Article

Depth-Adaptive Deep Neural Network Based on Learning Layer Relevance Weights

Arwa Alturki *, Ouiem Bchir and Mohamed Maher Ben Ismail

Department of Computer Science, College of Computer and Information Sciences, King Saud University, Riyadh 11362, Saudi Arabia
* Correspondence: aralturki@ksu.edu.sa

Abstract: In this paper, we propose two novel Adaptive Neural Network Approaches (ANNAs), which are intended to automatically learn the optimal network depth. In particular, the proposed class-independent and class-dependent ANNAas address two main challenges faced by typical deep learning paradigms. Namely, they overcome the problems of setting the optimal network depth and improving the model interpretability. Specifically, ANNA approaches simultaneously train the network model, learn the network depth in an unsupervised manner, and assign fuzzy relevance weights to each network layer to better decipher the model behavior. In addition, two novel cost functions were designed in order to optimize the layer fuzzy relevance weights along with the model hyper-parameters. The proposed ANNA approaches were assessed using standard benchmarking datasets and performance measures. The experiments proved their effectiveness compared to typical deep learning approaches, which rely on empirical tuning and scaling of the network depth. Moreover, the experimental findings demonstrated the ability of the proposed class-independent and class-dependent ANNA to decrease the network complexity and build lightweight models for less overfitting risk and better generalization.

Keywords: deep neural network; dynamic neural network; adaptive neural network

1. Introduction

Machine learning techniques have been deployed in a variety of areas, such as computer vision [1,2], pattern recognition [3], natural language processing [4], and speech recognition [5–7]. These techniques have granted computers the ability to learn automatically predictive and descriptive models by exploiting the available data, with no need for explicit programming. Recently, deep learning-based solutions [4–8] have emerged as promising alternatives and demonstrated their effectiveness in several fields [9,10]. In particular, Convolutional Neural Networks (CNNs) [9] have been extensively applied to address image classification [10], pattern recognition [11], image segmentation [12], and image captioning [13] problems. Similarly, Recurrent Neural networks [9] have been exploited to tackle video classification [14], human activity recognition [15], speech enhancement [16], and audio tagging [17] challenges. Moreover, in the context of Hyper-Spectral Image (HSI) analysis, CNNs are adopted for graph-structured data [18,19]. The latter employ semi-supervised information to guide the learning process. Furthermore, autoencoders are exploited for HSI clustering techniques, where special graph convolutional embedding is incorporated to preserve the adaptive layer-wise feature locality [20,21].

The effectiveness of deep learning paradigms has been attributed to their learning ability, achieved through non-linear processing of massive input data and multiple network hidden layers in order to extract relevant features and learn their appropriate representations.

The design of a Deep Neural Network (DNN) with an acceptable level of complexity remains challenging [22] and requires intensive experimentation and domain knowledge [2,5]. Particularly, two key hyper-parameters control the architecture of DNNs
and affect their performance [23]: the depth and the width of the network. Specifically, the network depth represents the number of layers in the network. On the other hand, the network width corresponds to the number of processing units (nodes) in a given layer. Since these two parameters can span over a large range of values, the scaling of DNN models becomes arduous and even more acute.

1.1. Motivation

Setting the depth hyper-parameter is crucial for the effectiveness of the model because each layer aggregates and recombines the previous layer outputs to learn new features sets. Thus, the deeper the neural network is, the more complex the features are [9]. This yields a hierarchy of increasing abstraction and complexity; known as feature hierarchy [24]. In other words, setting the depth endows the deep learning paradigm the capability to determine the complexity of the learned features, and therefore impacts its overall effectiveness [24].

1.2. Problem Statement

Researchers have tackled the problem of setting the depth parameter [9,23,25] through empirical tuning of the network topology based on the validation of the learned models [9]. However, the search space relevant to such a manual procedure is exorbitantly large due to the number of candidate combinations of hyper-parameter values. This computationally expensive solution is impractical for real-world applications. In order to narrow the search space, other approaches [9] initialized the network hyper-parameters by exploiting the hyper-parameter value set for other existing DNN models built for similar machine learning tasks. Lately, automated search strategies, such as search schemes and heuristic searches, have been also used as promising alternatives to set DNN hyper-parameters [9,26]. These schemes include random searches [26] and grid searches [26], while heuristic searches include genetic algorithms [27] and Bayesian optimization [28]. Nevertheless, such automated searches exhibit scalability issues when dealing with large neural networks involving large datasets, which is the case in real-world applications [24].

1.3. Contribution

In this paper, we propose novel Adaptive Neural Network Approaches (ANNAs) that learn the optimal DNN depth and associate fuzzy relevance weights to each DNN layer. Specifically, we propose two algorithms to simultaneously learn (i) the depth parameter and (ii) the weight of each layer, and then train the DNN model. This aims at reducing the number of layers. Moreover, the interpretability of the resulting deep neural network model is enhanced through the learned weights assigned to each layer. The learning of the network depth as well as the layer weights is formulated as an optimization problem through the design of novel cost functions in addition to the loss function used to train the DNN model.

The rest of this paper is organized as follows: Existing dynamic deep learning approaches that automatically learn the depth and width of neural networks are surveyed in Section 2. The proposed ANNA approaches are presented in Section 3. The experiments conducted to validate and assess the proposed work are depicted in Section 4. Finally, Section 5 concludes the article and outlines future works.

2. Literature Review

Over the last decade, researchers have tackled the automatic scaling of DNN width through dynamic expansion and/or pruning of the network neurons [29–34]. In the literature, such approaches are referred to as dynamic deep learning or adaptive deep learning. To the best of our knowledge, no existing works have addressed the automatic learning of DNN depth, except the research in [23], where an Adaptive Structural Learning of Artificial Neural Networks (AdaNet) was introduced. It aims at learning dynamically the architecture of the network by expanding both its width and depth. Researchers
have dealt with the problems resulting from very deep networks by recalling the input instances \([27,35,36]\), skipping some layers \([37]\), or combining them \([38]\), without actually optimizing the network depth.

In the following, we outline existing dynamic deep learning approaches relevant to this research. The deep learning approaches that iteratively expand and/or prune neurons to learn the appropriate network width and depth are reviewed in the first subsection. The second part of this review is dedicated to the alternative deep learning approaches that rely on bypassing or combining network layers.

### 2.1. Dynamic Architectures

Deep learning approaches that expand \([31,32]\) and/or prune \([33,34]\) the network neurons iteratively to determine the appropriate network architecture have been presented as an emerging alternative to conventional empirical network scaling. In \([29]\), the authors addressed the problem of determining the optimal number of neurons for a denoising auto-encoder \([39]\). Their incremental learning algorithm consists of two procedures: (i) adding neurons and (ii) merging neurons. The approach is based on a cost function that is intended to determine the outliers. It combines the generative loss \((L_{\text{gen}})\) and the discriminative loss \((L_{\text{disc}})\) terms as follows:

\[
L(x, y) = L_{\text{disc}}(x, y) + \lambda L_{\text{gen}}(x)
\]

where \(x\) is the input example, \(y\) is the groundtruth label, and \(L_{\text{gen}}\) is the cross-entropy loss between \(x\) and its reconstructed image \(\hat{x}\). The cross-entropy loss between \(y\) and the predicted label \(\hat{y}\) is represented using \(L_{\text{disc}}\). When the set of outliers reaches a predefined critical size, the two following processes are iteratively deployed:

1. Find the most similar pair of neurons whose cosine distance is minimal and merge them.
2. Add a new neuron that minimizes the following cost function:

\[
\arg \min_{\theta_N} \frac{1}{|B|} \sum_{i \in B} L\left(x^i, y^i\right)
\]

where \(\theta_N\) is the new neuron parameter and \(B\) is a training batch. This approach is effective for applications with large data, where the distribution changes over time. However, it exhibits slow learning speed due to the large parameter space and the iterative training process, which prevents parallel executions.

