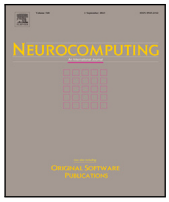
Neurocomputing 591 (2024) 127767

Contents lists available at ScienceDirect

Neurocomputing

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



On the effects of recursive convolutional layers in convolutional neural networks

Johan Chagnon [a,*], Markus Hagenbuchner [a], Ah Chung Tsoi [a], Franco Scarselli [b]

[a] University of Wollongong, Northfields Avenue, 2500, Wollongong, Australia
[b] University of Siena, Via Roma, Siena, 53100, Italy

ARTICLE INFO

Communicated by K. Li

Keywords:
Convolutional neural network
Recursive neural network
Dynamic depth
Image classification

ABSTRACT

The Recursive Convolutional Layer (RCL) is a module that wraps a recursive feedback loop around a convolutional layer (CL). The RCL has been proposed to address some of the shortcomings of Convolutional Neural Networks (CNNs), as its unfolding increases the depth of a network without increasing the number of weights. We investigated the "naïve" substitution of CL with RCL on three base models: a 4-CL model, ResNet, DenseNet and their RCL-ized versions: C-FRPN, R-ResNet, and R-DenseNet using five image classification datasets. We find that this one-to-one replacement significantly improves the performances of the 4-CL model, but not those of ResNet or DenseNet. This led us to investigate the implication of the RCL substitution on the 4-CL model which reveals, among a number of properties, that RCLs are particularly efficient in shallow CNNs. We proceeded to re-visit the first set of experiments by gradually transforming the 4-CL model and the C-FRPN into respectively ResNet and R-ResNet, and find that the performance improvement is largely driven by the training regime whereas any depth increase negatively impacts the RCL-ized version. We conclude that the replacement of CLs by RCLs shows great potential in designing high-performance shallow CNNs.

1. Introduction

Over the last decade, Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision [2,3]. They produce state-of-the-art results and continue to push the limits frequently by using various structural and algorithmic improvements. Recent CNNs may be broadly classified into two categories: Unstructured architecture – designed using an architectural search strategy [4,5], and, Structured – designed using a well structured model [2]. Among the Structured category, a number of them are based on structuring the network into a stem, followed by a number of stages, where the transition between stages would be a reduction in the spatial dimension and a corresponding increase in the channel dimension, followed by a classifier consisting of one or more fully connected layers. Networks which belong to this group include the 4-CL (4 Convolutional Layer) model [1],[1] AlexNet [6], VGG (Visual Geometry Group) network [7], the Inception Network [8], ResNet [9], ResNeXt [10], DenseNet [11]. Apart from the 4-CL model, the others contain often one or more residual blocks: a residual block consists of a direct feedthrough connection between the input and output of the block, in parallel with some configurations of CLs, like a cascade of two CLs in the case of ResNet; the details could be different for different types of CNNs in this category, in each stage; the key is the direct feedthrough connection from the input of the block to the output of the residual block. These residual blocks would ensure that in the backpropagation of the error the issue of vanishing gradient [12] would not occur. It is this feature which allows the design of deeper and deeper networks, may be consisting of 1202 trainable layers [13]. However, in such models, each additional residual block increases the number of trainable weights, thus rendering the training process longer, and often for small to medium sized datasets, this would have a high possibility of overfitting. Moreover, this tendency of designing networks of increasing depth appear to contrast what is known in the visual cortex of a primate's brain: this consists of no more than six layers of neurons, which have lateral and feedback connections, apart from the feedforward connections [14].

In contrast, one of the advantages of an RCL (Recursive Convolutional Layer), which simply wraps a recursive feedback loop around an CL, when the RCL is unfolded a number of times this increases the apparent depth of the network but would not incur any increase in the

* Corresponding author.
E-mail addresses: jjrc826@uowmail.edu.au (J. Chagnon), markus@uow.edu.au (M. Hagenbuchner), act@uow.edu.au (A.C. Tsoi), franco@diism.unisi.it (F. Scarselli).

[1] In [1], this is called an 'CNN' and we have changed this to '4-CL' model which is more descriptive of its structure, as well as to avoid confusion with the more generic name, which is used to describe any network which consists of possibly many CLs arranged in different configurations of the residual blocks.

https://doi.org/10.1016/j.neucom.2024.127767
Received 29 June 2023; Received in revised form 27 January 2024; Accepted 23 April 2024
Available online 26 April 2024

number of trainable weights [1,15,16]. This is made possible via the sharing of weights [1,15,16]. To the best of our knowledge RCLs do not appear to have been studied on these deep networks which use residual blocks.

Therefore, we propose to address this issue: lack of knowledge on the effects of replacing some or all of the CLs in a network, particularly deep networks, like ResNet, or DenseNet, by an RCL. We choose these two particular deep networks, because they are most similar to the 4-CL model, in that they are using the same number of stages, and they could have the same stage compute ratio (see Section 4). The differences are that the 4-CL model involves only CLs in its stages, while ResNet, DenseNet contain residual blocks consisting of only cascaded CLs, in their respective stages. While ResNet is one of the most popular deep networks, used in many applications, DenseNet could be considered as a more complex version of ResNet with feedforward connections from the previous stage to all succeeding stages.

We first conducted an experiment on three base models of increasing complexity: the 4-CL model, the ResNet, the DenseNet, and their RCL-ized counterparts: the C-FRPN (Convolutional-Fully Recursive Perceptron Netowrk), R-ResNet and R-DenseNet, where "R-" denotes replacement of all CLs by RCLs, in the corresponding model on five popular image classification datasets. Section 6.1 compares the classification performances and it is observed (A) the 4-CL model benefits greatly from RCL-ization, and (B) the performance of the deep networks, ResNet and DenseNet show only mild or no improvements when compared with their R-variants.

These observations motivate us to explore why the replacement of CLs by RCLs can yield benefits on the 4-CL model. Towards this end, we present the result of experiments in Section 6.2 through to Section 6.7, addressing the depth, the number of times that an RCL is unfolded, the positioning of the RCLs and their weight distributions on the underlying network. We find that, in practice, RCLs share some similarities in increasing the depth of a network when they are being unfolded even though the increase in depth is not incremental, yet they can achieve greater weight efficiency, due to the fact that the unfolded CLs share the same weights, when compared to simply adding more CLs. In addition, several characteristics of RCL-ization of the 4-CL model are uncovered:

- We identify two cases where the dataset might hurt the capabilities of RCLs: (1) When the dataset is very small, the use of RCLs increases the risk of overfitting when compared to using CLs; (2) When the data can be fitted relatively well with only a few CLs, the positive effects of using RCLs might become less.
- We address some aspects explaining the advantages provided by RCLs: we show that the classification of the feature maps gradually improves with the number of unfolding, and that the weights in trained RCLs are of greater diversity when compared to the trained weights in CLs.
- We show that models relying on RCLs are much more robust to architectural design choices, since the depth flexibility allows to better capture features at different scales (through the increase in the receptive field as the RCL unfolds), a capability that greatly simplifies the design of CNNs.

Armed with knowledge of these properties of RCL-ization of the 4-CL model, we re-visit the first set of experiments, of comparing the performances of the three base models: 4-CL model, ResNet, and DenseNet, and their RCL-ized variants, by conducting an extra experiment on the dataset with the least number of training data: CUB200, by incrementally transforming step by step, out of 11 possible steps, the 4-CL mode to the ResNet, and the C-FRPN to the R-ResNet respectively. This set of experiments, presented in Section 6.8, reveals that (1) the RCL-ized version of the 4-CL model can match the performance of the much deeper ResNet by simply adopting the same training regime as that used by ResNet and (2) that indeed any attempt in increasing the depth of the network could result in detrimental effects. This confirms the two observations (A) and (B) made in the first set of experiments. Armed with these insights we are then able to provide general suggestions one where and how to use RCLs in CNNs.

The rest of this paper is structured as follows: Section 2 provides an overview of work which use RCLs and identifies that further work is required to have a better understanding of the capabilities of RCLs. Sections 3 and 4 describe the dataset, the three base models considered and their RCL-ized versions. Sections 5 and 6 introduce the experimental setup and provide the results of the benchmark and the following experiments. We summarize our findings in Section 7 and to explicitly provide guidelines for an appropriate application of RCLs in shallow CNN designs. Section 8 completes the paper by offering some conclusions on the work presented and by providing some future research directions. Appendix A provides further details of the ResNet and DenseNet architectures, while Appendix B provides further details of the steps used when transforming the 4-CL model and the C-FRPN model to the ResNet and R-ResNet respectively.

2. Related work

In this section, we will first consider the context of our experiments, in terms of deep CNNs in Section 2.1 and then in Section 2.2 we will consider more specifically on the properties of RCL-ization which would contain details of convolutional operation, replacing a CL by a RCL, and other possible approaches to increasing the depth of a network, with no increase on the number of trainable weights in the resulting network. These other approaches are indicated only as a pointer for future work, and not in the scope of this paper.

2.1. Convolutional neural networks

The introduction of CNNs (see e.g., [17]) into deep learning (DL) has revolutionized the field of computer vision, in that it is shown it can achieve state-of-the-art results (see e.g., [18]) when compared with e.g., transformer based methods (see e.g., [19,20]), on many image classification benchmark datasets. It is quite difficult to pinpoint this extraordinary success of such a simple concept: convolution operation with trainable kernel parameters. This is because over the years, this simple concept has been used in ingenous ways to exploit its effectiveness in using trainable weights, as compared to the less effective ways in using fully connected layers, like Multilayer Perceptrons (MLPs), or the self-attention mechanism in transformers [20], the ways in which such network may be trained, using multiple loss functions [21], or in their architectural design, in taking advantages of the increase in channel dimension when the spatial dimension is reduced (see e.g., [22]), and allow the information extracted in the channel dimension to be more effectively utilized (see e.g., [22]). There are literally thousands of papers which have been published in the last few decades on CNNs (see e.g., [4,5] for the sub-field of network architectural search alone) and this defies effort in trying to categorize such volume of outputs in the community.

If one considers these outputs from an architectural point of view, then one may categorize these publications into two broad categories: Unstructured, and Structured. The Unstructured category could be characterized by the fact that an eventual architecture may be found by systematic search on an architectural search space of multiple dimensions, while optimizing on some loss functions [4,5]. Typically this produces an architecture which is a composition of the components in its search space [4,5]. The Structured category often starts with an architecture which has been found useful, and then work on a variation of such an architecture to obtain improved performance (e.g., [23] which adopted interpolation of the latent space of an autoencoder [24] to an existing structure.). Then there are some approaches, e.g., [25] which evolved from a ResNet model, by searching the architectural space consisting of different components, e.g., location and type of normalization strategy, location and type of activation functions, adoption

of the ResNeXt architecture, etc. which may thus be considered as a hybrid between the Unstructured and Structured categories.

In the Structured category, one important sub-category is known as deep CNNs, or deep networks; it is called deep, because such networks are characterized by its depth, in terms of trainable weight layers, of the resulting architecture. Networks in this sub-category include: AlexNet [6], VGG(Visual Geometry Group)-Net [7], Inception Network [8], MobileNet [26], MobileNetv2 [27], ResNet (Residual Network) [9], WRN (Wide Residual Network) [13], ResNeXt Network [10], RCNN [1,15]. A common characteristics of this group of networks is that, except for AlexNet, VGG-Net, and RCNN, it usually consists of a stem (a CL), followed by a number of stages, each stage containing a number of residual blocks, and the transition between stages incurs a reduction in the spatial dimension with a corresponding increase in the channel dimension. The structure of the residual block could be one way of differentiating why there are a number of networks in this sub-category. A residual block consists of two parallel paths between the input and output of the residual block: one direct feed-through, while the other one contains a residual module. For ResNet, and DeneNet, the residual module comprises of typically two CLs; for the Inception Network, it typically contains a number of CLs with different kernels, e.g., 3×3, 3×1, 1×3, connected in parallel; for MobileNet, the module comprises of decomposition of a convolutional kernel operation into two components, a 1×1, followed by a depth-wise convolutional operation (essentially performing $k \times k$ convolutional kernel operations, where k is the size of the kernel, on each channel and then stack the results together), and for others, it is often, a combination of selected few of all these operations, inside the residual module. For the RCNN, it consists of three stages, where stage 1 contains no CL, stage 2, and 3 contain two CLs each, but no residual blocks. We call this model a 4-CL model, to emphasize the number of CLs used in the RCNN. For DenseNet [11], apart from the same ResNet backbone, it also contains direct feedforward connections from a preceding stage to all succeeding stages, but not vice versa, in terms of any feedback loops; it is a feedforward only network.

The design of residual blocks offers a nice way to overcome an issue which exists in the training of MLPs, viz., the vanishing gradient issue [12]. Essentially when a network consisting of many hidden layers, the backprogation error, might become vanishing small, and thus, rendering the weight update process ineffective, especially affecting the layers towards the input end of the network. With a residual block structure, this limitation is largely eliminated, as the backpropagated error would be unlikely to become vanishing small because the input to the residual block is also present in the output of the block. Therefore, allowing the design of deeper and deeper networks for large datasets. For example it is claimed by [13] that in their version of the ResNet it contains 1202 trainable weight layers.

An issue of the residual block structure is that the number of weights increases with each additional residual block. This could cause overfitting in small datasets. This fact is well recognized, and one way of overcoming this issue is to use smaller sized networks for small sized datasets, and large sized network for large sized datasets. So, for example, in ResNet, there are the ResNet-X models, where 'X' could be T(iny), S(mall), B(ig), or L(arge). The differences among these ResNet-X models are the number of stages, and the number of residual blocks, in each stage, parametrized by the stage compute ratio. For example, a ResNet-T model consists of four stages, and a stage compute ratio of (3, 3, 9, 3), i.e., three residual blocks in stage 1, 2, and 4, and nine residual blocks in stage 3. Even this is often too large a model for small datasets, and therefore there are ResNet models, using, say, three stages, and a stage compute ratio of (3,3,3).

