Efficient Adaptive Inference for Deep Convolutional Neural Networks using Hierarchical Early Exits

Nikolaos Passalis[a], Jenni Raitoharju[b], Anastasios Tefas[a], Moncef Gabbouj[b]

[a] *Department of Informatics, Aristotle University of Thessaloniki, Greece*
[b] *Faculty of Information Technology and Communication Sciences, Tampere University, Finland*

Abstract

Early exits are capable of providing deep learning models with adaptive computational graphs that can readily adapt on-the-fly to the available resources. Despite their advantages, existing early exit methods suffer from many limitations which limit their performance, e.g., they ignore the information extracted from previous exit layers, they are unable to efficiently handle feature maps with large sizes, etc. To overcome these limitations we propose a Bag-of-Features (BoF)-based method that is capable of constructing efficient hierarchical early exit layers with minimal computational overhead, while also providing an adaptive inference method that allows for early stopping the inference process when the network is confident enough for its output, leading to significant performance benefits. To this end, the BoF model is extended and adapted to the needs of early exits by constructing additive shared histogram spaces that gradually refine the information extracted from the various layers of a network, in a hierarchical manner, while also employing a classification layer reuse strategy to further reduce the number of parameters needed per exit layer. Note that the proposed method is generic and can be readily combined with any neural network architecture. The effectiveness of the proposed method is demonstrated using five different image datasets, proving that early exits can be readily transformed into a practical tool, which can be effectively used in various real-world embedded applications.

Keywords: Adaptive Inference, Early Exits, Bag-of-Features, Deep Convolutional Neural Networks, Hierarchical Representations

1. Introduction

The advent of Deep Learning (DL) led to spectacular applications, including but not limited to autonomous cars [1], accurate medical diagnosis and disease prognosis [2, 3], intelligent buildings [4], and powerful methods for human-computer interaction [5, 6]. Despite its enormous success in the aforementioned domains, DL suffers from a significant drawback: DL models often consist of millions of parameters and, as a result, they require using powerful and energy-consuming hardware both

[*] Corresponding author: Nikolaos Passalis, *Email address:* passalis@csd.auth.gr

for training and deployment. This limitation significantly reduces their flexibility, increases the deployment costs, and slows down the penetration of DL in many domains, where these requirements cannot be satisfied. Many methods have been proposed in the literature for overcoming the aforementioned drawbacks, ranging from quantization and model compression approaches [7] to neural network pruning [8] and knowledge distillation methods [9, 10, 11].

Even though these methods can indeed lead to developing faster and more lightweight DL models, they also often lead to models that perform worse than the original ones (in terms of accuracy). Furthermore, the developed models are *static*, i.e., they are unable to adapt to the available computational resources on the fly. However, in many applications there is a need for *dynamic* networks that are able to adapt on demand to the current conditions. This can be better understood by the following example. Consider a real-time face recognition system, where the faces that are depicted in an acquired frame must be accurately recognized under strict real time constraints. Note that the time that will be spent for the recognition process is proportional to the number of faces that appear in a given frame. If an estimate for the maximum number of persons that will appear in a frame is known beforehand, then we can appropriately design a lightweight DL model that will always work within the given time constraints. However, what will happen if more persons appear in a given frame? Most of the existing methods will spend the predefined portion of time for recognizing the depicted faces and then either stop without recognizing the rest of the people or spend more time than the allowed for the recognition, reducing the quality of service. On the other hand, a dynamic model would be able to adapt to the available load by reducing the time needed for the recognition of each person, possibly by providing slightly less refined and accurate predictions, meeting the real time constraints of the system.

Therefore, in such applications we need models that will be able to dynamically adapt to the available load, e.g., by being able to provide faster (and possibly less accurate) predictions when the load is higher (e.g., when a lot of persons appear in a frame) and more refined and accurate (yet slower) predictions when the load is lower (e.g., when only a few persons appear in a frame). It is worth noting that apart from this case, the need for dynamically adapting a model to the current computational resources without retraining occurs in many other applications, e.g., deploying mobile applications on smartphones where it is usually infeasible to train separate models according to the computational resources available to the vast variety of different phone models [12].

Among the most promising approaches for overcoming these limitations are DL models with adaptive computational graphs, such as [13, 14, 15]. These models provide an easy and straightforward way to dynamically adapt the model on-the-fly to the available computational resources by altering the complexity of the model's graph, i.e., by choosing a different computational path according to the available resources. One straightforward way to achieve this is by adding early exits at various layers of the network [12, 13, 15]. These early exits allow for estimating the final output of the network at various points of the inference process, without having to feed-forward through the whole network. Early exits provide, in theory, a solid way to adapt the inference to the available resources. Unfortunately, they do not always lead to acceptable performance [15], since they have to deal with enormous feature maps (especially for the earlier layers of a convolutional neural network). To this end, aggressive subsampling methods are usually employed, e.g., global average pooling [16]. As a result, these methods ignore both the spatial information and the distribution of the extracted feature vectors, reducing the performance of early exits (in terms of accuracy). Even though this problem is partly addressed in [12] by using a series of densely connected structures, this also requires a significant number of structural changes in the architecture of a network and cannot be easily used with existing neural networks. Furthermore, most of the existing early exit-based approaches completely ignore the information extracted by the previous exit layers, throwing away information that is readily available and can be potentially used to further increase the prediction accuracy for the subsequent exit layers.

In this work we propose using the Bag-of-Features (BoF) model [17], for compiling compact, yet rich and discriminative representations from the feature maps of each layer where an early exit is used. The BoF model was originally proposed to provide compact representations for complex objects that consist of multiple feature vectors, while keeping as much information as possible about the corresponding object. More specifically, BoF works as follows: a) A feature extractor is used to extract feature vectors from an object, e.g., multiple SIFT vectors can be extracted from an image [18]. b) Then, the extracted feature vectors are quantized into a number of bins, defined by a set of vectors called *codewords*. c) Finally, a compact histogram representation is extracted for each object simply by counting the number of feature vectors that were quantized into each bin. In this way, BoF provides an efficient way to compress large collections of feature vectors, such as those extracted from the earlier layers of a neural network, into compact histogram representations that can overcome the limitations of the existing methods used for providing early exits.

However, using the BoF model for providing efficient early exit implementations is not straightforward. First, note that the histogram space compiled at each exit layer is different, which can prohibit the efficient construction of hierarchical representations that take into account the representations extracted from the previous early exits, significantly limiting the performance of the model. To overcome this limitation we propose extracting additive hierarchical histogram representations, that gradually refine the estimations from the previous layers, by implicitly constructing a shared histogram space for all the exit layers. Thus, the codewords are learned in such way that the histogram representations extracted from the various levels of the network are compatible with each other. This also allows for having a common classification layer, which is shared among all the early exits, further reducing the number of parameters needed for adding an early exit to the network. Furthermore, the proposed method also supports dynamic adaptive inference that allows for stopping the inference process at an early exit, if the network is confident enough for its decision. This allows for spending less time for classifying easy input samples, while spending more time processing the harder ones. As it is demonstrated through this paper, these modifications lead to enormous improvements over both traditional static DL models, as well as over existing early exit approaches.