The researchers in \([30]\) outlined an approach able to determine the optimal number of neurons per layer during the training phase of the deep network. The network is trained using an Adaptive Radial-angular gradient descent (AdaRad) \([30]\) by adding and eliminating redundant neurons. The redundant neurons, referred to as zero units, which have zero incoming weights (fan-in regularizer) are removed during the training phase. After removing the zero units from certain layers, the algorithm adds a random unit to that layer to increase its width and yield new features. However, eliminating a neuron with zero incoming weights is not effective since it is already neglected due to its zero input.

A Neurogenesis Deep Learning (NDL) algorithm that adds new neurons dynamically to the network layers while training a stacked auto-encoder was introduced in \([31]\). Specifically, the authors proposed to compute the reconstruction error at each auto-encoder level (a pair of encoder and decoder layers). In the case that the reconstruction error of a given level is higher than a pre-defined threshold, then the considered neurons cannot represent that input set. Therefore, NDL incrementally adds a new set of neurons when the model fails to effectively represent a number of samples, called outliers. The last added neurons are initialized through the training of their corresponding level/layer using outlier samples. Then, the updated level is trained with unseen input data and samples from seen data are replayed in order to stabilize the previous auto-encoder representations. This expanding process is repeated as long as the number of outliers of a given level is larger than a predefined threshold. One should note that NDL is memory expensive because
it requires architectural expansion and storage of old samples for the retraining process. Moreover, it requires additional training time due to the repeated forwarded passes.

Another Dynamically Expanded Network (DEN) algorithm that dynamically adds neurons to the convolutional neural network is presented in [32]. Specifically, it prevents semantic drift by duplicating the neurons that caused it. In fact, semantic drift occurs when the learned model fits the new input instances and gradually “forgets” the trained model using the earlier input instances. DEN measures the amount of semantic drift for each neuron as the distance between the incoming weights of two consecutive input instances. If the distance is larger than a given threshold, it reflects the occurrence of semantic drift. To overcome this problem, DEN duplicates the neuron responsible for the semantic drift. Then, it re-conducts the model training. Although DEN dynamically expands the neurons in order to adapt the architecture to the input instances, it requires additional hyper-parameters for regularization and thresholding that need to be carefully tuned. Moreover, due to the sensitivity of the model’s parameters, this approach faces a scalability problem when the network size increases. The researchers in [33] introduced an approach to automatically set the width of each layer for convolutional neural networks while training the model. First, their algorithm trains an initial network, then it removes some neurons during the fine-tuning phase. This is achieved using a group sparsity regularizer [33], which is integrated as penalty for the network parameters. In [33], the parameters of a single neuron define a group, and the corresponding group sparsity regularizer is obtained using the following equation:

$$r(\Theta) = \sum_{l=1}^{L} \lambda_l \sqrt{P_l} \sum_{n=1}^{N_l} \|\theta_n^l\|_2$$

where $\Theta$ is the network parameter, $L$ represents the network depth, and $\lambda_l$ stands for the weight of the penalty with respect to layer $l$. On the other hand, $P_l$ represents the vector of all of the neuron’s parameters in layer $l$, and $N_l$ is the number of neurons in layer $l$. Finally, $\theta_n^l$ represents the parameters of neuron $n$.

In order to prevent removing many neurons in the first few layers, and to retain enough information for the remaining ones, a small weight is assigned to the first few layers and a larger weight for the others. A neuron is removed when its parameters are nulled by the end of the training process. Although the experimental results showed that the proposed approach can reduce the number of neurons by up to 80%, this algorithm trains a new model from scratch for each new input instance, which yields high time complexity. Another pruning strategy to drop neurons from a pre-trained CNN model is depicted in [34]. It interleaves greedy search for the neurons to be pruned through backpropagation fine-tuning. More specifically, the search for the neurons to be pruned is formulated as an optimization problem based on the Taylor expansion approach [34], which aims at finding a pruned model equivalent to the original complete one. Therefore, the algorithm in [34] minimizes the difference between the accuracies of both models, the original and the pruned models. This results in pruning neurons, where the parameters have almost a flat gradient of the log-likelihood cost function with respect to the extracted features. As it can be noticed, this algorithm is extremely time consuming, since it repeatedly trains the network until convergence, measures the performance of the model, solves the optimization problem using the Taylor expansion approximation, conducts fine tuning, and prunes neurons. In addition, the number of pruned neurons is small and yields minimal change on the original architecture.

An Adaptive Supervised Learning Algorithm (AdaNet) in which the network architecture is learned while training the learned model is outlined in [23]. AdaNet starts with a plain linear model and iteratively adds extra layers and units. In particular, at each iteration, two candidate subnetworks are generated randomly: one with the same depth as the current network and another one layer deeper. Then, AdaNet augments the current neural network with the subnetwork that has the lower sum of surrogates of the empirical error and a norm-1 based regularization term. It is important to note that the neurons of
previously learned subnetworks can serve as input to the newly learned subnetwork. Thus, the newly added deeper subnetwork can exploit previously learned embedding.

2.2. Layer Bypassing and Combination

Rather than learning the optimal network depth, some related works over-specify this hyper-parameter and monitor the deep network performance by recalling the input instances [27,35,36], skipping some layers [37], or combining them [38]. In [35], the researchers introduced a Highway Feedforward Network (HFN) inspired by the typical Long Short-Term Memory (LSTM) recurrent networks [9]. HFN relies on two non-linear transformations, namely the transform and the carry gates. In particular, when the transform gate is activated, the corresponding layer performs a non-linear transformation. On the other hand, if the carry gate is activated, then the corresponding layer, referred to as the highway layer, is shunted. This process is achieved through the objective function below:

\[
y = f(x, W) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T))
\]

where \( f \) is the feedforward network transformation, \( W \) represents the network weights, \( T(\cdot) \) is the transform gate, and \( 1 - T(\cdot) \) represents the carry gate. \( T(\cdot) \) is defined as:

\[
T(x) = \sigma(W_T x + b_T)
\]

where \( \sigma \) is the rectified linear activation function [9], \( W_T \) represents the weights of the transform gate, \( x \) represents the input instances, and \( y \) is the predicted output. Initializing the transform bias \( b_T \) to a negative value will make the network initially biased towards carry behavior. In [35], the carry gate is responsible for pruning the layer and thus reduces the depth for a certain input instance. Therefore, certain layers are kept or discarded with respect to each input instance. This makes the network model prone to overfitting. Moreover, the adopted gates require extra parameters and more computational complexity. Another Deep Residual Learning for Image Recognition (ResNet) is proposed in [36]. It tackles the problems resulting from very deep networks by connecting the input layers to the second transformation output and sums the output in the ReLU function. These short connections are perceived as identity mapping of the input layers. This allows recalling the input instances and prevents the vanishing problem. However, the summation of the identity mapping function and block output in ResNet [36] may impede the information flow in the network [25]. Attention-based adaptive spectral–spatial kernel ResNet (A2S2K-ResNet) for hyperspectral image (HSI) classification is introduced in [40]. Specifically, the algorithm proposes an attention-based adaptive spectral–spatial kernel module to learn 3D convolution kernels for HSI classification. Then, the joint spectral–spatial features are extracted from a crafted spectral–spatial ResNet.

