This is a repository copy of *Deep Residual Compensation Convolutional Network without Backpropagation*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/198786/

Version: Accepted Version

Proceedings Paper:

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Deep Residual Compensation Convolutional Network without Backpropagation

1st Mubarakah Alotaibi
Computer Science
University of York, Taif University
York, UK, Taif, Saudi Arabia
mmma512@york.ac.uk

2nd Richard C Wilson
Computer Science
University of York
York, UK
richard.wilson@york.ac.uk

Abstract—**PCANet and its variants provided good accuracy results for classification tasks. However, despite the importance of network depth in achieving good classification accuracy, these networks were trained with a maximum of nine layers. In this paper, we introduce a residual compensation convolutional network, which is the first PCANet-like network trained with hundreds of layers while improving classification accuracy. The design of the proposed network consists of several convolutional layers, each followed by post-processing steps and a classifier. To correct the classification errors and significantly increase the network's depth, we train each layer with new labels derived from the residual information of all its preceding layers. This learning mechanism is accomplished by traversing the network's layers in a single forward pass without backpropagation or gradient computations. Our experiments on four distinct classification benchmarks (MNIST, CIFAR-10, CIFAR-100, and TinyImageNet) show that our deep network outperforms all existing PCANet-like networks and is competitive with several traditional gradient-based models.**

Index Terms—**PCANet, DCTNet, DCCNet, CCANet, OSNet, LDANet, Multi-layer PCANet, Image Classification**

I. INTRODUCTION

A. Background and Related Works

Deep learning is a non-task-specific learning technique that uses hierarchical structures to automatically learn representations from raw data [1]. Since Alexnet [14] won the 2012 ImageNet challenge, convolutional neural networks (CNNs) have been widely used for image classification with great success. VGG [19], ResNet [10] and Network in network [16] are a few examples of standard CNNs. Despite their success, CNNs are usually trained with a large number of parameters, requiring intensive parameter updates and a lot of data for training, which might increase the computational cost even with GPU-equipped computing resources [7].

Chan et al. [3] presented PCANet as an alternative to deep CNNs for classification tasks on small datasets. The network structure comprises two cascaded principal component analysis (PCA) layers followed by binary hashing, block-wise histogram and a classifier. Unlike CNNs, PCANet learns its filter bank in a non-iterative layer-by-layer fashion through PCA applied to image-based patches. This training mechanism provides a faster training time advantage to PCANet over conventional CNNs. With its simple architecture, PCANet has achieved state-of-the-art performance across several datasets, including MNIST, extended Yale B, AR and FERET.

The success of PCANet has inspired a family of related strategies. Generally speaking, PCANet-related research has fallen into one of two categories: those focusing on improving the features representation process or those attempting to increase network depth. The work in this paper fits within the second category.

In an effort to enhance PCANet's feature representation, several articles studied networks that maintained PCANet's fundamental structure while producing different features using different convolutional filters. DCTNet [18], CCANet [20] and ICANet [22] are examples of PCANet-like networks that create their filter banks using unsupervised approaches. The LDANet [3], DCCNet [6] and OSNet [8] are a few examples of those that use supervised approaches to produce their filter banks. DFSNet [7] is a good example of a network that uses semi-supervised filters.

In order to increase the network depth, several studies have attempted to address PCANet's primary issue, the features explosion problem, which limits its depth to only two layers. The block-wise histogram is one factor that contributes to this problem, as the number of bins required to calculate the histogram features grows exponentially with the number of filters in the second stage. Fan et al. [5] significantly reduced PCANet's features by replacing the histogram pooling with second-order pooling. The per-channel convolution mechanism is another factor that contributes to the increasing dimensionality problem in PCANet. PCANet+ [17] provided an alternative solution to this issue by proposing a filter ensemble mechanism that aggregates the feature maps over all channels, similar to CNNs. By adopting CNN-like filters, as in [17], replacing PCANet's binarisation step with the z-score method and using second-order pooling and late fusion, Alotaibi and Wilson [2] were able to expand PCANet depth to nine layers and outperform the original design. In contrast to prior studies that aimed to increase network depth by tackling the dimensionality issue, our objective is to explore the design of a hundreds-layer PCANet-like network by optimising the classification process at each layer.

B. Proposed Network

In this article, we present a residual compensation convolutional learning framework to achieve accuracy from a considerably increased depth while simultaneously correcting network errors as we traverse it. The proposed network inherits the simplicity of PCANet-like networks and is trained in a single forward-pass without gradient computations or backpropagation. A comprehensive description of the network architecture, along with its training procedures, is described in Section II. The experimental section (III) consists of three subsections. In Section III-B, we evaluate the performance of the proposed network on MNIST, CIFAR-10, CIFAR-100 and TinyImageNet against gradient-based and non-gradient architectures without data augmentation. Section III-C examines the influence of several network parameters, such as the number of filters (III-C1) and the learning rate (III-C2), on the accuracy of the proposed model. The final subsection (Section III-D) addresses the use of data augmentation to improve the model's accuracy. Finally, the conclusions and future works are discussed in Section IV.

II. NETWORK ARCHITECTURE

A. Problem Formulation

Consider a classification problem with N training images $X^{(1)} \in \mathbb{R}^{m \times n \times d \times N}$, where m and n are the images' spatial dimensions, and $d \in \{1, 3\}$ is the number of channels for greyscale or colour images. Let T represent the C classes to which the images originally belonged. The objective of our deep residual compensation convolution network (ResCNet) is to construct a PCANet-like network that achieves high accuracy from a considerable network depth and is trained without using gradient descent or backpropagation. Precisely, the network structure should be structured hierarchically such that, as network depth increases, succeeding layers compensate for the classification errors of previous layers. By combining the predicted probabilities of the network layers, the model should obtain a high accuracy.

