Neural Networks 152 (2022) 380–393

Contents lists available at ScienceDirect

Neural Networks

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

Replacing pooling functions in Convolutional Neural Networks by linear combinations of increasing functions

Iosu Rodriguez-Martinez [a], Julio Lafuente [a], Regivan H.N. Santiago [b], Graçaliz Pereira Dimuro [a,e], Francisco Herrera [c,d], Humberto Bustince [a,*]

[a] Department of Statistics, Computer Science and Mathematics, Public University of Navarre, Pamplona, 31006, Navarre, Spain
[b] Department of Computer Science and applied Mathematics, Universidade Federal do Rio Grande, 1524, Rio Grande, Brasil
[c] Andalusian Research Institute in Data Science and Computational Intelligence, University of Granada, Granada, 18071, Granada, Spain
[d] Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, 21589, Saudi Arabia
[e] Centro de Ciências Computacionais, Universidade Federal do Rio Grande, 96201-900, Rio Grande, Brasil

ARTICLE INFO

Article history:
Received 16 September 2021
Received in revised form 7 March 2022
Accepted 27 April 2022
Available online 6 May 2022

Keywords:
Convolutional Neural Networks
Pooling function
Order statistic
Generalized Sugeno integral

ABSTRACT

Traditionally, Convolutional Neural Networks make use of the maximum or arithmetic mean in order to reduce the features extracted by convolutional layers in a downsampling process known as pooling. However, there is no strong argument to settle upon one of the two functions and, in practice, this selection turns to be problem dependent. Further, both of these options ignore possible dependencies among the data. We believe that a combination of both of these functions, as well as of additional ones which may retain different information, can benefit the feature extraction process. In this work, we replace traditional pooling by several alternative functions. In particular, we consider linear combinations of order statistics and generalizations of the Sugeno integral, extending the latter's domain to the whole real line and setting the theoretical base for their application. We present an alternative pooling layer based on this strategy which we name "CombPool" layer. We replace the pooling layers of three different architectures of increasing complexity by CombPool layers, and empirically prove over multiple datasets that linear combinations outperform traditional pooling functions in most cases. Further, combinations with either the Sugeno integral or one of its generalizations usually yield the best results, proving a strong candidate to apply in most architectures.

1. Introduction

Since the breakthrough of Krizhevsky, Sutskever, and Hinton (2012) on the Imagenet competition, Convolutional Neural Networks, or CNNs (Fukushima & Miyake, 1982; LeCun, Bengio, & Hinton, 2015), have set the state-of-the-art for image processing tasks. In this context, extensive research has been dedicated to developing new CNN designs: more heavily parameterized models have been proved to produce more robust and optimal architectures, offering impressive results for image classification (He, Zhang, Ren, & Sun, 2016; Liu & Deng, 2015); production environment constraints (e.g. smartphones or autonomous vehicles) have led to the development of more "compressed" but still competitive architectures (Howard et al., 2017; Huang, Liu, Van Der Maaten, & Weinberger, 2017; Tan & Le, 2019). Still, most of these strategies keep operating according to the same basic operations already presented in Fukushima and Miyake (1982): convolution, which extracts local features of a given image; and pooling, which downsamples those extracted features sequentially.

Although in the case of the pooling process there have been some new proposals (Forcén, Pagola, Barrenechea, & Bustince, 2020; Graham, 2014; He, Zhang, Ren, & Sun, 2015; Zeiler & Fergus, 2013), most state-of-the-art models default to traditional maximum or average pooling. However, there is still no clear guide as to when to settle for one of the two options. Some theoretical studies defend that maximum pooling favours sparser feature representations (Boureau, Bach, LeCun, & Ponce, 2010; Boureau, Ponce, & LeCun, 2010) which are common in CNNs, but offer no explanation as to why average pooling performs better for some modern architectures (Huang et al., 2017). Therefore, the selection of pooling aggregation appears to be dependent on the input data as well as the precise model employed, acting as an additional hyperparameter.

Further, both maximum and average pooling ignore all possible relationship among the values to be reduced, potentially ignoring important spatial dependencies among the data.

Functions from aggregation theory (Beliakov, Sola, & Sánchez, 2016) such as fuzzy integrals can alleviate this problem, since

* Corresponding author.
E-mail address: bustince@unavarra.es (H. Bustince).

https://doi.org/10.1016/j.neunet.2022.04.028

they may model the dependencies among input values (Dimuro et al., 2020). These functions have been used with success in the past to this end (Dias et al., 2018), as well as in the context of other image processing tasks (Bardozzo et al., 2021; Bueno, Dias, Pereira Dimuro, Santos, & Bustince Sola, 2019; Mendoza, Melin, & Licea, 2009), so we believe they can improve the classic behaviours.

With respect to the selection of maximum or average pooling, in Lee, Gallagher, and Tu (2018) the authors test a convex combination of both methods as an alternative that, not only mitigates their individual shortcomings, but improves upon both of them in the general case. We think that this approach can be further extended if we consider a combination of additional functions which can retain different information.

In this paper, we propose to replace the pooling function of a CNN by functions other than the maximum and arithmetic mean, taking into account their respective behaviour. Since the maximum ignores all values but one, we consider other order statistics. By contrast, the arithmetic mean takes into account all the aggregated information, so we propose the Sugeno integral (Beliakov et al., 2016), as well as some new generalizations (Bardozzo et al., 2021), as an alternative that models the relationship between the input values.

It is important to remark that, although different in nature, the maximum and the arithmetic mean also share some properties. One such property is the monotonicity, in particular the increasingness. In the context of information fusion computing, the increasingness property is very important, since it guarantees that the higher the "quality of the information" provided by the inputs, the best the quality of the information of the aggregated value representing them (Dimuro, Costa, , & Claudio, 2000), on the light of Domain Theory (Abramsky & Jung, 1994; Stoltenberg-Hansen, Lindström, & Griffor, 1994). Motivated by this, we would like to keep this property in our proposal.

Thus, we present the linear combination of the previous increasing functions as a replacement for classic pooling layers, and refer to this strategy as Combination Pooling layer, or CombPool layer for short. We model the contribution of each function through new parameters of the model, whose values are optimized via the backpropagation algorithm, alike the rest of weights of a CNN. In order to obtain a resulting function which preserves increasingness, we study the conditions that these parameters must satisfy and ensure that they hold.

To empirically show the usefulness of the CombPool layer, we have replaced the pooling layers of three different CNN architectures of different complexity. The performance of every model is evaluated over five different datasets, showing improvements with respect to the traditional models. We compare our results with the traditional pooling functions, as well as some of the additional mentioned alternatives. We show that, in general, combinations which use a generalization of the Sugeno integral outperform most of them.

The paper is structured in the following way: Section 2 refreshes important notions related with CNNs and pooling functions, as well as several of the used functions; Section 3 presents the definitions and theorems that set the theoretical justification for the performed experiments and introduces the new pooling layer; Section 4 describes the different considered CNN architectures and datasets used in our experimentation, while Section 5 presents the obtained results; a brief discussion is held in Section 6, in which the strengths of the proposal are commented; finally, Section 7 ends the paper with some conclusions and ideas for future works.

Additionally, all lemmas and mathematical proofs related with the theory presented in Section 3 have been gathered in Appendix, with the intention of easing the readability of the paper.

2. Preliminaries

In this section, we recall some important notions, both related to CNN architectures as well as to increasing functions, which will be subsequently taken into account on the theoretical and experimental parts.

2.1. Convolutional neural networks

In the field of Deep Learning (Schmidhuber, 2015), convolutional neural networks (CNN) are a type of neural network designed for dealing with data where local information is of relevance, such as in temporal series (Abdel-Hamid, Deng, & Yu, 2013), image (Krizhevsky et al., 2012) or video (Le Callet, Viard-Gaudin, & Barba, 2006). These models perform two different processes over the input data: the first one is universal to all tasks and consists on the extraction of relevant features from the raw data, mapping input samples to a latent vector space. The second one uses the extracted features as input and can vary depending on the specific problem at hand, e.g. classifying samples into predefined categories or segmenting the different objects represented on a given image. Hereafter, we focus on image classification.

CNN receive their name in relation to the first aforementioned process, since the feature extraction is performed via the convolution of image kernels over an input. By input, we understand a matrix $X \in \mathbb{R}^{h\times w\times c}$, which can represent either an image of size $h \times w$, where each pixel $X_{ij} \in \mathbb{R}^c$ is represented by c colour channels ($c = 1$ for grey scale images, $c = 3$ for colour images) or a feature image, where each channel $X^d \in \mathbb{R}^{h\times w}$ represents the presence or absence of a feature over all parts of a given image. Each of those c feature maps will have been produced, at the same time, by the convolution of another kernel representing a specific feature over all parts of a previous input.

To be more precise, the output produced by a feature kernel $W \in \mathbb{R}^{k_1\times k_2\times c}$ for a $k_1 \times k_2$ window (where $k_1, k_2 \in \mathbb{N}$ are odd numbers) of an input $X \in \mathbb{R}^{h\times w\times c}$ centred on the pixel $X_{m,n}$ is given by the following expression:

$$Y_{m,n} = \sum_{d=1}^{c}\sum_{j=1}^{k_2}\sum_{i=1}^{k_1} W_{ij}^d \cdot X^d_{m-1+i-\frac{k_1-1}{2},n-1+j-\frac{k_2-1}{2}} \tag{1}$$

The values of these kernels are "learnt" similarly to the weights of any conventional neural network: they are randomly initialized and optimized iteratively using some variation of the gradient descent algorithm, usually stochastic gradient descent (Ruder, 2016).

CNN are composed of several convolutional layers (in addition to other types of layers) that operate sequentially over the input data so that the output generated by the lth layer is provided as input for the $(l+1)$th layer. Each convolutional layer is composed of c_l feature kernels that are applied to the received input, generating c_l different feature images that are stacked together before sending the output to the following layer.

Although the objective of the feature extractor is the reduction of the input image, the previous process has the opposite effect. In order to ensure this reduction, pooling layers are added at different points during the network. When a pooling layer receives an input $X \in \mathbb{R}^{h\times w\times c}$, each independent channel is separated in disjoint windows of size $k_1 \times k_2$. After that, the values of each window are aggregated by means of some function. The most popular ones are the maximum and the average.

2.2. Pooling functions

When LeCun, Bottou, Bengio, and Haffner (1998) presented the LeNet architecture, they used the average as pooling function, which attains reasonable performance for their particular model. However, some theoretical and empirical studies show clear improvements over similar architectures when using maximum pooling (Boureau, Bach, LeCun, & Ponce, 2010; Scherer, Müller, & Behnke, 2010), and it seems to have become the standard practice for most models (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; Szegedy et al., 2015). Even then, other equally competitive architectures perform better when applying average pooling (Huang et al., 2017), which hinders the selection over the best pooling function.

To this respect, several authors have provided alternatives to the classic pooling methods. Some of the most well known alternative pooling options differ in the objective they pursue, however: Spatial Pyramid Pooling was introduced as a means to remove the fixed size constraint of a CNN architecture, subsampling the output produced by the last convolutional layer of the network and generating fixed-length inputs for the classifier (He et al., 2015); Fractional Max Pooling puts the focus on the regions to be downsampled, which are randomly or pseudo-randomly chosen, instead of on the method in which to aggregate its values (Graham, 2014); fully convolutional networks remove the need of applying pooling functions altogether through their substitution by strided convolutional layers (i.e., convolutional layers in which filters skip over certain positions) (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2015) although they add more parameters to the model as a consequence.

There have been other attempts at substituting the aggregation function of the pooling operation nevertheless. In Zeiler and Fergus (2013), the authors present a stochastic method which randomly chooses one of the values to downsample according to a multinomial distribution based on their magnitude. In Lee et al. (2018), a convex combination of both average and maximum pooling is presented, in which the coefficients of each aggregation are learnt through various different strategies: "mixed" pooling sets the coefficients as learnable parameters of the model directly; the "gated" version uses kernels which generate different coefficients for each region to be downsampled based on its values, via a logistic regression (similarly to a channel-wise convolution); the "tree" version uses a differentiable decision tree to learn the proportion of each coefficient. A tentative to use the Choquet integral (Choquet, 1953–1954) in pooling layers was proposed by Dias et al. (2018, 2019).

In the experimental section we compare our proposal against some of these strategies, in addition to classic pooling strategies.

2.3. Increasing functions

We recall now some increasing functions that we use later in order to define linear combinations to set as pooling functions.

It will be assumed from now on that $2 \leq n \in \mathbb{N}$, $1 \leq r \in \mathbb{N}$.

A map $\mathbf{A}: \mathbb{R}^n \to \mathbb{R}$ is said to be an increasing function if $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$, that is, $x_i \leq y_i$ for all $i \in \{1, \ldots, n\}$, implies $\mathbf{A}(\mathbf{x}) \leq \mathbf{A}(\mathbf{y})$.

Set $N = \{1, \ldots, n\}$ and Σ_n as group of all possible permutations of N. For all $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and $\sigma \in \Sigma_n$, denote $\mathbf{x}_\sigma = (x_{\sigma(1)}, \ldots, x_{\sigma(n)})$; if $x_{\sigma(1)} \leq \cdots \leq x_{\sigma(n)}$, denote $\sigma \in \mathbf{x}_{(\nearrow)}$.