To overcome the ResNet [36] limitation reported above, the authors in [25] introduced DenseNet. It is a variation of ResNet architecture with a different connectivity pattern. Specifically, in order to recall the previous layers, DenseNet uses direct connections from any layer to all subsequent layers. As a result, each layer receives a concatenation of the preceding layers. Between two dense blocks (stacked layers), there is a transition layer that reduces the number of layers. The reduction is conducted according to a predefined compression factor \( \alpha \). If a dense block contains \( m \) stacked layers, the transition layer reduces them to \( \alpha m \), where \( 0 \leq \alpha \leq 1 \). In order to alleviate the vanishing gradient problem and significantly overcome the information loss issue in very deep networks, a stochastic depth algorithm that reduces the network depth during the training is outlined in [37]. More specifically, for each mini-batch, the stochastic depth algorithm randomly selects a block of stacked layers and bypasses them by replacing their transformation functions with the identity function. Due to the hierarchical structure of the network, the blocks are selected according to linear decaying survival probability [37], which ensures preserving the earlier layers since they are used by the later layers.
A deep fusion scheme that aggregates several base networks is outlined in [38]. Each one of these base networks is divided into $B$ blocks, where each block applies a sequence of non-linear transformations or identity functions. The algorithm combines the $\ell^{th}$ blocks of all networks by summing them and then conveying them as input to the remaining blocks. Base networks are typically shallow networks, where each block consists of one convolutional layer. It is important to note that this deep fusion process does not introduce additional parameters or computational complexity.

3. Depth-Adaptive Deep Neural Network

We propose to automatically learn the optimal DNN depth along with fuzzy relevance weights for each DNN layer. Particularly, we introduce a technique to simultaneously learn the network depth and train the deep neural network model. A novel cost function that optimizes the relevance weight of each layer along with the model hyperparameter is designed to formulate the learning procedure. In other words, we propose to prune the non-relevant layers and accordingly update the deep neural network architecture. As a result, two Adaptive Neural Network Adaptive approaches (ANNAs) are proposed to learn the relevance of each layer, which alleviates both the depth setting problem and the interpretability problem. Specifically, the first ANNA approach is class independent, and the learned layer fuzzy relevance weights do not depend on the pre-defined classes. On the other hand, the second ANNA approach learns layer fuzzy relevance weights that depend on the classes set for the classification problem. One should note that the proposed approaches are generic and applicable to any supervised deep neural network, regardless of its activation function and/or regularization technique.

In particular, as can be seen in Figure 1, the network has $\mathcal{L}$ layers, where the $\ell^{th}$ layer deploys a non-linear transformation $\mathcal{F}$ on the input $x_j$ to produce the output $y_j$ using the following:

$$y_j = \mathcal{F}(x_j; W^\mathcal{F})$$

where $\{x_j\}_{i=1}^N$ is a set of $N$ instances fed as input to the deep learning algorithm, and $W^\mathcal{F}$ represents the set of network parameters (neuron weights). Depending on $\mathcal{F}$, the feedforward network can be a Fully Connected Network (FCN) or a Convolutional Neural Network (CNN).

![Figure 1](image-url)  
Figure 1. The block diagram of the proposed system.

As shown in Figure 1, we introduce a Layer Relevance Weight Learning component. It aims at learning iteratively fuzzy relevance weights for the network layers. For this purpose, we define the distance $d_{jk}^{\ell}$ between the input vector and its mapped vector $x_j^\ell$ at
layer $\ell$. The distance $d_{j,k}^\ell$ represents the degree to which two instances $x_j$ and $x_k$ are related according to their representations at layer $\ell$. This distance is obtained using the Euclidian distance as follows:

$$d_{j,k}^\ell = \|x_j^\ell - x_k^\ell\|_2$$  \hspace{1cm} (7)

We also define the membership coefficient $\gamma_{ij}$ as

$$\gamma_{ij} = \begin{cases} 1, & \text{if } j \in c_i \\ 0, & \text{if } j \not\in c_i \end{cases}$$  \hspace{1cm} (8)

Note that each input instance $x_j$ belongs to a given class $c_i$.

In this paper, we introduce two approaches to learn fuzzy relevance weights with respect to each layer $\ell$. These weights reflect the contribution of each layer in the classification task. In fact, a small fuzzy weight indicates that the layer does not contribute sufficiently and therefore can be discarded. Alternatively, a large fuzzy weight indicates that the corresponding layer contributes heavily in the classification and therefore should be maintained. For this purpose, two approaches are proposed. The first ANNA approach is class independent. In other words, the fuzzy weights are learned globally for all classes, and they do not depend on the pre-defined classes. On the other hand, the second ANNA approach learns fuzzy weights with respect to the problem classes.

3.1. Class-Independent ANNA

In order to learn the layer fuzzy relevance weights, $w_t^\ell$, at iteration $t$, after training each batch, we designed the following objective function:

$$J = \sum_{i=1}^{C} \sum_{u=1}^{M^t} \sum_{j \in B_u} \sum_{k \in B_u} \gamma_{ij} \gamma_{ik} \sum_{\ell=1}^{L_t} d_{j,k}^\ell w_t^\ell$$  \hspace{1cm} (9)

Subject to

$$0 \leq w_t^\ell \leq 1 \text{ and } \sum_{\ell=1}^{L_t} w_t^\ell = 1$$  \hspace{1cm} (10)

where $C$ is the number of classes, $M^t$ is number of the processed batches at $t$, and $B_u$ is the set of instances that belongs to the batch processed at iteration $t$. Similarly, $d_{j,k}^\ell$ represents the distance between $x_j^\ell$ and $x_k^\ell$ as defined in (7), while $w_t^\ell$ is the relevance of layer $\ell$ at iteration $t$. The constant $q$ controls the degree of fuzziness of the learned weights, and $L_t$ represents the number of layers at iteration $t$.

The objective function defined in (9) combines the intra-class distances weighted using a fuzzy relevance weight with respect to each layer $\ell$. Therefore, the minimization of $J$ allows the learning of $w_t^\ell$, the relevance of the representation of layer $\ell$ at iteration $t$. In order to minimize $J$ with respect to $w_t^\ell$, we use the Lagrange multiplier technique [41].

The derivative of (9) with respect to $w_t^\ell$ is

$$\frac{\partial J}{\partial w_t^\ell} = \sum_{i=1}^{C} \sum_{u=1}^{M^t} \sum_{j \in B_u} \sum_{k \in B_u} \gamma_{ij} \gamma_{ik} d_{j,k}^\ell (w_t^\ell)^{q-1} - \lambda$$  \hspace{1cm} (11)

Setting $\frac{\partial J}{\partial w_t^\ell}$ to zero yields

$$w_t^\ell = \frac{1}{H_t^1} \frac{\sum_{i=1}^{C} \sum_{u=1}^{M^t} \sum_{j \in B_u} \sum_{k \in B_u} \gamma_{ij} \gamma_{ik} d_{j,k}^\ell}{\sum_{i=1}^{C} \sum_{u=1}^{M^t} \sum_{j \in B_u} \sum_{k \in B_u} \gamma_{ij} \gamma_{ik}}$$  \hspace{1cm} (12)

where $H_t^1$ is

$$H_t^1 = \sum_{i=1}^{C} \sum_{u=1}^{M^t} \sum_{j \in B_u} \sum_{k \in B_u} \gamma_{ij} \gamma_{ik} d_{j,k}^\ell$$  \hspace{1cm} (13)
One should note that \( H^i_\ell \), in (13) represents the sum of the intra-class distances according to the representation provided by layer \( \ell \). Thus, \( w^i_\ell \) is inversely proportional to the intra-class distances. Therefore, when the intra-class distances are large, the weight \( w^i_\ell \) is small, reflecting a poor representation of layer \( \ell \) at iteration \( t \). On the other hand, when the intra-class distances are small, \( w^i_\ell \) is large, reflecting the relevance of the representation of the layer \( \ell \) at iteration \( t \).

