# Heuristics to reduce linear combinations of activation functions to improve image classification

Rogério Ferreira de Moraes*, Raphael dos S. Evangelista*, Andre Luiz da S. Pereira*, Yanexis Pupo Toledo*, Leandro A. F. Fernandes*, Luis Martí†
* Universidade Federal Fluminense (UFF), Niterói, Brazil
† Inria Chile Research Center, Las Condes, Chile
Email: {rogeriofm, revangelista, andreluizpereira, ypupo}@id.uff.br, laffernandes@ic.uff.br, lmarti@inria.cl

*Abstract*—**Image classification is one of the classical problems in computer vision, and CNNs (Convolutional Neural Networks) are widely used for this task. However, the choice of a CNN can vary depending on the chosen dataset. In this context, we have trainable activation functions that are crucial in CNNs and adapt to the data. One technique for constructing these functions is to write them as a linear combination of other activation functions, where the coefficients of this combination are learned during training. However, if we have a large number of activation functions to combine, the computational cost can be very high, and manually testing and choosing these functions may be impractical, depending on the number of available activation functions. To alleviate the difficulty of choosing which activation functions should be part of the linear combination, we propose two heuristics: Linear Combination Approximator by Coefficients (LCAC) and Major and Uniform Coefficient Extractor (MUCE). Our heuristics provide an efficient selection of a subset of activation functions so that their results are better or equivalent to the linear combination that uses all 34 available activation functions in our experiments (C34), considering the image classification problem. Compared to the C34 function, the LCAC function was better or equivalent in 62.5%, and the MUCE function in 87.5% of the conducted experiments.**

## I. Introduction

Image classification has a wide range of applications, including medical image analysis [1], remote sensing [2], and cosmology [3]. There is extensive research focused on improving CNNs for image classification [4], [5].

There are numerous types of research to improve these CNNs to make them more effective, and the activation functions are studied in one of these branches [6]–[8]. Activation functions have a fundamental role because, according to the choice of this function, we can improve the network performance [9]. Among the activation functions, the fixed functions do not change during the learning process, and the learned functions are built in the learning process phase of the network [10]. Since 2017, there has been a significant increase in the number of papers related to learned activation function [10]. From the techniques developed for trainable activation functions, we highlight the parameterized activation functions and functions based on ensemble methods. Parameterized activation functions are functions derived from standard activation functions with the addition of some parameters to be learned. Activation functions based on ensemble methods are a mixture of several different activation functions, where these functions are combined using specific parameters that will be learned. A common approach is to linearly combine these functions, learning the coefficients during training [9], [10], which is the focus of this work.

Although linear combinations of activation functions allow learning their coefficients in the network learning phase, they may not be satisfactory. It is because we can have different results according to the chosen set of activation functions. Also, manually choosing which function remains may be unfeasible because if we think of using $n$ different activation functions, we will have $2^n - 1$ possible combinations, which overgrow as $n$ grows. Another problem to be addressed is that the more activation functions we use, the greater the time spent on training, and the more memory will be required. To tackle this problem, we propose two heuristics, which help to choose functions from the set of activation functions available, so that the result is equivalent to or better than choosing to use all functions in the linear combination.

In our experiments on the image classification problem, we use 25 activation functions implemented by the PyTorch framework and 9 recent activation functions from literature, resulting in the 34 activation functions. Hence, the function named C34 stands for the linear combination of these 34 activation functions. In this paper, we try to choose a subset of the considered function by applying the proposed heuristics: Linear Combination Approximator by Coefficients (LCAC) and Major and Uniform Coefficient Extractor (MUCE). With those heuristics, we reduce the number of activation functions, leading to equivalent or better results than combining all the available activation functions using the C34 function. For our experiments, we chose the datasets MNIST, KMNIST, FashionMNIST, SVHN, CIFAR10, and CIFAR100. For the neural network architectures, we chose the ResNet-18 as the simplest, MobileNet, SuffleNet as networks proposed to be used in mobile, and the ResNet-50 network, which is a more robust network. The obtained results demonstrate that the heuristics LCAC and MUCE manage to be better or equivalent to C34, at 62.5% and 87.5%, respectively, by analyzing the Table I. If we disregard the MobileNet network, the MUCE heuristic was better or equivalent in all cases.

## II. Related work

There has been a growing interest in research on activation functions. Among the works, we can mention research on fixed activation functions, and a greater tendency towards trainable activation functions. Among the authors that proposed fixed activation functions, we can mention Elfwing *et al.* [11], who proposed SILU and dSiLU. Alcaide  [12] proposed a multiple of the SILU function, where the multiplicity factor was tuned according to the problem. Chieng *et al.* [13] proposed the function Flatten-T Swish, whose feature is to work with a function that merges ReLU and Sigmoid, using a parameter $T$. For $T = 0$, the function is a product of ReLU and Sigmoid. Qin *et al.* [14] proposed a function derived from Sigmoid. Roy *et al.* [15] proposed the LiSHT function. Farzad *et al.* [16] proposed the Elliott function, which is similar to the Sigmoid.

