JYX
JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

This is a self-archived version of an original article. This version may differ from the original in pagination and typographic details.

Author(s): Terziyan, Vagan; Malyk, Diana; Golovianko, Mariia; Branytskyi, Vladyslav

Title: Hyper-flexible Convolutional Neural Networks based on Generalized Lehmer and Power Means

Year: 2022

Version: Published version

Copyright: © 2022 The Author(s). Published by Elsevier Ltd.

Rights: CC BY 4.0

Rights url: https://creativecommons.org/licenses/by/4.0/

Please cite the original version:

Terziyan, V., Malyk, D., Golovianko, M., & Branytskyi, V. (2022). Hyper-flexible Convolutional Neural Networks based on Generalized Lehmer and Power Means. Neural Networks, 155, 177-203. https://doi.org/10.1016/j.neunet.2022.08.017

Neural Networks 155 (2022) 177–203

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Hyper-flexible Convolutional Neural Networks based on Generalized Lehmer and Power Means

Vagan Terziyan [a,*], Diana Malyk [b], Mariia Golovianko [b], Vladyslav Branytskyi [b]

[a] Faculty of Information Technology, University of Jyväskylä, Finland
[b] Department of Artificial Intelligence, Kharkiv National University of Radio Electronics, Ukraine

ARTICLE INFO

Article history:
Received 4 April 2022
Received in revised form 7 August 2022
Accepted 15 August 2022
Available online 23 August 2022

Dataset link: https://github.com/Adversarial-Intelligence-Group/flexnets

Keywords:
Convolutional Neural Network
Flexibility
Adversarial robustness
Pooling
Lehmer Mean
Power Mean

ABSTRACT

Convolutional Neural Network is one of the famous members of the deep learning family of neural network architectures, which is used for many purposes, including image classification. In spite of the wide adoption, such networks are known to be highly tuned to the training data (samples representing a particular problem), and they are poorly reusable to address new problems. One way to change this would be, in addition to trainable weights, to apply trainable parameters of the mathematical functions, which simulate various neural computations within such networks. In this way, we may distinguish between the narrowly focused task-specific parameters (weights) and more generic capability-specific parameters. In this paper, we suggest a couple of flexible mathematical functions (Generalized Lehmer Mean and Generalized Power Mean) with trainable parameters to replace some fixed operations (such as ordinary arithmetic mean or simple weighted aggregation), which are traditionally used within various components of a convolutional neural network architecture. We named the overall architecture with such an update as a hyper-flexible convolutional neural network. We provide mathematical justification of various components of such architecture and experimentally show that it performs better than the traditional one, including better robustness regarding the adversarial perturbations of testing data.

© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Machine learning (ML) in general and various (deep) models and corresponding applications of artificial neural networks in particular have achieved great success and adoption in recent years. Convolutional neural networks (CNNs) are important members of the deep learning family of models capable of processing visual information. They originated from the neocognitron model (Fukushima, 1979), which promoted further studies (Hubel & Wiesel, 1959) on visual cortex. The typical CNN is based on end-to-end supervised learning by backpropagation. Such networks automatically learn to optimize the filters (or kernels) unlike former algorithms where these filters were hand-engineered; and such automatic feature extraction is a major advantage of CNNs. Being an excellent tool for solving practical problems, CNNs still have certain conceptual limitations and are criticized for: (1) lack of general intelligence; (2) poor cross-domain generalization; and (3) week connection with the neuroscience research.

Firstly, can we say that models and applications based on CNNs are actually intelligent? Yes, but the essential share of the intelligence within these models is actually the intelligence of humans who design them. These models are used as an artificial intelligence (AI) tool, which is controlled by human experts (both data scientists and domain experts) who take care of the essential intelligent tasks like problem formulation, feature selection, model architecture, initial and running settings, learning and validation processes, understanding the outcomes, etc. CNNs, as a domain of deep neural networks' applications handling real-world problems, is an obvious leader with an "unreasonable effectiveness" (Sejnowski, 2020) among the AI methods and tools. A crucial aspect of this success and the corresponding "deep learning revolution" (Sejnowski, 2018) are novel neural architectures, which have mostly been developed manually.

Secondly, unfortunately, CNNs are learned so that they become highly tuned to the training data, and, while performing well within a particular image dataset, they are much weaker if not useless for the same task with slightly different data due to hidden overfitting (Sa-Couto & Wichert, 2021). Multiple implementations of CNNs show that the small rescaling of the input image can drastically change the network's prediction. Azulay and Weiss (2019) argue that the convolutional architecture does not give the desired invariance since it ignores the classical sampling theorem and learns to be invariant to transformations only for images that are very similar to typical images from the training

* Corresponding author.
E-mail addresses: vagan.terziyan@jyu.fi (V. Terziyan), diana.malyk@nure.ua (D. Malyk), mariia.golovianko@nure.ua (M. Golovianko), vladyslav.branytskyi@nure.ua (V. Branytskyi).

https://doi.org/10.1016/j.neunet.2022.08.017
0893-6080/© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

dataset. Therefore, the problem of insuring invariance to small image transformations in CNNs while preserving high accuracy remains unsolved (Azulay & Weiss, 2019).

Thirdly, in spite of the fact that neuroscience and AI share a long history of collaboration (AI is often used as a tool for neuroscience research of understanding brain functions and vice-versa — AI models are often inspired by neuroscience research), current AI systems are still remarkably simplistic in comparison to brain networks and still incapable of modeling brain functions accurately (Macpherson et al., 2021). For example, comparatively simplistic CNNs do not yet come close to fully modeling the myriad of processes around biological visual processing. Both neuroscience and AI have the problem of interpreting neural activity. As noticed by He and Toyoizumi (2021), interpretation (especially for the AI community) of how the deep network solves the task is still challenging and there is no consensus about what such interpretation would be.

One popular attempt to address some of these challenges and also a popular direction towards intellectualizing CNNs as well as other deep learning models (particularly automating of the network configuration management functions) is known as AutoML (automation of ML). There is growing interest in automated neural architecture search methods (Elsken et al., 2019) as one possible dimension of the AutoML activity. Such automation may result in self-configurable ML (Wong, 2018) in general and self-configurable deep learning (Isensee et al., 2021) in particular, which is an important feature of future strong AI. Regarding the deep learning pipeline, AutoML methods (in addition to neural architecture search) also cover data preparation, feature engineering, and hyperparameter optimization (He et al., 2021). Use of AutoML methods with CNNs is difficult due to the specifics of CNN's architecture. However, recently there have appeared several successful examples of applying neural architecture search approaches to CNNs. Chen and Tsai (2021) suggested a neural architecture search algorithm based on metaheuristics. The basic idea is to use the "expected value" instead of the "objective value" to evaluate options for neural architectures, which makes the algorithm perform better than genetic neural architecture search, especially for complex image classifications. Zhang, Liu et al. (2022) and Zhang, Weng et al. (2022) propose a new paradigm of automatic design of a suitable CNN architecture for scene classification. Their framework has two phases. In the search phase, special calculation strategies distinguish the optimal and other operational components and special added noise handles the training vs. validation gap. In the evaluation phase, the optimal cell architecture is stacked to construct the final network. Experiments with such a framework demonstrated the impressive classification performance comparably to the manually designed CNN models. Hu et al. (2022) suggested an image-based neural architecture automatic search framework as an alternative to CNN. On the basis of a full image, a search space as a cell-based structure is constructed to enrich the feature representation. The optimal cells will be determined by employing gradient descent and the final trained CNN architecture is automatically constructed by stacking the optimal cells. The experiments demonstrated a competitive performance in classification for some datasets.

However, some of the limitations regarding the generalization capacity of CNNs are difficult to address as they are related to the fixed backbone mathematics hidden within the corresponding models. According to Sejnowski (2020), the mathematical theory of deep learning and the complexity of deep learning models (with millions of parameters and trained with millions of labeled examples) are evolving towards approaching the complexity of the real world. Deep learning networks as trainable input–output mapping functions have special mathematical properties that we are just beginning to understand and the majority of questions are still left unanswered. There is a clear contrast between the complexity of real neurons (inherited from cell biology) and the simplicity of the artificial neurons in neural network models, and missing features are likely to be important but have not yet been exploited in model networks. Neural networks nowadays are capable of using hierarchical representation and superposition of simple mathematical atomic functional components to represent, approximate and compute (with high expressive power and learnability) a wide range of complex functions (Filom et al., 2021). However, Sejnowski (2020) claims that having found one class of functions to describe the complexity of signals in the world, perhaps we missed some more efficient ones and those we have not yet explored may go beyond our intuitions.

We believe that the chain of mathematical functions, according to which CNNs operate, e.g., data preprocessing, convolution, pooling, neurons and various activation functions, loss and backpropagation, etc., would benefit from a more flexible mathematical basis than they have now. Currently, the majority of these CNN components include the mathematical operation of a weighted sum (weighted average) of inputs, and the weights affect only the result of the operation and not control the mathematical configuration of the operation itself. Training such weights by backpropagation makes CNN highly tuned to the particular training data and poorly reusable for other datasets. Traditional hyperparameters (manually fixed and controlled or AutoML driven variables, which determine the network operation and training structure) such as number of hidden layers, number of neurons at each layer, choice of activation function at various places, number of epochs, batch size, etc., solve the problem of reuse (generalization, inheritance) only partially. With any hyperparameters, the mathematical essence of each functional component of the network remains the same. We also believe that mathematical description of a CNN must have (in addition to weights and hyperparameters) at least one additional type of parameter for each of its component functions to enable higher level control over the mathematical properties of the function and which will be reusable (or inheritable aka genetic code) for other problems and datasets.

In this paper, we aim to explore the generalized and parameter-driven mean functions (Lehmer and Power means) to replace (where appropriate) the inflexible weighted sums or averages within mathematical functional components of CNNs. Our assumption checked within this study (mainly by mathematical proofs) was that the additional parameters could be trained (together with the traditional weights) and used for making CNNs more flexible, not only to better fit the training data but also to better generalize with the previously unseen data. An additional expected bonus provided by such flexibility would be better robustness of the CNNs towards adversarial perturbations (attacks) of the test data.

The remaining text of the paper will be organized as follows: in Section 2, we introduce the generalized (Lehmer and Power) mean functions, prove and study useful properties of these functions and their derivatives; in Section 3, we demonstrate the use and advantages of the generalized (Lehmer and Power) mean functions as a generalized pooling function in CNNs (instead of inflexible max-pooling); in Section 4, we demonstrate (both mathematically and experimentally) the use and advantages of the generalized mean functions as a generalized convolution function in CNNs (instead of inflexible traditional convolution); in Section 5, we demonstrate the use and advantages of the generalized mean functions as a generalized data preprocessing (normalization) functions; in Section 6, we study similar generalizations regarding the inflexible ReLU and SoftMax activation functions; in Section 7, we suggest the generalized and more flexible version of a neuron (e.g. from the fully connected layers of a CNN architecture) by applying the generalized

Lehmer vs. Power means; in Section 8, we put all the generalized puzzles together to define a hyper-flexible CNN architecture and we also provide some hints and possible tricks to train such architecture with weights and multiple extra parameters; in Section 9, we briefly discuss some challenges and opportunities of the proposed architecture; and we conclude in Section 10. More technical details and the code can be found at: https://github.com/Adversarial-Intelligence-Group/flexnets.

Taking into account that modifications of the conceptually different components of the CNN architecture will be discussed in this paper, the majority of related work will be referred to throughout the paper within the corresponding subsections.

2. Introducing the generalized (Lehmer and power) mean functions

To prepare the necessary background for moving towards generalized means, we briefly recall the concepts of arithmetic and quasi-arithmetic means.

Arithmetic Mean (AM) of a set of real numbers $x_1, x_2, \ldots, x_n$ is known to be represented by formula (1).

$$AM\left(x_1, x_2, \ldots, x_n\right) = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{\sum_{i=1}^{n} x_i}{n}. \tag{1}$$

The Kolmogorov Mean (KM), defined by Kolmogorov (1930) or a Quasi-Arithmetic Mean of a set of real numbers, is the generalization of AM, computed according to the formula (2), where φ is any injective, continuous and strictly monotonic function and φ^{-1} is an inverse of it.

$$\begin{aligned} KM\left(x_1, x_2, \ldots, x_n\right) &= \varphi^{-1}\left\{AM\left[\varphi\left(x_1\right), \varphi\left(x_2\right), \ldots, \varphi\left(x_n\right)\right]\right\} \\ &= \varphi^{-1}\left[\frac{\sum_{i=1}^{n} \varphi\left(x_i\right)}{n}\right]. \end{aligned} \tag{2}$$

Therefore, Kolmogorov Mean represents a set of possible mean functions depending on the choice of function φ. For the case when $\varphi = x$ and, therefore, its inverse $\varphi^{-1} = x$, we have $KM\left(x_1, x_2, \ldots, x_n\right) = AM\left(x_1, x_2, \ldots, x_n\right)$. If, for example, φ would be a simple power function: $\varphi = x^{\delta}$ $(\delta \neq 0)$; and, therefore, its inverse would be: $\varphi^{-1} = x^{\frac{1}{\delta}}$, then, according to formula (2), we have:

$$\begin{aligned} KM^{[\varphi \to x^{\delta}]}\left(x_1, x_2, \ldots, x_n\right) &= \left\{AM\left[x_1^{\delta}, x_2^{\delta}, \ldots, x_n^{\delta}\right]\right\}^{\frac{1}{\delta}} \\ &= \left(\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^{\delta}\right)^{\frac{1}{\delta}}, \end{aligned}$$

which is a particular case $KM^{[\varphi \to x^{\delta}]}$ of the Kolmogorov Mean and which is actually a Power Mean function (see more in Section 2.2).

There were also some studies trying to extend the scope of the Kolmogorov Mean by replacing the Arithmetic Mean within formula (2) with some other popular means (Chakrabarty, 2018).

2.1. Lehmer mean and its generalization

Now let us focus on the Lehmer Mean (LM) popular mean function (presented in formula (3)) aiming to explore the unique and useful properties of it (Beliakov & Špirková, 2016; Terziyan, 2017).

$$LM_{\beta}\left(x_1, x_2, \ldots, x_n\right) = \frac{x_1^{\beta+1} + x_2^{\beta+1} + \cdots + x_n^{\beta+1}}{x_1^{\beta} + x_2^{\beta} + \cdots + x_n^{\beta}} = \frac{\sum_{i=1}^{n} x_i^{\beta+1}}{\sum_{i=1}^{n} x_i^{\beta}}. \tag{3}$$

Firstly, we suggest using LM (formula (3)) as a replacement for AM in formula (2). Therefore, we get the following LM-based modification of the KM basic formula (2) (i.e., Quasi-Lehmer Mean or $KM^{[AM \to LM_{\beta}]}$ instead of Quasi-Arithmetic Mean or KM): $KM^{[AM \to LM_{\beta}]}\left(x_1, \ldots, x_n\right) = \varphi^{-1}\left\{LM_{\beta}\left[\varphi\left(x_1\right), \ldots, \varphi\left(x_n\right)\right]\right\} = \varphi^{-1}\left\{\frac{\sum_{i=1}^{n}[\varphi(x_i)]^{\beta+1}}{\sum_{i=1}^{n}[\varphi(x_i)]^{\beta}}\right\}$, which is a particular case of KM from formula (2) where AM from formula (1) is replaced with LM_{β} from formula (3). This would be a valid replacement because the properties of LM match with the basic requirements for the mean functions (Beliakov & Špirková, 2016; Terziyan, 2017) as well as AM does.

Secondly, we can make various modifications of the Quasi-Lehmer Mean by manipulating with φ function as a parameter the same way as we manipulated with φ to get various modifications of the Quasi-Arithmetic Mean above. To make the modification of the Quasi-Lehmer Mean, we suggest φ to be a generic exponent function ($\varphi = \alpha^x$) and, therefore, its inverse φ^{-1} would be a logarithm function ($\varphi^{-1} = \log_{\alpha} x$). As a result of such modifications, we introduce a new mean function represented by formula (4) as a Generalized Lehmer Mean (GLM) with two parameters (real numbers): $\alpha > 1$ and β:

$$\begin{aligned} &\forall \alpha\ (\alpha \in \mathbb{R};\ \alpha > 1), \forall \beta\ (\beta \in \mathbb{R}), \forall n\left(n \in \mathbb{N}^{+}\right), \forall x_i\ (x_i \in \mathbb{R}), \\ &GLM_{\beta}^{\alpha}\left(x_1, x_2, \ldots, x_n\right) = KM^{\begin{bmatrix} AM \to LM_{\beta} \\ \varphi \to \alpha^x \end{bmatrix}}\left(x_1, x_2, \ldots, x_n\right) \\ &= \log_{\alpha}\left\{LM_{\beta}\left[\alpha^{x_1}, \alpha^{x_2}, \ldots, \alpha^{x_n}\right]\right\} \\ &= \log_{\alpha}\left[\frac{\alpha^{(\beta+1)\cdot x_1} + \alpha^{(\beta+1)\cdot x_2} + \cdots + \alpha^{(\beta+1)\cdot x_n}}{\alpha^{\beta\cdot x_1} + \alpha^{\beta\cdot x_2} + \cdots + \alpha^{\beta\cdot x_n}}\right] \\ &= \frac{\ln\left[\frac{\sum_{i=1}^{n} \alpha^{(\beta+1)\cdot x_i}}{\sum_{i=1}^{n} \alpha^{\beta\cdot x_i}}\right]}{\ln \alpha}. \end{aligned} \tag{4}$$

One reason for choosing the exponent function ($\varphi = \alpha^x$) was to enable arbitrary real numbers as the values of vector $\vec{x}$ unlike the traditional Lehmer Mean (3), which requires these values to be strictly positive.

2.2. Power mean and its generalization

Other popular mean function could be modified using similar logic as above. One may use the Power Mean (PM) or Hölder Mean (Bullen, 2013), as represented in formula (5), as a replacement of the Arithmetic Mean in formula (2).

$$PM_{\delta}\left(x_1, x_2, \ldots, x_n\right) = \left(\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^{\delta}\right)^{\frac{1}{\delta}}. \tag{5}$$

Firstly, we get the following PM-based modification of the KM basic formula (2) (i.e., Quasi-Power Mean or $KM^{[AM \to PM_{\delta}]}$ instead of Quasi-Arithmetic Mean or KM):

$$\begin{aligned} KM^{[AM \to PM_{\delta}]}\left(x_1, \ldots, x_n\right) &= \varphi^{-1}\left\{PM_{\delta}\left[\varphi\left(x_1\right), \ldots, \varphi\left(x_n\right)\right]\right\} \\ &= \varphi^{-1}\left\{\left(\frac{1}{n} \cdot \sum_{i=1}^{n}\left[\varphi\left(x_i\right)\right]^{\delta}\right)^{\frac{1}{\delta}}\right\}, \end{aligned}$$

which is a particular case of KM from formula (2) where AM from formula (1) is replaced with PM_{δ} from formula (5). This would be a valid replacement because the properties of PM match with the basic requirements for the mean functions (Bullen, 2013) as well as AM does.

Secondly, we can make various modifications of the Quasi-Lehmer Mean by manipulating with φ function as a parameter the same way as we manipulated with φ to get various modifications of the Quasi-Arithmetic Mean above. To make the modification of the Quasi-Lehmer Mean, we suggest φ to be a generic exponent

function ($\varphi = \gamma^x$) and, therefore, its inverse φ^{-1} would be a logarithm function ($\varphi^{-1} = \log_\gamma x$). As a result of such modification, we introduce a new mean function represented by formula (6) as a Generalized Power Mean (*GPM*) with two parameters (real numbers): $\gamma > 1$ and $\delta \neq 0$ (it is known that if $\delta \to 0$, then Power Mean → Geometric Mean).

$$\forall\gamma\ (\gamma \in \mathbb{R};\ \gamma > 1)\,, \forall\delta\ (\delta \in \mathbb{R})\,, \forall n(n \in \mathbb{N}^+), \forall x_i\ (x_i \in \mathbb{R})\,,$$

$$GPM_\delta^\gamma\ (x_1, x_2, \ldots, x_n) = KM\begin{bmatrix} AM \to PM_\delta \\ \varphi \to \gamma^x \end{bmatrix}$$

$$= \log_\gamma \left\{PM_\delta\left[\gamma^{x_1}, \gamma^{x_2}, \ldots, \gamma^{x_n}\right]\right\}$$

$$= \log_\gamma \left[\left(\frac{1}{n}\cdot\sum_{i=1}^{n}\gamma^{x_i\cdot\delta}\right)^{\frac{1}{\delta}}\right]$$

$$= \frac{\ln\left(\sum_{i=1}^{n}\gamma^{x_i\cdot\delta}\right) - \ln n}{\delta\cdot\ln\gamma}. \tag{6}$$

Here we have the same reason for choosing the exponent function ($\varphi = \gamma^x$), i.e., to enable arbitrary real numbers for the $\vec{x}$ vector unlike the traditional Power Mean (5), which requires these values to be strictly positive.

What is interesting regarding these generalized mean functions (4) and (6)? The Generalized Lehmer Mean (also the Generalized Power Mean) could be considered as a group of "children" of two functions as "parents": (1) the core Kolmogorov Mean, which provides for its children a useful set of properties and the controlling parameter α (aka "gene"); (2) the Lehmer Mean (also the Power Mean), which also provides its unique and useful set of properties and the controlling parameter β. Each possible pair of the parameter values or a unique "genetic code" (α, β) represents one particular instance of the Generalized Lehmer Mean (also the Generalized Power Mean) family. In the context of neural architectures, such a generalization with two additional parameters, which includes other popular mean functions as special cases, provides very useful additional flexibility in controlling the behavior of architectural components. See, for example, how Amari (2007) defines a one-parameter family of stochastic models' integration (called α-integration), which could explain how the brain handles mixtures of probability distributions and which involves various averages of numbers such as arithmetic, geometric, and harmonic ones.

Brief summary of the subsection: The original *KM* (formula (2)) has been defined as a Quasi-Arithmetic Mean; it has *AM* (formula (1)) as a fixed (aka constant) kernel mean function. *KM* also has a flexible parameter φ, which could be an arbitrary invertible function. Due to this parameter, different mean functions can be obtained by choosing a particular function φ. We, however, decided to change the status of the kernel *AM* function in formula (2) and treat it also as a variable, which could be an arbitrary mean function (not just *AM*). For example, by choosing *LM* (formula (3)) with flexible parameter β as a kernel mean function instead of *AM*, and by choosing φ to be a generic exponent function ($\varphi = \alpha^x$), we got a new generalized mean function *GLM* (formula (4)) with two flexible parameters α and β. Similarly, by choosing *PM* (formula (5)) with flexible parameter δ as a kernel mean function instead of *AM*, and by choosing φ to be a generic exponent function ($\varphi = \gamma^x$), we got a new generalized mean function *GPM* (formula (6)) with two flexible parameters γ and δ. Due to the known and unique properties of *LM* and *PM*, we expect the corresponding generalized functions *GLM* and *GPM* to have potentially useful properties for CNN analytics, which will be studied in the following sections.

2.3. Useful properties of the generalized means

Let us prove several important facts regarding generalized functions (4) and (6) and their behavior.