The sum of the intra-class distances, \( H^i_\ell \), can be rewritten in terms of \( H^i_{\ell-1} \) as

\[
H^i_\ell = \sum_{i=1}^{C} \sum_{j \in B^i} \sum_{k \in B^i} \gamma_{ij} \gamma_{ik} d^i_{jk} + H^i_{\ell-1}
\]

(14)

Similarly, \( w^i_\ell \) can be expressed in terms of \( H^i_{\ell-1} \) as follows:

\[
w^i_\ell = \frac{\sum_{i=1}^{C} \sum_{j \in B^i} \sum_{k \in B^i} \gamma_{ij} \gamma_{ik} d^i_{jk} + H^i_{\ell-1}}{\sum_{i=1}^{C} \sum_{j \in B^i} \sum_{k \in B^i} \gamma_{ij} \gamma_{ik} d^i_{jk} + H^i_{\ell-1}}
\]

(15)

This relaxes the need to store all the mapped vectors \( x^j_\ell \) over all batches. In fact, only the last value of \( H^i_{\ell-1} \) with respect to each layer needs to be saved. We should notice here that in addition to reflecting the relevance of each layer, the weights \( w^i_\ell \) allow the learned model to memorize the previous batches’ input instances through their mapping, which is embedded in \( H^i_{\ell-1} \). This addresses the semantic drift issue [9].

### 3.2. Class-Dependent ANNA

The second ANNA approach is similar to the first approach. Nevertheless, it learns layers’ weights, \( w^i_{\ell,t} \), with respect to each class, \( i \). Therefore, the proposed class-dependent objective function is designed as follows:

\[
J = \sum_{i=1}^{C} \sum_{u=1}^{M^u} \sum_{j \in B^u} \sum_{k \in B^u} \gamma_{ij} \gamma_{ik} \sum_{\ell=1}^{\ell^u} d^\ell_{jk} w^i_{\ell,t} - \lambda_i \left( \sum_{\ell=1}^{\ell^u} w^i_{\ell,t} - 1 \right)
\]

(16)

After deriving \( J \) with respect to \( w^i_{\ell,t} \) and proceeding with the same optimization steps introduced in the class independent approach, we obtain the updated equation of the weights \( w^i_{\ell,t} \):

\[
w^i_{\ell,t} = \frac{\sum_{i=1}^{C} \sum_{j \in B^i} \sum_{k \in B^i} \gamma_{ij} \gamma_{ik} d^i_{jk} + H^i_{\ell-1}}{\sum_{i=1}^{C} \sum_{j \in B^i} \sum_{k \in B^i} \gamma_{ij} \gamma_{ik} d^i_{jk} + H^i_{\ell-1}}
\]

(17)

where

\[
H^i_{\ell,t} = \sum_{u=1}^{M^u} \sum_{j \in B^u} \sum_{k \in B^u} \gamma_{ij} \gamma_{ik} d^i_{jk}
\]

(18)

In order to relax the need to store all the mapped vectors \( x^j_\ell \) over all batches, \( H^i_{\ell,t} \) is expressed as

\[
H^i_{\ell,t} = \sum_{j \in B^i} \sum_{k \in B^i} \gamma_{ij} \gamma_{ik} d^i_{jk} + H^i_{\ell-1}
\]

(19)

The layer fuzzy relevance weight is then calculated as the maximum of the class fuzzy relevance weights, as follows:

\[
w^i_\ell = \max_i (w^i_{\ell,t})
\]

(20)

Note that in both two proposed approaches, the fuzzy relevance weights are learned from the data and their values range from 0 to 1. Algorithm 1 depicts the steps of the Layer Fuzzy Relevance Weight Learning adopted by both ANNA approaches.
Algorithm 1: Layer Fuzzy Relevance Weight Learning Algorithm

Input:
\[ \{x^\ell_j\}_{j \in B^\ell} : \text{the mapping of the input instances that belong to the current batch } B^\ell \]
\[ \{\gamma_{ij}\}_{j \in B^\ell} : \text{the membership coefficients according to the training set} \]
\[ \{i = 1 \ldots C\} : \text{the sum of the intra-class distances at the previous batch } B^\ell-1 \]

Output:
\[ \{w_t^\ell\}_{\ell = 1 \ldots L} : \text{the learned layers fuzzy relevance weights at iteration } t \]

Compute the distance \( d_{jk}^\ell \) for all pairs \( (x^\ell_j, x^\ell_k) \in B^\ell \) using (7).

Compute the intra-class distance \( H_t^\ell \) for each layer \( \ell \in L_t \) using (14) for class-independent approach or (19) for class-dependent approach.

Compute the fuzzy relevance weight \( w_t^\ell \) for each layer \( \ell \in L_t \) using (15) for class-independent approach or (20) for class-dependent approach.

3.3. Layer Pruning

After the learning process of the layer fuzzy relevance weights terminates, the layer pruning component is initiated in order to drop the non-relevant layers. The layer pruning concerns the intermediate layers. In other words, the input and output layers are excluded. The layer pruner algorithm finds the layer with the maximum weight and prunes all following layers because increasing the network depth would yield poor representation. This indicates that the representation is enhanced by increasing the network depth. Algorithm 2 describes the steps of the Layer Pruner algorithm.

Algorithm 2: Layer Pruning Algorithm

Input:
\[ \{w_t^\ell\}_{\ell = 1 \ldots L} : \text{the learned layers’ fuzzy relevance weights at iteration } t \]

Output:
\[ \mathcal{F}(; W^F) : \text{the pruned network with the learned network depth } L_t+1 \]
\[ \max_{\ell} \leftarrow \arg \max \{ \{w_t^\ell\}_{\ell = 1 \ldots L}\} \]
Prune all layers after \( \max_{\ell} \)

Finally, Algorithm 3 summarizes the main steps of the proposed approaches, and Figure 2 shows the proposed approach flowchart.

Algorithm 3: Depth-Adaptive Deep Neural Network

Input:
\[ \{x^j\}_{j = 1 \ldots N} : \text{input instances} \]
\[ \{y^j\}_{j = 1 \ldots N} : \text{the ground truth of the input instances} \]
\[ \mathcal{F}(; W^F) : \text{initialized deep feedforward neural network with initial depth } L \]

Output:
\[ \mathcal{F}(; W^F) : \text{trained deep feedforward neural network with learned depth } L_{Optimal} \]

REPEAT
Select a batch \( B^\ell \) from \( \{x^j\}_{j = 1 \ldots N} \)
Train the network \( \mathcal{F}(B^\ell; W^F) \)
Learn the layer fuzzy relevance weights using Algorithm 1
Prune layers using Algorithm 2
UNTIL \( \mathcal{F}(; W^F) \) convergences
4. Experimental Setting

4.1. Datasets

The proposed method is assessed using the standard MNIST handwritten digits dataset [42] and CIFAR-10 tiny images dataset [43]. MNIST includes 60,000 training samples and 10,000 test samples distributed equally over 10 classes. Each sample consists of a $28 \times 28$ grayscale image showing a single handwritten digit (0 to 9). Figure 3a shows sample images from the MNIST dataset. Alternatively, CIFAR-10 consists of 60,000 $32 \times 32$ color images distributed equally over 10 classes. The dataset is organized into 50,000 training images and 10,000 test images. Figure 3b shows sample images from the CIFAR-10 dataset representing the different classes.

