Class-specific early exit design methodology for convolutional neural networks

Vanderlei Bonato[a,*], Christos-Savvas Bouganis[b]

[a] *Institute of Mathematical and Computing Sciences, The University of São Paulo, São Carlos, São Paulo - Brazil*
[b] *Department of Electrical and Electronic Engineering, Imperial College London, London - England*

Abstract

Convolutional Neural Network-based (CNN) inference is a demanding computational task where a long sequence of operations is applied to an input as dictated by the network topology. Optimisations by data quantisation, data reuse, network pruning, and dedicated hardware architectures have a strong impact on reducing both energy consumption and hardware resource requirements, and on improving inference latency. Implementing new applications from established models available from both academic and industrial worlds is common nowadays. Further optimisations by preserving model architecture have been proposed via early exiting approaches, where additional exit points are included in order to evaluate classifications of samples that produce feature maps with sufficient evidence to be classified before reaching the final model exit. This paper proposes a methodology for designing early-exit networks from a given baseline model aiming to improve the average latency for a targeted subset class constrained by the original accuracy for all classes. Results demonstrate average time saving in the order of 2.09× to 8.79× for dataset CIFAR10 and 15.00× to 20.71× for CIFAR100 for baseline models ResNet-21, ResNet-110, Inceptionv3-159, and DenseNet-121.

Keywords: Early exit, accuracy preservation, target class, and CNN.

*Contact author
Email address: vbonato@icmc.usp.br (Vanderlei Bonato)

1. Introduction

Traditional neural networks have evolved significantly in the last decade resulting in deep architectures that can solve complex problems, especially those related to image [1], text [2], and speech processing [3]. Convolutional Neural Network (CNN) is a state-of-the-art solution that is able to extract features for image classification directly from the pixels and it became the core of many machine learning systems, as for object detection and 3D scene layout understanding in real time [4, 5].

By working at pixel level performing convolution operations over a sequence of tree-dimensional feature maps (width, height, and channel-depth), a model may demand high computing capacities for data processing and storage. To mitigate this scaling problem while maintaining the model performance, a significant effort has been made on compression, mainly on connection pruning, weight quantisation, parameter sharing, and on network connectivity encoding [6, 7, 8]. New model architectures conceived to be more compact by adopting irregular weight connectivity have also been exploited, such as the approach from GoogLeNet that includes the *inception module* [9], the one from ResNet [10] that uses residual functions to define *shortcut connections* scheme between layers, and the one from DenseNet where a traditional model is split into dense connected blocks enabling each layer inside a block to take all preceding feature-maps from its block as input [11]. From embedded to data center systems all software and hardware deployments are benefited from these optimisations, reducing resource requirements, power consumption, and even improving prediction accuracy.

Another approach to mitigate the model's complexity is by adding early exit points to the neural network's topology to give the opportunity for "easy" inputs to be classified early on before reaching the final model's exit point. The solutions are usually implemented as a conditional neural network [12, 13, 14, 15], where confidence conditions are in place to decide when an early classification can be accepted. An in-depth discussion on the design, implementation, and

training of the early exit approached is presented in [16]. As [16] demonstrates, the introduction of early exit mechanisms reduces the average inference latency and minimises problems related to overfitting during the training of the network.

Solutions that have no early exits in place employ mechanisms to avoid going through certain layers based on the sample complexity, similar to the ResNet's shortcut schema. This is seen in [17, 18, 19], where layers can be fully or partially skipped dynamically. In [20, 21, 22, 23], layer payloads are balanced by determining which output channels/filters from the feature maps are used by the next layer. These solutions give the opportunity for a given sample to dynamically exploit the backbone model power and avoid the need of adding extra exits to the model. However, the algorithm complexity and the code size increase considerably since all the paths need to be available.

The use of cascades of increasing complexity networks has also been exploited for achieving early prediction of "easy" to be classified inputs. Given an input, a light-weight network is employed firstly in an attempt to classify the input. In case the confidence of classification is low, the input is passed to a more complex network until the desired confidence level is achieved or the last network in the cascade is reached [24, 25, 26]. In contrast to the conditional architecture, the cascade approach does not reuse the computations already performed, with the potential to impact negatively the overall processing time and energy consumption relative to a system that is based on the conditional architecture methodology.

The above approaches demonstrate that it is possible to improve classification accuracy while reducing the computation time when compared to regular models.

The aim of this paper is to reduce the average computational time to classify samples of a target class by extending a given CNN model that is used as a baseline. The proposed approach adds an additional exit to the given model, named early exit, to be used as a conditional early classification by taking advantage of the processing results (feature maps) already in place from the intermediate network layers. The resulted solution maintains the original CNN

accuracy for all classes.

The main contributions of this paper are:

- A method that identifies the best place to attach an early exit that maximises the time savings while preserving the original baseline model accuracy;
- The exploration of a dense connection between early exit branch and baseline model layers for simultaneous multi-layer feature extraction of a given target class;
- A class-orientated approach to allow inference prioritisation;
- A scalable approach to mitigate the effort to apply the proposed method over neural networks with a large number of layers.

The new target class exploration proposal has demonstrated to be more effective in reducing the average inference time when compared to solutions that have no class priority. This is a result of a more specialised early exit placement and a more precise confidence calibration. The exploration methodology requires training for the early exit branch only and is independent of the model architecture.

The paper is organised as follows: Section 2 presents works related to reducing the computational complexity for inference. Section 3 describes the proposed method that includes the mathematical formulation for guaranteeing the original baseline accuracy. Then, Section 4 starts showing the overhead effects caused by the early exit followed by a deep analysis of the model parameters over prediction and time saving. Then, the performance of the system for larger models is demonstrated considering CIFAR10 and CIFAR-100 datasets. A comparison to the related works is also included. Finally, Section 6 concludes the paper.

2. Related Work

Several works have been proposed to accelerate inference in convolutional neural networks, ranging from high level model optimisation to low level hardware tuning, including solutions based on early exiting, graph pruning, data quantisation, and hardware accelerators.

More related to this work, which is a data-driven solution that takes advantage of the sample complexity to mitigate computation, [12] proposes a Conditional Deep Learning Network (CDLN). Given a baseline trained model, a linear classifier block is added at the first convolutional layer and trained with the same dataset used for the baseline. A next classifier block is added to the following layer only if the classifications expected to be anticipated will compensate the additional computational cost inflicted. This process stops as soon as this criterion is not satisfied. During inference, a sample is considered classified whenever the activation of a block is above a user defined threshold. The authors claim that the conditional proposal may improve both energy consumption and prediction accuracy for the whole NN.

In [15] a Conditional Deep Neural Network (CDNN) with early exits to be mapped in IoT systems supported by Fog platforms is presented. Three supervised training modes named End-to-End, Layer-Wise, and Classifier-Wise are considered, where the Layer-Wise shown to be the preferred choice. A criterion to choose where to attach early exits is also presented. It was demonstrated that the conditional approach is able to provide better classification performance and lower computational time with respect to the baseline models and that a significant reduction in energy consumption is achievable when the appropriate training choice and early exit placement strategies are used.

Differently from CDLN approach, [13] proposes to use a multi-layer subnetwork as early exit branch instead of a simple linear classifier. The branch locations and architecture are decided empirically. A parametric loss function is used for each exit, including the baseline. The whole network is trained by back-propagating the losses via a joint optimisation problem formulation. A

sample is considered classified if the entropy of the branch activation is above a given threshold.

In [27] a Multi-Scale DenseNet (MSDNet) network architecture is presented, which is a bi-dimensional model that allow operations in horizontal (deph), vertical and diagonal directions. The objective is to be able to extract coarse features since from the initial convolutional layers, what is not the case from the previous approaches. Two model settings are exploited, one to do prediction in an environment where the processing time available is unbounded and the other where a time budget is known in advance to classify a batch of samples. The first produces the most recent prediction when the time is over and the second produces a prediction as soon as a classifier achieves a pre-determined confidence measure (softmax probability) that was set according to the budget. Training is applied to the whole network, including the classifier modules.

