Adaptive Convolutional Kernels

Julio Zamora-Esquivel, Adan Cruz Vargas, Paulo Lopez Meyer, Omesh Tickoo
Intel Labs, Mexico.
julio.c.zamora.esquivel@intel.com

Abstract

The quest for increased computer vision recognition performance has led to the development of high complexity neural network architectures, each time with evolving deeper topologies. To avoid high computing resource requirements of such complex networks, and to enable operation on devices with limited resources, this work introduces the concept of adaptive kernels applied to convolutional layers. Motivated by the non-linear perception response in human visual cells, the input image is used to define the weights of a dynamically changing kernel, named adaptive kernel. This novel adaptive kernel is used to perform a second convolution operation over the input image in order to generate the output features. Adaptive kernels enable accurate recognition with significant lower memory requirements; this is accomplished by reducing the number of kernels and the number of layers needed as compared to typical CNN configurations. Additionally, the use of adaptive kernels allow the decrease by 2X the number of epochs required for training, and the number of activation function computations. Our experimental results show a reduction of 66X of the parameters needed of a CNN compared to LeNet when evaluated with the MNIST dataset, maintaining >99% of accuracy. Additionally, when using adaptive kernels implemented in a ResNet18, we observed a higher performance when compared to a known ResNet100 reported in the literature for CIFAR10 and it also gets better accuracy for ImageNet database.

1. Introduction

Convolutional Neural Networks (CNN) have demonstrated their capacity to achieve state-of-the-art (SoA) accuracy in different computer vision tasks (CV), e.g. image classification, semantic segmentation, and object detection. Most of the proposed research in this area rely on deeper and deeper architectures, comprising of millions of trainable parameters, in order to increase the recognition performance of the models. The benefit behind having deeper architectures is that more complex features can be abstracted as we add more layers to a neural network (NN). However, such models are not suitable to be implemented for edge computing, e.g. in embedded devices, cellphones, or drones, due to its size and computational cost. In recent years, there have been several approaches proposed like ShuffleNet [26], MobileNet [7], HENet [19], and SqueezeNet [4], in an attempt to generate smaller size models that require less computational resources, and with a low trade-off in terms of accuracy performance.

Previous reported research has demonstrated that the response of visual cells in the human eye is a non-linear function of their stimuli [23]. Thus, finding non-linear models that best represent visual data could increase the recognition performance of CV tasks. Given that a typical convolutional layer in a CNN is represented by a linear system, its ability to express the above mentioned response is limited by the type of intermediate layers, and the number of neurons in them. The use of a non-linear neuron, as it was firstly explored to solve the XOR problem [15] which cannot be solved by a first order neuron but can be by a second order neuron, seems an appropriate way to tackle this non-linearity issue. Ideally, such non-linear approaches should be able to provide similar performance as compared to traditional CNNs, albeit at a much lower computation and memory costs. On the other hand, in the CV field, the filter used to extract borders in an image is different to the filter used to extract corners, etc. Our method uses the input image to define the filter that extracts better features depending on the input image.

Motivated by the second order neurons approach, a convolutional kernel is proposed in this work that includes defined non-linear transformations, and that results in similar performance as the SoA algorithms reported in the literature, while yielding a significant reduction in required memory. In addition, this method can be implemented with most of the existent CNN architectures, like ResNet, generating an improved version of it.

The main contributions of this work are: (i) we define the non-linear convolutions designed for high image classification accuracy under memory constraints; (ii) using the proposed non-linear convolutions, we present a deep NN design that is partially pre-defined and is capable of completing self-definition during the pattern evaluation phase, including dynamically defining the convolutional kernels on-the-fly, depending of the input pattern; (iii) we present a method to tackle problems associated with higher order NN. For example, the problem of saturation of the activation due to the N-order of multiplications used can be solved by constraining every new dynamically generated weight to a pre-known range defined by the activation function, e.g. if a hyperbolic tangent function is used, the dynamically generated weights would be all within the $(-1, 1)$ range; and (iv) we make available for testing a pytorch-based implementation located in [24].

In the following sections we describe and prove the points mentioned above: Section 2 mentions previous related approaches reported in the literature; Section 3 explains in detail our proposed method; Section 4 shows the experimental results obtained by our implementation for different datasets; and in Section 5, we present the conclusions drawn from the results obtained.

