V-CNN: A versatile light CNN structure for image recognition on resources-constrained platforms

Radu Dogaru
Natural Computing Laboratory,
Dept. of Applied Electronics and Information Eng., University "Politehnica" of Bucharest
Bucharest, Romania
radu.dogaru@upb.ro

Ioana Dogaru
Natural Computing Laboratory,
Dept. of Applied Electronics and Information Eng., University "Politehnica" of Bucharest
Bucharest, Romania
ioana.dogaru@upb.ro

***Abstract*— This paper introduces V-CNN, a versatile structure inspired from previously introduced light models such as NL-CNN. What makes the V-CNN structure efficient is the process of optimizing its hyper-parameters. Consequently, several aspects in proper design for building of efficient (validation accuracy near state of the art while keeping the complexity of the structure) V-CNN structures, are considered and detailed in this paper. While V-CNN includes as particular cases several previously defined models such as NL-CNN and XNL-CNN, it can be better tailored for efficient deployment given a specific dataset. A V-CNN model with 1.5 million parameters obtained 91.55% validation accuracy on CIFAR-10 dataset, surpassing the previous result of 90.60% using the NL-CNN. The V-CNN model offers the following advantages: i) using optimization hints described herein it allows maximal efficiency (good accuracy at low complexity) for a wide variety of datasets; ii) has a low number of layer primitives, thus making easier their specific design for deployment on various TinyML or EdgeAI platforms, including FPGAS.**

Keywords—convolutional neural network, resources constrained, EdgeAi, TPU, TinyML

I. Introduction

Many actual machine learning applications require integration of classifiers and other types of inference engines into low-power low-resources platforms. Recently, in the framework of TinyML [1] numerous solutions are proposed aiming to keep functional performance (e.g. the accuracy on test set) at a high level (up to the state-of-the-art solution for a given dataset) while reducing the implementation complexity (number of parameters, quantization, low latency).

Light architectures were recently proposed, among them the widely known (and readily available in major deep-learning frameworks such as Tensorflow or Pytorch) EfficientNet [2] with is more recent V2 variant [3] or MobileNet [4] specifically designed for implementation in mobile platforms. Other compact models were proposed in [5][6] as well as in other works. Major machine learning frameworks now include a large collection of such models (see https://keras.io/api/applications/) . In most cases the models are still large (with at least tens of millions in the number of their trainable parameters). Also, their structure is rather "bushy" relying on a wide variety of primitives that may raise difficulties when such primitives have to be defined in TinyML platforms. To provide neat-structures with a limited set of primitives and with enough number of hyper-parameters to make models efficient, in [7] we investigated how non-linear convolution may be exploited in a classic deep-layer CNN architecture. Thus, a specific architecture called NL-CNN where the first two macro-layers were nonlinear achieved very good validation accuracies, comparable and even better than MobileNet models. The basic element in NL-CNN is a "macro-layer" where besides the number of filters, as in any convolution layer, one may tune a *nl* parameter (the degree of non-linearity). In the companion paper [8] we extended the concept to an improved type of model (XNL-CNN) capable to deal with higher resolution images. With a proper optimization of its hyper-parameters the model achieves very good accuracies, in many cases like state-of-the-art values obtained with consecrated models [2]-[4], but here with a lower complexity.

Both NL-CNN and XNL-CNN models imposed some constraints on how the number of filters can vary as a function of the macro-layer order. They also considered nonlinear macro-layers only for the first two levels. Herein we propose a freely available [9] versatile model V-CNN where such constraints were eliminated, thus opening an interesting perspective for a better tailoring of the model to the specific data, an aspect that is not considered in most of the available models (usually one picks an available model, say EfficientNetB0 and applies it to some particular data – with not much hyper-parameters for tuning).

A detailed description of the model is given in Section II. Section III details the effects of tuning various parameters of the model and gives hints in choosing the hyper-parameters for best performance. Additional drop layers are discussed in Section IV and as indicated in the Concluding remarks, it turns out that with a proper tuning the V-CNN is versatile and can be easily adapted to various dataset balancing well good accuracy performance with low complexity of the model.

