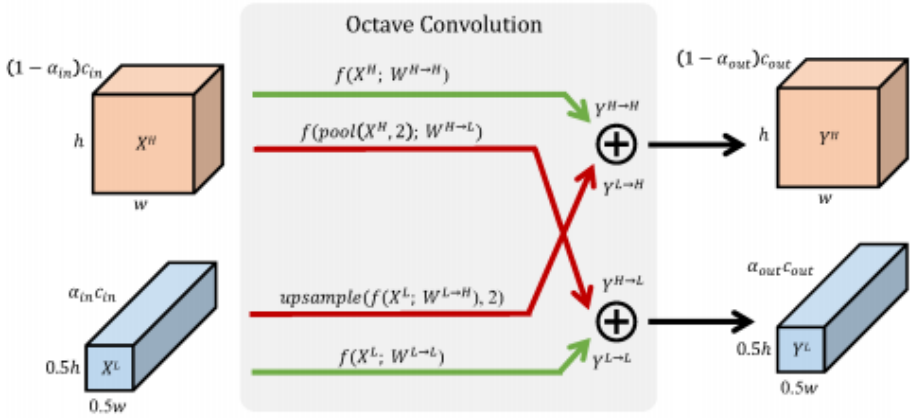
However, there is a way to fully exploit the feature representation, by using the Octave Convolution, that can directly operate on factorized tensors $X = \{X^H; X^L\}$ without any overhead. The goal here is to be able to update correctly the intra-frequency information, but also to exchange inter-frequency information efficiently.



The high-frequency output will be the addition of a convolution on the high-frequency input with an up-sampling of a convolution on the low-frequency input. And the low-frequency output will be the addition of a convolution on the low-frequency input with a convolution after a pooling of the high-frequency input.

Formally, let X, Y be the factorized input and output tensors. Then the high- and low-frequency feature maps of the output $Y = \{Y^H, Y^L\}$ will be given by $Y^H = Y^{H\to H} + Y^{L\to H}$ and $Y^L = Y^{L\to L} + Y^{H\to L}$, where $Y^{A\to B}$ denotes the convolution update from feature map group A to group B.

With this $Y^{L\to L}$ and $Y^{H\to H}$ denote the intra-frequency information, while $Y^{L\to H}$ and $Y^{H\to L}$ denote the inter-frequency communication.



Detailed design of the Octave Convolution. Green arrows correspond to information updates while red arrows facilitate information exchange between the two frequencies.

To be able to compute these terms, we need to split the convolutional kernel W into two components $W = [W^H, W^L]$ responsible for convolving with X^H and X^L respectively. Each component can be further divided into intra- and inter- components: $W^H = [W^{H\to H} + W^{L\to H}]$ and $W^L = [W^{L\to L} + W^{H\to L}]$.

Finally, we can rewrite the output of the Octave Convolution like this:

$$Y^H = f(X^H, W^{H\to H}) + upsample(f(X^H, W^{L\to H}), 2)$$

$$Y^L = f(X^L, W^{L\to L}) + f(pool(X^H, 2), W^{H\to L})$$

3.3. Implementation Details

Throughout the network we set $\alpha_{in} = \alpha_{out} = \alpha$, except for the first and last convolution.