2. Related work

Different ways of increasing the accuracy of NN have been addressed in the existing scientific literature. Most of these rely on the use of CNNs, as they are generalized linear models, and their level of abstraction is low [12]. Some works have dealt with this low abstraction by having additional layers [10, 21], resulting in a considerable increase in the accuracy over different datasets, e.g. CIFAR-10 [9] and ImageNet [3]. However, although the network depth has shown a crucial importance in NN performance, the difficulty to train these models also increases, and moreover, the accuracy of the network might tend to drop [5].

A proposal to tackle this problem is the use of ResNet blocks [5], which given a set of inputs X, with their associated labels Y and a function $H(x)$ that maps X to Y, the network defines a building block $y = F(x)+x$, where $F(x)$ represents the residual mapping to be learned. This residual network allows to train a deeper network without being affected by degradation, using a larger amount of layers and parameters. Because of this, these type of models are not suitable for embedded devices.

In the work presented in [12] a non-linear function approximator is proposed as a solution to increase the level of abstraction of the feature extractor. The typical convolutional kernel is replaced with a micro network, i.e. the non-linear approximator. A multilayer perceptron is used as the instantiation of this micro network, and by sliding the micro network over the input in a similar manner as a CNN, the feature maps are obtained, but this is not used to define new filters as we do.

In the approach proposed in [2], a dynamic filter module is presented, where the filters used for the convolution are generated dynamically depending on the input. This dynamic filter module consists of two parts: a filter generating network, which creates sample specific parameters given an input, and a the dynamic filtering layer, which applies the parameters to a different input. Although [2] presents similar components to our proposed work, it is important to notice the differences between them: in [2], a CNN is trained to get the convolutional kernels, and then an additional network is defined to generate such kernels, where this second network is trained independently. In our work, we define a second order convolutional kernel trained using a novel training rule, which is explained in detail in the following section. This second order convolutional kernel allows the training of smaller networks that results in comparable performance.

3. Method

An adaptive kernel (K) is defined by a dynamic filter that changes its weights by itself depending on the input image. This adaptive kernel can be generated using an array of traditional kernels Q. For instance, each element (u, v) in a 3×3 Adaptive Kernel could be generated by a 3x3 linear filter $Q_{u,v}$, as shown in Figure 1.

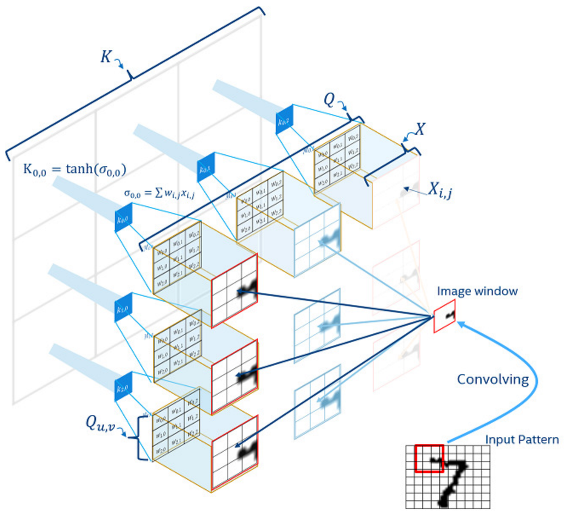
K
Q
X
$X_{i,j}$
$K_{0,0} = \tanh(\sigma_{0,0})$
$\sigma_{0,0} = \sum w_{i,j} x_{i,j}$
Image window
Convolving
Input Pattern
$Q_{u,v}$

Figure 1. One adaptive kernel created by the convolution of the input image with a matrix of kernels: a (3×3) region of the input image is convoled simultaneously with 9 filters to generate $(3 \times 3)\sigma_{u,v}$ outputs, the hyperbolic tangent of the outputs $tanh(\sigma_{u,v})$ is computed to get each $K_{u,v}$ element of the dynamic kernel.