As for the trainable functions, we have the parameterized functions and the functions built as an ensemble of functions. Among the parameterized functions, we can mention the PReLu [17], which is equal to the ReLU function when the input of the function is positive, and it has an added weight to be learned by the network when the input is negative. The Flexible ReLU function, proposed by Qiu *et al.* [18], adds a parameter to the domain and another to the image of the ReLU function. Clevert *et al.* [19] proposed the ELU, which is a combination of ReLU when the input is positive, and in the negative part, a parameter $\alpha$ to be learned is introduced. The image of this function varies between $]-\alpha, +\infty[$. Other architectures similar to ELU were proposed, such as SELU [20], PELU [21], CELU [22], MPELU [23], FELU [24], PREU [25], EELU [26], PDELU [27].

From the works that use an ensemble of activation functions, we can mention [28] that proposed three ways to combine functions. The first used a linear combination between the ELU and LReLU functions, the second used product and sum with Sigmoid, LReLU, and ELU, and the third one was building a network using the PReLU, PELU, and Sigmoid functions. In [29], activation functions are learned in a sub-network with a single activation layer defined as

$$VAF(a_i) = \sum_{j=1}^{k} b_j g(\alpha_j a_i + \alpha_{0j}) + \beta_0,$$

where $a_i$ is the input and $\alpha_j, \alpha_{0j}, \beta_j$ and $\beta_0$ are the parameters learned from the data during the training process and $g$ is an activation function. The activation functions proposed by Scardapane *et al.* [30] are based on the linear combination of the kernel functions. Sütfeld *et al.* [31] proposed a function formed by the linear combination of the functions Tanh, ELU, ReLU, Id, and Swish, where the coefficients are learned. In [32], the function is constructed as

$$h(x) = \sum_{S=1}^{(S+1)/2} a_+^S \max(0, x-b^S) + \sum_{S=1}^{(S+1)/2} a_-^S \max(0, -x-b^s),$$

where $b^S \geq 0$ is fixed during training and $a_+^S, a_-^S$ are trainable parameters. Note that for $x \geq b^S$, we have a sum of ReLU functions translated from $b^S$.

## III. Methodology

An activation function derived from the linear combination of other activation functions can be expressed as:

$$f(x) = \sum_{i=1}^{n} a_i f_i(x), \tag{1}$$

where $a_i \in \Re$ the coefficients that must be learned in the network training phase, and $f_i$ are fixed or learned activation functions, for $i \in \{1, 2, \cdots, n\}$. Notice that $a_i = 0$ has the same effect as removing the corresponding activation function $f_i$ from the linear combination. For this, it is necessary to have a bijection between the $a_i$'s coefficients and the $f_i$'s activation functions. An arbitrary ordering of how the $f_i$'s activation functions will appear in the linear combination is chosen to promote bijection. Once chosen, this ordering is maintained.

After training, we will have the activation function $f$ generated by the $f_i$ activation functions. Our objective with the two proposed heuristics is to find a proper subset of activation functions $\{g_t, 1 \leq t \leq m, m \leq n\} \subset \{f_i, 1 \leq i \leq n\}$ in such a way that it significantly reduces the number of activation functions placed in the linear combination, while the accuracy of the network using this new activation function $g(x) = \sum_{t=1}^{m} a_t g_t(x)$ can lead to results equivalent to or better than the network using $f(x)$.

### A. Model evaluation

We used the Stratified $k$-Fold Cross Validation [33] to evaluate the accuracy of the learned activation function, which divides the data into $k$ disjoint parts (a.k.a. folds) while preserving the proportion of samples of each class in the folds. Then, $k$ iterations are made so that, at each iteration, one of the $k$ folds of samples is chosen as the validation set, and the remaining $k-1$ folds are set as the training set. Given a subset of activation functions $\{g_t, 1 \leq t \leq m, m \leq n\} \subset \{f_i, 1 \leq i \leq n\}$, for each iteration $l, 1 \leq l \leq k$ we will have a learned function $f^l(x) = \sum_{t=1}^{m} a_t g_t(x)$. To evaluate the accuracy of the set of learned $f^l$ functions, we calculate the accuracy of each of these functions concerning the validation set. The average accuracy estimates how much the model can predict using the learned activation function. We use Welch's t-test [34] to compare how much one result obtained with the average accuracy is better than another. The null hypothesis of Welch's t-test is that the averages are statistically equal.