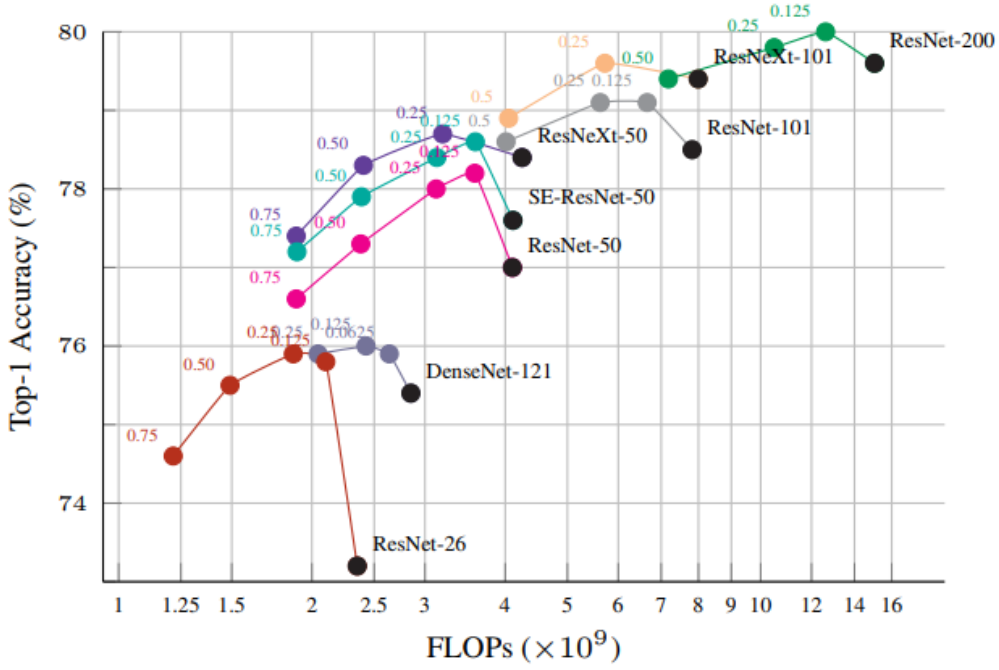
For the first convolution, we need to convert a vanilla feature representation to a two-frequency representation, we set $\alpha_{in} = 0$ and $\alpha_{out} = \alpha$. In this case, the low-frequency input is disabled.

For the last convolution, we need to go back from a multi-frequency representation to a vanilla representation, so we set $\alpha_{in} = \alpha$ and $\alpha_{out} = 0$. In this case, the low-frequency output is disabled.

3.4. Results on ImageNet

In the publication, several state-of-the-art CNNs architectures are tested on the ImageNet dataset with several values for alpha. The Octave Convolution showed consistent improvement of the computational cost (FLOPs) and also an improvement in the accuracy for some values of alpha versus the baseline (alpha=0).

The FLOPs-Accuracy trade-off is in a concave curve and the performance peak appears at alpha = 0.125 or 0.25.



4. Experimental Evaluation

The goal of this project is to see if the Octave Convolution is generalizable as a good replacement of vanilla convolution. In particular, we want to see if we can extend its good results to other datasets and/or architectures.

4.1. Datasets

We worked with two datasets: CIFAR-10 and Stanford Dogs Dataset.

The CIFAR-10 dataset is comprised of 60,000 images (50,000 training / 10,000 validation) of dimensions 32x32 representing 10 different classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck).

The second dataset we will use is the Stanford Dogs Dataset, which is the one we used in the last project. It is a classification dataset of dog breeds. It is comprised of 20,580 images over 120 breeds. We resized the images to 32x32 and did a train/test split of 80%/20%. For some architectures we only took a subset of breeds due to the very poor performance with all breeds.

4.2. Architectures

We tested the Octave Convolution over three architectures, very different from the ones in the publication. These were very complex state-of-the-art networks and here we only tested three basic structures.

First a 7-layers CNN, close to the one used in the previous project, and kind of similar to VGG architectures but simpler. It is comprised of 5 convolution layers followed by 2 fully-connected layers.

The next one is a bit deeper with 9 layers, we added one convolution layer and one fully-connected layer.

And the last one is based on the “Network in Network” model, where the idea is to add a multi-layer perceptron after each convolutional layer (which is called a MlpConv layer), and then do a global average pooling instead of fully-connected layers at the end. This take on convolutional filter design inspired the Inception line of deep architectures from Google. Here we used a network comprised of 3 MlpConv followed by the global average pooling.