In order to generate this new adaptive kernel K, the convolution operation of each 3×3 filter $Q_{u,v}$ with the input image X, and using a sliding window of 3×3 (for this example), generates a component $\sigma_{u,v}$, defined as:

$$\sigma_{u,v} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} Q_{(u,v)_{i,j}} x_{i,j}, \tag{1}$$

At this point, the hyperbolic tangent is applied to generate the new value $K_{u,v}$ of kernel. By using the $tanh$ activation function, we guarantee the range of values will be within $(-1, 1)$

$$K_{u,v}(\sigma) = tanh(\sigma_{u,v}), \tag{2}$$

This new dynamically generated kernel K is convolved again with the input image X to generate: (Figure 2).

$$S = \sum_{u,v} x_{u,v} K_{u,v}. \tag{3}$$

In the final step, the output pixel is computed using hyperbolic tangent as the activation function like $f(s) = tanh(S)$. Other activation functions could be used in the same fashion, e.g. sigmoid or Relu. As mentioned above, we opted for hyperbolic tangent in order to constraint weights to be in the range of $(-1, 1)$. By sliding the window through all the input image X, we generate the final filtered image. Given that the convolutional kernel $K_{u,v}$

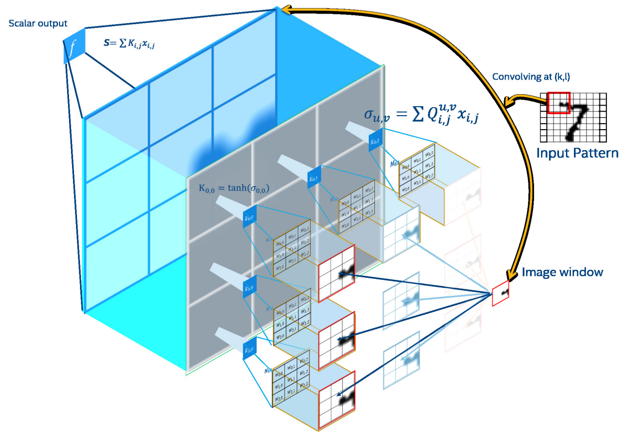
Scalar output
f
$S = \sum K_{i,j} x_{i,j}$
Convolving at (k,l)
$\sigma_{u,v} = \sum Q^{u,v}_{i,j} x_{i,j}$
Input Pattern
$K_{0,0} = \tanh(\sigma_{0,0})$
Image window

Figure 2. A single resulting pixel output from the convolution of the input image and the high order kernel that generates a dynamic kernel, which is convolved with the input image again.

is not static, the computation of σ would require to store the feature and kernel maps, making an inefficient usage of GPU parallelism. In order to speedup its computation and take advantage of current CNN libraries, it is possible to compute the convolution of the entire input image X with each filter (as illustrated in Figure 3):

$$\sigma = \sum_{u,v} Q_{u,v} \otimes I(u,v), \tag{4}$$

where $I(u, v)$ represents the input image X shifted u columns and v rows, and "$\otimes$" represents the convolution operation. In this way, the computation of σ is simplified; it is important to notice that the $K_{u,v}$ elements should be properly gathered from the image $K_{i,j}$.

For the adaptive kernel, a new training rule is obtained by means of the gradient descent technique as a way to adjust its weights using $Q_{u,v}$ to refer to each linear filter, and using (i, j) to refer the elements (weights) of each filter $Q_{u,v}$. This means the element $Q_{(u,v)_{(i,j)}}$ is a scalar value and represents a weight. Since the hyperbolic tangent was used as the activation function, the training rule is defined by:

$$Q^{t+1}_{(u,v)_{(i,j)}} = Q^{t}_{(u,v)_{(i,j)}} + \gamma E(1 - f(S)^2)\delta_{(u,v)_{(i,j)}} \tag{5}$$

where γ is the learning rate, E is the pixel error value for a window position, $f(S) = tanh(s)$ is the output value for that pixel, and δ is defined by:

$$\delta_{(u,v)_{(i,j)}} = (1 - K^2_{u,v})(x_{u,v})(x_{i,j}) \tag{6}$$

For testing purposes, different experiments were evaluated to compare against SoA for the MNIST, CIFAR-10 and ImageNet datasets, explained in detail in the next section, also, a locally collected dataset was used, to prove the significant memory compression without affecting the performance of the network using the proposed models. Additionally, the usage of adaptive kernels was explored in a fully connected layer.