### B. Initial foundations for building heuristics

At each iteration $l, 1 \leq l \leq k$, promoted by Stratified $k$-Fold Cross Validation, we must choose a subset of functions

$$G_l = \{g_t, 1 \leq t \leq m_l, m_l < n\} \subset \{f_i, 1 \leq i \leq n\}. \tag{2}$$

Note that if $G_l$ were equal in each iteration, the activation function chosen by the technique would be

$$f = \sum_{i=1}^{m} x_i g_i, \; gi \in G_1 = \ldots = G_k, \tag{3}$$

where $x_i$ would be the learned coefficients. This would prevent us from having distinct linear combinations in the iterations performed. However, this does not happen. We define a threshold from a $P$ function to work around the problem.

**Definition 1** ($P$ function)**.** *Let $k$ be the number of partitions applied to the data using the Stratified $k$-Fold Cross Validation technique. For $1 \leq l \leq k$ consider collection of subsets*

$$G_l = \{g_t, 1 \leq t \leq m_l, m_l \leq n\} \subset \{f_i, 1 \leq i \leq n\}, \quad (4)$$

*and let $v^i_{max} = \max\{v \in \{1, \dots, k\}, f_i \in \bigcap_{j=1}^{v} G_{l_j}\}$. Then the $P$ function is defined as:*

$$P : \{f_i, 1 \leq i \leq n\} \to \Re, \quad P(f_i) = \frac{1}{k} v^i_{max}. \quad (5)$$

**Definition 2** (Intersection threshold)**.** *If we construct a subset*

$$G = \{g_t, 1 \leq t \leq m, m \leq n\} \subset \{f_i, 1 \leq i \leq n\} \quad (6)$$

*from the collection of subsets*

$$G_l = \{g_t, 1 \leq t \leq m_l, m_l \leq n\} \subset \{f_i, 1 \leq i \leq n\}, \quad (7)$$

*with intersection threshold $p \in [0, 1]$, and $1 \leq l \leq k$, it means $P(f_i) \geq p \ \forall f_i \in G$.*

The matrices $B$ and $C$ will be used in the LCAC and MUCE techniques, respectively, for choosing the activation functions that will compose the linear combination of the function proposed by the technique. For that, consider

$$B = [b_{ij}]_{k \times n}, \quad (8)$$

matrix of the activation function coefficients learned, where each row $l$ of the matrix was learned in the $l - th$ iteration using the Stratified $k$-Fold Cross Validation. In order to apply the MUCE technique, it is essential to normalize the columns of matrix $B$, which we will explain in Section III-B2. In turn, matrix $C$ is constructed from matrix $B$ by normalizing its columns using the Softmax function:

$$C = [c_{ij}]_{k \times n}, \text{ where } c_{ij} = \frac{e^{b_{ij}}}{\sum_{p=1}^{k} e^{b_{pj}}} \text{ and } b_{ij} \in B. \quad (9)$$

*1) Linear Combination Approximator by Coefficients (LCAC):* The LCAC heuristics consists of choosing the coefficients of the linear combination of activation functions that are closest to zero to then eliminate them and keep their complement. For this, consider the matrix $B$ given by Equation 8. Let $D$ be the matrix obtained by normalizing the lines of $B$ using the Softmax function. Consider

$$D_l = \{d_{lj} \in D, 1 \leq j \leq n\}, \quad 1 \leq l \leq k. \quad (10)$$

We choose a elimination threshold $T$ such that for each $D_l$, we must eliminate the activation functions corresponding to the $d_{lj_1}, \dots, d_{lj_m} \in D_l$ coefficients, which satisfy the $d_{lj_1} + \ldots + d_{lj_m} \leq T$ inequality. Thus, we will obtain the set $G_l$ given by Equation 2 for each iteration $l, \quad 1 \leq l \leq k$. With the sets $G_1, \dots, G_k$, we will choose an intersection threshold $p$ according to Definition 2 to decide which activation functions will be part of the function proposed by this technique.

It is worth noting that the smaller the value of the $T$ elimination threshold, the closer to zero the coefficients of the activation functions that will be eliminated.

The idea behind LCAC is that functions with very low coefficients contribute less to the sum of the linear combination and therefore can be removed. Of course, this is not always true, as even if the coefficient is very low, the activation function can have very high values, compensating for the low coefficient. However, if the activation functions were bounded, $|f_i| \leq M_i \in \Re^+$, such as the sigmoid function, we would have that guarantee. But this would limit the technique, removing important functions like ReLU.

*2) Major and Uniform Coefficient Extractor (MUCE):* The MUCE heuristic chooses activation functions $f_i, 1 \leq i \leq n$, which have higher coefficients and less variation about $k$ interactions done in training using Stratified $k$-Fold Cross Validation. This implies that the columns of matrix C (see Equation 9) that have coefficients corresponding to these functions will have larger, more uniform values.