Figure 3. Sample images from benchmark datasets. (a) Sample images from MNIST dataset; (b) Sample images from CIFAR-10 dataset.
4.2. Experiment Descriptions

4.2.1. Experiment 1: Empirical Investigation of the Effect of the Plain Network Depth on the Model Performance

First, the performance of plain networks, including different numbers of hidden layers, is evaluated. Specifically, the number of layers is tuned from 1 to 50. Figure 4 illustrates a sample plain network that consists of an input layer and 50 convolutional layers followed by one dense layer. The details and settings of this plain network architecture are reported in Table 1. As can be seen, all layers have the same filter size, which is 32, and the kernel size equals \((3 \times 3)\).

![Figure 4. CNN block diagram for 50 convolutional hidden layers.](image)

Table 1. Detailed architecture of 50 convolutional layer baseline (Plain-50).

<table>
<thead>
<tr>
<th>Layer Name</th>
<th>Output Size</th>
<th>50 Hidden Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv1</td>
<td>32 x 32</td>
<td>3 x 3, 32, stride 2</td>
</tr>
<tr>
<td>Conv2_x</td>
<td>32 x 32</td>
<td>[3 x 3, 32] x 50</td>
</tr>
<tr>
<td>Output</td>
<td>1 x 1</td>
<td>fully connected, softmax</td>
</tr>
</tbody>
</table>

4.2.2. Experiment 2: Performance Assessment of ANNA Approaches Applied to Plain Networks

The two ANNA approaches are applied to the 50 hidden layers’ model architecture described in Table 1. In order to evaluate the performance of the two obtained models, the class-independent ANNA and the class-dependent ANNA are trained in order to learn the depth of the network, the relevance weight of each layer, and the model parameters. As result, both proposed models converged to one hidden layer-network for the two considered datasets, namely MNIST [42] and CIFAR-10 [43]. This is in accordance with the findings of the previous experiment. Thus, the proposed approaches are able to learn automatically the optimal number of hidden layers. This is achieved through the appropriate learning of the layers’ relevance weights.

4.2.3. Experiment 3: Performance Assessment of ANNA Approaches Applied to ResNet Plain Models

In this experiment, the proposed ANNA approaches are evaluated using a plain baseline network inspired by VGG nets [44] and introduced in ResNet [36]. Specifically, for these models, layers having the same feature map size have the same number of filters. Alternatively, when the feature map size is halved due to the increase in the convolution stride, the number of filters is doubled. In particular, two versions of these plain networks are considered. Namely, these are the 18 convolutional layer baseline, referred to as plain-18, and the 34 convolutional layer baseline, referred to as plain-34. Table 2 reports the detailed architectures of these two plain networks for the CIFAR-10 dataset [43]. Moreover, Figure 5 shows the block diagrams of the 18-layer and 34-layer plain baseline networks.
Table 2. Detailed architecture of 18 convolutional layer baseline and 34 convolutional layer baseline models.

<table>
<thead>
<tr>
<th>Layer Name</th>
<th>Output Size</th>
<th>18 Layers</th>
<th>34 Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv1</td>
<td>19 × 19</td>
<td>7 × 7, 64, stride 2</td>
<td>3 × 3, maxpool, stride 2</td>
</tr>
<tr>
<td>Conv2_x</td>
<td>9 × 9</td>
<td>[3 × 3, 64] × 2</td>
<td>[3 × 3, 64] × 3</td>
</tr>
<tr>
<td>Conv3_x</td>
<td>5 × 5</td>
<td>[3 × 3, 128] × 2</td>
<td>[3 × 3, 128] × 4</td>
</tr>
<tr>
<td>Conv4_x</td>
<td>3 × 3</td>
<td>[3 × 3, 256] × 2</td>
<td>[3 × 3, 256] × 6</td>
</tr>
<tr>
<td>Conv5_x</td>
<td>2 × 2</td>
<td>[3 × 3, 512] × 2</td>
<td>[3 × 3, 512] × 3</td>
</tr>
<tr>
<td>Output</td>
<td>1 × 1</td>
<td>average pool, fully connected, softmax</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. Block diagrams of (a) the 18 convolutional layer baseline; and (b) the 34 convolutional layer baseline models.

ResNet-18 [36] and ResNet-34 [36] are the residual versions of plain-18 and plain-34, respectively, where the shortcut connections are inserted. As such, the performance results of applying the two proposed ANNAs approaches to plain-18 and plain-34 models are compared to the performances of ResNet-18 and ResNet-34. This experiment is conducted on MNIST [42] and CIFAR-10 [43] datasets. The same experimental settings as the previous experiments are considered.
4.2.4. Experiment 4: Performance Comparison of ANNA Approaches with State-of-the-Art Models

In this experiment, the performance of the proposed approaches is compared to the only state-of-the-art approach that seeks to learn the appropriate depth. In addition, it is also compared to two recent approaches that rely on bypassing and combining network layers. The considered state-of-the-art approaches are AdaNet [23], ResNet [36], and DenseNet [25]. They are the most closely related approaches to the proposed ones. In fact, as mentioned earlier, AdaNet [23] learns the network architecture while training the model. It starts with a plain model and iteratively expands the network by adding subnetworks iteratively. Specifically, it randomly generates two subnetworks at each iteration and selects one of the them to be appended to the existing network. Alternatively, ResNet [36] uses shortcut connections to skip two layers and conveys the output of one layer as the input to the next layers. DenseNet [25] also exploits shortcut connections, but with a different connectivity pattern. It uses direct connections from any layer to all subsequent layers. As such, between two stacked layer blocks, there is a transition layer that reduces the number of layers. Moreover, the proposed approaches are compared to the Attention-based Adaptive Spectral–Spatial Kernels A$^{2}$S$^{2}$K-ResNet [40].

For the purpose of performance assessment, two datasets are utilized. These are the MNIST dataset [42] and CIFAR-10 dataset [43]. The AdaNet model uses simple DNN, which consists of two dense layers. As for ResNet models, ResNet-18 and ResNet-34 are considered. Finally, for DenseNet, DenseNet-121 and DenseNet-169 are evaluated. These state-of-the-art models are compared to ANNA approaches when applied to plain-18, plain-34, and plain-50 networks.

4.3. Implementation

The experiments are conducted in the Ubuntu 20.04 platform with the following hardware specifications: 64 GB dual channel RAM, Intel i7 10700 CPU, and single Nvidia GeoForce RTX 2080 Ti GPU with 11 GB internal memory. The proposed ANNA approaches and all state-of-the-art approaches are implemented in Python using the Keras (backend tensorflow) library.

For the training process, a stochastic gradient descent optimizer is used. The hyper-parameters are selected according to the performance on the validation set (10% of the training set). As such, in order to select the learning rate, different values are used, which are 0.001, 0.003, 0.01, 0.03, and 0.1. Then, the learning rate 0.01 is selected with the default momentum value of 0.9. Moreover, the categorical cross-entropy loss function, suitable for multi-class classification, is employed. In addition, various batch sizes are used. Namely, 16, 32, 64, 128, and 256 batch sizes are investigated, and the batch size 256 is selected. Each considered plain network is trained using one epoch, and the conducted experiment is repeated 10 times.

5. Results and Discussion

5.1. Experiment 1: Empirical Investigation of the Effect of the Plain Network Depth on the Model Performance

The accuracies obtained using the plain-50 networks on the MNIST test set [42] are reported in Figure 6a. As can be seen, the accuracy degrades as the network depth increases. The experiment results also show that networks with the numbers of hidden layers between 1 and 5 outperform the other networks. Thus, the depths from 1 to 5 can be considered as optimal ones.
In order to illustrate the effect of the network depth on the classification performance with respect to the different categories/classes, we display in Figure 6b the per-class accuracy achieved by the plain network with respect to different depths. As can be seen, the accuracies of class “2” achieved by the models with depth from 1 to 19 are similar. Nevertheless, as the depth increases, the accuracy fluctuates. Alternatively, for the other categories, the maximum accuracy is attained when using a single layer network. As one can notice, the accuracy starts decreasing slightly when the number of network layers is less than 7. A drastic drop of the accuracy also occurs when increasing the number of layers from 7 to 10. Note that a slight increase of the performance is reported when adding one layer to the 8-layer-depth network. In addition, 10-layer- and 15-layer-depth models exhibit some instability in their performance. Adding one more layer to a 15-layer-depth models results in a significant drop on the performance. Furthermore, increasing the number of layers to 50 causes instability in the corresponding models’ performance. Based on these experimental results, we can conclude that a single hidden-layer model yields the best results for the considered datase, and outperforms the other deeper plain models.