4. Experiments and Results

Our adaptive kernel implementation was written as a custom layer using the Caffe framework for Python [8], with its associated forward and backward propagation, and it was tested in two initial experiments: one over the MNIST dataset, and one over the CIFAR-10 dataset. For the MNIST experiment, we use Nesterov [16] as the training rule, with momentum set to 0.9, initializing the learning rate to 0.01, and a weight decay of 0.0005. We used the same weight initialization as in [5]; no additional data augmentation or preprocessing was performed for this dataset. For the CIFAR-10 [9], we used the same parameters, but with a learning rate initialized to 0.1, with a reduction by a factor of 5 every 20 epochs, and a weight decay of 0.0001; 32 × 32 crop is randomly sampled from a 40 × 40 image or its horizontal flip, with the per-pixel mean subtracted, and divided by the channel standard deviation, to have standardized data.

To further down on the analysis of adaptive kernels, we performed three additional experiments: we implemented a ResNet18 where we replace the initial layer with adaptive kernels; we used a locally collected data to train a model

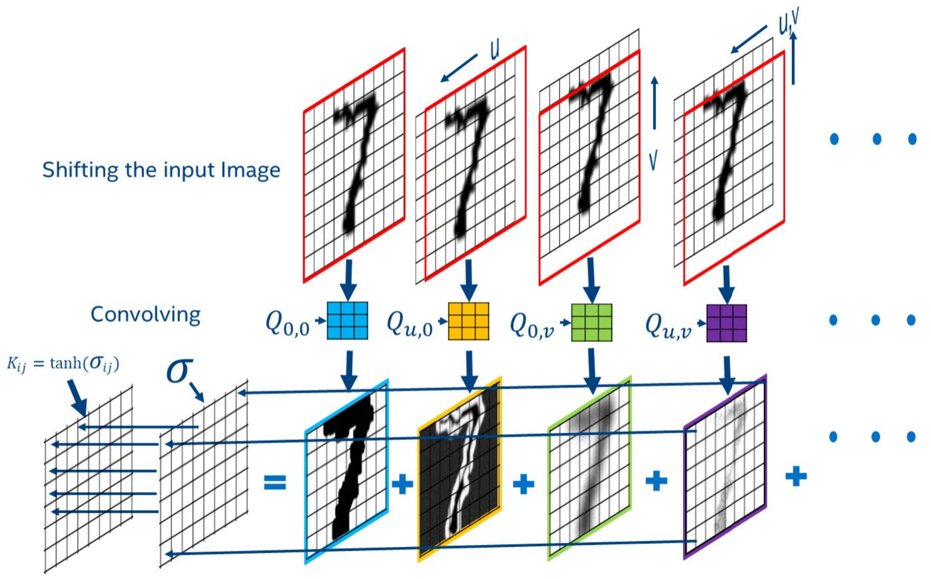

Figure 3. The input image shifted u columns v rows and convoluted with the filter $Q_{u,v}$ to compute all values of σ in parallel.

for robot navigation in a known environment, and we used adaptive kernels in a fully connected network. These experiments are described in the following subsections.

4.1. Experiment 1: MNIST

MNIST is a public dataset that consists of 60,000 28×28 gray scale images in 10 classes (handwritten numbers), with 6000 images per class [11]. There are 50,000 training images and 10,000 test images in the official data. Here, our approach has three main advantages: memory reduction, increment of accuracy outperforming traditional CNN models, and the learning speedup. The results produced by the implementation of the MNIST digits recognition show a big memory compression, using 66X less memory measured through parameter reduction; additionally, a higher accuracy was achieved 2X faster. The LeNet NN fully connected architecture we used as reference is described in detail in Table 2, and Table 3 describes our proposed topology for comparison purposes.

Figure 4 shows examples of how the adaptive kernel is changing at different locations at the input window. Figure 5 shows all the kernels generated for random sample for the digit seven. It can be noticed that for background pixels, the kernels are neutral, i.e. they do not extract any features.