4.3. Implementation

The first step was to find a working implementation of the convolution. Here is the one we used: https://github.com/koshian2/OctConv-TFKeras. It is an unofficial implementation that runs under TensorFlow / Keras. As an exemple on the repository, this implementation is run on the CIFAR-10 dataset with a Wide ResNet network (N=4, k=10).

We confirmed the implementation was working by double checking the results with this example. That led us to use the CIFAR-10 dataset for later experiments.

Alpha	Test accuracy (on repository)	Test accuracy (ours)
0	88.47 %	88.22 %
0.125		94.64 %
0.25	94.83 %	94.53 %
0.5	94.40 %	93.64 %
0.75	93.54 %	92.50 %

We can see that we get consistent results and thus can validate this implementation.

4.4. Results with CIFAR-10

With the 7-layers CNN (Figure 1), we can see that better performance is achieved without the Octave Convolution (alpha=0). Also, we observe a decrease in performance with an increase in alpha (this is consistent with the publication results) but for low values of alpha (0.125 or 0.25) we do not see a peak in performance.

With the 9-layers CNN (Figure 2), the experiment is not conclusive because the network is performing poorly with this dataset, even without the Octave Convolution.

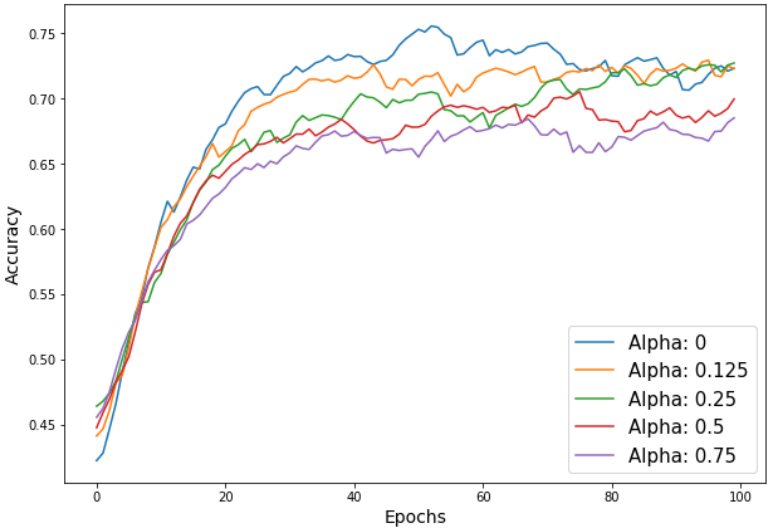


Figure 1: Test accuracy with 7-layers CNN (smoothed curve)

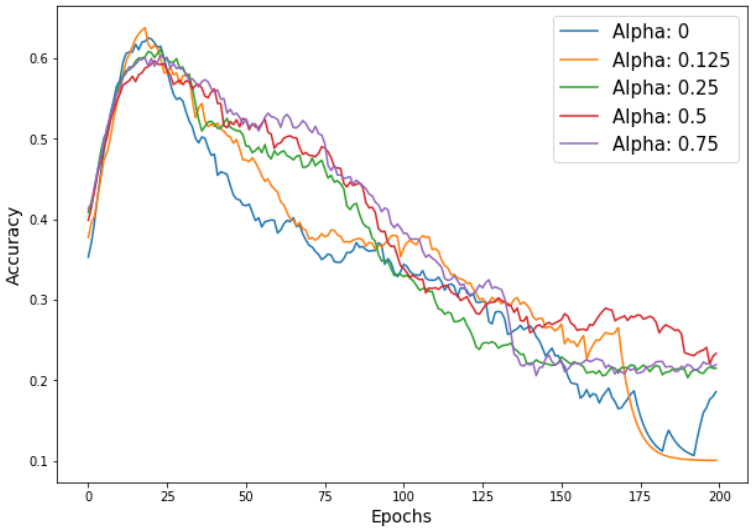


Figure 2: Test accuracy with 9-layers CNN (smoothed curve)

With the Network in Network model, or NiN (Figure 3), we see an impressive improvement when using the Octave Convolution, with a gain over 20 % in accuracy versus the baseline. However, it is hard to distinguish the performance between different values of alpha.