RCNN [1,15], on the other hand, use a different strategy to create the depth of the network. The RNN contains only 3 stages, with a stage compute ratio of (0, 2, 2), and uses a recursive convolutional layer (RCL) strategy, which wraps a recursive feedback loop around each of the CLs. This recursive feedback loop unfolds in time, say, N steps. This is equivalent to an N-CL network, with all N CLs sharing the same set of weights. In other words, the depth of the network comes from the unfolding of the recursive feedback loop. For a fixed weight budget, this unfolding of the CLs would therefore be more efficient when compared with those of deep networks such as e.g. ResNet. In this paper we call the process of replacing a CL by an RCL, RCL-ization. While the unfolding of an RCL is somewhat similar to a residual block, there are some differences in one crucial aspect: weight sharing makes the RCL-ized version of a network more efficient in the use of the number of trainable weights, when compared with deep networks. For the same number of weights, the RCL-ized version could be much deeper than, say, an ResNet.

It appears that no one has ever investigated the idea of RCL-ization, i.e., replacing each CL by an RCL, of a ResNet, or a DenseNet. We choose these two deep networks because ResNet is one of the most popular deep networks, it has been used in many image classification studies, first in its own right, and then as a backbone network for more advanced networks. And DenseNet because it can be considered an example of one of the many variations of the ResNet. One of the differences of DenseNet lies in the way it compensate for the increase in the number of weights by adjusting the reduction rate in the spatial dimension and the corresponding expansion in the channel dimension in the transition between stages. The residual block appear implicit in the downsampling of the spacial dimension in the transition stage. It is a deep network containing three stages with a stage compute ratio of (6, 6, 6). Other networks, e.g., Inception Network involves a change to the structure to the residual module, which in the case of ResNet, and DenseNet it is a cascade of two CLs. This is similar to the basic RCNN module used in [1,15].

We start the work presented in this paper with a proof of concept experiment, in addressing the issue at hand: what would be the effect of RCL-ization (replacing each CL in these three base models by RCL) of the 4-CL model, ResNet, and DenseNet on five popular image classification datasets. We will find that (A) RCL-ization benefits the C-FRPN, the RCL-ized version of the 4-CL model, and (B) RCL-ization of ResNet or DenseNet does not appear to have much effects on their respective RCL-ized versions: R-ResNet or R-DenseNet. This prompted us to investigate the effects of RCL-ization, e.g., whether one should replace all CLs by RCLs, or only some of them. We further investigate the order of the replacement of CL(s) by RCL(s), the number of times in the unfolding of an RCL, the diversity of the channel weights (feature maps as each channel constitutes one feature map) would have on the 4-CL model. Details are provided in Section 2.2 below. Armed with the knowledge of the properties of RCL-ization, this then prompted us to re-visit the proof of concept experiment, in an attempt to understand why R-ResNet, and R-DenseNet do not appear to have improvements over those of ResNet, and DenseNet respecitvely.

2.2. Recursive convolutional layers

The origins of the concept of RCL can be traced back to the Fully Recursive Perceptron Network (FRPN) [28], which can be interpreted as the one dimensional version of C-FRPN; it wraps a recursive feedback loop around a layer of neurons. The FRPN is itself an extension of MLPs, which is a class of simple neural networks comprised of an input layer, one or more hidden layers and an output layer, and where each neuron in a layer is connected to each neuron of the successive layer via a set of weights. MLPs are universal approximators [29] that are at the core of the neural network paradigm, and are commonly used in many regression and classification tasks. The performances of MLPs depend essentially on the way they are designed, which varies according to the activation functions, the number of hidden layers, and the number of neurons in each hidden layer. In particular, finding a sufficient number of layers and the number of neurons of each of them for a new learning problem can be challenging as there are no general rules to determine what these structural parameters should be (see e.g., [30]).

The FRPN differs in that there is only one single hidden layer, the neurons of which are inter-connected to all neurons in the layer including to themselves. Each of these neurons receives a sensory input that is called the "input bias". The hidden layer neurons are activated recursively: after a definite number of unfolding of the recursive loop, their output is then forwarded to an output layer. Formally, this can be described as follows: Let $\mathbf{u} \in \mathbb{R}^m$ be the input of the FRPN, $\mathbf{x}^{(t)} \in \mathbb{R}^n$ the output of the hidden layer at recursion t, with $x_i^{(t)}$ being the output of neuron i and $\mathbf{y} \in \mathbb{R}^l$ the output of the FRPN at $t = t_{max}$. Considering two set of bias $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{c} \in \mathbb{R}^p$, three sets of connections weights $\boldsymbol{\alpha} \in \mathbb{R}^{n\times m}$, $\boldsymbol{\beta} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{\gamma} \in \mathbb{R}^{p\times n}$, a non-linearity f and an output function g, we can define the equations of the FRPN as follows:

$$x_i^{(t+1)} = f\left(\sum_{j=1}^{m} \alpha_{ij} u_j + \sum_{k=1}^{n} \beta_{ik} x_k^{(t)} + b_i\right) \tag{1}$$

$$y_l = g\left(\sum_{k=1}^{n} \gamma_{lk} x_k^{(t_{max})} + c_l\right) \tag{2}$$

where $\mathbf{x}^{(t=0)} = \mathbf{0}$ and $t \in [\![0, t_{max} - 1]\!]$. In the original implementation of the FRPN, Eq. (1) is applied recursively until t_{max} is reached or until the convergence criterion in Eq. (3) is satisfied:

$$\|\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}\| < \epsilon, \tag{3}$$

where $\epsilon > 0$ is a predefined threshold. By using $\beta_{ik} = 0$, we see that Eq. (1) describes the equation of an MLP with one hidden layer. This shows that FRPNs generalizes MLPs, so they are a universal approximator, while they remove the constraint of setting the number of layers and their respective sizes. The only structural parameter in FRPNs is the number of neurons in the hidden layer, which makes them easier to design than MLPs. Furthermore, it is clear that FRPNs simulate MLPs with any number of hidden layers, with no extra cost of parameters. In addition, it is shown that FRPNs reach better performances than MLPs, even when the total number of parameters in the MLP and the FRPN are approximately the same [28].

Based on this pioneering work it has been proposed to apply the same principle to CLs to obtain RCLs as follows: Let $\mathbf{u} \in \mathbb{R}^{C\times h\times w}$ be the input, where C is the number of channels, and h, w are the feature maps height and length, respectively. For the convolution, we have the weights $\mathbf{w} \in \mathbb{R}^{C\times N\times N}$ and bias $b \in \mathbb{R}$. Then the classic convolution equation, when the convolution is applied on a window of size $N \times N$ at position (s_x, s_y), can be shown as in Eq. (4)

$$x_{s_x,s_y} = b + \sum_{c=1}^{C} \sum_{i=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \sum_{j=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} w_{cij} u_{c,s_x+i,s_y+j}, \tag{4}$$

which can be denoted more compactly as $\mathbf{x} = \mathbf{w} * \mathbf{u} + b$. Recursion can be introduced in analogy to Eqs. (1)–(2), representing the action of an RCL with M kernels such that $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_M) \in \mathbb{R}^{M\times C\times N\times N}$, $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_M) \in \mathbb{R}^{M\times C\times N\times N}$, and biases $\mathbf{b} = (b_1, \ldots, b_M) \in \mathbb{R}^M$ as defined in Eqs. (5)–(6).

$$\mathbf{x}_m^{(t+1)} = f\left(b_m + \boldsymbol{\alpha}_m * \mathbf{u} + \boldsymbol{\beta}_m * \mathbf{x}^{(t)}\right) \tag{5}$$

$$\mathbf{x}^{(t)} = (\mathbf{x}_1^{(t)}, \ldots, \mathbf{x}_M^{(t)}), \tag{6}$$

where it is assumed that the initial state is set to zero, i.e., $\mathbf{x}^{(0)} = \mathbf{0}$. Notice that $b_m + \boldsymbol{\alpha}_m * \mathbf{u}$ is constant across the different time steps, whereas $\boldsymbol{\beta}_m * \mathbf{x}^{(t)}$ changes during recursion. Described as such, we see that RCLs can replace CLs individually in a CNN, which results in networks that can make use of recursion at different levels.

RCLs have first been considered for CNNs by [15]. The corresponding architecture, called the Recursive CNN (RCNN) has been designed by using AlexNet [6] as a baseline, consisting of a stem, and four convolutional layers, then replacing all but the first CL (the stem), with RCLs and using a fixed number of unfolding steps $t_{max} = 4$. The comparison was made using different number of parameters and for four standard benchmark datasets of image classification: MNIST [31], CIFAR10 [32], CIFAR100 [32], and SVHN [33]. Very much alike in the original FRPN paper [28], the authors reported that the RCNN requires less weights to match the results of its feedforward counterpart, and that the performance increases when the same number of weights is used. The authors in [15] suspected that the improvement in results may be attributed to the increase of receptive field that results from the increased depth but this is not explicitly demonstrated.

A more general approach, which expanded the research on RCLs, introduced the Convolutional-FRPN (C-FRPN) architecture [1]. It uses the same backbone as in [15] and focuses specifically on the number and the position of RCLs to understand what would be an optimal configuration. Another notable difference is that instead of using only a fixed number of time steps, the recursive loop is stopped dynamically and in accordance to Eq. (3). Here, the authors based their study on three datasets: CIFAR10, SVHN and on ISIC [34]. They show that for different network sizes, and reasonably large datasets it is always beneficial to replace CLs with RCLs, while keeping the same number of parameters. They also report that the performances of the C-FRPN are better than the performances of the feedforward version on CIFAR10 and SVHN, and worse on ISIC. This intriguing discrepancy is presumed to come from the small number of samples in the ISIC dataset used, which might amplify overfitting when using RCLs.

We will investigate these findings by considering alternative explanations based on additional differences in the properties of the datasets. For example, the ISIC dataset used in [1], using the same base model as the one used in [15], differs in many other ways from the other datasets used in their paper, e.g. the resolution of the sample which is seven times larger, or the nature of the dataset which represents skin lesions and is vastly different in distribution from the natural images found in CIFAR10 and the street numbers in SVHN [33]. Therefore, we aim to consolidate the ground knowledge behind the use of RCLs, in order to identify their strengths and weaknesses.

There has been attempts to overcome some of the limits of the RCL. Nayebi et al. [16] and Wang et al. [35] showed that the results previously achieved using standard recurrent cells as defined in Eq. (5) do not generalize well when trained on a much larger dataset, ImageNet-1k [36], which is a collection of 1.2 million images. Thus, those results do not support the claim that overfitting is sufficient to explain the comparatively bad generalization capability of the C-FRPN in specific cases. Both approaches designed a new RCL which relies on a property inherited from signal processing, called gating, which selectively modifies the flow of data in the RCL and, in analogy to how gates are used in the LSTM, weights the amplitude of the neuronal response. In this context, gates are designed to limit the growth of receptive field resulting from recurrency – the gating signal depends on information at the previous time step, rather than recursiveness – the gating signal depends on the signal at the current time step only. The gating signal in their cases can be designed in various ways, thus [16] performs a Neural Architecture Search to find a near-optimal configuration and [35] proposes a similar but simpler and very effective design. In both cases, the introduction of gates led to better performances, even on Imagenet-1k, making them comparable or better than their feedforward CNN counterpart. We note that the question of designing a gating signal in these RCNNs[2] base model could be an interesting topic for future research.

A variety of work have shown that CNN that uses an adaptive receptive field can obtain significant improvements. For example, Deeplab [37] obtained state-of-the-art results in image segmentation by dilating the convolutional kernels, which increases the receptive field of the convolutions. The use of deformable kernels also leads to an adaptive receptive field, which is shown to be favorable [38]. Similarly,

[2] We found that this term: RCNN could be confused with recurrent CNN, which is different from the recursive CNN which [15] uses. As this model contains one stem, and four CLs, we will denote this model as the 4-CL model in this paper henceforth.

the combination of separate fixed size kernels is at the core of the Inception network [8], which could be considered as using a variation of the design of the basic residual block used in ResNet; instead of having two CLs in cascade, using a combination of CLs with different kernels in parallel.

It is however unclear how gating would benefit very small datasets, which might be prone to other issues. In fact, gated RCLs happen to have better performances on a large dataset, but neither the gating RCLs, or the investigation of the receptive fields of RCLs, address specifically how RCLs or gated RCLs learn and what makes them more efficient in the general case. Rather than proposing a different design of the residual block, like that of Inception network, we propose here to establish different properties of simple, non-gated RCLs following Eq. (5). This allows us to understand more general and scalable properties.

3. Dataset presentation

All models in this paper are evaluated on five datasets that have been selected for their diversity and popularity, addressing various aspects, as follows:

CIFAR10. This dataset [32] contains 50,000 training and 10,000 test images of size 32×32 that are evenly distributed into 10 classes, namely: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. CIFAR10 represents natural scenes and is among the most commonly used datasets for benchmarking image classification models.

CIFAR100. This dataset [32] has the same characteristics as CIFAR10 except that the images are distributed into 100 classes. This implies that there are 10 times fewer images per classes, which makes the learning task harder. Comparing the results obtained on both datasets should give us a good indication on what happens on more difficult tasks.

SVHN. The Street View House Number dataset [33] is made of 73,257 training images and 26,032 test images representing street house numbers in natural scenes of large diversity, which varies in font, zoom-level, orientation, background. The images are of size 32×32, centered on a digit, and the task is to recognize the central digit. As with CIFAR10, it is a widely referenced dataset of low resolution.

ISIC. The International Skin Imaging Collaboration [34] provides a collection of challenges released from 2016 to 2020 in various tasks, notably image classification of skin lesions. Following the guidelines of [39], we extracted a curated and balanced dataset after merging the different collections and removing the duplicates. We are left with a dataset of 9810 images made from 4905 melanomas and 4905 other lesions, 20% is used to create a test set and each image is resized and center-cropped to be of size 224×224. This dataset was selected for its singularity with respect to the other as it contains few but medium-resolution images of medical nature.