The process performed by this method is similar to the one described in Section III-B1, with the difference that we use matrix $C$ to find the subsets of activation functions $G_1, \dots, G_k$. According to the elimination threshold $T$ and the intersection threshold $p$ (see Definition 2), activation functions that have a large variation in their coefficients over the $k$ iterations will be discarded. A fundamental factor for this is that the columns of $C$ are normalized. To evidence this fact, consider an activation function $f_i$. Suppose that over the $k$ iterations, its learned coefficients have $m$ values close to zero with $m < k$. This implies that the $i$-th column of $C$ has $m$ values close to zero. As matrix $C$ has columns normalized by the Softmax function, it implies that the sum of its columns equals 1. Furthermore, as the sum of the columns always gives the same value, the closer the $m$ coefficients are to zero, the greater the sum of the remaining $k - m$ coefficients, which increases the variation of this column between the smallest and largest values. Let $T$ be an elimination threshold. Then, for every row

$$C_l = \{c_{lj} \in C, 1 \leq j \leq n\}, 1 \leq l \leq k \quad (11)$$

that has $c_{lt_1}, \dots, c_{lt_u} \in C_l$, we eliminate the elements that satisfy the inequality

$$c_{lt_1} + \ldots + c_{lt_u} \leq T. \quad (12)$$

If the $m$ coefficients are small enough, we can guarantee that each of these $m$ coefficients is part of the Inequality 12 concerning their respective rows of matrix $C$. Thus, we will eliminate the $m$ coefficients, which means that $f_i \in \bigcap_{j=1}^{m} G_j$, that is, $P(f_i) = \frac{m}{k}$ (see Definition 1). Let $p$ be an intersection threshold (see Definition 2). Then $\frac{m}{k} = P(f_i) \geq p$, with $p \in [0, 1]$. We conclude that the greater the value of $p$, the greater the value of $m$. Finally, note that we assumed that the $m$ coefficients are small enough that each one forms part of the

Inequality 12 in its corresponding row of the $C$ matrix. However, the smaller the $m$ coefficients, the major the difference between the $m$ coefficients and the remaining $k-m$ elements. Suppose none of the $m$ coefficients are small enough to be part of Inequality 12. It implies that the difference between the $m$ coefficients and the remaining $k-m$ coefficients is smaller and thus more uniform, making $P(f_i) = \frac{0}{k} = 0$. This way, we guarantee that $f_i$ is part of the linear combination proposed by the technique. One last observation is that we always remove the coefficients closest to zero, leaving the largest ones.

## IV. Datasets, architectures and activation functions

Our analysis was based on 6 well-known datasets and 4 simple but robust architectures. Also, we used 34 activation functions, which provided a wide range of situations.

### A. Datasets

The datasets we used in the experiment were: The **CIFAR-100** [35] dataset contains 100 classes with 60000 $32 \times 32$ color images, with 600 images per class. There are 500 training images and 100 test images per class. The **CIFAR-10** [35] dataset contains ten classes with 60000 $32 \times 32$ color images, with 6000 images per class. There are 5000 training images and 1000 test images per class. The **SVHN** [36] dataset contains 10 classes with 630420 $32 \times 32$ color images. There are 73257 training images, 26032 test images, and 531131 additional images. SVHN is obtained from house numbers in Google Street View images. The **MNIST** [37] dataset contains 10 classes with 70000 $28 \times 28$ grayscale images. There are 60000 training images and 10000 test images. MNIST is a set of handwritten images of numbers from zero to nine. The **Kuzushiji-49** [38] **(KMNIST)** dataset contains 49 classes with 270912 $28 \times 28$ grayscale images, where the amount per class is non-proportional. The division between training and testing is $^6/_7$ for training and $^1/_7$ for testing for each class. The **Fashion-MNIST** [39] dataset contains 10 classes with 70000 $28 \times 28$ grayscale images. There are 60000 training images and 10000 test images.

It is worth mentioning that the CIFAR-10 and CIFAR-100 datasets, being similar, their classes are disjoint. These datasets are the most challenging, and the CIFAR-100 has greater difficulty performing the training. This happens because the amount of images per class is smaller. With the MNIST dataset, architectures achieve greater accuracy.

### B. Architectures

Looking for variety in network architectures, we took the simplest, most used, and mobile-oriented ones: **ResNet-18** [40] is one of the simplest in the ResNets family. **ResNet-50** [40] is of medium depth compared to ResNet family networks. It is a popular network. **ShuffleNetV2** [41] is a low-cost architecture with a good training speed. **MobileNetV3** [42] is a low-cost computing network, more mobile-oriented.