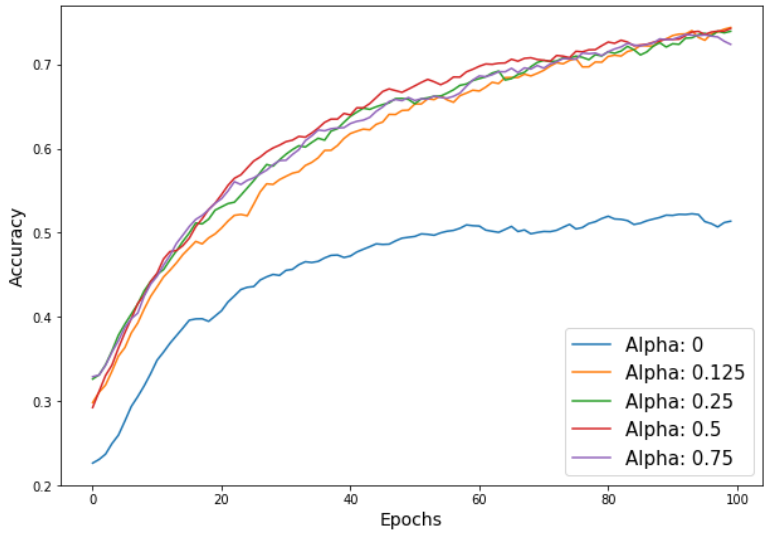


Figure 3: Test accuracy with NiN (smoothed curve)

Figure 4 shows the Accuracy-FLOPs trade-off curve – similar to the one used in the research paper. The FLOPs are determined theoretically, and the accuracy shown for each alpha is the maximum over the 100-epochs training.