The main contribution of this work is proposing a Bag-of-Features-based approach, fully adapted to the needs of early exits, overcoming most of the limitations of existing related approaches. More specifically, in this paper:

1. A BoF-based formulation [19], which is capable of a) maintaining more information regarding the distribution of the extracted feature vectors and b) keeping more spatial information in the extracted representation, than the existing early exit methods, is proposed. Note that the latter is especially important for earlier exits, where the receptive field of the convolutional layers is usually smaller.
2. An efficient hierarchical aggregation scheme, that works by implicitly constructing a common histogram representation space and then gradually refining the estimations of the network, is proposed. This allows for taking into account the information that was already extracted by the earlier layers. Note that most of the existing formulations ignore this information [15].
3. A classification layer reuse approach is employed to further reduce the number of parameters required for each early exit, minimizing the cost of adding additional exit layers.
4. An adaptive classification approach, that allows for selecting the most appropriate early exit according to the difficulty of each input sample, is proposed, allowing for reducing the load to the system, as well as reducing the energy consumption of DL models.

It is worth noting that the proposed method can be readily combined with any neural network architecture, since it requires no model-specific changes to the base network. Finally, the ability of the proposed method to transform early exits into a practical tool, that can be applied in many challenging real-world applications, is demonstrated using extensive experiments on five datasets.

The rest of this paper is structured as follows. First, the related work is briefly discussed in Section 2, while the proposed method is analytically derived in Section 3. Then, the proposed method is extensively evaluated in Section 4, while conclusions are drawn in Section 5.

2. Related Work

This work is related to DL models with adaptive computational graphs, which allow for dynamically adapting the model to the available computational resources/difficulty of the input samples. Using early exits is perhaps the most straightforward way to provide DL models with adaptive computational graphs [14], as demonstrated in [12, 13, 15, 16]. BranchyNet [13] proposed to include exit layers at various points of the computational graph of the model by including branches that are composed of 3×3 convolution layers, followed by the appropriate classification layers. A similar approach, which skips the added complexity of convolutional layers by employing global average pooling layers to downsample the input to the each early exit was further examined in Elastic Networks [15, 16]. However, these methods often do not lead to acceptable performance, since they discard valuable spatial information, as well as useful information regarding the distribution of the extracted feature vectors, as experimentally demonstrated in Section 4. Even though this problem is partly addressed in [12] by using a series of densely connected structures, this also requires a significant number of structural changes in the architecture of a network and cannot be easily used with existing neural networks, rendering this approach significantly harder to implement compared to the rest of the proposed early exit approaches.

In this work we provide a powerful method that can overcome these limitations by proposing a hierarchical BoF-based approach that can also keep all the information extracted from the various exit layers of a neural network with minimal additional overhead. Also, it is worth noting that early exits have been used in past for a different purpose: early exits were used for reinforcing the gradients at various layers, reducing in this way the training problems that are related to vanishing gradient phenomena [20]. This work is also related to the traditional dictionary learning methods for the BoF model, for which a very rich literature exists [21, 22, 23]. However, these approaches have not been designed to work with early exits, i.e., dealing with features extracted from various levels of a network, constructing common representation spaces for these features, using additive hierarchical representations and reusing the same classification layers at various points of the same network in order to provide efficient early exit implementations. In [24], the use of BoF-based pooling was examined for the first time in the context of early exits. However, compared to the proposed method, this approach leads to significantly larger intermediate representations and requires more parameters, especially when applied in a hierarchical setting, as also demonstrated in the sensitivity analysis provided in Section 4.

To the best of our knowledge, this is the first work in which a hierarchical BoF-based formulation is employed for constructing additive histogram representation spaces that can be used for providing efficient hierarchical early exits. At the same time, the proposed method also provides an efficient adaptive inference mechanism that allows for effectively selecting the most appropriate early exit, given that the network is confident enough for its classification decision.

3. Proposed Method

First, the necessary background and notation are introduced in Subsection 3.1. Then, the proposed Bag-of-Features-based method for providing efficient early exit implementations is analytically derived in Subsection 3.2. Furthermore, a powerful, yet efficient hierarchical early exit scheme, that builds upon the proposed BoF method, is described in Section 3.3. Finally, an adaptive inference method, that allows for early stopping the inference process when the network is confident enough, is proposed in Subsection 3.4.

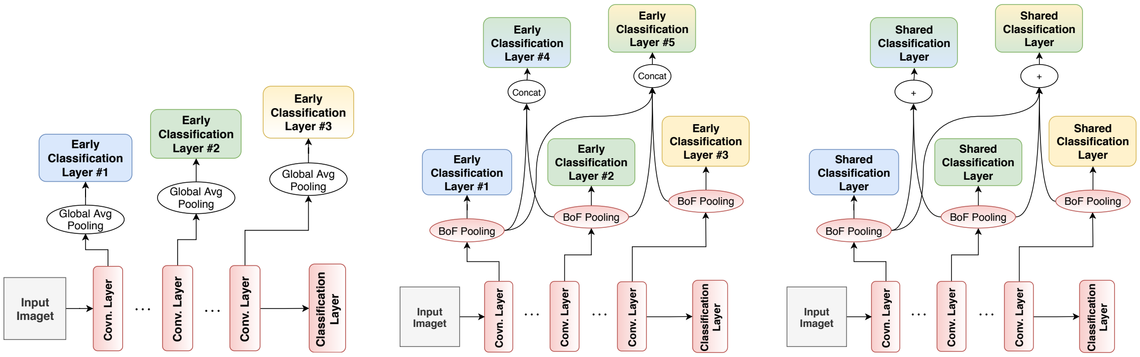
3.1. Background and Notation

Let $f_{\mathbf{W}}(\mathbf{x}, i)$ denote the output of the i-th layer of a neural network composed of m layers, where the notation $\mathbf{W}$ is used to denote the parameters of the network and $\mathbf{x}$ denotes the input to the neural network. This work focuses on convolutional neural networks, where the input to the neural network is an image, i.e., $\mathbf{x} \in \mathbb{R}^{W \times H \times C}$, where W, H and C are the width, height and number of channels of the image respectively. Note that this is without loss of generality, since the proposed method can be also readily applied for networks operating on other signals, such as audio [25], time-series [26], etc. Also, note that the output of the i-th convolutional layer is a feature map that is also denoted by $\mathbf{y}^{(i)} = f_{\mathbf{W}}(\mathbf{x}, i) \in \mathbb{R}^{W_i \times H_i \times C_i}$, where W_i and H_i are the width and height of the feature map extracted from the i-th layer of the network, while C_i denotes the number of filters used in the corresponding convolutional layer. The final output of the neural network, denoted by $\mathbf{y} = f_{\mathbf{W}}(\mathbf{x}, m) \in \mathbb{R}^{N_C}$, is a vector containing the probabilities that the input image $\mathbf{x}$ belongs to each of the N_C classes. Note that even though the aforementioned setup considers only classification problems, it is straightforward to extend it to handle other tasks as well, such as regression tasks.

