XNL-CNN: An Improved Version Of The NL-CNN Model, For Running With TPU Accelerators And Large Image Datasets

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

Adrian-Dumitru Mirică
Doctoral School of Electronics Telecommunications and Information Technology, University "Politehnica" of Bucharest
Bucharest, Romania
adrian.mirica@stud.etti.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 XNL-CNN, an extended version of the previously introduced, compact NL-CNN model. While the NL-CNN was designed with typical Tiny-ML constraints in order to make possible integration of various AI image recognition solutions into low power, low complexity computing platforms, the same stands for its extended version discussed herein. In addition, we consider additional layers to cope with larger image sizes, as found in most actual relevant datasets (e.g. the Cars196 or Flowers104). Moreover, since working with large image sizes requires intensive computational power, TPU accelerators readily available on Kaggle platform were used for training and evaluation of our model. In comparison to other, widely known resources-constrained architectures, such as EfficientNet or MobileNet our model offers the following advantages: i) a larger set of hyper-parameters allowing the user to do better designs (maximal accuracy for minimal resources), adapted 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).

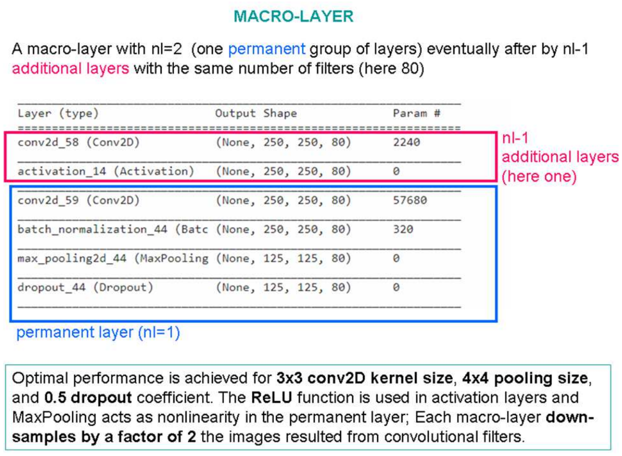
Many such architectures were recently proposed, among them the widely known (and readily available in major deep-learning frameworks such as Tensorflow or Pytorch) EfficientNet [2] with is more recent V2 variant [3] or MobileNet [4] specifically designed for implementation in mobile platforms. Other compact models were proposed in [5][6]. In [7] we investigated how non-linear convolution may be exploited in a classic deep-layer CNN architecture (somehow similar to AlexNet) concluding that a specific architecture called NL-CNN where the first two macro-layers (each emulating nonlinear convolution with basic layer primitives available in Tensorflow-Keras) achieves very good test accuracies, comparable and even better, for the same limited number of parameters to 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). The NL-CNN operates well only on datasets containing low-resolution (up to 64x64) images. For instance, using the auto model classification set Cars196 [8] (without augmentations or class reduction) the NL-CNN achieves at most a validation accuracy of 63% (image size 160x160, 0.88 million parameters). Trying to work with larger image sizes results in memory crashes when GPU is used, concluding that working with larger image sizes would require TPU.

Since many of the actual datasets contain larger resolution (fine-grained) images (up to 512x512) it was desirable to extend this model to a more general one called XNL-CNN (X stands from eXtended). 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. A comparison with other models for the same training conditions and datasets is given in Section IV for the cars196 datasets while concluding remarks are given in Section V. Using an optimal 200x200 image resolution allows XNL-CNN to improve classification accuracy to 77.62% (3.18 million parameters) compared well to accuracies obtained by other deep architectures with similarly low number of parameters.

II. THE XNL-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=1* (classic convolution) there is a permanent (always present) layer. For *nl>1* "nonlinear convolution" is emulated by adding *nl-1* 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*=1) macro-layers. As detailed in [7] significant increases in accuracy require the addition of the **BatchNormalization** and **Dropout** layers as stressed in [7].