II. The V-CNN model

A. The macro-layer

As introduced in [7] the basic module composing the XNL-CNN is the "macro-layer" depicted in Fig.1. As shown, in the case *nl=0* (classic convolution) there is a permanent (always present) layer. For *nl>0* "nonlinear convolution" is emulated by adding *nl* additional (convolution + nonlinear-activation) layers. As detailed in [7] with proper tuning of *nl*

significant increases in accuracy results in comparison with the "classic" deep model using linear (*nl*=0) macro-layers. As detailed in [7] significant increases in accuracy require the addition of the **BatchNormalization** and **Dropout** layers. A seen in Section IV, the optimization of drop factor can lead to significant improvements.

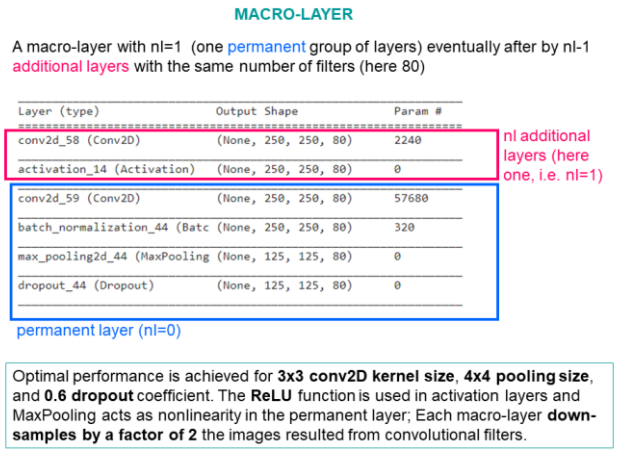

Fig. 1. The macro-layer, basic constituent unit of the XNL-CNN

B. *The V-CNN model and its parameters*

The V-CNN model is *simply a stacking of a given number M of macro-layers.* Then a flatten (*flat=1*) or global_average (*flat*=0) layer is added to provide input for a multi-layer perceptron (MLP) layer. If *hid*=[], there is only the output, *softmax* layer while if choosing *hid=[x,y]* one can add any number of desired dense layers (with ReLu nonlinearity) for instance 2 such layers in the above case (first with *x* and second with *y* neurons), each with a dropout.

For a given model one needs only to specify 3 lists: *fil*=[f1,f2,..fM] (a list giving the number of filters per each macro-layer) *nl*=[nl1,nl2,..nlM] where the degree of nonlinearity is specified (nl=0 means a "linear", i.e. classic layer) and a *hid*=[hid1,hid2, …] corresponding to the hidden dense layers in the output classifier. To keep a low complexity it is recommended to have a void list *hid=[]* while better tuning the convolutional macro-layers. For example, the instance of a "fat" model yet capable to solve CIFAR100 with a good performance (67.35% using no augmented data) is described by the following call:

```
model=create_v_cnn_model(input_shape,
num_classes, flat=0, fil=[100,
150,200, 200], nl=[2,1,1,0],
hid=[100])
```

representing a V-CNN with 4 macro-layers (images in the output of the convolutional chain are 2x2 sized) and a progressive (rising ramp) filter profile (details in Section III) with an additional dense hidden layer with 100 neurons. A graphical representation of the model (using https://netron.app/) is provided in Figure 2. The limited number of primitives (layers) are clearly emphasized in this figure, in contrast to the relatively "bushy" (structurally, complex) EfficientNetB0 model.

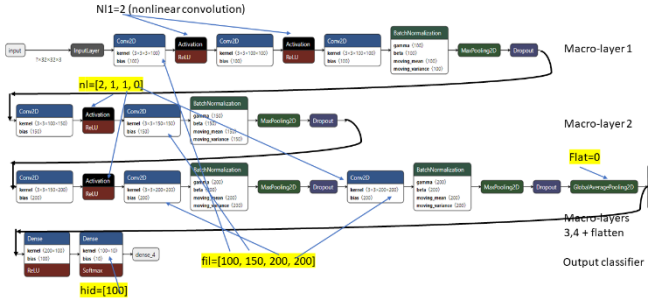

Fig. 2. A specific V-CNN model with 4 macro-layers

After training 300 epochs with the CIFAR10 dataset the above model achieved a validation accuracy of 91%. In contrast, a much larger (4 million parameters) pretrained EfficientNetB0 applied to the same dataset provides a smaller, 84.75% accuracy!