Theorem 1-A.

$$\forall\alpha\ (\alpha \in \mathbb{R};\ \alpha > 1)\,, \forall\beta\ (\beta \in \mathbb{R})\,, \forall n\left(n \in \mathbb{N}^+\right), \forall x_i\ (x_i \in \mathbb{R})\,,$$
$$\text{MIN}\ (x_1, x_2, \ldots, x_n) \le GLM_\beta^\alpha\ (x_1, x_2, \ldots, x_n)$$
$$\le \text{MAX}\ (x_1, x_2, \ldots, x_n)\,.$$

Proof.

$$GLM_\beta^\alpha\ (x_1, x_2, \ldots, x_n) = \log_\alpha\left\{LM_\beta\left[\alpha^{x_1}, \alpha^{x_2}, \ldots, \alpha^{x_n}\right]\right\}.$$

According to the Lehmer Mean properties (Terziyan, 2017):

$$\min_i\left[\alpha^{x_i}\right] \le LM_\beta\left[\alpha^{x_1}, \alpha^{x_2}, \ldots, \alpha^{x_n}\right] \le \max_i\left[\alpha^{x_i}\right].$$

Due to monotonic growth of the logarithm function with the base $\alpha > 1$:

$$\log_\alpha\left\{\min_i\left[\alpha^{x_i}\right]\right\} \le \log_\alpha\left\{LM_\beta\left[\alpha^{x_1}, \alpha^{x_2}, \ldots, \alpha^{x_n}\right]\right\}$$
$$\le \log_\alpha\left\{\max_i\left[\alpha^{x_i}\right]\right\}.$$

Due to monotonic growth of the exponent function for $\alpha > 1$:

$$\log_\alpha\left\{\alpha^{\min_i x_i}\right\} \le GLM_\beta^\alpha\ (x_1, x_2, \ldots, x_n) \le \log_\alpha\left\{\alpha^{\max_i x_i}\right\}.$$

Finally we have:

$$\min_i x_i \le GLM_\beta^\alpha\ (x_1, x_2, \ldots, x_n) \le \max_i x_i$$
$$\text{or}: \quad \text{MIN}\ (x_1, x_2, \ldots, x_n) \le GLM_\beta^\alpha\ (x_1, x_2, \ldots, x_n)$$
$$\le \text{MAX}\ (x_1, x_2, \ldots, x_n)\,. \quad \square$$

Theorem 1-B.

$$\forall\gamma\ (\gamma \in \mathbb{R};\ \gamma > 1)\,, \forall\delta\ (\delta \in \mathbb{R})\,, \forall n\left(n \in \mathbb{N}^+\right), \forall x_i\ (x_i \in \mathbb{R})\,,$$
$$\text{MIN}\ (x_1, x_2, \ldots, x_n) \le GPM_\delta^\gamma\ (x_1, x_2, \ldots, x_n)$$
$$\le \text{MAX}\ (x_1, x_2, \ldots, x_n)\,.$$

Proof. … is similar to the Proof of Theorem 1-A.

Comment regarding Theorems 1-A and 1-B: These theorems prove the key property, which is required from the new generalized means *GLM* and *GPM* to be eligible mean functions, i.e., they keep their values within the interval of their variables' distribution.

Theorem 2-A. *If $\alpha \to 1$, then the Generalized Lehmer Mean in formula (4) will tend to the Arithmetic Mean for all β, i.e.: $GLM_\beta^1\ (x_1, x_2, \ldots, x_n) = AM\ (x_1, x_2, \ldots, x_n)$.*

Proof. The challenge here is that (if $\alpha \to 1$) formula (4) gives an indeterminate form $\frac{0}{0}$ as follows:

$$GLM_\beta^1\ (x_1, x_2, \ldots, x_n) = \frac{\ln\left[\frac{1^{(\beta+1)\cdot x_1} + 1^{(\beta+1)\cdot x_2} + \ldots + 1^{(\beta+1)\cdot x_n}}{1^{\beta\cdot x_1} + 1^{\beta\cdot x_2} + \ldots + 1^{\beta\cdot x_n}}\right]}{\ln 1} = \frac{0}{0}.$$

Therefore, we are representing formula (4) for this case as a limit, to which we apply the L'Hospital's rule as follows:

$$GLM_\beta^1\ (x_1, x_2, \ldots, x_n)$$
$$= \lim_{\alpha\to 1}\frac{\ln\left[\frac{\alpha^{(\beta+1)\cdot x_1} + \alpha^{(\beta+1)\cdot x_2} + \ldots + \alpha^{(\beta+1)\cdot x_n}}{\alpha^{\beta\cdot x_1} + \alpha^{\beta\cdot x_2} + \ldots + \alpha^{\beta\cdot x_n}}\right]}{\ln\alpha}$$

$$\frac{\partial\left\{\ln\left[\frac{\alpha^{(\beta+1)\cdot x_1}+\alpha^{(\beta+1)\cdot x_2}+\cdots+\alpha^{(\beta+1)\cdot x_n}}{\alpha^{\beta\cdot x_1}+\alpha^{\beta\cdot x_2}+\cdots+\alpha^{\beta\cdot x_n}}\right]\right\}}{\partial\alpha}=\frac{\partial\left\{\ln\left[\frac{\sum\alpha^{(\beta+1)\cdot x}}{\sum\alpha^{\beta\cdot x}}\right]\right\}}{\partial\alpha}$$
$$=\frac{\sum\alpha^{\beta\cdot x}}{\sum\alpha^{(\beta+1)\cdot x}}\cdot\frac{(\beta+1)\cdot\sum\left(x\cdot\alpha^{(\beta+1)\cdot x-1}\right)\cdot\sum\alpha^{\beta\cdot x}-\beta\cdot\sum\left(x\cdot\alpha^{\beta\cdot x-1}\right)\cdot\sum\alpha^{(\beta+1)\cdot x}}{\left(\sum\alpha^{\beta\cdot x}\right)^2}$$
$$=\frac{(\beta+1)\cdot\sum\left(x\cdot\alpha^{(\beta+1)\cdot x-1}\right)\cdot\sum\alpha^{\beta\cdot x}-\beta\cdot\sum\left(x\cdot\alpha^{\beta\cdot x-1}\right)\cdot\sum\alpha^{(\beta+1)\cdot x}}{\left(\sum\alpha^{\beta\cdot x}\right)\cdot\left(\sum\alpha^{(\beta+1)\cdot x}\right)}\overbrace{=}^{\alpha=1}\frac{(\beta+1)\cdot\left(\sum x\right)\cdot n-\beta\cdot\left(\sum x\right)\cdot n}{n\cdot n}=\frac{\sum x}{n}$$
$$=\frac{x_1+x_2+\cdots+x_n}{n}.$$

Box I.

$$=\frac{\frac{\partial_{\alpha=1}\left\{\ln\left[\frac{\alpha^{(\beta+1)\cdot x_1}+\alpha^{(\beta+1)\cdot x_2}+\cdots+\alpha^{(\beta+1)\cdot x_n}}{\alpha^{\beta\cdot x_1}+\alpha^{\beta\cdot x_2}+\cdots+\alpha^{\beta\cdot x_n}}\right]\right\}}{\partial\alpha}}{\frac{\partial_{\alpha=1}\{\ln\alpha\}}{\partial\alpha}}$$

The numerator of the last fraction is as follows ($\frac{\partial\left\{\ln\left[\frac{\alpha^{(\beta+1)\cdot x_1}+\alpha^{(\beta+1)\cdot x_2}+\cdots+\alpha^{(\beta+1)\cdot x_n}}{\alpha^{\beta\cdot x_1}+\alpha^{\beta\cdot x_2}+\cdots+\alpha^{\beta\cdot x_n}}\right]\right\}}{\partial\alpha}$ is given in Box I).

The denominator is simply as follows:

$$\frac{\partial\{\ln\alpha\}}{\partial\alpha}=\frac{1}{\alpha}\overbrace{==}^{\alpha=1}1.$$

We have finally:

$$GLM^1_\beta(x_1,x_2,\ldots,x_n)=\frac{x_1+x_2+\cdots+x_n}{n}$$
$$=AM(x_1,x_2,\ldots,x_n).\quad\square$$

Theorem 2-B. *If $\gamma\to 1$, then the Generalized Power Mean in formula (6) will tend to the Arithmetic Mean for all δ, i.e.: $GPM^1_\delta(x_1,x_2,\ldots,x_n)=AM(x_1,x_2,\ldots,x_n)$.*

Proof. The challenge here is that (if $\gamma\to 1$) formula (6) gives an indeterminate form $\frac{0}{0}$ as follows:

$$GPM^1_\delta(x_1,x_2,\ldots,x_n)=\frac{\ln\left(\sum_{i=1}^n 1^{x_i\cdot\delta}\right)-\ln n}{\delta\cdot\ln 1}=\frac{0}{0}.$$

Therefore, we are representing formula (6) for this case as a limit, to which we apply the L'Hospital's rule as follows:

$$GPM^1_\delta(x_1,x_2,\ldots,x_n)=\lim_{\gamma\to 1}\frac{\ln\left(\sum_{i=1}^n\gamma^{x_i\cdot\delta}\right)-\ln n}{\delta\cdot\ln\gamma}$$
$$=\frac{\partial_{\gamma=1}\left\{\ln\left(\sum_{i=1}^n\gamma^{x_i\cdot\delta}\right)-\ln n\right\}}{\frac{\partial_{\gamma=1}\{\delta\cdot\ln\gamma\}}{\partial\gamma}};$$

$$\frac{\partial\left\{\ln\left(\sum_{i=1}^n\gamma^{x_i\cdot\delta}\right)-\ln n\right\}}{\partial\gamma}=\frac{\delta\cdot\sum_{i=1}^n x_i\cdot\gamma^{x_i\cdot\delta-1}}{\sum_{i=1}^n\gamma^{x_i\cdot\delta}}$$
$$\overbrace{==}^{\gamma=1}\frac{\delta}{n}\cdot\sum_{i=1}^n x_i;\quad\frac{\partial\{\delta\cdot\ln\gamma\}}{\partial\gamma}=\frac{\delta}{\gamma}\overbrace{==}^{\gamma=1}\delta;$$

$$GPM^1_\delta(x_1,x_2,\ldots,x_n)=\frac{\sum_{i=1}^n x_i}{n}=AM(x_1,x_2,\ldots,x_n).\quad\square$$

Comment regarding Theorems 2-A and 2-B: These theorems prove that *AM* is a special case of both generalized means *GLM* and *GPM* given the particular choice of the parameters. This is important for further use of the generalized functions in CNNs because both of them will perform at least as good as any twin component, which is based on *AM* (also weighted one).

As we can see, in the case of $\alpha\to 1$, the Generalized Lehmer Mean tends to the Arithmetic Mean for any real $\beta\in(-\infty,\infty)$ and even: $GLM^{\alpha\to 1}_{\beta\to\mp\infty}(x_1,x_2,\ldots,x_n)\to AM(x_1,x_2,\ldots,x_n)$. However, it will be important to know, what would be the values of the Generalized Lehmer Mean when $\alpha>1$ (for both cases: $\beta\to-\infty$ and $\beta\to\infty$). Similar observations we have for the generalized Power Mean. In the case of $\gamma\to 1$ the Generalized Power Mean tends to the Arithmetic Mean for any real $\delta\in(-\infty,\infty)$ and even: $GPM^{\gamma\to 1}_{\delta\to\mp\infty}(x_1,x_2,\ldots,x_n)\to AM(x_1,x_2,\ldots,x_n)$. However, it will be important to know, what would be the values of the Generalized Power Mean when $\gamma>1$ (for both cases: $\delta\to-\infty$ and $\delta\to\infty$).

Theorem 3-A.

(a) $\forall\alpha\,(\alpha\in\mathbb{R};\ \alpha>1)$, $GLM^\alpha_{\beta\to\infty}(x_1,x_2,\ldots,x_n)\to\mathrm{MAX}(x_1,x_2,\ldots,x_n)$;
(b) $\forall\alpha\,(\alpha\in\mathbb{R};\ \alpha>1)$, $GLM^\alpha_{\beta\to-\infty}(x_1,x_2,\ldots,x_n)\to\mathrm{MIN}(x_1,x_2,\ldots,x_n)$.

Proof (a).

$$GLM^\alpha_{\beta\to\infty}(x_1,x_2,\ldots,x_n)$$
$$=\lim_{b\to\infty}\frac{\ln\left[\frac{\alpha^{(\beta+1)\cdot x_1}+\alpha^{(\beta+1)\cdot x_2}+\cdots+\alpha^{(\beta+1)\cdot x_n}}{\alpha^{\beta\cdot x_1}+\alpha^{\beta\cdot x_2}+\cdots+\alpha^{\beta\cdot x_n}}\right]}{\ln\alpha}$$
$$=(\textit{due to the Lehmer Mean properties})=\frac{\ln\{\max_i[\alpha^{x_i}]\}}{\ln\alpha}$$
$$=(\textit{due to monotonic growth of the exponent function for }\alpha>1)$$
$$=\frac{\ln[\alpha^{\max_i x_i}]}{\ln\alpha}=\frac{\ln\alpha\cdot\max_i x_i}{\ln\alpha}=\max_i x_i.$$

Therefore: $GLM^\alpha_{\beta\to\infty}(x_1,x_2,\ldots,x_n)\to\mathrm{MAX}(x_1,x_2,\ldots,x_n)$. $\square$

Proof (b).

$$GLM^\alpha_{\beta\to\infty}(x_1,x_2,\ldots,x_n)$$
$$=\lim_{b\to-\infty}\frac{\ln\left[\frac{\alpha^{(\beta+1)\cdot x_1}+\alpha^{(\beta+1)\cdot x_2}+\cdots+\alpha^{(\beta+1)\cdot x_n}}{\alpha^{\beta\cdot x_1}+\alpha^{\beta\cdot x_2}+\cdots+\alpha^{\beta\cdot x_n}}\right]}{\ln\alpha}$$
$$=(\textit{due to the Lehmer Mean properties})=\frac{\ln\{\min_i[\alpha^{x_i}]\}}{\ln\alpha}$$
$$=(\textit{due to monotonic growth of the exponent function for }\alpha>1)$$
$$=\frac{\ln\left[\alpha^{\min_i x_i}\right]}{\ln\alpha}=\frac{\ln\alpha\cdot\min_i x_i}{\ln\alpha}=\min_i x_i.$$

Therefore: $GLM^{\alpha}_{\beta\to-\infty}(x_1, x_2, \ldots, x_n) \to \text{MIN}(x_1, x_2, \ldots, x_n)$. □

Theorem 3-B.

(a) $\forall\gamma\ (\gamma \in \mathbb{R};\ \gamma > 1)$, $GPM^{\gamma}_{\delta\to\infty}(x_1, x_2, \ldots, x_n) \to \text{MAX}(x_1, x_2, \ldots, x_n)$;
(b) $\forall\gamma\ (\gamma \in \mathbb{R};\ \gamma > 1)$, $GPM^{\gamma}_{\delta\to-\infty}(x_1, x_2, \ldots, x_n) \to \text{MIN}(x_1, x_2, \ldots, x_n)$.

Proof. … is similar to the Proof of Theorem 3-A.

Comment regarding Theorems 3-A and 3-B: These theorems prove that the value of both generalized means (*GLM* and *GPM*) applied to some input vector of real numbers $\vec{x}$ could be controlled either towards the minimal value from $\vec{x}$ by an infinite decrease of parameter β for *GLM* (or δ for *GPM*), or towards the maximal value from $\vec{x}$ by an infinite increase of parameter β for *GLM* (or δ for *GPM*). This is important for further use of the generalized functions because their outcome can be controlled by their parameters to be within the full spectrum of values from minimal to maximal values of $\vec{x}$.

Theorem 4-A.

(a) $\exists!\beta^*\ (\beta^* \in \mathbb{R};\ -1 \le \beta^* \le 0)$, $GLM^{\alpha\to\infty}_{\beta^*}(x_1, x_2, \ldots, x_n) \to AM(x_1, x_2, \ldots, x_n)$;
(b) $\forall\beta\ (\beta \in \mathbb{R};\ \beta > \beta^*)$, $GLM^{\alpha\to\infty}_{\beta>\beta^*}(x_1, x_2, \ldots, x_n) \to \text{MAX}(x_1, x_2, \ldots, x_n)$;
(c) $\forall\beta\ (\beta \in \mathbb{R};\ \beta < \beta^*)$, $GLM^{\alpha\to\infty}_{\beta<\beta^*}(x_1, x_2, \ldots, x_n) \to \text{MIN}(x_1, x_2, \ldots, x_n)$;
(d) $\forall\alpha\ [(n = 2) \to (\beta^* = -0.5)]$.

Proof (b) partial. Let us arrange $x_1, x_2, \ldots, x_n$ in descending order so that $\underbrace{x_1 = x_2 = \cdots = x_k}_{=\text{MAX}(x_1,x_2,\ldots,x_n)}, x_{k+1}, x_{k+2}, \ldots x_n$ (where $1 \le k \le n$). Then:

$$GLM^{\alpha\to\infty}_{0}(x_1, x_2, \ldots, x_n) = \lim_{\alpha\to\infty} \frac{\ln\left(\frac{\alpha^{x_1}+\alpha^{x_2}+\cdots+\alpha^{x_n}}{n}\right)}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty} \frac{\ln(\alpha^{x_1}+\alpha^{x_2}+\cdots+\alpha^{x_n}) - \ln n}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty} \frac{\ln\left[\alpha^{x_1}\cdot\left(\frac{\alpha^{x_1}}{\alpha^{x_1}}+\cdots+\frac{\alpha^{x_k}}{\alpha^{x_1}}+\frac{\alpha^{x_{k+1}}}{\alpha^{x_1}}\ldots+\frac{\alpha^{x_n}}{\alpha^{x_1}}\right)\right] - \ln n}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty} \frac{\ln\alpha^{x_1} + \ln\left(\frac{\alpha^{x_1}}{\alpha^{x_1}}+\cdots+\frac{\alpha^{x_k}}{\alpha^{x_1}}+\frac{\alpha^{x_{k+1}}}{\alpha^{x_1}}\ldots+\frac{\alpha^{x_n}}{\alpha^{x_1}}\right) - \ln n}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty} \frac{x_1\cdot\ln\alpha + \ln\left(k+\frac{\alpha^{x_{k+1}}}{\alpha^{x_1}}+\cdots+\frac{\alpha^{x_n}}{\alpha^{x_1}}\right) - \ln n}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty} \frac{x_1\cdot\ln\alpha + \ln k - \ln n}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty}\left(x_1 + \frac{\ln k - \ln n}{\ln\alpha}\right) = x_1 = \text{MAX}(x_1, x_2, \ldots, x_n).$$ □

Proof (c) partial. Let us arrange $x_1, x_2, \ldots, x_n$ in ascending order so that $\underbrace{x_1 = x_2 = \cdots = x_r}_{=\text{MIN}(x_1,x_2,\ldots,x_n)}, x_{r+1}, x_{r+2}, \ldots x_n$ (where $1 \le r \le n$). Then:

$$GLM^{\alpha\to\infty}_{-1}(x_1, x_2, \ldots, x_n) = \lim_{\alpha\to\infty} \frac{\ln\left(\frac{n}{\alpha^{-x_1}+\alpha^{-x_2}+\cdots+\alpha^{-x_n}}\right)}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty} \frac{\ln n - \ln\left(\alpha^{-x_1}+\alpha^{-x_2}+\cdots+\alpha^{-x_n}\right)}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty} \frac{\ln n - \ln\left[\alpha^{-x_1}\cdot\left(\frac{\alpha^{-x_1}}{\alpha^{-x_1}}+\cdots+\frac{\alpha^{-x_r}}{\alpha^{-x_1}}+\frac{\alpha^{-x_{r+1}}}{\alpha^{-x_1}}\ldots+\frac{\alpha^{-x_n}}{\alpha^{-x_1}}\right)\right]}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty} \frac{\ln n - \ln\alpha^{-x_1} - \ln\left(\frac{\alpha^{-x_1}}{\alpha^{-x_1}}+\cdots+\frac{\alpha^{-x_r}}{\alpha^{-x_1}}+\frac{\alpha^{-x_{r+1}}}{\alpha^{-x_1}}\ldots+\frac{\alpha^{-x_n}}{\alpha^{-x_1}}\right)}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty} \frac{\ln n + x_1\cdot\ln\alpha - \ln\left(r+\frac{\alpha^{-x_{r+1}}}{\alpha^{-x_1}}+\cdots+\frac{\alpha^{-x_r}}{\alpha^{-x_1}}\right)}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty} \frac{\ln n - \ln r + x_1\cdot\ln\alpha}{\ln\alpha}$$
$$= \lim_{\alpha\to\infty}\left(\frac{\ln r - \ln n}{\ln\alpha} + x_1\right) = x_1 = \text{MIN}(x_1, x_2, \ldots, x_n).$$ □

Proof (a) and finalized (b) and (c). According to (b), we have: $GLM^{\alpha\to\infty}_{0}(x_1, x_2, \ldots, x_n) = \text{MAX}(x_1, x_2, \ldots, x_n)$.

According to (c), we have: $GLM^{\alpha\to\infty}_{-1}(x_1, x_2, \ldots, x_n) = \text{MIN}(x_1, x_2, \ldots, x_n)$.

Due to: $\text{MIN}(x_1, x_2, \ldots, x_n) \le AM(x_1, x_2, \ldots, x_n) \le \text{MAX}(x_1, x_2, \ldots, x_n)$ and taking into account that our *GLM* function is monotonic, we can say that somewhere within the $[-1,0]$ interval there is the only value β^* of variable β such that:

$GLM^{\alpha\to\infty}_{\beta^*}(x_1, x_2, \ldots, x_n) \to AM(x_1, x_2, \ldots, x_n)$. □

From the last statement (and due to monotonic nature of our function), we can directly derive more strict inequalities for (b) and (c) parts of the theorem as follows:

$GLM^{\alpha\to\infty}_{\beta>\beta^*}(x_1, x_2, \ldots, x_n) \to \text{MAX}(x_1, x_2, \ldots, x_n)$;

$GLM^{\alpha\to\infty}_{\beta<\beta^*}(x_1, x_2, \ldots, x_n) \to \text{MIN}(x_1, x_2, \ldots, x_n)$.

□

Proof (d). Consider case when $n = 2$ and $\beta = -0.5$. Then we have:

$GLM^{\alpha}_{-0.5}(x_1, x_2) = \log_\alpha\left[\frac{\alpha^{0.5\cdot x_1}+\alpha^{0.5\cdot x_2}}{\alpha^{-0.5\cdot x_1}+\alpha^{-0.5\cdot x_2}}\right] = \log_\alpha\left[\frac{\alpha^{0.5\cdot x_1}\cdot\alpha^{0.5\cdot x_2}\cdot\left(\alpha^{0.5\cdot x_1}+\alpha^{0.5\cdot x_2}\right)}{\alpha^{0.5\cdot x_1}+\alpha^{0.5\cdot x_2}}\right] = \log_\alpha\left(\alpha^{0.5\cdot x_1+0.5\cdot x_2}\right) = \frac{x_1+x_2}{2} = AM(x_1, x_2)$, which means that (for the cases when $n = 2$) $\beta = -0.5 = \beta^*$ for all α including $\alpha \to \infty$. □

Theorem 4-B.

(a) $\forall\gamma,\ GPM^{\gamma}_{\delta\to0}(x_1, x_2, \ldots, x_n) = AM(x_1, x_2, \ldots, x_n)$.
(b) $\forall\delta\ (\delta \in \mathbb{R};\ \delta < 0)$, $GPM^{\gamma\to\infty}_{\delta}(x_1, x_2, \ldots, x_n) \to \text{MIN}(x_1, x_2, \ldots, x_n)$.
(c) $\forall\delta\ (\delta \in \mathbb{R};\ \delta > 0)$, $GPM^{\gamma\to\infty}_{\delta}(x_1, x_2, \ldots, x_n) \to \text{MAX}(x_1, x_2, \ldots, x_n)$.