B. Design Overview

The network architecture, as shown in Figure 1, consists of multiple convolutional layers, each followed by post-processing steps and a linear discriminant analysis (LDA) classifier. Each convolutional layer, except the first layer, receives as input the concatenation of its previous layer's outputs and the original features. After post-processing the feature maps of the first layer, an LDA classifier is trained to categorise the extracted features using the original classes. By contrast, the LDA classifiers of the subsequent layers, referred to as compensation layers, are trained using new classes learnt from the residual information of the preceding layers. In fact, each layer produces two outputs ($\tilde{Y}^{(i)}$ and $T^{(i+1)}$ in Figure 1, where i represents the ith layer). The first provides the predicted probabilities of the network at that layer, whereas the second represents the new classes produced for training the subsequent layer. To produce the network's outputs in each layer ($\tilde{Y}^{(i)}$ in Figure 1), the predicted probabilities of that layer are added to or subtracted from those of its preceding layers. This combination mechanism maintains the network's predicted probabilities of being in the range of between 0 and 1 and reduces the error of preceding layers. Section II-C describes the network's components, including the convolutional layers (Section II-C1), post-processing procedures (Section II-C2) and residual mechanism (Section II-C3).

C. Network Components

1) Convolutional Layers: Let $X^{(1)} \in \mathbb{R}^{m \times n \times d \times N}$ represent N training images, T is their C classes, and $O^{(i)} \in \mathbb{R}^{m \times n \times d_i \times N}$ represents the feature maps produced by the ith layer. The $i + 1$ layer receives as input the output of the ith layer concatenated with the original features; this input is represented as $X^{(i+1)} \in \mathbb{R}^{m \times n \times (d+d_i) \times N}$. We then divide the input images ($X^{(i+1)}$) into patches of $k \times k$ size and a stride of 1 pixel, where k is the filter size and is a user-predefined parameter. The resulting matrix is $P \in \mathbb{R}^{k^2(d_i+d) \times \tilde{m}\tilde{n}N}$, where $\tilde{m} = (m-k)+1$, $\tilde{n} = (n-k)+1$ and m and n are the width and the height of the images, respectively. The filter learning process relies on applying any non-gradient-based method to the extracted local patches P and collecting their weights to represent the filters used in the $i+1$ layer. The PCA filter bank by Low, Teoh, and Toh [17] is an example of such filters. The authors first centralised the extracted patches to their mean to obtain $\bar{P} \in \mathbb{R}^{k^2(d_i+d) \times \tilde{m}\tilde{n}N}$. They then applied PCA to the centralised patches, where the principal components of $\bar{P}\bar{P}^T$ can be computed by solving the following optimisation problem:

$$\min_{V \in R^{(k^2) \times (d_i+d)}} ||\bar{P} - VV^T\bar{P}||_F^2,$$
$$s.t. \ V^TV = I, \tag{1}$$

where I represents the identity matrix. The convolutional output of the $i + 1$ layer can then be expressed as follows:

$$O^{(i+1)} = X^{i+1} * W^{(i)} \in \mathbb{R}^{m \times n \times d_{i+1}}, \tag{2}$$

where d_{i+1} represents the number of filters in layer $i + 1$, $X^{(i+1)}$ is the input images that are zero padded to obtain the same image size outputs, $*$ represents the convolution operation, and $W^{(i)}$ denotes the PCA filters expressed as follows:

$$W^{(i)} = \max_{k \times k \times (d_i+d)} q_s, \ s = 1, 2, \ldots, d_{i+1}, \tag{3}$$

where q_s is the sth principal eigenvector of $\bar{P}\bar{P}^T$.

The stacked-LDA filter bank is another example of filters generated without relying on gradient descent or backpropagation. Appendix A discusses these filters in detail. These filters are computed using an iterative process that involves selecting a subset of the localised patches P and then applying an LDA classifier to train the selected patches with their classes. The algorithm searches for patches with separable classes and accumulates their weights, which are subsequently used as stacked-LDA filters.

This article focuses mainly on the network architecture rather than investigating the filter type used. In our
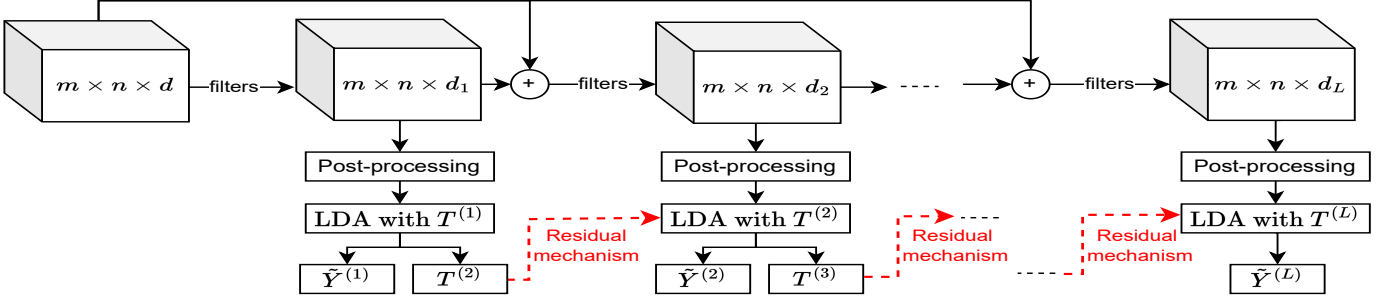
Fig. 1. The deep residual compensation convolutional network architecture. $T^{(1)}$ represents the original classes.

experiments (Section III), we use semi-supervised stacked-LDA filters created by combining 50% of the supervised stacked-LDA filters with 50% of the unsupervised PCA filters. However, different filter types can be employed, such as those used by Ng and Teoh [18], Yang et al. [20] and Gatto et al. [7].