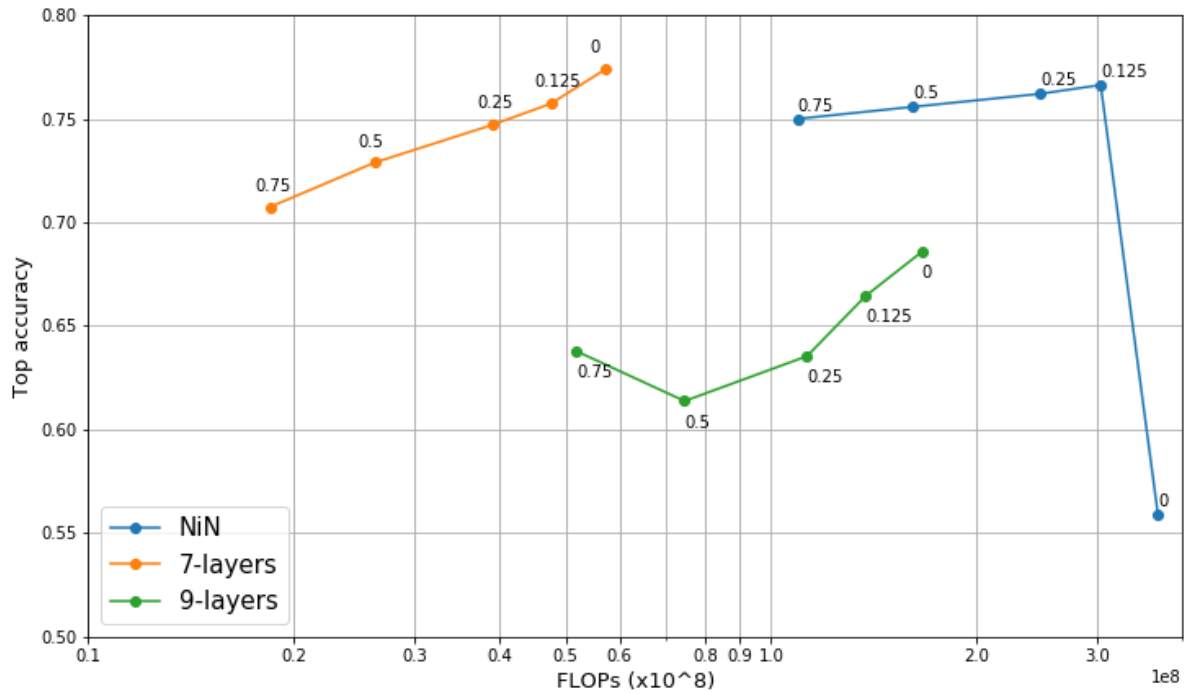


Figure 4: Accuracy-FLOPs trade-off curve on CIFAR-10

4.5. Results with Stanford Dogs Dataset

With the 7-layers CNN (Figure 5), it is the same results as with the CIFAR-10 dataset. We get a better performance without the Octave Convolution and we see a decrease in performance with an increase of alpha.

With the 9-layers CNN (Figure 6), it is much better than with the CIFAR-10 dataset. The network is more stable and we see the expected results, which means a better performance versus the baseline for alpha equals 0.125 or 0.25, with a peak performance at 0.125, and then a slight decrease in accuracy for bigger values – 0.5 and 0.75.

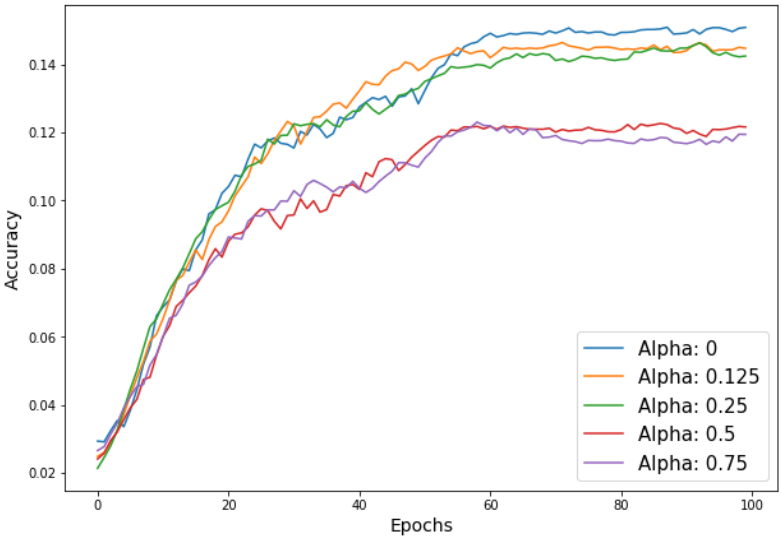


Figure 5: Test accuracy with 7-layers CNN (smoothed curve)

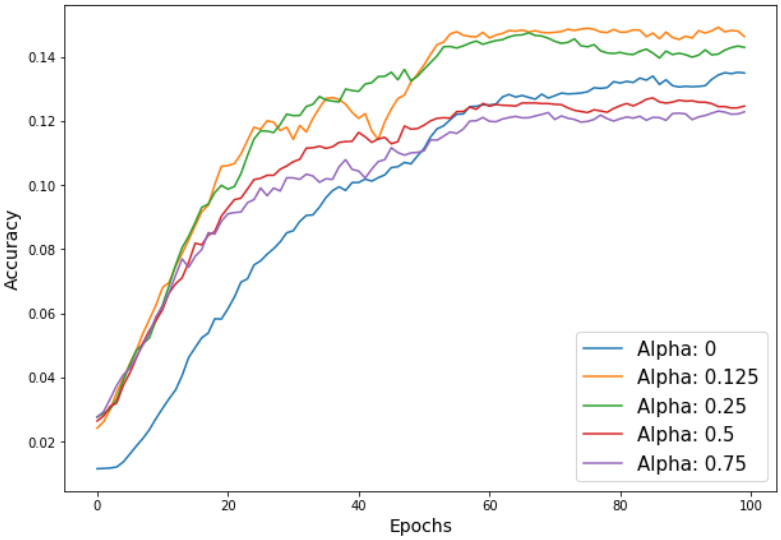


Figure 6: Test accuracy with 9-layers CNN (smoothed curve)