There have been many different NN models proposed for the MNIST classification problem. In Table 4, a subset of these models are presented, selecting only the ones with $> 99\%$ accuracy, and a small number of parameters.

In this context, our adaptive kernel technique presents the smallest CNN model that reaches $> 99\%$ accuracy without any pre-processing in only 5 epochs; in contrast, the

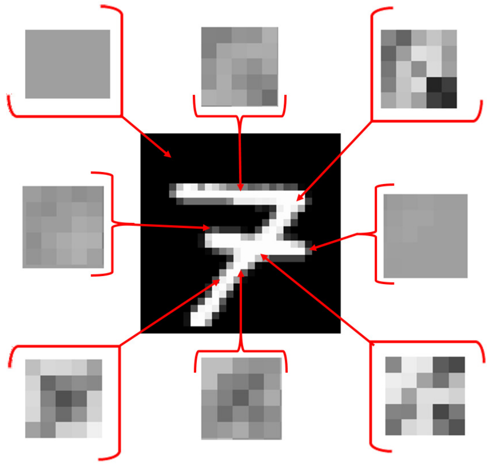

Figure 4. A single kernel generated in different positions of the input image.

LeNet reached 97% after 9 epochs. In terms of the number of operations, LeNet has 2.29M MAC operations, while our method has 1.23M MAC operations for the same database. Here, the number of trainable parameters used by our model is only 6.52K because a symmetric tensor was used, this means that instead of 25 different kernels of 5x5 size, we have only use 15 of the same size.

In order to perform a hyper-parameter sensibility analysis, eight different models with three layers were trained using MNIST. By increasing the number of kernels in the first layer, a saturation on the accuracy can be seen, but this

Table 1. Hyper-parameter sensibility

Model	M1	M2	M3	M4	M5	M6	M7	M8
Adaptive	4	5	6	7	4	5	6	7
Conv.	10	10	10	10	20	20	20	20
F.C.	10	10	10	10	10	10	10	10
Accuracy	98.14	98.17	98.24	98.30	98.57	98.65	98.85	99.04
#parameters	6K	6.8K	7.7K	8.6K	9.5k	10.6K	11.7K	12.8K

Table 2. LeNet CNN Topology as in tutorial

Layer	Units	Type
Layer1	20 Kernels	Conv $5x5$
Layer2	50 Kernels	Conv $5x5$
Layer3	500 Neurons	FC
Layer4	10 Neurons	FC

Table 3. Our Neural Network Topology

Layer	Units	Type
Layer1	5 Kernels	Adaptive $5x5_{5x5}$
Layer2	10 Kernels	Conv $5x5$
Layer3	20 Neurons	FC
Layer4	10 Neurons	FC

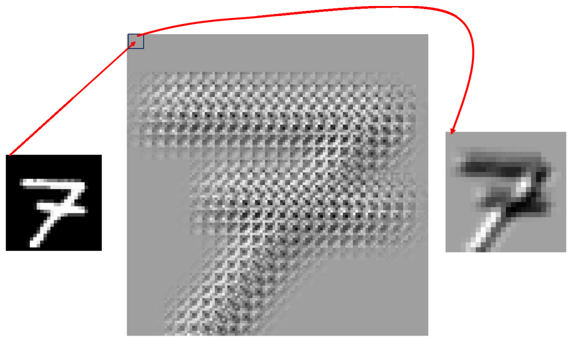

Figure 5. Every input window is convolved by a different filter generated on the fly using the input image.

Table 4. MNIST Accuracy vs Memory for >99% accuracy

Neural Network	#Parameters
LeNet [1]	431K
LetNet5 [25]	60K
50-50-200-10NN [14]	226K
Best Practices [17]	132.5K
Adaptive Kernels CNN	**6.52K**

saturation can be mitigated by increasing the number of kernels in the second layer, as presented in Table 1.

4.2. Experiment 2: CIFAR-10

The CIFAR-10 is a public data set that consists of 60,000 32×32 color images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images in the official release. In this context, CIFAR-10 was used for testing with horizontal flipping, padding, and 32×32 random cropping for data augmentation of the training dataset. As our target model is aimed towards embedded devices where the memory is critical, the goal is to have the smallest model and the highest accuracy possible. The proposed topology was implemented as follows: only the first layer uses adaptive kernels of $(3 \times 3)_{(3\times3)}$, in order to highlight the impact of a single layer in the full topology, and being the first layer where the main feature extraction takes place; additionally, it has 8 convolutional layers, and ends with ten outputs in one final fully connected layer.