The previously mentioned experiments conducted on the MNIST [42] dataset were conducted on CIFAR-10 [43] dataset. The accuracies achieved using CIFAR-10 [43] are shown in Figure 7a. The obtained results affirm the findings of the MNIST [42] experiment. Specifically, the accuracy decreases as the network depth increases. Moreover, the results show that the networks of depths between 1 and 3 achieved better performance than deeper models.

As can be seen in Figure 7b, except for class “cat”, all other classes achieved the maximum accuracy when using 1 to 3-layer-depth networks. Moreover, adding one layer to a 7-layer-depth network causes a slight decrease in the performance. Furthermore, 8 to 50-layer-depth models exhibit instability in their performance. As for class “cat”, the performance suffers from fluctuations as the depth changes. However, the general accuracy of this category enhances as the depth increases. This behavior explains the overall accuracy drop for the CIFAR-10 dataset [43].

Figure 6. MNIST dataset performance results. (a) Accuracy achieved by the plain-50 network with respect to different depths using the MNIST dataset; (b) Per-class accuracies achieved by the plain-50 network with respect to different depths using the MNIST dataset.
5.2. Experiment 2: Performance Assessment of ANNA Approaches Applied to Plain Networks

Figure 8 exhibits sample layer relevance weights learned by the proposed class-independent ANNA on the MNIST dataset [42] after training the model using 1, 4, 6, and 15 batches. As can be seen, after training the model using one batch, the maximum weight is assigned to layer 46. Therefore, according to the proposed pruning approach, all hidden layers from 47 to 50 are pruned, which results in a 46 hidden-layer network. Similarly, after training the model using 4 batches, layer 27 is assigned the maximum weight. Hence, all hidden layers from 28 to 46 are pruned, which results in a 27 hidden-layer network. In the same way, after training the model using 6 batches, layers from 23 to 27 are pruned, which yields a 22 hidden-layer network. After 15 batches are conveyed to the network, layers from 9 to 22 are pruned, which results in 8 hidden-layer network. At the end, the learning process converges to a one hidden-layer network when the model is trained using 41 batches.

Similarly, Figure 9 depicts sample layer weights learned using the class-dependent ANNA on the MNIST dataset [42] after training the model using 1, 4, 6, and 15 batches. As can be seen, after training the model using the first batch, the last four layers are pruned, since layer 46 has the largest weight. After using 4 batches, layer 27 is assigned the largest weight, thus all subsequent layers are pruned. Layer 22 becomes the most relevant layer after batch 6, hence layers from 23 to 27 are pruned. In the same way, the highest relevance weight is assigned to layer 8 after 15 batches are conveyed to the network. Therefore, layers 9 to 22 are pruned. Later on, when using 9 batches, the model converges to one single hidden-layer network. As such, the number of learnable parameters decreases from 713,610, at the beginning of the training process, to 260,458 when the proposed approach converges. In fact, the proposed class-dependent and class-independent ANNA approaches yield lightweight models, reduce the overfitting risk, and improve the model generalization.
network layers and their impact on the discrimination power of the learned models. Furthermore, the obtained layer fuzzy relevance weights allow a straightforward interpretation of the network behavior. In other words, they reflect the importance of the different layer relevance models, either class-dependent or class-independent, guided the model generalization. Therefore, layers 9 to 22 are pruned. Later on, when using 9 batches, the model converges to one single hidden layer after batch 6, hence layers from 23 to 27 are pruned. In the same way, the layer 22 becomes the most relevant layer after batch 6, hence layers from 23 to 27 are pruned. As can be seen, after training the model using the first batch, the last four layers are pruned. Thus all subsequent layers are pruned. Layer 22 becomes the most relevant layer after batch 6, hence layers from 23 to 27 are pruned. In the same way, the layer 22 becomes the most relevant layer after batch 6, hence layers from 23 to 27 are pruned.

Figure 8 reports the overall accuracy and the per-class accuracies obtained using the class-independent and class-dependent ANNA as when using the MNIST dataset [42]. As can be seen, similar performance is attained with respect to all digit categories [42]. Moreover, both ANNA approaches yield comparable classification performance. This proves that the layer relevance models, either class-dependent or class-independent, guided the learning process towards the optimal network depth dynamically while training the model. Furthermore, the obtained layer fuzzy relevance weights allow a straightforward interpretation of the network behavior. In other words, they reflect the importance of the different network layers and their impact on the discrimination power of the learned models.
The class-independent and class-dependent ANNA approaches converge to one hidden-layer network. This is in accordance with the findings of the empirical experimentation conducted in experiment 1. Moreover, the proposed class-independent and class-dependent ANNAs yield lightweight models, since the number of learnable parameters decreased from 790,986 to 337,834.

Figures 11 and 12 show sample weights learned by class-independent and class-dependent ANNA, respectively, when using the CIFAR-10 dataset [43]. These sample weights are obtained after training the model using 1, 5, 6, and 7 batches. Both proposed approaches converge to one hidden-layer network. This is in accordance with the findings of the empirical experimentation conducted in experiment 1. Moreover, the proposed class-independent and class-dependent ANNAs yield lightweight models, since the number of learnable parameters decreased from 790,986 to 337,834.

The class-independent and class-dependent ANNAs achieved comparable time performance, with 556.116 and 554.564 s, respectively, to complete the training process. Evidently, learning the network depth using one of the proposed ANNA approaches is more efficient than empirical investigation of the optimal depth, where 6720 s was required to conduct the whole experiment, aiming at determining manually the optimal network depth.
while attaining an accuracy of 79% for the “automobile” class. As mentioned in exper-
iment 1, the accuracy obtained for class “cat” improves as the network depth increases.
This explains the poor performance of the one hidden-layer network for class “cat”. This
affected the overall accuracy of both ANNA approaches.

Class-independent and class-dependent ANNA approaches achieved comparable classi-
fication performances. To be more accurate, for the CIFAR-10 dataset [43], the class-dependent
ANNA approach achieved a slightly better performance than the class-independent ANNA.
However, both the ANNA approaches succeeded in determining an appropriate network
depth and enhancing the overall classification performance through the learning process.

Figure 12. Layers’ relevance weights learned by the proposed class-dependent ANNA on CIFAR-10
dataset after training the model using: (a) 1 batch; (b) 5 batches; (c) 6 batches; (d) 7 batches.

The overall accuracy and the per-class accuracies obtained using the class-independent
and the class-dependent ANNAS when using CIFAR-10 dataset [43] are reported in Figure 13.
As can be seen, the performance varies with respect to the image categories. For instance,
the models achieved an accuracy of 10% and 30% for “cat” and “truck” classes, respectively,
while attaining an accuracy of 79% for the “automobile” class. As mentioned in experi-
ment 1, the accuracy obtained for class “cat” improves as the network depth increases.
This affected the overall accuracy of both ANNA approaches.

Class-independent and class-dependent ANNA approaches achieved comparable classi-
fication performances. To be more accurate, for the CIFAR-10 dataset [43], the class-dependent
ANNA approach achieved a slightly better performance than the class-independent ANNA.
However, both the ANNA approaches succeeded in determining an appropriate network
depth and enhancing the overall classification performance through the learning process.