Proof (a).

$$GPM^{\gamma}_{0}(x_1, x_2, \ldots, x_n) = \frac{\ln\left(\sum_{i=1}^{n}\gamma^{x_i\cdot0}\right) - \ln n}{0\cdot\ln\gamma} = \frac{\ln n - \ln n}{0\cdot\ln\gamma} = \frac{0}{0};$$

$$GPM^{\gamma}_{\delta\to0}(x_1, x_2, \ldots, x_n) = \lim_{\delta\to0}\frac{\ln\left(\sum_{i=1}^{n}\gamma^{x_i\cdot\delta}\right) - \ln n}{\delta\cdot\ln\gamma} = \frac{\frac{\partial_{\delta=0}\left\{\ln\left(\sum_{i=1}^{n}\gamma^{x_i\cdot\delta}\right) - \ln n\right\}}{\partial\delta}}{\frac{\partial_{\delta=0}\{\delta\cdot\ln\gamma\}}{\partial\delta}};$$

$$\frac{\partial\{\delta\cdot\ln\gamma\}}{\partial\delta} = \ln\gamma \overset{\delta=0}{=\!=} \ln\gamma;$$

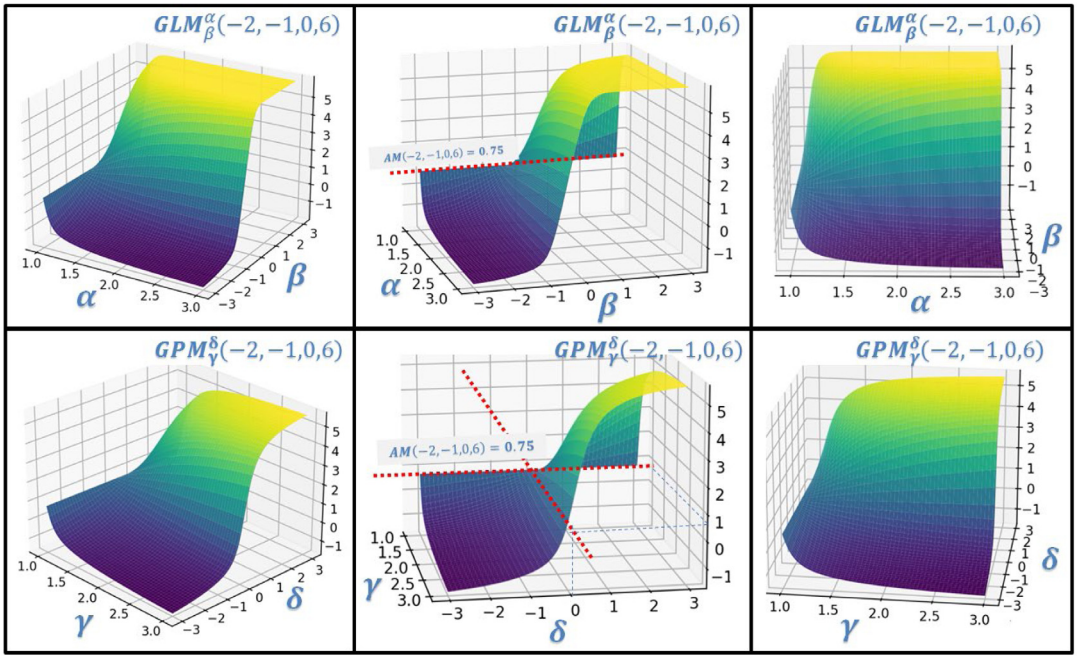

Fig. 1. Comparing the plots of the Generalized Lehmer Mean (regarding parameters Alpha and Beta) and the Generalized Power Mean (regarding parameters Gamma and Delta) for a concrete example of four numbers (−2,−1,0,6). The red straight lines belong to the surface of the corresponding function and indicate where the function is equal to the Arithmetic Mean (which is equal to 0.75) of the four numbers. It can be seen that both functions behave according to the proven theorems and that they have certain similarities with each other.

$$\frac{\partial\left\{\ln\left(\sum_{i=1}^{n}\gamma^{x_i\cdot\delta}\right)-\ln n\right\}}{\partial\delta}=\frac{\partial_{\delta=0}\left[\ln\left(\sum_{i=1}^{n}\gamma^{x_i\cdot\delta}\right)\right]}{\partial\delta}$$
$$=\frac{\ln\gamma\cdot\sum_{i=1}^{n}\left(x_i\cdot\gamma^{x_i\cdot\delta}\right)}{\sum_{i=1}^{n}\gamma^{x_i\cdot\delta}}\overbrace{==}^{\delta=0}\frac{\ln\gamma\cdot\sum_{i=1}^{n}x_i}{n};$$

$$GPM^{\gamma}_{\delta\to 0}\left(x_1,x_2,\ldots,x_n\right)=\frac{\ln\gamma\cdot\sum_{i=1}^{n}x_i}{\ln\gamma\cdot n}=\frac{\sum_{i=1}^{n}x_i}{n}$$
$$=AM\left(x_1,x_2,\ldots,x_n\right).\quad\square$$

Proof (b) and (c). Taking into account the monotonic nature and properties of the Power Mean function, and applying the inequalities from Theorem 1-B, we can derive:

$$\left\{(\gamma_1>\gamma_2)\to\left[GPM^{\gamma_1}_{\delta<0}\left(x_1,x_2,\ldots,x_n\right)\right.\right.$$
$$\left.\left.<GPM^{\gamma_2}_{\delta<0}\left(x_1,x_2,\ldots,x_n\right)\right]\right\}$$
$$\to\left[GPM^{\gamma\to\infty}_{\delta<0}\left(x_1,x_2,\ldots,x_n\right)\to\mathrm{MIN}\left(x_1,x_2,\ldots,x_n\right)\right].\quad\square$$

$$\left\{(\gamma_1>\gamma_2)\to\left[GPM^{\gamma_1}_{\delta>0}\left(x_1,x_2,\ldots,x_n\right)\right.\right.$$
$$\left.\left.>GPM^{\gamma_2}_{\delta>0}\left(x_1,x_2,\ldots,x_n\right)\right]\right\}$$
$$\to\left[GPM^{\gamma\to\infty}_{\delta>0}\left(x_1,x_2,\ldots,x_n\right)\to\mathrm{MAX}\left(x_1,x_2,\ldots,x_n\right)\right].\quad\square$$

Comment regarding Theorems 4-A and 4-B: These theorems prove that there exists a unique value β^* of the parameter β in *GLM* (applied to some input vector of real numbers $\vec{x}$) such that an infinite increase of parameter α forces *GLM* either: (a) towards the minimal value from $\vec{x}$ (if another parameter β of *GLM* has value $\beta<\beta^*$); (b) towards the maximal value from $\vec{x}$ (if $\beta>\beta^*$); or (c) towards the *AM* of $\vec{x}$ (if $\beta=\beta^*$). For the case of *GPM*, infinite increase of parameter γ forces *GPM* either: (a) towards the minimal value from $\vec{x}$ (if another parameter δ of *GPM* has value $\delta<0$); (b) towards the maximal value from $\vec{x}$ (if $\delta>0$); (c) (a) towards the *AM* of $\vec{x}$ (if $\delta=0$). These are important for further use of the generalized functions because their outcome can be controlled by their parameters to be within the full spectrum of values from minimal (through simple average) to maximal values of $\vec{x}$.

Fig. 1 illustrates the behavior of both functions (*GLM* and *GPM*) for a concrete example and in accordance with the proven theorems.

Brief summary of the subsection: One may see (analytically, in accordance with the proven theorems, and visually in Fig. 1) that both functions have almost ideal properties for potential ML purposes: the functions behave monotonically; the desired speed of increase/decrease can be controlled by flexible parameters (both parameters of each function have different influence to the properties of the functions); parameters can be learned (if necessary) within a neural network architecture by backpropagation towards finding the desired behavior of the generalized functions, which could be used for smooth and fast further learning of other neural network parameters.

2.4. Derivatives of the generalized (Lehmer and power) mean functions

Further use of the suggested generalized functions requires formulas for partial derivatives regarding the controlling parameters.

Partial derivative of the *GLM* function $GLM^{\alpha}_{\beta}\left(\vec{x}\right)$ regarding parameter α can be seen in formula (7).

$$\frac{\partial\left[GLM^{\alpha}_{\beta}\left(\vec{x}\right)\right]}{\partial\alpha}=\frac{1}{\alpha\cdot\ln\alpha}\cdot\left\{(\beta+1)\cdot\frac{\sum_{i=1}^{n}\left[x_i\cdot\alpha^{(\beta+1)\cdot x_i}\right]}{\sum_{i=1}^{n}\alpha^{(\beta+1)\cdot x_i}}\right.$$
$$\left.-\beta\cdot\frac{\sum_{i=1}^{n}\left(x_i\cdot\alpha^{\beta\cdot x_i}\right)}{\sum_{i=1}^{n}\alpha^{\beta\cdot x_i}}-GLM^{\alpha}_{\beta}\left(\vec{x}\right)\right\}.\qquad(7)$$

Partial derivative of the *GLM* function $GLM_{\beta}^{\alpha}(\vec{x})$ regarding parameter β can be seen in formula (8).

$$\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\beta}=\frac{\sum_{i=1}^{n}\left[x_i\cdot\alpha^{(\beta+1)\cdot x_i}\right]}{\sum_{i=1}^{n}\alpha^{(\beta+1)\cdot x_i}}-\frac{\sum_{i=1}^{n}\left(x_i\cdot\alpha^{\beta\cdot x_i}\right)}{\sum_{i=1}^{n}\alpha^{\beta\cdot x_i}}. \tag{8}$$

Computations in according to formulas (7) and (8) can be optimized using the following computational schema:

(I) Compute the following four sums within the same $\overrightarrow{1,n}$ cycle:

$$S_1=\sum_{i=1}^{n}\alpha^{\beta\cdot x_i};\ S_2=\sum_{i=1}^{n}\alpha^{(\beta+1)\cdot x_i};\ S_3=\sum_{i=1}^{n}\left(x_i\cdot\alpha^{\beta\cdot x_i}\right);$$

$$S_4=\sum_{i=1}^{n}\left[x_i\cdot\alpha^{(\beta+1)\cdot x_i}\right];$$

(II) Compute the following three values:

$$\boldsymbol{A}=\frac{S_4}{S_2};\quad \boldsymbol{B}=\frac{S_3}{S_1};\quad \boldsymbol{C}=\frac{\ln\frac{S_2}{S_1}}{\ln\alpha};$$

(III) Compute synchronously both partial derivatives from the formulas (7) and (8) as follows:

$$\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\alpha}=\frac{1}{\alpha\cdot\ln\alpha}\cdot\left[(\beta+1)\cdot\boldsymbol{A}-\beta\cdot\boldsymbol{B}-\boldsymbol{C}\right];$$

$$\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\beta}=\boldsymbol{A}-\boldsymbol{B}.$$

Two special cases of zero derivatives are as follows:

(i.) $(x_1=x_2=\cdots=x_n)\ \rightarrow\ \left\{\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\alpha}=\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\beta}=0\right\}$, because:

$$S_1=n\cdot\alpha^{\beta\cdot x_1};\ S_2=n\cdot\alpha^{(\beta+1)\cdot x_1};\ S_3=n\cdot x_1\cdot\alpha^{\beta\cdot x_1};$$

$$S_4=n\cdot x_1\cdot\alpha^{(\beta+1)\cdot x_1};$$

$$\boldsymbol{A}=x_1;\quad \boldsymbol{B}=x_1;\quad \boldsymbol{C}=x_1;$$

$$\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\alpha}=\frac{1}{\alpha\cdot\ln\alpha}\cdot\left[(\beta+1)\cdot\boldsymbol{A}-\beta\cdot\boldsymbol{B}-\boldsymbol{C}\right]$$

$$=\frac{1}{\alpha\cdot\ln\alpha}\cdot\left[(\beta+1)\cdot x_1-\beta\cdot x_1-x_1\right]=0$$

and $\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\beta}=\boldsymbol{A}-\boldsymbol{B}=x_1-x_1=0.$

(ii.) $(\alpha\rightarrow 1)\rightarrow\left\{\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\alpha}=\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\beta}=0\right\}$, because:

$$S_1=n;\quad S_2=n;\quad S_3=\sum_{i=1}^{n}x_i;\quad S_4=\sum_{i=1}^{n}x_i;$$

$$\boldsymbol{A}=\frac{\sum_{i=1}^{n}x_i}{n};\quad \boldsymbol{B}=\frac{\sum_{i=1}^{n}x_i}{n};\quad \boldsymbol{C}=\frac{\sum_{i=1}^{n}x_i}{n}\ \text{(Theorem 2-A)};$$

$$\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\alpha}=\frac{1}{\alpha\cdot\ln\alpha}\cdot\left[(\beta+1)\cdot\boldsymbol{A}-\beta\cdot\boldsymbol{B}-\boldsymbol{C}\right]$$

$$=\frac{1}{\alpha\cdot\ln\alpha}\cdot\left[(\beta+1)\cdot\frac{\sum_{i=1}^{n}x_i}{n}-\beta\cdot\frac{\sum_{i=1}^{n}x_i}{n}-\frac{\sum_{i=1}^{n}x_i}{n}\right]=0$$

and $\frac{\partial\left[GLM_{\beta}^{\alpha}(\vec{x})\right]}{\partial\beta}=\boldsymbol{A}-\boldsymbol{B}=\frac{\sum_{i=1}^{n}x_i}{n}-\frac{\sum_{i=1}^{n}x_i}{n}=0.$

Let us provide similar formalisms regarding the *GPM* function. Partial derivative of the *GPM* function $GPM_{\delta}^{\gamma}(\vec{x})$ regarding parameter γ can be seen in formula (9).

$$\frac{\partial\left[GPM_{\delta}^{\gamma}(\vec{x})\right]}{\partial\gamma}=\frac{1}{\gamma\cdot\ln\gamma}\cdot\left[\frac{\sum_{i=1}^{n}\left(x_i\cdot\gamma^{x_i\cdot\delta}\right)}{\sum_{i=1}^{n}\gamma^{x_i\cdot\delta}}-GPM_{\delta}^{\gamma}(\vec{x})\right]. \tag{9}$$

Partial derivative of the *GPM* function $GPM_{\delta}^{\gamma}(\vec{x})$ regarding parameter δ can be seen in formula (10).

$$\frac{\partial\left[GPM_{\delta}^{\gamma}(\vec{x})\right]}{\partial\delta}=\frac{1}{\delta}\cdot\left[\frac{\sum_{i=1}^{n}\left(x_i\cdot\gamma^{x_i\cdot\delta}\right)}{\sum_{i=1}^{n}\gamma^{x_i\cdot\delta}}-GPM_{\delta}^{\gamma}(\vec{x})\right]. \tag{10}$$

Computations of formulas (9) and (10) can be optimized using the following computational schema:

(I) Compute the following two sums within the same $\overrightarrow{1,n}$ cycle:

$$\boldsymbol{T_1}=\sum_{i=1}^{n}\gamma^{x_i\cdot\delta};\quad \boldsymbol{T_2}=\sum_{i=1}^{n}\left(x_i\cdot\gamma^{x_i\cdot\delta}\right);$$

(II) Compute the following two values:

$$\boldsymbol{D}=\frac{\boldsymbol{T_2}}{\boldsymbol{T_1}};\quad \boldsymbol{E}=\frac{\ln\boldsymbol{T_1}-\ln n}{\delta\cdot\ln\gamma};$$

(III) Compute synchronously both partial derivatives from the formulas (9) and (10) as follows:

$$\frac{\partial\left\{GPM_{\delta}^{\gamma}(\vec{x})\right\}}{\partial\gamma}=\frac{1}{\gamma\cdot\ln\gamma}\cdot(\boldsymbol{D}-\boldsymbol{E});$$

$$\frac{\partial\left\{GPM_{\delta}^{\gamma}(\vec{x})\right\}}{\partial\delta}=\frac{1}{\delta}\cdot(\boldsymbol{D}-\boldsymbol{E}).$$

Three special cases of zero derivatives are as follows:

(i.) $(x_1=x_2=\cdots=x_n)\ \rightarrow\ \left\{\frac{\partial\left[GPM_{\delta}^{\gamma}(\vec{x})\right]}{\partial\gamma}=\frac{\partial\left[GPM_{\delta}^{\gamma}(\vec{x})\right]}{\partial\delta}=0\right\}$, because:

$$\boldsymbol{T_1}=n\cdot\gamma^{x_1\cdot\delta};\quad \boldsymbol{T_2}=n\cdot x_1\cdot\gamma^{x_1\cdot\delta};\quad \boldsymbol{D}=x_1;\quad \boldsymbol{E}=x_1;$$

$$\frac{\partial\left\{GPM_{\delta}^{\gamma}(\vec{x})\right\}}{\partial\gamma}=\frac{1}{\gamma\cdot\ln\gamma}\cdot(x_1-x_1)=0;$$

$$\frac{\partial\left\{GPM_{\delta}^{\gamma}(\vec{x})\right\}}{\partial\delta}=\frac{1}{\delta}\cdot(x_1-x_1)=0.$$

(ii.) $(\gamma\rightarrow 1)\rightarrow\left\{\frac{\partial\left[GPM_{\delta}^{\gamma}(\vec{x})\right]}{\partial\gamma}=\frac{\partial\left[GPM_{\delta}^{\gamma}(\vec{x})\right]}{\partial\delta}=0\right\}$, because:

$$\boldsymbol{T_1}=n;\quad \boldsymbol{T_2}=\sum_{i=1}^{n}x_i;\quad \boldsymbol{D}=\frac{\sum_{i=1}^{n}x_i}{n};\quad \boldsymbol{E}=\frac{\sum_{i=1}^{n}x_i}{n}$$

(Theorem 2-B);

$$\frac{\partial\left\{GPM_{\delta}^{\gamma}(\vec{x})\right\}}{\partial\gamma}=\frac{1}{\gamma\cdot\ln\gamma}\cdot\left(\frac{\sum_{i=1}^{n}x_i}{n}-\frac{\sum_{i=1}^{n}x_i}{n}\right)=0;$$

$$\frac{\partial\left\{GPM_{\delta}^{\gamma}(\vec{x})\right\}}{\partial\delta}=\frac{1}{\delta}\cdot\left(\frac{\sum_{i=1}^{n}x_i}{n}-\frac{\sum_{i=1}^{n}x_i}{n}\right)=0.$$

(iii.) $(\delta\rightarrow 0)\rightarrow\left\{\frac{\partial\left[GPM_{\delta}^{\gamma}(\vec{x})\right]}{\partial\gamma}=\frac{\partial\left[GPM_{\delta}^{\gamma}(\vec{x})\right]}{\partial\delta}=0\right\}$, because:

$$\boldsymbol{T_1}=n;\quad \boldsymbol{T_2}=\sum_{i=1}^{n}x_i;\quad \boldsymbol{D}=\frac{\sum_{i=1}^{n}x_i}{n};\quad \boldsymbol{E}=\frac{\sum_{i=1}^{n}x_i}{n}$$

(Theorem 4-B);

$$\frac{\partial\left\{GPM_{\delta}^{\gamma}(\vec{x})\right\}}{\partial\gamma}=\frac{1}{\gamma\cdot\ln\gamma}\cdot\left(\frac{\sum_{i=1}^{n}x_i}{n}-\frac{\sum_{i=1}^{n}x_i}{n}\right)=0;$$

$$\frac{\partial\left\{GPM_{\delta}^{\gamma}(\vec{x})\right\}}{\partial\delta}=\frac{1}{\delta}\cdot\left(\frac{\sum_{i=1}^{n}x_i}{n}-\frac{\sum_{i=1}^{n}x_i}{n}\right)=0.$$

Brief summary of the subsection and the section in general: One may see that the partial derivatives (for both suggested generalized functions regarding their parameters) are defined

everywhere and behave monotonically within the domain of their parameters. Also, there exists the way to efficiently compute these derivatives. Therefore, learning pairs of parameters for both *GLM* and *GPM* by backpropagation will not be a problem. *GLM* and *GPM* could be a good and flexible replacement (aka generalization) for the (weighted) sum, Max, Min, and dot-product (and, therefore, for pooling, convolution, activation and normalization) fixed operations mainly used in the traditional CNN analytics.

3. Pooling in convolutional neural networks as generalized means

Convolutional layers of a Convolutional Neural Network (CNN) summarize the presence of certain features in some given image by systematically applying learned filters, and, as a result, they produce the corresponding set of the feature maps. These feature maps, as such, are sensitive to the location of the features in the image. This sensitivity can be addressed by down-sampling the feature maps, making them more robust to changes in the position of the feature within the image (Brownlee, 2019). Adding pooling layers into a CNN architecture is one popular approach to reduce dimension (or down-sample) each feature map, aiming to get a summary of the presence of a particular feature within the batches (groups of adjacent pixels within the square size sub-regions) of the feature map. The pooling operation involves sliding (with a certain step or *stride*) a two-dimensional square filter (or *kernel*) over the feature map and summarizing the features lying within the batch covered by the filter.

3.1. Generalized pooling functions

Traditionally, the pooling (summarization) operation is fixed, rather than being learned. Two common pooling functions are *average pooling* and *max pooling*, which summarize the average and most active presence of a feature in each batch, respectively. Such inflexible operations cannot guarantee the minimization of training errors. To overcome this drawback, Sun et al. (2017) propose a learned pooling operation obtained by end-to-end training which is called LEAP (LEArning Pooling). According to their suggested architecture, one shared linear combination of the neurons in the region is learned for each feature map, which makes average pooling just one special case of LEAP pooling (when weights for the neurons are equal). Such an operation would work like a kind of locality-sensitive weighted average with the learning weights, which makes it in a way similar to learning pooling filters or tree pooling as suggested by Lee et al. (2017). Another straightforward way to train pooling operation (to produce values within the interval between the average and max pooling) would be assigning the learnable weight σ to the average pooling value and the corresponding weight $1-\sigma$ to the max pooling value and compute the pooling outcome as a weighted average of both. Such an approach is known as a mixed max-average pooling (Lee et al., 2017).

In this paper, we suggest another approach to the flexibility of pooling operations. We take into account the properties of *GLM* and *GPM* (i.e., their non-linear monotonic behavior within the min–max interval controlled by two parameters) and apply such means instead of fixed pooling operations. The benefit of such flexibility would be the possibility of learning such values for controlling parameters, which would provide the best fit for the particular training image datasets and potentially improve the overall performance of image classification. A very important advantage of such pooling is that our generalized means (applied as pooling operations) will take into account not only the values from the pooling filter but also the distribution of these values.

Let us see in Fig. 2 how the *GLM* as pooling operation can be controlled by its parameters (α and β) for a particular 2×2 pooling filter ($x_1=2, x_2=5, x_3=5, x_4=8$). Similar controlling effects could be expected also if pooling with the *GPM*.

In Fig. 2(a), one can see that the change of β from negative to positive values results on changing the Generalized Lehmer Mean value as a pooling outcome from MIN pooling to MAX pooling via AVERAGE pooling. The speed of this change is controlled by parameter α (i.e., for $\alpha=1$ there is no change and if α grows the corresponding speed grows). In Fig. 2(b), one can see that growing α could make the pooling outcome to: (a) stay unchanged at AVERAGE pooling when $\beta=-0.5$; (b) grow from AVERAGE pooling to MAX pooling when $\beta>-0.5$ (the more β the faster grow); (c) decrease from AVERAGE pooling to MIN pooling when $\beta<-0.5$ (the less β the faster decrease).