A preliminary work on early exiting is also presented by [14] focusing on the application of the cross-entropy loss function to combine multiple exist losses during training and to give confidence level during inference. No exploration regarding to early exit location was done.

SkipNet [17] anticipates classifications by skipping layers (shortcuts) dynamically during inference using small gating networks. The whole network is trained in two phases; first relaxing some parameters for the gating network then a dedicated algorithm is applied to refine the gating network separately. Overall, the training time increased about 30-40%. The network accuracy depends on the gating binary decisions to skip or not certain layers. A skipping policy to balance the computational cost and classification accuracy is used, which can meet the state of the art baseline model accuracy by reducing computational time saving. An extension of this work is presented in [19], where a layer can be partially skipped as well, outperforming the previous SkipNet in time saving and prediction accuracy.

In [20] a network that can prune filters/channels dynamically during inference according to the input sample is introduced. The pruning problem is seen as Markov decision process and reinforcement learning is applied to learn prun-

ing rules. Results demonstrate that the proposed network was able to keep the prediction accuracy much higher than a traditional model with equivalent computational complexity.

In Slimmable neural networks [22] the model complexity is dynamically adapted by exploiting the width of the feature map. A switchable batch normalization is introduced to deal with statistics among different model variants (switch). It was shown that by sharing weights among model variants the prediction accuracy can be improved when compared to a model that was trained only with the channels that it needs. For instance, for ImageNet an improvement of 3.3% was achieved for MobileNet v1 featuring only 12% of its parameters. [21] extends this work by generalising the training to arbitrary feature map width.

In GaterNet [23] the network complexity is reduced by filter selection similar to the approach used by the Slimmable network, however here the samples dictate which filters to use during inference dynamically. A conventional backbone network is used along with a parallel network named gater network to activate the backbone filters according to the samples. Back-propagation is used for training starting from pre-trained models on the same task, since it is considered a difficult task to learn these gates from scratch. The model could reduce the prediction error up to 1.83% against ResNet-164 on CIFAR100 dataset by increasing the total number of parameters in about 20%. No processing time gain is reported.

HydraNet [18] is a dynamic network architecture that specialises components during training to be chosen by a gating mechanism during inference. Branches are created from group of classes that are similar visually, which are chosen according to the input sample by gate functions. A common part named stem, the branches and the gate functions are trained jointly. Results for CIFAR100 shows that HydraNets achieve a significant computational cost reduction by keeping the same baseline model accuracy level, obtaining for Hydra-Res-d4 a reduction by two thirds of the MADD costs when compared to the ResNet-164.

IDK prediction [24] is a framework that composes a set of pre-trained models to operate in cascade to accelerate inference. An extra class to measure the

uncertainty of the base model predictions is included, which is used as a trigger to activate the next cascade model. The trigger threshold is obtained from softmax entropy information. It was demonstrated that a reduction of 24% in the computation cost can be achieved for a composition that maintains the base model's accuracy. It was also shown that by allowing a small accuracy degradation it is possible to have a much higher cost reduction. The authors suggest that this solution is suitable for the edge-cloud computational systems, where the light-weight models are deployed at the edge.

Quantisation and pruning techniques have also been investigated under a data driven setting for speeding up the inference through an early exit structure. They are particularly useful to deploy networks on edge devices sensitive to energy consumption and hardware cost. In [25] two networks operating in cascade are quantised for lower and higher precision, respectively. A confidence evaluation unit decides against a user-specified error tolerance when an inference needs to reach the higher precision unit or not. The unit is tuned a priori considering the probability vector of whole class set. No training is necessary. Tailored to dedicated hardware, a considerable speed-up could be achieved for a scenario where only a high accuracy network would exist. While in [8], pruning is exploited by considering that convolutional models have a considerable amount of repeated data (weight repetition) that are non zero and overlap across filters.

Early exit, that can be used standalone or integrated to other strategies, is a model driven approach whose efficiency relies on finding a place(s) to attach an exit that maximises time saving, on the exit branch architecture, and on the adopted strategy to guarantee a minimal accuracy level. As seen from the literature review, except from the MSDNet work [27] that populates every layer from the second layer with a classifier, the remaining works do not explore the connection possibilities available throughout all network layers. Our proposed work presents a solution that searches for the best connection point for a given target class taking into account all network layers in the space exploration and also the original baseline accuracy. The best place is the one that maximises

time saving while the original accuracy is preserved.

3. Proposed Methodology

3.1. Problem formulation

Given a trained neural network model B as baseline, an early exit branch e along with a connection type to connect e to B, and a target class $k \in V$, where V denotes the set of classes classified by B, the aim is to find the place (layer) to attach e to B that minimises the average execution time to classify samples belonging to k and the average overhead time inflicted to the rest of the samples while maintaining the original B accuracy. The connection type is a boolean variable where (On) indicates that a dense connection layer where e is attached to B is introduced that takes as inputs the outputs of the current and the all previous layers, where (Off) indicates that only the connection where e attaches to B is kept. Thus, this variable indicates whether or not to Keep Previous connections and it is defined as (KP).

Considering the processing time t of B, the addition of an early exit adds overhead e_i to t resulting in t'_i (1), where i indicates the connection point layer. The larger the i is, the higher is the overhead inflicted by samples not classified at the early exit point. t can be divided in $t-\delta_i$ for the layers beyond i (potential layers to skip) and δ_i for the layers up to i (layers already computed), as seen in (2). The resulted time $t - \delta_i$ is weighted according to a classification rate γ_i (4) as given by (3).

The γ_i denotes the positive classification rate at e exit (Out:EE_i) for k. The objective is to maximise γ_i subject to maintain the original classification accuracy given by β for the classes $V-k$ and η for k. The objective of maintaining the original baseline accuracy is captured by the following equations: (5) for classifications of $V - k$ at B exit (Out:B); (6) for classifications of k at (Out:EE_i); and (7) for global classifications of k at both (Out:B) and (Out:EE_i). Please note that (Out:EE_i) is a binary classification while in (Out:B) it is a multi-class classification where only true positive tp and false negative fn are taken

into account, allowing to register classification results only in relation to actual classes. For instance, given a sample and its actual class, the result indicates if it was classified correctly (tp) or incorrectly as another class (fn), thus the accuracy of a target class k is equal to $tp/(tp + fn)$. The maximisation of the global accuracy is seen as a multiclass problem, as samples that are not tp_k or fn_k will be registered as some V class. Table 1 presents a description for all variables used in this formulation.

$$t + e_i = t'_i \tag{1}$$

$$(t - \delta_i) + \delta_i + e_i = t'_i \tag{2}$$

$$(1 - \gamma_i)(t - \delta_i) + \delta_i + e_i = t''_i \tag{3}$$

Maximise:

$$\gamma_i = \begin{cases} (tp_{_{out:EE_i}} + fp_{_{out:EE_i}})/|k|, & \text{if (5) \& (6) \& (7).} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

$$acc_{out:B_{(V-k)}} = (tp_{out:B} - tp_{k_{out:B}})/(|V| - (tp_{out:EE_i} + fp_{out:EE_i}) - (tp_{k_{out:B}} + fn_{k_{out:B}})) >= \beta \tag{5}$$

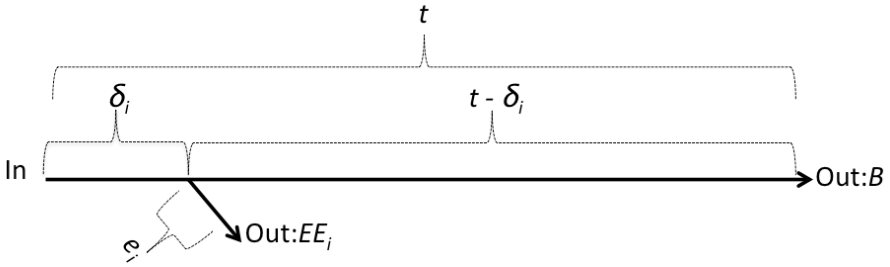
$$acc_{out:EE_i} = (tx_{_{out:EE_i}})/(tx_{_{out:EE_i}} + fx_{_{out:EE_i}}) >= \eta \tag{6}$$

$$acc_{out:EE_{iGlobal}} = (tp_{_{out:EE_i}} + tp_{k_{out:B}})/(tp_{_{out:EE_i}} + tp_{k_{out:B}} + fn_{k_{out:B}}) >= \eta \tag{7}$$

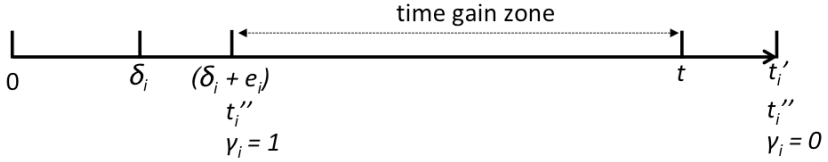
Figure 1 illustrates where these variables are located considering an e_i added to B and shows a timeline identifying t'' variation zone ($\gamma_i = 1$ and $\gamma_i = 0$ are the best and worst scenarios, respectively). The approach leads to latency gain only if $\delta_i + e_i < t$, which requires the overhead e_i to be compensated by successful early classifications.