Table 5. CIFAR-10 Classification error vs Number of parameters

Neural Network	Depth	#Parameters	Error%
All-CNN [22]	9	1.3M	7.25
MobileNetV1 [7]	28	3.2M	10.76
MobileNetV2 [20]	54	2.24M	7.22
shuffleNet 8G [26]	10	0.91M	7.71
shuffleNet 1G [26]	10	0.24M	8.56
HENet [19]	9	0.7M	10.16
ResNet18 [6]	20	0.27M	8.75
Adaptive Kernels	10	**0.2M**	7.48

In order to compare against the related work described before, some of the latest topologies used for CIFAR-10 pattern recognition problem are included (limiting our analysis to those having less than 2.5M number of trainable parameters). Our intention is not to outperform the accuracy of these topologies. Instead, the idea is to achieve similar results with a significantly smaller model. This approach can potentially enable us to target an efficient implementation into embedded systems. Table 5 shows a summary of results from the smallest models recently reported for the CIFAR-10 classification problem.

While there is no simple way to determine the efficiency of a NN, our target is the highest accuracy with the least amount of memory as shown in Table 5. Comparing with

All-CNN [22] our solution represents a 6X memory reduction with only a 0.2% drop in accuracy; comparing with MobileNetV2, we have 11X compression with only a 0.2% drop in accuracy .

4.3. Experiment 3: ResNet18 + Adaptive kernels

We used the ResNet18[6] topology to implement adaptive kernels in it. In this experiment, only the first layer was changed to use 64 adaptive kernels instead of 64 convolutional kernels, as in the typical topology, in order to show the contribution of one adaptive layer. Although the topology with an adaptive layer has less feature maps in the first layer, it can achieve better performance when compared to a larger ResNet100 as can be seen in Table 6.

Table 6. Combining Adaptive layers with ResNet18

Neural Network	Depth	#Parameters	Error%
ResNet18	18	11M	6.98
ResNet50	50	25.6M	6.38
ResNet100	100	44.5M	6.25
Adaptive + ResNet18	18	**11.1M**	5.55

As table 6 shows in CIFAR-10, the modified ResNet18 that use our Adaptive kernels over performed the ResNet100, clearly combining or Adaptive layer with ResNet100 the accuracy could increase even more, but the number of parameters and the amount of operations is too high to be considered in embedded applications.

4.4. Experiment 4:ImageNet

Evaluation of Adaptive kernels for ImageNet data set. Our model is based on the well known topology ResNet18[18], replacing one layer to use our adaptive kernels, the accuracy increased 0.4%, results in table 7

Table 7. Comparing Error for ResNet18* (with adaptive Kernels) Vs reported ResNet topologies for ImageNet

Neural Network	Depth	#Param	Top1-Error%
ResNet18	18	11M	30.24
ResNet50	50	25.6M	23.85
ResNet100	100	44.5M	22.63
ResNet18*	**18**	**11M**	**29.8**

4.5. Experiment 5: Embedded Application

In this experiment, a locally collected dataset was used, in order to train a NN model that drives a robot to navigate inside of a known room. Given an input image I (200×200) from the robot camera, it estimates a required direction $\alpha \in [0, 360]$, and a distance $d \in [0, 100]$, that drives the robot to reach the center $c(x_c, y_c)$ of a known region.

Table 8. DroNet Neural Network Topology

Layer	Units	Type
Layer1	32 Kernels	Conv 5×5
Layer2-5	k-Kernels	ResNet 5,3,1
Layer6-8	2k-Kernels	ResNet 5,3,1
Layer9-11	4k-Kernels	ResNet 5,3,1
Layer12	120 Neurons	FC
Layer12’	100 Neurons	FC

Table 9. Our Neural Network Topology

Layer	Units	Type
Layer1	5 Kernels	Adaptive $5 \times 5_{5\times5}$
Layer2-5	k-Kernels	ResNet 5,3,1
Layer6-8	2k-Kernels	ResNet 5,3,1
Layer9-11	4k-Kernels	ResNet 5,3,1
Layer12	120 Neurons	FC
Layer12’	100 Neurons	FC

For this purpose, a NN architecture was created inspired by DroNet [13] described in Table 8, but instead of a standard convolutional layer, an adaptive convolutional layer was used (Table 9) this is because the DroNet uses 7.2M parameters and need to reduce to 0.2M parameters (32X reduction) to enable the usage in small drones. The last two layers of the network were trained considering a classification problem, in order to estimate a steering angle class and a distance class.