CUB200. The Caltech-UCSD Birds 200 dataset [40] consists of 5994 training images and 5794 test images of birds belonging to 200 species. As for ISIC, the images are resized and center-cropped to be of size 224×224. With an average 30 training images per class, and considering that the classes are quite similar from one another, this classification task is particularly difficult, which is why this dataset is often used to evaluate the fine-grained capability of a model. It is a smaller, but more difficult dataset of medium-resolution images than ISIC.

4. Model description

RCL-ization. Throughout this paper, we address the impact of substituting CLs with RCLs in a CNN. We choose to refer to this process as "RCL-ization". This is done in a way such that the replacement of a CL with RCL will not require additional weights. For a fairer comparison, between the feedforward model and its RCL-ized version, we systematically balance the number of weights of the RCL-ized model with its feedforward counterpart. This balancing process, whether it is obtained by reducing equally the number of feature maps per layer or by decreasing the number of modules, is always implied when we refer to RCL-ization in this paper. The way the number of weights is balanced in this paper vary according to the nature of the base model, but it is always specified. What we refer to as "naïve" RCL-ization refers to the process of replacing every CL of the feedforward model, except the stem, with RCL without giving any consideration on where the CL is located.

4-CL model and C-FRPN. First, we compare two architectures: a feedforward CNN, the 4-CL model, and its RCL-ized version, except the stem: the C-FRPN.[3] The C-FRPN model is as shown in Fig. 1. It may be observed that the first module, portrayed in yellow at position $P0$, the stem, is the only feedforward CL in the network, and uses a kernel size of 5×5 with a stride of 1. The following modules in position $P1$ to $P4$ are RCLs denoted as RCL1, RCL2, RCL3, and RCL4 corresponding to their position within the network. The four RCLs use a kernel size of 3×3 with a stride of 1. Additionally, we use a maxpooling layer after layer $P0$ and $P2$ with a kernel size of 2×2 and a stride of 2 to halve the spatial dimensions but a doubling of the number of channels. The output of each module is followed by a ReLU and a Local Response Normalization of size 13, which then goes through a dropout layer with a forget rate of 0.5 except for the last module. Inside RCLs, ReLU and Local Response Normalization are applied after each unfolding step. The feature maps obtained after the fourth RCL are converted into a feature vector using a global average pooling layer, which is then fed to a fully connected network acting as a classifier. The 4-CL model is simply obtained by replacing each RCLs in the C-FRPN with CLs. In the terminologies used to describe ResNet (see the paragraph below), the 4-CL model is a three stage model, with a stage compute ratio of (0, 2, 2), with stage 1 containing no CLs, and stage 2, and 3 contain two CL each. The transition between stages incurs a reduction by half in the spatial dimension but a doubling in the number of channels, is conducted through the max pooling layer. As stage 1 is degenerate, containing no CLs, this is collapsed into the stem. The design of this C-FRPN is consistent with that in [1].

Additional RCL configuration. The experiments conducted in this paper aim to not only to compare these two architectures, but more globally to evaluate the effect of replacing an CL with an RCL in any position. To this end, we also present a category of models, that we refer to as models with mixed configurations. Because the recursive links in an RCL would add more weights to a CL when using the same amount of filters, we balance the number of parameters of every model by reducing the number of channels in RCLs in accordance to Table 1, which follows the procedure of [1]. The procedure allows for a fair comparison between the 4-CL and its RCL-ized version. Table 1 shows, for example, that using the 4-CL model with 42 channels per CLs would require $\approx$ 67k trainable weights, which is equivalent to the number of parameters used by a C-FRPN with 30 channels per layer.

RCL-ization in ResNet and DenseNet. Additionally, we RCL-ize two deep network architectures: ResNet [9] and DenseNet [11]. This is because in some earlier work [16,35] there is indication that there might be difficulties when using RCLs in some models when they applied it on a large dataset: ImageNet-1k [36], and this motivated them to use gated recurrent units or to experiment with deformable kernels. Here, we aim to determine if this observation might be caused by the architectural choice of the base model rather than the size of the

[3] Here to be consistent with later usage, we should have called this R-4-CL model, where 'R' denotes the RCL-ized version. But we decided to use the acronym 'C-FRPN' (Convolutional Fully Recursive Perceptron Network) introduced in [1] as this avoids the awkward notation of 'R-4-CL' model.

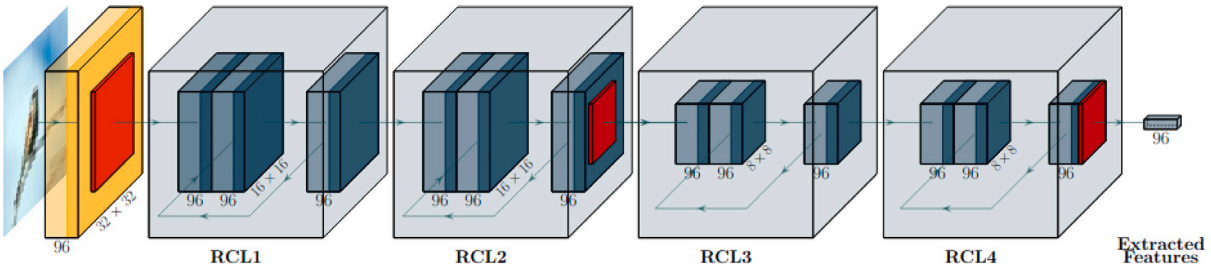


Fig. 1. The C-FRPN architecture, here with 672k weights. The first convolutional layer is shown in orange and pooling layers are shown in red.

Table 1
Number of channels per layer in the architectures used in this paper. For the mixed configuration, the number of channels used is shown depending on the module and its position in the network. The corresponding models roughly match the total number of weights that are represented under each columns.

Experiment	Layer	17k	67k	264k	400k	527k	672k
4-CL model	*P*0 to *P*4	21	42	85	104	120	135
C-FRPN	*P*0 to *P*4	15	30	60	74	85	96
Mixed configurations	CLs from *P*0 to *P*4	21	42	85	104	120	135
	RCLs from *P*4 to *P*3	15	30	60	74	85	96
	RCLs at position *P*4	13	26	52	64	73	83

dataset itself, therefore, we consider these two popular deep network models: ResNet and DenseNet. These two deep network models: ResNet and DenseNet are selected because they are most similar to the 4-CL model in that in the residual module, they contain two CLs in cascade, while in the 4-CL model, each stage, except stage 1, contains two CLs in cascade as well.

The ResNet architecture introduces direct feedthrough connection between the input and the output of a residual block. We followed the implementation procedure detailed in the original paper [9] for the specific training of CIFAR10. Among the different sizes considered, we selected ResNet-44, which has approximately as many parameter as the biggest 4-CL model and C-FRPN used in our paper viz. 672k trainable weights. The details of ResNet-44 are presented in Appendix A. At a high level, it consists of a stem, followed by three stages with a stage compute ratio of (7, 7, 7), i.e., there are 7 residual blocks in each stage, and each residual module contains two CLs.

DenseNet may be considered as one variant of the ResNet. As the details of DenseNet are provided in Appendix A here we will only describe a high level view of what is a DenseNet, in particular DenseNet-BC, which contains 760k trainable weights, the closest match to the number of trainable weights in the 4-CL model: 672k. This consists of three stages, with a stage compute ratio of (16, 16, 16), i.e., each stage contains 16 dense blocks. Each dense block consists of two CLs, one with a 1×1 kernel, followed in a 3×3 kernel. For convenience we will denote this as a 2-CL module. In some ways, this 2-CL module may be similar to the residual module in ResNet. One difference between DenseNet nad ResNet is in the ways in which the 2-CL module is connected. In DenseNet, each 2-CL module receives inputs from all its preceeding 2-CL modules. Thus, if there is only one 2-CL module, then this will be the same as a residual block. But if there are more than one 2-CL module, then it will be different, as apart from receiving an input from its own, it will receive additional inputs from all its preceeding 2-CL modules. Therefore in a stage with more than one 2-CL modules, there will be more direct feedthrough connections than those in a stage in ResNet. These direct feedthrough connections would affect the number of channels increases between stages, different from that of ResNet, which if the spatial dimension is decreased, say, by a half, then the number of channels would be doubled. For the DenseNet, the transition layer is performed by two CLs, the first one is 1×1, which halves the input number of channels, followed by a 2×2 average pooling which halves the spatial dimension. To compute the number of channels at any ℓth layer in a stage would involve a concept called growth rate k. Then for the ℓth 2-CL module, it will have $k_0 + k \times (\ell - 1)$ number of channels, where k_0 is the number of channels at the input of the stage. So for example, for a growth rate $k = 12$, and, after the stem, say, the number of channels is 24. Then after the first stage, consisting of 16 2-CL modules, the number of channels would be given by $24 + 16 \times 12 = 216$. Here $\ell = 17$ because we are considering the output of the first stage and not the input to the 16th 2-CL module. After the first 1×1 CL in the transition stage, the number of channels would be halved, i.e., $\frac{216}{2} = 108$. After the 2×2 average pooling layer, as this only affects the halving of the spatial dimension, and does not affect the number of channels, the output contains 108 channels which will serve as input to the second stage, which contains 16 2-CL modules. After the second stage, the number of output channels would be $108 + 16 \times 12 = 300$. Now it is clear that the number of channels grows according to a growth k, which could be larger than those in the ResNet at the same stage in the network.

Table 2
Number of layers per stage in the architectures considered. The number of feature maps (channels) per layer is preserved with respect to the original architectures. Numbers have been selected using the following scheme: decrease uniformly the number of layers per stage until the number of parameters from the unfolding of the recursive loop in the RCL-ized version becomes lower than for the feedforward counterpart. Then, increase the number of layers uniformly only in the first two stages before the number of parameters exceeds the one in the feedforward version. Finally, adjust the number of layers in the first stage to match the number of parameters of the base model.

Model	Stage 1	Stage 2	Stage 3	# Params
ResNet	7	7	7	659k
R-ResNet	6	6	3	661k
DenseNet	16	16	16	769k
R-DenseNet	15	15	15	761k

A note on the RCL-ization of these two base models: ResNet-44 and DenseNet-BC, we do not RCL-ize every CL, even in the case of naïve RCL-ization. We do not RCL-ize the CLs which are in the downsampling operation, nor the stem, nor the 1×1 CL in the 2-CL module in the DenseNet. To balance the number of parameters, we reduce the number of layers per stage as shown in Table 2.

As before, we use the prefix "R" to designate the recursive version of the model. The normalization layer embedded within each RCL has been changed to batch normalization [41] instead of Local Response Normalization to further harmonize the type of normalization used in the network. It is introduced such that a different batch normalization layer is used for each unfolding step.

5. Experimental setup

When not mentioned in the respective original papers of the three base models: 4-CL model [1], ResNet [9], and DenseNet [11], we determined a set of hyperparameters by training samples randomly selected from 80% of the training set and evaluating on the rest of the training set. Each model is then trained on the full training set and evaluated on the test set using that set of hyperparameters. Each result in this paper is the average and the standard deviation reported as $mean \pm std$ of five runs using different random seeds. The seed affects on the weight initialization, the order of presentation of the training samples in each batch and the dropout layers' outputs. By reporting $mean \pm std$ this provides an indication of the consistency of our results.

The 4-CL model and C-FRPN. The two model are trained in the different configurations as presented in Table 1. All are trained using an Adam optimizer with a L2 penalty of $5e-4$, a learning rate of $1e-4$ and a fixed number of training epochs, analogue to [1]. We use 500 training epochs for CIFAR10, CIFAR100, SVHN and CUB200 and 75 epochs for ISIC. To increase the diversity of the datasets, we used data augmentation as in [1]: on CIFAR10, CIFAR100, SVHN and CUB200, the images are randomly mirrored and randomly cropped over an area of 75% of the original image size. On ISIC, we randomly mirror the images and rotate them. We used mini-batches of size that could be appropriately handled by our GPU[4]: 128 for CIFAR10, CIFAR100 and SVHN, 8 for ISIC, and 16 for CUB200. Unless explicitly mentioned we always used $\epsilon = 0.1$ and $t_{max} = 8$ for the unfolding of every RCL.

ResNet and DenseNet and their respective R-versions. The training on CIFAR10, CIFAR100 and SVHN follows the same data augmentation scheme as the one described for CIFAR10 in [9], that is, using random cropping combined with 0-padding to preserve the initial resolution of the image, and using random horizontal flipping. Along with this, images are initially resized using bilinear interpolation on ISIC and CUB200 to be of size 112×112 and a random rotation is also applied on ISIC.

The same training parameters are used in the feedforward and recursive versions. We use $\epsilon = 0.1$ that is divided by a factor of 10 once the training reaches 50% and 75% of the total number of epochs, this for each models and datasets. For ResNet, we use a batch size of 128 for CIFAR10, CIFAR100, and SVHN and train the models on respectively 200, 200 and 40 epochs. On ISIC and CUB200, a batch size of 32 is used on 200 epochs. On DenseNet, we train CIFAR10, CIFAR100 and SVHN on respectively 300, 300 and 40 epochs using a batch size of 64. ISIC and CUB200 are trained on 200 epochs using a batch size of 32. Due to the large number of layers considered, we used 4 steps in the unfolding of the RCL to fit the model into the memory available on our GPU.

6. Experiments on RCLs

This section presents the results obtained from the different experiments as well as the observation and interpretation of our findings.

6.1. Performances comparison

We compare the performances of the feedforward and recursive versions of the different models on datasets presented earlier. For the C-FRPN and the 4-CL model, we use the version made of 672k weights which is the largest we considered and as such is the most representative of the effects on larger models, alleviating the potential effects that would be observed from overfitting. The results are shown in Table 3.

[4] An RTX3080 with 12 GB of memory was used for the experiments.