With the NiN architecture (Figure 7) we only used half of the classes (60 over 120) because of the poor performance with 120 breeds. There is a clear improvement with the use of the Octave Convolution. But surprisingly, the performance increase with alpha.

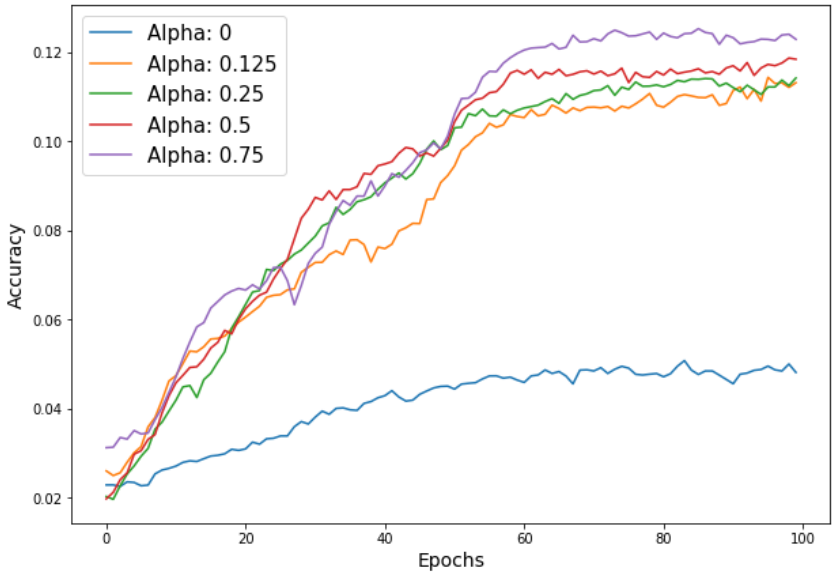


Figure 7: Test accuracy with NiN (smoothed curve)

Figure 8 shows the Accuracy-FLOPs trade-off and we can see that the 9-layers CNN showed very consistent results according to the publication results, with a peak of performance at alpha=0.125.