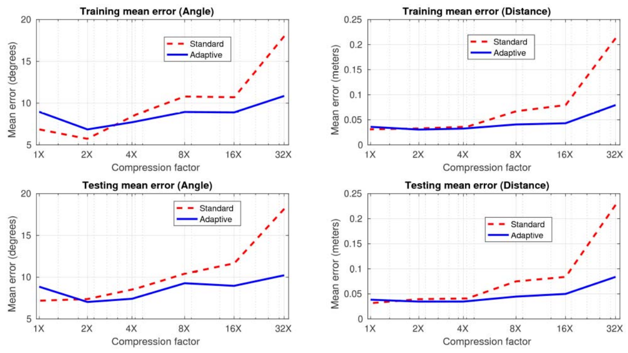

Figure 6. Accuracy vs parameter reduction with standard and adaptive models, training (top) and testing (bottom).

The compression of the model is achieved by reducing the number of filters in the three ResNet blocks. In Table 9 we have k, $2k$, and $4k$ kernels for each block, respectively. When the compression increases, accuracy drops, but the adaptive kernels helps to keep a better accuracy in comparison with the traditional convolutional layers, (Figure 6).

4.6. Experiment 6: Adaptive Kernels in a Fully Connected Network

The methodology we are using for adaptive convolutional layers can be applicable for models based on fully connected layers, by moving from a vector to a matrix of weights Q defined by:

$$S = \sum_u w_u x_u \rightarrow S = \sum_u x_u K_u \tag{7}$$

where K_u is not a constant pre-trained value, it is generated based on the input vector doing $K_u(\sigma) = tanh(\sigma_u)$, and σ_u is the inner product of Q_u vector with the input x:

$$\sigma_u = \sum_{i=0}^{N-1} Q_{(u)_i} x_i. \tag{8}$$

In this manner, the weight K is dependent of the input vector. In order to compare this method against the traditional fully connected layers, a 2D artificial scenario was used to simplify visualization of the results, where the problem consists on separating five different classes. This context is presented in Figure 7. The hyper-parameters were de-

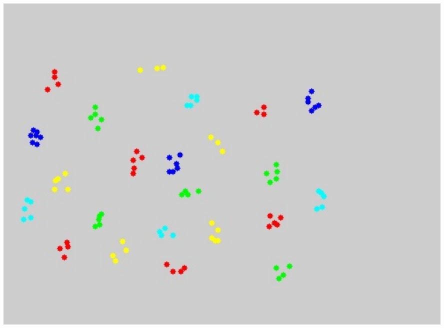
Figure 7. Simple 2D classification scenario

fined based on an incremental process, testing two and three fully connected layers until the majority of the input patterns where correctly classified. As a result, the smallest topology that best solves the problem is 20-20-5, i.e. two hidden with 20 units each and one output layer with 5 units. Classification results for this fully connected model are colored in Figure 8, obtained after 923 iterations with simple gradient descendent.

In contrast, our proposed adaptive fully connected topology produced the best results using 13-5, i.e. one hidden layer with 13 units and one output layer with 5 units. The result generated after 404 iterations of simple gradient descendent is shown colored in Figure 9.

For this experiment, the use of the adaptive fully connected layer resulted in a reduction of the number of parameters from 540 to only 182 (3X smaller), less number of layers needed (two instead of three), and a faster convergence in 404 epoch instead of 923 (2X less epochs).