Given a training set of N images $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, along with a target vector $\mathbf{t}_i \in \mathbb{R}^{N_C}$, which encodes the classification label for each image, then the network is trained using back-propagation to minimize a loss function $\mathcal{L}$:

$$\mathbf{W}' = \mathbf{W} - \eta \sum_{j=1}^{N} \frac{\partial \mathcal{L}(f_{\mathbf{W}}(\mathbf{x}_j, m), \mathbf{t}_j)}{\partial \mathbf{W}}, \tag{1}$$

where the notation $\mathbf{W}'$ is used to denote the updated parameters of the neural network and η is the used learning rate. Usually the optimization is performed in batches, the classification target $\mathbf{t}_i$ is a one-hot encoding of the class of each sample, while the cross-entropy loss is employed as the loss


Input Imaget
Covn. Layer
Conv. Layer
Conv. Layer
Classification Layer
Global Avg Pooling
Global Avg Pooling
Global Avg Pooling
Early Classification Layer #1
Early Classification Layer #2
Early Classification Layer #3
...
...
Input Imaget
Covn. Layer
Conv. Layer
Conv. Layer
Classification Layer
BoF Pooling
BoF Pooling
BoF Pooling
Concat
Concat
Early Classification Layer #1
Early Classification Layer #2
Early Classification Layer #3
Early Classification Layer #4
Early Classification Layer #5
...
...
Input Imaget
Covn. Layer
Conv. Layer
Conv. Layer
Classification Layer
BoF Pooling
BoF Pooling
BoF Pooling
+
+
Shared Classification Layer
Shared Classification Layer
Shared Classification Layer
Shared Classification Layer
Shared Classification Layer
...
...

(a) (b) (c)

Figure 1: Comparing three different ways to use early exits. Fig. 1a demonstrates one typical way to use early exits to estimate the final output of the network at various points of its computational graph. Early exits can be improved by replacing global average pooling with the Bag-of-Features model to compile hierarchical early exits, as shown in Fig. 1b. Early exits can be further improved by sharing the same classification layer among different early exits and using additive histogram representations, as shown in Fig. 1c. This allows for reducing the number of parameters and forming implicit common representations spaces.

function:

$$\mathcal{L}(\mathbf{y}, \mathbf{t}) = -\sum_{i=1}^{N_C} [\mathbf{t}]_i \log([\mathbf{y}]_i), \tag{2}$$

where the notation $[\mathbf{y}]_i$ is used to refer to the i-th element of vector $\mathbf{y}$.

3.2. Efficient Early Exits using Bag-of-Features Aggregation

Employing early exits provides a way to estimate the final output of the network at various points of its computational graph, without having to feed-forward the whole network [15]. That is, an additional estimator:

$$g^{(i)}_{\mathbf{W}_i}\left(\mathbf{y}^{(i)}\right) = g^{(i)}_{\mathbf{W}_i}\left(f_{\mathbf{W}}\left(\mathbf{x}, i\right)\right) \in \mathbb{R}^{N_C} \tag{3}$$

is employed at the i-th layer to estimate the final output of the network without feed-forwarding the network until the last layer. This process is illustrated in Fig. 1a. The notation $\mathbf{W}_i$ is used to refer to the parameters of the i-th early exit. Each early exit usually employs a feature aggregation method, e.g., global average pooling, to extract a compact representation from each feature map, and a classification layer that estimates the final output of the network. Using an efficient feature aggregation approach is of crucial importance, since directly using the raw feature maps would lead to enormous classification layers (due to the large size of the feature maps, especially for the early layers of the network).

Existing methods usually apply global average pooling to extract a compact representation $\mathbf{s}^{(i,avg)}$

out of the intermediate feature maps of the i-th layer of a network:

$$\mathbf{s}^{(i,avg)} = \frac{1}{W_i H_i} \sum_{k=1}^{W_i} \sum_{l=1}^{H_i} [\mathbf{y}^{(i)}]_{kl} \in \mathbb{R}^{C_i}, \tag{4}$$

where the notation $[\mathbf{y}^{(i)}]_{kl}$ is used to refer to the feature vector extracted from the location (k, l) of the feature map of the i-th layer. The extracted averaged representation is then fed to a fully connected layer that is trained to estimate the final classification output of the network, as shown in Fig. 1a.

However, using plain global average pooling possibly discards a great amount of valuable information, as also discussed in Section 1 and experimentally demonstrated in Section 4. Therefore, to overcome this limitation, in this work a trainable BoF-based aggregation scheme is used to compile a compact representation that can be then fed to the used fully connected classification layer. First, a set of prototype vectors (also known as codewords) $\mathbf{v}_{ij} \in \mathbb{R}^{C_i}$ are used to model the distribution of the feature vectors extracted from the i-th layer. The set of these vectors $\mathcal{V}_i = \{\mathbf{v}_{i1}, \mathbf{v}_{i2}, \dots, \mathbf{v}_{iN_K}\}$, where N_K is the number of codewords used, is called dictionary or codebook. Note that a different codebook $\mathcal{V}_i$ must be used for each exit layer, since the distribution of the feature vectors extracted from each layer is different. Then, we can estimate the probability of observing each feature vector $[\mathbf{y}^{(i)}]_{kl}$, extracted from the i-layer of the network, for a given image $\mathbf{x}$ using Kernel Density Estimation [27] as:

$$p([\mathbf{y}^{(i)}]_{kl} | \mathbf{x}) = \sum_{j=1}^{N_K} [\mathbf{s}^{(i)}]_j K([\mathbf{y}^{(i)}]_{kl}, \mathbf{v}_{ij}) \in [0, 1], \tag{5}$$

where $K(\cdot)$ is a kernel function and $\mathbf{s}^{(i)} \in \mathbb{R}^{N_K}$ are the parameters that control the density estimation. Employing a maximum likelihood estimator allows for estimating these parameters as:

$$\mathbf{s}^{(i)} = \arg\max_{\mathbf{s}} \sum_{k=1}^{W_i} \sum_{l=1}^{H_i} \log \left(\sum_{j=1}^{N_K} [\mathbf{s}]_j K([\mathbf{y}^{(i)}]_{kl}, \mathbf{v}_{ij}) \right) \in \mathbb{R}^{N_K}. \tag{6}$$

Indeed, as demonstrated in [27], these image specific parameters can be trivially calculated, giving rise to the well known soft-BoF formations [22, 28]. Therefore, the representation extracted from the i-th layer of the network is calculated as:

$$\mathbf{s}^{(i)} = \frac{1}{W_i H_i} \sum_{k=1}^{W_i} \sum_{l=1}^{H_i} \mathbf{u}_{ikl} \in \mathbb{R}^{N_K}, \tag{7}$$

where $[\mathbf{u}_{ikl}]_j = \frac{K([\mathbf{y}^{(i)}]_{kl}, \mathbf{v}_{ij})}{\sum_{m=1}^{N_K} K([\mathbf{y}^{(i)}]_{kl}, \mathbf{v}_{im})} \in [0, 1]$. The histogram vector $\mathbf{s}^{(i)}$ essentially provides a compact summary that describes the semantic content of an image at various levels of granularity, maintaining more information regarding the actual *distribution* of the vectors $[\mathbf{y}^{(i)}]_{kl}$ than the average representation ($\mathbf{s}^{(i,avg)}$).