The 4-CL model, ResNet-44 and DenseNet-BC represent subsequent evolution of the state-of-the-art model for image classification, and this is well-reflected in the performances of the feedforward network. Furthermore, we find that replacing CLs by RCLs are significantly beneficial on the C-FRPN, but not on R-ResNet and R-DenseNet in which the performances remain within the standard deviation of their respective feedforward architectures. This may provide a different perspective on the remarks made in [16,35] that plain RCLs do not perform well on large image datasets like ImageNet-1k. Rather, it seems to be correlated with the fact that more modern type of architectures are being used on par with that dataset, and those architectures may not generally benefit from recursion. Hence, the focus of our study takes into account the type of architecture used, which we address in Section 6.8, Section 6.3 and in Section 6.4.

Focusing on the results of 4-CL model and the C-FRPN we observe that the latter outperforms its feedforward model the 4-CL model, on each dataset, but with varying degrees of success. In CIFAR100 and CUB200, the C-FRPN is particularly efficient, and remains significantly better on CIFAR10 and SVHN. On ISIC, we observe that the 4-CL model has performances that are within the standard deviation of the C-FRPN, hence the advantage of RCLs here is not significant. A similar observation was reported earlier about another dataset that was also based on ISIC, in [1]. It was then attributed to the small size of the training set, which may have particularly increased overfitting in RCLs. The results obtained on CUB200, which is of smaller size than ISIC, significantly favors the C-FRPN and hence might contradict this hypothesis. Therefore there is a need to clarify this claim and to better characterize the datasets for which the introduction of RLCs would be beneficial.

From this first analysis, we can observe that both the datasets and architectures have to be considered when introducing RCLs. Toward this goal, we go through a variety of experiments in the following subsections addressing both aspect to obtain a better understanding of the benefits of RCLs and its limits.

The findings obtained on this first experiment show that the C-FRPN can achieve better performances in some situations which are not typically present in ResNet and DenseNet. This raises the question of what makes RCLs effective on the C-FRPN, so as to pinpoint what are its strengths and its limitations. In order to address this, we pursue this study by focusing on the C-FRPN and its feedforward version, the 4-CL model. This model has the advantage of being made of a much simpler design which is more convenient as we aim to extract general properties that might be scalable to any CNN, and not affected by an extraction process which would be specific to an architecture.

6.2. The benefit of an RCL depends on its position

To further study the impact of RCL, we measured the accuracy that results from RCL-izing various position of the 4-CL model. We trained 16 models that differs with respect to the nature of each of their modules at position $P1$ to $P4$. The trained models satisfy the row of Table 1 which refers to that of mixed configuration, for a usage of 672k weight. For each position Li, we computed the relative gap in accuracy between the mean of the set R_{Li} of models that uses an RCL at position Li and the mean of the set of the remaining models C_{Li}, which uses a CL at position Li. We then compute the relative gap in accuracy at position Li following Eq. (7):

$$rg(Li) = \frac{mean(accuracy(R_{Li})) - mean(accuracy(C_{Li}))}{mean(accuracy(R_{Li}))} \tag{7}$$

Therefore, a positive output in position Li means that on average, an RCL is more beneficial than a CL at the same position.

Fig. 2 shows that the gain in accuracy resulting from the use of an RCL is not uniform but is dependent on its position. The results obtained on CIFAR10, CIFAR100 and SVHN reveal that RCLs that are

Table 3
Test accuracy of the different feedforward models compared to their recursive version. Each cell represents the mean and standard deviation out of 5 runs, as: $mean \pm std$.

Model	CIFAR10	CIFAR100	SVHN	ISIC	CUB200
4-CL model	86.60 ± 0.28	57.94 ± 0.32	94.62 ± 0.07	78.14 ± 0.52	19.95 ± 1.18
C-FRPN	**89.18** ± 0.34	**61.13** ± 0.69	**96.28** ± 0.11	**78.39** ± 0.54	**29.48** ± 3.03
ResNet	**92.92** ± 0.14	**69.35** ± 0.54	96.09 ± 0.05	**80.62** ± 0.35	54.26 ± 1.57
R-ResNet	92.17 ± 0.63	68.64 ± 0.58	**96.12** ± 0.23	80.08 ± 0.75	**55.27** ± 1.71
DenseNet	95.16 ± 0.09	76.75 ± 0.27	**96.30** ± 0.03	**81.01** ± 0.47	**60.60** ± 0.46
R-DenseNet	**95.23** ± 0.18	**76.83** ± 0.33	96.29 ± 0.07	80.90 ± 0.72	59.19 ± 0.65

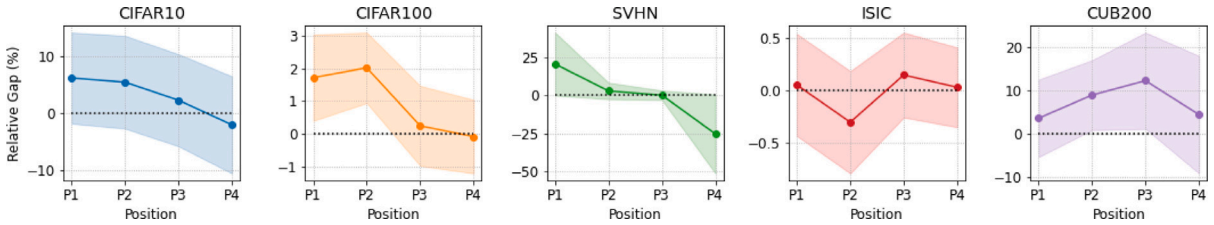


Fig. 2. Relative gap in accuracy when replacing an RCL with a CL for each position. The line plot and the transparent area represents respectively the mean and the standard deviation obtained out of 5 runs.

placed closer to the input layer of a network leads to the best performance gain.[5] In contrast, RCLs that are placed in position $P3$ or $P4$ only result in little to no benefit and can even prove to be detrimental. On CUB200, RCLs performs better when placed in intermediate positions in the network.

Focusing on the results obtained on ISIC, we observe that the benefit resulting from using an RCL at any position is never significant nor detrimental. This implies that the underwhelming performances obtained on this dataset are not likely due to an inappropriate model configuration, but that the recursion mechanism itself has little impact on ISIC. Consequently, this can show that if the accuracy of a model does not improve significantly following the replacement of a CL by an RCL in the early layers of the network, it is unlikely to benefit from any additional RCL at any subsequent position.

This experiment allows us to only partially confirm an observation made in [1]: for datasets that benefit from RCLs, the replacement of CLs by RCLs should be prioritized in the early layers. Because CUB200 contradicts this behavior, further investigation is required to understand why using an RCL in some positions should be favored according to the dataset. Additionally, we found out that if the performances do not improve after using an RCL then adding any further RCLs is unlikely to provide additional benefit. In the next subsection, we extend our analysis to models of different sizes, by reducing the number of parameters used.

6.3. RCLs adapt better to narrow architectures

We proceed to evaluate the performances of the C-FRPN and the 4-CL model on different sizes of the models, as described in Table 1. With respect to the previous experiment, the models considered in this section contain less filters per RCL/CL module, hence we refer to these models as being of smaller width, or "narrower". The results are displayed in Fig. 3 using a boxplot representation showing the values obtained in each size category, and a line plot showing the evolution of the relative gap in accuracy between both models.

This type of experiment has been conducted previously in [1] which concluded that the C-FRPN performed comparatively better when the number of weight decreases. We propose to confirm this hypothesis by reproducing their experiment on our datasets and by displaying explicitly the gap in accuracy using the relative gap, which we define this time using $rg = \frac{accuracy(CFRPN) - accuracy(4CL)}{accuracy(CFRPN)}$ for each size category. Hence, a positive rg means that the accuracy of the C-FRPN is higher in that size category.

Using Fig. 3(b), we corroborate the results obtained in [1] on CIFAR10 and SVHN: the benefit of RCLs can be more apparent on narrowed down versions of the models. We found however that such observation is limited to a certain category of datasets. On more complex tasks involving many classes as in CIFAR100 and CUB200, the relative gap in accuracy drops for smaller networks. We further investigated the influence of the number of classes by randomly grouping each class into 10 super-classes. On this modified CIFAR100, the results showed that the relative gap in accuracy obtained using the smallest models becomes much larger, with a value of 6.79%, which relates to the trends observed on CIFAR-10.[6] Hence it is likely that this effect is not specific to a particular type of data, but instead a consequence of our weight matching scheme which provides more feature maps to CLs, an asset particularly important when there is more classes to deal with.

On ISIC, we note that the relative gap in accuracy remains very stable, which can be linked with the fact that RCLs do not provide significant benefits on this dataset. Decreasing the number of feature maps per modules does not change this behavior, which is another aspect supporting that overfitting might not be responsible for these performances on ISIC.

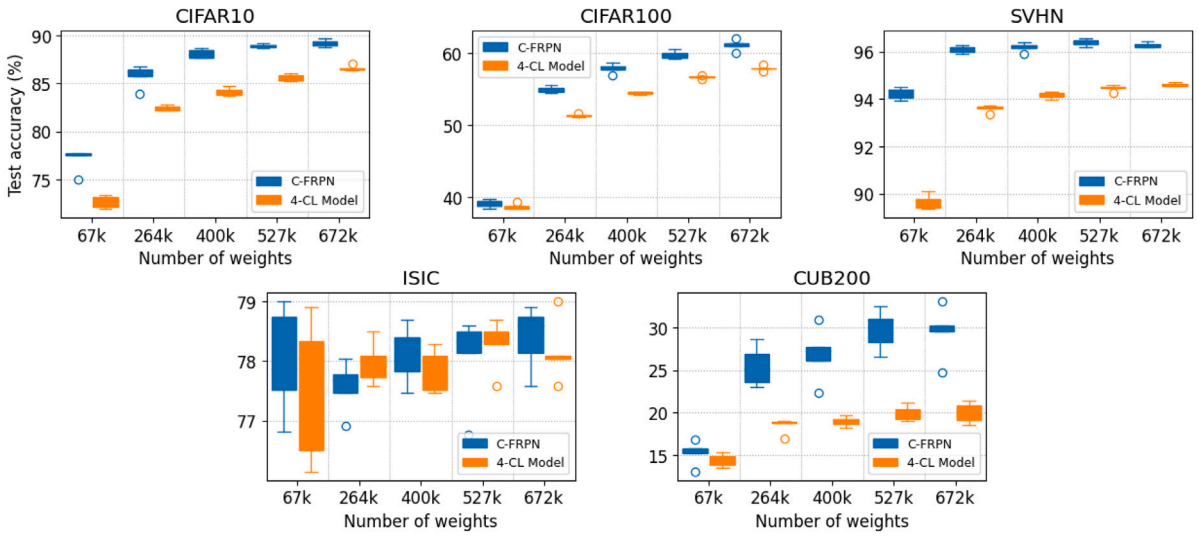
To this extent, we partially confirm the observations made in [1] and state that RCLs adapt better to narrow architectures when they provide benefit. Nevertheless, instead of reducing the size of the models by using less weights per modules, we can also consider removing some modules to obtain results more representative of the usage of RCLs in shallower networks.

6.4. RCLs adapt better to shallow architectures

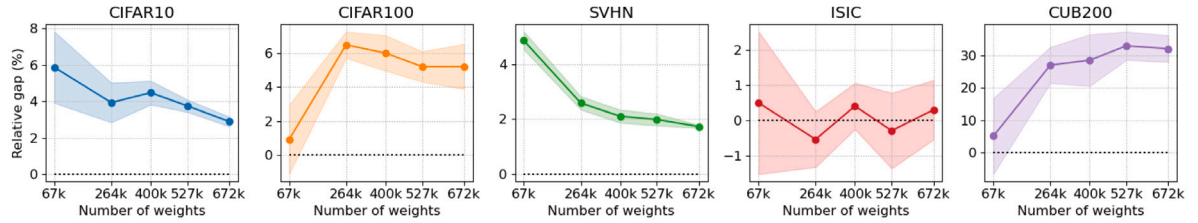
One can consider using RCLs in shallower networks, i.e., in networks which contains less convolutional layers, which results in a lower

[5] This plot is an average of the accuracy given by 16 models and thus is a simplified way of showing the influence of the positions of RCLs, but on a particular configuration and depending on the dataset, a model does not strictly follow such a rule. The average accuracy obtained in SVHN drops significantly when an RCL is used in $P4$ while a CL is used in $P1$. This does not happen in the other configurations training on SVHN, so the usage of 4 RCLs still performs better than the 4-CL model despite what the figure suggests.

[6] We could not confirm these observations on the modified CUB200 as it resulted in models that were barely better than random guessing, a consequence of the large intra-class variability it introduced.



(a) Absolute values of the accuracy of the C-FRPN and the 4-CL model. Whiskers represent the range of the data and outliers are shown as a circle.



(b) Relative gap in accuracy between the C-FRPN and the 4-CL model.

Fig. 3. Comparison of the test accuracy of the C-FRPN and the 4-CL model using models of different sizes, as described in Table 1.

depth. Here we address this aspect by comparing the usage of RCL and CL modules in three version of the 4-CL model and the C-FRPN:

- The 4-CL model and C-FRPN as presented in Section 4, which includes 4 RCL/CL modules, for a total number of 672k weights.
- The same base architectures except that the modules at position $P3$ and $P4$ are removed and the second maxpooling layer is moved between $P1$ and $P2$. It thus contains only two RCL/CL modules.
- Additional architectures which consists of a single RCL/CL module at position $P1$ and directly followed by the global average pooling layer.

We refer to each category according to the number of RCL/CL modules it contains. The number of feature maps per module is kept the same for each experiments, which is as shown in Table 1, 96 for RCLs and 135 for CLs. As a consequence, if we do not consider the CL at position $P0$ and the fully connected layer, a model with two modules has half the number of parameters as a model with four. Likewise, a model with a single module will have a fourth as many parameters.