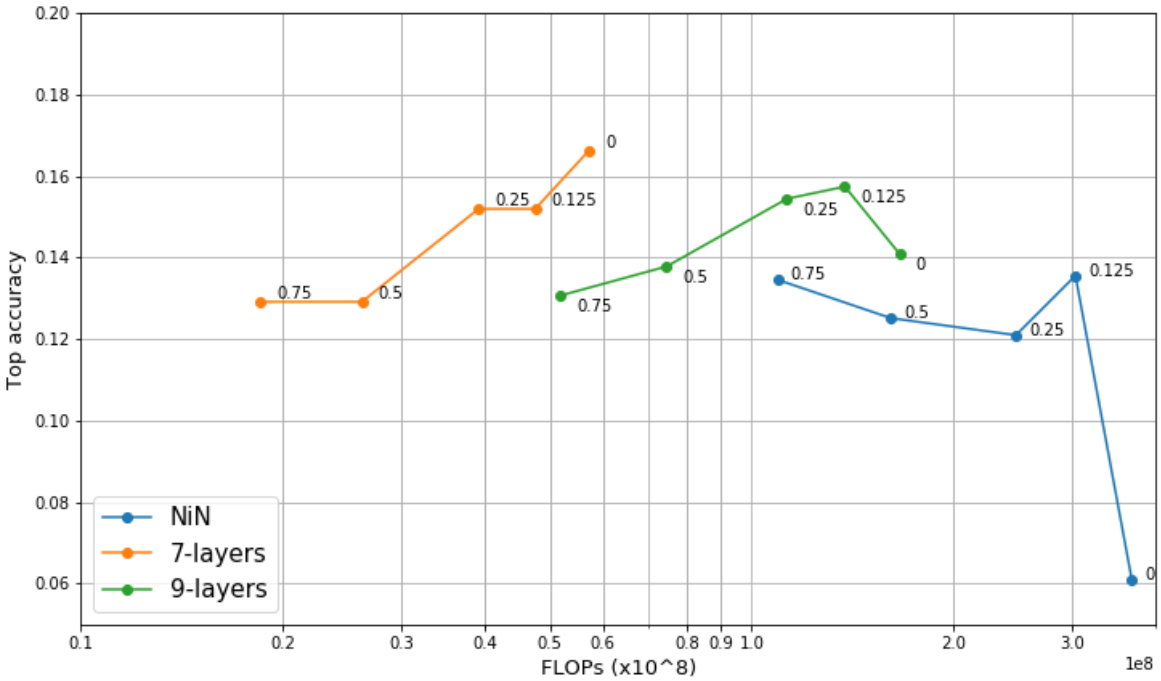


Figure 8: Accuracy-FLOPs trade-off curve on Stanford Dogs Dataset

5. Analysis

We tried 6 combinations of architecture/dataset, and on these 6 couples, 3 of them behaved like announced in the paper and showed an increase in performance with the Octave Convolution, 2 did not show any improvement with the Octave Convolution and the last one did not perform well (even without the Octave Convolution).

5.1. Influence of the architecture

When we look more closely at the architectures, it seems that those with more convolutional layers benefit more from the Octave Convolution. Indeed, the NiN architecture with 9 convolution layers showed a substantial improvement over the baseline in both cases. Then, the 9-layers network with 5 convolution layers showed a little increment of 2% over the baseline with the Stanford Dogs Dataset. Whereas, the 7-layers network did not show any improvement.

5.2. Execution time

For the 7-layers and 9-layers CNN, we did not observe a significant change in the execution time of an epoch, with or without the Octave Convolution. This might be because most of the parameters in these networks are located in the fully-connected layers. However, the NiN model, which is only composed of convolutional layers showed very different execution times depending on the value of alpha:

Alpha	Execution time (s) 1 epoch - CIFAR-10	Execution time (s) 1 epoch – Dogs Dataset
0	102	12
0.125	129	21
0.25	115	19
0.5	84	14
0.75	63	10

It illustrates the fact that the bigger alpha is, the less time it takes to compute, with the exception of alpha=0 (ie. without Octave Convolution), but that may be because the implementation is different.

6. Conclusion

The Octave Convolution addresses the problem of spatial redundancy in vanilla CNN models, and uses a new convolution operator that processes the high- and low-frequency features separately.

By replacing the vanilla convolution operator by the Octave Convolution, it demonstrated not only a substantial saving in memory and computation, but also an increase in performance when used with state-of-the-art networks.

In this work, we tried to replicate those results on simpler architectures with different datasets. We got mixed results, as we were not able to systematically improve the performance versus the baseline. However, we displayed that some architectures – like the Network-in-Network, could greatly benefit from using this new operator.

In the light of the effect of the Octave Convolution can have in a network's performance, it might be interesting to go further and extend the multi-scale separation to more than 2 scales.

Bibliography

- Drop an Octave: Reducing Spatial Redundancy in Convolutional Neural Networks with Octave Convolution, Y. Chen, H. Fan, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, J. Feng – April 2019

- How fast is my model ?, Blog Machine Think – June 2018

- Network in Network, M. Lin, Q. Chen, S. Yan – March 2014

- https://github.com/koshian2/OctConv-TFKeras - Octave Convolution Implementation