To implement the BoF model a normalized RBF layer, followed by a recurrent accumulation layer, is employed, as proposed in [19]. Furthermore, to simplify the implementation we also use a hyperbolic (sigmoid) kernel in order to calculate the similarity between each the codeword and each extracted feature vectors [29]:

$$K([\mathbf{y}^{(i)}]_{kl}, \mathbf{v}) = \frac{1}{2}\left(\tanh(c_1[\mathbf{y}^{(i)}]_{kl}^T\mathbf{v} + c_2) + 1\right) \tag{8}$$

where $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, and c_1 and c_2 are the kernel parameters (typically set to $c_1 = 1$ and $c_2 = 0$). Note that the kernel is appropriately scaled to $0 \ldots 1$, ensuring its compatibility with the quantization process.

The proposed BoF-based aggregation scheme is shown in Fig. 1b, where first we employ BoF-based aggregation to compile a compact representation $\mathbf{s}^{(i)}$ for each image out of each layer, which is subsequently fed to the final fully connected early classification layer (Early classification layers #1, #2 and #3). Furthermore, note that the quantization process can be applied at various spatial levels, as in various spatial pyramid aggregation schemes [30], giving rise to the Spatial BoF [19]. This process allows for retaining more spatial information, which can provide a significant benefit for the exits placed on earlier layers, where the effective receptive field of the convolutional layers is smaller.

Each early exit is trained to estimate the same targets using a representation extracted from the output of the i-th layer of the network $\mathbf{y}^{(i)}$, i.e.,

$$\mathbf{W}_i' = \mathbf{W}_i - \eta \sum_{j=1}^{N} \frac{\partial \mathcal{L}\left(g^{(i)}_{\mathbf{W}_i}(\mathbf{y}_j), \mathbf{t}_j\right)}{\partial \mathbf{W}}. \tag{9}$$

Note that early exits can be either trained after training the base network or by simultaneously training both the base network and the early exits. Even though the latter option can allow the network to adapt to the added early exits (and, as a result, lead to better accuracy for the early exits), it can potentially harm the classification performance for the main network, e.g., if too many early exits are used. To avoid this issue, in this work we first train the base network, we fix its parameters $\mathbf{W}$, and then train all the exit layers simultaneously. It is also worth noting that the network can be directly trained to predict a soft target vector, as produced by the neural network, instead of the binary target $\mathbf{t}_j$, following the neural network distillation principle [31]:

$$\mathbf{W}_i' = \mathbf{W}_i - \eta \sum_{j=1}^{N} \frac{\partial \mathcal{L}\left(g^{(i)}_{\mathbf{W}_i}(\mathbf{y}_j), f_{\mathbf{W}}(\mathbf{x}_j)\right)}{\partial \mathbf{W}}. \tag{10}$$

In this way, the network can be trained to directly estimate the actual output of the network, instead of being trained to predict the same targets as the output of the classification layer of the network.

3.3. Hierarchical Bag-of-Features Aggregation

The BoF-based aggregation method proposed in the previous Subsection is capable of extracting rich and discriminative representations from the feature map of each layer. However, all the information that exists in the representations extracted from the earlier exit layers is discarded at each exit layer. Therefore, instead of merely relying on the representation extracted from the current layer, in this paper we propose compiling an incremental hierarchical representation that exploits the representations extracted from the previous exit layers, effectively overcoming the aforementioned limitation. To this end, the final representation $\mathbf{s}^{(i,h)}$ extracted from the i-th layer is calculated as:

$$\mathbf{s}^{(i,h)} = \begin{cases} \mathbf{s}^{(i)} & \text{if } i = 1 \\ \mathbf{s}^{(i)} \frown \mathbf{s}^{(i-1,h)} & \text{if } i > 1 \end{cases}, \tag{11}$$

where the notation $\mathbf{a} \frown \mathbf{b}$ is used to denote the concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$. This process is depicted in Fig. 1b (Classification Layer #4 and #5). It is worth noting that the representation extracted from the previous early exits can be cached and directly reused for the following early exits with no additional cost. As a result, this approach provides an easy and straightforward way to further refine the predictions of the early exits, increasing the classification accuracy of the network, as also demonstrated in Section 4.

However, this hierarchical approach can lead to increasingly larger exit layers as more exits are used, since the length of the representation $\mathbf{s}^{(i,h)}$ gradually increases by N_K as the number of exit layers increases. To maintain the advantage of using hierarchical representations, without the added complexity of using increasingly larger exit layers we propose a simple, yet efficient, aggregation approach. Instead of simply concatenating the representations together, we propose forming an additive common histogram representation space for all the exit layers. That is, the updated histograms are calculated as:

$$\mathbf{s}^{(i,ch)} = \begin{cases} \mathbf{s}^{(i)} & \text{if } i = 1 \\ \mathbf{s}^{(i)} + \alpha \mathbf{s}^{(i-1,ch)} & \text{if } i > 1 \end{cases}, \tag{12}$$

where α is a decaying factor for the previous histogram (typically set to $\alpha = 1$ when only a few exit layers are used). This approach allows for gradually refining the histogram estimation, while keeping the potentially useful information extracted from the previous layers. Finally, this additive method allows for using a single classification layer that can be reused for all the exit layers, instead of learning separate classification layers for each early exit, which further promotes forming common representation spaces and reduces the number of parameters required for the early exits. This approach indeed leads to a significant reduction of the number of parameters required (since only one classification layer is required regardless of the number of exit layers used), with minimal impact on the classification accuracy, as also experimentally demonstrated in Section 4. The proposed classification layer sharing approach is shown in Fig. 1c. Note that each early exit is still equipped with a separate codebook used for appropriately constructing the corresponding histogram $\mathbf{s}^{(i)}$. Also, the hyper-parameter α can be set to lower values to prevent significant distribution shifts that could negatively affect the shared classification layers. Nonetheless, this behavior was not observed during the conducted experiments. Therefore, $\alpha = 1$ was used for all the conducted experiments.

3.4. Adaptive Inference with Early Exits

There are several ways to feed-forward a network that is equipped with early exits. Perhaps the most straightforward approach is to use a specific early exit according to the available computational resources. Therefore, an earlier early exit is used when there are limited computational resources available, while latter early exits are used when more computational resources are available. For example, we can consider a real-time embedded system designed to perform face recognition from CCTV footage using a deep neural network. When more people appear in a frame then the time required to perform face recognition increases, since each face must be fed to the network. However, using early exits allows for dynamically reducing the number of calculations required for the recognition by appropriately selecting an early exit, allowing for meeting the real-time constraints by slightly reducing the quality of service without over-engineering the system, e.g., using an unnecessary powerful processor to handle every possible number of faces in real time.