Table 1: Variable descriptions from Equation (1) to (7).

Variable	Description
i	Layer number
t	Execution time of the baseline model
t_i'	Maximum execution time of an EE_i model
t_i''	Execution time dependent on classification rate
δ_i	Execution time from the first layer up to layer i
e_i	Execution time of the early exit branch
k	Target class
V	Whole classes of B
γ_i	Classification rate
β	Original accuracy of B to classify $V - k$
η	Original accuracy of B to classify k
tp, tn, fp, fn	true positive, true negative, false positive, and false negative
tx, fx	true and false (each x can be positive or negative)



(a) EE_i model variables.



(b) EE_i timeline: best case $\gamma_i = 1$ and worst case $\gamma_i = 0$.

Figure 1: Representation of the variables considering one early exit e_i added to a baseline model B and its respective timeline.

3.1.1. Extending to multiple early exits

By including more than one early exit creates opportunities for samples not classified at a given layer i to be classified later by another layer j before

reaching the final output of B (Out:B). The extra computational time added is formulated by extending $t - \delta_i$ to Equation (8), where e_j denotes the execution time of the additional exit. Equations (9) and (10) show the sequence to add γ_j in order to compute t''_j from the window time remained from EE_i.

$$t - \delta_i + e_j = t'_j \tag{8}$$

$$((t - \delta_i) - \delta_j) + \delta_j + e_j = t'_j \tag{9}$$

$$(1 - \gamma_j)((t - \delta_i) - \delta_j) + \delta_j + e_j = t''_j \tag{10}$$

Equation (10) does not consider the rate of samples that reach the layer j, which depends on the classification rate γ_i. This is modelled by joining equations (10) and (3) resulting in (11). A graphical representation of this time function is shown in 2. As the samples that reach (Out:EE_j) are only the ones considered no positive at (Out:EE_i), the γ_j function (12) subtracts from $|k|$ the samples already classified. The restrictions and maximisation of γ_j are the same as previously presented, except that $j > i$.

$$(1 - \gamma_i)((1 - \gamma_j)((t - \delta_i) - \delta_j) + \delta_j + e_j) + \delta_i + e_i = \hat{t}'' \tag{11}$$

Maximise:

$$\gamma_j = (tp_{j_{out:EE_j}} + fp_{j_{out:EE_j}})/(|k| - tp_{i_{out:EE_i}} - fp_{i_{out:EE_i}}) \tag{12}$$

As seen, to extend from one exit to two exits, the expression $(t - \delta_i)$ from equation 3 resulted in equation 10. Following the same pattern, equation 11 could be extended to accommodate a third exit, that in this case, the expression to expand would be $((t - \delta_i) - \delta_j)$. The above process can be repeated to accommodate as many exits as necessary. However, as defined by the formulation, the later an early exit is added, the smaller is the time gain window available, which can be compensated if more samples are classified. Therefore, a trade-off between these two elements exist, which depends on how the features of B model are distributed throughout the layers and how the class k samples map such features during inference.

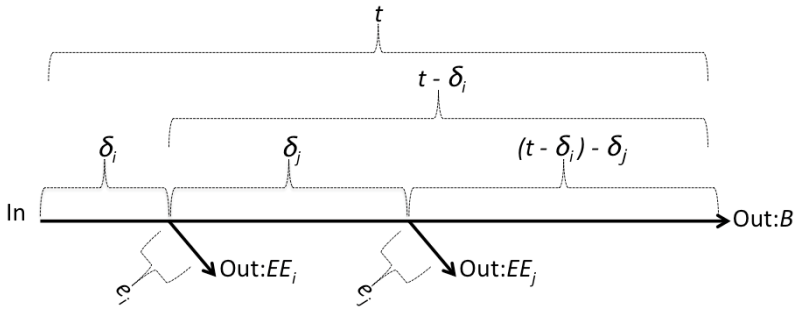


Figure 2: Representation of the variables to join EE_i and EE_j added to B.

3.2. Generating and training EE models

For each convolution layer i of B two EE models can be generated by connecting e to B, one for connection type (KP=On) and another for (KP=Off). Then, each $2n$ EE models, where n is the quantity of B layers, is trained to classify k at the branch e exit (Out:EE_i) and V at the B exit (Out:B). The training phase only updates e branch parameters, keeping B as provided by the user.

Training is considered completed when the accuracy of (Out:EE_i) over the validation dataset stops to improve (Out:B accuracy is constant as B parameters are fixed) during 10 iterations, which has demonstrated to be sufficient for the complexity of the branch e. The overfitting problem is mitigated by the fact that the training can not change the B parameters and e per se is a simple classifier that is unlikely to overfit for the adopted datasets. A more sophisticated training strategy could be applied if B were not fixed, as the one presented in [28] where a cascade learning is applied, which considers that there is always an output adjacent to the layer being trained. This would require multiple early exits and an increased time budget for training the whole EE per target class.

Figure 3 illustrates a baseline model with three convolution layers connected to an early exit branch at Layer 2 with (KP=On) or (KP=Off). When (KP=On) a dense connection between B and e is created from the current connection layer i to the first layer. The early exit branch architecture is also presented, which

features a ReLU, followed by a Pooling and a Dense layers with Softmax activation. The Flatten and Concatenate components are used when necessary to obtain data from the previous layers when (KP=On). The ReLU usually found in the B models could be used instead of this one from the branch, however B models may have a BatchNormalization layer between the convolution operation and ReLU, restricting the direct access to the feature maps. The classification through a single Dense layer is the same adopted by the traditional models Inception [9], ResNet [10], and DenseNet [11]. Thus, an EE model captures the layer where an early exit branch e is attached to B, the connection type between e and B, and the model parameters to classify samples at (Out:EE) and (Out:B).

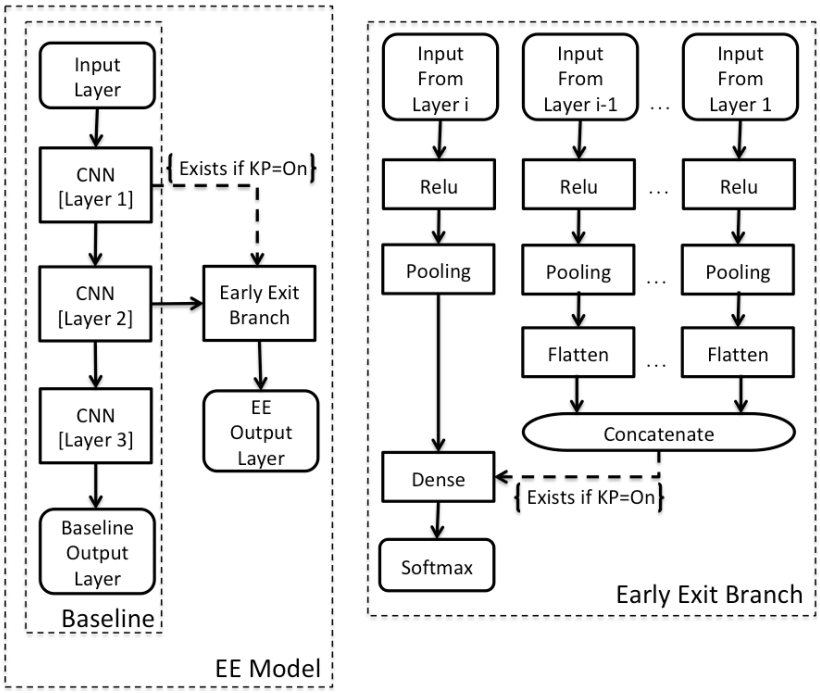


Figure 3: An EE model example along with an early exit branch architecture. If (KP=On) all previous feature maps from the convolutional layers i (in this case $i = 2$) are used as input to the branch. Flatten and concatenate layers are interfaces to deal with the map dimension variations.