### C. Activation functions

In this work, we seek to include a variety of activation functions. We chose to choose fixed activation functions as we wanted to avoid having to train more parameters. We take 25 fixed activation functions fixed implemented by PyTorch, what are the functions ReLU [43], Hardtanh [44], Hard-Swish [42], ReLU6 [45], ELU [19], SELU [20], CELU [22], Leaky ReLU [46], PReLU [17], RReLU [43], Sigmoid, Log Sigmoid [47], Hardshrink, Tanhshrink, Softsign [48], Softplus [49], Softmin [50], Softmax [51], Softshrink, Log-Softmax, Tanh [52], Sigmoid, Hardsigmoid [53], Gelu [54], SiLU [11] and Mish [55]. We also added 9 more fixed activation functions what are Bipolar Sigmoid [56], dSiLU [43], Flatten-T Swish [13], Elliott [16], LiSHT [15], ReSech [57], sSigmoid [58], ELiSH [59] and HardELish [59]. It is worth noting that some activation functions are similar, but we avoid removing them to prevent bias.

## V. Experimental analysis

We implemented the LCAC and MUCE heuristics using Python 3.7.13, PyTorch 1.12.0+cu113, PyTorch Lightning 1.7.1, TorchMetrics 0.9.3, and Lmdb 0.99. We used ten Google Colab accounts, with each account running two experiments. It took us around 60 days to carry out all the experiments, including the hyperparameter's sweep.

### A. General settings

In our experiment, the test subsets were not used. We split the training part of the datasets using the Stratified $k$-Fold Cross Validation with $k = 10$. We described in Section III-A how we obtained the accuracies of each activation function proposed by the techniques and how we compare how one activation function is statistically better or equivalent. In Welch's t-test, if the p-value is less than 0.05, the null hypothesis that the results are equivalent is rejected. The intersection threshold $p$ (see Definition 2) used in the LCAC and MUCE heuristics was set to $p = 0.9$. The elimination threshold $T$ (see Subsection III-B1) for LCAC was set to $T = 0.1$. For MUCE, we set the elimination threshold $T = 0.7$, except mobile architecture using the CIFAR-100 dataset, for which we use $T = 0.999$ due to vanishing or exploding gradients.

There were a few cases in which the networks suffered of exploding or vanishing gradients. So we had to discard some learning rate values, which reached reasonable accuracy rates at the beginning of the training, but which did not manage to reach the end. Another important observation is that the results could be better by choosing $p$ and $T$ thresholds for each combination of architecture and dataset. However, due to a lack of computational power, we chose the same thresholds for all cases, except for the ones cited below.

### B. Training details

The supplementary material presents the intervals used to perform hyperparameter sweep. The choice of the interval for sweeping the learning rate was empirical by observing the behavior of each network architecture. After choosing each range, a scan of all possibilities was performed using as stopping criterion Hyperband [60] with $min_{iter} = 3$ and $\eta = 3$. The supplementary material also presents the final set of hyperparameter values used in our experiments. We fixed the batch size for the data at 50 due to a lack of memory when using the activation function formed by the combination of the 34 activation functions available. The dataset was divided into ${}^{k-1}/_{k}$ for training and ${}^{1}/_{k}$ for validation. A scheduler strategy updated the learning rate by reducing it by 0.1 when the validation accuracy has stopped improving by five epochs. We used early stopping as stopping to monitor the validation accuracy and interrupted training in case it did not increase after ten epochs. We saved the last model.

### C. Results

We performed three analyses to compare the efficiency of the activation function obtained by the heuristics LCAC and MUCE concerning the activation function obtained by the linear combination of the 34 activation functions (C34). In the first half of Table I, we see the analysis of comparing C34, LCAC, and MUCE with the original activation function and seeing how much C34, LCAC, and MUCE are better or equivalent. This table highlighted in bold when the technique was better or when there was a tie Let's observe in this case that a tie favors the original activation function because it is less costly in terms of memory usage and processing time. So, in the comparison of the original function with C34, the original was better or equivalent 12 times, and C34 was better 12 times, which gives 50% each. Performing the same analysis with the original, LCAC, and MUCE, we find that in both cases, the original was better or equivalent in 37.5% of the cases, while LCAC and MUCE were better in 62.5%.

In the second half of Table I, the comparison between C34, LCAC, and MUCE takes place. The tie favors LCAC and MUCE as they consume less memory and processing time. Analyzing C34 and LCAC, C34 outperformed in 37.5% of the cases, while LCAC was better or equivalent in 62.5% of the cases. Now, comparing C34 and MUCE, C34 had better performance in 12.5% of the cases, while MUCE was superior or equivalent in 87.5% of the cases.