The V-CNN model (code is available [9]) must be properly tuned for the best compromise between very good validation accuracy and a low complexity, depending on the specific resources and constraints of a given project. Section III provides an in-depth analysis of the effect of tuning each of the parameters using a "difficult" dataset namely CIFAR100. This work considers only low-resolution image recognition datasets (no more than 32x32 with at most 3 colurs) since we are mainly interested to see how various hyper-parameters influence the efficiency of the model. Taking [8] as a starting point one can easily extend the V-CNN model for high resolution images. All experiments were carried out using the Kaggle platform with GPU P100 accelerator.

III. Optimizing Hyperparameters – CIFAR 100 Case Study

In the next we consider several aspects in choosing the hyper-parameters and argument based on experience with a difficult dataset (CIFAR100 – 100 classes of color small resolution images)

The reference V-CNN model (denoted as {0}) was an *M=4* macro-layer model with *fil=[100,100,100,100], nl=[1,1,0, 0], hid=[], flat=1* . In the next, to avoid clutter, for each new various models described as "model {x}" only changes from a specified reference model will be given.

A. *Influence of macro-layer numbers*

The results are summarized in table I. The only difference between models {0}-{3} is the number of elements *M* in the *fil* and *nl* list. For *fil* all values are 100 and for *nl* only the first 2 values in the list are 1, the remaining are 0.

TABLE I. Influence of Macro-Layer Number on Validation Accuracy

M (model)	2 {3}	3 {1}	4 {0}	5 {2}
Validation 'Acc (%)	**49.1**	**55.4**	53	53.8
Million of parameters	**0.91**	**0.524**	0.495	0.55

As noted, the optimal number of macro-layers is *M=3* and degrades significantly if less.

Note also that with *M=3* the size of the images in the output of the convolutional chain is 4x4. From model {1} we considered the effect of adding a hidden layer resulting in {4} (hid=[100]) with 52.25% accuracy and 0.534Mpar size and {5} (hid=[2000]) with 54.88% accuracy but a large size of 2Mpars. This result substantiates that in general, hidden layers in the output classifier are not necessary.

B. *Optimizing the NL profile*

In Table II we consider changes from the best model {1} in exploring various *nl* lists to locate optimal nonlinearity profiles.

TABLE II. EFFECTS OF NONLINEARITY PROFILES

{model} nl profile	**Param. (million)**	**Val. Accuracy (%)**
{6} [0,0,1]	**0.434**	**52.02**
{7} [0,1,0]	**0.434**	**53.75**
{8} [0,1,1]	**0.524**	**53.31**
{9} [1,0,0]	**0.434**	**53.18**
{10} [1,0,1]	**0.524**	**54.3**
{1} [1,1,0]	**0.524**	**55.4**
{12} [1,1,1]	**0.614**	**55.56**
{13} [2,1,1]	**0.704**	**55.76**
{14} [2,2,1]	**0.8**	**54.40**

Clearly, adding nonlinearity helps improve the accuracy (with a slight increase in the number of parameters) but as expected by the Occam's rule, adding it too much (like in model {14}) would worsen the result. We suggest that such an optimal profile has to be searched for each particular dataset although it appears that adding nonlinearity on the first macro-layers impacts more on accuracy than using it on the final macro-layers. Consequently, so far model {13} has the best performance on CIFAR100.

C. *Optimizing the filter profile*

Previous research in [7] and [8] indicates that raising the number of filters will have a positive impact on accuracy, although it will also increase the size and eventually saturate the performance. In previous models we used an exponential growth for the number of filters in respect to the order of the macro-layer. Herein we consider 4 other filter profiles, in addition to the "flat profile" in model {13} (i.e. equal number of filters) to emphasize the best choice. Table III summarizes results indicating that a linear "rising ramp" profile gives the best accuracy improvement over {13}. All parameters are as in {13} except the *fil* list given in the table for each new case.

Clearly, a "rising ramp" profile (here by adding 50 filters to each new macro-layer) gets the best compromise between performance and compactness. One may consider various initial number of filters and gain (additional filters per macro-layer) for their specific datasets. The best model in our case was denoted {19} which gives 57.52% accuracy for a model size of about 1.5 million parameters. Other profiles with a similar overall number of parameters are less efficient than the optimal {19} model.