2) Post-processing Steps: The feature maps of each convolutional layer are post-processed using a ReLU non-linear activation function, followed by second-order pooling and a multi-level spatial pyramid pooling. The ReLU function is applied to the feature maps of each layer but not between the layers. The feature maps are then pooled locally using the second-order pooling mechanism described by Alotaibi and Wilson [2]. Let $X_j^{(i)} \in \mathbb{R}^{m \times n \times d_i}$ denote the jth activations map in the ith layer, where m and n are the spatial dimensions of the images, and d_i represents the number of filters in layer i. The calculation of the second-order pooling starts by dividing the tensors of $X_j^{(i)}$ into patches of the same size, which could be overlapped, e.g. $(r \times c)$. Each of these patches is then normalised using the z-score method, defined as $z = \frac{x-\mu}{\sigma}$, where μ and σ represent the mean and the standard deviation of the data. Next, the channel-wise covariance matrix of each patch, after reshaping each of them to $rc \times d_i$, is computed. Because of the symmetry property of the covariance matrix, the number of second-order features is calculated as the number of patches $\times (\frac{d_i \times d_i}{2} + \frac{d_i}{2})$. The multi-level spatial pooling is then used to pool the second-order features. The multi-level spatial pyramid pooling calculation is identical to that implemented by Chan et al. [3]. Again, different post-processing procedures can be utilised; however, in our experiments (Section III), we found these steps to be the most effective for achieving high accuracy in the databases we used.

3) Residual Mechanism: This mechanism is non-iterative; we add layers sequentially. Each layer is trained in a single pass with new labels learnt from the residual information of all its previous layers. The first layer's features are classified using an LDA classifier trained with the original classes. To produce the first layer's posteriors ($\tilde{Y}^{(1)}$ in Figure 1), we use the following sigmoid function on the output of the LDA classifier:

$$S(x) = \frac{1}{1 + e^{-x/\sigma}}, \tag{4}$$

where σ is the sigmoid scale parameter. To identify the new labels required to train the second layer ($T^{(2)}$ in Figure 1), we first find the residual errors between the current layer's predicted outputs ($\tilde{Y}^{(1)}$) and the original classes ($Y \in \mathbb{R}^{N \times C}$) in one-hot encoding, as follows:

$$R^{(1)} = \lambda Y - \tilde{Y}^{(1)}, \tag{5}$$

where $0 \leq \lambda \leq 1$ controls the maximum likelihood a class may attain. $T^{(2)}$ can then be defined as the classes with the maximum absolute residual errors, as follows:

$$T_i^{(2)} = \text{class}(|R_{ij}^{(1)}|, \forall j) \quad i = [1, \dots, N], \tag{6}$$

where $\text{class}(x)$ denotes the name of the class whose value has the largest residual error magnitude. Since our network is developed primarily for classification tasks, it concentrates on labelling rather than regressing the correction of posteriors.

The second layer, as shown in Figure 1, receives an input $X^{(2)} \in \mathbb{R}^{m \times n \times (d+d_1) \times N}$ that is a concatenation of the first layer's feature maps and the original images. After finding the second layer's features, our objective is to learn a correction term based on the second layer classification results, which is then added to the posteriors of the first layer ($\tilde{Y}^{(1)}$) to provide more accurate predictions ($\tilde{Y}^{(2)}$). The correction term can either be positive or negative to maintain the network probabilities at the second layer ($\tilde{Y}^{(2)}$) to have values in the range of 0 to λ (Equation 5). In other words, after training the second layer's LDA classifier using the new classes ($T^{(2)}$), the posteriors acquired can be added to or subtracted from the first layer's posteriors to correct them. The indicator variable ($s^{(2)}$) indicates whether to add to or subtract from the first layer's posteriors and is defined as the signum function of the maximum absolute residual errors of the previous layers, as follows:

$$s_i^{(2)} = \text{sign}(R_{i*}^{(1)}), \quad i^* = \text{argmax}_i |R_i^{(1)}|. \tag{7}$$

When the indicator values are positive, it indicates that the probabilities of the first and second layers are added, and when they are negative, the probabilities of the second layer are subtracted from those of the first. Consider a classification task of three classes A, B and C. Assume that, given a single image whose actual class is A, the predicted probabilities of the first layer using this image are 0.4, 0.6 and 0 for the three classes A, B and C, respectively. To add a second layer, the residual error using $\lambda = 0.8$ (Equation 5) is computed as 0.4, -0.6 and 0 for each of the three classes. Hence, the new label to train the second layer for this image is B, as it has the largest magnitude, and the indicator is -1 because the maximum absolute residual error has a negative sign. Assume that the second layer's features are trained using class B and provided perfect prediction with probabilities of 0, 1 and 0 for the three classes. Because the indicator value is negative, we subtract them from the predicted outputs of the first layer, resulting in 0.4, -0.4 and 0 for the three classes. Consequently, we reduce the error, and the predicted class of the second layer is A, which corresponds to the actual class of the image. To implement that and generalise it for the test set, as we do not know the classes, we divide the database into positive and negative samples based on their indicator variable values. The positive samples have positive indicator values, whereas the negative samples have negative indicator values. We then train two LDA classifiers for each layer; one is trained on the negative samples using $\{T_n^{(2)} \subset T^{(2)} : s^{(2)} = -1\}$, and the other is trained with the positive samples and their classes $\{T_p^{(2)} \subset T^{(2)} : s^{(2)} = 1\}$. The negative classifier is trained by assuming that each class in the positive samples is a negative class (its class is zero). Similarly, during the training of the positive classifier, each class in the negative samples is treated as a negative class. Thus, both classifiers have access to all training data. The network's outputs at the second layer ($\tilde{Y}^{(2)}$) can then be expressed as follows:

$$\tilde{Y}^{(2)} = \tilde{Y}^{(1)} + \alpha\left[\frac{n_p}{N}\tilde{Y}_p^{(2)} - \frac{n_n}{N}\tilde{Y}_n^{(2)}\right], \tag{8}$$

where $\tilde{Y}_n^{(2)}$ and $\tilde{Y}_p^{(2)}$ are the N predictions made by the classifiers trained on negative and positive samples, n_n and n_p denote the number of negative and positive examples, respectively, N is the total number of training samples and α is a learning rate. The learning rate, similar to that used in neural networks, is introduced to reduce oscillations and provide faster convergence. However, the learning rate in our network also acts as a weight to integrate the probabilities of multiple layers, similar to weighted sum techniques.