We set the same limit for the number of functions in the linear combination due to computational costs, but it is possible to achieve better results by adjusting this limit for each network and dataset. It is important to highlight that MobileNet encountered issues of gradient explosion and vanishing in some cases, affecting the choice of the learning rate. By excluding this network from the analysis, the MUCE heuristic was better or equivalent in all cases.

TABLE I
ACCURACIES OF CNNS USING THE ORIGINAL, C34, LCAC, AND MUCE ACTIVATION FUNCTION. BOLD HIGHLIGHTS BETTER RESULTS OR A TIE.

| **CNN/Dataset** | **ResNet-18** | | **ResNet-50** | | **ShuffleNet** | | **MobileNet** | |
|---|---|---|---|---|---|---|---|---|
| – | Orig | C34 | Orig | C34 | Orig | C34 | Orig | C34 |
| **Mnist** | **.993** | **.994** | .989 | **.994** | .990 | **.992** | **.992** | **.992** |
| **KMnist** | .990 | **.993** | **.990** | **.989** | .985 | **.988** | **.988** | **.987** |
| **FashionMnist** | **.928** | .925 | **.926** | **.926** | **.916** | **.919** | **.921** | .879 |
| **SVHN** | **.929** | **.931** | **.930** | .926 | .881 | **.911** | .690 | **.917** |
| **CIFAR-100** | **.443** | **.438** | .413 | **.455** | .356 | **.400** | .305 | **.399** |
| **CIFAR-10** | .747 | **.777** | **.778** | .767 | .618 | **.734** | .657 | **.736** |
| – | Orig | LCAC | Orig | LCAC | Orig | LCAC | Orig | LCAC |
| **Mnist** | .993 | **.994** | .989 | **.994** | .990 | **.992** | **.992** | **.992** |
| **KMnist** | .990 | **.993** | .990 | **.992** | .985 | **.987** | **.988** | .956 |
| **FashionMnist** | **.928** | **.927** | **.926** | .922 | **.916** | **.917** | **.921** | **.920** |
| **SVHN** | **.929** | .915 | .930 | **.936** | .881 | **.902** | .690 | **.839** |
| **CIFAR-100** | **.443** | **.455** | .413 | **.447** | .356 | **.391** | .305 | **.379** |
| **CIFAR-10** | .747 | **.776** | **.778** | .753 | .618 | **.706** | .657 | **.745** |
| – | Orig | MUCE | Orig | MUCE | Orig | MUCE | Orig | MUCE |
| **Mnist** | .993 | **.994** | .989 | **.993** | .990 | **.992** | **.992** | .981 |
| **KMnist** | .990 | **.992** | .990 | **.992** | .985 | **.989** | **.988** | **.988** |
| **FashionMnist** | **.928** | **.928** | **.926** | **.927** | **.916** | **.918** | **.921** | **.924** |
| **SVHN** | .929 | **.934** | **.930** | **.929** | .881 | **.912** | .690 | **.918** |
| **CIFAR-100** | .443 | **.475** | .413 | **.464** | .356 | **.411** | **.305** | **.312** |
| **CIFAR-10** | .747 | **.787** | **.778** | **.779** | .618 | **.740** | .657 | **.726** |
| – | C34 | LCAC | C34 | LCAC | C34 | LCAC | C34 | LCAC |
| **Mnist** | **.994** | **.994** | **.994** | **.994** | **.992** | **.992** | **.992** | **.992** |
| **KMnist** | **.993** | **.993** | .989 | **.992** | **.988** | **.987** | **.987** | .956 |
| **FashionMnist** | **.925** | **.927** | **.926** | .922 | **.919** | **.917** | .879 | **.920** |
| **SVHN** | **.931** | .915 | .926 | **.936** | **.911** | .902 | **.917** | .839 |
| **CIFAR-100** | **.438** | **.455** | **.455** | .447 | **.400** | **.391** | **.399** | .379 |
| **CIFAR-10** | **.777** | **.776** | **.767** | .753 | **.734** | .706 | .736 | **.745** |
| – | C34 | MUCE | C34 | MUCE | C34 | MUCE | C34 | MUCE |
| **Mnist** | **.994** | **.994** | **.994** | **.993** | **.992** | **.992** | **.992** | .981 |
| **KMnist** | **.993** | **.992** | .989 | **.992** | **.988** | **.989** | .987 | **.988** |
| **FashionMnist** | .925 | **.928** | **.926** | **.927** | **.919** | **.918** | .879 | **.924** |
| **SVHN** | .931 | **.934** | .926 | **.929** | **.911** | **.912** | **.917** | **.918** |
| **CIFAR-100** | .438 | **.475** | **.455** | **.464** | .400 | **.411** | **.399** | .312 |
| **CIFAR-10** | .777 | **.787** | .767 | **.779** | .734 | **.740** | **.736** | .726 |

## VI. CONCLUSIONS AND FUTURE WORK

In this paper, we broach the technique of learned activation functions, and more specifically, activation functions constructed by the linear combination of other functions, where the network learns the coefficients of this combination. We approach the problem of reducing the number of activation functions using the LCAC and MUCE heuristics. In this paper, we use 34 activation functions and name the activation function resulting from combining these functions C34. This work was directed to classification problems. The LCAC and MUCE techniques had better results compared to C34, in particular the MUCE heuristics, which in 87.5% of the cases was better or equivalent to C34.