The rule of having one EE model per i layer can be restricted to reduce the problem dimension. For large models we propose to sample the network architecture p times to increase its granularity. Each network sample is seen as a layer to attach e and to propagate the connections (KP=On) or (KP=Off), even though internally there exist ($l \bmod p$) layers, where l is the number of layers of model B. The result section includes both approaches.

3.3. Tuning confidence level threshold

A confidence level threshold is tuned for each EE model by analysing its inference performance (Out:EE_i). To accept a sample, the output activation value needs to be equal or greater to a threshold α_i, which captures the confidence level of the classification. Thus, by varying α_i, the classification rate γ_i and consequently the accuracy levels are affected.

In this work, for each EE_i model, the maximum γ_i is found in a space $0.95 <= \alpha_i <= 1.0$. Besides accuracy restrictions, an EE_i is considered a viable solution if also the average execution time gain $t - t_i''$ is positive.

3.4. Selecting EE viable models

After defining the parameters to maximise time saving subject to confidence level and accuracy restrictions imposed by (4), all EE models with positive results are considered as viable solutions. Each one has its potential for time saving (gain) to k versus time inflicted (loss) to $V - k$. In this paper a solution that maximise gain/loss is the one adopted from the Pareto frontier.

4. Performance Evaluation

The proposed framework was evaluated in the Google Colabatory Platform (Colab) with Python v3.6.9 and Tensorflow v2.1.0 as backend for Keras v2.2.5 running on a Tesla P100-PCIE-16GB GPU. Results are based on the CIFAR10 and CIFAR100 datasets [29], which were partitioned into 40k images for training, 10k for validation and 10k for testing, where each sample is a 32x32 colour image. The parameters of B can be set from learning transfer or be trained

only once, taking into account the whole V, while EE needs for each target class k to train its e branch section. The branch has the architecture shown in figure 3 with the polling layer parameters adjusted to fit the input feature map dimensions. Results are generated by adopting as baseline models ResNet-21, ResNet-110, InceptionV3-159, and DenseNet-121. The first one is used to produce a detailed analysis while the others are used to demonstrate scalability.

4.1. Connection type and platform overhead analyses

In order to assess the performance of the approach, an analysis is performed that decouples the optimisations performed by the utilised platform (Tensorflow) to deal with multi-output models from the computational overhead added by the proposed approach. The analysis starts by presenting how an EE model featuring a single early exit branch performs if only the time inflicted to the model as a consequence of the arithmetic operations of e are considered. This time is an estimation obtained from the actual execution of B in Tensorflow. By measuring the latency of executing B, the execution time of e is estimated assuming the same execution time across operations. Thus, adding the latency of B to the latency of e, an estimate for the execution time of EE is finally obtained.

For this analysis we considered a 21 layers ResNet as baseline model B and its classes 1 (automobile), 5 (dogs), and 7 (horses) as target classes that range from low to high accuracy levels. The graphs in figure 4 show the estimated average execution time to classify input samples belonging to target class k along with the time to classify input samples belonging to V set considering (KP=On) and (KP=Off). The parameters β, η, and the upper and lower bounds for α are shown above each graph. The line $BaselineB$ corresponds to the actual execution time t and it provides the reference to verify how each EE_i performs (time t''). From figures 4(a), 4(c), and 4(e) triangle and stars are the time for classes V, including k samples not classified at (Out:EE_i). Dot lines are the time for k classified at (Out:EE_i). Results are demonstrated for each layer i. Figures 4(b), 4(d), and 4(f) demonstrate how each EE_i performs in

relation to (Out:EE_i) and (Out:B). The lower the time in both directions the better. As the cost inflicted by e is quite small compared to the whole B cost, there are several configurations that reduce the average time for both k and V. Additionally, it is seen that for this example (KP=On) achieves the best results for (Out:EE_i) and (Out:B).

4.1.1. Wall clock execution time

The previous experiments are repeated and the actual wall clock EE_i execution times are measure. The graphs in figure 5 have changed for the V classes, since the time overhead by running the multi-output model in the evaluation platform is directly inflicted to them. Even though samples classified at the early exit as tp or fp help to reduce the average execution time of the classes $V-k$, in this case it was not sufficient to compensate all the extra time added. However, for the target class k the reduction still significant for the early layers. As can be noticed, when approaching the latter layers t'' increases considerable above t, as the closest to the (Out:B) the smaller becomes the time zone gain (as seen in figure 1) and the larger is e. When (KP=On) the EE_is tend to provide more opportunities for time saving, even though the cost inflicted to B is higher reflecting the branch network complexity.

Table 2 shows results for the selected models that maximise the proportion of time saving $t-t''$ over time inflicted $t'-t$, which were obtained by prediction exploration via α variation. The column *gain* is the time reduction for k and column *loss* is the time inflicted to $V-k$, both given by how many times ($\times$) it is in relation to the baseline time t. The columns $acc_{out:}$ show the accuracy for predictions over the evaluation dataset, corresponding to $acc_{out:B(|V|-k)}$, $acc_{out:EE_i}$, and $acc_{out:EE_{i_{Global}}}$ computed from equations (5), (6), and (7), respectively. As seen, they satisfy the restrictions imposed by β and η. When (KP=On) the i tends to be smaller as the dense connection of e_i to B can detect multiple kernel features simultaneously, not depending only on the current layer. The smaller the i the greater is the time gain zone, which helps to compensate the extra complexity imposed by dense connection.