The very important feature of the generalized means is their dependence on the distribution of the inputs, i.e., their deviation from the mean. Consider the example of two 2×2 pooling filters: ($x_1=2, x_2=5, x_3=5, x_4=8$) and ($x_1=2, x_2=3, x_3=7, x_4=8$), both of which will return the same values for MAX pooling, AVERAGE pooling and MIN pooling. However, the values of the generalized pooling with the same parameters for both filters will be different because of different standard deviation of values within the filters:

$$\begin{aligned}&\text{MINPOOLING}\,(2,5,5,8)=\text{MINPOOLING}\,(2,3,7,8)\\&=GLM^{\alpha\to\infty}_{\beta<-0.5}\,(2,5,5,8)\\&=GLM^{1<\alpha<\infty}_{-\infty<\beta<-0.5}\,(2,3,7,8)=GLM^{\alpha}_{\beta\to-\infty}\,(2,5,5,8)\\&=GLM^{1<\alpha<\infty}_{-\infty<\beta<-0.5}\,(2,3,7,8)=2;\end{aligned}$$

$$\begin{aligned}&\text{AVERAGEPOOLING}\,(2,5,5,8)=\text{AVERAGEPOOLING}\,(2,3,7,8)\\&=GLM^{\alpha}_{-0.5}\,(2,5,5,8)\\&=GLM^{\alpha}_{-0.5}\,(2,3,7,8)=GLM^{1}_{\beta}\,(2,5,5,8)\\&=GLM^{1}_{\beta}\,(2,3,7,8)=5;\end{aligned}$$

$$\begin{aligned}&\text{MAXPOOLING}\,(2,5,5,8)=\text{MAXPOOLING}\,(2,3,7,8)\\&=GLM^{\alpha\to\infty}_{\beta>-0.5}\,(2,5,5,8)\\&=GLM^{1<\alpha<\infty}_{-\infty<\beta<-0.5}\,(2,3,7,8)=GLM^{\alpha}_{\beta\to\infty}\,(2,5,5,8)\\&=GLM^{1<\alpha<\infty}_{-\infty<\beta<-0.5}\,(2,3,7,8)=8.\end{aligned}$$

However, due to the harmonic/contraharmonic effect (Terziyan, 2017) of the Lehmer Mean (the parent of *GLM*), we have:

$$GLM^{1<\alpha<\infty}_{-\infty<\beta<-0.5}\,(2,5,5,8)>GLM^{1<\alpha<\infty}_{-\infty<\beta<-0.5}\,(2,3,7,8)\,,\ \text{and}$$
$$GLM^{1<\alpha<\infty}_{-0.5<\beta<\infty}\,(2,5,5,8)<GLM^{1<\alpha<\infty}_{-0.5<\beta<\infty}\,(2,3,7,8)\,,$$
$$\text{because}\ \sigma\,(2,5,5,8)<\sigma\,(2,3,7,8)\,,$$

where σ is a standard deviation of the corresponding numbers.

For the *GPM* function, the last two inequalities will look like this:
$GPM^{1<\gamma<\infty}_{-\infty<\delta<0}\,(2,5,5,8)>GPM^{1<\gamma<\infty}_{-\infty<\delta<0}\,(2,3,7,8)$, and
$GPM^{1<\gamma<\infty}_{0<\delta<\infty}\,(2,5,5,8)<GPM^{1<\gamma<\infty}_{0<\delta<\infty}\,(2,3,7,8)$, because of the same reason.

The flexibility of the generalized pooling in general must bring some improvement to the performance of the CNNs, which use pooling operation, comparably to the fixed MAX or AVERAGE pooling, both of which have certain drawbacks as shown in Fig. 3. One can see in Fig. 3, (a) and (b), parts of which are adopted from the figure by Yu et al. (2014), that both MAX Pooling (a) and AVERAGE Pooling (b) may have their drawbacks. However, one

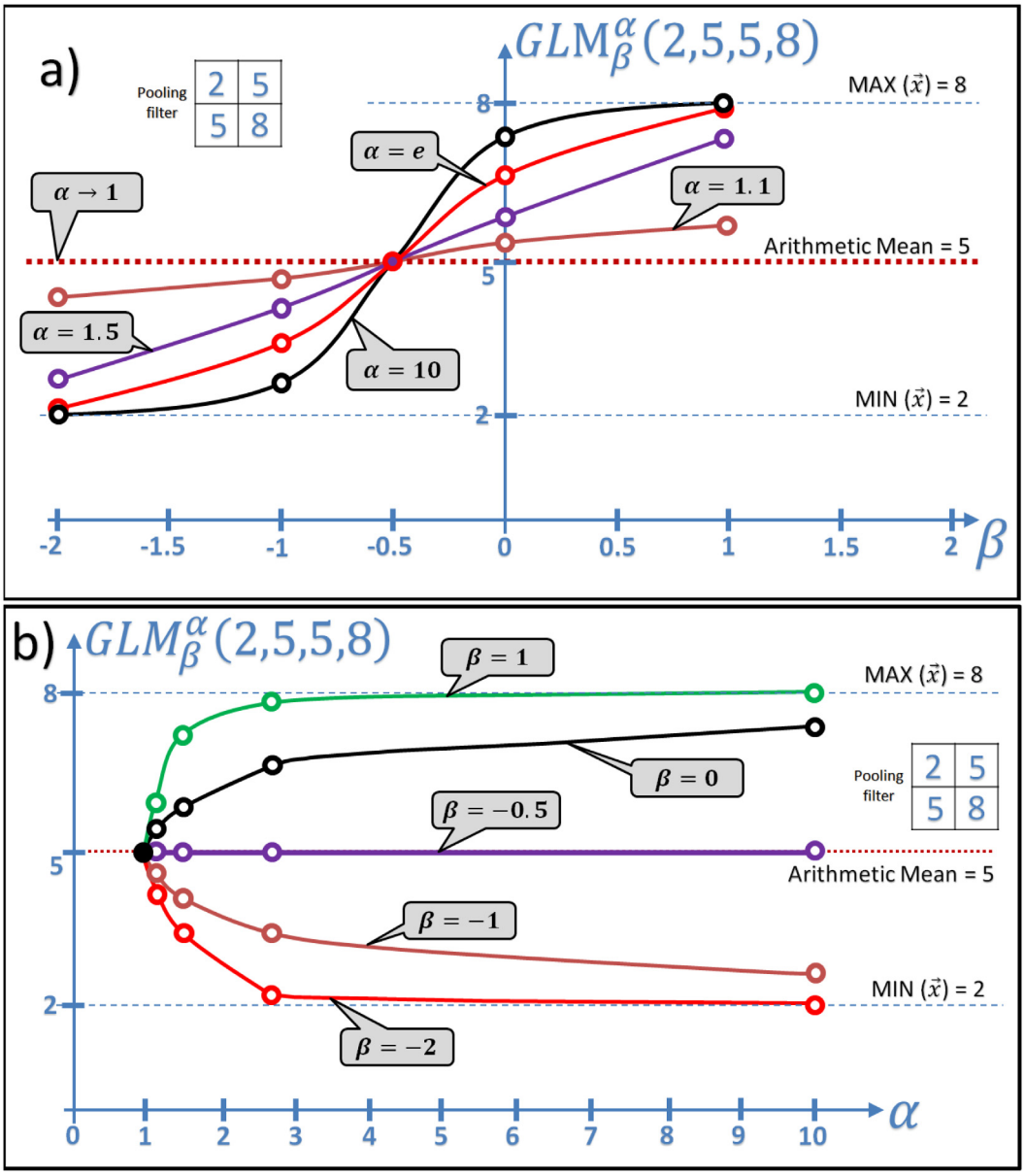

Fig. 2. Visualizing behavior of *GLM* as pooling operation for a particular example of pooling filter. One can see how the generalized pooling can be controlled by changing: (a) parameter BETA; or (b) parameter ALPHA.

can see that the trainable Generalized Pooling (either *GLM*-based or *GPM*-based) will perform equally well (c) in both toxic cases.

If the pooling operation is applied to such image datasets and with such image labeling, for which the fixed MAX pooling is not the best option, it is reasonable to apply the generalized pooling functions and to automatically learn their optimal parameters (by backpropagation) to improve the classification performance. In any case, with the generalized pooling operations, one can expect some improvement comparably to traditional MAX pooling due to the fact that MAX pooling is just one specific case of generalized pooling. However, for the majority of traditional datasets, such improvement (or impact of various manipulations beyond the MAX pooling operation) could be very small as it has been shown in Bieder et al. (2021).

Brief summary of the subsection: Pooling in CNNs is a natural component for applying generalized mean functions. Depending on the parameters, both *GLM* and *GPM* are capable of working as either MAX, MIN, or AVERAGE pooling functions or in between. These parameters, which essentially influence the behavioral dynamics of the functions, can be learned by backpropagation and, therefore, CNN can discover the most suitable pooling option in different layers.

3.2. Experimenting with the generalized pooling

Taking into account that we are aiming to generalize several core components of CNN architecture using trainable *GLM* and *GPM* as the basic formalisms behind these components, the structure for the different groups of the corresponding proof-of-concept experiments will be quite similar to each other. Therefore, aiming not to burden the article with similarly looking details, we will provide a more complete description regarding the generalized pooling (particularly *GLM*-based) experiments in this subsection and we will provide shorter summaries of experiments with other components in further corresponding sections. More details and the code can be found in the open space here: https://github.com/Adversarial-Intelligence-Group/flexnets.

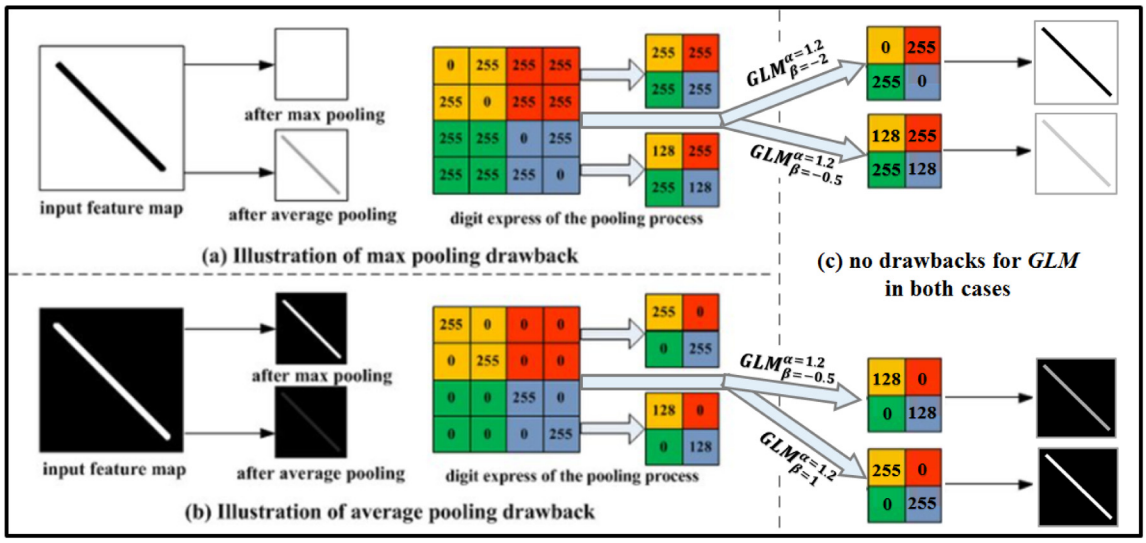

Fig. 3. (a) MAX Pooling drawback example; (b) AVERAGE Pooling drawback example; (c) advantage of the generalized pooling over MAX and AVERAGE pooling in the example.

Experiments with the *GLM*-based Pooling (GLP) have the objectives to address the following questions: (a) Does the use of trainable GLP components in the network lead to better classification accuracy? (b) Does the use of trainable GLP components in the network lead to increased adversarial stability of the network in case of attacks? — If so, what is the relationship between the degree of adaptability of the network and its security? (c) Does the proposed GLP perform this function better than other flexible pooling [as a benchmark for GLP, in addition to the traditional and fixed MaxPool, we used LPPool function (Sermanet et al., 2012; pytorch.org/docs/stable/generated/torch.nn. LPPool2d.html), which is based on Power Mean formula (5) and which we have implemented as an adaptive function with trainable parameter δ]?

Our experiments involved learning separate CNN classifiers for each of the three pooling alternatives (MaxPool, LPPool and GLP). For the experiments, we have chosen CIFAR-10 (Krizhevsky, 2009) as a popular dataset for testing image classifiers. It contains 6000 color images (size: $3 \times 32 \times 32$) labeled into 10 classes.

As a backbone CNN architecture for the experiments, we have chosen the one shown in Fig. 4. It contains three blocks (b1, b2, and b3) of functional components containing the convolution layers, the batch normalization and subsampling, and the fourth block, which acts as a classifier and consists of linear layers. ReLUs are used as activation functions. The network also uses Dropout with a probability of 0.15, to prevent overfitting of the model. The pooling layer in each of three blocks (b1, b2, and b3) is used as a parameter (with the options MaxPool, LPPool and GLP) that is set when a model is created. This makes it more convenient to conduct a series of experiments that require various combinations of pooling options in different blocks of the architecture.

First, let us see the comparison of performance of the three CNNs trained for different pooling options used in all the three blocks (b1, b2, and b3), i.e., MaxPool applied everywhere vs. LPPool applied everywhere vs. GLP applied everywhere. The plots showing the loss of each model depending on the training epochs are shown in Fig. 5.

The corresponding test accuracy for all three CNNs is shown in Table 1.

One can see that the most flexible (two trainable parameters) of three pooling options, the GLP, not only performs better at training, which is not surprising, but also generalizes better than adaptive (one trainable parameter) LPPool and fixed MaxPool.

Table 1
The values for the test accuracy achieved by the three CNNs (with different pooling options) on the CIFAR-10 dataset.

Pooling Model	MaxPool (**b1** & **b2** & **b3**)	LPPool (**b1** & **b2** & **b3**)	GLP (**b1** & **b2** & **b3**)
Accuracy (%)	84.24	84.02	**85.26**

We have also experimented by applying adaptive pooling options not only in all the three blocks of the CNN but locally within single blocks and within pairs of blocks as reported in Table 2. We may see that the most impactful would be applying GLP closer to the "tail" (block **b3**) of the network, i.e., the accuracy for different combinations is ordered as follows:

(**b1** & **b2** & **b3**) > (**b2** & **b3**) > (**b1** & **b3**) > (**b3**) > all other combinations of applying GLP.

At the same time, as it can be seen from Table 2, GLP clearly performs (trains and generalizes) better than less adaptive LPPool in all the combinations, and the scope of applying pooling in the network makes this difference greater.

We performed the next set of experiments to check adversarial robustness (i.e., addressing the adversarial examples) of the CNN classifiers trained with different pooling options. This kind of experiment is important because there is a common opinion that adversarial robustness and generalization of the model are often conflicting objectives (Tsipras et al., 2018) and particularly for the classifiers trained with the CIFAR-10 dataset (Pedraza et al., 2021) where the greater robustness to adversarial examples has been achieved in the overfitting regime (see also Deniz et al., 2020). We, however, being inspired by the optimistic critics from Stutz et al. (2019) regarding the conflict between robustness and generalization, also try to challenge this common opinion experimentally with the GLP-based CNN architecture. In our experiments, an adversarial attack, particularly Projected Gradient Descent (PGD) attack, on the trained networks has been simulated. PGD attack is the strongest white-box attack, which means that the attacker has access to the model's weights and, therefore, to the corresponding gradients, and focused adversarial perturbations of the network input are iteratively generated that can fool the CNN classifier (Madry et al., 2019). PGD has several parameters to control the strength of the attack. The main one is

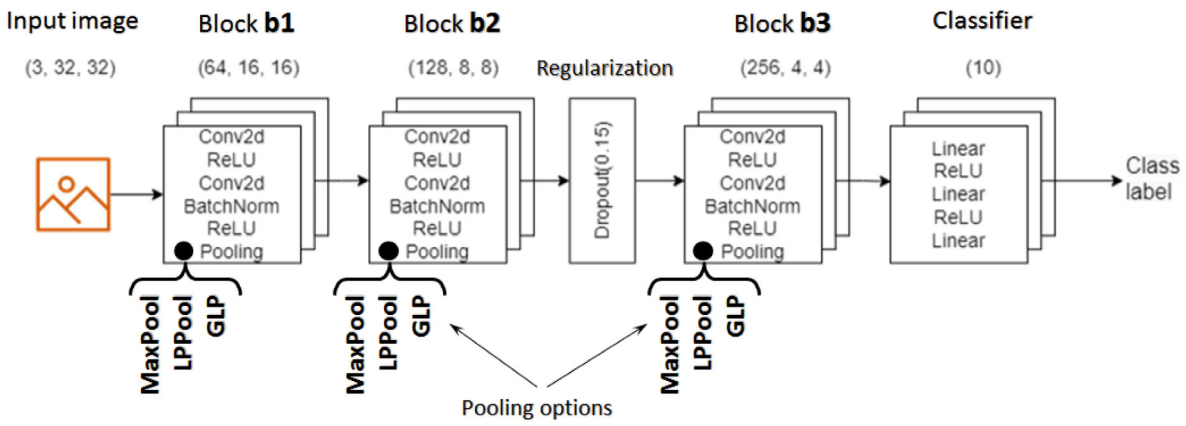

Fig. 4. The basic architecture of CNN with different options of the Pooling function within different blocks used for the experiments.

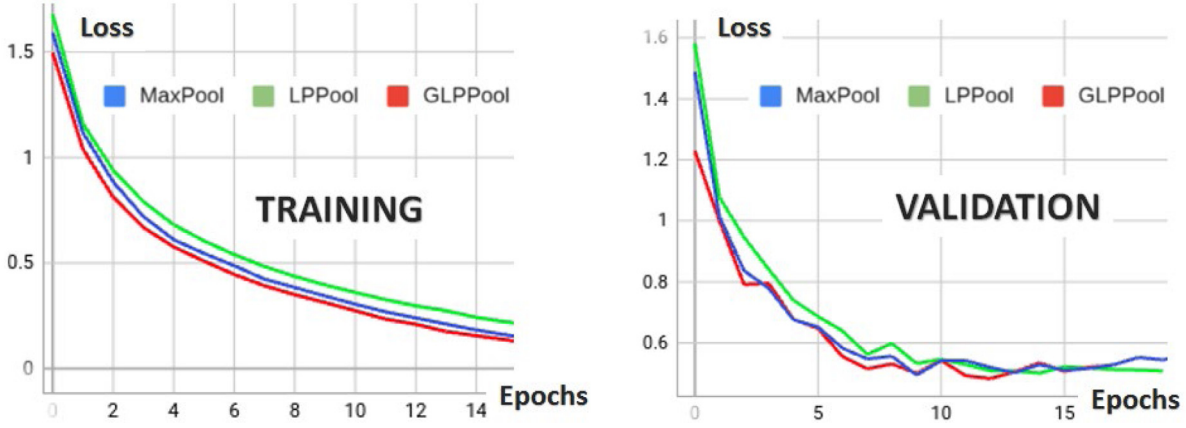

Fig. 5. The loss value dynamics at training and validation on the CIFAR-10 dataset. Three CNNs are compared: one which is using MaxPool in all three architectural blocks; another one, which is using only LPPool, and the last one, which is using GLP everywhere.

Table 2
The values for test accuracy achieved by the CNNs (with three different pooling options applied within different blocks of the network) on the CIFAR-10 dataset.

Accuracy (%)	Blocks in CNN, where pooling has been applied						
	b1	**b2**	**b3**	**b1** & **b2**	**b1** & **b3**	**b2** & **b3**	**b1** & **b2** & **b3**
Pooling Model (MaxPool)	–	–	–	–	–	–	84.24
Pooling Model (LPPool)	83.80	84.28	84.40	84.23	84.25	85.18	84.02
Pooling Model (GLP)	84.04	84.45	84.61	84.42	84.73	85.24	**85.26**

ξ, which controls the magnitude of perturbations. We checked the competitive performance of the three trained CNNs (with different pooling options applied in all the blocks of the network) under ten simulated attacks with gradually amplified magnitude. Fig. 6 illustrates that the flexibility of the pooling option provides better adversarial robustness, i.e., adaptive LPPool performs better than MaxPool, and the most flexible GLP performs better than LPPool with all values of the perturbation magnitude.

More important observations are provided in Table 3. One can see that the stronger the attack is, the greater the impact of pooling flexibility on the adversarial robustness of the model is expected. The last column in the table, where the percentage difference between the accuracy of inflexible MaxPool and flexible GLP [i.e., 100% * (GLP-accuracy minus MaxPool-accuracy)/ MaxPool-accuracy] clearly demonstrates this. This is a very important fact in favor of the use of flexible components based on generalized functions (such as *GLM*) in CNN architecture, which not only improve (not too much but significantly) the generalization performance but also and quite essentially the adversarial robustness of the CNN classifier. The corresponding plot, which demonstrates this impact dynamics, is shown in Fig. 7.

Brief summary of the subsection: Experiments show that suggested functions slightly but clearly improve the generalization performance of the classical CNNs and, what is surprising, they essentially improve the robustness of the CNN architecture against the adversarial inputs.

3.3. Related work on generalizing pooling operation

Let us consider some basic popular pooling functions and the main directions of their generalization in recent studies.

The max pooling function (see, e.g., Krizhevsky et al., 2012), which is considered as the main pooling option and helps in extracting the sharpest low-level features of an image, is as follows:

$$\text{MaxPooling}\,(x_1, x_2, \ldots, x_n) = \max_{\forall i} x_i.$$

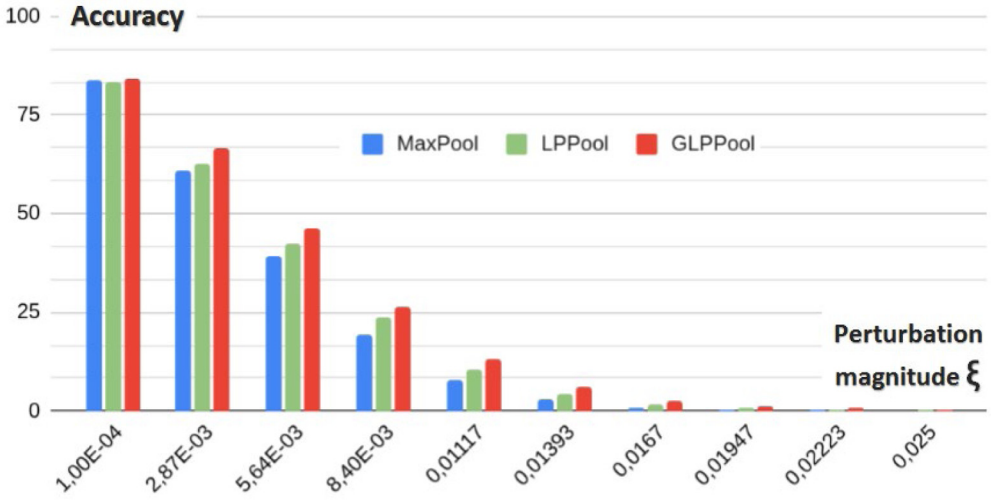

Fig. 6. Dynamics of the validation accuracy decrease for each of the three CNN classifiers (based on either MaxPool, LPPool or GLP) on the adversarial samples depending on the strength of the PGD attack (perturbation magnitude).

Table 3
The comparison of the adversarial robustness achieved by the three CNNs (with different pooling options applied in all three blocks) on the CIFAR-10 dataset.

PGD attack magnitude (ξ)	Pooling Model (MaxPool)	Pooling Model (LPPool)	Pooling Model (GLP)	%-age change (GLP vs. MaxPool)
0.00000	84.24	84.02	85.26	+1.2108
0.00010	83.94	83.21	84.97	+1.2271
0.00287	60.71	62.58	66.35	+9.2901
0.00564	39.00	42.24	46.31	+18.744
0.00840	19.51	23.53	26.36	+35.110
0.01117	7.96	10.42	13.11	+64.699
0.01393	2.89	4.16	5.99	+107.63
0.01670	0.94	1.55	2.59	+175.52
0.01947	0.37	0.59	1.22	+229.73
0.02223	0.20	0.30	0.72	+260.00
0.02500	0.08	0.13	0.30	+275.00

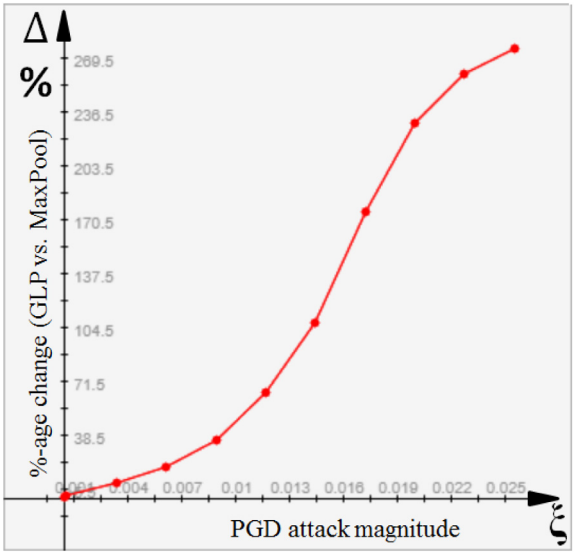

Fig. 7. The plot demonstrates how much (measured by the percentage difference) the GLP-based CNN classifier surpasses (in accuracy) the MaxPool-based CNN classifier depending on the strength of the PGD attack (perturbation magnitude).