Consider $r \in \{1, \ldots, n\}$. Denote by $\mathbf{OS}_r$ the rth order statistic, that is, the function $\mathbf{OS}_r : \mathbb{R}^n \to \mathbb{R}$ given for all $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, by $\mathbf{OS}_r(\mathbf{x}) = x_{\sigma(r)}$, where $\sigma \in \mathbf{x}_{(\nearrow)}$, independently of the chosen σ.

We have, in particular, the maximum $\mathbf{max} = \mathbf{OS}_n$ and the minimum $\mathbf{min} = \mathbf{OS}_1$.

The arithmetic mean, is the function $\mathbf{AM} : \mathbb{R}^n \to \mathbb{R}$, given by $\mathbf{AM}(\mathbf{x}) = (x_1 + \cdots + x_n)/n$, for all $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$.

Now we recall the definition of fuzzy measure

Definition 2.1. A fuzzy measure on N is a map $\nu : 2^N \to [0, +\infty)$ such that

1. $\nu(\emptyset) = 0$ and
2. $S \subseteq T \subseteq N$ implies $\nu(S) \leq \nu(T)$.

Definition 2.2. The Sugeno integral associated to the fuzzy measure ν is the map

$$\mathbf{S}_\nu : \mathbb{R}^n \to \mathbb{R}$$

given, for $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, by

$$\mathbf{S}_\nu(\mathbf{x}) = \mathbf{max}\left(\mathbf{min}(x_{\sigma(1)}, \nu(N_1^\sigma)), \ldots, \mathbf{min}(x_{\sigma(n)}, \nu(N_n^\sigma))\right),$$

where $\sigma \in \mathbf{x}_{(\nearrow)}$ and $N_i^\sigma = \{\sigma(i), \ldots, \sigma(n)\}$.

A possible extension of the notion of Sugeno integral was presented in Bardozzo et al. (2021). We will revise it and offer an alternative definition in the following section.

3. Combination of increasing functions for the pooling function

As we have already said, when taking into consideration the classic pooling layers, the maximum and the arithmetic mean are the two most common options. Both of these functions are particular cases of aggregation operators, which respect boundary and monotonicity conditions. That is, both functions $\mathbf{F} : [a, b]^n \to [a, b]$ satisfy that $\mathbf{F}(a, \ldots, a) = a$ and $\mathbf{F}(b, \ldots, b) = b$, and that $F(x_1, \ldots, x_n) \leq F(y_1, \ldots, y_n)$ if $x_i \leq y_i$ for all $i \in \{1, \ldots, n\}$. Therefore, both functions are increasing functions.

In this section we study the conditions required for linear combinations of increasing functions to result into increasing functions. We consider order statistics, the arithmetic mean, the Sugeno integral and one of its generalizations. We conclude the section with the introduction of the CombPool layer, a pooling layer which enables the use of these linear combinations ensuring that the studied conditions are satisfied.

For this work, no other property is imposed, since we want to avoid restricting the representability capabilities of the CNN models. Due to the optimization strategy of neural networks, based on the gradient descent algorithm, setting further constraints usually results in weaker models rather than more robust ones.

3.1. Increasing function construction via linear combination of increasing functions

In the following we study linear combinations of increasing functions from $\mathbb{R}^n \to \mathbb{R}$, in some cases from $[0, +\infty)^n \to [0, +\infty)$.

We set $\mathbf{0} = (0, \ldots, 0)$, $\mathbf{1} = (1, \ldots, 1)$, $\mathbf{e}_i = (0, \ldots, \underset{i}{1}, \ldots, 0)$.

Let $\mathbf{A}_1, \ldots, \mathbf{A}_r : \mathbb{R}^n \to \mathbb{R}$ be increasing functions. We denote

$$\mathcal{I}(\mathbf{A}_1, \ldots, \mathbf{A}_r) = \{(\alpha_1, \ldots, \alpha_r) \in \mathbb{R}^n \mid \sum_{i=1}^{r} \alpha_i \mathbf{A}_i : \mathbb{R}^n \to \mathbb{R} \text{ is an increasing function}\}.$$

Proposition 3.1. *$(\mathcal{I}(\mathbf{A}_1, \ldots, \mathbf{A}_r), +)$ is a semigroup with neutral element $\mathbf{0}$. Moreover, let $\mathbb{R}^+$ denote the set of all positive real numbers, $(\mathbb{R}^+ \cup \{0\})^r \subseteq \mathcal{I}(\mathbf{A}_1, \ldots, \mathbf{A}_r)$.*

Negative coefficients may appear in elements of $\mathcal{I}(\mathbf{A}_1, \ldots, \mathbf{A}_r)$:

Remark 3.2. Consider $(\alpha_1, \ldots, \alpha_r) \in \mathcal{I}(\mathbf{A}_1, \ldots, \mathbf{A}_r)$, where $\mathbf{A}_i : \mathbb{R}^n \to \mathbb{R}$ is an increasing function, $i = 1, \ldots, r$, and $\mathbf{A} = \alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$; if $\alpha_i \neq 0$ we have

$$\mathbf{A}_i = \frac{1}{\alpha_i}\mathbf{A} - \frac{\alpha_1}{\alpha_i}\mathbf{A}_1 - .\overset{\widehat{i}}{.}. - \frac{\alpha_r}{\alpha_i}\mathbf{A}_r,$$

hence $\left(\frac{1}{\alpha_i}, -\frac{\alpha_1}{\alpha_i}, .\overset{\widehat{i}}{.}., -\frac{\alpha_r}{\alpha_i}\right) \in \mathcal{I}(\mathbf{A}, \mathbf{A}_1, .\overset{\widehat{i}}{.}., \mathbf{A}_r)$.

However it could be that negative elements are not allowed in a particular $\mathcal{I}(\mathbf{A}_1, \ldots, \mathbf{A}_r)$. This is always the case if $r = 1$:

Proposition 3.3. *Let $\mathbf{A} : \mathbb{R}^n \to \mathbb{R}$ be a non-constant increasing function. Let $\alpha \in \mathbb{R}$. Then $\alpha\mathbf{A}$ is an increasing function if and only if $\alpha \geq 0$.*

Observe that if $\mathbf{A} : \mathbb{R}^n \to \mathbb{R}$ is a constant function, then we have that $\alpha\mathbf{A}$ is also constant for all $\alpha \in \mathbb{R}$.

Notation 3.4. *Consider $k \in N$, $a, b \in \mathbb{R}$, $a \leq b$. We set*

$$(a, b \mid k+1) = a(\mathbf{e}_1 + \cdots + \mathbf{e}_k) + b(\mathbf{e}_{k+1} + \cdots + \mathbf{e}_n)$$

if $k < n$ and $(a, b \mid n) = a(\mathbf{e}_1 + \cdots + \mathbf{e}_n)$.

3.2. Combinations of order statistics and arithmetic mean

For the linear combination of order statistics to be an increasing function, the following must hold:

Proposition 3.5. *Consider $i_1, \ldots, i_r \in N$, $i_1 < \cdots < i_r$. Then, for all order statistics $\mathbf{OS}_{i_1}, \ldots, \mathbf{OS}_{i_r}$, it holds that*

$$\mathcal{I}(\mathbf{OS}_{i_1}, \ldots, \mathbf{OS}_{i_r}) = \{(\alpha_1, \ldots, \alpha_r) \mid \alpha_1, \ldots, \alpha_r \geq 0\}$$

As by definition $\mathbf{AM} = \frac{1}{n}(\mathbf{OS}_1 + \cdots + \mathbf{OS}_n)$, we have

Proposition 3.6. *Consider $i_1, \ldots, i_r \in N$, $i_1 < \cdots < i_r$, $r < n$. Then, for all order statistics $\mathbf{OS}_{i_1}, \ldots, \mathbf{OS}_{i_r}$, it holds that*

$$\mathcal{I}(\mathbf{AM}, \mathbf{OS}_{i_1}, \ldots, \mathbf{OS}_{i_r}) = \{(\alpha, \beta_1, \ldots, \beta_r) \mid \alpha, \alpha + n\beta_1, \ldots, \alpha + n\beta_r \geq 0\}.$$

Analogously, for all order statistics $\mathbf{OS}_1, \ldots, \mathbf{OS}_n$, we have that

$$\mathcal{I}(\mathbf{AM}, \mathbf{OS}_1, \ldots, \mathbf{OS}_n) = \{(\alpha, \beta_1, \ldots, \beta_n) \mid \alpha + n\beta_1, \ldots, \alpha + n\beta_n \geq 0\}.$$

3.3. Combinations with Sugeno integral

We write $\nu_S = \nu(S)$ for $S \subseteq N$, $\mathbf{x}_\sigma = (x_{\sigma(1)}, \ldots, x_{\sigma(n)})$, $\nu_i^\sigma = \nu(N_i^\sigma)$, $N_i = \{i, \ldots, n\}$ and $\nu_i = \nu(N_i)$. So $\mathbf{S}_\nu(\mathbf{x}) = \max_{i=1}^n(\min(x_{\sigma(i)}, \nu_i^\sigma))$. We know that

$$\mathbf{S}_\nu(x_1, \ldots, x_n) = \max_{S \subseteq N}[\min(\nu_S, (\min_{i \in S} x_i))].$$

Definition 3.7. A fuzzy measure $\nu : 2^N \to [0, +\infty)$ is said to be strict in $k \in N$ if either $k = n$ or there exists $\sigma \in \Sigma_n$ such that $\nu_k^\sigma > \nu_{k+1}^\sigma$. ν is strict if it is strict in k, for all $k \in N$.

Proposition 3.8. *Consider $i_1, \ldots, i_r \in N$, $i_1 < \cdots < i_r$, $r < n$. Assume that there exists $k \in N \setminus \{i_1, \ldots, i_r\}$ such that the fuzzy measure $\nu : 2^N \to [0, +\infty)$ is strict in k. Then, for all order statistics $\mathbf{OS}_{i_1}, \ldots, \mathbf{OS}_{i_r}$, and Sugeno integral $\mathbf{S}_\nu$, it holds that*

$$\mathcal{I}(\mathbf{OS}_{i_1}, \ldots, \mathbf{OS}_{i_r}, \mathbf{S}_\nu) = \{(\alpha_1, \ldots, \alpha_r, \beta \mid \alpha_1, \ldots, \alpha_r, \beta \geq 0)\}.$$

Corollary 3.9. *Consider $i_1, \ldots, i_r \in N$, $i_1 < \cdots < i_r < n$. For each fuzzy measure $\nu : 2^N \to [0, +\infty)$ one has, for all order statistics $\mathbf{OS}_{i_1}, \ldots, \mathbf{OS}_{i_r}$, and Sugeno integral $\mathbf{S}_\nu$,*

$$\mathcal{I}(\mathbf{OS}_{i_1}, \ldots, \mathbf{OS}_{i_r}, \mathbf{S}_\nu) = \{(\alpha_1, \ldots, \alpha_r, \beta) \mid (\alpha_1, \ldots, \alpha_r, \beta \geq 0)\}.$$

Proposition 3.10. *Let $\nu : 2^N \to [0, +\infty)$ be a fuzzy measure. If*

$$\alpha_1, \ldots, \alpha_n, \alpha_1 + \beta, \ldots, \alpha_n + \beta \geq 0,$$

then, for all order statistics $\mathbf{OS}_{i_1}, \ldots, \mathbf{OS}_{i_r}$, and Sugeno integral $\mathbf{S}_\nu$, $\alpha_1\mathbf{OS}_1 + \cdots + \alpha_n\mathbf{OS}_n + \beta\mathbf{S}_\nu$ is increasing. If $\alpha_1\mathbf{OS}_1 + \cdots + \alpha_n\mathbf{OS}_n + \beta\mathbf{S}_\nu$ is increasing and ν is strict in $k \in N$, then $\alpha_k + \beta \geq 0$; hence if ν is strict, one has that

$$\mathcal{I}(\mathbf{OS}_1, \ldots, \mathbf{OS}_n, \mathbf{S}_\nu) = \{(\alpha_1, \ldots, \alpha_n, \beta) \mid \alpha_1, \ldots, \alpha_n, \alpha_1 + \beta, \ldots, \alpha_n + \beta \geq 0\}.$$

Proposition 3.11. *Let $\nu : 2^N \to [0, +\infty)$ be a fuzzy measure. We have*

$$\mathcal{I}(\mathbf{AM}, \mathbf{S}_\nu) = \{(\alpha, \beta) \in \mathbb{R}^2 \mid \alpha, \alpha + n\beta \geq 0\}.$$

3.4. Generalized Sugeno integrals and combinations

Definition 3.12. Let $\mathbb{U}$ be a connected subset of $\mathbb{R}$ such that $0 \in \mathbb{U}$. A $\mathbb{U}$-fuzzy measure on N is a map $\nu : 2^N \to \mathbb{U}$ such that

1. $\nu(\emptyset) = 0$ and
2. $S \subseteq T \subseteq N$ implies $\nu(S) \leq \nu(T)$.

Obviously $\operatorname{im}\nu \subseteq [0, +\infty)$, but we need this more general assumption on $\mathbb{U}$ in the following.

Definition 3.13. Let $\mathbb{U}$ and $\mathbb{I}$ be two connected subsets of $\mathbb{R}$ such that $0 \in \mathbb{U} \subseteq \mathbb{I}$. Let $\nu : 2^N \to \mathbb{U}$ be a $\mathbb{U}$-fuzzy measure.