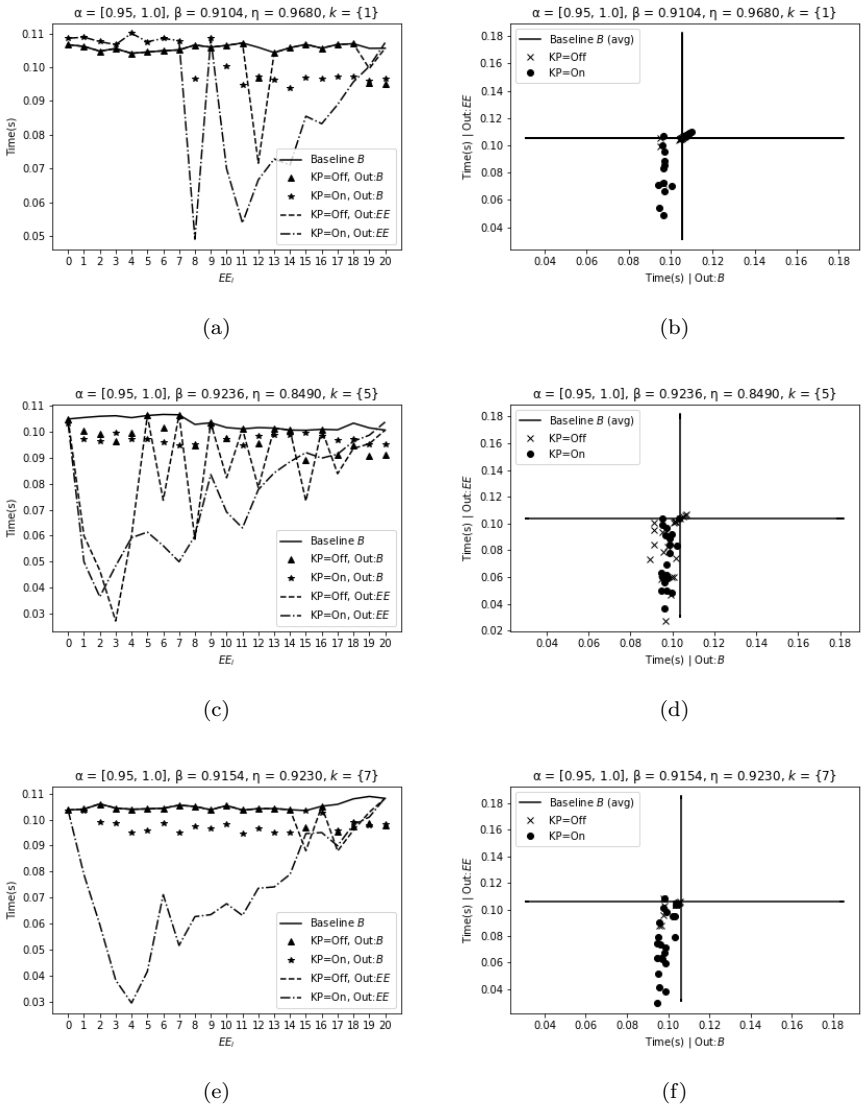


Figure 4: Synthetic average execution time to classify at (Out:EE_i) and (Out:B). Baseline B is the real execution time of B used as reference for performance evaluation. 4(a), 4(c) and 4(e) show the results throughout the layers i and 4(b), 4(d) and 4(f) compare the configurations.

4.1.2. Extending the analysis for all classes

Based on the previous evidence that when (Kp=On) a greater set of EEs has $t'' > 0$, the analysis is extended keeping (Kp=On) for all classes of V. Table

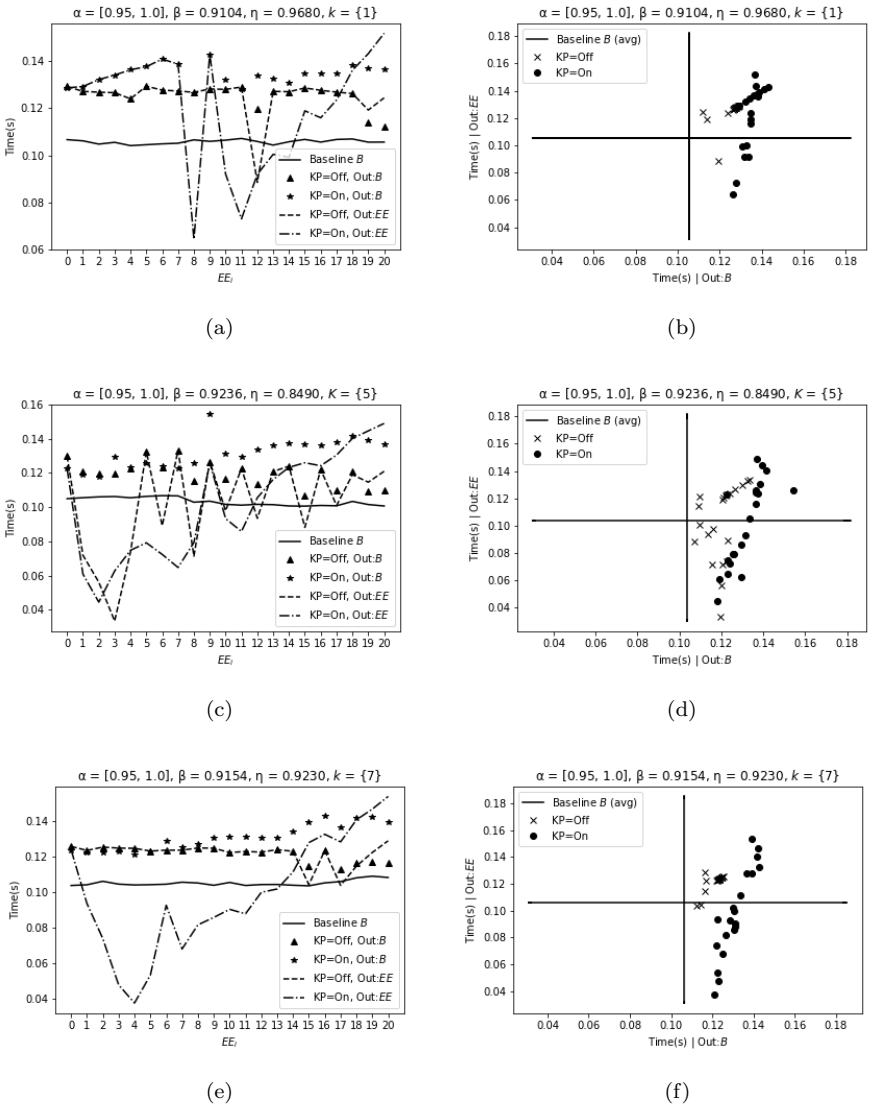


Figure 5: Average execution time to classify at (Out:EE_i) and the overhead inflicted to (Out:B). $BaselineB$ is the original execution time of B and is the reference to evaluate time gain and time loss. 5(a), 5(c) and 5(e) show the results throughout the layers i and 5(b), 5(d) and 5(f) shows the balance between gain and loss of each configuration.

3 presents results for the evaluation dataset, where the best EE_i for each class is the one that maximises $t-t''$. Then new results from the test dataset (never seen

Table 2: Selected models for target classes {1}, {2}, and {3}, obtained for model categories (KP=On) and (KP=Off).

KP	k	i	α	β	η	gain (×)	loss (×)	$\text{acc}_{out:}$ B_{V-k}	EE_i	EE_{i_G}
Off	{1}	12	0.97	0.910	0.968	1.20	1.24	0.912	0.970	0.978
On	{1}	8	0.98	0.910	0.968	1.66	1.31	0.912	0.971	0.981
Off	{5}	3	0.95	0.924	0.849	3.17	1.24	0.928	0.865	0.884
On	{5}	2	0.95	0.924	0.849	2.34	1.22	0.927	0.920	0.873
Off	{7}	17	0.95	0.915	0.923	1.02	1.18	0.917	0.946	0.941
On	{7}	4	0.95	0.915	0.923	2.78	1.27	0.917	0.946	0.943

by the NN), identified by (’), are included, which are produced from EE_is that accept samples at their early exits based only on α thresholds. Additionally, ζ and *prec* are introduced to indicate the original k and the achieved (Out:EE_i) precision, respectively. *prec* informs the rate of tp per fp for (Out:EE_i). Note that fp represents samples belonging to $V-k$ classes that will not anymore be evaluated at (Out:B). As hard samples tend to be classified as fp at the early exit, it is expect to have $prec < \zeta$. By increasing α such difference tends to reduce, however the greater α the smaller tp as well. So the key point is have a system where fp are only samples that would be miss classified by B anyway. The table shows that accuracy $acc_{out:x}$ has improved for both datasets as a consequence of fp at (Out:EE_i). Regarding to time saving, it was possible to obtain at least one EE per class where the *gain* is bigger than the *loss*, demonstrating that the proposed approach is able to reduce the average processing time for all target k while keeping at least the original B accuracy.