Figure 13. Overall accuracy and per-class accuracies obtained using the proposed ANNA approaches
when using CIFAR-10 dataset.
5.3. Experiment 3: Performance Assessment of ANNA Approaches Applied to ResNet Plain Models

Table 3 reports the achieved performances on the MNIST dataset [42] in terms of accuracy, error rate, F1-measure, and area under curve (AUC). Specifically, it compares the results of the application of ANNAS to plain-18 and plain-34, to the results of plain-18, plain-34, ResNet-18, ResNet-34, and plain-50 CNN networks. As can be seen, the proposed ANNA approaches succeed in guiding the learning process into a lightweight version of plain-18 and plain-34. More specifically, it converges to 2-layer network in the case of plain-18 and a 3-layer network in the case of plain-34. Moreover, the learned lightweight models outperformed the plain baseline networks and ResNet models. In fact, the 2-convolutional layer network learned by the class-dependent ANNA using the plain-18 model achieved the highest accuracy of 89.9%.

Table 3. Performance measure comparison on MNIST dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Error</th>
<th>F1-Score</th>
<th>AUC</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain-18</td>
<td>61.04</td>
<td>38.96</td>
<td>0.55</td>
<td>0.85</td>
<td>18 layers</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>61.67</td>
<td>38.33</td>
<td>0.59</td>
<td>0.87</td>
<td>18 layers</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-18</td>
<td>87.82</td>
<td>12.18</td>
<td>0.87</td>
<td>0.98</td>
<td>2 layers</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-18</td>
<td><strong>89.9</strong></td>
<td><strong>10.1</strong></td>
<td><strong>0.9</strong></td>
<td><strong>0.99</strong></td>
<td>2 layers</td>
</tr>
<tr>
<td>Plain-34</td>
<td>14.48</td>
<td>85.52</td>
<td>0.07</td>
<td>0.5</td>
<td>34 layers</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>46.59</td>
<td>53.41</td>
<td>0.4</td>
<td>0.77</td>
<td>34 layers</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-34</td>
<td>86.18</td>
<td>13.82</td>
<td>0.86</td>
<td>0.98</td>
<td>3 layers</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-34</td>
<td><strong>87.97</strong></td>
<td><strong>12.03</strong></td>
<td><strong>0.88</strong></td>
<td><strong>0.98</strong></td>
<td>3 layers</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-50</td>
<td>87.58</td>
<td>12.42</td>
<td>0.88</td>
<td>0.98</td>
<td>2 layers</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-50</td>
<td>86.61</td>
<td>13.39</td>
<td>0.87</td>
<td>0.98</td>
<td>2 layers</td>
</tr>
</tbody>
</table>

Bold number indicates best performance.

As depicted in Table 3, the accuracy achieved by the plain-34 baseline network did not exceed 14.48%. This is an expected result due to the degradation problem, which affects networks with high depth. In fact, ResNet models tackled this problem through the use of skip connections. As can be seen from Table 3, ResNet does perform better than the plain model. Nevertheless, the class-independent and class-dependent ANNAS were able to better overcome the plain-34 network degradation problem by achieving an accuracy of 86.18% and 87.97%, respectively. Similar results can be seen in the performances of plain-18, ResNet-18 and ANNAS approaches. Therefore, the compact models learned by ANNAS proposed approaches outperformed the ResNet-18 and ResNet-34 models.

Comparing these results with the performance of applying ANNAS on the 50-layer CNN model shows that a simple network design with an optimal depth learned by the proposed ANNAS approaches outperforms plain-18, ResNet-18, plain-34, and ResNet-34. The analysis of these results based on the accuracy performance holds for the other performances depicted in Table 3, namely the error rate, the F1-score, and AUC. In fact, the considered performance measures are consistent with each other.

Figure 14a compares the per-class accuracies of ResNet models with class-dependent ANNA applied on plain-18 and plain-34. As can be seen, for categories “1”, “2”, and “3”, the class-dependent ANNA models outperformed ResNet-18 (and ResNet-34 for category “3”) by a significant margin. For the remaining categories, the class-dependent ANNA models presented a competitive or better performance than ResNet models. Figure 14b shows the receiver operating characteristic curve (ROC) for the ResNet models and class-dependent ANNA models. As exhibited by Figure 14, the proposed ANNAS approaches perform significantly better than ResNet models on the MNIST dataset [42].
Appl. Sci. 2023, 13, 398

(a) Overall accuracy and per-class accuracies achieved by ResNet and class-dependent ANNA on MNIST dataset; (b) ResNet and class-dependent ANNA ROC curves on MNIST dataset.

Table 4 shows the achieved accuracies on the CIFAR-10 dataset [43]. The proposed ANNA approaches succeeded in guiding the learning process on the plain-18 network into a compact one-layer model. Specifically, class-independent and class-dependent ANNAs with plain-18, in one epoch, achieved an accuracy of 56.48% (17% higher than ResNet-18) and 56.14% (16% higher than ResNet-18), respectively. Furthermore, applying ANNA approaches on the plain-34 network resulted in a 2-layer model. The obtained models outperformed ResNet-34, with a difference in terms of accuracy of 9% for class-independent ANNA and 7% for class-dependent ANNA. In order to investigate the effect of per-class accuracies on the overall performance, Figure 15a presents the per-class accuracies of ResNet-34, with a difference in terms of accuracy of 9% for class-independent ANNA and 7% for class-dependent ANNA. In order to investigate the effect of per-class accuracies on the overall performance, Figure 15a presents the per-class accuracies of ResNet in comparison with class-dependent ANNA applied on plain-18 and plain-34. As can be noticed, the achieved accuracies by the class-independent ANNA models with respect to classes “cat”, “deer”, and “dog” are significantly higher than the accuracies obtained by the ResNet models. This affects the overall performance. The ROC curves of the ResNets and class-independent ANNAs on the CIFAR-10 dataset [43], as displayed in Figure 15b, show consistent results with the obtained accuracies and F1-scores.

Table 4. Performance measure comparison on CIFAR-10 dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Error</th>
<th>F1-Score</th>
<th>AUC</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain-18</td>
<td>44.28</td>
<td>55.72</td>
<td>0.44</td>
<td>0.86</td>
<td>18 layers</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>48.2</td>
<td>51.8</td>
<td>0.47</td>
<td>0.88</td>
<td>18 layers</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-18</td>
<td>56.48</td>
<td>43.52</td>
<td>0.56</td>
<td>0.91</td>
<td>1 layer</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-18</td>
<td>56.14</td>
<td>43.86</td>
<td>0.56</td>
<td>0.91</td>
<td>1 layer</td>
</tr>
<tr>
<td>Plain-34</td>
<td>26</td>
<td>74</td>
<td>0.18</td>
<td>0.76</td>
<td>34 layers</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>49.36</td>
<td>50.64</td>
<td>0.48</td>
<td>0.88</td>
<td>34 layers</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-34</td>
<td>53.91</td>
<td>46.09</td>
<td>0.53</td>
<td>0.9</td>
<td>2 layers</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-34</td>
<td>52.9</td>
<td>47.1</td>
<td>0.53</td>
<td>0.9</td>
<td>2 layers</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-50</td>
<td>47.71</td>
<td>52.29</td>
<td>0.45</td>
<td>0.87</td>
<td>2 layers</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-50</td>
<td>49.2</td>
<td>50.8</td>
<td>0.47</td>
<td>0.88</td>
<td>2 layers</td>
</tr>
</tbody>
</table>

Bold number indicates best performance.
As shown by the reported results in Figure 16, the learned layer weights increase as the depth increases. This provides an interpretation of the importance of the layers. In fact, it indicates that the deeper layers are more relevant than initial ones since they learn more complex and abstract features.
with state-of-the-art methods, the number of floating-point operations (FLOPs) were calculated for each model. As it can be seen in Table 7, the compact models of the ANNA approaches have lower complexity than the state-of-the-art models. Particularly, the model resulting from applying ANNA approaches to plain-18 achieved the least number of FLOPs (0.00666) and a high accuracy of 89.9% on the MNIST dataset. Similarly, they achieved the highest accuracy with the lowest FLOPs (0.00695) on CIFAR-10. Thus, the

Figure 16. Layers’ relevance weights learned by ANNA approaches when applied to Plain-18 and Plain-34 on MNIST and CIFAR-10 datasets.