To add more layers, we repeat the steps used to add the second one. Algorithm 1 summarises the procedures of the network's training with L layers, assuming that the second-order features are known. To add layer i, the new labels $T^{(i)}$ and indicator variable ($s^{(i)}$) are computed based on the previous layer's residual error $R^{(i-1)}$, as shown in Equation 10. The network's output at that layer can then be defined using Equation 11. In general, for deeper residual compensation

layers, the new classes ($T^{(L)}$) learnt from the residual errors of the previous layers can be defined as follows:

$$\begin{aligned}
T^{(L)} &= \text{class}(|R^{(L-1)}|) \\
&= \text{class}(|Y - \tilde{Y}^{(L-1)}|) \\
&= \text{class}(|Y - [\tilde{Y}^{L-2} + \frac{\alpha}{N}(n_p^{(L-2)}\tilde{Y}_p^{(L-2)} - n_n^{(L-2)}\tilde{Y}_n^{(L-2)})]|) \\
&= \vdots \\
&= \text{class}(|Y - [\tilde{Y}^{(1)} + \frac{\alpha}{N}\sum_{i=2}^{L-1}(n_p^i\tilde{Y}_p^i - n_n^i\tilde{Y}_n^i)]|),
\end{aligned}$$
(9)

where $\text{class}(x)$ is the name of the class whose value has the largest residual error magnitude, Y is the original classes in one-hot encoding, n_p^i and n_n^i are the number of positive and negative samples in layer i, respectively, and N denotes the total number of samples in the database.

III. EXPERIMENTS

A. Databases

We used four standard benchmarks in our experiments: CIFAR-10 [13], CIFAR-100 [13], MNIST [4] and TinyImageNet [15]. The MNIST database comprises 60,000 training examples and 10,000 test images of size 28×28, drawn from the same distribution, normalised and centred in a fixed-size image. The CIFAR-10 database consists of 10 classes with 50,000 images for training and 10,000 test images. The images of size $32 \times 32 \times 3$ have a low resolution with different poses and angles. CIFAR-100 is similar to CIFAR-10 but with 100 classes. The TinyImageNet database consists of 100,000 training images of size $64 \times 64 \times 3$. The images are divided into 200 categories, with 500 images each. The validation and the test sets contain 10,000 images each, with 50 images per class. The test set is not labelled, and our experiments' performance is reported on the validation set.

B. Image Classification without Data Augmentation

In this section, we evaluate the proposed network on the MNIST, CIFAR-10, CIFAR-100 and TinyImageNet databases (Section III-A) without data augmentation. To determine the network parameters, we examine a variety of configurations, each of which has different parameters, and report the results of the configuration that works the best.

1) Parameter Settings: The optimal settings identified for the MNIST, CIFAR-10, CIFAR-100 and TinyImageNet databases are listed in Table I. All architectures, except the MNIST database, used 3×3-pixel filters created by combining PCA [17] and stacked-LDA filters (Appendix A) at a 50% ratio. The MNIST database used 13×13 PCA filters in its first layer and 3×3 PCA filters in its residual layers. In this section and throughout the rest of this article, we used the same number of filters for all layers and stopped adding layers when the training error rate approached 0%. Therefore, the number of filters per layer is 60 in the MNIST's architecture, 50 in the CIFAR's architectures and 40 in TinyImageNet's architecture. The number of layers beyond which an accuracy

Algorithm 1 Deep Residual Compensation Convolutional Network Training

---

Input: Second-order features: $\{F^{(i)}, \ i = [1, 2, \ldots, L]\}$, L number of layers, C classes $T^{(1)} \in \mathbb{R}^{N \times 1}$, learning rate: α and λ to determine the highest probability a class can reach.

Output: Model's accuracy: *accuracy*

1: Generate $Y \in \mathbb{R}^{N \times C}$, the one-hot encoding of $T^{(1)}$.
2: $i \leftarrow 1$ and $\tilde{Y}^{(0)} \leftarrow 0$.
3: $s^{(i)} = 1^{N \times 1}$ {fill the first layer's indicator variable with 1. }
4: **while** $i < L$ **do**
5:   **if** $i > 1$ **then**
6:     Find the residual errors, new classes and indicator variable, as follows:

$$R^{(i-1)} = \lambda Y - \tilde{Y}^{(i-1)}.$$
$$T_j^{(i)} = \text{class}(|R_{jk}^{(i-1)}|, \forall k),$$
$$s_j^{(i)} = \text{sign}(R_{j^*}^{(i-1)}), \ j^* = \text{argmax}_j |R_j^{(i-1)}|, \quad (10)$$
$$j = [1, 2, \ldots, N].$$

7:   **end if**
8:   Find $F_n^{(i)} \subset F^{(i)}$ and $T_n^{(i)} \subset T^{(i)}$, for which their indicator values are negative.
9:   Find $F_p^{(i)} \subset F^{(i)}$ and $T_p^{(i)} \subset T^{(i)}$, for which their indicator values are positive.
10:   **if** $T_p^{(i)} \neq \emptyset$ **then**
11:     $L_1 = \text{LDA}(F_p^{(i)}, T_p^{(i)})$.
12:     $\tilde{Y}_p^{(i)} = \text{prediction}(L_1, F^{(i)})$.
13:   **else**
14:     $\tilde{Y}_p^{(i)} \leftarrow 0$.
15:   **end if**
16:   **if** $T_n^{(i)} \neq \emptyset$ **then**
17:     $L_2 = \text{LDA}(F_n^{(i)}, T_n^{(i)})$.
18:     $\tilde{Y}_n^{(i)} = \text{prediction}(L_2, F^{(i)})$.
19:   **else**
20:     $\tilde{Y}_n^{(i)} \leftarrow 0$.
21:   **end if**
22:   Compute the current's layer output $\tilde{Y}^{(i)}$, as

$$\tilde{Y}^{(i)} = \tilde{Y}^{(i-1)} + \alpha\left[\frac{n_p}{N}\tilde{Y}_p^{(i)} - \frac{n_n}{N}\tilde{Y}_n^{(i)}\right], \quad (11)$$

  where n_p and n_n represent the number of positive and negative samples, respectively.
23:   $i \leftarrow i + 1$
24: **end while**
25: Compute the model's accuracy at layer L using $\tilde{Y}^{(L)}$.