5. Conclusion

In this work, we introduced the concept of adaptive convolutional kernels, capable of redefining dynamically the convolutional kernel during the inference time and depending on the input image. The experimental results obtained show that our technique not only reduces the memory size of the model, but also reduces the time during the training process. Additionally, our results suggest that the adaptive convolutional kernels generalize better than traditional CNNs. This is because kernels adapt dynamically to extract better features depending on the input image. In terms of efficiency, our method showed very compelling results, generating 6X lighter solutions with less than 0.2% accuracy drop. Based on this, our solution should be able to impact directly the computational cost of inference on embedded systems, increasing the operational scope of applications for these systems. As in traditional CNNs, the increase in the number of kernels in a layer tends to produce saturation, with a marginal increment of accuracy, thus the topology definition also plays an important role. In our experiments it was observed that less adaptive kernels in a layer generate comparable or even a better level of abstraction than a higher number of traditional convolutional kernels within a layer. For instance, in ResNet18, the 64 convolutional filters in the first layer were replaced with adaptive filters,

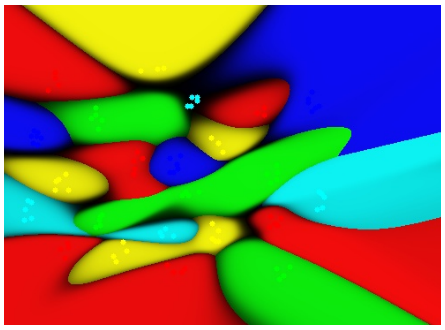
Figure 8. Fully connected layer based classification, using 20-20-5 topology

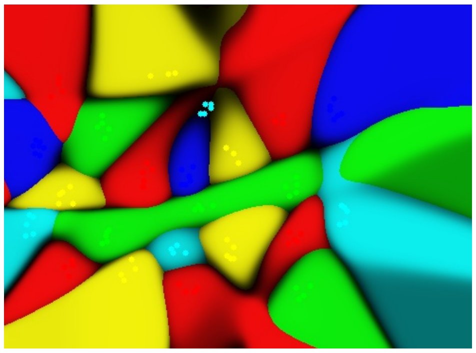
Figure 9. Adaptive fully connected layer based classification, using 13-5 topology

producing even better results when compared to the larger ResNet100 topology. Lastly, it was proven that fully connected layers can also benefit from this technique, where one adaptive layer can replace two traditional layers.

References

[1] BAIR/BVLC. Lenet architecture in caffe tutorial. *Github*, 2018.

[2] B. D. Brabandere, X. Jia, T. Tuytelaars, and L. V. Gool. Dynamic filter networks. *CoRR*, abs/1605.09673, 2016.

[3] J. Deng, W. Dong, R. Socher, L. jia Li, K. Li, and L. Fei-fei. Imagenet: A large-scale hierarchical image database. In *In CVPR*, 2009.

[4] M. W. M. K. A. W. J. D. K. K. Forrest N. Iandola, Song Han. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size. *ICLR2017*, 2017.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.

[9] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[11] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[12] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[13] A. Loquercio, A. I. Maqueda, C. R. D. Blanco, and D. Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 2018.

[14] S. C. Y. L. Marc Ranzato, Christopher Poultney. Efficient learning of sparse representations with an energy-based model. *NIPS2006*, 2006.

[15] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[16] Y. Nesterov. A method of solving a convex programming problem with convergence rate O(1/sqr(k)). *Soviet Mathematics Doklady*, 27:372–376, 1983.

[17] J. C. P. Patrice Y. Simard, Dave Steinkraus. Best practices for convolutional neural networks applied to visual document analysis. *ICDAR 2003*, 2003.

[18] Pytorch. https://pytorch.org/docs/stable/torchvision/models.html, 2019.

[19] R. Z. Qiuyu Zhu. Henet: A highly efficient convolutional neural networks optimized for accuracy, speed and storage. *arXiv:1803.02742*, 2018.

[20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[22] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[23] R. G. Szulborski and L. A. Palmer. The two-dimensional spatial structure of nonlinear subunits in the receptive fields of complex cells. *Vision Research*, 30(2):249 – 254, 1990.

[24] https://github.com/adapconv/adaptive-cnn. Adaptive convolutional neural networks source code: Pythorch implementation. *Our Code*, 2018.

[25] Y. B. P. H. Yann Lecun, Lon Bottou. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[26] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.