We say that the maps $\mathrm{F} : \mathbb{I} \times \mathbb{U} \to \mathbb{I}$ and $\mathrm{G} : \mathbb{I}^n \to \mathbb{U}$ are ν-admissible if the map $\mathbf{A} : \mathbb{I}^n \to \mathbb{I}$ given, for $x_1, \ldots, x_n \in \mathbb{I}$, by

$$\mathbf{A}(x_1, \ldots, x_n) = \mathrm{G}(\mathrm{F}(x_{\sigma(1)}, \nu(N_1^\sigma)), \ldots, \mathrm{F}(x_{\sigma(n)}, \nu(N_n^\sigma))),$$

where $\sigma \in \mathbf{x}_{(\nearrow)}$ and $N_i^\sigma = \{\sigma(i), \ldots, \sigma(n)\}$, is well defined (that is, the result does not change for $\sigma, \sigma' \in \mathbf{x}_{(\nearrow)}$) and increasing. Then we set $\mathbf{A} = \mathbf{A}(\mathrm{F}, \mathrm{G}, \nu)$ and name it the Sugeno-like $(\mathrm{F}, \mathrm{G}, \nu)$-function,

We write $\nu_S = \nu(S)$ for $S \subseteq N$, $\mathbf{x}_\sigma = (x_{\sigma(1)}, \ldots, x_{\sigma(n)})$, $\nu_i^\sigma = \nu(N_i^\sigma)$, $N_i = \{i, \ldots, n\}$ and $\nu_i = \nu(N_i)$. So, for $x_1, \ldots, x_n \in \mathbb{I}$, we have that

$$\mathbf{A}(x_1, \ldots, x_n) = \mathrm{G}(\mathrm{F}(x_{\sigma(1)}, \nu_1^\sigma), \ldots, \mathrm{F}(x_{\sigma(n)}, \nu_n^\sigma)).$$

Examples 3.14.

- With $\mathbb{U} = \mathbb{I} = \mathbb{R}$, if $\mathrm{F}(x, y) = \min(x, y)$ and $\mathrm{G}(\mathbf{x}) = \max(x_1, \ldots, x_n)$, we have the Sugeno integral S_ν given in 2.2. Assume, moreover, that the fuzzy measure ν is symmetrical (that is, $\nu(A) = \nu(B)$ if $A, B \subseteq N$ and $|A| = |B|$; we set then $\nu_i^\sigma = \nu_i$, as this only depends on i and, as usually it is not necessary to specify the permutations σ, also $x_{(i)} = x_{\sigma(i)}$). If $\mathrm{G}(x_1, \ldots, x_n) = x_1 + \cdots + x_n$, we set $\mathbf{A}(\mathrm{F}, \mathrm{G}, \nu) = \mathbf{A}(\mathrm{F}, \Sigma, \nu)$, so that, if $\mathbf{x} \in \mathbb{I}^n$,

  $$\mathbf{A}(\mathrm{F}, \Sigma, \nu)(\mathbf{x}) = \sum_{i=1}^n \mathrm{F}(x_{(i)}, \nu_i).$$

  For instance with $\mathbf{x} = (0, 1 \mid i+1)$, $\mathbf{y} = (0, 1 \mid i)$, $i \in N$, we have $x_{(i)} < y_{(i)}$ and $x_{(j)} = y_{(j)}$ if $i \neq j \in N$. Thus F and Σ are ν-admissible if and only if $x, y \in \mathbb{I}$, $x \leq y$ implies $\mathrm{F}(x, \nu_i) \leq \mathrm{F}(y, \nu_i)$ for $i = 1 \ldots, n$.

- For $F(x, y) = xy$, set $\mathbf{D}_\nu = \mathbf{A}(\Pi, \Sigma, \nu)$. Then $\mathbf{D}_\nu$ is the aggregation function given by $\mathbf{D}_\nu(\mathbf{x}) = \sum_{i=1}^{n} x_{(i)}\nu_i$, that is, $\mathbf{D}_\nu = \sum_{i=1}^{n} \nu_i \mathbf{OS}_i$.
  Let now $\mathbb{U} = [0, 1]$, $\mathbb{I} = [0, +\infty)$, ν satisfying $\nu(N) = 1$,
- Let H be the Hamacher t-norm corresponding to the null parameter and restricted to $[0, 1]$, that is, for $x \in [0, +\infty)$, $y \in [0, 1]$,

$$H(x, y) = \begin{cases} xy/(x + y - xy) & \text{if } (x, y) \neq (0, 0), \\ 0 & \text{if } (x, y) = (0, 0). \end{cases}$$

  Set $\mathbf{T}_\nu = \mathbf{A}(H, \Sigma, \nu)$. Thus $\mathbf{T}_\nu(\mathbf{x}) = \sum_{i=1}^{n} H(x_{(i)}, \nu_i)$.

Remarks 3.15. In cases 2 and 3 above it is necessary to impose the restriction on ν to be symmetrical: consider for instance the case $n = 2$ and assume that $\nu(\{i\}) = n_i$, $i = 1, 2$ and $n_1 \neq n_2$. Take $\mathbf{x} = (1, 1)$; the identity id and the transposition $\tau = O(1, 2)$ both belong to $\mathbf{x}_{(\nearrow)}$, and

$$x_{\mathrm{id}(1)}\nu_1^{\mathrm{id}} + x_{\mathrm{id}(2)}\nu_2^{\mathrm{id}} = \nu_N + n_2 \neq x_{\tau(1)}\nu_1^{\tau} + x_{\tau(2)}\nu_2^{\tau} = \nu_N + n_1$$

hence to be $\mathbf{D}_\nu$ well defined the symmetry of ν is necessary. We obtain the same result in relation to $\mathbf{T}_\nu$.

The restriction in the case 3 to $\mathbb{U} = [0, 1]$, $\mathbb{I} = [0, +\infty)$ is stated to avoid the occurrence of indeterminations.

In the following, consider the case of $\mathbf{D}_\nu = \mathbf{A}(\Pi, \Sigma, \nu)$, where $\nu : 2^N \to \mathbb{R}$ is a symmetric fuzzy measure.

It is immediate that $\mathbf{D}_\nu(c\,\mathbf{x}) = c\,\mathbf{D}_\nu(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$, $c \in [0, +\infty)$, that is, $\mathbf{D}_\nu$ is positively homogeneous.

Of course the above situation does not appear if $\nu_1 = 1$, except in the trivial case in which further $\nu_2 = \cdots = \nu_n = 0$.

Proposition 3.16. *Let us assume that $\nu_n \neq 0$ and $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$. Then $\mathbf{D}_\nu(\mathbf{x}) = \mathbf{D}_\nu(\mathbf{y})$ if and only if $\mathbf{x} = \mathbf{y}$.*

Proposition 3.17. *Consider $M \subseteq N$ and set $M' = N \setminus M$. We have*

$$\mathcal{I}(\mathbf{OS}_1, \ldots, \mathbf{OS}_M, \mathbf{D}_\nu) = \{(\alpha_1, \ldots, \alpha_M, \beta) \mid (\alpha_i + \beta\nu_i \geq 0 \text{ if } i \in M) \text{ and } (\beta\nu_i \geq 0 \text{ if } i \in M')\}.$$

Corollary 3.18. *Taking into account Proposition 3.17 and the fact that $\nu_1 \geq \nu_2 \geq \cdots \geq \nu_n \geq 0$, the following result holds.*

$$\mathcal{I}(\mathbf{AM}, \mathbf{D}_\nu) = \{(\alpha, \beta) \mid (\beta, \alpha + n\beta\nu_n \geq 0) \text{ or } (\beta \leq 0 \text{ and } \alpha + n\beta\nu_1 \leq 0)\}.$$

Proposition 3.19. *Let ν be strict. Then, we have*

$$\mathcal{I}(\mathbf{D}_\nu, \mathbf{S}_\nu) = \{(\alpha, \beta) \mid \alpha, \alpha\nu_n + \beta \geq 0\}.$$

3.5. CombPool layer: Combinations of increasing functions as pooling functions

Based on the previous theoretical developments, we now proceed to introduce a replacement for the classic max and average pooling layers, which we refer to as CombPool layer. Its process is as follows: we initially choose r increasing functions $\mathbf{A}_1, \ldots, \mathbf{A}_r$ such that $\mathbf{A}_i : \mathbb{R}^n \to \mathbb{R}$, for all $i = 1, \ldots, r$ and generate r coefficients $\alpha_1, \alpha_2, \ldots, \alpha_r \in \mathbb{R}, \alpha_i \geq 0$, for all $i = 1, \ldots, r$. Then, when reducing all $\mathbf{x} = (x_1, \ldots, x_n)$ values of each disjoint window of size $k_1 \times k_2$, with $k_1 \cdot k_2 = n$, of a feature image channel X^c, the n increasing functions are used, generating n different outputs $y_1, y_2, \ldots, y_n$. The combined output produced by our layer is calculated as the result of the following increasing function combination:

$$y = \sum_{i=1}^{r} \alpha_i^2 \cdot \mathbf{A}_i(\mathbf{x}) = \sum_{i=1}^{r} \alpha_i^2 \cdot y_i \tag{2}$$

An example of the resulting pooling process is represented in Fig. 1.

Coefficients $\alpha_1, \ldots, \alpha_r$ are set as additional parameters of the model, in the same way as the weights of a multilayer perceptron or the kernels of a CNN. In this way we can take profit of the backpropagation algorithm of neural networks in order to fine-tune their values. Notice that, in order to ensure that the resulting pooling function is increasing, we do not weight each output y_i with α_i, but with its squared value, a strategy that satisfies that monotonicity and differentiability hold for all possible combinations without needing to impose constraints to the learning process of α_i. Despite some of the Propositions in Sections 3.1–3.4 allowing for different parameterizations for the combinations to remain increasing, we find the strategy of squaring the coefficients to be the most efficient one. The conditions required by Propositions 3.5, 3.6, 3.10, 3.11, 3.17, Corollary 3.18 and Proposition 3.19, which cover all the combinations considered in our experimentation, are all satisfied following this approach.

Although learning a different set of coefficients for each window of the input is a valid approach, there are different strategies we can follow in order to reduce the number of learnable parameters of the resulting model, and therefore its complexity. We have tested the following options for each α_i coefficient, based on the approach presented in Lee et al. (2018): the same parameter for all windows and channels in the same layer; a different parameter for each independent window (shared among all channels); a different parameter for each channel (shared for each independent window); a different parameter for each channel and independent window.

In the end, however, results do not differ enough as to justify taking all combinations into account, and we have decided to report the models that use a different coefficient for each channel (shared by all windows), which appears to be a good compromise between representability capacity for the model and number of extra parameters.

If the increasing property of CombPool layers were to be dropped, the exponent of the coefficients could be removed from Eq. (2), and any function $\mathbb{R}^n \to \mathbb{R}$ could be applied as function $\mathbf{A}_i$. However, examples following this approach usually yield worse results. As a rather informal example, take Fig. 2. Here we have tested the performance of the same model on the same dataset when setting the function $F(\mathbf{x}) = \alpha \max \mathbf{x} + \beta \frac{1}{n} \sum_{i=1}^{n} x_i$ as pooling function. Instead of setting α and β as learnable parameters, we fix them to one of the values in $\{-1, -0.5, -0.1, 1\}$. The best results are obtained when both values are fixed to 1, which corresponds to the only increasing function out of all of them, and that accuracy rates tend to worsen the "more decreasing" the function becomes with respect to one of its terms.

4. Experimental framework

We now present the models and datasets which have been used to test the effectiveness of the CombPool layer.

4.1. Deep learning architectures

We have tested the effect of replacing the classic pooling layers of several different well known architectures by the CombPool layer presented in Section 3.5. We have chosen representative

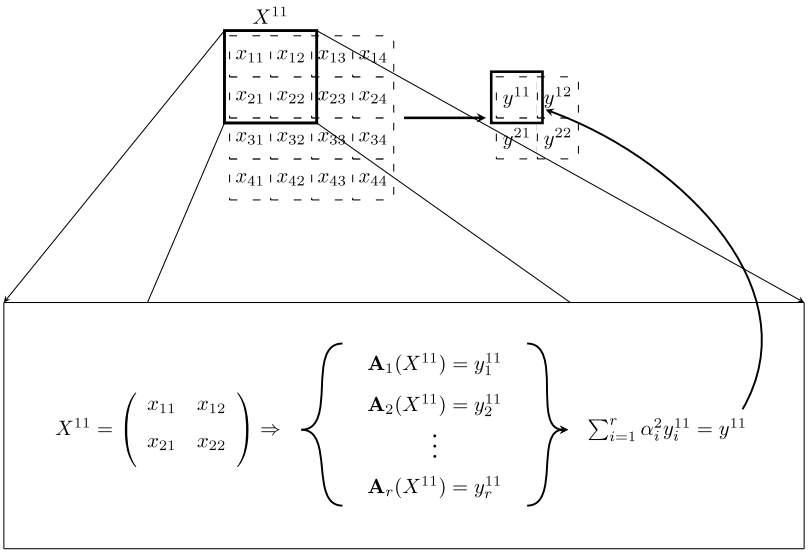

Fig. 1. Example of the combination of pooling functions. We compute r reductions for each different window, which are afterwards weighted by r parameters $\alpha \in \mathbb{R}$.