MACRO-LAYER

A macro-layer with nl=2 (one permanent group of layers) eventually after by nl-1 additional layers with the same number of filters (here 80)

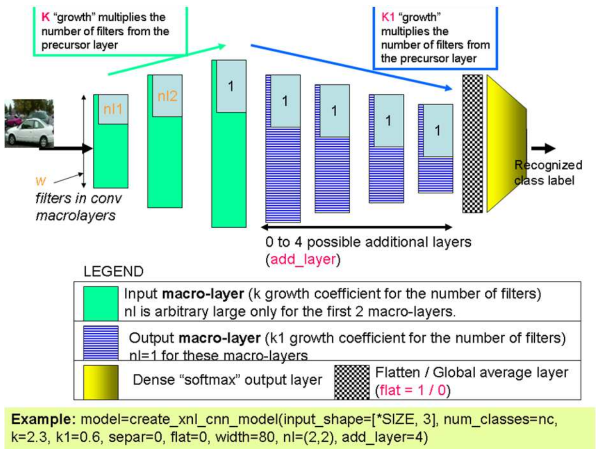
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_58 (Conv2D)           (None, 250, 250, 80)      2240
_________________________________________________________________
activation_14 (Activation)   (None, 250, 250, 80)      0
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 250, 250, 80)      57680
_________________________________________________________________
batch_normalization_44 (Batc (None, 250, 250, 80)      320
_________________________________________________________________
max_pooling2d_44 (MaxPooling (None, 125, 125, 80)      0
_________________________________________________________________
dropout_44 (Dropout)         (None, 125, 125, 80)      0
_________________________________________________________________
```

Optimal performance is achieved for **3x3 conv2D kernel size**, **4x4 pooling size**, and **0.5 dropout** coefficient. The **ReLU** function is used in activation layers and MaxPooling acts as nonlinearity in the permanent layer; Each macro-layer **down-samples by a factor of 2** the images resulted from convolutional filters.

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

B. The XNL-CNN model and its parameters

The macro-modules are then assembled into the XNL-CNN model depicted in Figure 2. While the first 3 macro-layers have a coefficient of growth ***k*** like in NL-CNN [7], for the remaining layers (**add_layer** tunable parameter, depending on the image size) a second (***k1***) growth coefficient was added. If add_layer=4, no growth coefficient is applied (the last macro-layer has the same number of filters as the previous one).


K "growth" multiplies the number of filters from the precursor layer
K1 "growth" multiplies the number of filters from the precursor layer
nl1
nl2
1
1
1
1
1
Recognized class label
w
filters in conv macrolayers
0 to 4 possible additional layers (add_layer)
LEGEND
Input **macro-layer** (k growth coefficient for the number of filters) nl is arbitrary large only for the first 2 macro-layers.
Output **macro-layer** (k1 growth coefficient for the number of filters) nl=1 for these macro-layers
Dense "softmax" output layer
Flatten / Global average layer (flat = 1 / 0)

Example: model=create_xnl_cnn_model(input_shape=[*SIZE, 3], num_classes=nc, k=2.3, k1=0.6, separ=0, flat=0, width=80, nl=(2,2), add_layer=4)

Fig. 2. The XNL-CNN model

In order to reduce model's complexity and latency there is only a dense output **softmax** layer. According to our experiments if the chain of convolution layers is properly designed, there is no need to use additional hidden dense layers (otherwise consuming many resources). Figure 2 displays a typic call of the XNL-CNN model creating function. The **flatten** layer can be replaced by a **global_average** layer, particularly when working with large number of filters ***w***.

C. Hints regarding the choice of parameters

The model as defined above (code is available [9]) must be properly tuned for the best compromise between a very good validation accuracy and 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 but here some basic hints are provided to properly tune the parameters. First, the **add_layer** parameter must be considered, in direct relationship to input image size (it is assumed that a variable **SIZE** is already known after loading the data, as well as the number of classes **num_classes** variable): roughly **add_layer=0** corresponds to **32x32 image sizes**, while a doubling of the image size would require an increment (+1) of add_layer. Consequently, **add_layer=4 would correspond 512x512** image sizes. Then, one may take as a good starting point ***nl=(2,2), k=2***, k1=0.7 and vary the ***w*** parameter (e.g. increasing it from 10 to higher values with a certain step (e.g. 10) aiming to increase the validation accuracy after training on the given dataset. Usually after *w* reaching some specific limit, the accuracy is not anymore increasing at the expense of a large number of parameters. One can also try shifting from **flatten** to **global_average** in order to reduce the number of parameters. Finer tuning of ***k*** and ***k1*** may improve the overall performance even more. Also, when compact models are required **separ=1** will replace standard convolution with a separable one, but in this case usually validation accuracy decreases.