---

gain was no longer observed was 937 for the CIFAR-10 database, 436 for the CIFAR-100 database, 231 for the MNIST database and 512 for the TinyImageNet database. We utilised 7×7-block second-order pooling for MNIST with a stride of four pixels, 16×16 for CIFAR-100 with a four-pixel stride, 16×16 for CIFAR-10 with a one-pixel stride, and 32×32 for TinyImageNet with an eight-pixel stride. In all architectures, we pooled the second-order features using three-level spatial pyramid pooling of 4×4, 2×2 and 1×1 subregions. The only data preprocessing was min-max normalisation applied to the input of each convolutional layer, and probabilities were retrieved from all datasets except MNIST using the sigmoid function with a scale value of 16 (Equation 4). In the MNIST database, we used the following softmax function to generate the probabilities in each layer:

$$\text{softmax}(y_i) = \frac{\exp(\beta y_i)}{\sum_{j=1}^{C} \exp(\beta y_j)}, \quad (12)$$

where β is assigned to 0.001, C denotes the number of classes, and y represents the outputs of the LDA classifier. During the training phase, λ (Equation 5) was set to 0.8, and the learning rate was fixed at $\alpha = 1$ for the MNIST database and $\alpha = 0.4$ for the CIFAR-10 database. The remaining databases used an initial learning rate of 1, which was dropped by 10% every 10 layers as follows:

$$\alpha = \alpha - \frac{10}{100}\alpha. \quad (13)$$

We stopped reducing the learning rate when it reached 0.387 and 0.478 for CIFAR-100 and TinyImageNet databases, respectively.

TABLE I
NETWORK ARCHITECTURES USING THE MNIST, CIFAR-10, CIFAR-100 AND TINYIMAGENET DATABASES

Filter size	SOP	Output size
The MNIST database: $28 \times 28 \times 1$		
$13 \times 13 \times 1 \times 60$	7×7, Stride $= 4$	$28 \times 28 \times 60$
$[3 \times 3 \times 60 \times 60] \times 230$	7×7, Stride $= 4$	$28 \times 28 \times 60$
The CIFAR-10 database: $32 \times 32 \times 3$		
$3 \times 3 \times 3 \times 50$	16×16, Stride $= 1$	$32 \times 32 \times 50$
$[3 \times 3 \times 50 \times 50] \times 936$	16×16, Stride $= 1$	$32 \times 32 \times 50$
The CIFAR-100 database: $32 \times 32 \times 3$		
$3 \times 3 \times 3 \times 50$	16×16, Stride $= 4$	$32 \times 32 \times 50$
$[3 \times 3 \times 50 \times 50] \times 435$	16×16, Stride $= 4$	$32 \times 32 \times 50$
The TinyImageNet database: $64 \times 64 \times 3$		
$3 \times 3 \times 3 \times 40$	32×32, Stride $= 8$	$64 \times 64 \times 40$
$[3 \times 3 \times 40 \times 40] \times 511$	32×32, Stride $= 8$	$64 \times 64 \times 40$

2) Performance Analysis: Table II reports the accuracy of ResCNet compared with some gradient-based and non-gradient-based networks on the MNIST, CIFAR-10, CIFAR-100 and TinyImageNet databases without data augmentation. The non-gradient-based models reported in Table II are the best-performing PCANet-like models from the perspective of the datasets we considered. Such networks include PCANet

[3], LDANet [3], DFSNet [7] and Multi-layer PCANet [2]. To show where our network fits in the literature of commonly used networks, we compared the performance achieved by our network to those obtained by standard gradient-based convolutional networks. Maxout [9], Network in network [16], stochastic pooling [21] and ResNet [10] are the gradient-based networks listed in the table. The accuracy of 164-ResNet with pre-activation was reported by Huang et al. [11], while the results of ResNet-18 and ResNet-34 were reported by Jeevan [12].

According to Table II, for the MNIST database, our model with 231 layers achieved an accuracy of 99.52%, making it superior to all non-gradient-based models, such as PCANet, Multi-Layer PCANet and LDANet, in terms of accuracy. It improved on the best results of the non-gradient-based networks by roughly 0.12%. In addition, the findings presented in the table demonstrated that our network produced results comparable with standard gradient-based networks, such as the Maxout network, Network in network and stochastic pooling.

As shown in Table II, for the CIFAR-100 database, the accuracy attained by our proposed network was the highest among all the networks. The accuracy of 64.91% was around 8% higher than that of Multi-Layer PCANet and stochastic pooling, more than 9% higher than that of ResNet-110 and ResNet-32, 14% higher than the original PCANet and more than 2% higher than that of Maxout and ResNet with stochastic depth. Moreover, the results achieved by our network were roughly equivalent to those obtained by ResNet with 164 layers and Network in network using the dropout technique.

For TinyImageNet, as shown in Table II, our network achieved the highest accuracy among all the networks without any data augmentation. This performance was around 2% higher than that of the ResNet-34 model and 1% better than that of the ResNet-18 model.

According to the results shown in Table II for the CIFAR-10 database, our model outperformed PCANet and all of its variants, in terms of accuracy. The accuracy of the proposed network was around 10% better than the performance achieved by the original PCANet and 6% higher than that of Multi-Layer PCANet. Although ResCNet's accuracy was around 1% lower than that of the Maxout network and 2% worse than that of Network in network, our model achieved accuracy comparable with that of ResNet-32, 1% higher than ResNet-18 and ResNet-110, and 3% greater than stochastic pooling. In general, ResCNet with more than 900 layers showed an excellent performance of 87.54% on the CIFAR-10 database with no data augmentation, making it the first PCANet-like network to reach such a number of layers and such a performance.

In general, Table II demonstrates that ResCNet outperformed all PCANet-like networks in terms of accuracy and the number of layers required for training. It also shows that our model, which is trained without complicated non-linear functions or regularisation techniques, achieves accuracy similar to that of standard convolutional networks such as Network-in-Network, Maxout, stochastic pooling, and several residual networks. Even though the number of layers in our network is relatively large, these layers are added sequentially, one after the other, without iterations or intensive parameter updates. In addition, we used a small number of filters in each layer for all of our architectures, with no configuration exceeding 60 filters per layer. To the best of our knowledge, ResCNet is the first non-gradient-based propagation-free network to be trained with hundreds of layers.