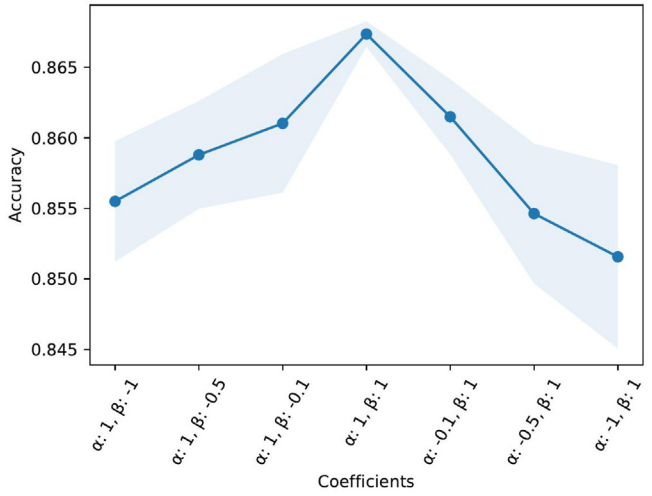

Fig. 2. Results obtained for different combinations of α and β when setting the values of the pooling function $F(\mathbf{x}) = \alpha \max \mathbf{x} + \beta \frac{1}{n} \sum_{i=1}^{n} x_i$ for the same experimental setting as one of the values in $\{-1, -0.5, -0.1, 1\}$. Results worsen the "more negative" one of the terms becomes.

models of exponentially increasing parameter count and overall complexity, to test the effectiveness of CombPool layers on different contexts. In this way, we intend to get an intuition on whether CombPool layers may be more beneficial on simpler or more complex learning tasks. The chosen models have been the following ones:

- **Architecture 1: LeNet-5**
  The structure of this network, presented in LeCun et al. (1998) is the one associated with the canonical CNN. It is composed of two blocks of convolution and pooling layers (traditionally average pooling), followed by a 3 layer perceptron which acts as classifier. We have set the ReLU function as activation function for the hidden layers and the Softmax function for the output layer. We have also inserted "Batch Normalization" layers after each pooling layer (Ioffe & Szegedy, 2015) in order to improve model convergence and mitigate overfitting. The architecture is detailed in Table 1.
- **Architecture 2: Network in Network**
  This architecture is identical to the one presented in Lee et al. (2018), and we have tested it because of the similarity between our proposals. It is based, in turn, in the concept of "Network in Network" models, which replace convolution layers by multilayer perceptrons (i.e. universal function approximators) and the final classifier by a Global Average Pooling layer (Lin, Chen, & Yan, 2014).
  Furthermore, this architecture version includes "hidden layer supervision" (Lee, Xie, Gallagher, Zhang, & Tu, 2015), a strategy which includes several classifiers at different points through the network in order to reinforce the gradient flow during training time and alleviate the "vanishing gradient" problem which deep models tend to suffer. Details of the architecture are available in Table 2.
- **Architecture 3: DenseNet**
  This CNN is a direct adaptation of the DenseNet architecture presented in Huang et al. (2017), and we have chosen it for being a clear representative of modern deep CNNs, composed by dozens of layers grouped in similar "blocks". DenseNets iterate on the concept of identity connections which ResNet architectures include as a mean to mitigate vanishing gradients, connecting the output of a layer with the input of the two following ones. In the case of DenseNet, each layer is connected with all the following ones.
  DenseNet networks are composed of a series of "dense blocks", formed by repetitions of Batch Normalization, ReLU, Convolutional and, optionally, Dropout layers (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). Since the output of each layer is processed by all the remaining layers, the authors refer by the factor k to the number of feature images which each convolutional layer adds to the "global knowledge" of the network. In order to avoid the number of filters of each following layer to keep increasing, 1×1 convolutions are set before each convolutional layer, in order to fix the size of these filters to a certain size.
  Pooling layers are added between every two dense blocks in order to progressively reduce the extracted feature dimensionality. 1×1 convolutions can be added again in order to reduce the number of channels of the global knowledge of the network G by a factor θ. Table 3 sums up the architecture.

Table 1
Architecture 1 description.

Layer type	Output size	Kernel size
Conv1	32 × 32 × 64	3 × 3
ReLU	32 × 32 × 64	–
Pool1	16 × 16 × 64	2 × 2
BatchNorm1	16 × 16 × 64	–
Conv2	16 × 16 × 64	3 × 3
ReLU	16 × 16 × 64	–
Pool2	8 × 8 × 64	2 × 2
BatchNorm2	8 × 8 × 64	–
Flatten	4096	–
MLP1	384	–
MLP2	192	–
MLP3	10	–

Table 2
Architecture 2 description. The output of the layers named "HLSupervision" are not used as input for the following layer, but output as a prediction vector that will be taken into account when computing the value of the loss function. The following layer takes its input from the layer previous to the "HLSupervision" one.

Layer type	Output size	Kernel size
Conv1	32 × 32 × 128	3 × 3
ReLU	32 × 32 × 128	–
HLSupervision1	10	–
Conv2	32 × 32 × 128	3 × 3
ReLU	32 × 32 × 128	–
HLSupervision2	10	–
Mlpconv1	32 × 32 × 128	1 × 1
Pool1	16 × 16 × 128	3 × 3
Conv3	16 × 16 × 192	3 × 3
ReLU	16 × 16 × 192	–
HLSupervision3	10	–
Conv4	16 × 16 × 192	3 × 3
ReLU	16 × 16 × 192	–
HLSupervision4	10	–
Mlpconv2	16 × 16 × 192	1 × 1
Pool2	8 × 8 × 192	3 × 3
Conv5	8 × 8 × 256	3 × 3
ReLU	8 × 8 × 256	–
HLSupervision5	10	–
Conv6	8 × 8 × 256	3 × 3
ReLU	8 × 8 × 256	–
HLSupervision6	10	–
Mlpconv3	8 × 8 × 256	1 × 1
Mlpconv4	8 × 8 × 10	1 × 1
GlobalAvgPool	10	8 × 8

Although our main study focuses on these 3 initial architectures, we have also tested the use of CombPool layers in other currently relevant areas. For these tests, we also consider the following two architectures:

- **Architecture 4: RegNetX**
  The authors of Radosavovic, Kosaraju, Girshick, He, and Dollár (2020) focused, not in presenting a particular architecture, but a method for statistically searching for the most effective architecture design decisions. They started studying a family of models based on the RESNet architecture, and progressively restricted the model until reaching a family of models named RegNetX. The specific details of the research goes beyond the scope of this work and can be found in the original paper.
  Interestingly, RegNetX models dispense with pooling layers, similarly to other current models such as RESNet, MobileNet (Howard et al., 2017) and EfficientNet (Tan & Le, 2019). Instead, they perform feature downsampling through convolution layers with stride bigger than 1. However, most of these architectures do use Global Average Pooling layers, which aggregate all the values on each output channel through the arithmetic mean. We have tested if the replacement of Global Average Pooling by Global CombPool layers could be of interest for its consideration in these architectures.
  For our experiments, we have worked with the RegNetX-200MF version. All hyperparameters are kept the same with respect to the original paper, with the difference that we work with smaller image sizes.

Table 3
Description of the third architecture. k represents the growth rate of the network, which has been set to 12. G specifies the number of global knowledge channels up until each layer, and increases by k after each convolution of a dense block. We explicitly represent the compression of this global knowledge indicating at the output of the transition block convolutions that the number of feature images becomes $G/2$, since we set $\theta = 0.5$. However, we return to using G on following layers in favour of clearer notation.

Layer type		Output size	Kernel size
Initial block	Conv	32 × 32 × $2k$	3 × 3
	BatchNorm	32 × 32 × $2k$	–
	ReLU	32 × 32 × $2k$	–
Dense block 1 (×16)	BatchNorm	32 × 32 × G	–
	ReLU	32 × 32 × G	–
	Conv	32 × 32 × $4k$	1 × 1
	BatchNorm	32 × 32 × $4k$	–
	ReLU	32 × 32 × $4k$	–
	Conv	32 × 32 × k	3 × 3
Transition block 1	BatchNorm	32 × 32 × G	
	ReLU	32 × 32 × G	
	Conv	32 × 32 × $G/2$	1 × 1
	Pool	16 × 16 × $G/2$	
Dense block 2 (×16)	BatchNorm	16 × 16 × G	–
	ReLU	16 × 16 × G	–
	Conv	16 × 16 × $4k$	1 × 1
	BatchNorm	16 × 16 × $4k$	–
	ReLU	16 × 16 × $4k$	–
	Conv	16 × 16 × k	3 × 3
Transition block 2	BatchNorm	16 × 16 × G	
	ReLU	16 × 16 × G	
	Conv	16 × 16 × $G/2$	1 × 1
	Pool	8 × 8 × $G/2$	
Dense block 3 (×16)	BatchNorm	8 × 8 × G	–
	ReLU	8 × 8 × G	–
	Conv	8 × 8 × $4k$	1 × 1
	BatchNorm	8 × 8 × $4k$	–
	ReLU	8 × 8 × $4k$	–
	Conv	8 × 8 × k	3 × 3
GlobalAvgPool		G	8 × 8
MLP		10	–

- **Architecture 5: Scalable Vision Transformer**
  The original transformer architecture was presented in Vaswani et al. (2017) and has seen huge success in the Natural Language Processing (NLP) domain, with results as impressive as those of Brown et al. (2020) and Devlin, Chang, Lee, and Toutanova (2018), among others.
  It is based on the concept of "self-attention", a mechanism by which the relationship between the different values of an input is weighted, in order to give more importance to the most relevant ones. In the context of NLP, this would allow to measure the relationship between every pair of words in a sentence, according to a given objective. Actually, transformers use "multi-head attention" layers, which perform the previous process multiple times, each one focusing on a particular feature of the input. Similarly to DenseNets or ResNets, transformers are composed of a concatenation of similar blocks which perform the same operations. A detailed explanation of the architecture goes beyond the scope of this paper, and can be found in Dosovitskiy et al. (2020).

Table 4
Datasets MNIST (LeCun et al., 1998), Fashion-MNIST (Xiao, Rasul, & Vollgraf, 2017), EMNIST (Cohen, Afshar, Tapson, & van Schaik, 2017), CIFAR10 and CIFAR100 (Krizhevsky, Hinton, et al., 2009) used in the experimental part. By "balanced" EMNIST we refer to the balanced partition of all the available subsets of the EMNIST dataset.

Dataset	Train	Test	Classes	Colour	Description
MNIST	60000	10000	10	No	Digits from 0 to 9
Fashion MNIST	60000	10000	10	No	Clothing categories
Balanced EMNIST	112800	18800	47	No	Digits and characters
CIFAR10	50000	10000	10	Yes	Real life images
CIFAR100	50000	10000	100	Yes	Real life images

NLP Transformers split an input sentence into parts and encode each of them generating a different input "token". In Dosovitskiy et al. (2020), the authors propose an adaptation of this strategy to work on image data instead, which they refer to as Vision Transformer (ViT). Unlike CNNs which work with whole images, for ViT inputs are previously split into disjoint patches which are encoded to generate the distinct input tokens. These input tokens are afterwards sequentially processed via a series of blocks of layers until obtaining the probability vector which can be used to classify the image.
In our experiments we have worked with a particular implementation of ViT, known as Hierarchical Visual Transformer (HVT) which appends pooling layers between the different blocks of the transformer (Pan, Zhuang, Liu, He, & Cai, 2021). Outputs are sequentially reduced in dimensionality, and the later blocks operate over the reduced feature vectors. Our preliminary results are presented in Section 5.5.

All models have been optimized via stochastic gradient descent with momentum and L2 regularization, setting Cross Entropy as loss function. We use batch sizes of 128 samples.

4.2. Datasets

The performance of all the different models has been evaluated on the datasets presented in Table 4.

All models have been trained for 50 epochs over all datasets but CIFAR10/CIFAR100, in which they have been trained for 200 epochs. Architecture 3, due to its significantly higher amount of parameters, has been trained for 300 epochs over that same dataset. In order to try to limit the influence of stochastic initialization over our final results, each model has been independently trained 5 times and we report their mean accuracy, with the exception of the third architecture over datasets other than CIFAR10/CIFAR100. We consider that the variance over the results obtained for those datasets with that architecture were so small that it did not justify the computational and temporal cost required.

5. Experimental study

In this section we test the effectiveness of the CombPool layer. We compare the differences among the different combinations, as well as with models that use individual functions as their pooling method, including the classic maximum and average pooling. Additionally, we include results obtained with other pooling methods presented in Section 2.2, namely stochastic and gated pooling. Stochastic pooling replaces the deterministic behaviour of max pooling by randomly choosing one of the input values, sampling from a multinomial distribution constructed from those values. Gated pooling computes a combination of pooling functions through a convex combination of the type $F(\mathbf{x}) = \alpha\mathbf{A}_1(\mathbf{x}) + \beta\mathbf{A}_2(\mathbf{x})$, with the difference that α and β are computed as the result of a linear transformation of the values in $\mathbf{x}$. In that sense, rather than learning those parameters directly, the coefficients of the linear transformations are learnt.