The min pooling option is used sometimes (see, e.g., Socher et al., 2011) for cases, which require extracting the most irrelevant features from an image or when the image has a light background (i.e., min pooling is expected to select the darker pixels). The function works as follows:

$$\text{MinPooling}\,(x_1, x_2, \ldots, x_n) = \min_{\forall i} x_i.$$

The average pooling function, which has been introduced for CNNs by LeCun et al. (1998), is as follows:

$$\text{AveragePooling}\,(x_1, x_2, \ldots, x_n) = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i.$$

Generally, average pooling helps to extract the smoothest (rather than sharpest or irrelevant) features.

Babenko and Lempitsky (2015) suggested sum pooling as a straightforward way (with a little risk of overfitting, according to their claim) to aggregate deep convolutional features for image retrieval as follows:

$$\text{SumPooling}\,(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i.$$

Another way to aggregate deep convolutional features could be *L2* pooling (see. e.g., Estrach et al., 2014) as follows:

$$\text{L2Pooling}\,(x_1, x_2, \ldots, x_n) = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

Generalization of the above inflexible functions goes by either combining them together, or applying them in a spatially different scope by manipulating with the hyperparameters, or adding

some extra parameters (either manually controlled or learnable ones) to make these functions more flexible.

For example, Tolias et al. (2016) perform max pooling over regions of the image and finally successfully apply a simple aggregation (either sum pooling or *L2* pooling) of the regional descriptors.

Park et al. (2021) have suggested an elegant generalization of min pooling named conditional min pooling as follows:

$\text{ConditionalMinPooling}\,(x_1, x_2, \ldots, x_n) = \min_{\forall i, x_i^{(\tau)} \neq 0} x_i$, where hyperparameter τ ("tolerance") indicates the percentage of zeros that can be tolerated within the pooling window. Their study claims to solve some issues (e.g., overfitting) of the basic pooling techniques and slightly improve the CNN performance in the crawling data experiments.

A generalized mean pooling function (also known as *Lp* pooling, which is a more general case of *L2* pooling) has been introduced by Gulcehre et al. (2014) as a Power Mean (formula (5) in this paper) as follows:

$$\text{GeneralizedMeanPooling}\,(x_1, x_2, \ldots, x_n) = \text{LpPooling}\,(x_1, x_2, \ldots, x_n) = \left(\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^p\right)^{\frac{1}{p}}.$$

The pooling parameter p can be manually set or learned since this operation is differentiable and can be part of the back-propagation. For example, Radenović et al. (2018) used trainable generalized mean pooling layer in CNN and showed that it boosts image retrieval performance. The MultiGrain network architecture (Berman et al., 2019), used for image classification and object retrieval, contains a generalized mean pooling layer (with manual setup of parameter p) that takes advantage of high-resolution images with a network trained at a lower resolution. By adjusting the pooling layer parameter, they were able to increase the resolution of images used at inference time, while maintaining a small resolution at training time.

Lee et al. (2017) suggested a trainable aggregation of max and average pooling (named as mixed max-average pooling) as follows:

$$\text{MixedMaxAveragePooling}\,(x_1, x_2, \ldots, x_n) = \sigma \cdot \left(\frac{1}{n} \cdot \sum_{i=1}^{n} x_i\right) + (1 - \sigma) \cdot \max_{\forall i} x_i,$$

where σ is a learnable weight. Lee et al. (2017) reported improved CNN performance when mixed max-average pooling was used in place of average or max pooling and that these benefits come with a light increase in computational overhead during training.

Attention-based pooling, with many different modifications (see, e.g., Bi et al., 2019, 2021; Roig et al., 2021) and similarities to the already mentioned learning pooling (LEAP) approach by Sun et al. (2017), has the following mathematical backbone:

$$\text{AttentionBasedPooling}\,(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} \omega_j \cdot x_i, \left(\sum_{i=1}^{n} \omega_i = 1\right),$$

where ω_i are trainable attention weights.

Bi et al. (2019) reported on APDC-Net architecture, which applies such attention-based pooling. Several aerial scene classification experiments demonstrate that this architecture outperformed other state-of-the-art methods with much fewer parameters. Another MS2AP architecture reported in Bi et al. (2021), applied a weighted pooling whose weight matrix comes from the attention module. Extensive experiments demonstrate that the suggested architecture has substantially outperformed current state-of-the-art methods by a large margin. Roig et al. (2021) suggested a method to generate compact image representations through an attention mechanism, named generalized local attention pooling. It is supposed to be placed at an intermediate level of the backbone architecture, resulting in lower memory requirements. Attention weights there generate robust compact image representations. Their experiments show that, in many cases, such a pooling mechanism outperforms other pooling methods.

An interesting and popular generalization of max pooling function is known as K-max pooling (Kalchbrenner et al., 2014). Instead of finding one maximum feature, the function selects and operates over K features with the highest value, i.e.:

$$\text{K-MaxPooling}\,(x_1, x_2, \ldots, x_n) = \frac{1}{K} \cdot \sum_{j=1}^{K} \max_{\forall i} x_i.$$

The weighted K-max pooling function, which is named as a Generalized Pooling Operator by Chen et al. (2021) and used there for visual semantic embedding, uses trainable weights θ_j for aggregating K maximum values as follows:

$$\text{GeneralizedPoolingOperator}\,(x_1, x_2, \ldots, x_n) = \sum_{j=1}^{n} \theta_j \cdot \max_{\forall i} x_i, \left(\sum_{j=1}^{n} \theta_j = 1\right).$$

The global K-max pooling (Hanselmann & Ney, 2020) formulation extends the network with K parameters that regulate the contribution of the K maximal activations to the averaging operation in each feature map as follows:

$$\text{GlobalK-MaxPooling}\,(x_1, x_2, \ldots, x_n) = \frac{1}{K} \cdot \sum_{j=1}^{K} \omega_j \cdot \max_{\forall i} x_i, \left(\sum_{j=1}^{K} \omega_j = 1\right).$$

Such a function is traditionally used in CNNs between the last convolution layer and the classification layer to break efficiently down the spatial dimension of the final convolution layer and into a single vector.

Brief summary of the subsection and the section in general: We proposed a way to compute and flexibly adapt pooling functions based on generalized means, which is capable of discovery of suitable parameters for pooling operation in CNNs by learning. New pooling functions are able to find automatically the way to behave in different layers of the CNN architecture (including MIN, MAX and AVERAGE pooling functions as particular cases of it). Experiments show certain improvements in the generalization performance of these functions in comparison to some classical ones and essential improvement of the robustness against adversarial inputs. Pooling functions suggested in this paper (which are based on *GLM* and *GPM*) are designed so that they are able to learn parameters for such a particular option of the pooling function, which will behave the same way as any of the generalizations described in this subsection. This is due to the possibility to produce smoothly the values from MIN to MAX via different kinds of averages (including the weighted ones if necessary).

4. Convolution as a generalized mean

In this section, we aim to show that the generalized mean functions could work not only as flexible pooling but also as flexible convolution options in CNNs.

4.1. Generalized convolution and its derivatives

As was mentioned above, CNNs consist of convolutional layers, which apply filters (kernels) sliding over the input image to produce feature maps as a result of a convolution operation. The convolution operation (discrete convolution in CNNs) is equivalent to taking a dot product between the vector of filter values and the corresponding vector of values from the input image, i.e., measures the vectors' similarity (pattern match) as a component of the resulting feature map.

The *dot product* $\vec{x} \cdot \vec{y}$ of two vectors $\vec{x} = \{x_1, x_2, \ldots, x_n\}$ and $\vec{y} = \{y_1, y_2, \ldots, y_n\}$ (or a convolution $CONV(\vec{x}, \vec{y})$ of input vector $\vec{x}$ with the filter $\vec{y}$) is defined as:

$$\vec{x} \cdot \vec{y} = CONV\left(\vec{x}, \vec{y}\right) = \sum_{i=1}^{n} (x_i \cdot y_i), \tag{11}$$

which (being applied multiple times according to the procedure of filter sliding over the image) is the core of traditional convolution operation.

Therefore, the convolution function computes the inner product between a weight vector and the vectorized image patches extracted by a sliding window. There were efforts to generalize the convolution operation. For example, Ghiasi-Shirazi (2019) proposed two classes of surrogate functions to generalize the convolution operator, which are based on the convex combination of the dot-product and the negative squared Euclidean distance functions. The parameters of the generalized functions are trained with the back-propagation algorithm. The applicability and reasonable performance of the proposed generalization has been proven by the experiments with the MNIST dataset. A certain kind of trainable convolution has been suggested in Wang and Wang (2018) for a CNN-based multichannel concatenation acoustic model, which extracts speech-specific information from multichannel noisy speech signals. The concatenation is managed by a trainable convolutional layer where the convolution operation (within each channel and which templates are shared among the channels) is a complex multiplication extracting the speech-specific information. Such trainable concatenation models are shown to be more effective comparably to the traditional summation models. We believe that *GLM* (or *GPM*) with trainable parameters could be another good option, which may replace summarization for a similar purpose. Another interpretation of the generalized (trainable) convolution is considered by Zhang, Liu et al. (2022) and Zhang, Weng et al. (2022) as an enabler of arbitrary shapes of the filters in CNNs. A novel generalized convolutional operator dynamically transforms (i.e., learns the shape) the convolution kernels based on the input features and is capable of extracting the transform-invariant features. Such an approach is reported to improve the robustness and generalization of convolution without sacrificing the performance on common images and to outperform the original convolution operation.

While following the concept of flexibility applied in our study, we suggest more generic definitions of dot product (and, therefore, operation of convolution) by using our generalized means *GLM* and *GPM* as follows:

• Generalized Lehmer Dot Product or Generalized Lehmer Convolution (*GLC*):

$$\forall \alpha \left(\alpha \in \mathbb{R};\ \alpha > 1\right), \forall \beta \left(\beta \in \mathbb{R}\right), \forall n (n \in \mathbb{N}^+),$$
$$\forall x_i \left(x_i \in \mathbb{R}, x_i \in \vec{x}\right), \forall y_i \left(y_i \in \mathbb{R}, y_i \in \vec{y}\right)$$
$$GLC_{\beta}^{\alpha}\left(\vec{x}, \vec{y}\right) = n \cdot GLM_{\beta}^{\alpha}\left[(x_1 \cdot y_1), (x_2 \cdot y_2), \ \ldots, (x_n \cdot y_n)\right]$$
$$= n \cdot \log_{\alpha}\left[\frac{\alpha^{(\beta+1)\cdot x_1 \cdot y_1} + \alpha^{(\beta+1)\cdot x_2 \cdot y_2} + \cdots + \alpha^{(\beta+1)\cdot x_n \cdot y_n}}{\alpha^{\beta \cdot x_1 \cdot y_1} + \alpha^{\beta \cdot x_2 \cdot y_2} + \cdots + \alpha^{\beta \cdot x_n \cdot y_n}}\right]$$
$$= \frac{n}{\ln \alpha} \cdot \ln\left[\frac{\sum_{i=1}^{n} \alpha^{(\beta+1)\cdot x_i \cdot y_i}}{\sum_{i=1}^{n} \alpha^{\beta \cdot x_i \cdot y_i}}\right]; \tag{12}$$

• Generalized Power Dot Product or Generalized Power Convolution (*GPC*):

$$\forall \gamma \left(\gamma \in \mathbb{R};\ \gamma > 1\right), \forall \delta \left(\delta \in \mathbb{R}\right), \forall n (n \in \mathbb{N}^+),$$
$$\forall x_i \left(x_i \in \mathbb{R}, x_i \in \vec{x}\right), \ \forall y_i \left(y_i \in \mathbb{R}, y_i \in \vec{y}\right),$$
$$GPC_{\delta}^{\gamma}\left(\vec{x}, \vec{y}\right) = n \cdot GPM_{\delta}^{\gamma}\left[(x_1 \cdot y_1), (x_2 \cdot y_2), \ \ldots, (x_n \cdot y_n)\right]$$
$$= n \cdot \log_{\gamma}\left[\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \gamma^{x_i \cdot y_i \cdot \delta}\right)^{\frac{1}{\delta}}\right]$$
$$= \frac{n}{\delta \cdot \ln \gamma} \cdot \left[\ln\left(\sum_{i=1}^{n} \gamma^{x_i \cdot y_i \cdot \delta}\right) - \ln n\right]. \tag{13}$$

Due to the similarity between the definitions of the generalized means (formulas (4) and (6)) and the generalized convolution (formulas (12) and (13)) operations, it is easy to formulate properties of the generalized convolution using the previously proven theorems as follows:

(a) $n \cdot \max_{i=\overline{1,n}}(x_i \cdot y_i) \le GLC_{\beta}^{\alpha}\left(\vec{x}, \vec{y}\right) \le n \cdot \max_{i=\overline{1,n}}(x_i \cdot y_i)$, which is directly derived from Theorem 1-A;

(b) $n \cdot \max_{i=\overline{1,n}}(x_i \cdot y_i) \le GPC_{\delta}^{\gamma}\left(\vec{x}, \vec{y}\right) \le n \cdot \max_{i=\overline{1,n}}(x_i \cdot y_i)$, which is directly derived from Theorem 1-B;

(c) $GLC_{\beta}^{1}\left(\vec{x}, \vec{y}\right) = CONV\left(\vec{x}, \vec{y}\right)$, which is directly derived from Theorem 2-A;

(d) $GPC_{\delta}^{1}\left(\vec{x}, \vec{y}\right) = CONV\left(\vec{x}, \vec{y}\right)$, which is directly derived from Theorem 2-B;

(e) $GLC_{\beta\to\infty}^{\alpha}\left(\vec{x}, \vec{y}\right) \to n \cdot \max_{i=\overline{1,n}}(x_i \cdot y_i)$; $GLC_{\beta\to-\infty}^{\alpha}\left(\vec{x}, \vec{y}\right) \to n \cdot \min_{i=\overline{1,n}}(x_i \cdot y_i)$, which are directly derived from Theorem 3-A;

(f) $GPC_{\delta\to\infty}^{\gamma}\left(\vec{x}, \vec{y}\right) \to n \cdot \max_{i=\overline{1,n}}(x_i \cdot y_i)$; $GPC_{\delta\to-\infty}^{\gamma}\left(\vec{x}, \vec{y}\right) \to n \cdot \min_{i=\overline{1,n}}(x_i \cdot y_i)$, which are directly derived from Theorem 3-B;

(g) $\exists! \beta^* \left(-1 \le \beta^* \le 0\right), GLC_{\beta^*}^{\alpha\to\infty}\left(\vec{x}, \vec{y}\right) \to CONV\left(\vec{x}, \vec{y}\right)$;
$GLC_{\beta>\beta^*}^{\alpha\to\infty}\left(\vec{x}, \vec{y}\right) \to n \cdot \max_{i=\overline{1,n}}(x_i \cdot y_i)$; $GLC_{\beta<\beta^*}^{\alpha\to\infty}\left(\vec{x}, \vec{y}\right) \to n \cdot \min_{i=\overline{1,n}}(x_i \cdot y_i)$;
$\forall \alpha [(n = 2) \to (\beta^* = -0.5)]$, which are directly derived from Theorem 4-A;

(h) $GPC_{\delta\to0}^{\gamma}\left(\vec{x}, \vec{y}\right) = CONV\left(\vec{x}, \vec{y}\right)$; $GPC_{\delta<0}^{\gamma\to\infty}\left(\vec{x}, \vec{y}\right) \to n \cdot \min_{i=\overline{1,n}}(x_i \cdot y_i)$;
$GPC_{\delta>0}^{\gamma\to\infty}\left(\vec{x}, \vec{y}\right) \to n \cdot \max_{i=\overline{1,n}}(x_i \cdot y_i)$, which are directly derived from Theorem 4-B.

Further use of the suggested generalized convolution functions requires formulas for partial derivatives regarding controlling parameters. Such derivatives could be easily derived following the analogy with the derivatives of the *GLM* and *GPM* functions.

Partial derivative of the Generalized Lehmer Convolution function $GLC_{\beta}^{\alpha}\left(\vec{x}, \vec{y}\right)$ regarding parameter α can be seen in formula (14).

$$\frac{\partial\left[GLC_{\beta}^{\alpha}\left(\vec{x}, \vec{y}\right)\right]}{\partial \alpha}$$
$$= \frac{n}{\alpha \cdot \ln \alpha} \cdot \left\{(\beta+1) \cdot \frac{\sum_{i=1}^{n}\left[x_i \cdot \alpha^{(\beta+1)\cdot x_i \cdot y_i}\right]}{\sum_{i=1}^{n} \alpha^{(\beta+1)\cdot x_i \cdot y_i}}\right.$$
$$\left. - \beta \cdot \frac{\sum_{i=1}^{n}\left(x_i \cdot y_i \cdot \alpha^{\beta \cdot x_i \cdot y_i}\right)}{\sum_{i=1}^{n} \alpha^{\beta \cdot x_i \cdot y_i}} - GLC_{\beta}^{\alpha}\left(\vec{x}, \vec{y}\right)\right\}. \tag{14}$$

Partial derivative of the Generalized Lehmer Convolution function $GLC_{\beta}^{\alpha}\left(\vec{x},\vec{y}\right)$ regarding parameter β can be seen in formula (15).

$$\frac{\partial\left[GLC_{\beta}^{\alpha}\left(\vec{x},\vec{y}\right)\right]}{\partial\beta}=n\cdot\left\{\frac{\sum_{i=1}^{n}\left[x_i\cdot y_i\cdot\alpha^{(\beta+1)\cdot x_i\cdot y_i}\right]}{\sum_{i=1}^{n}\alpha^{(\beta+1)\cdot x_i\cdot y_i}}-\frac{\sum_{i=1}^{n}\left(x_i\cdot\alpha^{\beta\cdot x_i\cdot y_i}\right)}{\sum_{i=1}^{n}\alpha^{\beta\cdot x_i\cdot y_i}}\right\}. \tag{15}$$

Computations of formulas (14) and (15) can be optimized using the following computational schema:

(I) Compute the following four sums within the same $\overrightarrow{1,n}$ cycle:

$$S_1=\sum_{i=1}^{n}\alpha^{\beta\cdot x_i\cdot y_i};\quad S_2=\sum_{i=1}^{n}\alpha^{(\beta+1)\cdot x_i\cdot y_i};$$

$$S_3=\sum_{i=1}^{n}\left(x_i\cdot y_i\cdot\alpha^{\beta\cdot x_i\cdot y_i}\right);\quad S_4=\sum_{i=1}^{n}\left[x_i\cdot y_i\cdot\alpha^{(\beta+1)\cdot x_i\cdot y_i}\right];$$

(II) Compute the following three values:

$$\boldsymbol{A}=\frac{S_4}{S_2};\quad \boldsymbol{B}=\frac{S_3}{S_1};\quad \boldsymbol{C}=\frac{\ln\frac{S_2}{S_1}}{\ln\alpha};$$

(III) Compute synchronously both partial derivatives from formulas (14) and (15) as follows:

$$\frac{\partial\left[GLC_{\beta}^{\alpha}\left(\vec{x},\vec{y}\right)\right]}{\partial\alpha}=\frac{n}{\alpha\cdot\ln\alpha}\cdot\left[(\beta+1)\cdot\boldsymbol{A}-\beta\cdot\boldsymbol{B}-\boldsymbol{C}\right];$$

$$\frac{\partial\left[GLC_{\beta}^{\alpha}\left(\vec{x},\vec{y}\right)\right]}{\partial\beta}=n\cdot(\boldsymbol{A}-\boldsymbol{B}).$$

Now let us make similar manipulations with the *GPC* function. Partial derivative of the $GPC_{\delta}^{\gamma}\left(\vec{x},\vec{y}\right)$ regarding parameter γ can be seen in formula (16).

$$\frac{\partial\left[GPC_{\delta}^{\gamma}\left(\vec{x},\vec{y}\right)\right]}{\partial\gamma}=\frac{n}{\gamma\cdot\ln\gamma}\cdot\left[\frac{\sum_{i=1}^{n}\left(x_i\cdot y_i\cdot\gamma^{x_i\cdot y_i\cdot\delta}\right)}{\sum_{i=1}^{n}\gamma^{x_i\cdot y_i\cdot\delta}}-GPC_{\delta}^{\gamma}\left(\vec{x},\vec{y}\right)\right]. \tag{16}$$

Partial derivative of the $GPC_{\delta}^{\gamma}\left(\vec{x},\vec{y}\right)$ regarding parameter δ can be seen in formula (17).

$$\frac{\partial\left[GPC_{\delta}^{\gamma}\left(\vec{x},\vec{y}\right)\right]}{\partial\delta}=\frac{n}{\delta}\cdot\left[\frac{\sum_{i=1}^{n}\left(x_i\cdot\gamma^{x_i\cdot y_i\cdot\delta}\right)}{\sum_{i=1}^{n}\gamma^{x_i\cdot y_i\cdot\delta}}-GPC_{\delta}^{\gamma}\left(\vec{x},\vec{y}\right)\right]. \tag{17}$$

Computations of formulas (16) and (17) could be optimized using the following computational schema:

(I) Compute the following two sums within the same $\overrightarrow{1,n}$ cycle:

$$\boldsymbol{T_1}=\sum_{i=1}^{n}\gamma^{x_i\cdot y_i\cdot\delta};\quad \boldsymbol{T_2}=\sum_{i=1}^{n}\left(x_i\cdot y_i\cdot\gamma^{x_i\cdot y_i\cdot\delta}\right);$$

(II) Compute the following two values:

$$\boldsymbol{D}=\frac{\boldsymbol{T_2}}{\boldsymbol{T_1}};\quad \boldsymbol{E}=\frac{\ln\boldsymbol{T_1}-\ln n}{\delta\cdot\ln\gamma};$$

(III) Compute synchronously both partial derivatives from the formulas (16) and (17) as follows:

$$\frac{\partial\left[GPC_{\delta}^{\gamma}\left(\vec{x},\vec{y}\right)\right]}{\partial\gamma}=\frac{n}{\gamma\cdot\ln\gamma}\cdot(\boldsymbol{D}-\boldsymbol{E});$$

$$\frac{\partial\left[GPC_{\delta}^{\gamma}\left(\vec{x},\vec{y}\right)\right]}{\partial\delta}=\frac{n}{\delta}\cdot(\boldsymbol{D}-\boldsymbol{E}).$$

Table 4
The values for the test accuracy achieved by the three CNNs (with different convolution options) on the CIFAR-10 dataset.

Convolution Model	CONV (**b1** & **b2** & **b3**)	GLC (**b1** & **b2** & **b3**)	GPC (**b1** & **b2** & **b3**)
Accuracy (%)	84.24	**84.88**	84.35

Table 5
The values for the test accuracy achieved by 9 different configurations of CNNs (with three different convolution options applied within different blocks of the network) on the CIFAR-10 dataset.

Accuracy (%)	Blocks in CNN, where convolution has been applied			
	b1	**b2**	**b3**	**b1** & **b2** & **b3**
Configuration	X	CONV	CONV	X
	CONV	X	CONV	X
	CONV	CONV	X	X
Convol. Model (X = CONV)	84.24	84.24	84.24	84.24
Convol. Model (X = GLC)	84.07	84.12	84.42	**84.88**
Convol. Model (X = GPC)	83.95	84.27	84.35	84.35

Brief summary of the subsection: We have shown that both *GLM* and *GPM* generalized mean functions could be used also as a generalization of the convolution operation (particularly for applying such an update within CNN architectures). The core formalism behind the basic convolution (or a sliding dot-product between the kernel and the section of the signal) simply has the following structure: $\vec{x}\cdot\vec{y}=\sum_{i=1}^{n}(x_i\cdot y_i)=n\cdot AM(\overrightarrow{x\cdot y})$. Therefore, by replacing *AM* with our generalized means, we are obtaining more flexible (controlled by two trainable parameters) convolution operators with much richer properties. Similarly to generalized pooling, the controlling parameters are able to essentially influence the behavioral dynamics of the new convolution and they can be learned by backpropagation due to the monotonic and continuous derivatives. Therefore, we may expect that any CNN while learning can discover the most suitable convolution settings at different layers.