TABLE II
ACCURACY (%) OF RESCNET COMPARED WITH DIFFERENT METHODS ON THE CIFAR-10 (C10), CIFAR-100 (C100), MNIST AND TINYIMAGENET (T200) DATABASES WITHOUT DATA AUGMENTATION

Non-gradient-based networks				
Method	C10	C100	MNIST	T200
PCANet-2	77.14	51.62	99.34	30
LDANet	78.33	–	99.38	–
DFSNet-3	81.06	–	–	–
Multi-Layer PCANet	81.72	57.86	99.40	40.87
ResCNet (ours)	87.54	**64.9**	99.52	**44.37**
Gradient-based networks				
Method	C10	C100	MNIST	T200
Stochastic pooling	84.87	57.49	99.53	–
Maxout network, with dropout	88.32	61.43	**99.55**	–
Network in network, with dropout	**89.59**	64.32	99.53	–
Network in network, without dropout	85.49	–	–	–
110 ResNet	86.82	55.26	–	–
ResNet stochastic depth	-	62.20	–	–
164-ResNet (pre-activation)	-	64.42	–	–
ResNet-18	86.29	59.15	–	43.02
ResNet-32	87.97	56.05	–	42.65

C. Network Parameters

This section explains how to configure ResCNet's parameters by analysing their impact on the model's accuracy using the CIFAR-10 database. The experiments in this section are divided into two subsections, as follows:

- **Experiment 1:** Testing the effect of the number of filters on the accuracy of ResCNet
- **Experiment 2:** Studying the impact of learning rate on the model's performance

1) Number of Filters: This experiment aimed to determine how changing the number of filters affects the accuracy of ResCNet. For this, we designed two networks with the same settings but different numbers of filters. The first, referred to as ResCNet-30, used 30 filters in each of its layers, while the second (ResCNet-50) used 50 filters in all of its layers. The two networks followed the same parameters settings as those described in Section III-B for the CIFAR-10 database. However, we used an 8x8 second-order pooling block size.

Figure 2 shows the accuracy of training and testing ResCNet-30 and ResCNet-50 on the CIFAR-10 database without data augmentation. According to the figure, both networks achieved the same level of accuracy, although ResCNet-30 required more layers. ResCNet-50 obtained a training accuracy of 100% at layer 127, while ResCNet-30 achieved 100% at layer 781. The testing accuracy for both 127- and 781-layer networks was around 86.28%. The best testing accuracy of 86.33% was obtained with 100-layer ResCNet-50 and 500-layer ResCNet-30. The findings in this section suggested that,

unlike previous PCANet-like networks in which the number of filters in each layer should be determined in advance, we could obtain the same performance by employing any number of filters based on the available resources. However, we needed more filters in each layer to achieve the desired performance faster. The results in this section also showed that the second-order pooling block size affected the accuracy of ResCNet. For instance, the highest accuracy reached for the CIFAR-10 database (87.54% in Section III-B) was around 1% better than that achieved in this section (86.33%), with the only difference between the models being the modification of the second-order pooling block size.
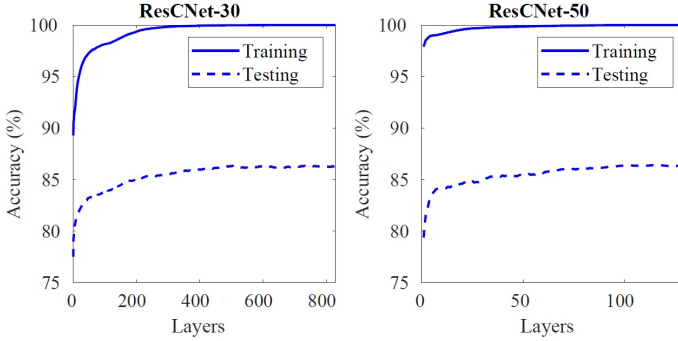

Fig. 2. The accuracy (%) of ResCNet-30 and ResCNet-50 on the training and testing sets of the CIFAR-10 database with no data augmentation.

2) Learning Rate: This experiment aimed to show the impact of the learning rate (α in Equation 9) on ResCNet's accuracy. Similar to neural networks, the learning rate is introduced to prevent oscillations and promote faster convergence. We designed two networks with the same parameters but different learning rates and evaluated them on the CIFAR-10 database. Both networks used 50 filters in all of their layers. The first network (ResCNet-50–1) was trained with a learning rate of $\alpha = 1$, whereas the second (ResCNet-50-0.4) was trained using $\alpha = 0.4$. The other network's parameters were the same as those described in Section III-B for the CIFAR-10 database.

Table III compares the accuracy of ResCNet-50–1 with that of ResCNet-50–0.4 using a different number of layers on the CIFAR-10 database. The training accuracy of ResCNet-50–1 reached 100% at layer 384, with a testing accuracy of 86.91%. On the other hand, ResCNet-50–0.4 obtained a 100% training accuracy at layer 970, with a testing accuracy of 87.41%. The optimal testing accuracy of ResCNet-50–1 was achieved at layer 208, with 87.2% testing accuracy, while ResCNet-50–0.4 obtained its best performance of 87.54% at layer 937. The results achieved in this section demonstrated that the learning rate $\alpha = 0.4$ was small, as we needed more than 600 layers for the network to reach its convergence. If the training error rate oscillates, which is not the case in CIFAR-10, the learning rate must be decreased to avoid oscillations.

TABLE III
ACCURACY (%) ON THE CIFAR-10 DATABASE TEST SET USING
RESCNET-50-1 AND RESCNET-50-0.4

Network	# Layers	Accuracy (%)
ResCNet-50–1	348	86.91
ResCNet-50–1	208	87.02
ResCNet-50–0.4	970	87.41
ResCNet-50–0.4	937	**87.54**
ResCNet-50–0.4	600	87.09

D. Image Classification with Data Augmentation

This section aims to enhance the classification accuracy of ResCNet using data augmentation on three databases, namely, CIFAR-10, CIFAR-100 and TinyImageNet. The ResCNet architectures in this section shared the same design and parameters as those in Section III-B but had 490, 507 and 480 layers for the CIFAR-10, CIFAR-100 and TinyImageNet databases, respectively. For CIFAR-10, we modified the pooling stride to four pixels and re-implemented a 193-layer ResCNet without data augmentation for adequate comparison.