Whenever the Sugeno integral $\mathbf{S}_\nu$ or its $\mathbf{D}_\nu$ generalization is used, we set the "power measure" ν_{pm} given by $\nu_{pm} = |X|^q/n$ for all $X \subseteq N$ as ν, a well known fuzzy measure which has been used with success in plenty of applications (Bardozzo et al., 2021; Lucca et al., 2019). Notice that, as commented in Remarks 3.15, in order for $\mathbf{D}_\nu$ to be well defined, ν must be symmetrical, which ν_{pm} satisfies. We have set the value of $q = 2$ based on the results of preliminary tests and previous experience in other domains.

5.1. Pooling by means of individual functions

Initially, we tested several models that use a single function for the pooling process. Apart from average and maximum pooling, we have tested the minimum, the median, the Sugeno integral and the $\mathbf{D}_\nu$ generalization. The test results are shown in Table 5.

Although for both architectures 1 and 3 the best results are obtained with either the arithmetic mean or the maximum, it is interesting noting how for architecture 2, the generalization of the Sugeno integral performs the best for most datasets. In fact, as will be shown later, these options outperform even the combination of maximum and arithmetic mean, which is the one presented in Lee et al. (2018). However, in other cases, such as for the architecture 3 on CIFAR10 and CIFAR100, the $\mathbf{D}_\nu$ generalization is the one that performs the worst, which lets clear that the best function is both architecture and dataset dependant.

Another conclusion is that the more complex models seem to have higher adaptability to non-standard functions whereas architecture 1 performs by far the best when using either average or maximum pooling, specially when faced with more complex datasets. Notice that, when considering the results obtained on CIFAR10 or CIFAR100, the only accuracy rates close to the ones reported when using the maximum or the average, are both versions of the Sugeno integral on CIFAR100, which perform marginally worse nonetheless. This might be due to the fact that the higher number of parameters of the rest of models allows them to model more complex functions which can adapt more easily to other pooling functions.

5.2. Pooling using combinations of increasing functions

We have repeated the previous experimentation replacing the individual pooling layers by CombPool layers, testing plenty of increasing function combinations. The accuracy rate for all models and datasets is presented in Table 6.

As stated in the previous section, architecture 1 has difficulty adapting to non-standard pooling functions. It is specially noteworthy for datasets EMNIST, CIFAR10 and CIFAR100, the hardest ones, where it only provides comparable results when one of the standard functions is part of the combination.

Similarly, for the MNIST and FASHION dataset, most models offer results close to the individual variants, with the combinations that include the $\mathbf{D}_\nu$ generalization performing the best in average. Specifically, architecture 2 obtains the best result of any combination when using some of them.

If we focus on architecture 3, for datasets EMNIST and CIFAR10 all groups of combinations manage to outperform the best individual accuracy obtained using the arithmetic mean. In particular, combining the $\mathbf{D}_\nu$ generalization with the arithmetic mean obtains the best results for CIFAR10, while combining it with the maximum offers the second best accuracy over EMNIST, only slightly behind the combination of the $\mathbf{D}_\nu$ integral and the arithmetic mean. For CIFAR100 none of the combinations

Table 5
Accuracy rate for architectures that use individual functions. Column Ai indicates references results obtained with the *i*th architecture.

	MNIST			FASHION			EMNIST			CIFAR10			CIFAR100		
	A1	A2	A3	A1	A2	A3	A1	A2	A3	A1	A2	A3	A1	A2	A3
AM	**99.37**	99.09	99.10	**93.24**	91.81	**93.79**	**87.58**	88.59	**89.52**	77.08	87.65	**89.25**	**46.55**	54.72	**70.78**
Max	99.24	99.39	99.34	92.80	92.99	92.92	87.31	89.11	89.27	**77.39**	87.85	87.99	45.68	**57.58**	68.48
Min	98.98	99.38	99.27	92.13	**93.03**	92.99	86.22	89.05	89.40	70.24	87.61	88.28	42.26	51.72	68.86
Median	98.98	99.17	99.03	91.80	92.28	93.48	86.58	88.81	89.31	70.62	87.07	88.76	39.29	51.65	69.58
$\mathbf{S}_\nu$	99.14	99.24	**99.39**	92.01	92.07	93.56	86.51	88.71	89.44	72.47	86.79	88.97	44.73	54.71	68.42
$\mathbf{D}_\nu$	99.26	**99.46**	99.08	93.05	92.67	92.86	86.64	**89.27**	89.31	73.42	**88.70**	87.20	43.52	54.11	68.97

Table 6
Accuracy rate for models that replace classic max and mean pooling layers by CombPool layers. Each row presents the obtained result for a given combination of functions. Combinations have been grouped according to the order in which they have been presented in Section 3.

	MNIST			FASHION			EMNIST			CIFAR10			CIFAR100		
	A1	A2	A3	A1	A2	A3	A1	A2	A3	A1	A2	A3	A1	A2	A3
Min + Max	99.30	**99.30**	99.02	92.56	91.91	92.77	87.31	**88.68**	89.26	**77.02**	87.42	88.55	**45.64**	55.14	68.8
Min + Max + Median	**99.30**	99.21	**99.14**	**92.63**	**91.93**	**93.51**	**87.52**	88.50	**89.62**	76.91	**87.43**	**89.77**	45.35	**55.84**	70.21
AM + Min	99.30	**99.27**	99.11	92.41	92.13	92.59	86.97	88.66	89.57	75.04	87.23	89.48	45.22	45.22	69.83
AM + Max	99.32	99.25	99.27	**92.86**	**92.47**	**93.63**	**87.35**	88.43	89.62	**77.23**	**87.78**	86.99	**45.80**	55.43	70.19
AM + Min + Max	99.30	99.25	**99.35**	92.68	91.91	92.99	87.14	88.86	**89.75**	77.17	87.25	88.52	45.78	55.79	69.40
AM + Min + Max + Median	**99.33**	99.26	99.29	92.72	92.17	92.85	87.34	**88.91**	89.22	77.04	87.57	**89.83**	46.27	**55.98**	69.95
$\mathbf{S}_\nu$ + Min	99.07	99.27	99.24	92.23	92.18	93.25	86.49	88.53	89.26	72.41	87.46	88.58	43.50	54.35	68.24
$\mathbf{S}_\nu$ + Max	99.22	**99.43**	99.43	92.96	**92.39**	93.19	87.11	88.52	89.66	77.09	87.60	89.48	46.37	54.83	67.66
$\mathbf{S}_\nu$ + Min + Max	99.27	99.36	99.45	92.71	92.01	93.18	**87.33**	**88.87**	89.41	**77.30**	87.01	89.42	46.29	55.71	69.43
$\mathbf{S}_\nu$ + Min + Max + Median	99.24	99.27	98.95	92.78	92.04	**93.36**	87.10	88.83	89.49	77.03	87.29	**89.66**	45.68	**56.08**	70.04
$\mathbf{S}_\nu$ + AM	**99.34**	99.21	**99.49**	**93.21**	91.93	93.25	87.07	88.35	**90.03**	76.93	**88.23**	86.99	**46.46**	53.92	70.00
$\mathbf{D}_\nu$ + Min	99.21	99.33	99.15	92.23	**92.60**	92.78	86.15	88.77	89.09	72.19	88.51	89.03	42.94	55.44	69.45
$\mathbf{D}_\nu$ + Max	99.28	99.27	99.22	92.80	92.55	93.32	86.95	**89.09**	**89.97**	76.80	88.20	89.58	**46.16**	**55.97**	69.25
$\mathbf{D}_\nu$ + Min + Max	99.27	**99.38**	99.27	92.66	92.53	92.69	87.37	88.69	89.46	**77.81**	**88.61**	89.83	45.78	55.73	**70.31**
$\mathbf{D}_\nu$ + Min + Max + Median	99.32	99.35	99.22	**92.87**	92.57	**93.79**	**87.46**	89.06	89.85	76.15	88.30	89.75	45.19	55.77	69.98
$\mathbf{D}_\nu$ + AM	**99.32**	99.35	**99.39**	92.69	92.56	93.21	87.37	88.40	89.61	76.39	88.40	**89.87**	44.89	55.04	69.56
Stochastic	**99.34**	**99.35**	**99.53**	92.49	**92.17**	**93.64**	**87.45**	**88.91**	**89.81**	**77.43**	87.29	**91.57**	**47.77**	**58.18**	**70.68**
Gated	99.33	**99.35**	99.51	**92.91**	92.03	93.12	87.23	88.27	89.71	**77.43**	**88.61**	90.41	46.38	57.26	69.99

outperform the arithmetic mean, but combinations with the $\mathbf{D}_\nu$ generalization still offer competitive results.

We have also compared our results with both stochastic and gated pooling, which perform on par with the best functions, even though there is some important variance on simpler datasets. As previously stated, architecture 3 is the one which benefits the most from using non-standard pooling functions, and both methods show clear superior results for dataset CIFAR10. The comparison with gated pooling is not completely fair however, since we could employ the "gated" strategy for learning the coefficients of our CombPool layers and improve its respective best results. We will test this option in the next subsection.

5.3. Gated CombPool layers

Even though our main study focus is the increasing function combinations and not the coefficient learning strategy, results on Table 6 which show a clear higher accuracy for models which employ gated pooling raise the question of whether that strategy can be adapted in CombPool layers. In order to achieve this, we simply replace each coefficient a_i by a kernel $W_i \in \mathbb{R}^{k_1 \times k_2}$. Coefficients a_i are therefore computed for each different window via the channel-wise convolution of W_i over each window of the input.

In order to test if the presented combinations still outperform the average and maximum combination when using gated pooling, we have focused on one of the hardest setups, the one which trains architecture 3 on the CIFAR10 dataset and have trained a model using the $\mathbf{D}_\nu$ generalization and arithmetic mean combination, which performs the best when following the mixed strategy. Table 7 shows that using the gated version of this combination increases the accuracy offered by mixed pooling by one point, and the one obtained by the original gated pooling combination by half a point.

Table 7
Effect of the "gated" strategy applied to the CombPool layer. We focus on the $\mathbf{D}_\nu$ and arithmetic mean combination, the best combination for the "mixed" strategy in one of the hardest setups (Architecture 3 trained on the CIFAR10 dataset). The responsive nature of "gated" pooling results in improved models for both cases, but the presented combination still surpasses the maximum and arithmetic mean combination in this case.

Method	Accuracy
Mixed AM + Max	86.99
Mixed $\mathbf{D}_\nu$ + AM	89.87
Gated AM + Max	90.41
Gated $\mathbf{D}_\nu$ + AM	90.89

5.4. Global CombPool layers

Here we explore the modification of another pooling process in CNNs: "Global pooling". Global pooling is similar to classical pooling, with the difference that it is applied to all values of a feature channel at once, aggregating them into a single value. It is applied after the feature extraction process of a CNN in order to either "flatten" the resulting features into a "feature vector" to be fed for the classifier, or substituting the classifier altogether. It was first introduced in Lin et al. (2014) with this latter objective in mind, replacing the Multilayer Perceptron of their proposed model by a last convolution layer with as many convolution filters as classes followed by a Global Average Pooling layer. In this context, global pooling had the objective of forcing the last convolution filters of the model to learn transformations directly associated with each of the predicted classes, so that each one would search for the most representative feature of a given class.

Table 8
Accuracy rate for models that replace classic Global Average pooling by Global CombPool layers.

	CIFAR10			CIFAR100		
	A2	A3	A4	A2	A3	A4
AM	86.11	91.08	94.13	57.16	**70.97**	**74.95**
Max + AM	**86.97**	**91.29**	93.77	57.23	68.79	71.43
Max + $\mathbf{S}_\nu$	83.04	91.28	93.40	50.96	65.4	58.96
Max + $\mathbf{D}_\nu$	85.99	90.26	93.27	52.85	66.54	66.66
AM + $\mathbf{S}_\nu$	86.58	91.00	**94.25**	**57.72**	69.68	74.30
AM + $\mathbf{D}_\nu$	86.33	91.08	93.51	52.71	69.10	71.43

Nowadays, Global Average Pooling has become a staple of modern CNN architectures. Although in the literature the arithmetic mean is the only typical implementation of Global Pooling, we believe that CombPool layers can be directly used in order to replace this process. To illustrate this possibility, we have tested both models 2 and 3 (which already used Global Average Pooling), in conjunction with the more recently proposed architecture 4. The results are presented in Table 8.

As can be seen, some of the tested CombPool layers can outperform Global Average Pooling. In particular, the average and Sugeno integral combination obtains the best results for 2 of the performed tests, while offering decent accuracy rates in the rest of settings, as does the maximum and arithmetic mean combination. It is also clear that combinations which do not include the arithmetic mean as a member offer poor results. In fact, those instances obtain some of the worst all around results. This makes sense if we take into account that, when using the maximum in order to aggregate many values, most of the available information is discarded.

5.5. CombPool Layers in vision transformers

In this section we study the possibility of applying CombPool layers to the novel ViT architecture, in particular to the HVT implementation which uses pooling layers in a similar way to DenseNets.

Due to the complexity of HVT training, we have employed the same hyperparameter configurations as specified by the authors of Dosovitskiy et al. (2020). In particular, we have tested the configurations they refer to as HVT_Ti-1, HVT_S-1 and HVT_S-4, which differ both in parameter count and in number of pooling layers employed. Models HVT_Ti-1 and HVT_S-1 use a single pooling layer before the last block of the model, while HVT_S-4 distributes 4 pooling layers along the architecture.