4.2. Summary of the experiments with the generalized convolution

Experiments in this subsection follow the same schema, the same CIFAR-10 dataset and the same skeleton for the CNN architecture (Fig. 4) as the experiments with the generalized pooling (Section 3.2). Here we have three competing options: standard convolution operation *CONV* vs. two suggested generalized options *GLC* and *GPC*.

Table 4 (similarly to Table 1) shows the test accuracy of three CNNs trained with different convolution operators (i.e., *CONV*, *GLC* and *GPC* applied in all three blocks of the architecture). It can be seen that both generalized options slightly improve the performance of the CNN with standard convolution, and *GLC* option performs slightly better than *GPC*.

Table 5 (similarly to Table 2) shows the validation accuracy of various CNNs trained with different convolution operators (i.e., *CONV*, *GLC* and *GPC* applied in different blocks of the architecture). Here we have the convolution operator in all the three blocks and we replace the standard convolution with different generalized options in different places to get 9 different CNN configurations as shown in Table 5. It can be seen that the convolution generalization effect has a stronger impact on the accuracy for the deeper blocks of the network while the *GLC* option in all the configurations performs slightly better than *GPC*.

Experiments with adversarial robustness (similar to those from Table 3) regarding *GLC* and *GPC* show that these functions have a positive impact on the robustness of the basic CNN architecture. However, this impact is quite insignificant comparably to the one achieved in the generalized pooling cases. The *GLC* is slightly more robust than *GPC*. The code and details can be found at: https://github.com/Adversarial-Intelligence-Group/flexnets.

Brief summary of the subsection and the section in general: Suggested way to compute and flexibly adapt the convolution operation based on generalized means is capable of discovery of suitable parameters for convolution in CNNs by learning. New convolution operations are able to automatically find the way to behave in different layers of the CNN architecture (i.e., behaving the same way as a standard convolution, or giving values larger or smaller than the standard convolution does). Experiments show that these functions slightly improve the generalization performance of the classical CNNs and they do not have any negative impact on the robustness of the CNN architecture against the adversarial inputs.

5. Flexible/trainable data preprocessing

Preprocessing of input data is an important operation, which aims to improve the performance of deep neural networks in general and CNNs in particular (Raad et al., 2021). Standardization is often used in neural networks to preprocess the available data by putting different variables on the same scale, which allows comparing scores between them. The most commonly used technique is Z-score normalization, which is calculated using the arithmetic mean and standard deviation of the given data. Such standardization centers the feature columns at mean 0 with standard deviation 1 so that the feature columns get the form of a normal distribution, which makes it easier to learn the neural network weights.

A standard Z-score computation schema works as follows. Assume that we have a set of real numbers $x_1, x_2, \ldots, x_n$ as an input data vector $\vec{x}$ ("population"). Normalized values are traditionally computed as follows (z-score normalization of the values from the data vector):

$\tilde{x}_i = ZNORM(x_i) = \frac{x_i - \mu}{\sigma}$, where μ is the mean of the population and σ is the standard deviation of the population, i.e.:

$$\mu = \frac{x_1 + x_2 + \cdots + x_n}{n} = AM\left(x_1, x_2, \ldots, x_n\right) \text{ and}$$
$$\sigma = \sqrt{\frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \cdots + (x_n - \mu)^2}{n}}$$
$$= PM_2\left[(x_1 - \mu), (x_2 - \mu), \ldots, (x_n - \mu)\right],$$

where *AM* — Arithmetic Mean as defined by formula (1) and *PM* — Power Mean as defined by formula (5) with fixed parameter $\delta = 2$.

However, such fixed (inflexible) statistics used in the Z-score normalization technique could be sensitive to outliers in the data, highlighting the need for a more robust and efficient normalization procedure as argued in Jain et al. (2005) or adaptive normalization as suggested by Passalis et al. (2020).

Trainable and, therefore, more flexible normalization could be introduced by using our generalized mean functions *GLM* and *GPM* with trainable parameters instead of fixed statistics. We suggest a generalized Z-score-like normalization function with trainable parameters on the basis of *GLM* as *LNORM* and a similar normalization function on the basis of *GPM* as *PNORM*. We define these functions as follows:

$$LNORM_{\alpha_1,\beta_1,\alpha_2,\beta_2}(x_i) = \frac{x_i - L\mu_{\alpha_1,\beta_1}}{L\sigma_{\alpha_2,\beta_2}},$$

$$\text{where } L\mu_{\alpha_1,\beta_1} = GLM_{\beta_1}^{\alpha_1}\left(\vec{x}\right), \text{ and } L\sigma_{\alpha_2,\beta_2} = GLM_{\beta_2}^{\alpha_2}\left(\overrightarrow{x - L\mu_{\alpha_1,\beta_1}}\right); \tag{18}$$

$$PNORM_{\gamma_1,\delta_1,\gamma_2,\delta_2}(x_i) = \frac{x_i - P\mu_{\gamma_1,\delta_1}}{P\sigma_{\gamma_2,\delta_2}},$$

$$\text{where } P\mu_{\gamma_1,\delta_1} = GPM_{\delta_1}^{\gamma_1}\left(\vec{x}\right), \text{ and } P\sigma_{\gamma_2,\delta_2} = GPM_{\delta_2}^{\gamma_2}\left(\overrightarrow{x - P\mu_{\gamma_1,\delta_1}}\right). \tag{19}$$

As one can see, such normalization includes computing generalized means and generalized standard deviations for the data, which gives traditional values for means and deviations only for specific values of the parameters. Therefore, such functions enable more flexible computing of statistics and finding optimal parameters for it via backpropagation learning, which will fit the best the data normalization needs.

Brief summary of the section: We have noticed that the core approach applied to data standardization (particularly Z-score normalization) and, therefore, preparing the data for the ML processing by neural networks, involves mean functions with a fixed structure. These are: *AM* (formula (1)) used to compute statistical means and *PM* (formula (5)) with fixed parameter $\delta = 2$ to compute the standard deviation. Therefore, we suggested giving more freedom to the Z-score normalization procedure by replacing the fixed means with the generalized means (*GLM* or *GPM*). By this way, the optimal configuration for the Z-score normalization could be learned by backpropagation. We defined Z-score generalized based on *GLM* as *LNORM* (formula (18)); and Z-score generalized based on *GPM* has been defined as *PNORM* (formula (19)). We have not done any experiments specifically for the normalization component, however we include corresponding *LNORM* component to the experiments together with other components (see further sections).

6. Flexible/trainable ReLU and SoftMax

Applying the Rectified Linear Unit (ReLU) as an activation function is known to be a popular supplementary step to the convolution operation in CNNs, which aims to increase the nonlinearity in the processed images.

The traditional ReLU definition is as simple as this: $ReLU\left(x\right) = \max\left(0, x\right)$.

It is known that various sophisticated attempts to generalize the ReLU activation function, such as, e.g., Gao et al. (2020), are related to the "dying ReLU" problem (zero activation for negative arguments). The search for a good universal (Yuen et al., 2021) or trainable (He et al., 2015) activation function instead of a fixed ReLU is still ongoing. He et al. (2015) suggested (and proved effectiveness for image classification) such a simple modification PReLU (parametric ReLU), which has trainable parameter α and looks as follows:

$$PReLU\left(x\right) = \max\left(0, x\right) + \alpha \cdot \min\left(0, x\right).$$

Yuen et al. (2021) suggests using the approximation of the ReLU function with the trainable Softplus function as a universal activation function (UAF). Taking into account that the Softplus

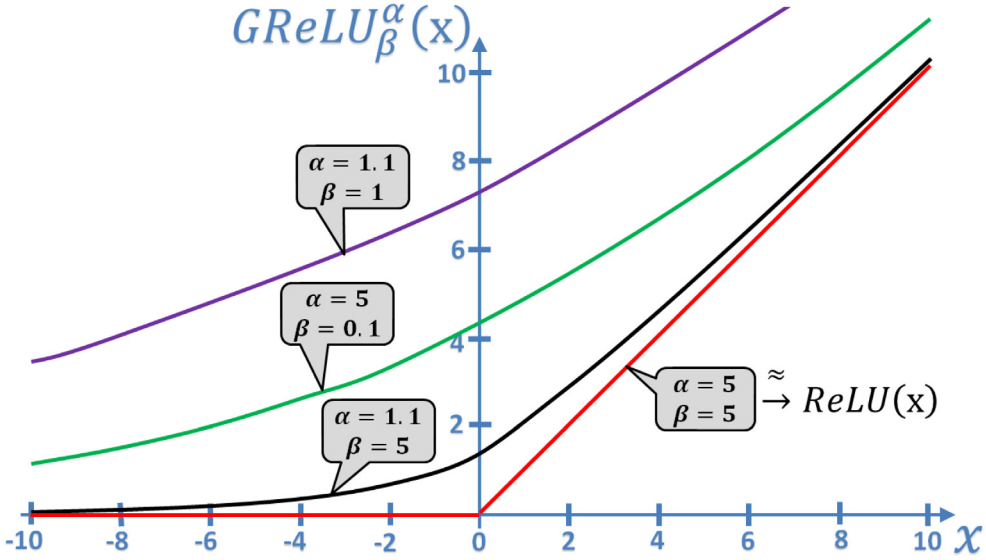

Fig. 8. Examples of the Generalized ReLU function with different parameters.

function is a smooth approximation of ReLU, i.e.:

$$Softplus\ (x) = \ln\left(1 + e^{x}\right), \text{ or in other popular form:}$$

$$SoftPlus_{\beta}\ (x) = \frac{1}{\beta} \cdot \ln\left(1 + e^{\beta \cdot x}\right) \xrightarrow{\beta \gg 1} ReLU(x),$$

… Yuen et al. (2021) define the UAF by adding five controlling and trainable parameters *A*, *B*, *C*, *D* and *E* to the Softplus function as follows:

$$UAF\ (x) = \ln\left(1 + e^{A \cdot (x+B) + C \cdot x^2}\right) - \ln\left(1 + e^{D \cdot (x-B)}\right) + E$$

… aiming to give more degrees of freedom for the UAF. However, unlike regular activation functions, the UAF requires quite complex and unique training procedure to achieve the best performance.

Therefore, as a compromise, we suggest the following generalized ReLU function (*GReLU*) by adding just one extra trainable parameter $\alpha > 1$ in addition to the already existing one $\beta > 0$:

$$GReLU_{\beta}^{\alpha}\ (x) = \frac{1}{\beta} \cdot log_{\alpha}\left(1 + \alpha^{\beta \cdot x}\right) = \frac{ln\left(1 + \alpha^{\beta \cdot x}\right)}{\beta \cdot ln\,\alpha}. \tag{20}$$

Fig. 8 illustrates the plots of the examples of the *GReLU* function (20) with different parameters. One can see that the case when $\alpha = 5$ and $\beta = 5$ results to the function, which is almost equal to the traditional ReLU.

Corresponding partial derivatives of the *GReLU* function regarding each parameter are as follows:

$$\frac{\partial\left[GReLU_{\beta}^{\alpha}\ (x)\right]}{\partial \alpha} = \frac{1}{\alpha \cdot ln\,\alpha} \cdot \left[\frac{x \cdot \alpha^{\beta \cdot x}}{1 + \alpha^{\beta \cdot x}} - \frac{ln\left(1 + \alpha^{\beta \cdot x}\right)}{\beta \cdot ln\,\alpha}\right];$$

$$\frac{\partial\left[GReLU_{\beta}^{\alpha}\ (x)\right]}{\partial \beta} = \frac{1}{\beta} \cdot \left[\frac{x \cdot \alpha^{\beta \cdot x}}{1 + \alpha^{\beta \cdot x}} - \frac{ln\left(1 + \alpha^{\beta \cdot x}\right)}{\beta \cdot ln\,\alpha}\right].$$

SoftMax is another popular activation function, which is traditionally used in the last layer of the neural network to normalize an arbitrary vector of real numbers $\vec{z}$ (logits) towards the probability distribution over *K* predicted output classes as follows:

$$SoftMax(z_i)\ = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}.$$

SoftMax can be used with the trainable parameter for enabling an arbitrary base instead of a fixed "e" and, therefore for optimizing the use of such more flexible SoftMax using backpropagation, like it is done, e.g., in (He et al., 2018):

$$SoftMax_{\alpha}\ (z_i) = \frac{\alpha^{z_i}}{\sum_{j=1}^{K} \alpha^{z_j}}. \tag{21}$$

One popular form for the parametrized SoftMax is the one, which assumes that $\alpha = e^{\frac{1}{T}}$ in (21), where parameter T – "temperature" is introduced as follows:

$$SoftMax_T(z_i) = \frac{e^{\frac{z_i}{T}}}{\sum_{j=1}^{K} e^{\frac{z_j}{T}}}.$$

We, however, believe that SoftMax could be used even more efficiently if we make it more flexible by using the opportunities provided by the *LNORM* (formula (18)) and *PNORM* (formula (19)) normalization prior to applying SoftMax. Therefore, we introduce two new generalized options for the SoftMax functions (*GLSoftMax* and *GPSoftMax*) as follows:

$$GLSoftMax_{\alpha_1,\beta_1,\alpha_2,\beta_2}(z_i) = \frac{e^{LNORM_{\alpha_1,\beta_1,\alpha_2,\beta_2}(z_i)}}{\sum_{j=1}^{K} e^{LNORM_{\alpha_1,\beta_1,\alpha_2,\beta_2}(z_j)}}, \tag{22}$$

where $LNORM_{\alpha_1,\beta_1,\alpha_2,\beta_2}(z_i)$ is computed according to formula (18);

$$GPSoftMax_{\gamma_1,\delta_1,\gamma_2,\delta_2}(z_i) = \frac{e^{PNORM_{\gamma_1,\delta_1,\gamma_2,\delta_2}(z_i)}}{\sum_{j=1}^{K} e^{PNORM_{\gamma_1,\delta_1,\gamma_2,\delta_2}(z_j)}}, \tag{23}$$

where $PNORM_{\gamma_1,\delta_1,\gamma_2,\delta_2}(z_i)$ is computed according to formula (19).

Our experiments with *LNORM*-driven SoftMax (*GLSoftMax*) follow the same schema, the same CIFAR-10 dataset and the same skeleton for the CNN architecture (Fig. 4) as the experiments with the generalized pooling (Section 3.2) and the generalized convolution (Section 4.2). Here we have two competing options: standard SoftMax vs. *GLSoftMax*. The latter generalized option appears to be sensitive to its parameters' initialization (for some initializations, the validation accuracy was higher than that of ordinary SoftMax, while for some slightly lower). This indicates that the optimization process for the *GLSoftMax* parameters during learning is sometimes stacked within local minimums, and, therefore, a more sophisticated training procedure is still needed to get

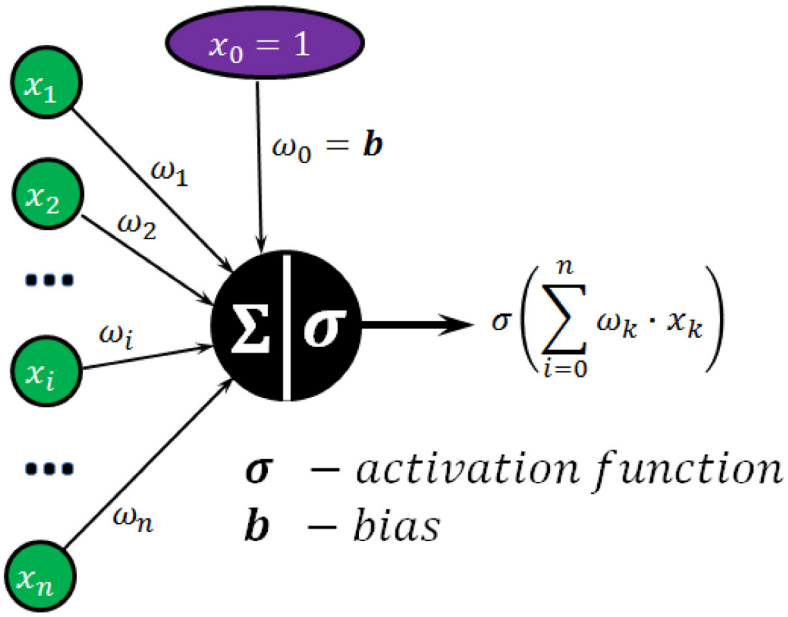

Fig. 9. A traditional neuron of a neural network in general or from a fully connected layer of a CNN in particular.

the full potential and accuracy impact from the *GLSoftMax*. However, as well as the generalized pooling, the generalized SoftMax performed better with the adversarial samples, i.e., demonstrating adversarial robustness. The code and details can be found at: https://github.com/Adversarial-Intelligence-Group/flexnets.

Brief summary of the section: Firstly, we suggest using a couple of free parameters to approximate the standard ReLU function with the Softplus as a basis to get (learn) parameter-driven modifications to it when necessary. Secondly, we suggest generalizing the fixed SoftMax function using the new generalized means and particularly a flexible normalization function driven by them. The values from the basic SoftMax definition (formula (21)) are suggested to be replaced with the flexibly (parameter-driven) normalized (by either *LNORM* or *PNORM*) values. While the traditional SoftMax has a kind of "winner takes all" effect, the generalized version of SoftMax could learn to smooth this effect when appropriate or make it even sharper. Preliminary experiments with the learnable SoftMax functions *GLSoftMax* and *GPSoftMax* show that accuracy improvement could be achieved but the actual impact depends on the choice of initial values for the parameters. Therefore, some special parameter training procedures are still needed to get the full potential from the suggested generalization.

7. Neurons with more flexibility

Within an artificial neural network in general and in a CNN (fully connected layers) in particular, a neuron is a mathematical function that computes the weighted average of its input, which is passed through a (usually nonlinear) activation function as shown in Fig. 9.

The study by Güler (2021) suggests that a formal model of a neuron must not necessarily be as simple as just a linear combination of its inputs, but such a model must be capable of integrating synaptic inputs with a wide spectrum of nonlinearities through collective action of its dendritic branches.

A generalized neuron model, which in a way aggregates the inputs and the activation function, has been suggested and tested by Chaturvedi and Malik (2005). The generalized neuron integrates two artificial neurons: one, which traditionally summarizes weighted inputs and applies sigmoid activation; and another one, which gives weighted product aggregation with Gaussian activation. Due to such sophisticated and integrated activation, such a generalized neuron is more flexible (another degree of flexibility) to suite different problems and it is proven to be able to learn faster with much smaller training data. Experiments performed by Dumitrache and Popescu (2016) show that such generalized neurons could be efficient to tackle complex problems within hybrid models together with genetic algorithms.

Another model of a generalized neuron is suggested by Fil and Chu (2020) as an evolution of the multispike tempotron (a single spiking neuron model that can solve complex supervised classification tasks). The aim was to simplify the tempotron model while retaining its ability to learn. Suggested generalized models were interpreted as a chemical system, i.e., a conceptual bridge between neural network computations and molecular information processing, and it enables alternative training (e.g., error trace learning in addition to error backpropagation). Such generalization show good efficiency in classifying multiple spatiotemporal patterns.

Zhang and Zhou (2021) suggest a novel bio-plausible neuron as a flexible transmitter model, which employs a pair of parameters to model the neurotransmission. Such a model includes the commonly used neuron model as its particular case and it is biologically more realistic and capable of handling complex spatiotemporal data. Their study encourages the feasibility of developing artificial neural networks with enhanced neuronal plasticity.

However, still the majority of neural architectures consider a neuron as a fixed operation of the weighted average regarding the inputs and then apply an activation function to the result of such operation. What if to replace such a fixed operation with a more flexible one like, e.g., driven by *GLM* or *GPM*, so that the structure of the operation will be controllable with the trainable parameters and, in the same time, it will provide an already nonlinear output, which will not require additional activation function? In this way, one may enable the learning influence (weight) of each input independently (as it was always before) and at the same time learn additional parameters, which model nonlinear cumulative effects of the distribution of all the inputs.

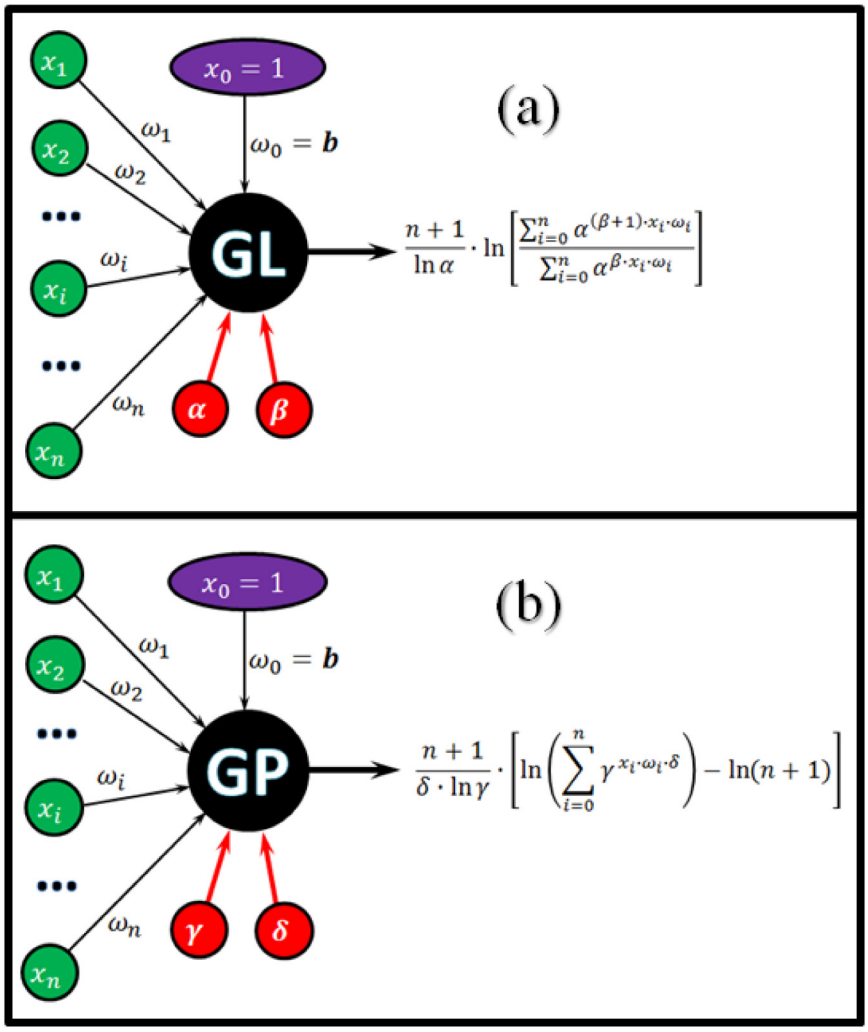

Fig. 10. A basic schema of the generalized neurons with trainable parameters in addition to weights: (a) GL-Neuron; (b) GP-Neuron.

In this paper, we introduce two types of neurons with trainable weights and with two additional trainable parameters (see Fig. 10).

The first neuron type is a Generalized Lehmer Neuron or a GL-Neuron and the second one is a Generalized Power Neuron or a GP-Neuron. These neurons will model the following functions (see Fig. 10(a) and (b)) using the generalized convolution operations as defined by formulas (12) and (13) respectively:

$GLC_{\beta}^{\alpha}(\vec{x},\vec{\omega}) = \frac{n+1}{\ln\alpha} \cdot \ln\left[\frac{\sum_{i=0}^{n} \alpha^{(\beta+1)\cdot x_i\cdot\omega_i}}{\sum_{i=0}^{n} \alpha^{\beta\cdot x_i\cdot\omega_i}}\right]$, similar to formula (12) with taking the bias into account (Fig. 10(a));

$GPC_{\gamma}^{\delta}(\vec{x},\vec{\omega}) = \frac{n+1}{\delta\cdot\ln\gamma} \cdot \left[\ln\left(\sum_{i=0}^{n} \gamma^{x_i\cdot\omega_i\cdot\delta}\right) - \ln(n+1)\right]$, similar to formula (13) with taking the bias into account (Fig. 10(b)).