TABLE III. INFLUENCE OF FILTER PROFILE

{model} Filter (fil) profile	**Param. (million)**	**Val. Accuracy (%)**
{13} [100,100,100]	**0.704**	**55.76**
[20, 60, 100] rising	**0.35**	**52.46**
{18} [100,200,100] Inverted V	**1.15**	**56.47**
{19} [100, 150, 200] Rising ramp	**1.47**	**57.52**
{20} [200,150,100] Falling ramp	**1.585**	**55.49**
{21} [200,100,200] V-shape	**1.86**	**57.09**

IV. INFLUENCE OF THE "DROP" PARAMETER

In previous NL-CNN and XNL-CNN we considered the drop=0.5 parameter for the "drop" layers. Herein, all previous experiments considered the case *without a drop layer* (*drop=0*). Therefore, it would be interesting to observe if such a layer is indeed necessary at the end of each macro-layer and to optimize the associated *drop* parameter. Table IV summarizes the results with models based on {19}.

TABLE IV. INFLUENCE OF DROP PARAMETER

Drop value	**0**	**0.25**	**0.4**	**0.5**	**0.6**	**0.75**
Accuracy (%)	**57.52**	**58.62**	**60.14**	**60.79**	**63.71**	**55.67**

The optimal model ({19} with drop layers and *drop*=0.6) will be denoted as model {24}. The above results are important, stressing the necessity of a drop layer. When its parameter is finely tuned, a 6% rise in accuracy is achieved without increasing the number of trainable parameters.

V. OPTIMAL V-CNN MODELS FOR REFERENCE DATASETS AND COMPARISON WITH OTHER LIGHT CNNS

So far, an optimal **V-CNN model** was derived (**model {24}**) with **63.71% accuracy and 1.47 million parameters.** For comparison we considered an **EfficientNetB0** implementation (see [9]) for the same dataset and the result was worse, namely: **57.72% accuracy for 4.177 million** of parameters. Can we do better with further tuning V-CNN?

The answer is "yes" since we considered the addition of a fourth layer **in model {25}** defined by *fil*= **[100,150,200,200] and *nl*=[2,1,1,0].** With a slight size increase of **1.59 million parameters** after 300 training epochs it achieved **65.92%.** *Moreover, turning flat=0 (GlobalAverage flattening) we obtained* ***model {25b} with 1.53 million parameters and a validation accuracy of 67.35%.*** It is interesting to observe that it has 10% increase in performance over the predefined EfficientNetB0 (where, indeed, there was no hyper-parameter tuning).

A. *Comparative performance for various datasets*

Here we consider *V-CNN {25b} as a reference model and investigate how this model performs when other various datasets are used.* We considered low resolution datasets, most also used in [7], for comparison and reference. Two lower complexity models were derived from {25b} by simply

dividing fil by 2 (e.g. *fil* =[50,75,100,100]) denoted as {25b-**light**} and by 5 (e.g. *fil* =[20,30,50,50]) denoted {25b-**xlight**}.

The best models are reported in Table V and made available in [9] as well as support code for the V-CNN.

TABLE V. PERFORMANCE ON VARIOUS DATASETS

Dataset (model)	Params (million)	Valid. Accuracy %	Best acc in [7] (%)
CIFAR 100	1.53	**67.35**	**57.9 in [10]**
CIFAR100 Light	0.39	62.2	-
CIFAR100 X-light	0.065	42.46	-
CIFAR10	1.516	**91.12**	**90.69**
CIFAR10 (augmented data)	**1.516**	**91.55**	
CIFAR10 Light	0.38	89.40	
CIFAR10 Xlight	0.061	83.06	
SVHN	1.516	96.35	**96.15**
SVHN light	**0.38**	**96.72**	
SVHN xlight	0.062	95.55	
MNIST light	0.38	99.68	**99.70**
MNIST xlight	**0.061**	**99.70**	
F-MNIST light	**0.38**	**94.01**	**93.62**
F-MNIST light	0.061	93.4	
EMNIST light	**0.38**	**90.83**	**90.47**

In the above F-MNIST stands for Fashion-MNIST and EMNIST for Balanced EMNIST. More details about all datasets may be found in https://paperswithcode.com/datasets

It is interesting to observe that depending on the specific dataset a "fat" (original {25b}) model can be overfit while by simply resizing the filter profile one can easily find a well-balanced model. For instance, in Figure 3 the training curves for CIFAR100 (light) indicate such a well-balanced situation.