III. TUNING AND OPTIMIZING HYPERPARAMETERS

In the next the influence on validation accuracy and number of parameters is considered in respect to the most relevant parameters. In all cases the Cars196 dataset [8] was considered. To accelerate the tuning process only 100 training epochs were considered. Here the aim was to investigate the effect of each parameter. One can retrain for more epochs (usually 500 suffice for this dataset) using the best parameter choice.

A. Influence of non-linearity

We did (including [7]) extensive experiments with various datasets and found that using nonlinear convolution is effective only for the first 2 layers and with *nl<4*. Consequently, Table I summarizes accuracy values obtained for various combinations. The rest of the XNL-CNN parameters are: **k=2.3, k1=0.7, separ=0, flat=0, width=80, add_layer=3** and image size is **200x200**.

TABLE I. TEST SET ACCURACIES FOR VARIOUS NL=(NL1,NL2)

Nl2 / Nl1	*1*	*2*	*3*
1	65.8	72.0	-
2	74.2	**76.8**	73.2
3	74.6	75.8	72.28

As noted, the "linear" model has the worse accuracy (65.8%) and a significant (maximal) improvement with 11% is achieved for nl=(2,2). Figure 3 gives a synthetic view of these results.

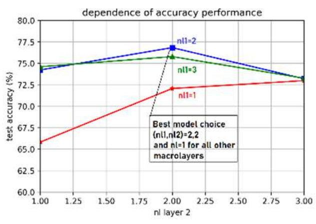


Fig. 3. Influence of the nonlinearity factor on the validation (test) accuracy

B. *Image size*

In Figure 4 the effect of various image sizes is considered for a lighter XNL-CNN architecture with the following parameters: k=2, k1=0.7, separ=0, flat=1, width=55, nl=(2,2). The **add_layer** parameter is specified within the figure. In addition, Figure 5 shows how the number of parameters depends on the image size.

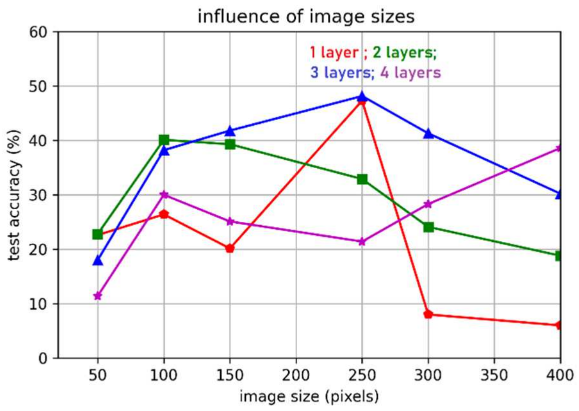


Fig. 4. Dependence of validation accuarcy on the image size.