The advantages of the proposed heuristics are the reduction of the number of activation functions, resulting in a statistically better or equivalent resulting function. Additionally, it is a simple implementation where the choice of the number of activation functions is a hyperparameter. The disadvantage of our heuristic is that it requires the use of stratified $k$-fold cross-validation, which can lead to longer execution times depending on the value of $k$ and the size of the dataset.

Due to our lack of computing power, we do not test for larger datasets like ImageNet. Also, we only implemented linear combinations, but we might consider seeing its behavior with other ensemble structures. We would also like to analyze the behavior of the proposed heuristics in other scenarios, such as image detection, recognition, and segmentation.

## Acknowledgment

ANID International Centers of Excellence Program 10CEII-9157/CTI220002 Inria Chile, Inria Challenge OcéanIA, STI-CAmSud EMISTRAL 21-STIC-08, CLIMATAmSud GreenAI 21-CLIMAT-07, and Inria associated team SusAIn.

## References

[1] G. Litjens *et al.*, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017.

[2] Y. Li *et al.*, "Deep learning for remote sensing image classification: A survey," *WIREs Data Mining Knowl Discov*, vol. 8, p. e1264, 2018.

[3] N. Hambly *et al.*, "The SuperCOSMOS Sky Survey — II. Image detection, parametrization, classification and photometry," *MNRAS*, vol. 326, pp. 1295–1314, 2001.

[4] S. S. Nath *et al.*, "A survey of image classification methods and techniques," in *ICCICCT*, 2014, pp. 554–557.

[5] W. Wei *et al.*, "Development of convolutional neural network and its application in image classification: a survey," *Optical Engineering*, vol. 58, p. 040901, 2019.

[6] S. Kiranyaz *et al.*, "1D convolutional neural networks and applications: A survey," *MSSP*, vol. 151, p. 107398, 2021.

[7] Z. Li *et al.*, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," *TNNLS*, vol. 33, pp. 6999–7019, 2022.

[8] J. *et al.* G., "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018.

[9] S. R. Dubey *et al.*, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, 2022.

[10] A. Apicella *et al.*, "A survey on modern trainable activation functions," *Neural Netw.*, vol. 138, pp. 14–32, 2021.

[11] S. Elfwing *et al.*, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Netw.*, vol. 107, pp. 3–11, 2018.

[12] E. Alcaide, "E-swish: Adjusting Activations to Different Network Depths," arXiv:1801.07145, 2018.

[13] H. H. Chieng *et al.*, "Flatten-T Swish: a thresholded ReLU-Swish-like activation function for deep learning," *IJAIN*, vol. 4, pp. 76–86, 2018.

[14] Y. Qin *et al.*, "The optimized deep belief networks with improved logistic Sigmoid units and their application in fault diagnosis for planetary gearboxes of wind turbines," *TIE*, vol. 66, pp. 3814–3824, 2018.

[15] S. K. Roy *et al.*, "LiSHT: Non-Parametric Linearly Scaled Hyperbolic Tangent Activation Function for Neural Networks," arXiv:1901.05894, 2019.

[16] A. Farzad *et al.*, "A comparative performance analysis of different activation functions in LSTM networks for classification," *NCA*, vol. 31, pp. 2507–2521, 2019.

[17] K. He *et al.*, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification," in *ICCV*, 2015, pp. 1026–1034.

[18] S. Qiu *et al.*, "FReLU: Flexible Rectified Linear Units for Improving Convolutional Neural Networks," in *ICPR*, 2018, pp. 1223–1228.

[19] D. Clevert *et al.*, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," in *ICLR*, 2016.

[20] G. Klambauer *et al.*, "Self-Normalizing Neural Networks," in *NeurIPS*, 2017, pp. 971–980.

[21] L. Trottier *et al.*, "Parametric Exponential Linear Unit for Deep Convolutional Neural Networks," in *ICMLA*, 2017, pp. 207–214.

[22] J. T. Barron, "Continuously Differentiable Exponential Linear Units," arXiv:1704.07483, 2017.

[23] Y. Li *et al.*, "Improving deep neural network with Multiple Parametric Exponential Linear Units," *Neurocomputing*, vol. 301, pp. 11–24, 2018.