Table IV compares the accuracy of ResCNet with and without data augmentation on the CIFAR-10, CIFAR-100 and TinyImageNet databases. With horizontal flipping being the only data augmentation used, the accuracy of ResCNet was enhanced across all three databases. This improvement was around 1% for the TinyImageNet database, 2% for the CIFAR-10 database and 3% for the CIFAR-100 database. For the CIFAR-10 database, the accuracy achieved with data augmentation was also 1% higher than the best result (87.54%) reported in Section III-B.

This section's results showed the importance of data augmentation for improving the model's generalisation and accuracy. Incorporating other data augmentation types are needed to increase the model's accuracy further. Since ResCNet is not currently being trained in a batch-based manner, adding more data augmentation types is challenging as they need to be computed in advance. In future works, we will investigate the possibility of converting the current work into a batch-based system as in gradient-based models.

TABLE IV
ACCURACY (%) OF RESCNET ON THE CIFAR-10, CIFAR-100 AND
TINYIMAGENET DATABASES

Data augmentation	CIFAR-10	CIFAR-100	TinyImageNet
✗	86.82	64.9	44.37
✓	**88.35**	**67.8**	**45.91**

IV. CONCLUSIONS

In this article, we proposed ResCNet, a PCANet-like network that trains each layer with new labels derived from the residual data of all preceding layers. Our proposed network increased the network depth to more than 950 layers, making it the first non-gradient-based propagation-free network to achieve this number. Moreover, ResCNet's performance was

comparable to that of standard gradient-based models and superior to PCANet and all of its variants. Increasing the size of the databases by one type of data augmentation resulted in a considerable improvement in accuracy, particularly in CIFAR-100, where it reached 3%. However, increasing the number of samples leads to higher computational costs. To overcome this issue and in future work, we will investigate the possibility of transforming the current network into a batch-based system, similar to the neural network. In addition, the network may be developed further by modifying the filter types or by using other metrics to define the residual errors and, thus, the new classes.

REFERENCES

[1] Md Zahangir Alom et al. "The history began from alexnet: A comprehensive survey on deep learning approaches". In: *arXiv Preprint arXiv:1803.01164* (2018).

[2] Mubarakah Alotaibi and Richard C Wilson. "Multi-layer PCA Network for Image Classification". In: *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer. 2021, pp. 292–301.

[3] Tsung-Han Chan et al. "PCANet: A simple deep learning baseline for image classification?" In: *IEEE Transactions on Image Processing* 24.12 (2015), pp. 5017–5032.

[4] Li Deng. "The mnist database of handwritten digit images for machine learning research [best of the web]". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[5] Chunxiao Fan et al. "PCANet-II: When PCANet Meets the Second Order Pooling". In: *IEICE Transactions on Information and Systems* 101.8 (2018), pp. 2159–2162.

[6] Bernardo B Gatto and Eulanda M dos Santos. "Discriminative canonical correlation analysis network for image classification". In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 4487–4491.

[7] Bernardo B Gatto et al. "A semi-supervised convolutional neural network based on subspace representation for image classification". In: *EURASIP Journal on Image and Video Processing* 2020.1 (2020), pp. 1–21.

[8] Bernardo Bentes Gatto, Eulanda Miranda dos Santos, and Kazuhiro Fukui. "Subspace-based convolutional network for handwritten character recognition". In: *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*. Vol. 1. IEEE. 2017, pp. 1044–1049.

[9] Ian Goodfellow et al. "Maxout networks". In: *International Conference on Machine Learning*. PMLR. 2013, pp. 1319–1327.

[10] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[11] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4700–4708.

[12] Pranav Jeevan. "Convolutional Xformers for Vision". In: *arXiv Preprint arXiv:2201.10271* (2022).

[13] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: *2009* (2009).

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[15] Ya Le and Xuan Yang. "Tiny imagenet visual recognition challenge". In: *CS 231N* 7.7 (2015), p. 3.

[16] Min Lin, Qiang Chen, and Shuicheng Yan. "Network in network". In: *arXiv preprint arXiv:1312.4400* (2013).

[17] Cheng-Yaw Low, Andrew Beng-Jin Teoh, and Kar-Ann Toh. "Stacking PCANet+: An overly simplified convnets baseline for face recognition". In: *IEEE Signal Processing Letters* 24.11 (2017), pp. 1581–1585.

[18] Cong Jie Ng and Andrew Beng Jin Teoh. "DCTNet: A simple learning-free approach for face recognition". In: *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. IEEE. 2015, pp. 761–768.

[19] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv Preprint arXiv:1409.1556* (2014).

[20] Xinghao Yang et al. "Canonical correlation analysis networks for two-view image recognition". In: *Information Sciences* 385 (2017), pp. 338–352.

[21] Matthew D Zeiler and Rob Fergus. "Stochastic pooling for regularization of deep convolutional neural networks". In: *arXiv preprint arXiv:1301.3557* (2013).

[22] Yongqing Zhang et al. "ICANet: A simple cascade linear convolution network for face recognition". In: *EURASIP Journal on Image and Video Processing* 2018.1 (2018), pp. 1–7.

In this appendix, we introduce the stacked-LDA method, a model that stacks two Linear discriminant analysis layers. The first layer of the model is trained using new labels that are produced by clustering similar instances of a certain class. The second layer, on the other hand, is trained using the original classes. This general concept of the stacked-LDA model is described in Section A-A. We propose an iterative method to create clusters of similar groups. The description of our method is explained in Section A-B. In addition, Section A-C discusses the procedures for using the stacked-LDA algorithm as convolution filters. Finally, the parameter settings we used in our experiments in the main paper were described in Section A-D.