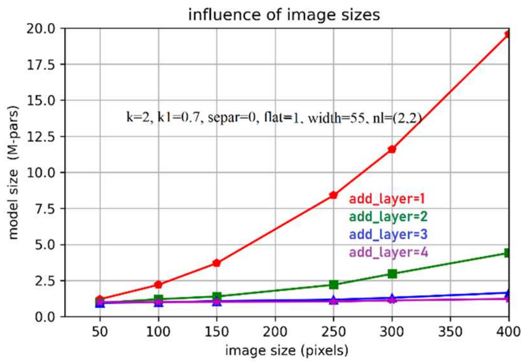


Fig. 5. Dependence of number of model parameters on the image size

C. *Proper choice of the number of macro-layers*

To investigate the effects of the **add_layer** parameter when image size is considered for different values (shown in figure) the same light XNL-CNN model as above was considered, and the results are depicted in Figure 6 (for accuracies) and Figure 7 (for training times)

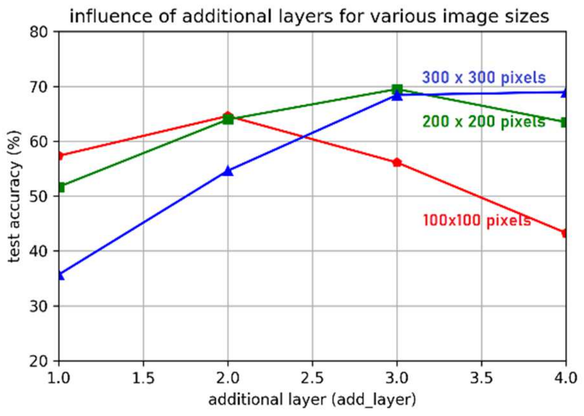


Fig. 6. Influence of add_layer on validation accuracy.

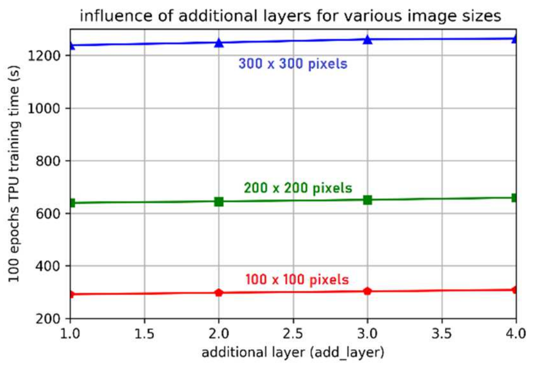


Fig. 7. Training time (TPU accelerator) dependence on image size and additional layers.

These results indicate that adding layers does not significantly influence the training time. The major effect on training time is given by data size (image size) thus justifying the choice of the minimal image size producing a reasonable accuracy (e.g. 200x200 instead of 300x300 as it results from Figure 6).

IV. Influence of the K parameter

The "expansion" *k* parameter allows a finer tuning aiming to keep accuracy at maximum while minimizing the number of parameters. Figure 8 shows that an optimal value exists, while taking larger values results in an important overfit with declining accuracy and increasing model size (Figure 9).

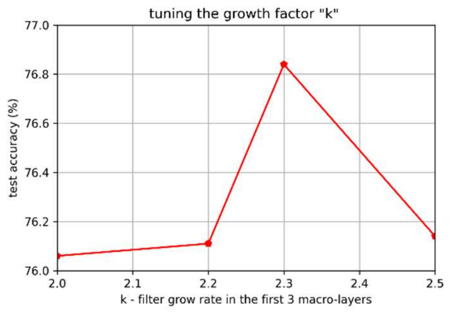


Fig. 8. Finding the optimal *k* in order to get maximal accuracy

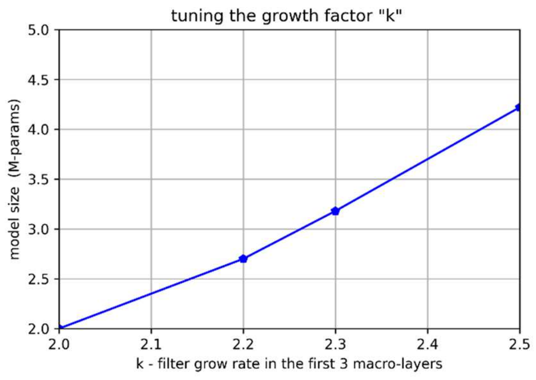


Fig. 9. Model size versus *k* parameters.

A similar analysis carried on for *k1* let to the conclusion that optimal accuracy is achieved when *k1=0.7.* It may be possible to have other optimal values when datasets are changed.