Table 4 provides the figures for fn at (Out:B) for the original B and for the EE_is taking into account all target classes. Comparing to B we can see for most cases a reduction of fn for all classes, being more intense for the target one. The samples reduced for $V-k$ classes are due to they were classified as fp at (Out:EE_i). It is important to notice that not every fp sample would be an fn sample at (Out:B), it depends on how hard a sample is for both exits. The important is that restrictions imposed by β and η to $acc_{out:B_{(V-k)}}$ and $acc_{out:EE_{iGlobal}}$ guarantee a solution where the proportion of fn per sample

Table 3: Results for (KP=On) for all V classes. EE_{i_G} represents $EE_{i_{Global}}$ and (') are the results for the test dataset, differentiating them from the evaluation dataset.

k	i	α	β	η	ζ	gain (×)	loss (×)	acc$_{out:}$ B_{V-k}	EE_i	EE_{i_G}	prec	gain' (×)	loss' (×)	acc$_{out:}$ B_{V-k}'	EE_i'	EE_{i_G}'	prec'
{0}	3	0.95	0.914	0.937	0.915	2.66	1.23	0.915	0.949	0.950	0.676	2.76	1.23	0.919	0.948	0.948	0.667
{1}	8	0.98	0.910	0.968	0.947	1.66	1.31	0.912	0.971	0.981	0.799	1.67	1.31	0.915	0.971	0.988	0.802
{2}	5	0.95	0.918	0.896	0.879	1.66	1.24	0.921	0.919	0.913	0.626	1.66	1.24	0.922	0.916	0.876	0.598
{3}	1	0.95	0.927	0.816	0.850	1.96	1.21	0.929	0.840	0.841	0.415	2.18	1.21	0.928	0.838	0.885	0.415
{4}	4	0.97	0.914	0.934	0.897	1.49	1.27	0.919	0.937	0.945	0.657	1.50	1.27	0.921	0.936	0.929	0.654
{5}	2	0.95	0.924	0.849	0.877	2.34	1.22	0.927	0.920	0.873	0.568	2.26	1.22	0.931	0.926	0.870	0.588
{6}	6	0.99	0.912	0.950	0.935	1.85	1.29	0.914	0.958	0.961	0.756	1.81	1.29	0.916	0.961	0.966	0.770
{7}	4	0.95	0.915	0.923	0.958	2.78	1.27	0.917	0.946	0.943	0.693	3.04	1.27	0.920	0.938	0.956	0.664
{8}	5	0.95	0.912	0.953	0.948	2.64	1.28	0.913	0.961	0.973	0.737	2.60	1.28	0.914	0.965	0.972	0.758
{9}	5	0.99	0.914	0.936	0.952	1.82	1.27	0.914	0.953	0.951	0.768	1.78	1.27	0.915	0.961	0.960	0.804

classified is reduced from B to EE_i.

Table 4: Number of samples classified as fn at (Out:B) for the original B and for the EE_is.

Class k	i	α	{0}	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}	{9}	total	%
Model B	-	-	63	32	104	184	66	151	50	77	47	64	838	0.00
{0}	3	0.95	50	32	92	183	65	149	49	74	34	61	789	-5.84
{1}	8	0.98	61	19	104	183	65	149	50	77	43	42	793	-5.36
{2}	5	0.95	59	32	87	174	57	138	45	72	47	63	774	-7.63
{3}	1	0.95	63	31	100	159	61	132	45	73	46	64	774	-7.63
{4}	4	0.97	61	32	95	171	55	134	45	66	47	64	770	-8.11
{5}	2	0.95	62	32	94	157	62	127	49	72	46	63	764	-8.83
{6}	6	0.99	61	31	95	172	62	148	39	77	47	64	796	-5.01
{7}	4	0.95	62	32	98	175	60	139	50	57	46	63	782	-6.68
{8}	5	0.95	53	31	96	182	65	151	50	76	27	55	786	-6.20
{9}	5	0.99	62	24	104	183	66	151	49	74	43	49	805	-3.93

4.1.3. Multiple early exits impact

The results shown so far impose the constrains that only one EE at a time could be added to the baseline model B. The experiments were repeated for k target {1}, {3}, and {7} considering now multiple EEs where n exits may exist to provide extra time saving. For the three cases when (Kp=On) all gains were worst while when (Kp=Off) for {5} the *gain* improved from 3.17× to 3.24×,

however the *loss* went up from 1.24× to 1.44×. This is because the current overhead of the adopted platform is enough high to restrict the advantages of spreading classification throughout additional early exits. Another observation is that for configuration (Kp=On) the accumulative approach is less effective. Indirectly, they have similar objectives as the dense connections consider features from the current and previous layers as well, but with the advantage of having only one exit.

4.2. Scaling to deeper models

The previous section has demonstrated a complete analysis of accuracy, precision, time gain and loss for ResNet-21 model over the CIFAR10 dataset, where for each layer i an EE_i was generated and trained. As i represents a layer number, for each target class k there are 21 models, which multiplied by the 10 classes of the dataset results in 210 EE models that needed to be trained. Considering the possible configurations of (KP) variable, this figure doubles to 420 models. This section shows results for scaling to the deeper models ResNet-110, Inceptionv3-159, and DenseNet-121 using the same CIFAR10 and the CIFAR100 superclass labels. Thus, to avoid training a larger number of models, these deeper models were sampled 20 times as described in Section 3.2, where each sample has an early exit branch e with connection type (Kp=On).

Figure 6 shows the log speed-up for the top-3 compounded results for each target class or superclass k, selected according to their proportion of gain in relation to loss (gain/loss). This proportion depends on α_i, which was exploited from 1.0 to 0.95. Restrictions to accuracy were applied to maintain EE_i as the original baseline model. For CIFAR10 the arithmetic mean gain is 5.09× for Inceptionv3-159, 5.77× for ResNet-110, and 8.79× for DenseNet-121, while for the previous model ResNet-21 it is 2.09×. For CIFAR100 the obtained gain is 15.00× for Inceptionv3-159, 17.26× for ResNet-110, and 20.71× for DenseNet-121. The results show that the time savings for these models increase, which is expected since the deeper the models the more opportunity exists during training for easier samples to be exit earlier.

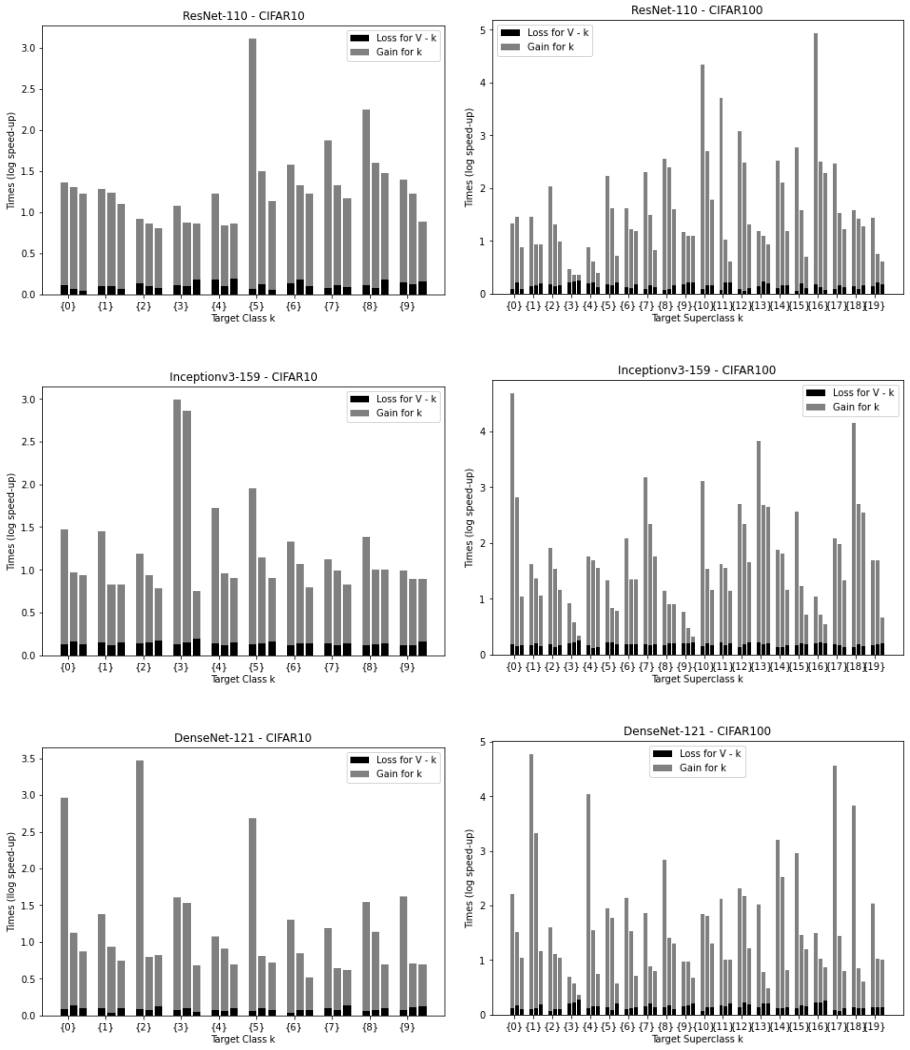


Figure 6: Average execution time gain for target class k and time loss for $V - k$ when applied to baseline models ResNet (110 layers), Inceptionv3 (159 layers) and DenseNet (121 layers). Gain and loss are shown in times (log speed-up) in relation to the reference model execution time. For each k the top-3 compounded results are in order. Results are from the datasets CIFAR10 and CIFAR100 (superclass).