5.4. Experiment 4: Performance Comparison of ANNA Approaches with State-of-the-Art Models

Table 6 reports the performance of all considered models. As can be seen, on the MNIST dataset [42], AdaNet models achieved the highest performances with an accuracy equal to 92.45%. Nevertheless, ANNA models gave competitive results, with an accuracy of 89.9% for class-dependent ANNA applied to plain-18, and around 87% for the other models. Moreover, the proposed approaches outperformed ResNet, DenseNet, and $A^2S^2K$-ResNet models for MNIST data. As for the CIFAR-10 dataset [43], ANNA approaches applied to plain-18 surpassed the accuracy of the state-of-the-art approaches by at least by 13%.

Table 6. Performance measure comparison of ANNA and state-of-the-art approaches on MNIST dataset and CIFAR-10 dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>MNIST</th>
<th></th>
<th>CIFAR-10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Error</td>
<td>Accuracy</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>48.2</td>
<td>51.8</td>
<td>48.2</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>46.59</td>
<td>53.41</td>
<td>49.36</td>
</tr>
<tr>
<td>AdaNet</td>
<td>92.45</td>
<td>7.55</td>
<td>15.7</td>
</tr>
<tr>
<td>DenseNet-121</td>
<td>24.5</td>
<td>73.5</td>
<td>49.86</td>
</tr>
<tr>
<td>DenseNet-169</td>
<td>19.1</td>
<td>80.9</td>
<td>42.86</td>
</tr>
<tr>
<td>$A^2S^2K$-Resnet</td>
<td>40.78</td>
<td>59.22</td>
<td>31.04</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-18</td>
<td>87.82</td>
<td>12.18</td>
<td>56.48</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-18</td>
<td>89.9</td>
<td>10.14</td>
<td>56.14</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-34</td>
<td>86.18</td>
<td>13.82</td>
<td>53.91</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-34</td>
<td>87.97</td>
<td>12.03</td>
<td>52.9</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-50</td>
<td>87.58</td>
<td>12.42</td>
<td>47.71</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-50</td>
<td>86.61</td>
<td>13.39</td>
<td>49.2</td>
</tr>
</tbody>
</table>

Bold number indicates best performance.
proposed approach succeeded in learning lightweight models with high performance and low complexity.

Table 7. Computational complexity and time comparison of ANNA and state-of-the-art approaches on MNIST dataset and CIFAR-10 dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>MNIST</th>
<th>CIFAR-10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FLOPs (G)</td>
<td>Time (s)</td>
</tr>
<tr>
<td></td>
<td>Training</td>
<td>Testing</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>0.0713</td>
<td>23.89</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>0.147</td>
<td>37.88</td>
</tr>
<tr>
<td>AdaNet</td>
<td>0.00953</td>
<td>12.3</td>
</tr>
<tr>
<td>DenseNet-121</td>
<td>0.102</td>
<td>57.5</td>
</tr>
<tr>
<td>DenseNet-169</td>
<td>0.124</td>
<td>80.55</td>
</tr>
<tr>
<td>A²S²K-Resnet</td>
<td>0.0984</td>
<td>15.09</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-18</td>
<td>0.00666</td>
<td>292.26</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-18</td>
<td>0.00666</td>
<td>263.29</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-34</td>
<td>0.0114</td>
<td>523.73</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-34</td>
<td>0.0114</td>
<td>412.88</td>
</tr>
<tr>
<td>Class-Independent ANNA + Plain-50</td>
<td>0.0155</td>
<td>659.91</td>
</tr>
<tr>
<td>Class-Dependent ANNA + Plain-50</td>
<td>0.0155</td>
<td>664.83</td>
</tr>
</tbody>
</table>

Concerning the training time, as shown in Table 7, ANNA approaches exhibit large training time compared to the state-of-the-art approaches. This can be explained by the fact that the proposed approaches start with an over-estimation of the number of layers and then discard the non-relevant ones while training the model, whereas the learned optimal number of layers on the considered datasets is between 1 to 3. Nevertheless, the training is done offline; thus, it is not significant for real-time applications that use the already trained model.

Regarding the testing time, ANNAs approaches outperform the considered state-of-the-art approaches. For instance, the DenseNet-121 testing time (on both datasets) is four times (at least) larger than all ANNA model testing times. Moreover, despite the good performance of AdaNet on the MNIST dataset [42], it has a large testing time in comparison with the ANNA models. This also applies for the CIFAR-10 dataset [43]. This can be explained by the light weight of ANNA models. In fact, the learned model architecture has fewer layers than the other models.

6. Conclusions

Designing DNNs that exhibit the appropriate level of complexity and depth is a challenging task. In this paper, we proposed two adaptive deep learning approaches intended to learn the optimal DNN depth. Moreover, both approaches assign fuzzy relevance weights to the network layers. In fact, the learned weights yield better interpretability of the model and the extracted features. Furthermore, the proposed approaches train the DNN model and learn the layers fuzzy relevance weights jointly. Specifically, the DNN architecture is updated by pruning the non-relevant layers. The learning process is formulated as an optimization of a novel cost function. The proposed approaches were assessed using standard datasets and performance measures. The experiments proved that the proposed approaches can automatically learn the optimal number of layers. Moreover, when compared to state-of-the-art approaches, ANNAs gave competitive results on MNIST and outperformed the other models on CFAR-10. Furthermore, it has been shown that the proposed approaches overtake the state-of-the-art approaches in terms of time efficiency and number of learnable parameters.

As future works, we suggest research to learn the architecture of deep learning networks by incrementally adding layers instead of starting with an overestimated number,
and the, discarding the non-relevant layers while training the model. This would reduce the training time, since the experiments showed that networks with a small optimal number of layers perform better than deeper ones.

**Author Contributions:** Conceptualization, A.A., O.B. and M.M.B.I.; methodology, A.A., O.B. and M.M.B.I.; software, A.A.; validation, A.A. and O.B.; formal analysis, A.A. and O.B.; investigation, A.A. and O.B.; data curation, A.A.; writing—original draft preparation, A.A. and O.B.; writing—review and editing, A.A. and O.B.; visualization, A.A. and O.B.; supervision, O.B. and M.M.B.I. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not Applicable.

**Informed Consent Statement:** Not Applicable.

**Data Availability Statement:** Publicly available datasets were analyzed in this study. MNIST data can be found here: [http://yann.lecun.com/exdb/mnist/](http://yann.lecun.com/exdb/mnist/) (accessed on 1 November 2020); CIFAR-10 data can be found here: [https://www.cs.toronto.edu/~kriz/cifar.html](https://www.cs.toronto.edu/~kriz/cifar.html) (accessed on 1 April 2021).

**Acknowledgments:** The authors are grateful for the support of the Research Center of the College of Computer and Information Sciences, King Saud University. The authors thank the Deanship of Scientific Research and RSSU at King Saud University for their technical support.

**Conflicts of Interest:** The authors declare no conflict of interest.

**References**


**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.