The results for the performed experiments are presented in Table 9. As it can be seen, CombPool layers perform in pair with maximum pooling layers, but they only slightly outperform them for HVT_Ti-1. However, we find this initial results promising and believe that further experimentation and hyperparameter finetuning could improve the performance of CombPool layers for HVT. We would like to explore this possibility in-depth in a future work.

6. Discussion

Table 10 sums up the best results obtained among the tests presented in Sections 5.1 and 5.2 which use CombPool layers. Based on it, as well as on the results of the previous experiments, we find some interesting behaviours.

Firstly, the combination of functions results in a valuable strategy for incorporating and taking profit of functions with poor individual performance. For example, the minimum, median and Sugeno integral perform the worst for several of the architectures presented, but their combination with other functions give raise to some of the best solutions for each experiment. This opens the possibility to testing other less common functions in the future.

In general, the $\mathbf{D}_\nu$ generalization appears to be the most interesting function to be used as part of a CombPool layer. We suspect that its good performance can be explained from its similarity to the classic convolution expression. However, in contrast to the convolution operation, which weighs the importance of each value by means of learnable parameters, the Sugeno generalization uses a fuzzy measure, whose values can be fixed beforehand preventing an increase on the number of parameters of the model.

Additionally, it appears that the CombPool addition results more beneficial the more complex the models and datasets used are. While for simple datasets and models its use should be avoided, most combinations offer above average results for architecture 2 and 3 in the hardest datasets.

Finally, when compared with other pooling strategies, Table 11 shows that CombPool layers perform on par to most methods, and its best combinations usually outperform more heavily parameterized strategies such as standard gated pooling. Interestingly, the $\mathbf{D}_\nu$ integral pooling already offers a good performance by itself, obtaining the best results for architecture 2 in three out of the five datasets.

We have also showed that adapting the learning mechanism for the increasing function combination of Gated pooling can further increase the performance of CombPool layers, at the expense of adding more parameters to the model.

Similarly, we have seen that CombPool layers can be considered to replace Global Average Pooling in modern architectures, specially when considering combinations among the arithmetic mean and either the Sugeno integral or the maximum.

7. Conclusions and future work

In this work we have tested the performance of three fairly different convolutional network architectures and replaced their pooling functions by linear combinations of increasing functions. We have proven that the arithmetic mean and maximum are not the only realistical options when testing new models, and that the strategy of combining the results of multiple functions can improve the final performance of the algorithm.

The main takeaways of our experiments are the following:

- Simple architectures lack the capability of modelling complex functions that might benefit from using non standard functions. When using combinations of functions in these architectures, adding either the maximum or the arithmetic mean as part of the combination has been the only reliable option for obtaining favourable results.
- Although some individual functions, such as the Sugeno integral, are poor choices as pooling functions, combining them with other aggregations can improve their performance.
- The $\mathbf{D}_\nu$ Sugeno integral can obtain good results by itself, and offers reliable good results for all architectures when used as part of a combination of increasing functions. Most combinations report results above the mean result of all tests, and outperform the ones obtained by the combination of the two standard pooling methods.

Due to the positive results obtained through the generalization of the Sugeno integral, we intend on testing the performance of models which include the Choquet integral and its generalizations as a term of the CombPool layer.

We have also found that CombPool layers can be used as a substitution for the usual Global Average Pooling employed in modern architectures. Similarly, they can be introduced in

Table 9
Accuracy rate for different configurations of Hierarchical Vision Transformer on datasets CIFAR10 and CIFAR100. Maximum pooling, average pooling and CombPool layers are being tested.

	CIFAR10						CIFAR100					
	HVT-Ti-1		HVT-S-1		HVT-S-4		HVT-Ti-1		HVT-S-1		HVT-S-4	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Max	86.30	99.17	**88.86**	**99.13**	**90.41**	**99.13**	**58.22**	**82.86**	**55.04**	**77.02**	**57.93**	**78.64**
AM	83.91	98.94	86.65	98.85	86.83	98.76	55.84	81.16	52.95	75.71	51.87	75.21
Max + Avg	86.24	99.17	86.92	99.01	88.40	98.91	58.15	82.71	53.79	75.95	55.49	76.64
Max + $\mathbf{S}_\nu$	**86.60**	**99.18**	86.26	99.09	88.82	99.00	58.15	82.50	51.98	73.69	51.34	74.20
Max + $\mathbf{D}_\nu$	86.59	99.41	88.45	99.12	88.82	99.04	57.69	82.79	55.01	76.61	53.36	74.78
Avg + $\mathbf{S}_\nu$	84.49	99.06	86.21	98.91	87.11	98.76	55.95	80.98	52.34	74.81	51.75	74.29
Avg + $\mathbf{D}_\nu$	85.95	99.00	88.45	99.12	88.81	98.90	57.11	81.92	54.73	76.08	55.28	77.12

Table 10
Summary of the top 3 best combinations of pooling functions for each architecture over each of the evaluated datasets.

	MNIST dataset		
	Arch. 1	Arch. 2	Arch. 3
1st Best	AM	$\mathbf{D}_\nu$	$\mathbf{S}_\nu$ + AM
Accuracy	99.37	99.46	99.49
2nd Best	$\mathbf{S}_\nu$ + AM	$\mathbf{S}_\nu$ + Max	$\mathbf{S}_\nu$ + Min + Max
Accuracy	99.34	99.43	99.45
3rd Best	AM + Min + Max + Median	Max	$\mathbf{S}_\nu$ + Max
Accuracy	99.33	99.39	99.43
	FASHION dataset		
	Arch. 1	Arch. 2	Arch. 3
1st Best	AM	Min	$\mathbf{D}_\nu$ + Min + Max + Median
Accuracy	93.24	93.03	93.79
2nd Best	$\mathbf{S}_\nu$ + AM	Max	AM
Accuracy	93.21	92.99	93.79
3rd Best	$\mathbf{D}_\nu$	$\mathbf{S}_\nu$	AM + Max
Accuracy	93.05	92.67	93.63
	EMNIST dataset		
	Arch. 1	Arch. 2	Arch. 3
1st Best	AM	$\mathbf{D}_\nu$	$\mathbf{S}_\nu$ + AM
Accuracy	87.58	89.27	90.03
2nd Best	Min + Max + Median	Max	$\mathbf{D}_\nu$ + Max
Accuracy	87.52	89.11	89.97
3rd Best	$\mathbf{D}_\nu$ + Min + Max + Median	$\mathbf{D}_\nu$ + Max	$\mathbf{D}_\nu$ + Min + Max + Median
Accuracy	87.46	89.09	89.85
	CIFAR10 dataset		
	Arch. 1	Arch. 2	Arch. 3
1st Best	$\mathbf{D}_\nu$ + Min + Max	$\mathbf{D}_\nu$	$\mathbf{D}_\nu$ + AM
Accuracy	77.81	88.70	89.87
2nd Best	Max	$\mathbf{D}_\nu$ + Min + Max	$\mathbf{D}_\nu$ + Min + Max
Accuracy	77.39	88.61	89.83
3rd Best	$\mathbf{S}_\nu$ + Min + Max	$\mathbf{D}_\nu$ + Min	AM + Min + Max + Median
Accuracy	77.30	88.51	89.83
	CIFAR100 dataset		
	Arch. 1	Arch. 2	Arch. 3
1st Best	AM	Max	AM
Accuracy	46.55	57.58	70.78
2nd Best	$\mathbf{S}_\nu$ + AM	$\mathbf{S}_\nu$ + Min + Max + Median	$\mathbf{D}_\nu$ + Min + Max
Accuracy	46.46	56.08	70.31
3rd Best	$\mathbf{S}_\nu$ + Max	AM + Min + Max + Median	Min + Max + Median
Accuracy	46.37	55.98	70.21

Table 11
Comparison between the best CombPool layer for each model and other pooling methods. We compare our proposal against average pooling, max pooling and the Sugeno integral generalization pooling, as well as stochastic pooling and gated pooling.

	MNIST			FASHION			EMNIST			CIFAR10			CIFAR100		
	A1	A2	A3	A1	A2	A3	A1	A2	A3	A1	A2	A3	A1	A2	A3
CombPool (best)	99.34	99.43	99.49	93.21	92.60	**93.79**	87.52	89.09	**90.03**	**77.81**	88.61	89.87	46.46	56.08	70.31
AM	**99.37**	99.09	99.10	**93.24**	91.81	**93.79**	**87.58**	88.59	89.52	77.08	87.65	89.25	46.55	54.72	70.78
Max	99.24	99.39	99.34	92.80	**92.99**	92.92	87.31	89.11	89.27	77.39	87.85	87.99	45.68	57.28	68.48
$\mathbf{D}_\nu$	99.26	**99.46**	99.08	93.05	92.67	92.86	86.64	**89.27**	89.31	73.42	**88.70**	87.20	43.52	54.11	68.97
Stochastic	99.34	99.35	**99.53**	92.49	92.17	93.64	87.45	88.91	89.81	77.43	87.29	**91.57**	**47.77**	**58.18**	**70.68**
Gated	99.33	99.35	99.51	92.91	92.03	93.12	87.23	88.27	89.71	77.43	88.61	90.41	46.38	57.26	69.99

the novel ViT architecture without difficulty, although a more in-depth study is necessary in order to find the most suitable implementation for this model. We would like to explore the particular application of CombPool layers to ViT models in a future work.

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.

Acknowledgements

The authors gratefully acknowledge the financial support of Tracasa Instrumental (iTRACASA), Spain and of the Gobierno de Navarra - Departamento de Universidad, Innovación y Transformación Digital, Spain, as well as that of the Spanish Ministry of Science, Spain (project PID2019-108392GB-I00 (AEI/10.13039/501100011033)) and the project PC095-096 FUSIPROD. F. Herrera is supported by the Andalusian Excellence project, Spain P18-FR-4961. G.P. Dimuro is supported by CNPq, Brazil (301618/2019-4) and FAPERGS, Brazil (19/2551-0001279-9).

Appendix. Mathematical proofs

Proof of Proposition 3.1. Consider $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$.

If $(\alpha_1, \ldots, \alpha_r), (\beta_1, \ldots, \beta_r) \in \mathcal{I}(\mathbf{A}_1, \ldots, \mathbf{A}_r)$, we have

$$\begin{aligned}((\alpha_1+\beta_1)\mathbf{A}_1 + \cdots + (\alpha_r + \beta_r)\mathbf{A}_r)(\mathbf{x}) &= \\ &= (\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r)(\mathbf{x}) + (\beta_1\mathbf{A}_1 + \cdots + \beta_r\mathbf{A}_r)(\mathbf{x}) \\ &\leq (\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r)(\mathbf{y}) + (\beta_1\mathbf{A}_1 + \cdots + \beta_r\mathbf{A}_r)(\mathbf{y}) \\ &= ((\alpha_1 + \beta_1)\mathbf{A}_1 + \cdots + (\alpha_r + \beta_r)\mathbf{A}_r)(\mathbf{y})\end{aligned}$$

Consider $0 \leq \alpha_1, \ldots, \alpha_r \in \mathbb{R}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$; then $\mathbf{A}_i(\mathbf{x}) \leq \mathbf{A}_i(\mathbf{y})$, hence $\alpha_i\mathbf{A}_i(\mathbf{x}) \leq \alpha_i\mathbf{A}(\mathbf{y})$; so

$$\begin{aligned}&(\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r)(\mathbf{x}) = \alpha_1\mathbf{A}_1(\mathbf{x}) + \cdots + \alpha_r\mathbf{A}_r(\mathbf{x}) \leq \\ &\leq \alpha_1\mathbf{A}_1(\mathbf{y}) + \cdots + \alpha_r\mathbf{A}_r(\mathbf{y}) = (\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r)(\mathbf{y});\end{aligned}$$

thus $(\alpha_1, \ldots, \alpha_r) \in \mathcal{I}(\mathbf{A}_1, \ldots, \mathbf{A}_r)$. The rest is immediate. □

Lemma A.1. *If $\mathbf{A}_1, \ldots, \mathbf{A}_r : \mathbb{R}^n \to \mathbb{R}$ are functions such that $\mathbf{A}_i(\mathbf{0}) = 0$ and $\mathbf{A}_i(\mathbf{1}) = 1$, $i = 1, \ldots, r$, and $\alpha_1, \ldots, \alpha_r \in \mathbb{R}$ are such that $\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$ is increasing, then $\alpha_1 + \cdots + \alpha_r \geq 0$*

Proof. With $\mathbf{A} = \alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$, $0 = \mathbf{A}(\mathbf{0}) \leq \mathbf{A}(\mathbf{1}) = \alpha_1 + \cdots + \alpha_r$. □

Proposition A.2. *Given increasing functions $\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_r : \mathbb{R}^n \to \mathbb{R}$, for each $0 \leq \alpha_1, \ldots, \alpha_r \in \mathbb{R}$ there exists an increasing function $\mathbf{B} : \mathbb{R}^n \to \mathbb{R}$ such that $\mathbf{A} = \mathbf{B} - \alpha_1\mathbf{A}_1 - \cdots - \alpha_r\mathbf{A}_r$.* □

Proof. Let us define:

$$\mathbf{B} = \mathbf{A} + \alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$$

Since $1 + \alpha_1 + \cdots + \alpha_r > 0$, it follows that $\mathbf{B}$ is increasing. the result follows. □

Proof of Proposition 3.3. By hypothesis there exist $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ such that $\mathbf{A}(\mathbf{x}) < \mathbf{A}(\mathbf{y})$. Take $\mathbf{z} \in \mathbb{R}^n$ such that $\mathbf{z} < \mathbf{x}$ and $\mathbf{z} < \mathbf{y}$. Then we have that $\mathbf{z} < \mathbf{y}$ and $\mathbf{A}(\mathbf{z}) < \mathbf{A}(\mathbf{y})$. As $\alpha < 0$, then $\alpha\mathbf{A}(\mathbf{z}) > \alpha\mathbf{A}(\mathbf{y})$ and $\alpha\mathbf{A}$ is not increasing. □

Lemma A.3. *Consider $2 \leq r \in \mathbb{N}$. For $i = 1, \ldots, r$, let $\mathbf{A}_i : \mathbb{R}^n \to \mathbb{R}$ be increasing functions and $\alpha_i \in \mathbb{R}$ such that $\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$ is also an increasing function. If there exist $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} < \mathbf{y}$, such that*

$$\mathbf{A}_r(\mathbf{x}) < \mathbf{A}_r(\mathbf{y}) \text{ and } \mathbf{A}_i(\mathbf{x}) = \mathbf{A}_i(\mathbf{y}),\ i = 1, \ldots, r-1,$$

then $\alpha_r \geq 0$.