Partial derivatives of $GLC_{\beta}^{\alpha}(\vec{x},\vec{\omega})$ (or the function, which is provided by the GL-Neuron) regarding parameters α and β are the same as described in formulas (14) and (15), i.e.:

$$\frac{\partial\left[GLC_{\beta}^{\alpha}(\vec{x},\vec{\omega})\right]}{\partial\alpha} = \frac{n+1}{\alpha\cdot\ln\alpha}\cdot\left\{(\beta+1)\cdot\frac{\sum_{i=0}^{n}\left[x_i\cdot\alpha^{(\beta+1)\cdot x_i\cdot\omega_i}\right]}{\sum_{i=0}^{n}\alpha^{(\beta+1)\cdot x_i\cdot\omega_i}} - \beta\cdot\frac{\sum_{i=0}^{n}\left(x_i\cdot\omega_i\cdot\alpha^{\beta\cdot x_i\cdot\omega_i}\right)}{\sum_{i=0}^{n}\alpha^{\beta\cdot x_i\cdot\omega_i}} - GLC_{\beta}^{\alpha}(\vec{x},\vec{\omega})\right\};$$

$$\frac{\partial\left[GLC_{\beta}^{\alpha}(\vec{x},\vec{\omega})\right]}{\partial\beta} = (n+1)\cdot\left\{\frac{\sum_{i=0}^{n}\left[x_i\cdot\omega_i\cdot\alpha^{(\beta+1)\cdot x_i\cdot\omega_i}\right]}{\sum_{i=0}^{n}\alpha^{(\beta+1)\cdot x_i\cdot\omega_i}} - \frac{\sum_{i=0}^{n}\left(x_i\cdot\alpha^{\beta\cdot x_i\cdot\omega_i}\right)}{\sum_{i=0}^{n}\alpha^{\beta\cdot x_i\cdot\omega_i}}\right\}.$$

Partial derivative of $GLC_{\beta}^{\alpha}(\vec{x},\vec{\omega})$ regarding each weight ω_i is as follows:

$$\frac{\partial GLC_{\beta}^{\alpha}(\vec{x},\vec{\omega})}{\partial\omega_i} = (n+1)\cdot x_i\cdot\alpha^{\beta\cdot x_i\cdot\omega_i}\cdot\left[\frac{\alpha^{x_i\cdot\omega_i}}{\sum_{i=0}^{n}\alpha^{(\beta+1)\cdot x_i\cdot\omega_i}} - \frac{1}{\sum_{i=0}^{n}\alpha^{\beta\cdot x_i\cdot\omega_i}}\right]. \quad (24)$$

Partial derivatives of $GPC_{\delta}^{\gamma}(\vec{x},\vec{\omega})$ (or the function, which is provided by the GP-Neuron) regarding parameters γ and δ are the same as described in formulas (16) and (17), i.e.:

$$\frac{\partial\left[GPC_{\delta}^{\gamma}(\vec{x},\vec{\omega})\right]}{\partial\gamma} = \frac{n+1}{\gamma\cdot\ln\gamma}\cdot\left[\frac{\sum_{i=0}^{n}\left(x_i\cdot\omega_i\cdot\gamma^{x_i\cdot\omega_i\cdot\delta}\right)}{\sum_{i=0}^{n}\gamma^{x_i\cdot\omega_i\cdot\delta}} - GPC_{\delta}^{\gamma}(\vec{x},\vec{\omega})\right];$$

$$\frac{\partial\left[GPC_{\delta}^{\gamma}\left(\vec{x},\vec{\omega}\right)\right]}{\partial\delta}=\frac{n+1}{\delta}\cdot\left[\frac{\sum_{i=0}^{n}\left(x_i\cdot\gamma^{x_i\cdot\omega_i\cdot\delta}\right)}{\sum_{i=0}^{n}\gamma^{x_i\cdot\omega_i\cdot\delta}}-GPC_{\delta}^{\gamma}\left(\vec{x},\vec{\omega}\right)\right].$$

Partial derivative of $GPC_{\delta}^{\gamma}\left(\vec{x},\vec{\omega}\right)$ regarding each weight ω_i is as follows:

$$\frac{\partial GPC_{\delta}^{\gamma}\left(\vec{x},\vec{\omega}\right)}{\partial\omega_i}=\frac{(n+1)\cdot x_i\cdot\gamma^{x_i\cdot\omega_i\cdot\delta}}{\sum_{i=0}^{n}\gamma^{x_i\cdot\omega_i\cdot\delta}}. \tag{25}$$

For the generalized neuron implementation in deep neural networks, it would be more practical to assume that all the generalized neurons in each layer share (and learn by backpropagation) the same pair of values for the controlling parameters. It is also possible to apply some activation function (i.e., sigmoid, tanh or even SoftMax) between the layers to keep control over the values.

Our experiments with the GL-Neurons (GP-Neurons have been left for future experiments) we start just by comparing a multilayer perceptron (MLP) with three layers of traditional neurons and ReLU activation vs. three layers of GL-Neurons (GLN) with different activation options applied on top of the popular "Iris" dataset from the UCI ML Repository (https://archive.ics.uci.edu/ml/datasets/iris). As a result: for the MLP/ReLU the test performance has been achieved as of 86.7%; GLN/tanh — 87.5%; GLN/ReLU — 88.0%; GLN/(no activation) — 88.1%; GLN/Sigmoid — 90.0%; GLN/SoftMax — 93.3%.

To check whether GLN is capable of providing an added value if used to replace the fully connected layers in CNNs, we follow the same schema, the same CIFAR-10 dataset and the same skeleton for the CNN architecture (Fig. 4) as the experiments with the generalized pooling (Section 3.2) and the generalized convolution (Section 4.2). In spite of the fact that we do not yet have a special optimized procedure for training generalized models, we got an improvement in performance, i.e., standard MLP/ReLU as a classifier in the CNN architecture on top of CIFAR-10 (Fig. 4) achieved 84.24 accuracy on the test set, while our GLN layer got 84.33 value of accuracy.

Brief summary of the section: We checked whether the new generalization (i.e., our *GLM* and *GPM* formalisms) could be useful also as the new type of neurons for neural networks in general and CNNs in particular. This update makes the new generalized neuron not a simple linear combination of its inputs but enables a wider scope of possible outcomes and nonlinearities depending on the trainable parameters. The formalism behind the generalized neuron could work with or without any additional activation function. Due to the additional parameters, the training procedure must be optimized to get the full added value from such neurons. However, in our experiments (Iris and CIFAR-10 datasets) we managed to get small classification performance improvements comparably to standard architectures even with the standard procedures. Due to the mathematical properties behind the generalized neurons (i.e., certain settings for the parameters make at least a standard neuron and, in addition, many other modifications of it dynamically while learning), there is definitely a potential yet to be discovered.

8. Hyper-flexible CNNs and their training

We have studied additional (mathematical) flexibility of the major components (one-by-one) of a CNN architecture, considering each of them separately as a flexible (with additional trainable parameters) puzzle within traditional CNN. We saw that getting benefits from the additional flexibility demands arranging the training process of CNN accordingly. In this section, we put all the puzzles together to get a hyper-flexible CNN architecture. We understand that, on the one hand, such architecture would have huge potential as a new and flexible deep learning model. However, on the other hand, a lot of additional studies is needed to find an effective and efficient way to train such a network. In this section, we will provide just a few hints (aka heuristics) towards training such CNNs, but the full and hidden potential of such networks is yet to be explored.

Fig. 11 represents an option of the hyper-flexible CNN architecture. It includes: several "Convolution-ReLU-Pooling" triples of the generalized (each with own α and β parameters) components; generalized (four trainable parameters) feature vector normalization component before the fully-connected layers; several fully connected layers (each with own α and β parameters) of generalized neurons; final generalized SoftMax (four trainable parameters) layer; loss/backpropagation loop towards training all the weights and parameters.

One important statement regarding this architecture, which is evident and does not require proof, would be as follows: due to the fact that each additional parameter of any generalized component within the hyper-flexible CNN architecture has a possible initialization value, which makes such component equivalent to the corresponding component within the traditional CNN architecture, the hyper-flexible CNN after training will perform at least as good as its traditional CNN counterpart will.

Various methods, intuitive tricks and heuristics to improve training of deep networks (inspired both by AI mathematics and neuroscience research) have been discovered already, and the majority of innovations in deep learning are expected just in the field of efficient neural network training. A good bridging study was performed by Shen et al. (2021) where various regularization, batch and weight normalization techniques were presented to accelerate training in both artificial and biological deep neural networks.

We want to share one intuition regarding training deep neural networks, in which the trainable parameters of mathematical functions are combined with the trainable weights of the variables at each layer (such as, e.g., a hyper-flexible CNN in Fig. 11).

Consider an abstract example in Fig. 12, which shows a deep neural network, which simulates (to some reasonable extent) the work of a human arm while typing on a computer keyboard. The last layer of the network simulates "fingers", which are supposed to operate in direct contact with the keyboard keys having certain degrees of freedom in their movements; before the "fingers" layer, there is a "hand" layer, which is supposed to move together with "fingers", giving them more opportunities to access the needed keys; … then "forearm" with the same effect towards "hand", then "shoulder" affecting "forearm" freedoms, etc. Therefore, the complete network (an "arm") works as a complex function composed of the nested subfunctions as shown in Fig. 12. Each subfunction (layer) has both weights and parameters. What is the intuitive difference between these two? We believe that "weights" are the task specific variables, which are trained specifically for certain narrow "skill" (e.g., typing on a keyboard) using a particular set of training samples (aka training dataset). If you leave only the weights, then (as we argued before) the complete network (an "arm" in the example) will be highly tuned to the specific skill due to the hidden overfitting (Sa-Couto & Wichert, 2021). The "parameters" (in contrast to the weights) are supposed to be capability-specific variables and supposed to be trained so that corresponding functions could be reused for a larger scope of different tasks (e.g., different than just "typing" arm activity). Therefore, both weights and parameters are important, and we believe that a hyper-flexible CNN is a good example of something more generic than CNN deep architecture.

Another expected benefit from the parameters is the robustness of the network towards input perturbations at the testing

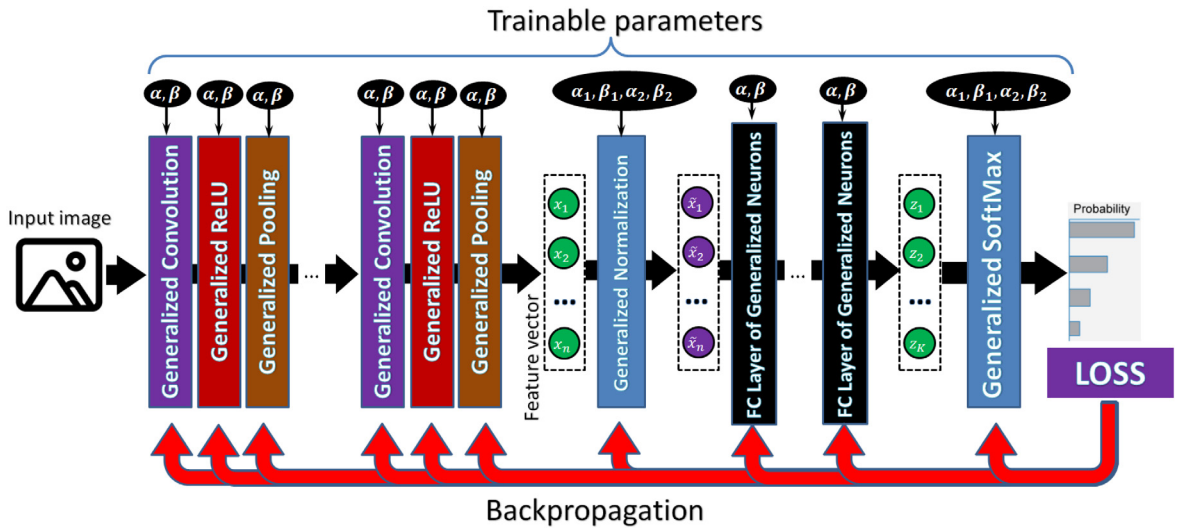

Fig. 11. An abstract architecture of a Hyper-Flexible CNN with trainable functional components (in addition to the traditional weights), which are based on either *GLM* or *GPM* functions. To keep the consistent designation, for the case of *GPM*, α and β parameters in the figure must be replaced with γ and δ.

phase. Training only the weights usually makes the network highly tuned to a particular task (typing over a particular keyboard in our example). Therefore, slightly modified (adversarial) inputs (aka evasion attacks) or noisy inputs or just a slightly modified keyboard from our example in Fig. 12 may confuse the network and reduce classification accuracy dramatically. However, the parameters (being less task dependent) are expected to improve the robustness and, therefore, the generalization performance of the network in such cases.

Our intuition regarding the abstract example network from Fig. 12 is that the network parameters would be better to train consequently from the end layer towards the beginning layer. With fixed ("frozen") initial parameters of the "arm" in our example (Fig. 12), only the "fingers" (i.e., corresponding parameters) are being trained to get maximal possible performance; then the "hand" is "defrosting" and included in the training process; then "forearm"; then "shoulder", etc.

One "soft" way to implement the above intuition would be layer-specific learning rates for the backpropagation training process. Singh et al. (2015) have applied layer-specific learning rates due to the "vanishing gradients" problem. The gradients which drive back-propagation could be extremely large for weights connecting deeper layers (near the output layer), and these gradients could be extremely small for shallow layers (near the input layer), which causes slow training of the network. Therefore, adaptive learning rates have been suggested to speed up learning in the shallow layers.

Layer-specific learning rates could be applied in many different ways. For example, as follows: $\eta_i = \frac{\eta_0}{\varphi^{N-i}}$, where η_0 is a setup for a maximal learning rate of the network (which could be the subject of adaptation/ change based on different techniques); η_i is a learning rate computed for the ith layer of the network; N is the total number of layers; $\varphi = \frac{\sqrt{5}+1}{2} \approx 1.618$ is the "Golden Ratio" constant. In this way, at each iteration of a possible η_0 update, e.g., according to the Adagrad (Duchi et al., 2011) or Adadelta (Zeiler, 2012) methods, the network parameters, which are closer to the final layer, will be trained faster as it is shown in the example in Fig. 13(a).

Similar thinking could be applied when defining and controlling the minibatch size (layer-wise) during the training process. Minibatch gradient descent is an evolution of a stochastic gradient descent used for large-scale optimization problems in ML (Li et al., 2014). Within our study, we suggest applying the layer-specific minibatch sizes as follows: $m_i = m_0 \cdot 2^{N-i}$, where m_0 is a setup for a minibatch size of the network (which could be a subject of adaptation/ change based on different techniques); m_i is a minibatch size computed for the ith layer of the network; N is total number of layers. In this way the parameters from the end of the network architecture will be updated more often during the backpropagation (minibatch gradient descent optimization) than the parameters from the beginning layers as is shown in the example in Fig. 13(b).

There is one place within the minibatch version of the stochastic optimization process where we can efficiently apply our generalized Lehmer and Power mean functions. This is related to the parameters update according to the computed loss function at each iteration of the training process. It is already shown (Knoblauch, 2021) that a more general error initialization method, which uses power functions as a new family of loss functions with parameters, can significantly improve the speed and convergence of backpropagation learning. Now we also want to modify the formalisms of updating the parameters, which are trained during gradient-descent driven backpropagation.

The traditional way to update parameters during a gradient descent is as follows:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{1}{m} \cdot \sum_{i=1}^{m} \nabla_\theta (LOSS(x^{(i)}, y^{(i)}, \hat{y}^{(i)}, \theta_t)), \text{ or:}$$

$$\theta_{t+1} = \theta_t - \eta \cdot AM \left\{ \nabla_\theta \left[LOSS(x^{(1)}, y^{(1)}, \hat{y}^{(1)}, \theta_t) \right], \ldots, \nabla_\theta \left[LOSS(x^{(m)}, y^{(m)}, \hat{y}^{(m)}, \theta_t) \right] \right\},$$

where θ is a parameter to be updated at current $(t+1)$th iteration; m is a minibatch size; η is a learning rate; $LOSS$ is a loss function; $\langle x^{(i)}, y^{(i)} \rangle$ is the ith data sample from the training set with input $x^{(i)}$ and true output $y^{(i)}$; $\hat{y}^{(i)}$ is an output produced by neural network for the input $x^{(i)}$ at tth iteration; ∇_θ is a component of a gradient vector, i.e., a partial derivative of the loss function regarding parameter θ; AM — Arithmetic Mean.

One can see that here might be yet another place where a more flexible mean function could be used, i.e., instead of AM, one can apply GLM with trainable parameters α and β or GPM with parameters γ and δ to control the gradient descent process:

$$\theta_{t+1} = \theta_t - \eta \cdot GLM_\beta^\alpha \left\{ \nabla_\theta \left[LOSS(x^{(1)}, y^{(1)}, \hat{y}^{(1)}, \theta_t) \right], \ldots, \nabla_\theta \left[LOSS(x^{(m)}, y^{(m)}, \hat{y}^{(m)}, \theta_t) \right] \right\} \quad \text{or}$$

$$\theta_{t+1} = \theta_t - \eta \cdot GPM_\delta^\gamma \left\{ \nabla_\theta \left[LOSS(x^{(1)}, y^{(1)}, \hat{y}^{(1)}, \theta_t) \right], \ldots, \right.$$

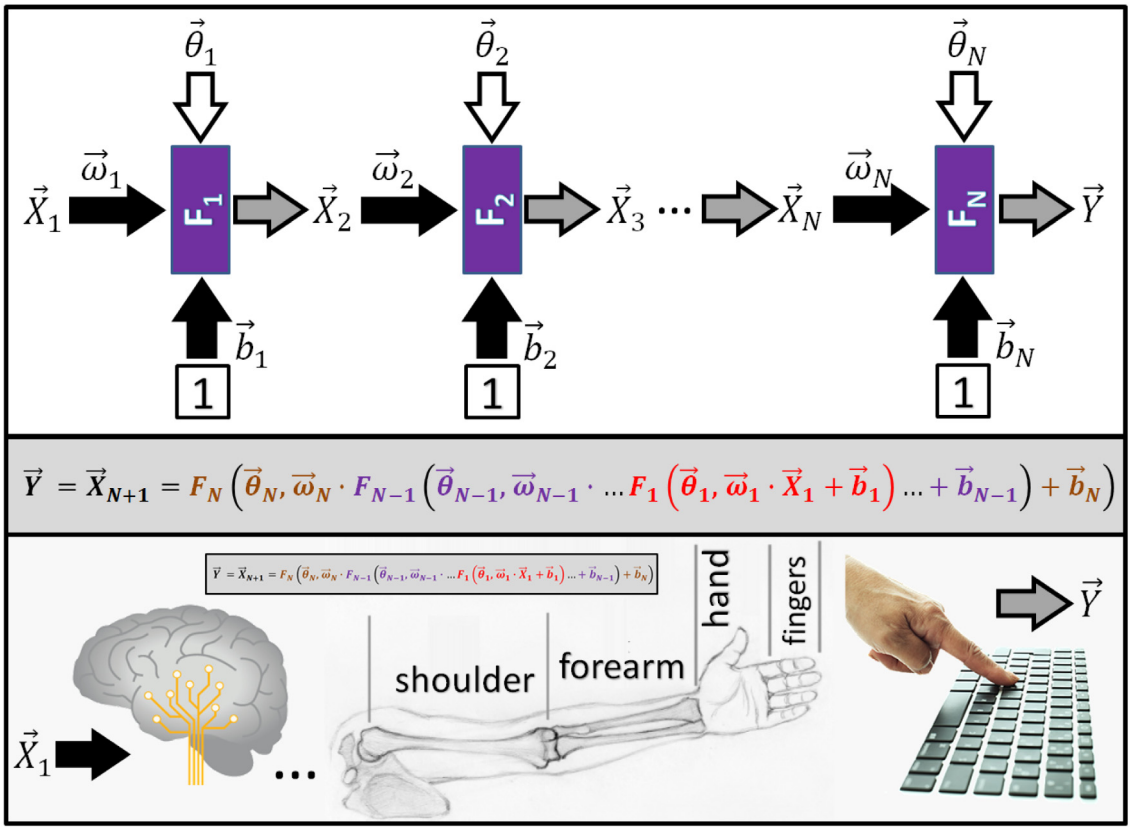

Fig. 12. An abstract analogy of a deep neural network and a human arm (as a composition of nested functions). Here each subfunction has trainable weights and parameters. We suppose that weights are responsible to tune the overall function to a specific task (e.g., typing on a keyboard) and the parameters define a more generic capability of the overall function, which goes beyond the specific task and can be reused within a broader scope of potential tasks.

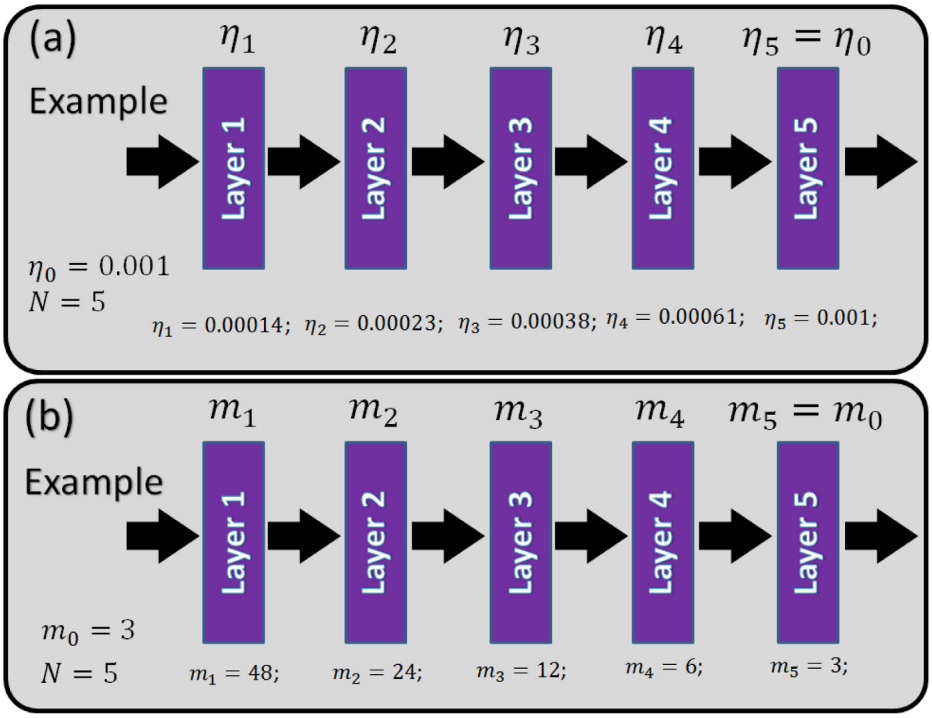

Fig. 13. Example of a control over the training process of a deep neural network parameters using the intuition ("the deeper layer – the faster training"): (a) layer-specific learning rates; (b) layer-specific minibatch sizes.

Table 6
The values for the test accuracy achieved by 6 different configurations of Hyper-Flexible CNN architecture compared with the standard CNN configuration on the CIFAR-10 dataset.

Model (with skeleton from Fig. 4)	Test Accuracy (%)
Standard components in all the three blocks	84.24
GLC (all blocks) + GLP (all blocks)	84.29
GLC (last block) + GLP (last block)	84.39
GLC (last block) + GLP (last block) + GLN	84.41
GPC (last block) + GLP (last block) + GLN	84.45
GLC (last block) + GLP (all blocks) + GLN	84.85
GPC (last block) + GLP (all blocks) + GLN	84.80

$$\nabla_\theta \left[LOSS(x^{(m)}, y^{(m)}, \hat{y}^{(m)}, \theta_t) \right] \Big\} .$$

As one of future objectives, we plan to check and also apply the promising extension of the stochastic gradient descent towards better generalization by trying the reinforced stochastic gradient descent suggested by Huang and Toyoizumi (2017). This method uses the accumulated history of the gradient evolution during training and could significantly accelerate the learning process and its generalization capacity, especially in the case of hyper-flexible CNN learning with multiple higher-order parameters.