Comparing the same target class between these two models for CIFAR10, there exist variation in time gain as a consequence of how features maps are

spread throughout the layers during B training. As B remains constant during e training, time gain becomes model architecture dependent. The same behaviour happens for CIFAR100, however when comparing the average time gain between CIFAR10 to CIFAR100 datasets, the latter is higher.

When working with the target class approach, the bigger the V the lesser the expected time gain, however for CIFAR100 it has not been the case. For the same B model, the superclass approach was demonstrated to be a more difficult classification problem than CIFAR10, where top-1 accuracy for the respective ResNet-110, DenseNet-121, and Inceptionv3-159 dropped from 0.922, 0.860, and 0.875 for CIFAR10 to 0.806, 0.664, and 0.618 for CIFAR100. Such classification performance for CIFAR100 results in a higher number of misclassified samples and these hard samples tend to be misclassified at the early exit as well. It is important to notice that the overall classifier accuracy may remain the same while the early exit precision degrades.

4.3. Comparing to existing work

[12], [13] and [17] are considered to be the closest approaches to the one presented in this paper, as they also apply dedicated early exit branches to a baseline network. [12] is excluded from the comparison, as it uses a particular network configuration over a different dataset, making it not possible to derive any informative comparison results. In the case of [17], which uses shortcuts to skip convolutional blocks, the authors evaluate their approach on a standard model over the same dataset as it is used for evaluation in the work, but the results are reported in terms of computational costs.

Table 5 shows the performance gains in both speed-up (gains in latency) or computational cost when the early exit methodology is applied compared to the original reference model for a number of network topologies (reference models) when the CIFAR10 dataset is targeted. The reported avg. gains for table rows identified as (target class) correspond to the average of the results obtained by each EE_i model that was generated for each target class independently. The table rows that are identified as (all classes) report speed-up/cost results where

all classes were considered by the EE model at the same time. In the case of [13], the performance gain results reported for LeNet-5 and AlexNet-8 correspond to design points that are able to maintain the baseline accuracy, which was not the case for the ResNet-110. In contrast, the results reported for [17] and for the proposed approach in the case of ResNet-110 correspond to design points that maintain the baseline accuracy.

Focusing on the obtained speed-ups by the various approaches, the results demonstrate that for large models the proposed approach, where a specific target class is targeted, leads to larger gains compared to the case where all classes are considered together. In the case of smaller reference models, the opposite is observed. For ResNet-110 we provide the results for both approaches in order to allow a comparison considering the same reference model, dataset, and accuracy. The results show that a speed-up improvement from 1.23× to 5.77× is achieved. The results also indicate that by utilising a dense connection (i.e. KP=On) and the proposed space exploration to find an exit tailored to a specific class, provide larger gains as the size of the model increases.

Table 5: Related works aiming to reduce either execution latency (speed-up) or computational cost (cost) for performing an inference by reducing the quantity of layers to be computed. The reported results are the average gains obtained in relation to a reference model for the same dataset.

	Reference model			Ref. model	Early exit
Work (approach)	Name	Param.	Dataset	accuracy	avg. gain
[13] (all classes)	LeNet-5	60.0k	CIFAR10	maintained	4.7× (speed-up)
[13] (all classes)	AlexNet-8	60.0M	CIFAR10	maintained	2.4× (speed-up)
[24] (all classes)	ResNet-18	-	CIFAR10	maintained	5.1% (cost)
[13] (all classes)	ResNet-110	1.7M	CIFAR10	not maintained	1.9× (speed-up)
[17] (all classes)	ResNet-110	1.7M	CIFAR10	maintained	50% (cost)
Our (all classes)	ResNet-110	1.7M	CIFAR10	maintained	1.23× (speed-up)
Our (target class)	ResNet-110	1.7M	CIFAR10	maintained	5.77× (speed-up)
Our (target class)	Inceptionv3-159	23.9M	CIFAR10	maintained	5.09× (speed-up)
Our (target class)	DenseNet-121	8.1M	CIFAR10	maintained	8.79× (speed-up)
Our (target class)	ResNet-21	0.3M	CIFAR10	maintained	2.09× (speed-up)

Table 6 presents results for a number of datasets and models. Please note that the perceived gains are strongly influenced by the dataset complexity and by the model's architecture, and thus both factors need to be taken into account when results from the considered approaches are compared. [17] and [27] report results for CIFAR100 dataset, achieving in their metric a reduction of computational cost of 37% and 10×, respectively, over the ResNet-110 model. In [15] a time saving of 1.25× and 1.59× are achieved for the datasets FER-2013 and SVHN, respectively, over the AlexNet model (the number of layers was not informed). All results are for solutions that have maintained the accuracy of the reference model. Even though a direct time saving comparison to our results is not possible since they are not from the same CIFAR100 superclass set, it can be inferred that our proposed methodology's average time saving gains of 15.00×, 17.26× and 20.71× are significant, as the achieved figures are even better than the ones from CIFAR10 dataset.

Table 6: Related works for different datasets and models. (sc) is an abbreviation of the word (superclass) and (Comp.) of the word (Computational).

Work (approach)	Model	Dataset	Ref. model accuracy	Early exit avg. gain
[17] (all classes)	ResNet-110	CIFAR100	maintained	37% (Comp. Cost)
[27] (all classes)	ResNet-110	CIFAR100	maintained	10× (Comp. Cost)
[18] (all classes)	ResNet-164	CIFAR100	maintained	3× (Comp. Cost)
[15] (all classes)	AlexNet	FER-2013	maintained	1.25× (Time)
[15] (all classes)	AlexNet	SVHN	maintained	1.59× (Time)
Our (target class)	Inceptionv3-159	CIFAR100 (sc)	maintained	15.00× (Time)
Our (target class)	ResNet-110	CIFAR100 (sc)	maintained	17.26× (Time)
Our (target class)	DenseNet-121	CIFAR100 (sc)	maintained	20.71× (Time)

5. Applicability, limitations and extension opportunities

The performance of the data-driven approach exploited in this paper was evaluated with a set of modern CNNs models mainly targeting the image classification problem. The convolutional part of CNNs has also been widely used

as a component in hybrid models for feature extraction or used as a fully convolutional network (FCN) to produce classification maps instead of feature maps. R-CNN [30], SPP-net [31] and Faster R-CNN [32] are examples of the hybrid models for object detection, while [33] and [34] are examples of FCNs for semantic segmentation at pixel level.

The essence of our proposed early exit method is to improve the average latency of classifications for a targeted subset class by providing a classification outcome as soon as the computed feature maps contain enough evidence to do so. The proposed approach can also be applied for region or pixel classifications, where a target class would be a particular object or a particular image segment. In situations where only feature extraction is performed, a classification via early exit could be included to predict whether the feature maps extracted so far are sufficient or not to provide enough information for the processing conducted by the next stages. A viability study of these assumptions need to take into account not only the potential time saving, but also the extra classification time imposed to the non-targeted classes and the associated training complexity.

Our approach of adopting accuracy as a metric to maintain the original performance has demonstrated to be viable. However, most of the challenging samples from all classes V that would be miss-classified at the baseline exit (Out:B) tend to migrate to the early exit (Out:EE_i), affecting the precision as already discussed in Section 4.1.2. This is possible because in Equation (6) the considered true and false classifications are from the whole set V, which is appropriate from the accuracy point of view, since it is constrained by the accuracy η given by the target class. Such behaviour may not be desirable for applications that consider a higher cost for miss-classification of the $V-k$ classes in the (Out:EE_i) than in (Out:B).