A. General Concept

The stacked-LDA algorithm relies on stacking two linear discriminant analysis (LDA) layers. The first LDA layer is trained using labels created from the original classes, while the second LDA layer is trained using the actual labels. To generate the classes of the first layer, similar samples from a given class are grouped to create a new class. For example, a class A with s instances can be subdivided into c new classes, each with a different number of instances. The minimum number of samples needed to represent a class is one. Suppose the first LDA can differentiate between the new classes ideally. In that case, the second LDA will receive the posteriors of the first LDA and be able to differentiate between the actual classes. For instance, if class A is subdivided into A_1 and A_2, the subsequent LDA will identify that both A_1 and A_2 are members of class A.

B. Stacked-LDA Algorithm

Let $X \in \mathbb{R}^{N \times M}$ represent N training samples, each with M dimensions, and $Target \in \mathbb{R}^{N \times 1}$ represent their original classes. Algorithm 2 describes the procedures for applying the stacked-LDA to the training set. The algorithm starts by picking a random class c from the actual classes. We then choose $N_{positive}$ random instances that belong to class c and $N_{negative}$ random examples that are not in class c, where $N_{positive}$ and $N_{negative}$ are the number of positive and negative samples and are user-predefined parameters. Next, an LDA classifier discriminates between the positive and the negative samples. After that, we check if our chosen random samples are linearly separable, which can be done by comparing the error rate of the LDA classifier with a small value of nearly zero called tolerance (tol) and is chosen by the user. If the LDA's error rate is lower than the tolerance, we consider the positive class to be a new class and collect the weights of the LDA. On the other hand, if the LDA's error rate is greater than the tolerance, the chosen samples are not similar and cannot be grouped. Therefore, the algorithm proceeds to find other classes that separate the data accurately in the same way until reaching the required number of classes ($N\_classes$). When algorithm 2 terminates, we can use the generated LDA's weights to find the output of the first LDA. Another LDA can then be applied to the output of the first LDA to classify them back using the original classes.

Algorithm 2 Stacked-LDA Algorithm

Input: Training set: $X \in \mathbb{R}^{N \times M}$, where $\{x_i^N \ , \ x_i \in \mathbb{R}^M\}$, original classes: $Target \in \mathbb{R}^{N \times 1}$, number of classes user wants to generate: $N\_classes$, number of positive samples: $N_{positives}$, number of negative samples: $N_{negatives}$ and tolerance of performance user can afford: tol

Output: LDA's weights: $weights \in \mathbb{R}^{N \times N\_classes}$ and LDA's bias or constant: $bias \in \mathbb{R}^{N\_classes}$

1: $weights \leftarrow [\,]$.
2: $bias \leftarrow [\,]$.
3: $i \leftarrow 1$.
4: **while** $i < N\_classes$ **do**
5:     Pick a random class c from the Target.
6:     Pick random $N_{positives}$ samples from class c ($S_{positives}$).
7:     Choose $N_{negatives}$ samples that are not in class c ($S_{negatives}$) randomly.
8:     Combine the negative and positive samples: $S \leftarrow [S_{positives}, S_{negatives}]$.
9:     $T \leftarrow [ones(N_{positives}), zeros(N_{negatives})]$.
10:     Find the linear discriminant analysis (LDA) between S and T: $L = \text{LDA}(S, T)$.
11:     Find the perfromance ($ErrorRate$) of S using L.
12:     **if** $ErrorRate < tol$ **then**
13:         $weights \leftarrow [weights, LDA's weights]$.
14:         $bias \leftarrow [bias, LDA's bias]$.
15:         $i \leftarrow i + 1$.
16:     **end if**
17: **end while**

C. Stacked-LDA Filters

For N training samples X: $\{X_i \in \mathbb{R}^{m \times n \times c}\}$ with actual classes $Target \in \mathbb{R}^{N \times 1}$, where m and n are the spatial dimensions of the image and c is the number of channels, we compute the stacked-LDA filters as follows:

1) Given a single image $X_i \in \mathbb{R}^{m \times n \times c}$ and a filter size $k_L \times k_L$, we extract and vectorise all overlapping patches of size $k_L \times k_L \times c$ each. The resulting matrix is $P_i \in \mathbb{R}^{(k_L^2 \times c) \times \tilde{m}\tilde{n}}$, where $\tilde{m} = (m - k_L) + 1$, $\tilde{n} = (n - k_L) + 1$ and m and n are the spatial dimensions of the image, respectively;

2) We repeat the previous step for all images in the dataset to obtain $P \in \mathbb{R}^{(k_L^2 \times c) \times \tilde{m}\tilde{n}N}$;

3) We create a vector $T \in \mathbb{R}^{1 \times \tilde{m}\tilde{n}N}$ that contains the class labels of the patches. A single patch is assigned a label equivalent to the class of its full image;

4) Using random samples and specific tolerance, we apply Algorithm 2 on P and T to obtain the stacked-LDA filters' weights W_s^L and bias B_s^L;

5) We can express the Stacked-LDA filters as follows:

$$
\begin{aligned}
W_s^L &= \operatorname*{mat}_{k_L \times k_L \times c} q_s, \ s = 1, 2, \ldots, d_L, \\
B_s^L &= \operatorname*{mat}_{1 \times 1 \times c} q_s, \ s = 1, 2, \ldots, d_L,
\end{aligned}
\tag{14}
$$

where d_L is the number of filters chosen by the user, which is equivalent to the number of classes in Algorithm 2;

6) We convolve the original images with the filters as follows:

$$
X_i^L = X_i^{L-1} * W_s^L + B_s^L \in \mathbb{R}^{m \times n \times d_L},
\tag{15}
$$

where $s = 1, 2, \ldots, d_L$ and X_i^{L-1} is zero-padded to obtain the same image size;

D. Parameter Settings

To compute the stacked-LDA filters in our experiments in the main paper, we set the number of positive samples ($N_{positives}$) to 2, while the number of negative samples ($N_{negatives}$) was 32. The tolerance (tol) in Algorithm 2 was set to zero. We used the LDA classifier with the one-versus-all decomposition method.