The proposed method can be also used to reduce the load, and as a result the energy consumption, of a system by dynamically feed-forwarding through the network according to the difficulty of each training sample and the prediction uncertainty of each exit layer for the specific sample. To this end, we calculate the average class activations at each exit layer i as:

$$\mu_i = \frac{1}{N}\frac{1}{N_C}\sum_{k=1}^{N_C}\sum_{j=1}^{N}[g^{(i)}_{\mathbf{W}_i}(\mathbf{y}_j)]_k, \tag{13}$$

where $g^{(i)}_{\mathbf{W}_i}$ is the classification output of the corresponding early exit. The average activations can be calculated either using the training set or, in order to acquire a more robust estimation, using a validation set. Then, the user specifies a confidence hyper-parameter β that controls when the inference process can stop early at the i-th exit layer, without having to feed-forward through the rest of the network. Therefore, the inference process can stop at the i-th layer when the maximum activation exceeds by β the average activation :

$$[g^{(i)}_{\mathbf{W}_i}(\mathbf{y}_j)]_k > \beta\mu_i, \tag{14}$$

where $k = \arg\max g^{(i)}_{\mathbf{W}_i}(\mathbf{y}_j)$. Note that setting different values for the hyper-parameter β allows for controlling the trade off between the classification accuracy of the network and the number of resources needed for classifying one input image. Larger values for the hyper-parameter β lead to more accurate classification decisions, while smaller values leads to faster (since the inference process can stop early), but less accurate classification decisions. It is worth noting that, as it is demonstrated in Section 4, the inference process can be stopped early for many input samples with a minimal impact on the classification accuracy.

4. Experimental Evaluation

The proposed method is extensively evaluated in this Section. First, the used datasets, evaluation setup, and network architectures are described in Subsection 4.1, while the evaluation results are provided and discussed in Subsection 4.2.

4.1. Datasets and Evaluation Setup

Five datasets were used for evaluating the proposed method: a) the MNIST image classification dataset [32], b) the Fashion MNIST fashion product classification dataset [33], as well as the more challenging c) CIFAR-10 object recognition dataset [34], d) FER-2013 facial expression dataset [35], and e) PlantVillage dataset [36], which contains leaf images from healthy and infected plants.

Three different neural network architectures were also employed for the evaluation. The first one, called "CNN-1", was used for the experiments conducted with the MNIST dataset. CNN-1 is composed of a 3×3 convolution layer with 32 filters, followed by 2×2 max pooling layer, a 3×3 convolution layer with 64 filters, another 2×2 max pooling layer, a fully connected layer with 1024 neurons, dropout with rate $p = 0.5$, and a final fully connected classification layer. For the Fashion MNIST dataset we used a more powerful network that uses twice the number of filters in the first two convolutional layers. Therefore, this network, called "CNN-2", is composed of a 3×3 convolution layer with 64 filters, followed by a 2×2 max pooling layer, a 3×3 convolution layer with 128 filters, another 2×2 max pooling layer, a fully connected layer with 1024 neurons layer, dropout with rate $p = 0.5$, and a final fully connected classification layer. The ReLU activation function was used for all the layers. For the CIFAR-10 and FER-2013 datasets a more powerful network, the MobileNet v.2 [37], was employed, after appropriately tuning the filter size of the first convolutional layer for each dataset. Images from the PlantVillage dataset were resized to 32×32 pixels, while random rotations (up to 30 degrees) and horizontal flips were used to augment the training dataset. We used classes that are related to the three following plants: "Pepper bell" (2 classes), "Potato" (3 classes), and "Tomato" (10 classes). Finally, the CNN-2 architecture was used for all the conducted experiments using this dataset. The categorical cross-entropy loss was used for training all the networks, combined with the Adam optimizer. The CNN-1 and CNN-2 models were trained for 50 epochs with learning rate 0.001, while the MobileNets and the networks used for the PlantVillage dataset were trained for 50 epochs with a learning rate of 0.001, followed by additional 50 training epochs with a learning rate of 0.0001.

Table 1: Comparing the classification error and computational overhead for different variants of the proposed method. The classification error (%), computational complexity (MMAC) and required number of additional parameters is reported. The notation "AH-BoF (F, X)" refers to using the proposed method with X codewords, while "AH-BoF (S, X)" refers to employing additive histogram spaces and re-using the employed classification layer. When the "AH-SBoF" variant is used, then spatial segmentation into four regions is enabled.

	Early Exit - 1		**Early Exit - 2**		**Hierarchical Exit**		
Method	**Error**	**MMAC**	**Error**	**MMAC**	**Error**	**MMAC**	**# Added Parameters.**
MNIST Dataset (CNN-1)							
AH-BoF (F, 16)	15.19%	0.31 (8%)	5.37%	2.48 (61%)	3.79%	2.57 (63%)	2.25k
AH-BoF (S, 16)	16.79%	0.31 (8%)	6.05%	2.48 (61%)	4.38%	2.57 (63%)	1.75k
AH-SBoF (F, 8)	5.00%	0.26 (6%)	3.03%	2.47 (60%)	1.86%	2.51 (61%)	2.10k
AH-SBoF (S, 8)	6.11%	0.26 (6%)	3.92%	2.47 (60%)	2.57%	2.51 (61%)	1.12k
AH-SBoF (F, 32)	**2.85%**	0.40 (10%)	**2.63%**	2.51 (61%)	1.81%	2.69 (66%)	8.30k
AH-SBoF (S, 32)	3.48%	0.40 (10%)	2.67%	2.51 (61%)	**1.57%**	2.69 (66%)	4.43k
Fashion MNIST Dataset (CNN-2)							
AH-BoF (F, 16)	17.05%	0.61 (5%)	15.26%	9.42 (74%)	12.98%	9.60 (76%)	3.78k
AH-BoF (S, 16)	18.70%	0.61 (5%)	16.21%	9.42 (74%)	13.95%	9.60 (76%)	3.28k
AH-SBoF (F, 8)	15.55%	0.52 (4%)	14.51%	9.40 (74%)	11.54%	9.48 (75%)	2.87k
AH-SBoF (S, 8)	17.61%	0.52 (4%)	15.94%	9.40 (74%)	14.18%	9.48 (75%)	1.89k
AH-SBoF (F, 16)	**13.34%**	0.61 (5%)	**12.93%**	9.42 (74%)	**10.54%**	9.60 (76%)	5.70k
AH-SBoF (S, 16)	14.22%	0.61 (5%)	13.52%	9.42 (74%)	11.30%	9.60 (76%)	3.76k
CIFAR-10 Dataset (MobileNet v.2)							
AH-BoF (F, 16)	17.68%	44.41 (47%)	11.05%	64.25 (68%)	10.81%	64.32 (68%)	3.27k
AH-BoF (S, 16)	18.36%	44.41 (47%)	11.57%	64.25 (68%)	11.19%	64.32 (68%)	2.77k
AH-BoF (F, 32)	14.55%	44.48 (47%)	10.05%	64.35 (68%)	9.90%	64.48 (68%)	6.50k
AH-BoF (S, 32)	14.92%	44.48 (47%)	10.61%	64.35 (68%)	9.78%	64.48 (68%)	5.52k
AH-BoF (F, 128)	**11.76%**	44.88 (47%)	**9.14%**	64.95 (69%)	**8.82**%	65.48 (69%)	25.89k
AH-BoF (S, 128)	11.90%	44.88 (47%)	9.47%	64.95 (69%)	8.99%	65.48 (69%)	22.03k
FER-2013 Dataset (MobileNet v.2)							
AH-BoF (F, 16)	50.35%	98.60 (47%)	44.33%	143.24 (68%)	43.61%	143.39 (68%)	3.07k
AH-BoF (S, 16)	52.55%	98.60 (47%)	45.17%	143.24 (68%)	44.39%	143.39 (68%)	2.72k
AH-BoF (F, 32)	48.70%	98.75 (47%)	**43.66%**	143.46 (68%)	42.57%	143.76 (68%)	6.11k
AH-BoF (S, 32)	51.62%	98.75 (47%)	45.17%	143.46 (68%)	43.13%	143.76 (68%)	5.42k
AH-BoF (F, 128)	**48.39%**	99.64 (47%)	43.83%	144.80 (69%)	42.69%	146.00 (69%)	24.35k
AH-BoF (S, 128)	48.51%	99.64 (47%)	44.97%	144.80 (69%)	**42.13%**	146.00 (69%)	21.65k