To evaluate the impact of the generalized components (studied in the previous sections) together within one Hyper-Flexible CNN architecture, we follow the same schema, the same CIFAR-10 dataset and the same skeleton for the CNN architecture (Fig. 4) as we used in previous sections. As particular cases (configurations) of the generic Hyper-Flexible CNN architecture (Fig. 11), we took the following ones for our experiments (see Table 6): (1) GLC (all blocks) + GLP (all blocks), i.e., the Generalized Lehmer Convolution and the Generalized Lehmer Pooling were applied together in all the blocks; (2) GLC (last block) + GLP (last block), i.e., the same as the previous one but applied in the last (third) block only; (3) GLC (last block) + GLP (last block) + GLN, i.e., the same as the previous one but, in addition, includes generalized Lehmer neurons as a classifier; etc. (altogether 6 flexible configurations vs. the basic CNN architecture). The results are collected in Table 6.

One may see that the hyper-flexibility gives certain improvements in accuracy, yet it adds more demands on the training procedure due to the big number of the parameters (additional to the weights) being trained together (i.e., stacking within local minimums may prevent finding the best parameters for each of the generalized components). Therefore, optimizing the training procedure specifically for the Hyper-Flexible CNNs is yet to be done and is the subject of our future studies.

Brief summary of the section: Hyper-Flexible CNNs composed of generalized components may replace the standard CNN architectures if and when an efficient training procedure is developed specifically for them. However, even a simple heuristics during the training process may give an accuracy improvement due to hyper-flexibility, as one can see from our preliminary experiments. The hidden potential of the Hyper-Flexible CNNs is much higher, which directly follows from the analytical properties of the flexible components (i.e., each of them may perform at least as effective as a standard one for particular parameter settings). When all the components find optimal values for their parameters, the positive impact on accuracy could be dramatic. Some preliminary components (thoughts, heuristics, and analytics) for the future optimization/training procedure have been briefly discussed.

9. Discussion

The slightly better classification performance and, in several cases, essentially better adversarial robustness (confirmed by the experiments) of the generalized and flexible CNN components comparably to the ones based on fixed analytics are not surprising. This happened due to more opportunities in backpropagation learning, i.e., in automated searching for the most appropriate options for the backbone functions (driven by special parameters) in addition to smoothening the effect of these functions further by adjusting weights. In our initial experiments, we used the well-known CIFAR-10 dataset, which is actually a relatively easy target for CNN. Therefore, one may not expect a dramatic improvement of the performance on such datasets while updating the basic CNN architecture with more flexible components. One of our initial intuitions was a concern that more flexible networks are needed to address the weaknesses of traditional CNN architectures. The same data can be labeled differently for different automated classification (on the basis of supervised ML) tasks. The reason (expert's mind) behind each particular labeling remains hidden from a neural network and supposed to be approximated (not necessarily completely discovered) during the network training process. This hidden labeling "rule" for some image datasets could be so sophisticated that CNN may fail to discover or approximate it appropriately. Consider the example in Fig. 14. The well-known MNIST dataset of handwritten digits is often used to compare the performance of various CNN architectures; and this dataset is used with quite a straightforward way of labeling images, i.e., 10 classes, which correspond to "0", "1", …, "9" class labels. CNNs relatively easily discover the features from input images according to what can be seen in the picture. However, one may apply another labeling logic, like the one shown in Fig. 14, where an expert labeled the image set not only according to "what can be seen in the picture" but included into consideration also "what cannot be seen in the picture". One may see one possible way to label images into eight classes as shown in Fig. 14. For example, Class_2 is defined as "all the images which do not have any (close to) vertical lines AND do not have any (close to) horizontal lines AND have at least one (close to) diagonal line". For a supervised ML problem defined in this way, an ordinary CNN architecture with traditional MaxPooling and convolutional layers (which performs quite OK for the ordinary labeling with 10 classes) may not be able to learn to address satisfactorily the classification into the eight new classes' problem. However, just here we may expect that (if hyper-flexible CNN is applied) there will be discovered such parameters for generalized pooling and convolution operations (among others) during training, which will adapt to such exotic labeling and perform classification accurately.

Brief summary of the section: Experiments on the popular image dataset within this study show that the generalized components themselves or applied together in Hyper-Flexible CNNs could have a certain positive impact regarding the classification accuracy and robustness. The main (intuitive for the discussion) message of this section, however, is that the Hyper-Flexible CNNs may be more useful if applied for more complex image classification cases and corresponding datasets where the logic behind the class labeling is well hidden.

10. Conclusions

In this paper, we attempted to mathematically justify the potential use of more flexible CNNs. The flexibility is provided due to generalized mathematical operations used throughout all the architecture (structure, components, hyperparameters, etc.) of a CNN. We analyzed two options of a mathematical "backbone" function for such generalization: *GLM* with trainable parameters α and β or *GPM* with parameters γ and δ. We noticed that the mathematical properties of these functions and their

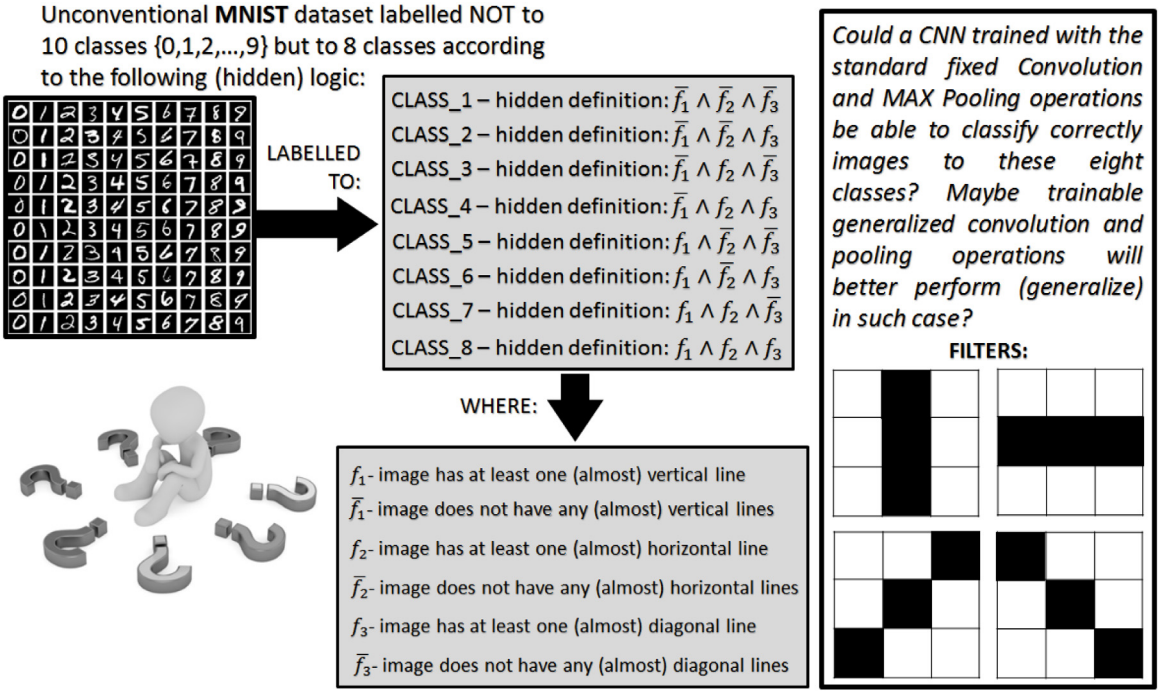

Fig. 14. This example shows that actual and hidden logic of an expert (supervisor) who is labeling the image dataset (before applying CNN to train a classifier) may be relatively complex. It is shown that traditional MNIST with 10 classes of handwritten digits could be relabeled to eight classes, which (potentially) may create difficulties for supervised learning with the traditional CNN architecture and may require more flexible architectures with generalized pooling/convolution layers.

derivatives guarantee better performance of CNN if and when the optimal parameters have been discovered everywhere within the architecture during backpropagation learning.

The main challenge of applying such hyper-flexible CNNs remains the training process. We suggested several heuristics to facilitate learning and improve its impact. However, more studies would be needed to find out solid training techniques for hyper-flexible CNNs capable of capturing all the hidden potential of these architectures.

Our preliminary (component-by-component) experiments demonstrate clear advantages of the hyper-flexible CNN architecture comparably to the traditional ones. However, we expect to discover much more potential of new architecture during our future work on improving the training process. It is too early to make a final judgment regarding the choice between "*GLM* vs. *GPM*" for the hyper-flexible CNN architecture. However, in the majority of our initial experiments, *GLM* performed better than *GPM*. Essential experimental work on different domains and datasets is still needed to more or less precisely evaluate the impact of the suggested hyper-flexibility.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Code is available via https://github.com/Adversarial-Intelligence-Group/flexnets.

References

Amari, S. I. (2007). Integration of stochastic models by minimizing α-divergence. *Neural Computation*, *19*(10), 2780–2796. http://dx.doi.org/10.1162/neco.2007.19.10.2780.

Azulay, A., & Weiss, Y. (2019). Why do deep convolutional networks generalize so poorly to small image transformations? *Journal of Machine Learning Research*, *20*(184), 1–25, https://jmlr.org/papers/volume20/19-519/19-519.pdf.

Babenko, A., & Lempitsky, V. (2015). Aggregating deep convolutional features for image retrieval. arXiv preprint arXiv:1510.07493.

Beliakov, G., & Špirková, J. (2016). Weak monotonicity of lehmer and gini means. *Fuzzy Sets and Systems*, *299*, 26–40. http://dx.doi.org/10.1016/j.fss.2015.11.006.

Berman, M., Jégou, H., Vedaldi, A., Kokkinos, I., & Douze, M. (2019). Multigrain: a unified image embedding for classes and instances. arXiv preprint arXiv:1902.05509.

Bi, Q., Qin, K., Zhang, H., Xie, J., Li, Z., & Xu, K. (2019). APDC-Net: Attention pooling-based convolutional network for aerial scene classification. *IEEE Geoscience and Remote Sensing Letters*, *17*(9), 1603–1607. http://dx.doi.org/10.1109/LGRS.2019.2949930.

Bi, Q., Zhang, H., & Qin, K. (2021). Multi-scale stacking attention pooling for remote sensing scene classification. *Neurocomputing*, *436*, 147–161. http://dx.doi.org/10.1016/j.neucom.2021.01.038.

Bieder, F., Sandkühler, R., & Cattin, P. C. (2021). Comparison of methods generalizing max- and average-pooling. arXiv preprint arXiv:2103.01746.

Brownlee, J. (2019). *Deep learning for computer vision: Image classification, object detection, and face recognition in Python*. Machine Learning Mastery.

Bullen, P. S. (2013). *Handbook of means and their inequalities, Vol. 560* (pp. 175–265). Springer Science & Business Media, http://dx.doi.org/10.1007/978-94-017-0399-4, Chapter III.

Chakrabarty, D. (2018). fH-Mean: One generalized definition of average. *Journal of Environmental Science, Computer Science and Engineering & Technology*, *7*(4), 301–314. http://dx.doi.org/10.24214/jecet.C.7.4.30114.

Chaturvedi, D. K., & Malik, O. P. (2005). Generalized neuron-based PSS and adaptive PSS. *Control Engineering Practice*, *13*(12), 1507–1514. http://dx.doi.org/10.1016/j.conengprac.2005.03.013.

Chen, J., Hu, H., Wu, H., Jiang, Y., & Wang, C. (2021). Learning the best pooling strategy for visual semantic embedding. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 15789–15798).

Chen, Y. P., & Tsai, C. W. (2021). An effective neural architecture optimization algorithm for CNN based on search economics. In *Proceedings of the 2021 ACM international conference on intelligent computing and its emerging applications* (pp. 40–45). http://dx.doi.org/10.1145/3491396.3506505.

Deniz, O., Pedraza, A., Vallez, N., Salido, J., & Bueno, G. (2020). Robustness to adversarial examples can be improved with overfitting. *International Journal of Machine Learning and Cybernetics*, *11*(4), 935–944. http://dx.doi.org/10.1007/s13042-020-01097-4.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(7), 2121–2159.

Dumitrache, I., & Popescu, D. C. (2016). Evolutionary networks with generalized neurons applied in battery SoC estimation. *Romanian Journal of Information Science and Technology*, *19*(1–2), 98–115, https://www.romjist.ro/content/pdf/07-idumitrache.pdf.

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, *20*(55), 1997–2017, https://jmlr.org/papers/v20/18-598.html.

Estrach, J. B., Szlam, A., & LeCun, Y. (2014). Signal recovery from pooling representations. *Proceedings of Machine Learning Research*, *32*(2), 307–315, http://proceedings.mlr.press/v32/estrach14.pdf.

Fil, J., & Chu, D. (2020). Minimal spiking neuron for solving multilabel classification tasks. *Neural Computation*, *32*(7), 1408–1429. http://dx.doi.org/10.1162/neco_a_01290.

Filom, K., Farhoodi, R., & Kording, K. P. (2021). On PDE characterization of smooth hierarchical functions computed by neural networks. *Neural Computation*, *33*(12), 3204–3263. http://dx.doi.org/10.1162/neco_a_01441.

Fukushima, K. (1979). Neural network model for a mechanism of pattern recognition unaffected by shift in position - neocognitron. *IEICE Technical Report, A*, *J62*(10), 658–665, https://ci.nii.ac.jp/naid/10026558204/en/.

Gao, H., Cai, L., & Ji, S. (2020). Adaptive convolutional ReLUs. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 34* (pp. 3914–3921). http://dx.doi.org/10.1609/aaai.v34i04.5805, (04).

Ghiasi-Shirazi, K. (2019). Generalizing the convolution operator in convolutional neural networks. *Neural Processing Letters*, *50*(3), 2627–2646. http://dx.doi.org/10.1007/s11063-019-10043-7.

Gulcehre, C., Cho, K., Pascanu, R., & Bengio, Y. (2014). Learned-norm pooling for deep feedforward and recurrent neural networks. In *Lecture notes in computer science: vol. 8724, Proceedings of the joint European conference on machine learning and knowledge discovery in databases* (pp. 530–546). Springer, http://dx.doi.org/10.1007/978-3-662-44848-9_34.

Güler, M. (2021). Multibranch formal neuron: An internally nonlinear learning unit. *Neural Computation*, *33*(10), 2736–2761. http://dx.doi.org/10.1162/neco_a_01428.

Hanselmann, H., & Ney, H. (2020). Elope: Fine-grained visual classification with efficient localization, pooling and embedding. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision* (pp. 1247–1256).

He, Z., & Toyoizumi, T. (2021). An information-theoretic progressive framework for interpretation. arXiv preprint arXiv:2101.02879.

He, Y. L., Zhang, X. L., Ao, W., & Huang, J. Z. (2018). Determining the optimal temperature parameter for softmax function in reinforcement learning. *Applied Soft Computing*, *70*, 80–85. http://dx.doi.org/10.1016/j.asoc.2018.05.012.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026–1034). http://dx.doi.org/10.1109/ICCV.2015.123.

He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, *212*, Article 106622. http://dx.doi.org/10.1016/j.knosys.2020.106622.

Hu, Z., Bao, W., Qu, K., & Liang, H. (2022). Image-based neural architecture automatic search method for hyperspectral image classification. *Journal of Applied Remote Sensing*, *16*(1), Article 016501. http://dx.doi.org/10.1117/1.JRS.16.016501.

Huang, H., & Toyoizumi, T. (2017). Reinforced stochastic gradient descent for deep neural network learning. arXiv preprint arXiv:1701.07974.

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, *148*(3), 574–591. http://dx.doi.org/10.1113/jphysiol.1959.sp006308.

Isensee, F., Jaeger, P. F., Kohl, S. A., Petersen, J., & Maier-Hein, K. H. (2021). nnU-Net: A self-configuring method for deep learning-based biomedical image segmentation. *Nature Methods*, *18*(2), 203–211. http://dx.doi.org/10.1038/s41592-020-01008-z.

Jain, A., Nandakumar, K., & Ross, A. (2005). Score normalization in multimodal biometric systems. *Pattern Recognition*, *38*(12), 2270–2285. http://dx.doi.org/10.1016/j.patcog.2005.01.012.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. arXiv preprint arXiv:1404.2188.

Knoblauch, A. (2021). Power function error initialization can improve convergence of backpropagation learning in neural networks for classification. *Neural Computation*, *33*(8), 2193–2225. http://dx.doi.org/10.1162/neco_a_01407.

Kolmogorov, A. (1930). On the notion of mean. In *Mathematics and mechanics, 1991, Vol. 1* (pp. 144–146). Kluwer.

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images*: *Technical Report*, Department of Computer Science, University of Toronto, http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, & et al. (Eds.), *Advances in neural information processing systems, 25*. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. http://dx.doi.org/10.1109/5.726791.

Lee, C. Y., Gallagher, P., & Tu, Z. (2017). Generalizing pooling functions in CNNs: Mixed, gated, and tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(4), 863–875. http://dx.doi.org/10.1109/TPAMI.2017.2703082.

Li, M., Zhang, T., Chen, Y., & Smola, A. J. (2014). Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 661–670). http://dx.doi.org/10.1145/2623330.2623612.

Macpherson, T., Churchland, A., Sejnowski, T., DiCarlo, J., Kamitani, Y., Takahashi, H., & Hikida, T. (2021). Natural and artificial intelligence: A brief introduction to the interplay between AI and neuroscience research. *Neural Networks*, *144*, 603–613. http://dx.doi.org/10.1016/j.neunet.2021.09.018.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2019). Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083v4.

Park, J., Kim, J. Y., Huh, J. H., Lee, H. S., Jung, S. H., & Sim, C. B. (2021). A novel on conditional min pooling and restructured convolutional neural network. *Electronics*, *10*(19), 2407. http://dx.doi.org/10.3390/electronics10192407.

Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2020). Adaptive normalization for forecasting limit order book data using convolutional neural networks. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing* (pp. 1713–1717). IEEE, http://dx.doi.org/10.1109/ICASSP40776.2020.9054321.

Pedraza, A., Deniz, O., & Bueno, G. (2021). On the relationship between generalization and robustness to adversarial examples. *Symmetry*, *13*(5), 817. http://dx.doi.org/10.3390/sym13050817.

Raad, K. B., Garderen, K. A., Smits, M., Voort, S. R., Incekara, F., Oei, E. H. G., Hirvasniemi, J., Klein, S., & Starmans, M. P. A. (2021). The effect of preprocessing on convolutional neural networks for medical image segmentation. In *Proceedings of the 18th IEEE international symposium on biomedical imaging* (pp. 655–658). IEEE, http://dx.doi.org/10.1109/ISBI48211.2021.9433952.

Radenović, F., Tolias, G., & Chum, O. (2018). Fine-tuning CNN image retrieval with no human annotation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *41*(7), 1655–1668. http://dx.doi.org/10.1109/TPAMI.2018.2846566.

Roig, C., Varas, D., Masuda, I., Riveiro, J. C., & Bou-Balust, E. (2021). Generalized local attention pooling for deep metric learning. In *Proceedings of the 25th international conference on pattern recognition* (pp. 9951–9958). IEEE, http://dx.doi.org/10.1109/ICPR48806.2021.9412479.

Sa-Couto, L., & Wichert, A. (2021). Simple convolutional-based models: Are they learning the task or the data? *Neural Computation*, *33*(12), 3334–3350. http://dx.doi.org/10.1162/neco_a_01446.

Sejnowski, T. (2018). *The deep learning revolution*. MIT Press.

Sejnowski, T. (2020). The unreasonable effectiveness of deep learning in artificial intelligence. *Proceedings of the National Academy of Sciences of the United States of America*, *117*(48), 30033–30038. http://dx.doi.org/10.1073/pnas.1907373117.

Sermanet, P., Chintala, S., & LeCun, Y. (2012). Convolutional neural networks applied to house numbers digit classification. In *Proceedings of the 21st international conference on pattern recognition* (pp. 3288–3291). IEEE.

Shen, Y., Wang, J., & Navlakha, S. (2021). A correspondence between normalization strategies in artificial and biological neural networks. *Neural Computation*, *33*(12), 3179–3203. http://dx.doi.org/10.1162/neco_a_01439.

Singh, B., De, S., Zhang, Y., Goldstein, T., & Taylor, G. (2015). Layer-specific adaptive learning rates for deep networks. In *Proceedings of the 14th IEEE international conference on machine learning and applications* (pp. 364–368). IEEE, http://dx.doi.org/10.1109/ICMLA.2015.113.

Socher, R., Huang, E., Pennin, J., Manning, C. D., & Ng, A. (2011). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In J. Shawe-Taylor, & et al. (Eds.), *Advances in neural information processing systems, 24*. https://proceedings.neurips.cc/paper/2011/file/3335881e06d4d23091389226225e17c7-Paper.pdf.

Stutz, D., Hein, M., & Schiele, B. (2019). Disentangling adversarial robustness and generalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 6976–6987). https://openaccess.thecvf.com/content_CVPR_2019/papers/Stutz_Disentangling_Adversarial_Robustness_and_Generalization_CVPR_2019_paper.pdf.

Sun, M., Song, Z., Jiang, X., Pan, J., & Pang, Y. (2017). Learning pooling for convolutional neural network. *Neurocomputing*, *224*, 96–104. http://dx.doi.org/10.1016/j.neucom.2016.10.049.

Terziyan, V. (2017). Social distance metric: From coordinates to neighborhoods. *International Journal of Geographical Information Science*, *31*(12), 2401–2426. http://dx.doi.org/10.1080/13658816.2017.1367796, Taylor & Francis.

Tolias, G., Sicre, R., & Jégou, H. (2016). Particular object retrieval with integral max-pooling of CNN activations. In *Proceedings of the international conference on learning representations* (pp. 1–12). ffhal-01842218. https://hal.inria.fr/hal-01842218/document.

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., & Madry, A. (2018). Robustness may be at odds with accuracy. arXiv preprint arXiv:1805.12152.

Wang, P., & Wang, D. (2018). Filter-and-convolve: A CNN based multichannel complex concatenation acoustic model. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing* (pp. 5564–5568). IEEE, http://dx.doi.org/10.1109/ICASSP.2018.8462395.

Wong, E. (2018). Self configuration in machine learning. arXiv preprint arXiv:1809.06463.

Yu, D., Wang, H., Chen, P., & Wei, Z. (2014). Mixed pooling for convolutional neural networks. In D. Miao, W. Pedrycz, D. Ślęzak, G. Peters, Q. Hu, & R. Wang (Eds.), *Lecture notes in computer science*: *vol. 8818*, *Rough sets and knowledge technology* (pp. 364–375). Cham: Springer, http://dx.doi.org/10.1007/978-3-319-11740-9_34.

Yuen, B., Hoang, M. T., Dong, X., & Lu, T. (2021). Universal activation function for machine learning. *Scientific Reports*, *11*(1), 18757. http://dx.doi.org/10.1038/s41598-021-96723-8.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

Zhang, Z., Liu, S., Zhang, Y., & Chen, W. (2022). RS-DARTS: A convolutional neural architecture search for remote sensing image scene classification. *Remote Sensing*, *14*(1), 141. http://dx.doi.org/10.3390/rs14010141.

Zhang, T., Weng, H., Yi, K., & Chen, C. L. (2022). OneDConv: Generalized convolution for transform-invariant representation. arXiv preprint arXiv:2201.05781.

Zhang, S. Q., & Zhou, Z. H. (2021). Flexible transmitter network. *Neural Computation*, *33*(11), 2951–2970. http://dx.doi.org/10.1162/neco_a_01431.