Proof. Let

$$\mathbf{A} = \alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$$

From the hypothesis of the Lemma, $\mathbf{A}$ is increasing. So:

$$0 \leq \mathbf{A}(y) - \mathbf{A}(x) = \alpha_r(\mathbf{A}_r(y) - \mathbf{A}_r(x))$$

Since $\mathbf{A}_r$ is also increasing, it follows that $\alpha_r \geq 0$. □

Lemma A.4. *Consider $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{R}^n$ such that $\mathbf{x} \leq \mathbf{y}$ and $\sigma, \rho \in \Sigma_n$ satisfying $\sigma \in \mathbf{x}_{(\nearrow)}$ and $\rho \in \mathbf{y}_{(\nearrow)}$. Then $\mathbf{x}_\sigma \leq \mathbf{y}_\rho$. In this situation we have further $\mathbf{x}_\sigma = \mathbf{y}_\rho$ if and only if $\mathbf{x} = \mathbf{y}$.*

Proof. Assume $x_i > x_{i+1}$; if $y_i > y_{i+1}$, with the transposition $\epsilon = (i\ i+1)$, that is, interchanging the elements in positions ith and $(i+1)$th in the considered tuple., we have $\mathbf{x}_\epsilon \leq \mathbf{y}_\epsilon$; if $y_i \leq y_{i+1}$, as $x_{i+1} < x_i \leq y_i$, we have $\mathbf{x}_\epsilon \leq \mathbf{y}$; iterating this process we obtain $\sigma, \rho \in \Sigma_n$ with $\mathbf{x}_\sigma \leq \mathbf{y}_\rho$ and $x_{\sigma(1)} \leq \cdots \leq x_{\sigma(n)}$. To ease the notation, assume that $x_1 \leq \cdots \leq x_n$ and $\mathbf{x} \leq \mathbf{y}$. Now, if $y_i > y_{i+1}$, we have $x_i \leq x_{i+1} \leq y_{i+1}$ and $x_{i+1} \leq y_{i+1} < y_i$, hence, with $\epsilon = (i\,i{+}1)$ again, we have $\mathbf{x} \leq \mathbf{y}_\epsilon$; reiterating the process we conclude that there exist $\sigma, \rho \in \Sigma_n$ such that $x_{\sigma(1)} \leq \cdots \leq x_{\sigma(n)}$, $y_{\rho(1)} \leq \cdots \leq y_{\rho(n)}$ and $\mathbf{x}_\sigma \leq \mathbf{y}_\rho$. Now, if $\sigma', \rho' \in \Sigma_n$ satisfy $x_{\sigma'(1)} \leq \cdots \leq x_{\sigma'(n)}$ and $y_{\rho'(1)} \leq \cdots \leq y_{\rho'(n)}$, we have $\mathbf{x}_\sigma = \mathbf{x}_{\sigma'}$ and $\mathbf{y}_\rho = \mathbf{y}_{\rho'}$.

Assume that $\mathbf{x}_\sigma = \mathbf{y}_\rho$ and set $a = \max(\mathbf{x})$; obviously $a = \max(\mathbf{y})$; assume that $x_{1_1}, \ldots, x_{i_r} = a$, $i_1 < \cdots < i_r \leq n$ and $x_j < a$ if $j \notin \{i_1, \ldots, i_r\}$; as $x_{i_k} \leq y_{i_k}$, we have $y_{1_1}, \ldots, y_{i_r} = a$ and $y_j < a$ if $j \notin \{i_1, \ldots, i_r\}$. Continue with $\max(N \setminus \{i_1, \ldots, i_r\})$. □

Proof of Proposition 3.5. With Proposition 3.1 it suffices to show that if $\alpha_1\mathbf{OS}_{i_1} + \cdots + \alpha_r\mathbf{OS}_{i_r}$ is increasing for $\alpha_1, \ldots, \alpha_r \in \mathbb{R}$, then $\alpha_1, \ldots, \alpha_r \geq 0$.

We may assume that $r > 1$. Let $j \in \{1, \ldots, r\}$ and consider i_j. Take $\mathbf{x} = (0, 1 \mid i_j + 1)$, $\mathbf{y} = (0, 1 \mid i_j)$ We have $\mathbf{OS}_{i_k}(\mathbf{x}) = \mathbf{OS}_{i_k}(\mathbf{y}) = 0$ if $k < j$ and $\mathbf{OS}_{i_k}(\mathbf{x}) = \mathbf{OS}_{i_k}(\mathbf{y}) = 1$ if $k > j$. On the other hand, $\mathbf{OS}_{i_j}(\mathbf{x}) = 0 < 1 = \mathbf{OS}_{i_j}(\mathbf{y})$. By Lemma A.3 we have $\alpha_j \geq 0$. □

Proof of Proposition 3.6. Assume that $\mathbf{A} := \alpha\mathbf{AM} + \beta_1\mathbf{OS}_{i_1} + \cdots + \beta_r\mathbf{OS}_{i_r}$ is increasing. Set $R = \{i_1, \ldots, i_r\}$. Then

$$\mathbf{A} = \frac{\alpha}{n}\sum_{i \in N \setminus R}\mathbf{OS}_i + \left(\frac{\alpha}{n} + \beta_j\right)\sum_{j=1}^{r}\mathbf{OS}_{i_j},$$

so by Proposition 3.5 $\mathbf{A}$ is increasing if and only if $\frac{\alpha}{n}, \frac{\alpha}{n} + \beta_1, \ldots, \frac{\alpha}{n} + \beta_r \geq 0$. □

Lemma A.5. *If $S \subseteq N$, then there exist $\sigma \in \Sigma_n$, $i \in N$ such that $N_i^\sigma = S$ (and so, for a fuzzy measure $\nu : 2^N \to [0, +\infty)$, $\nu_i^\sigma = \nu_S$.).*

Proof. If $S = N$, take $i = 1$ and σ the identity. Assume that $S \subset N$. Let $S = \{j_1, \ldots, j_r\}$, $|S| = r$, and $N \setminus S = \{k_1, \ldots, k_{n-r}\}$. Consider the permutation

$$\sigma = \begin{pmatrix} 1 & \ldots & n-r & n-r+1 & \ldots & n \\ k_1 & \ldots & k_{n-r} & j_1 & \ldots & j_r \end{pmatrix}.$$

Set $i = n - r + 1$. Then $N_i^\sigma = S$. □

Lemma A.6. *Consider* $i_1, \dots, i_r \in N$, $i_1 < \cdots < i_r$ *and* $\alpha_1, \dots, \alpha_r, \beta \in \mathbb{R}$. *If* $\alpha_1\mathbf{OS}_{i_1} + \cdots + \alpha_r\mathbf{OS}_{i_r} + \beta\mathbf{S}_\nu$ *is increasing, then* $\alpha_1, \dots, \alpha_r \geq 0$.

Proof. Let $j \in \{1, \dots, r\}$. Let $a, b \in \mathbb{R}$ such that $\nu_1 < a < b$ and take $\mathbf{x} = (a, b \mid i_j + 1)$, $\mathbf{y} = (a, b \mid i_j)$. Then $\mathbf{OS}_{i_s}(\mathbf{x}) = \mathbf{OS}_{i_s}(\mathbf{y})$ for all $s \in \{1, \dots \widehat{j} \dots, r\}$, $\mathbf{OS}_{i_j}(\mathbf{x}) = a < b = \mathbf{OS}_{i_j}(\mathbf{y})$ and $\mathbf{S}_\nu(\mathbf{x}) = \mathbf{S}_\nu(\mathbf{y}) = \nu_1$. By Lemma A.3, $\alpha_j \geq 0$. □

Lemma A.7. *If the fuzzy measure* $\nu : 2^N \to [0, +\infty)$ *is strict in* k *for some* $k \in N$, *then there exist* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} < \mathbf{y}$ *such that* $\mathbf{OS}_i(\mathbf{x}) = \mathbf{OS}_i(\mathbf{y})$ *for* $k \neq i \in N$, *and* $\mathbf{OS}_k(\mathbf{x}) = \mathbf{S}_\nu(\mathbf{x}) < \mathbf{OS}_k(\mathbf{y}) = \mathbf{S}_\nu(\mathbf{y})$.

Proof. Let $\sigma \in \Sigma_n$ such that $\nu^\sigma_k > \nu^\sigma_{k+1}$ (if $k = n$ take σ the identity on N, regardless of whether ν_n equals 0 or not). Let $a, b \in \mathbb{R}$, $\nu^\sigma_{k+1} \leq a < b \leq \nu^\sigma_k$ and take $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ in such a way that $\mathbf{x}_\sigma = (a, b \mid k + 1)$, $y_\sigma = (a, b \mid k)$. We have $\mathbf{OS}_i(\mathbf{x}) = \mathbf{OS}_i(\mathbf{y})$ if $i \neq k$ and $\mathbf{S}_\nu(\mathbf{x}) = \mathbf{OS}_k(\mathbf{x}) < \mathbf{OS}_k(\mathbf{y}) = \mathbf{S}_\nu(\mathbf{y})$. □

Proof of Proposition 3.8. By Lemma A.7 there exist $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} < \mathbf{y}$ such that $\mathbf{OS}_{i_j}(\mathbf{x}) = \mathbf{OS}_{i_j}(\mathbf{y})$, $j = 1, \dots, r$, and $\mathbf{S}_\nu(\mathbf{x}) < \mathbf{S}_\nu(\mathbf{y})$, hence if $\alpha_1\mathbf{OS}_{i_1} + \cdots + \alpha_r\mathbf{OS}_{i_r} + \beta\mathbf{S}_\nu$ is increasing, then $\beta \geq 0$ by Lemma A.3. Apply now Lemma A.6 and Proposition 3.1. □

Proof of Corollary 3.9. By Proposition 3.8, as ν is strict in n. □

Lemma A.8. *Consider* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$. *Set* $a_i = y_i - x_i$, $i = 1, \dots, n$. *Then, for all fuzzy measure* ν, $\mathbf{S}_\nu(\mathbf{y}) - \mathbf{S}_\nu(\mathbf{x}) \leq a_j$, *for some* $j \in \{1, \dots, n\}$.

Proof. Let $\emptyset \neq S \subseteq N$, $x_k = \min_{i\in S} x_i$ and $y_m = \min_{i\in S} y_i$. If $m = k$, then $(\min_{i\in S} y_i) - (\min_{i\in S} x_i) = y_k - x_k = a_k$; if $m \neq k$, $(\min_{i\in S} y_i) - (\min_{i\in S} x_i) = y_m - x_k \leq y_k - x_k = a_k$. Set $k(S) = k$. So we have immediately that $[\min(\nu_S, (\min_{i\in S} y_i))] - [\min(\nu_S, (\min_{i\in S} x_i))] \leq a_{k(S)}$.

Assume that $T, U \subseteq N$ are such that $\mathbf{S}_\nu(\mathbf{y}) = \min(\nu_T, (\min_{i\in T} y_i))$ and $\mathbf{S}_\nu(\mathbf{x}) = \min(\nu_U, (\min_{i\in U} x_i))$. Then $\min(\nu_T, (\min_{i\in T} x_i)) \leq \min(\nu_U, (\min_{i\in U} x_i))$ hence

$$\begin{aligned}\mathbf{S}_\nu(\mathbf{y}) - \mathbf{S}_\nu(\mathbf{x}) &= \min(\nu_T, (\min_{i\in T} y_i)) - \min(\nu_U, (\min_{i\in U} x_i)) \\ &\leq \min(\nu_T, (\min_{i\in T} y_i)) - \min(\nu_T, (\min_{i\in T} x_i)) \leq a_{k(T)}.\end{aligned}$$ □

Proof of Proposition 3.10. Set $\mathbf{A} = \alpha_1\mathbf{OS}_1 + \cdots + \alpha_n\mathbf{OS}_n + \beta\mathbf{S}_\nu$.