The percentage of MMAC with the respect to the total MMAC required to feed-forward the network are reported in parenthesis. The best results for each early exit are reported in bold, while the best overall results for each dataset are underlined.

Table 2: Comparing the proposed "AH-(S)BoF" method to two competitive early-exit approaches (ElasticNet and BranchyNet). The same notation as in Table 1 is used for the proposed method.

		Early Exit - 1		**Early/Hierarchical Exit - 2**		**# Added**
Method	**Dataset**	**Error**	**MMAC**	**Error**	**MMAC**	**Parameters.**
ElasticNet	MNIST	49.99%	0.22 (5%)	7.07%	2.45 (60%)	1.95k
BranchyNet	MNIST	18.48%	0.36 (9%)	3.01%	2.48 (60%)	3.65k
AH-SBoF (S, 8)	MNIST	**6.11%**	0.26 (6%)	**2.57%**	2.51 (61%)	1.12k
ElasticNet	FashionMNIST	32.51%	0.43 (3%)	16.91%	9.37 (74%)	3.87k
BranchyNet	FashionMNIST	25.02%	0.71 (6%)	12.51%	9.41 (74%)	7.11k
AH-SBoF (S, 16)	FashionMNIST	**14.22%**	0.61 (5%)	**11.30%**	9.60 (76%)	3.76k
ElasticNet	CIFAR-10 Dataset	27.04%	44.34 (47%)	15.74%	64.15 (68%)	3.23k
BranchyNet	CIFAR-10 Dataset	16.65%	44.51 (47%)	11.82%	64.40 (68%)	11.89k
AH-BoF (S, 32)	CIFAR-10 Dataset	**14.92%**	44.48 (47%)	**9.78%**	64.48 (68%)	5.52k
ElasticNet	FER-2013 Dataset	59.88%	98.45 (47%)	51.40%	143.01 (68%)	2.26k
BranchyNet	FER-2013 Dataset	49.21%	98.68 (47%)	44.41%	143.36 (68%)	5.96k
AH-BoF (F, 32)	FER-2013 Dataset	**48.70%**	98.75 (47%)	**42.57%**	143.76 (68%)	6.11k
ElasticNet	PlantVillage	29.72%	4.04 (23%)	5.54%	14.09 (81%)	4.85k
BranchyNet	PlantVillage	21.48%	4.38 (25%)	6.59%	14.32 (82%)	12.06k
AH-BoF (S, 32)	PlantVillage	**12.37%**	4.25 (24%)	**5.26%**	14.40 (83%)	5.69k

The percentage of MMAC with the respect to the total MMAC required to feed-forward the network are reported in parenthesis.

For all the conducted experiments two early exits were employed, together with one hierarchical exit that combines the information extracted from these two exits. For both the CNN-1 and CNN-2 models, the first early exit layer was placed after the 1st convolutional layer, while the second early exit layer was placed after the 2nd convolutional layer. The early exits were placed at the 5th and 7th convolutional layers, respectively, for the MobileNet model. The proposed method is called Adaptive Hierarchical BoF, abbreviated as "AH-BoF" for the rest of the paper. When a spatial segmentation scheme into 4 regions is used, the proposed method is abbreviated as "AH-**S**BoF". Note that when the Spatial BoF method is used, then the size of the extracted representation is increased by a factor of 4, since four different histograms are extracted (one for each spatial region). Two different variants of the proposed method are evaluated: a) concatenating the hierarchical representations (as described in (11)) and using separate classification layers for each early exit (denoted by "AH-(S)BoF (F, X)", where X refers to the number of used codewords), and b) using additive histogram spaces (as described in (12)) and classification layer reuse (denoted by "AH-(S)BoF (S, X)"). The proposed method is also compared to using global average pooling aggregation, as proposed in [16] (denoted by "ElasticNet"), as well as to using an additional convolutional layer before performing the aggregation proposed for early exits (denoted by "BranchyNet") [13]. Note that all the experiments were conducted using the PyTorch framework [38].

Table 3: Evaluating the proposed adaptive inference classification strategy for three different settings. The classification error and the average complexity for classifying each input sample ("Avg. MMAC") are reported.