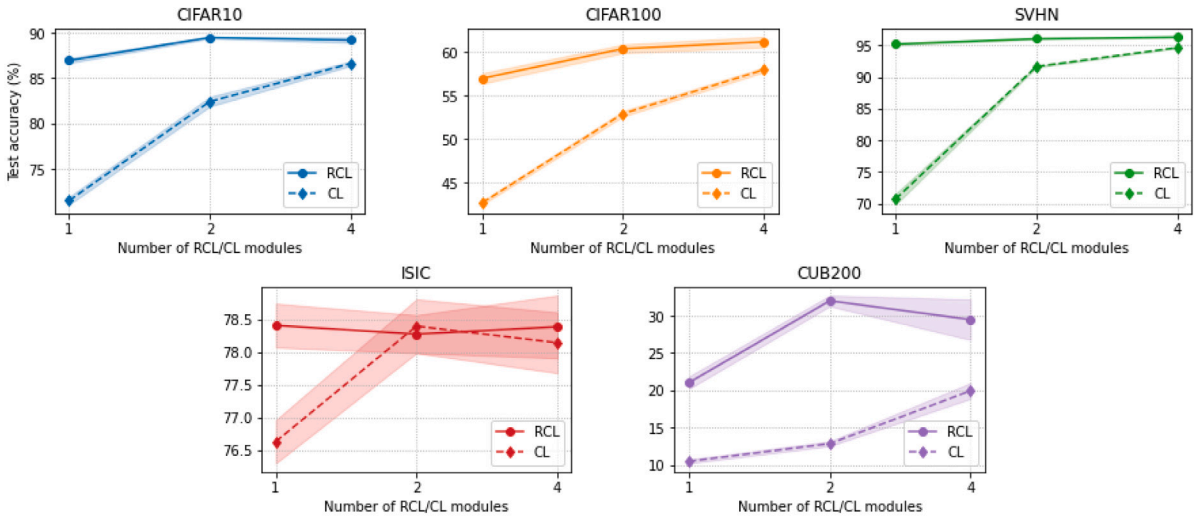
Fig. 4(a) shows how the accuracy varies in the shallower versions of the C-FRPN and the 4-CL model. We see once again that the results obtained on CIFAR10, CIFAR100, SVHN and in lesser degree CUB200 can be opposed to those obtained on ISIC.

- The results obtained on CIFAR10, CIFAR100 and SVHN follow a very similar trends: the accuracy of the C-FRPN remains quite stable despite the drastic decrease of the amount of parameters as it gets shallower. In contrast, the shallow versions of the 4-CL model result in a significant drop of accuracy.
- In CUB200, the drop in accuracy of the models using RCLs is more pronounced, albeit their value remains much superior than models with CLs.
- In ISIC, the accuracy obtained by the shallow versions of the C-FRPN and the 4-CL model differs only mildly.

We acknowledge that the classification tasks on CIFAR10, CIFAR100, SVHN and CUB200 typically requires a large degree of depth to be correctly fitted by models based on CLs. This is much less the case in ISIC, for which satisfying results are already achieved using only one CL module. Conjointly, this dataset is the only one for which the 4-CL model achieves a similar accuracy than the C-FRPN. By observing that an unfolded RCL can be seen as a sequence of CLs sharing the same weights, it is clear that RCLs introduces extra depth as recursion occurs. Because we learn from this experiment that RCLs underperform on ISIC, we can make the assumption that in general case, the benefit of RCLs comes primarily from a capacity to compensate for the lack of depth in a network.

Therefore, we can determine that RCLs should perform better in dataset that requires large depth to be fitted, and are unlikely to improve the performances otherwise. Additionally, we saw that models with RCLs are more robust to the number of modules. Not only does this mean that models with much less weights can obtain similar performances, this also simplifies the design of convolutional networks, as covering the search space for the optimal number of modules can be made with more flexibility.

In the context of much deeper networks, as in ResNet and DenseNet presented earlier, introducing RCLs might have only limited effects due to the fact that these models have been designed with CLs specifically to be of very large depth. The benefits resulting from a further increase of depth saturates and might be alleviated in such deep architectures, which is why we observed that RCLs were not improving performances in this context. Therefore, one might rather consider introducing RCL in a shallower architectures than in very deep ones to maximize the



(a) Visual comparison

Model	CIFAR10	CIFAR100	SVHN	ISIC	CUB200
1 RCL	86.93 ± 0.31	56.95 ± 0.67	95.16 ± 0.16	**78.41** ± 0.38	21.03 ± 0.94
1 CL	71.53 ± 0.48	42.68 ± 0.24	70.72 ± 0.96	76.63 ± 0.37	10.49 ± 0.31
2 RCL	**89.46** ± 0.12	60.35 ± 0.55	96.02 ± 0.11	78.28 ± 0.33	**32.0** ± 0.82
2 CL	82.43 ± 0.6	52.9 ± 0.36	91.63 ± 0.14	78.4 ± 0.46	12.86 ± 0.37
4 RCL	89.18 ± 0.34	**61.13** ± 0.69	**96.28** ± 0.11	78.39 ± 0.54	29.48 ± 3.03
4 CL	86.6 ± 0.28	57.94 ± 0.32	94.62 ± 0.07	78.14 ± 0.52	19.95 ± 1.18

(b) Numerical details

Fig. 4. Comparison of the accuracy obtained by the respective shallow versions of the C-FRPN and the 4-CL model.

efficiency of RCLs. This idea will be applied to R-ResNet in Section 6.8 which address the progressive "ResNet-ification" of the C-FRPN.

This subsection revealed that RCLs are very efficient in shallow networks, but also that certain datasets are less likely to benefit from RCLs. The ISIC dataset is also among the smallest considered in the study. This could show that the RCL is particularly prone to overfit on smaller datasets, a concern raised in [1]. We investigate this question in the next subsection.

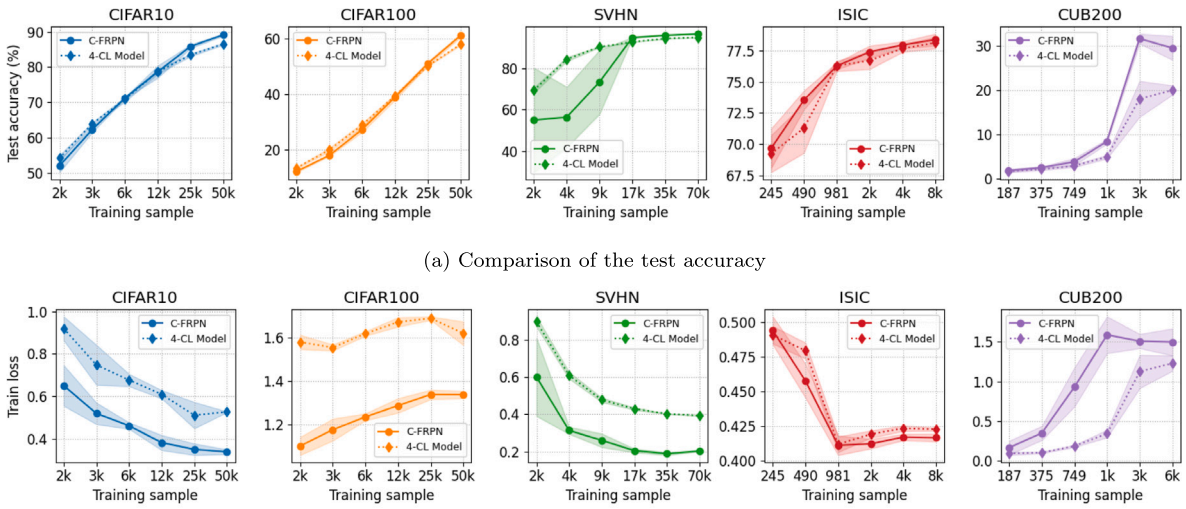
6.5. *RCLs are more subject to overfitting*

We observed in the previous subsections that the C-FRPN can make a more efficient use of the trainable parameters by re-using training samples at different stage. This implies that the C-FRPN creates a model that fits better the training samples. However, a better fit to the training samples often leads to overfitting and thus to reduced generalization capabilities. In order to investigate how the generalization capabilities of the C-FRPN are affected by overfitting, we trained the C-FRPN and the 4-CL model using 672k weights on undersampled versions of the training datasets and recorded the training loss and test accuracy in each case. This allows us to verify if the training loss is still representative of the general performances of the model.

Fig. 5 shows the result of training the models on undersampled versions of the training datasets. The undersampling is done randomly and differ for each run. We can make the following observations:

- Once again, the trends displayed on CIFAR10 and CIFAR100 are quite similar. When using fewer than 12.5% of the benchmark, the accuracy of the C-FRPN drops below the one of the 4-CL model, but this change is not reflected in the training loss. Hence, this shows that the C-FRPN can suffer more from overfitting than its feedforward counterpart on smaller datasets, resulting in lesser performances. On SVHN, the same effect can be observed, albeit in more drastic measure, where additionally the C-FRPN appears to become particularly unstable.
- In ISIC, the accuracy of the models appears to drop in a similar pace as the training set is reduced. Despite the instability resulting from the very small amount of training samples, the validation losses are relatively faithful to the trends observed in generalization.
- In CUB200, the drop in accuracy is more drastic in the C-FRPN, but both the training loss of the C-FRPN and the 4-CL model decreases as the size of the training set decrease. This shows that both models suffer heavily from overfitting, which was expected considering the low number of samples per class in the undersampled versions of the dataset.

We can conclude that in general, the C-FRPN suffers more from overfitting than its feedforward version. We also note that on ISIC, where the C-FRPN performed worse or similar to the latter, this was less evident. We hence confirm that overfitting is unlikely to be the main justification behind the poor performances of the C-FRPN on ISIC: the explanation is more likely to come from the nature of the task itself. We investigate in the next subsection, a possible reason by observing how recursion affects the accuracy of the models.



(a) Comparison of the test accuracy

(b) Comparison of the training loss

Fig. 5. Performance comparison of the C-FRPN and the 4-CL model on undersampled datasets, using [3.125%, 6.25%, 12.5%, 25%, 50%] of samples from the original training dataset. The horizontal axis is on a logarithmic scale.

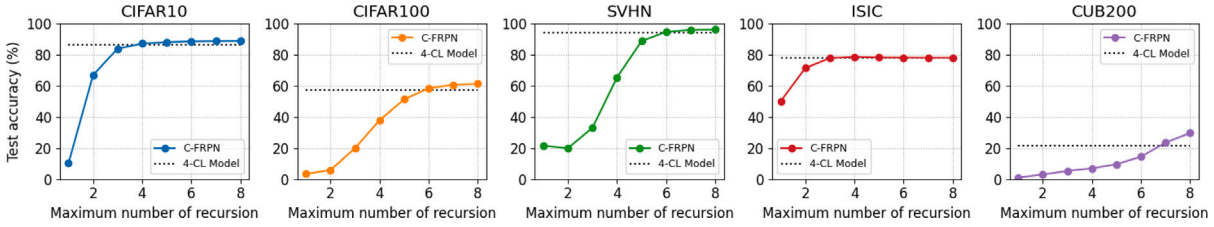


Fig. 6. Evolution of the test accuracy as the maximum number of recursion increases.

6.6. Effects of the number of recursions on the classification performance

This subsection focuses on C-FRPNs of size 672k to illustrate how RCLs are able to achieve better performances by taking benefit from recursion. Once fully trained, each model is evaluated using different values of the maximum number of recursions t_{max}, ranging from 1 to 8. We thus obtain the evolution of the accuracy of the model during recursion, which is displayed in Fig. 6.

We observe that in each dataset, the accuracy of the C-FRPN improves always monotonically under an increase of t_{max}. This shows that the feature maps are becoming increasingly informative for the classifier, until showing signs of saturation when t_{max} is large. The limit is reached at different time steps depending on the dataset, particularly early in ISIC and much later in CUB200. We note that when $t_{max} = 1$, the performances are hardly better than random guessing, and very far from those of CLs. This shows that the weights of RCLs and CLs are tuned very differently, and that the former learns to take benefit of recursion, as it shown by the increasing trend of the curves.

One could think that this effect explain why the RCL did not perform as well in R-ResNet and R-DenseNet, since a lower number of recursion was used in order to limit memory usage of the GPU. However, we tried halving the number of recursion in the C-FRPN and the 4-CL model and were able to observe the same effects as those reported with 8, and in the same range of accuracy. Using 8 recursion on R-ResNet and R-DenseNet did not bring any improvement either, which confirms the inefficiency of RCLs in those context.

This experiment showed that under the effect of recursion, the information contained in the feature maps that are produced by RCLs is gradually better processed by the classifier, as it is able to take advantage of this increased depth. It confirms that the optimal (virtual) network depth differs from learning problem to learning problem. Notably, it is revealed that a shallow network architecture suffices for the ISIC dataset whereas for another small dataset, the CUB200 dataset, a much deeper architecture is needed. We have thus shown that the cases in which RCLs perform worse than CL cannot just be linked to the size of the dataset. Our results instead imply that the benefits of RCLs could stem from the increased size of the receptive field, or from the ability to capture more diverse patterns. This will be investigated in the following subsection.

6.7. Weight diversity analysis

For each recursion happening inside an RCL, each filter interacts with the weights of all filters, including itself. These interactions are not explicitely present during the forward phase in convolutional layers. We show in this section that this leads to learning RCLs weights of larger diversity, which allows the C-FRPN to cover a broader spectrum of patterns. First, we introduce the distance measures used to evaluate the diversity of the weights in CL and RCL modules.