V. Optimal XNL-CNN models and comparison with other light CNNs

Using the hints mentioned in Section II and the results in Section III with some additional training (up to 500 epochs) for the Cars196 problem the best validation accuracy result was **77.62% (3.18 mega parameters**) for **XNL-CNN** model used in Table I; Using the same dataset a **pretrained Xception** model with an additional trainable dense layer gives a slightly improved accuracy of almost **81% but requires around 20** mega parameters while the pretrained **MobileNetv3 with 4.47** million parameters gives a validation accuracy of only **71%.**

In order to have a better idea how XNL-CNN compares to other pre-existent models, Table II considers results after running XNL-CNN against various pretrained CNN models. A different dataset is now used, namely the Kaggle competition dataset Flowers104 [10] where flowers images (here with 192 or 224 image sizes) must be classified into 104 categories. In all cases 200 training epochs were considered, the code was running on the Kaggle cloud platform and the with TPU-VM3.8 accelerator. The Adam() optimizer was employed in all cases.

TABLE II.

Model	**Millions of parameters**	**Valid. Accuracy %**	**Train time (s) /epoch**	**Image size**
XNL-CNN1	3.16 3.16	79.23 77.15	10 13	192 224
XNL-CNN2 *k=1.5, flat=1*	0.88 0.925	73.87 72.46	5 10	192 224
XNL-CNN3 *w=60*	0.54	75.26	10	224
XNL-CNN4 *k=2*	1.25	77.90	11	224
XNL-CNN5 *w=70*	1.68	78.82	12	224
XNL-CNN6 *k=k1=1* *w=100 add=4*	0.77	77.20	10	224
XNL-CNN7	**4.1**	**82.35**	14	224
MobileNetV3	3.1	16.4	6.5	224
EfficientNetB0	**4.18**	**89.97**	9	224
Xception	21.	74.8	7	224
VGG	14.76	69.13	7	224
ResNet101	42.8	19.37	9	224
Augmented dataset				
XNL_CNN8	0.88	**84.19**	8	192
EfficientNetB0	**4.18**	**89.52**	11	192

XNL-CNN1 has the same parameters as the model considered in Table I, while for XNL-CNN1-6 models the parameters changed from previous line, are specified.

The best performance (82.35%) is obtained with the XNL-CNN7 model (parameters: *k*=2.3, *k1*=0.6, *flat=0*, *w*=100, *nl*=(2,2), *add_layer*=3) and is surpassed with 7.5% only by the EfficientNetB0 model. With respect to most other state-of-the art CNN models XNL-CNN performs better and it also allows very light models by proper tuning (with minimal sacrifice in accuracy performance like XNL-CNN3 where size decreased 8 times for a slight 7% loss in accuracy). With more careful parameter optimization, the performance could be increased furthermore. Such an example is the light XNL-CNN8 model where a very good performance (84.19%) was obtained by using an augmented version of the same dataset. With the same augmented data set the (5 times bigger) EfficientNetB0 model performs only slightly better (89.5%).

VI. Concluding remarks

The focus of this work was to introduce and evaluate a convenient (light) CNN model for image recognition. It is an extension of the NL-CNN model in [7], capable of operating with fine-grained (large resolution) images. The model is based on a reduced number of primitives (the type of layers mentioned in Fig.1) making possible the convenient deployment on a wide variety of EdgeAI or TinyML platforms. In comparison to state-of-the-art compact models, the XNL-CNN, if properly tuned, provides comparable and sometimes better accuracy than state-of-the-art CNN models. Based only on 6 layers primitives, the proposed architecture may be easily implemented in any TinyML platform including FPGA-based ones [11]. Instead, the structure of the consecrated, pretrained models is rather bushy and their implementation into EdgeAI platforms may lead to certain difficulties. Another advantage of the XNL-CNN model is its versatility, i.e., the possibility to consider very light models for certain low-cost microcontroller platforms (e.g. XNL-CNN3) whereas state-of-the-art CNN architectures often require more than 2 million parameters).

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] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in Proceedings of the IEEE international conference on computer vision workshops, 2013, pp. 554–561.

[9] Github code: https://github.com/radu-dogaru/NL-CNN-a-compact-fast-trainable-convolutional-neural-net/blob/main/xnlcnn.py

[10] Flowers104: A kaggle competition dataset, available at https://www.kaggle.com/competitions/flower-classification-with-tpus

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