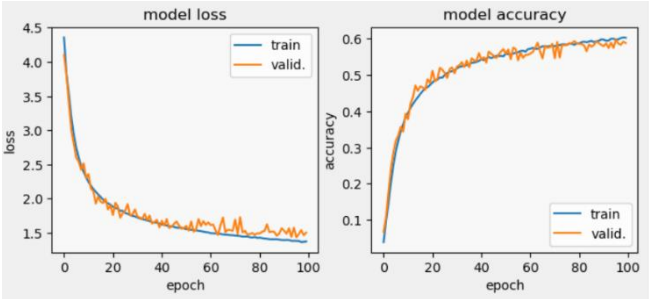

Fig. 3. Training curves for CIFAR100 (light)

Observe that in the case of SVHN dataset, the "light" version gives the best accuracy performance. Similarly, MNIST extra light (xlight) gives the best accuracy. Clearly a finer tuning of the filter profile for the specific dataset will improve the results. On all datasets, properly tuned V-CNN surpassed the previous accuracy performance demonstrated in NL-CNN [7] and as indicated in Section II and V they surpassed the EfficientNetB0 for both CIFAR100 and CIFAR10.

VI. CONCLUDING REMARKS

The focus of this work was to introduce and evaluate a convenient (light) CNN model for image recognition. It is a generalization of the NL-CNN model in [7] and it also includes XNL-CNN [8] as a particular case, being thus able to cope with a wide variety of datasets. Unlike most of the existent deep-CNN models (most included in major machine learning environments) V-CNN can be optimally tuned to any specific dataset given its large set of tunable parameters. Although tuning a large set of hyper-parameters may seem complicated, we show several simple ways to do it demonstrating very good accuracies (compatible with state-of-the-art results given the low complexity).

In [10] (Table 3), a CNN light model with about the same size as ours gives on CIFAR100 an accuracy of only 57.9% (compared to 67.35% in our model). The results on small images datasets surpassed our previous use of NL-CNN [7] and is expected that further investigation of V-CNN over fine-grained large resolution datasets will surpass the results of XNL-CNN in [8]. The model is based on a reduced set of primitives (the type of layers mentioned in Figs. 1-2) making thus possible the convenient deployment on a wide variety of EdgeAI or TinyML platforms including FPGA-based ones [11]. Instead, the structure of the consecrated, pretrained models is rather bushy and their implementation into EdgeAI platforms may lead certain difficulties.

REFERENCES

[1] L. Dutta, S. Bharali, "TinyML meets IoT: A comprehensive survey", Internet of Things (2021)

[2] Tan, M. and Le, Q. V., "Efficientnet: Rethinking model scaling for convolutional neural networks", ICML, 2019a.

[3] Tan, M. and Le, Q. V., "EfficientNetV2: Smaller Models and Faster Training", (2021), https://arxiv.org/abs/2104.00298

[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, 2018, pp. 4510-4520.

[5] I. Freeman, L. Roese-Koerner, A. Kummert, "EffNet: An Efficient Structure for Convolutional Neural Networks", 2018. Available from https://arxiv.org/abs/1801.06434 .

[6] S. F. Cotter, "MobiExpressNet: A Deep Learning Network for Face Expression Recognition on Smart Phones," 2020 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 2020, pp. 1-4, doi: 10.1109/ICCE46568.2020.9042973.

[7] R. Dogaru and I. Dogaru, "NL-CNN: A Resources-Constrained Deep Learning Model based on Nonlinear Convolution," 2021 12th International Symposium on Advanced Topics in Electrical Engineering (ATEE), Bucharest, Romania, 2021, pp. 1-4.

[8] R. Dogaru, A.D. Mirică and Ioana Dogaru, "XNL-CNN: An improved version of the NL-CNN model, for running with TPU accelerators and large image datasets", submitted, ISEEE2023.

[9] R. Dogaru and Ioana Dogaru, Github code and models in support of V-CNN: https://github.com/radu-dogaru/V-CNN

[10] A. Guerriero, M. R. Lyu, R. Pietrantuono, and S. Russo, "Assessing Operational Accuracy of CNN-based Image Classifiers using an Oracle Surrogate," Intelligent Systems with Applications, vol. 17, pp. 1–13, 2023.

[11] Lee H. S., Jeon J. W., "Accelerating Deep Neural Networks Using FPGAs and ZYNQ" In: 2021 IEEE Region 10 Symposium (TENSYMP), Jeju.