	Adaptive Inference (High Speed)		**Adaptive Inference (Balanced)**		**Adaptive Inference (High Precision)**	
Method	**Error**	**Avg. MMAC**	**Error**	**Avg. MMAC**	**Error**	**Avg. MMAC**
MNIST Dataset (CNN-1)						
Static Inference					0.68%	4.10
AH-BoF (F, 16)	4.16%	1.08	1.86%	2.07	0.73%	3.60
AH-BoF (S, 16)	4.45%	1.14	1.79%	2.23	0.72%	3.79
AH-SBoF (F, 8)	1.32%	0.69	0.94%	0.98	0.76%	1.69
AH-SBoF (S, 8)	1.61%	0.78	0.97%	1.19	0.70%	2.29
AH-SBoF (F, 32)	1.09%	0.67	0.74%	0.97	**0.68%**	**1.90**
AH-SBoF (S, 32)	1.06%	0.74	0.78%	1.08	0.68%	2.43
Fashion MNIST Dataset (CNN-2)						
Static Inference					7.82%	12.66
AH-BoF (F, 16)	10.10%	4.04	8.55%	5.86	7.85%	9.55
AH-BoF (S, 16)	10.18%	4.59	8.45%	6.84	7.84%	10.42
AH-SBoF (F, 8)	9.54%	3.62	8.38%	5.14	**7.81%**	**9.11**
AH-SBoF (S, 8)	10.45%	4.30	8.46%	6.34	7.84%	10.17
AH-SBoF (F, 16)	8.95%	3.52	8.17%	4.76	7.82%	9.92
AH-SBoF (S, 16)	9.13%	3.91	8.18%	5.50	7.82%	10.53
CIFAR-10 (MobileNet v.2)						
Static Inference					7.81%	94.61
AH-BoF (F, 16)	10.23%	53.20	8.81%	63.14	**7.76%**	**77.03**
AH-BoF (S, 16)	10.38%	53.32	8.72%	65.42	7.77%	80.75
AH-BoF (F, 32)	9.05%	52.73	8.25%	60.90	7.78%	70.77
AH-BoF (S, 32)	9.13%	52.84	8.37%	62.80	7.84%	74.84
AH-BoF (F, 128)	8.12%	52.09	8.03%	58.26	7.82%	64.33
AH-BoF (S, 128)	8.16%	52.36	7.98%	59.91	7.77%	67.21
FER-2013 (MobileNet v.2)						
Static Inference					38.84%	211.52
AH-BoF (F, 16)	43.58%	131.27	38.92%	184.97	39.03%	209.11
AH-BoF (S, 16)	43.86%	130.37	39.34%	193.07	39.20%	211.01
AH-BoF (F, 32)	41.73%	131.75	**38.59%**	**182.56**	38.62%	207.08
AH-BoF (S, 32)	44.36%	129.07	39.67%	189.62	39.73%	211.31
AH-BoF (F, 128)	41.32%	133.32	39.06%	176.74	39.26%	205.13
AH-BoF (S, 128)	41.74%	135.00	38.62%	182.47	38.70%	208.46
PlantVillage (CNN-2)						
Static Inference					1.24%	17.38
AH-BoF (F, 32)	3.28%	7.06	1.84%	8.96	1.27%	14.90
AH-BoF (S, 32)	3.60%	7.55	1.54%	10.40	1.24%	16.42
AH-BoF (F, 64)	2.52%	6.92	**1.55%**	**8.43**	1.28%	14.14
AH-BoF (S, 64)	2.75%	7.27	1.61%	9.41	1.25%	15.72

4.2. Evaluation Results

First, we provide a sensitivity analysis, where we evaluate the effect of different design choices on the performance of the proposed method. The evaluation results are reported in Table 1. The total multiply-accumulate operations (Million MAC, MMAC) are also reported for each network up to the corresponding exit, while the column "# Added Parameters" refers to the number of added parameters to the network due to the early exits. The number in parenthesis for the MMAC refers to the percentage of MMAC for the current early exit with respect to the total number of MMAC required to feed-forward the whole network. Note that even with spending as little as 5% of the total MMAC (Fashion MNIST combined with "AH-SBoF(F, 16)") leads to adequate results (classification accuracy $< 14\%$). As it will be also demonstrated later, this allows for early stopping the inference process at the earlier exits, allowing for acquiring significant performance benefits. The two different variants of the proposed method, namely the AH-BoF with concatenated histogram spaces ("F" variant) and the AH-BoF with additive histogram spaces and classification layer reuse ("S" variant), are also compared in Table 1. The lightweight "S" variant is capable of significantly reducing the number of parameters in all the cases, with only a minimal impact on the classification accuracy for the first two early exits. On the other hand, the lightweight "S" variant is actually outperforming the other methods for the hierarchical exit that combines the information extracted from the other two early exits for two of the evaluated datasets (MNIST and FER-2013 datasets).

There are also several interesting conclusions that can be drawn regarding the parameters of the BoF model. First, note that by appropriately tuning the number of used codewords we can effectively control the trade-off between the added parameters and the classification accuracy of the network. Also, note that the added early exits have a minimal effect on the computational complexity, since regardless the complexity of the used BoF model, the number of MMAC remains almost the same (especially for the more complex MobileNet network). Furthermore, in almost any case, increasing the number of codewords seems to have a positive effect on the classification accuracy. At the same time, using spatial segmentation ("SBoF") seems to be especially important when the early exits are used on convolutional layers with smaller receptive fields, e.g., CNN-1 and CNN-2.

The proposed method is also compared to the ElasticNet and BrancyNet approaches in Table 3. We selected the most competitive variant of the proposed method that is closer to the number of parameters used by the rest of the evaluated methods. Furthermore, we also tuned the number of filters in the BranchyNet model in order to avoid significantly exceeding the number of parameters used by the rest of the methods. Note that this is not always straightforward, since 3×3 convolutions are used in BranchyNet instead of 1×1, as the proposed method does, leading to 9-fold increase in the required number of parameters. The proposed method always leads to improved accuracy over the two evaluated methods, while requiring about the same number of parameters (or significantly less compared to some BranchyNet variants).Note that this is also true for many other variants of the proposed method, as reported in Table 1.

Next, we evaluated the effect of using the proposed adaptive inference strategy for early stopping the inference process when the network is confident enough. The evaluation results are reported in Table 3. Three different inference strategies were used, namely "High Speed" ($\beta = 1$), "Balanced" ($\beta = \frac{1+c\frac{1}{\mu_0}}{1+c}$) and "High Accuracy" ($\beta = \frac{c}{\mu_0}$), where μ_0 (calculated as in (13)) is used to estimate the uncertainty of the neural network at the first early exit and c is a hyper-parameter that controls the uncertainty limit for stopping at an early exit ($c = 0.99$ were used for all the conducted experiments). The "Static Inference" baseline refers to using the final classification layer of the network, without adding any early exit. Several interesting conclusions can be drawn from the results reported in Table 3. First, adding early exits and using the proposed method allows for significantly reducing the average inference time without harming the classification accuracy. For example, for the MNIST dataset the proposed method reduced the average MMAC by 4 times, while achieving the same classification performance. For the Fashion MNIST the error is slightly reduced to 7.81%, while the number of MMAC are reduced from 12.66 to 9.11. For the CIFAR-10 and FER-2013 datasets, using the proposed approach actually achieves lower classification error than the original network, hinting that earlier layers may contain useful information discarded by many neural network architectures, as also suggested by recent neural network architectures [39]. Furthermore, note that for the FER-2013 dataset the best results are obtained using an inference strategy that tends to stop at an earlier layer. The proposed adaptive inference method was capable of reducing the MMAC while achieving competitive classification performance for almost every evaluated case, demonstrating the practical usefulness of the proposed method for adaptively altering the computational graph of a DL model according to the difficulty of the input samples.