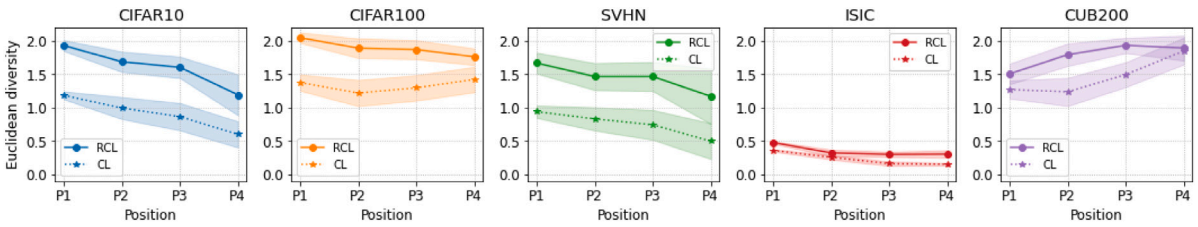
Evaluating weight diversity. We define a module-wise distance based on the average pairwise distance obtained for each possible couple of filters. Thus, if we consider a module M (either an RCL or a CL) and its filters F_i, then we have:

$$sim(M) = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} sim\left(F_i, F_j\right) \tag{8}$$

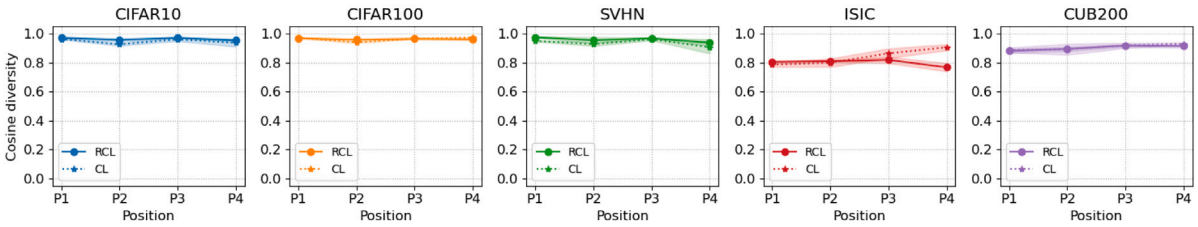
There are several ways to define the pairwise diversity metrics. We picked two of these based on their relevance and complementarity. Using two vectors u and v, we define:

- Euclidean distance:

$$sim_{euc}(u, v) = \frac{\|u - v\|_2}{\|u\|_2 \|v\|_2} \tag{9}$$



(a) Euclidean diversity



(b) Cosine diversity

Fig. 7. Average weight diversity per position.

- Cosine distance:

$$sim_{cos}(u, v) = 1 - \frac{u \cdot v}{\|u\|_2 \|v\|_2} \tag{10}$$

where $\|\cdot\|_2$ is the Euclidean norm and $u \cdot v$ is the scalar product between u and v. As a consequence an euclidean distance close to 0 means that weights have similar values and a cosine distance close to 0 means that weight vectors are spatially in the same direction. We see that the former focus on the range of the weights and the latter on their direction. Eventually both metrics point toward the same thing: the farther they are to 0 and the more the weight vectors differs. Using these metrics, we can now state that the higher $sim(M)$ is, the more the weights of the module can be considered diverse, i.e., not redundant.

The models uses the number of feature maps defined in the "Mixed configurations" category of the Table 1, for 672K overall parameters. To account for a fair comparison of the weight diversity, we restricted the number of filters used for the evaluation of diversity metrics of the CL and RCL to match the smallest possible number of filters inside a module, which is 83.

RCL learns more diverse weights. In the same way as in Section 6.2, we account for the diversity of modules from different positions by evaluating it for each of the 16 possible configurations.

Fig. 7 shows the average weight diversity according to the presence of an RCL or a CL at each position. The results reveals that the Euclidean diversity of RCLs is significantly larger in RCLs, whereas the cosine diversity is large and similar in both models, therefore the weights of RCLs are more diverse in term of range but are of similar direction with respect to CLs. Additionally we see that except in CUB200, RCLs and CLs have a larger Euclidean diversity when placed near the input, which could relate to the larger feature diversity of the input in early stages. Conversely in CUB200, the low inter-class deviations of the samples might necessitate weights of higher diversity at a later stage in the network. We have shown in Section 6.2 that RCLs where particularly beneficial in position $P3$, and we show in this section that it is the location for which the euclidean diversity is also the largest. This support the possibility that the euclidean weight diversity is strongly correlated with the performances of an RCL.

Models trained on ISIC are the only ones that show little difference in weight diversity and conversely the only ones for which the 4-CL model performs similarly to the C-FRPN. This can show us that RCLs are typically better when used in a context where the tasks requires high weight diversity. In such case, the larger weight diversity of RCLs allows to increase further the refinement process and hence to extract features that are of higher quality.

The relationship between weight diversity and the performances has been studied in many works. Ayinde et al. [42] defines a weight diversity metric in a very similar fashion as ours, and shows that regularizing the models in various ways to learn more diverse weights has a positive impact on performances, as long as it is not excessive. As in the models we studied here, they also showed that the increased weight diversity of these regularized models on CIFAR10 was mostly distributed at the beginning, which is further material to support that RCLs can be more effective in early layers in such tasks.

6.8. Effects of RCLs in ResNet

We found in the benchmark in Section 6.1 that RCLs were neither beneficial nor detrimental in ResNet and DenseNet, while they provide significant benefit in the 4-CL model. In our experiments, and in particular in Section 6.4 which considered shallower versions of the 4-CL model and the C-FRPN, we observed that RCL-ization was very efficient on shallower networks. This may provide a hint for us towards why RCL-ization do not benefit ResNet and DenseNet, since they are of considerably larger depth. However the depth is only one of the many differences between ResNet and the 4-CL model, and there are many other sources of discrepancy which might be responsible for the inhibited effects of RCL-ization. Therefore, we propose to gradually "ResNet-ify" the 4-CL model as well as the C-FRPN towards ResNet and R-ResNet respectively to observe how the benefit of RCL-ization are impacted by each transformation step. This allows us to pinpoint which R-ResNet characteristics are inhibiting RCLs, and to understand what makes ResNet to have a better performance than the 4-CL model used in this study. Note that we only study the effects of this process on the CUB200 datasets, as it is showing the largest discrepancy of results between the two architectures. A detailed description of each of the "ResNet-ification" steps is featured in Appendix B. Each new step also includes the changes made in the previous steps, irrespective of their performance, as in this study, we are more interested to see what the effect of introducing a particular step might be, rather than to use its performance to decide on if the step should be adopted, as such steps are already used in the ResNet, or the R-ResNet.

The outcome of this "ResNet-ification" process is depicted in Fig. 8. First, it can be observed that a large part of the performance improvement of the 4-CL model and the C-FRPN comes from the step 1 to 5,

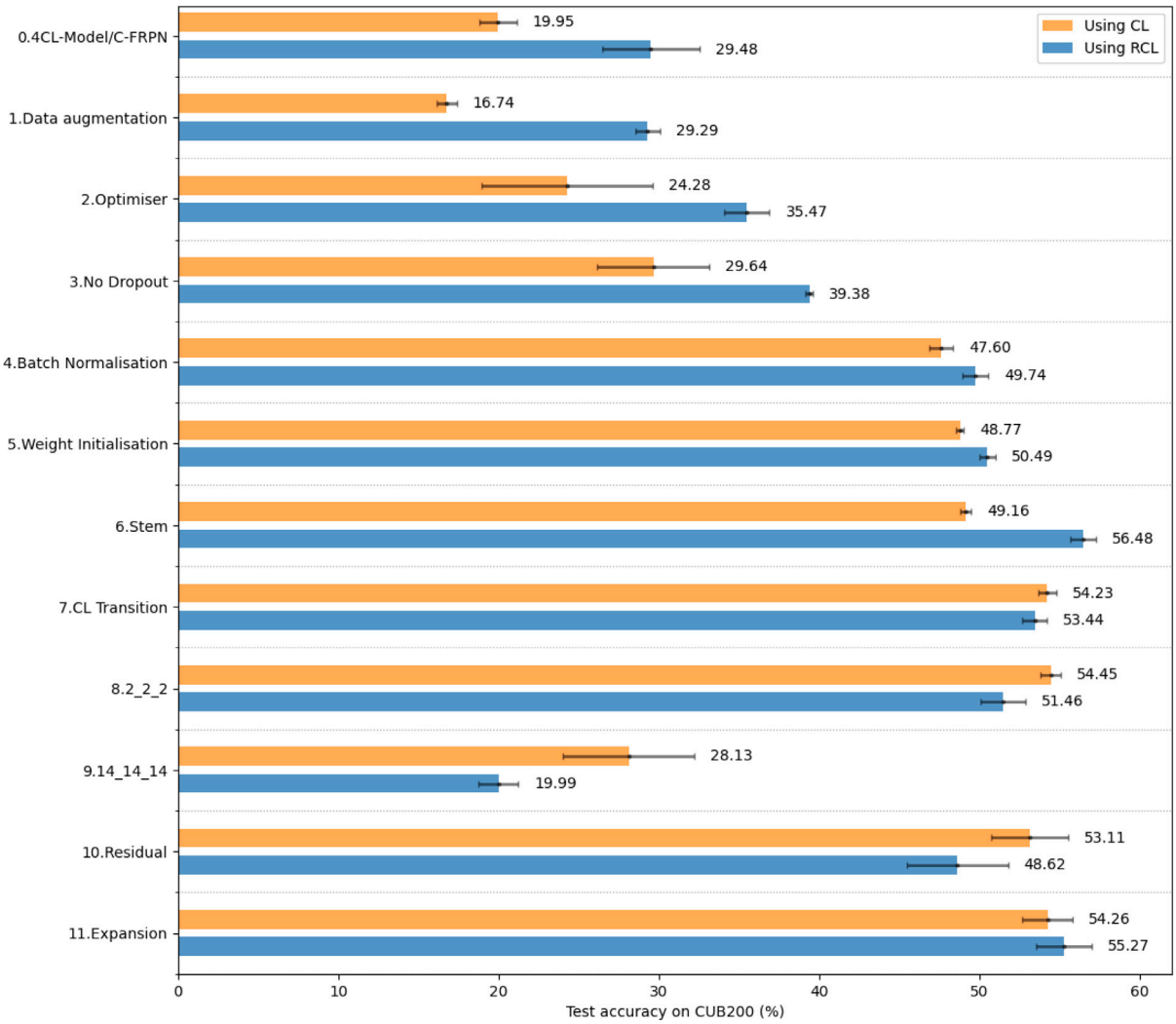


Fig. 8. Evolution of the accuracy resulting from the subsequent "ResNet-ification" steps applied to the 4-CL model (in orange) and the C-FRPN (in blue). The mean of 5 runs is displayed, and the horizontal lines shows the resulting standard deviation.

which are related to the different training schemes used by the 4-CL model and that of ResNet. Hence, the better performances of ResNet and R-ResNet when compared with the 4-CL model and the C-FRPN are largely driven by the training scheme used in ResNet. In comparison, the subsequent steps, which are of architectural nature, only account for about 6 point of the observed accuracy improvement. Specifically, we note the positive effect of residual connections after step 10. But it is noteworthy that the remarkable depth increase it permits does not actually enhance the performance of both the 4-CL model, or the C-FRPN. The transformed 4-CL model reaches its peak performance at step 8, and the C-FRPN at step 6 respectively.

Secondly, we observe that some features that are beneficial for the transformed 4-CL model are detrimental for the transformed C-FRPN. Those are visible at step 7 and 8, with respectively the usage of downsampling CL instead of max pooling and the introduction of two additional CLs (respectively two RCL) at stage 1. Additionally, step 9, which consists of matching the number of CL (respectively RCLs) as in ResNet (respectively R-ResNet) is detrimental for both, but is particularly harmful for the transformed C-FRPN. Each of these three steps involves using additional CLs or RCLs in the network, hence increasing its depth. This indicates that the positive effects resulting from an increase of depth are decreased when using RCLs. No other steps has been found to significantly hurt the benefits of RCL-ization. Therefore, this confirm that the large depth of ResNet is the main reason why RCL-ization has failed to improve the performances. Note that the performances of the transformed C-FRPN at step 6 exceeds those of ResNet, which shows that RCLs might still benefit ResNet when introduced more cautiously. It may be worthwhile to note that this step means to replace the stem, a CL by a RCL, something which has never been done on ResNet or R-ResNet previously. The reason why this has never been considered for ResNet or R-ResNet is that in the RCL-ization, we only consider the CLs inside a residual module. The stem contains only one CL, and therefore it is not a residual module. On the other hand, in the 4-CL model, the stages only contain CLs, and not residual blocks. Therefore, there is nothing to stop us to replace the stem by a RCL, and unfold it in the way similar to what is taken place in unfolding any CL inside a stage. Now from the experiments performed on finding out the properties of RCL-ization, it is found that relacing the CL close to the input end yields the most benefit. As the stem is the first in the 4-CL model, replacing this with a RCL, would obviously offer greater benefit than if only the first CL in stage 2 is RCL-ized. This observation while interesting, would need to be used in caution. This is because we often decide on if a new architecture is worth adopting by comparing its performance with those more established models, like ResNet. The RCL-ization of ResNet never considers the stem, and therefore this might not be a good comparison on equal basis. On the other hand this might be

useful if we wish to obtain more performance gain from an adopted architecture, then the idea of replacing the stem by a RCL might be an easy option.

This experiment has shown that a naïve RCL-ization of ResNet might be counter-productive. ResNet has been designed as a way to bypass the limitations associated with deep architectures and thus rely on a very large depth, whereas we found RCL to be particularly effective in contexts where a large depth is required. DenseNet being of even larger depth than Resnet, it is plausible that the underwhelming results of R-DenseNet suffer from that same aspect. Our results prompts speculation that for a much shallower model like the 4-CL model, if augmented, for example, by different types of CLs in parallel or in series, and by judicious selective RCL-ization, could potentially yield superior results when compared to the deeper networks encountered here.

Alternatively, the properties of RCL-ization might be applied to shorten state-of-the-art deep models, like ConvNeXt [43], by selectively RCL-ized its CLs in the ConvNeXt module (see Section 7 for more details on this idea).

This concludes the set of experiments conducted in this study. In the next section, we propose to summarize the results of our analysis and suggestions for further research.

7. Summary

This section summarizes the main contributions of this paper. This allows us to suggest where the knowledge gained from this paper fit in the wider area of research on finding a good model for small image datasets like those used in this paper.

Contributions.