[24] Z. Qiumei *et al.*, "Improved Convolutional Neural Network Based on Fast Exponentially Linear Unit Activation Function," *IEEE Access*, vol. 7, pp. 151 359–151 367, 2019.

[25] Y. Ying *et al.*, "Rectified Exponential Units for Convolutional Neural Networks," *IEEE Access*, vol. 7, pp. 101 633–101 640, 2019.

[26] D. Kim *et al.*, "Elastic exponential linear units for convolutional neural networks," *Neurocomputing*, vol. 406, pp. 253–266, 2020.

[27] Q. Cheng *et al.*, "Parametric Deformable Exponential Linear Units for deep neural networks," *Neural Netw.*, vol. 125, pp. 281–289, 2020.

[28] S. Qian *et al.*, "Adaptive activation functions in convolutional neural networks," *Neurocomputing*, vol. 272, pp. 204–212, 2018.

[29] A. Apicella *et al.*, "A simple and efficient architecture for trainable activation functions," *Neurocomputing*, vol. 370, pp. 1–15, 2019.

[30] S. Scardapane *et al.*, "Kafnets: Kernel-based non-parametric activation functions for neural networks," *Neural Netw.*, vol. 110, pp. 19–32, 2019.

[31] L. R. Sütfeld *et al.*, "Adaptive Blending Units: Trainable Activation Functions for Deep Neural Networks," in *Intelligent Computing*, 2020, pp. 37–50.

[32] M. Tavakoli *et al.*, "SPLASH: Learnable activation functions for improving accuracy and adversarial robustness," *Neural Netw.*, vol. 140, pp. 1–12, 2021.

[33] R. Kohavi *et al.*, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," in *IJCAI*, 1995, pp. 1137–1145.

[34] B. L. Welch, "The generalization of 'STUDENT'S'problem when several different population varlances are involved," *Biometrika*, vol. 34, pp. 28–35, 1947.

[35] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009.

[36] Y. Netzer *et al.*, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *NIPS*, 2011.

[37] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[38] C. Tarin *et a.*, "Deep Learning for Classical Japanese Literature," arXiv:1812.01718, 2018.

[39] X. Han *et al.*, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," arXiv:1708.07747, 2017.

[40] K. He *et al.*, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016, pp. 770–778.

[41] N. Ma *et al.*, "ShuffleNet v2: Practical Guidelines for Efficient CNN Architecture Design," in *ECCV*, 2018, pp. 116–131.

[42] A. Howard *et al.*, "Searching for MobileNetV3," in *ICCV*, 2019, pp. 1314–1324.

[43] X. Glorot *et al.*, "Deep Sparse Rectifier Neural Networks," in *AISTATS*, 2011, pp. 315–323.

[44] R. Collobert *et al.*, "Natural Language Processing (almost) from Scratch," *JMLR*, vol. 12, pp. 2493–2537, 2011.

[45] A. Krizhevsky and G. Hinton, "Convolutional Deep Belief Networks on CIFAR-10," University of Toronto, Tech. Rep., 2010.

[46] A. L. Maas *et al.*, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *ICML*, 2013, p. 3.

[47] R. Polyak *et al.*, "The Newton log-sigmoid method in constrained optimization," in *AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 1998, p. 4797.

[48] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010, pp. 249–256.

[49] C. Dugas *et al.*, "Incorporating Second-Order Functional Knowledge for Better Option Pricing," in *NIPS*, 2000, pp. 472–478.

[50] L. B. Romdhane, "A softmin-based neural model for causal reasoning," *IEEE Transactions on Neural Networks*, vol. 17, pp. 732–744, 2006.

[51] J. Bridle, "Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters," in *NIPS*, 1989, pp. 211–217.

[52] F.-C. Chen, "Back-propagation neural networks for nonlinear self-tuning adaptive control," *CSM*, vol. 10, pp. 44–48, 1990.

[53] M. Courbariaux *et al.*, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," in *NeurIPS*, 2015, pp. 3123–3131.

[54] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," arXiv:1606.08415, 2016.

[55] D. Misra, "Mish: A Self Regularized Non-Monotonic Activation Function," in *BMVC*, 2020.

[56] M. A. Mansor and S. Sathasivam, "Activation function comparison in neural-symbolic integration," in *AIP Conf Proc.*, 2016, p. 020013.

[57] A. N. Samatin Njikam and H. Zhao, "A novel activation function for multilayer feed-forward neural networks," *Applied Intelligence*, vol. 45, pp. 75–82, 2016.

[58] B. Xu *et al.*, "Revise saturated activation functions," arXiv:1602.05980, 2016.

[59] M. Basirat and P. M. Roth, "The Quest for the Golden Activation Function," arXiv:1808.00783, 2018.

[60] L. Li *et al.*, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," *JMLR*, vol. 18, pp. 6765–6816, 2017.