5. Conclusion

Deep learning models equipped with early exits are capable of providing adaptive computational graphs that allow for directly adapting a model to the currently available computational resources. However, existing methods for implementing early exits mainly employ naive aggregation methods, such as global average pooling, significantly restricting their performance. At the sample time, they usually ignore all the information that is extracted by the earlier exits, despite the fact that this information is often already extracted and available at no additional cost. In this paper, we proposed a Bag-of-Features (BoF)-based method that is capable of overcoming these limitations. To this end, the proposed method constructs efficient hierarchical early exit layers with minimal computational overhead, since it employs additive shared histogram spaces, that gradually refine the information extracted from the various layers of a network, in a hierarchical manner. At the same time, it employs a classification layer reuse strategy that allows for further reducing the number of parameters needed per exit layer. The proposed method can be further combined with an adaptive inference strategy that allows for early stopping the inference process when the network is confident enough for its output, leading to further performance benefits. It is worth noting that the proposed method is generic and can be readily combined with any neural network architecture, leading to a practical tool that can be effectively used in various real-world embedded applications, as demonstrated through the conducted experiments on five different datasets. At the same time, the significant improvements obtained using the proposed approach pave the way for developing more sophisticated and advanced methods for designing and implementing early exits. Among them, adaptive methods for selecting the most appropriate early exit for each input sample can be developed, instead of relying on a fixed and pre-defined threshold, allowing for potentially further increasing the efficiency and accuracy of the proposed method.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449 (OpenDR). This publication reflects the authors' views only. The European Commission is not responsible for any use that may be made of the information it contains.

Declarations of interest: None

References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436.

[2] J. Mehta, A. Majumdar, Rodeo: robust de-aliasing autoencoder for real-time medical image reconstruction, Pattern Recognition 63 (2017) 499–510.

[3] B. Gecer, S. Aksoy, E. Mercan, L. G. Shapiro, D. L. Weaver, J. G. Elmore, Detection and classification of cancer in whole slide breast histopathology images using deep convolutional networks, Pattern Recognition 84 (2018) 345–356.

[4] P. Li, Z. Chen, L. T. Yang, Q. Zhang, M. J. Deen, Deep convolutional computation model for feature learning on big data in internet of things, IEEE Transactions on Industrial Informatics 14 (2) (2018) 790–798.

[5] M. Patacchiola, A. Cangelosi, Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods, Pattern Recognition 71 (2017) 132–143.

[6] Z. Jiao, X. Gao, Y. Wang, J. Li, H. Xu, Deep convolutional neural networks for mental load classification based on eeg data, Pattern Recognition 76 (2018) 582–595.

[7] S. Han, H. Mao, W. J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, in: Proceedings of the International Conference on Learning Representations, 2016.

[8] J.-H. Luo, J. Wu, W. Lin, Thinet: A filter level pruning method for deep neural network compression, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5058–5066.

[9] N. Passalis, A. Tefas, Learning deep representations with probabilistic knowledge transfer, in: Proceedings of the European Conference on Computer Vision, 2018, pp. 268–284.

[10] W. Hao, Z. Zhang, Spatiotemporal distilled dense-connectivity network for video action recognition, Pattern Recognition 92 (2019) 13–24.

[11] T.-B. Xu, P. Yang, X.-Y. Zhang, C.-L. Liu, Lightweightnet: Toward fast and lightweight convolutional neural networks via architecture distillation, Pattern Recognition 88 (2019) 272–284.

[12] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, K. Q. Weinberger, Multi-scale dense networks for resource efficient image classification, in: Proceedings of the International Conference on Learning Representations, 2018.

[13] S. Teerapittayanon, B. McDanel, H.-T. Kung, Branchynet: Fast inference via early exiting from deep neural networks, in: Proceedings of the International Conference on Pattern Recognition, 2016, pp. 2464–2469.

[14] A. Veit, S. Belongie, Convolutional networks with adaptive inference graphs, in: Proceedings of the European Conference on Computer Vision, 2018, pp. 3–18.

[15] Y. Bai, S. S. Bhattacharyya, A. P. Happonen, H. Huttunen, Elastic neural networks: A scalable framework for embedded computer vision, in: Proceedings of the European Signal Processing Conference, 2018, pp. 1472–1476.

[16] Y. Zhou, Y. Bai, S. S. Bhattacharyya, H. Huttunen, Elastic neural networks for classification, in: Proceedings of the IEEE International Conference on Artificial Intelligence Circuits and Systems, 2019, pp. 251–255.

[17] J. Sivic, A. Zisserman, Video google: A text retrieval approach to object matching in videos, in: Proceedings of the IEEE International Conference on Computer Vision, 2003, pp. 1470–1477.

[18] D. G. Lowe, Object recognition from local scale-invariant features, in: Proceedings of the IEEE International Conference on Computer Vision, Vol. 2, 1999, pp. 1150–1157.

[19] N. Passalis, A. Tefas, Learning bag-of-features pooling for deep convolutional neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5766–5774.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Computer Vision and Pattern Recongition, 2015, pp. 1–9.

[21] A. Iosifidis, A. Tefas, I. Pitas, Discriminant bag of words based representation for human action recognition, Pattern Recognition Letters 49 (2014) 185–192.

[22] N. Passalis, A. Tefas, Neural bag-of-features learning, Pattern Recognition 64 (2017) 277 – 294.

[23] N. Passalis, A. Tefas, Learning bag-of-embedded-words representations for textual information retrieval, Pattern Recognition 81 (2018) 254–267.

[24] N. Passalis, J. Raitoharju, A. Tefas, M. Gabbouj, Adaptive inference using hierarchical convolutional bag-of-features for low-power embedded platforms, in: Proceedings of the IEEE International Conference on Image Processing, 2019, pp. 3048–3052.

[25] S. Pancoast, M. Akbacak, Bag-of-audio-words approach for multimedia event classification, in: Proceedings of the Conference of the International Speech Communication Association, 2012.

[26] N. Passalis, A. Tsantekidis, A. Tefas, J. Kanniainen, M. Gabbouj, A. Iosifidis, Time-series classification using neural bag-of-features, in: Proceedings of the European Signal Processing Conference, 2017, pp. 301–305.

[27] S. Bhattacharya, R. Sukthankar, R. Jin, M. Shah, A probabilistic representation for efficient large scale visual recognition tasks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2011, pp. 2593–2600.

[28] J. C. Van Gemert, J.-M. Geusebroek, C. J. Veenman, A. W. Smeulders, Kernel codebooks for scene categorization, in: Proceedings of the European Conference on Computer Vision, 2008, pp. 696–709.

[29] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, A. Iosifidis, Temporal logistic neural bag-of-features for financial time series forecasting leveraging limit order book data, arXiv preprint 1901.08280 (2019).

[30] K. He, X. Zhang, S. Ren, J. Sun, Spatial pyramid pooling in deep convolutional networks for visual recognition, in: Proceedings of the of the European Conference on Computer Vision, 2014, pp. 346–361.

[31] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, in: Proceedings of the NIPS Deep Learning and Representation Learning Workshop, 2015.

[32] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[33] H. Xiao, K. Rasul, R. Vollgraf, Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint 1708.07747 (2017).

[34] A. Krizhevsky, Learning multiple layers of features from tiny images, Tech. rep., University of Toronto (2009).

[35] O. Arriaga, M. Valdenegro-Toro, P. Plöger, Real-time convolutional neural networks for emotion and gender classification, arXiv preprint 1710.07557 (2017).

[36] D. Hughes, M. Salathé, et al., An open access repository of images on plant health to enable the development of mobile disease diagnostics, arXiv preprint 1511.08060 (2015).

[37] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, in: Proceedings of the Advances in Neural Information Processing Systems, 2019, pp. 8024–8035.

[39] G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708.