- We found that RCL-ization of a shallow network, like the 4-CL model would be beneficial for small datasets
- We found that RCL-ization of a deep network, like ResNet-44, DenseNet-BC, which are specifically designed for small datasets, would not benefit their performance.
- We found a number of effects related to RCL-ization on the 4-CL model, including that it will benefit its performance for even shallower network than the 4-CL model.

How would we view these findings in light of the larger research area of designing good and efficient networks for datasets across a number of scales? From our findings, it is clear that for a shallow network, like the 4-CL model would benefit from RCL-ization as it can increase its depth through unfolding the recursive loop without incurring any increase in the number of weights, a situation which plagues the deep networks, like ResNet-44, or DenseNet-BC, even though they are specifically designed to handle small datasets, from their respective large models. This is because whenever the ResNet includes an extra residual block in a stage, its number of weights will increase. For the DenseNet, similar situation occurs, and that it is even worse in this case, because of the additional feedthrough connections of its preceding 2-CL modules. Moreover there is a growth factor involved in its design, and this growth factor contributes to the growth of number of channels in the ℓth 2-CL module, according to the formula: $k_0 + k \times (\ell - 1)$, where k_0 is the number of input channels at the input of the stage, and ℓ denotes the ℓth 2-CL module in the stage. This would mean, dependent on the growth factor, k, the number of channels at the output could be much higher than those in a ResNet. As it is commonly known, an increase in the number of channels means more trainable weights. Therefore the DenseNet is less memory efficient in its use of weights than ResNet, which in turn is less memory efficient than the C-FRPN model which receives its depth through the unfolding of RCLs. This might be the reason why RCL-ization is not beneficial to deep networks when designed to handle small datasets.

But there are other networks, like ConvNeXt [43], ConvNeXt v2 [44] which achieve state-of-the-art performance on small datasets. For example, the ConvNeXt is based on a ResNet backbone, but then evolves towards the ConvNeXt model, through a series of limited search in the architectural components space, guided by the design in the Swin Transformers [19], the design of ResNeXt network [10], and partly guided from experience; these include: changing the stage compute ratio, the location and type of activation functions, the location and type of normalization method used. The ConvNeXt model obtained in such a fashion consists of a stem, a number of stages, each stage contains a number of residual blocks, each residual block consists of a ConvNeXt module, in parallel with the direct feedthrough from the input to the output of the residual block. The ConvNeXt module consists of three blocks, a depthwise convolution followed by a 1×1 block followed by another 1×1 block, with a layerwise normalization between the depthwise convolution and the first 1×1 block, and a GeLU between the first 1×1 block and the second 1×1 block. Such a model has been found to have state-of-the-art performances on small datasets as well as on some large datasets. ConvNeXt v2 [44] is to explore the advantages offered by masked autoencoder [45] in efficiently using the limited amount of training data, by masking out part of the input image, and then reconstruct the input image through a self supervised learning method, in a style similar to the autoencoder. So the CovNeXt v2 essentially use the ConvNeXt model as an encoder, and the weights of the model are found by minimizing a reconstruction loss between the original image and the reconstructed image. In this manner, the encoder would be encouraged to extract relevant information from the input image rather than the redundant information which is often present in natural images. Again this model has been found useful particularly in situations when there are small number of training samples.

Would, say, a shallow network, e.g., the 4-CL model or its shallower variant, likely to have better performance than such sophisticated models, like ConvNeXt or ConvNeXt v2? The honest response would be no. But with what we know of the properties of RCL-ization, it is possible that by replacing some of the CLs by RCLs in the ConvNeXt model, or the ConvNeXt v2 model, especially when the one using a depthwise convolution, in the residual modules, which are close to the input end. This may reduce the number of residual blocks required in the stage, thus making more efficient use of the number of weights, this may improve the performance of this limited RCL-ized version of ConvNeXt from that of the ConvNeXt model. In other words, it might be possible to utilize the advantages offered by RCL-ization to render a ConvNeXt network less deep, and thus making better use of the number of weights available. This could be one way in which our research fits in the larger body of work on the design of high performance CNNs on small datasets, or across a number of scales. This we believe would be a worthwhile future direction of research.

Our results show that RCL-ization is useful for networks that are not very deep and small datasets. However, the optimal depth of an architecture depends on the dimension of the training set. Our results imply that the depth under which RCLs are useful may increase for larger datasets. Similarly, the effective depth may increase also by techniques that allows to make more efficiently use the available data, such as the self learning of mask autoecoders in ConvNeXt v2.

8. Conclusions

This paper investigated the effects of RCL-ization, i.e., the substitution of CLs with RCLs, on three base models: an 4-CL model with a stage compute ratio (0, 2, 2), ResNet-44 with a stage compute ratio (7, 7, 7), and DenseNet-BC with a stage compute ratio (16, 16, 16) on five small sized datasets. Our findings reveal that while RCL-ization can significantly improve the accuracy of the 4-CL model, its advantages are diminished when applied in deep architectures like ResNet-44 and DenseNet-BC. We then investigated why RCL-ization works so well on the 4-CL model. The investigation revealed general properties of RCLs, and provided explanations to why RCLs work well in shallow networks.

We then re-visited the first set of experiments by a ResNet-ification experiment: the transformation of the 4-CL model and the C-FRPN, through a step by step adoption to ResNet-44, and R-ResNet-44. We found that any attempt in increasing the depth of the 4-CL model, would produce detrimental performances, thus confirming the observation that RCL-ization would not be beneficial to a deep network like ResNet-44. Moreover, our results revealed that the C-FRPN can match the performance of much deeper networks by using a suitably adjusted training regime. We thus discovered that RCLs have the capacity to efficiently compensate for the lack of depth in a shallow network, like the 4-CL model as they allow for an increase in depth, when a RCL unfolds, without incurring additional weights; this can be much more favorable in shallower networks, like the 4-CL model, than the deeper models, like ResNet-44, or DenseNet-BC on small datasets.

From our results it can be expected that shallow RCL-based models could complement the performances of advanced networks e.g., ConvNeXt [43], ConvNeXt v2 [44], designed for small datasets which would be a good topic for future research.

Furthermore, in this paper, we are using a recursive feedback loop which implies that the dependency on its past values is constant. It would be very interesting to experiment with a gated recursive feedback loop, where the gating signal depends on some small time segment in the signals elsewhere in the network, thus simulating a variable recursive feedback loop (see a somewhat similar idea for recurrent neural networks in [16]), or alternatively explore the idea of a deformable kernel [38] in the recursive feedback, 'simulating' the effects of a time varying kernel. This gating signal idea or time varying kernel would particularly be suited for small datasets, as they allow for variations in the feedback loop according to the signal strength elsewhere in the network. Both of these would form areas of future research.

CRediT authorship contribution statement

Johan Chagnon: Conceptualization, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft. **Markus Hagenbuchner:** Conceptualization, Formal analysis, Methodology, Project administration, Resources, Supervision, Validation, Writing – review & editing. **Ah Chung Tsoi:** Conceptualization, Investigation, Methodology, Supervision, Validation, Writing – review & editing. **Franco Scarselli:** Conceptualization, Formal analysis, Methodology, Supervision, Validation, Writing – review & editing.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgments

The result presented in this paper has been supported by the Australian Research Council in the form of the Discovery Project Grant (DP210102674).

Appendix A. Details on the ResNet and DenseNet architecture used

To provide a better overview of the rest of the architecture used in the paper, we also detail explicitly their structure using block diagrams and characterize their depth. It can be observed that while we use as much downsampling layers, those architectures are considerably deeper than the backbone CNN and the C-FRPN used in our study.

ResNet44. We use the implementation described in [9] for the architecture that is specifically used for CIFAR10, that we represent in Fig. A.9. It contains a stem (in yellow), and 3 stages, respectively shown in blue, green and orange. In this representation, batch normalization and activation layers are not displayed, but they are detailed in Fig. A.10. Curved arrow represents residual connections, and the dotted curved arrows are residual connections between features of different dimensions. As in the original paper, those residual connections are made possible by transforming the input: extracting half of the spatial dimension, followed by 0-padding, which doubles the channel dimension.

The ResNet44 architecture can be detailed as follows:

- one stem which is made of one 3×3 CL.
- Three stages which contains 7 residual blocks each. Each blocks consists of two cascaded 3×3 CL, in parallel with a feedthrough connection from the input to the output of the block. The first CL of the first residual block of stage 2 and 3 applies spatial downsampling and doubles the number of channels as represented in Fig. A.10(b).
- a global average pooling layer followed by fully connected layer which acts as a classifier.

Hence the depth of the network is $1+3\times7\times2+1 = 44$. When RCL-izing ResNet44, we replace all but the first CL of the first block of stage 2 and stage 3, which are those that are responsible for spatial downsampling. Note that while residual blocks and the unfolding of an RCL both include a form of skip connections, important differences remain. In RCLs, skip connections go directly from the input to deeper CL layers, while it is summed in the case of residual connections. Additionally, the skip connections of ResNet go from one block to the next. Hence, ResNet simply the flow of information between two consecutive blocks, whereas in the case of RCLs, the skip connections aims to copy the information going from the input to deeper layers of the unfolding of RCLs. Therefore, residual connections and unfolding of a RCL should not be mistaken.

DenseNet-BC. Similarly as in ResNet44, the original DenseNet paper [11] includes the implementation details of some DenseNet architectures that are applied to CIFAR10, CIFAR100 and SVHN. It is represented in Fig. A.11 where we used the same block description as in Fig. A.9. The specific architecture we are using is DenseNet-BC ($k = 12$, $L = 100$), where k is the growth rate, i.e., the amount of channels outputed by each layer within a dense block and L is the total depth of the network. This particular configuration features the smallest number of parameters among those considered in their paper, approximately 769k when training on CIFAR10. This parameter count is also the closest to the range we are considering for the other models that we benchmarked, which spans between approximately 659k and 672k parameters. DenseNet-BC is a variant of DenseNet, which is implied when referring to DenseNet. DenseNet-BC is more weight efficient than DenseNet, as it obtains better performances using the same number of weights and can reach more depth thanks to two mechanisms:

- BottleNeck layers: In each layer within a dense block, every 3×3 CL is preceded by a 1×1 CL. This 1×1 CL alters the number of channel of the input to $k\times4 = 12\times4 = 48$. Hence, the information obtained by concatenating the output of the preceding layers is either extended or compressed to a fixed number. This not only homogenizes the distribution of channels preceding each 3×3 CL but also significantly reduces the number of channels considered when more than 4 layers are involved in the dense block.
- Compression: Prior to the spatial pooling in the transition layers, a 1×1 CL is introduced that compresses the number of channels in the input by halving it. This leads to a drastic reduction in the number of parameters flowing through the network, allowing for considering a larger number of layers in each dense block.

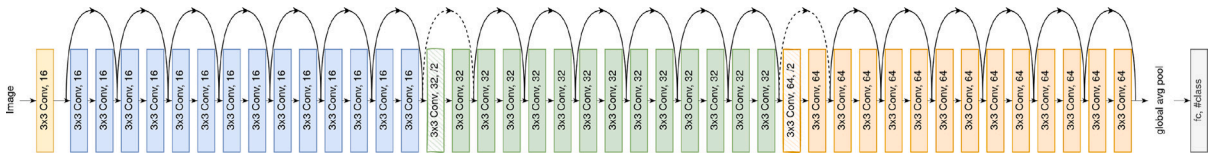


Fig. A.9. The ResNet44 architecture. In each block, the first item represents the type of module and the second the number of channel dimension of the output. CL responsible for pooling uses a stride of 2, which halves the spatial dimension. The description of those CL includes '/2' to reflect their nature.

(a) Residual connection

(b) Residual Block used for stage transition.

Fig. A.10. Details on the Residual Bocks of ResNet. The $\oplus$ symbol represents pointwise addition of the features.

Hence the name DenseNet-BC, where "B" and "C" stands for bottleneck and compression respectively. We can detail the full architecture of DenseNet-BC as follows:

- one stem which is made of one 3×3 CL.
- the first residual block which contains 16 layers. Each layer consists of a 1×1 CL followed by a 3×3 CL, and every layer receives input from all preceding layers within the block, including the input to the block itself.
- a first transition layer consisting of a 1×1 CL and an average pooling layer.
- the second residual block which contains 16 layers.
- a second transition layer consisting of a 1×1 CL and an average pooling layer.
- the third residual block which contains 16 layers.
- a global average pooling layer followed by fully connected layer which acts as a classifier.

In our study, we are using the parameter $L = 100$, which corresponds to a depth of 100. As an equal number of layer is considered within each stage, this depth is achieved by using a stage compute ratio of $[16, 16, 16]$. Indeed, when considering a stage made of 16 layers, it involves 32 parametric CLs, so the depth of each block is 32. DenseNet-BC is made of a stem, three dense blocks, two transition layers, and one fully connected layer. Each transition layer includes one 1×1 Convolutional Layer (CL). Therefore, the overall depth of the network is calculated as follows: $L = 1 + 32 \times 3 + 2 \times 1 + 1 = 100$. Using a stage compute ratio of $[16, 16, 16]$ imply that each subsequent stage will contains a higher number of channels. This is because the concatenation of the channels of the 16 layers more than doubles the number of channels of the input of the block, while the transition layer halves it, as can be observed in Fig. A.11(a). This is a notable difference compared to ResNet in which each stage uses exactly doubles the number of channels of the previous one.

The RCL-ization of DenseNet that we considered consist in replacing every 3×3 CL of the network with a 3×3 RCL. Therefore, we do not replace the 1×1 CL used in the bottleneck and the 1×1 CL used in the compression.

Appendix B. Details on the "ResNet-ification" steps

In the process to transform the 4-CL model and the C-FRPN into respectively ResNet and R-ResNet, 11 steps are incrementally adopted. We proceed to provide a description of each of them.

Step 0: 4-CL model/C-FRPN. We start with the 4-CL model and the C-FRPN

Step 1: Data augmentation. We adopt the same data augmentation scheme as the one used in ResNet [9]. This means that the images are downscaled to be 112×112 from 224×224 and the training images are 0-padded to allow for random cropping.