Assume that $\alpha_1, \dots, \alpha_n, \alpha_1 + \beta, \dots, \alpha_n + \beta \geq 0$. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$ and $\sigma \in \mathbf{x}_{(\nearrow)}$, $\rho \in \mathbf{y}_{(\nearrow)}$. Set $a_i = y_{\rho(i)} - x_{\sigma(i)}$, $1 \leq i \leq r$ (take in account Lemma A.4). Then

$$\mathbf{A}(\mathbf{y}) - \mathbf{A}(\mathbf{x}) = \alpha_1 a_1 + \cdots + \alpha_n a_n + \beta(\mathbf{S}_\nu(\mathbf{y}) - \mathbf{S}_\nu(\mathbf{x})).$$

Let us see that $\mathbf{A}(\mathbf{y}) - \mathbf{A}(\mathbf{x}) \geq 0$. If $\beta \geq 0$ it is clear. Assume that $\beta < 0$. By Lemma A.8 there exists $j \in N$ such that $\mathbf{S}_\nu(\mathbf{y}) - \mathbf{S}_\nu(\mathbf{x}) \leq a_j$, hence $\beta(\mathbf{S}_\nu(\mathbf{y}) - \mathbf{S}_\nu(\mathbf{x})) \geq \beta a_j$ and therefore

$$\mathbf{A}(\mathbf{y}) - \mathbf{A}(\mathbf{x}) \geq \alpha_1 a_1 + \cdots + (\alpha_j + \beta)a_j + \cdots + \alpha_n a_n \geq 0.$$

So $\mathbf{A}$ is increasing.

Assume now that this is the case. By Lemma A.6, $\alpha_1, \dots, \alpha_n \geq 0$.

Suppose that ν is strict in $k \in N$. By Lemma A.7, there exist $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} < \mathbf{y}$ such that $\mathbf{OS}_i(\mathbf{x}) = \mathbf{OS}_i(\mathbf{y})$, $i \neq k$, and $a := \mathbf{S}_\nu(\mathbf{x}) = \mathbf{OS}_k(\mathbf{x}) < b := \mathbf{S}_\nu(\mathbf{y}) = \mathbf{OS}_k(\mathbf{y})$. As $\mathbf{A}$ is increasing, $0 \leq \mathbf{A}(\mathbf{y}) - \mathbf{A}(\mathbf{x}) = (\alpha_k + \beta)(b - a)$. Thus $\alpha_k + \beta \geq 0$, $k = 1 \dots, n$. □

Proof of Proposition 3.11. Set $\mathbf{A} = \alpha\mathbf{AM} + \beta\mathbf{S}_\nu$; so $\mathbf{A} = \frac{\alpha}{n}(\mathbf{OS}_1 + \cdots + \mathbf{OS}_n) + \beta\mathbf{S}_\nu$, and the thesis follows from Proposition 3.10, as ν is strict in n. □

Lemma A.9. *Assume that* F *and* Σ *are admissible for the symmetrical fuzzy measure* ν *and let* $\mathbf{A} = \mathbf{A}(\mathrm{F}, \Sigma, \nu)$. *Assume that if* $x, y \in \mathbb{I}$, $x \leq y$ *and* $\mathrm{F}(x, \nu_i) = \mathrm{F}(y, \nu_i)$ *implies* $x = y$ *for* $i = 1 \dots, n$. *If* $\mathbf{x}, \mathbf{y} \in \mathbb{I}^n$ *and* $\mathbf{x} \leq \mathbf{y}$, *then* $\mathbf{A}(\mathbf{x}) = \mathbf{A}(\mathbf{y})$ *if and only if* $\mathbf{x} = \mathbf{y}$.

Proof. Let $\mathbf{x}, \mathbf{y} \in \mathbb{I}^n$ such that $\mathbf{x} \leq \mathbf{y}$ and $\mathbf{A}(\mathbf{x}) = \mathbf{A}(\mathbf{y})$. Then

$$0 = \mathbf{A}(\mathbf{y}) - \mathbf{A}(\mathbf{x}) = \sum_{i=1}^{n} (\mathrm{F}(y_{(i)}, \sigma_i) - \mathrm{F}(x_{(i)}, \sigma_i));$$

as by the hypothesis $\mathrm{F}(x_{(i)}, \sigma_i) \leq \mathrm{F}(y_{(i)}, \sigma_i)$, we have $\mathrm{F}(x_{(i)}, \sigma_i) = \mathrm{F}(y_{(i)}, \sigma_i)$, hence $x_{(i)} = y_{(i)}$, $1 \leq i \leq n$, and $\mathbf{x} = \mathbf{y}$. □

Proof of Proposition 3.16. By hypothesis $\nu_1 \geq \cdots \geq \nu_n > 0$, hence if $x, y \in \mathbb{R}$, $x\nu_i = y\nu_i$ implies $x = y$. Apply Lemma A.9. □

Proof of Proposition 3.17. We have

$$\begin{aligned}\sum_{i\in M} \alpha_i\mathbf{OS}_i + \beta\,\mathbf{D}_\nu &= \sum_{i\in M} \alpha_i\mathbf{OS}_i + \beta \sum_{i=1}^{n} \nu_i\mathbf{OS}_i \\ &= \sum_{i\in M} (\alpha_i + \beta\nu_i)\mathbf{OS}_i + \beta \sum_{i\in M'} \nu_i\mathbf{OS}_i;\end{aligned}$$

apply 3.5. □

Proof of Corollary 3.18. As $\mathbf{AM} = (\mathbf{OS}_1 + \cdots + \mathbf{OS}_n)/n$, by 3.17, $\alpha\mathbf{AM} + \beta\,\mathbf{D}_\nu$ is increasing if and only if

$$(\alpha/n) + \beta\nu_1 \geq 0, \dots, (\alpha/n) + \beta\nu_n \geq 0. \tag{1}$$

If $\beta \geq 0$, as $\nu_1 \geq \cdots \geq \nu_n$, then $\beta\nu_1 \geq \cdots \geq \beta\nu_n$, hence $(\alpha/n) + \beta\nu_1 \geq \cdots \geq (\alpha/n) + \beta\nu_n$, and (1) is equivalent to $(\alpha/n) + \beta\nu_n \geq 0$ or $\alpha + n\beta\nu_n \geq 0$.

If $\beta \leq 0$, then $\beta\nu_1 \leq \cdots \leq \beta\nu_n$ and conclude analogously. □

Proof of Proposition 3.19. We have $\alpha\,\mathbf{D}_\nu + \beta\,\mathbf{S}_\nu = \alpha\nu_1\,\mathbf{OS}_1 + \cdots + \alpha\nu_n\,\mathbf{OS}_n + \beta\,\mathbf{S}_\nu$. By 3.10, it is increasing if and only if $\alpha\nu_i, \alpha\nu_i + \beta \geq 0$ for $i = 1 \dots, n$. As ν is strict, $\nu_1 > 0$, hence in this case we have $\alpha \geq 0$; reciprocally, as $\nu_1 > \cdots > \nu_n$, if $\alpha\nu_n + \beta \geq 0$ and $\alpha \geq 0$, it follows the above condition. □

References

Abdel-Hamid, O., Deng, L., & Yu, D. (2013). Exploring convolutional neural network structures and optimization techniques for speech recognition. In *Interspeech*: *Vol. 11*, (pp. 73–75). Citeseer.

Abramsky, S., & Jung, A. (1994). Domain theory. In S. Abramsky, D. Gabbay, & T. Maimbaum (Eds.), *Handbook of logic in computer science*: *Vol. 3*, (pp. 1–168). Oxford Svience Publications.

Bardozzo, F., De La Osa, B., Horanská, L., Fumanal-Idocin, J., delli Priscoli, M., Troiano, L., et al. (2021). Sugeno integral generalization applied to improve adaptive image binarization. *Information Fusion*, *68*, 37–45. http://dx.doi.org/10.1016/j.inffus.2020.10.020.

Beliakov, G., Sola, H. B., & Sánchez, T. C. (2016). *A practical guide to averaging functions*. Springer.

Boureau, Y. -L., Bach, F., LeCun, Y., & Ponce, J. (2010). Learning mid-level features for recognition. In *2010 IEEE computer society conference on computer vision and pattern recognition* (pp. 2559–2566). IEEE.

Boureau, Y. -L., Ponce, J., & LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning* (pp. 111–118).

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165.

Bueno, J., Dias, C. A., Pereira Dimuro, G., Santos, H., & Bustince Sola, H. (2019). Aggregation functions based on the Choquet integral applied to image resizing. In *Proceedings of the 11th conference of the European society for fuzzy logic and technology*: *Vol. 1*, (pp. 460–466). Atlantis Press.

Choquet, G. (1953–1954). Theory of capacities, *Annales de l'Institut Fourier*, 5, 131–295.

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: Extending MNIST to handwritten letters. In *2017 international joint conference on neural networks* (pp. 2921–2926). http://dx.doi.org/10.1109/IJCNN.2017.7966217.

Devlin, J., Chang, M. -W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Dias, C. A., Bueno, J. C. S., Borges, E. N., Botelho, S. S. C., Dimuro, G. P., Lucca, G., et al. (2018). Using the Choquet integral in the pooling layer in deep learning networks. In G. A. Barreto, R. Coelho (Eds.), *Fuzzy information processing* (pp. 144–154). Cham: Springer International Publishing.

Dias, C., Bueno, J., Borges, E., Lucca, G., Santos, H., Dimuro, G., et al. (2019). Simulating the behaviour of Choquet-like (pre) aggregation functions for image resizing in the pooling layer of deep learning networks. In R. B. Kearfott, I. Batyrshin, M. Reformat, M. Ceberio, V. Kreinovich (Eds.), *Fuzzy techniques: Theory and applications* (pp. 224–236). Cham: Springer International Publishing.

Dimuro, G. P., Costa, A. C. R., & Claudio, D. M. (2000). A coherence space of rational intervals for a construction of IR. *Reliable Computing*, *6*, 139–178. http://dx.doi.org/10.1023/A:1009913122021.

Dimuro, G. P., Fernández, J., Bedregal, B., Mesiar, R., Sanz, J. A., Lucca, G., et al. (2020). The state-of-art of the generalizations of the Choquet integral: From aggregation and pre-aggregation to ordered directionally monotone functions. *Information Fusion*, *57*, 27–43.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2020). An image is worth 16 × 16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

Forcén, J., Pagola, M., Barrenechea, E., & Bustince, H. (2020). Learning ordered pooling weights in image classification. *Neurocomputing*, *411*, 45–53.

Fukushima, K., & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets* (pp. 267–285). Springer.

Graham, B. (2014). Fractional max-pooling. arXiv preprint arXiv:1412.6071.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *37*, 1904–1916.

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

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700–4708).

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd international conference on international conference on machine learning*: *Vol. 37*, (pp. 448–456). JMLR.org.

Krizhevsky, A., Hinton, G., et al. (2009). *Learning multiple layers of features from tiny images*. Citeseer.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Le Callet, P., Viard-Gaudin, C., & Barba, D. (2006). A convolutional neural network approach for objective video quality assessment. *IEEE Transactions on Neural Networks*, *17*, 1316–1327.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*, 2278–2324.

Lee, C., Gallagher, P., & Tu, Z. (2018). Generalizing pooling functions in CNNs: Mixed, gated, and tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*, 863–875. http://dx.doi.org/10.1109/TPAMI.2017.2703082.

Lee, C. -Y., Xie, S., Gallagher, P., Zhang, Z., & Tu, Z. (2015). Deeply-supervised nets. In *Artificial intelligence and statistics* (pp. 562–570).

Lin, M., Chen, Q., & Yan, S. (2014). Network in network. CoRR, arXiv:1312.4400.

Liu, S., & Deng, W. (2015). Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian conference on pattern recognition* (pp. 730–734).

Lucca, G., Sanz, J. A., Dimuro, G. P., Borges, E. N., Santos, H., & Bustince, H. (2019). Analyzing the performance of different fuzzy measures with generalizations of the Choquet integral in classification problems. In *2019 IEEE international conference on fuzzy systems* (pp. 1–6). IEEE.

Mendoza, O., Melin, P., & Licea, G. (2009). A hybrid approach for image recognition combining type-2 fuzzy logic, modular neural networks and the sugeno integral. *Information Sciences*, *179*, 2078–2101.

Pan, Z., Zhuang, B., Liu, J., He, H., & Cai, J. (2021). Scalable vision transformers with hierarchical pooling. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 377–386).

Radosavovic, I., Kosaraju, R. P., Girshick, R., He, K., & Dollár, P. (2020). Designing network design spaces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 10428–10436).

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks* (pp. 92–101). Springer.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*. URL http://lmb.informatik.uni-freiburg.de/Publications/2015/DB15a.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958.

Stoltenberg-Hansen, A., Lindström, I., & Griffor, E. B. (1994). *Cambridge tracts in theoretical computer science*: *Vol. 22*, *Mathematical theory of domains*. Cambridge: Cambridge university Press.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9).

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105–6114). PMLR.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), *Advances in neural information processing systems*: *Vol. 30*, Curran Associates, Inc..

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.

Zeiler, M. D., & Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. arXiv preprint arXiv:1301.3557.