Step 2: Optimiser. We use the same optimizer and learning rate scheduler as those used in ResNet [9]. This implies that SGD is being used alongside a learning rate which is no longer fixed, but divided by 10 at 50% and 75% of training.

Step 3: No dropout. The Dropout layers are discarded.

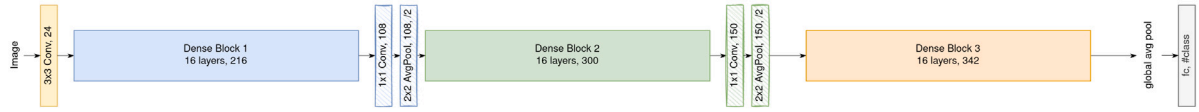
Step 4: Batch normalization. Batch normalization is used instead of Local Response Normalization. Biases are removed for CL and RCL layers, as it becomes redundant with batch normalization.

Step 5: Weight initialization. Similarly to ResNet, the Kaiming initialization [46] is being applied on the convolutional layers and on the linear layer.

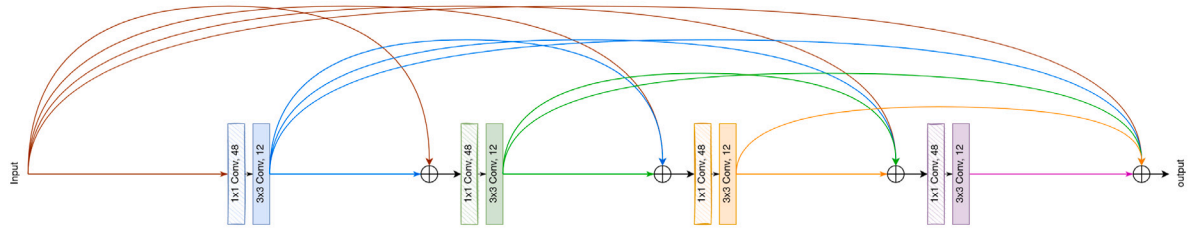
Step 6: Stem. The stem is changed to be a kernel of size 3×3 instead of 5×5. In the C-FRPN to R-ResNet transformation, the stem is also changed to be an RCL.

Step 7: CL transition. For downsampling, we replace maxpooling layers with CLs using a stride of 2.

Step 8: 2_2_2. We add 2 additional CL (respectively RCL) before the first downsampling layer. At this step if we exclude downsampling layers, we can represent the current model configuration as containing [2, 2, 2] CL (respectively RCL) per stage.



(a) The DenseNet-BC architecture (k=12, L=100)



(b) A DenseBlock of 4 layers. Note that the number of layers used in each block of DenseNet-BC (k=12, L=100) is actually 16. The $\bigoplus$ symbol represents concatenation of the channels.

Fig. A.11. Block diagram of the DenseNet architecture used and one of its DenseBlock.

Step 9: 14_14_14. In the case of the transformed 4-CL model, we use [14, 14, 14] CL per stage including the downsampling layers. In the case of the C-FRPN, we use [12, 11, 5] RCL per stage, and two CL for the spatial downsampling layers. Hence, those two configurations are identical to those of ResNet and R-ResNet described in Table 2 except that no residual connections are used.

Step 10: Residual connections. Residual connections, as described in Appendix A, are added.

Step 11: Expansion. We double the number of feature maps used after each CL downsampling layers. This step conclude the transformation. The models produced follows exactly the same architecture and training scheme as ResNet and R-ResNet.

Note that steps 6–9 and step 11 may introduce a significant number of additional parameters. To address this issue, we reduce the number of feature maps per layers (or the base number in the case of step 11) in both models transformation to preserve a similar number of parameters and ensure a fair comparison. The ordering of these steps has been determined in such a way that it favor computational efficiency and separate changes that are related to the training scheme (step 1–5) from those related to architectural changes (step 6–11). We acknowledge that the order of the steps might influence the perceived benefit of the corresponding feature but this coarse separation between training-related and architecture-related changes may still provide an indication of the general trends.

References

[1] A. Rossi, M. Hagenbuchner, F. Scarselli, A.C. Tsoi, A study on the effects of recursive convolutional layers in convolutional neural networks, Neurocomputing 460 (2021) 59–70.

[2] L. Alzubaidi, J. Zhang, A.J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M.A. Fadhel, M. Al-Amidie, L. Farhan, Review of deep learning: concepts, CNN architectures, challenges, applications, future directions, J. Big Data 8 (53) (2021) 1–74.

[3] M.M. Taye, Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions, Computation 11 (52) (2023) 1–23.

[4] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, F. Hutte, Neural architecture search: Insights from 1000 papers, 2023, arXiv:2301.08727.

[5] K.T. Chitty-Venkata, M. Emani, V. Vishwanath, A.K. Somani, Neural architecture search benchmarks: Insights and survey, IEEE Access 11 (2023) 25217–25236.

[6] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: F. Pereira, C. Burges, L. Bottou, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc., 2012.

[7] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: 3rd International Conference on Learning Representations (ICLR 2015), Computational and Biological Learning Society, 2015, pp. 1–14.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2015, pp. 1–9.

[9] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 770–778.

[10] S. Xie, R. Girshick, P. Dollár, Z. Tu, K. He, Aggregated residual transformations for deep neural networks, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017, pp. 5987–5995.

[11] G. Huang, Z. Liu, L.V.D. Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, IEEE Computer Society, Los Alamitos, CA, USA, 2017, pp. 2261–2269.

[12] Y. Bengio, P.Y. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (2) (1994) 157–166.

[13] Z. Wu, C. Shen, A. van den Hengel, Wider or deeper: Revisiting the ResNet model for visual recognition, 2019, pp. 119–133.

[14] K.S. Rockland, Notes on visual cortical feedback and feedforward connections, Front. Syst. Neurosci. 16 (784310) (2022) 1–5.

[15] M. Liang, X. Hu, Recurrent convolutional neural network for object recognition, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2015, pp. 3367–3375.

[16] A. Nayebi, D. Bear, J. Kubilius, K. Kar, S. Ganguli, D. Sussillo, J.J. DiCarlo, D.L. Yamins, Task-driven convolutional recurrent models of the visual system, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 31, Curran Associates, Inc., 2018.

[17] S. Lawrence, C.L. Giles, A.C. Tsoi, A.D. Back, Face recognition: a convolutional neural-network approach, IEEE Trans. Neural Netw. 8 (1) (1997) 98–113.

[18] C. Tang, Y. Zhao, G. Wang, C. Luo, W. Xie, W. Zeng, Sparse MLP for image recognition: Is self-attention really necessary? in: The Association for the Advancement of Artificial Intelligence (AAAI) Conference, 2022.

[19] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, B. Guo, Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[20] H. Zhang, J. Duan, M. Xue, J. Song, L. Sun, M. Song, Bootstrapping ViTs: Towards liberating vision transformers from pre-training, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 8944–8953.

[21] A.H. Farzaneh, X. Qi, Facial expression recognition in the wild via deep attentive center loss, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021, pp. 2402–2411.

[22] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, Q. Hu, ECA-Net: Efficient channel attention for deep convolutional neural networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 11534–11542.

[23] P. Mangla, N. Kumari, A. Sinha, M. Singh, B. Krishnamurthy, V.N. Balasubramanian, Charting the right manifold: Manifold mixup for few-shot learning, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2020, pp. 2218–2227.

[24] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, A. Courville, D. Lopez-Paz, Y. Bengio, Manifold mixup: Better representations by interpolating hidden states, in: International Conference on Machine Learning, 2019, pp. 6438–6447.

[25] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, S. Xie, A ConvNet for the 2020s, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 11976–11986.

[26] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017, arXiv preprint arXiv:1704.04861.

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.
[28] M. Hagenbuchner, A.C. Tsoi, F. Scarselli, S.J. Zhang, A fully recursive perceptron network architecture, in: 2017 IEEE Symposium Series on Computational Intelligence, SSCI, 2017, pp. 1–8.
[29] F. Scarselli, A.C. Tsoi, Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results, Neural Netw. 11 (1998) 15–37.
[30] M. Gori, F. Scarselli, Are multilayer perceptrons adequate for pattern recognition and verification? IEEE Trans. Pattern Anal. Mach. Intell. 20 (11) (1998) 1121–1132.
[31] Y. LeCun, C. Cortes, MNIST handwritten digit database, 2010.
[32] A. Krizhevsky, G. Hinton, Learning Multiple Layers of Features from Tiny Images, Tech. Rep., University of Toronto, Toronto, Ontario, 2009.
[33] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A.Y. Ng, Reading digits in natural images with unsupervised feature learning, in: NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011, 2011.
[34] N.C.F. Codella, D. Gutman, M.E. Celebi, B. Helba, M.A. Marchetti, S.W. Dusza, A. Kalloo, K. Liopyris, N. Mishra, H. Kittler, A. Halpern, Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (ISBI), hosted by the international skin imaging collaboration (ISIC), in: 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018), 2018, pp. 168–172.
[35] J. Wang, X. Hu, Convolutional neural networks with gated recurrent connections, IEEE Trans. Pattern Anal. Mach. Intell. 44 (07) (2022) 3421–3435.
[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255.
[37] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A.L. Yuille, DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs, IEEE Trans. Pattern Anal. Mach. Intell. 40 (4) (2018) 834–848.
[38] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, Y. Wei, Deformable convolutional networks, in: 2017 IEEE International Conference on Computer Vision, ICCV, 2017, pp. 764–773.
[39] B. Cassidy, C. Kendrick, A. Brodzicki, J. Jaworek-Korjakowska, M.H. Yap, Analysis of the ISIC image datasets: Usage, benchmarks and recommendations, Med. Image Anal. 75 (2022) 102305.
[40] C. Wah, S. Branson, P. Welinder, P. Perona, S. Belongie, Caltech-UCSD Birds-200–2011, Tech. Rep. CNS-TR-2011-001, California Institute of Technology, 2011.
[41] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML '15, JMLR.org, 2015, pp. 448–456.
[42] B.O. Ayinde, T. Inanc, J.M. Zurada, Regularizing deep neural networks by enhancing diversity in feature extraction, IEEE Trans. Neural Netw. Learn. Syst. 30 (9) (2019) 2650–2661.
[43] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, S. Xie, A ConvNet for the 2020s, in: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2022, pp. 11966–11976.
[44] S. Woo, S. Debnath, R. Hu, X. Chen, Z. Liu, I.S. Kweon, S. Xie, ConvNeXt V2: Co-designing and scaling ConvNets with masked autoencoders, in: The 36th Conference on Computer Vision and Pattern Recognition, CVPR, 2023.
[45] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, R. Girshick, Masked autoencoders are scalable vision learners, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 16000–16009.
[46] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, in: Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ICCV '15, IEEE Computer Society, USA, 2015, pp. 1026–1034.

Johan Chagnon received a "diplôme d'ingénieur" in computer science and applied mathematics in 2020 from the National School of Computer Science for Industry and Business (ENSIIE) at Évry-Courcouronnes, France. This degree is internationally recognized as a combined Bachelor's/Master's (BS/MS) in Engineering. In parallel, he achieved a Master's Degree in data science at Université Paris Saclay, Paris, France. He is currently a Ph.D. student working in the University of Wollongong, Australia in the field of deep learning and computer vision.

Markus Hagenbuchner is an associate professor at the School of Computer Science and Information Technology of the University of Wollongong UoW. He received the Ph.D. degree in Computer Science from the UoW in 2002. His main research interests are in machine learning, with emphasis on the development of novel learning algorithms, particularly with respect to neural networks for structured data, deep learning and their applications. He is a director of the Center of Artificial Intelligence at UoW. He has led numerous international research projects on machine learning for which he was awarded major funds from the Australian Research Council.

Ah Chung Tsoi received the Higher Diploma degree in electronic engineering from the Hong Kong Technical College, Hong Kong, in 1969 and the M.Sc. degree in electronic control engineering and the Ph.D. degree in control engineering from University of Salford, Salford, U.K., in 1970 and 1972, respectively. He was a Postdoctoral Fellow at the Inter-University Institute of Engineering Control at University College of North Wales, Bangor, North Wales and a Lecturer at Paisley College of Technology, Paisley, Scotland. He was a Senior Lecturer in Electrical Engineering in the Department of Electrical Engineering, University of Auckland, New Zealand, and a Senior Lecturer in Electrical Engineering, University College University of New South Wales, Australia, for five years. He then served as Professor of Electrical Engineering at University of Queensland, Australia; Dean, and simultaneously, Director of Information Technology Services, and then foundation Pro-Vice Chancellor (Information Technology and Communications) at University of Wollongong, before joining the Australian Research Council as an Executive Director, Mathematics, Information and Communications Sciences. He was Director, Monash e-Research Centre, Monash University, Melbourne, Australia. In April 2007, became a Vice President (Research and Institutional Advancement), Hong Kong Baptist University, Hong Kong. In recent years, he has been working in the area of artificial intelligence in particular neural networks and fuzzy systems. He has published in neural network literature. Recently, he has been working in the application of neural networks to graph domains, with applications to the world wide web searching, and ranking problems, and subgraph matching problems.

Franco Scarselli received the Laurea degree with honors in computer science from the University of Pisa, Pisa, Italy, in 1989 and the Ph.D. degree in computer science and automation engineering from the University of Florence, Florence, Italy, in 1994. He has been supported by foundations of private and public companies and by a postdoctoral of the University of Florence. In 1999, he moved to the University of Siena, Siena, Italy, where he was initially a Research Associate and is currently an Associate Professor at the Department of Information Engineering. He is the author of more than 50 journal and conference papers and has been involved in several research projects, founded by public institutions and private companies, focused on software engineering, machine learning, and information retrieval. His current theoretical research activity is mainly in the field of machine learning with a particular focus on adaptive processing of data structures, neural networks, and approximation theory. His research interests also include image understanding, information retrieval, and web applications.