Another limitation of our approach is with regards to the overheads inflicted to the $V-k$ classes that are classified at (Out:B). In a balanced dataset, the more classes exist in the baseline the more overhead (increased average time) will be inflicted to the non-targeted classes. As every input sample needs to go through the early exit branch, the probability per sample of being a true positive

is reduced as well. In this work we haven't exploited such impact beyond the 20 classes.

One possibility to reduce such overhead is by reducing the complexity of the dense connection (when KP=On) by analysing the significance of each connection (weight) to the prediction in order to prune it when possible. For EE models that implement more than one target class, the overhead could be reduced by sharing part of the dense connections, being dictated by the training phase which layer/connection could be shared.

Apart from the expansions previously suggested, to explore precision along with accuracy would probably have the most positive impact on the the current method. To do so, the constrains from Equation (4) need to incorporate the precision metric, which should also be considered in the training phase. A deep analysis of the output activation (α_i) influence in both precision and accuracy is another relevant exploration point.

6. Conclusion

The proposed methodology introduces early exit opportunities on a reference model targeting a specific class leading to improved average classification rate for this specific class, maintaining at the same time the original model accuracy. Even though samples that belong to the remaining classes suffer by adding extra latency, as every sample received by the network needs to pass through the early exit branch to be verified. The dense connection approach (KP=On) demonstrated to be efficient when the baseline model is maintained, as the introduced classification stage can access feature maps from all preceding layers. The evaluation of the proposed methodology demonstrated an average time savings in the order of 2.09 to 8.79 for the CIFAR10 dataset and 15.00to 20.71for CIFAR100 for baseline models ResNet-21, ResNet-110, Inceptionv3-159, and DenseNet-121.

Acknowledgement

The authors would like to acknowledge the São Paulo Research Foundation (FAPESP) for the project financial support, grant number 2019/05286-6.

References

[1] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, T. Darrell, Long-term recurrent convolutional networks for visual recognition and description, IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (4) (2017) 677–691. doi:10.1109/TPAMI.2016.2599174.

[2] F. Gargiulo, S. Silvestri, M. Ciampi, G. D. Pietro], Deep neural network for hierarchical extreme multi-label text classification, Applied Soft Computing 79 (2019) 125 – 138. doi:https://doi.org/10.1016/j.asoc.2019.03.041.

[3] N. Yalta, S. Watanabe, T. Hori, K. Nakadai, T. Ogata, CNN-based multichannel end-to-end speech recognition for everyday home environments, in: 27th European Signal Processing Conference (EUSIPCO), 2019, pp. 1–5.

[4] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779–788. doi:10.1109/CVPR.2016.91.

[5] S. Yang, D. Maturana, S. Scherer, Real-time 3D scene layout from a single image using convolutional neural networks, in: 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 2183–2189. doi:10.1109/ICRA.2016.7487368.

[6] Y. Cheng, D. Wang, P. Zhou, T. Zhang, Model compression and acceleration for deep neural networks: The principles, progress, and challenges, IEEE Signal Processing Magazine 35 (1) (2018) 126–136. doi:10.1109/MSP.2017.2765695.

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

[8] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, C. W. Fletcher, UCNN: Exploiting computational reuse in deep neural networks via weight repetition, in: Proceedings of the 45th Annual International Symposium on Computer Architecture, ISCA '18, IEEE Press, Piscataway, NJ, USA, 2018, pp. 674–687. doi:10.1109/ISCA.2018.00062.
URL https://doi.org/10.1109/ISCA.2018.00062

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Vol. 00, 2015, pp. 1–9. doi:10.1109/CVPR.2015.7298594.

[10] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: B. Leibe, J. Matas, N. Sebe, M. Welling (Eds.), Computer Vision – ECCV 2016, Springer International Publishing, Cham, 2016, pp. 630–645.

[11] G. Huang, Z. Liu, L. v. d. Maaten, K. Q. Weinberger, Densely connected convolutional networks, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2261–2269. doi:10.1109/CVPR.2017.243.

[12] P. Panda, A. Sengupta, K. Roy, Conditional deep learning for energy-efficient and enhanced pattern recognition, in: 2016 Design, Automation Test in Europe Conference Exhibition (DATE), 2016, pp. 475–480.

[13] S. Teerapittayanon, B. McDanel, H. T. Kung, Branchynet: Fast inference via early exiting from deep neural networks, in: 2016 23rd International Conference on Pattern Recognition (ICPR), 2016, pp. 2464–2469. doi:10.1109/ICPR.2016.7900006.

[14] H. Barad, H. Tang, Fast inference with early exit (a case study), https://www.intel.com/content/www/us/en/artificial-intelligence/posts/fast-inference-with-early-exit.html, last access: 06/2020 (2020).

[15] E. Baccarelli, S. Scardapane, M. Scarpiniti, A. Momenzadeh, A. Uncini, Optimized training and scalable implementation of conditional deep neural networks with early exits for fog-supported iot applications, Information Sciences 521 (2020) 107 – 143. doi:https://doi.org/10.1016/j.ins.2020.02.041.
URL http://www.sciencedirect.com/science/article/pii/S0020025520301249

[16] S. Scardapane, M. Scarpiniti, E. Baccarelli, A. Uncini, Why should we add early exits to neural networks?, Cognitive Computation 12 (5) (2020) 954–966. doi:10.1007/s12559-020-09734-4.

[17] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, J. E. Gonzalez, Skipnet: Learning dynamic routing in convolutional networks, in: The European Conference on Computer Vision (ECCV), 2018.

[18] R. T. Mullapudi, W. R. Mark, N. Shazeer, K. Fatahalian, Hydranets: Specialized dynamic architectures for efficient inference, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[19] J. Shen, Y. Wang, P. Xu, Y. Fu, Z. Wang, Y. Lin, Fractional skipping: Towards finer-grained dynamic CNN inference, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 5700–5708.
URL https://aaai.org/ojs/index.php/AAAI/article/view/6025

[20] J. Lin, Y. Rao, J. Lu, J. Zhou, Runtime neural pruning, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 2178–2188.

[21] J. Yu, T. Huang, Universally slimmable networks and improved training techniques, in: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 1803–1811. doi:10.1109/ICCV.2019.00189.

[22] J. Yu, L. Yang, N. Xu, J. Yang, T. Huang, Slimmable neural networks, in: International Conference on Learning Representations, 2019.
URL https://openreview.net/forum?id=H1gMCsAqY7

[23] Z. Chen, Y. Li, S. Bengio, S. Si, You look twice: Gaternet for dynamic filter selection in CNNs, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 9164–9172. doi:10.1109/CVPR.2019.00939.

[24] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, J. Gonzalez, Idk cascades: Fast deep learning by learning not to overthink, in: UAI, 2018.

[25] A. Kouris, S. I. Venieris, C. Bouganis, Cascade CNN: Pushing the performance limits of quantisation in convolutional neural networks, in: 2018 28th International Conference on Field Programmable Logic and Applications (FPL), 2018, pp. 155–1557. doi:10.1109/FPL.2018.00034.

[26] A. Kouris, S. Venieris, C.-S. Bouganis, A throughput-latency co-optimised cascade of convolutional neural network classifiers, IEEE, 2019.
URL http://hdl.handle.net/10044/1/75445

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

[28] E. S. Marquez, J. S. Hare, M. Niranjan, Deep cascade learning, IEEE Transactions on Neural Networks and Learning Systems 29 (11) (2018) 5475–5485. doi:10.1109/TNNLS.2018.2805098.

[29] A. Krizhevsky, V. Nair, G. Hinton, CIFAR-10 and CIFAR-100 (Canadian Institute for Advanced Research), http://www.cs.toronto.edu/ kriz/cifar.html, last access: 02/2020 (2020).

[30] R. Girshick, J. Donahue, T. Darrell, J. Malik, Region-based convolutional networks for accurate object detection and segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 38 (1) (2016) 142–158. doi:10.1109/TPAMI.2015.2437384.

[31] K. He, X. Zhang, S. Ren, J. Sun, Spatial pyramid pooling in deep convolutional networks for visual recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 37 (9) (2015) 1904–1916. doi:10.1109/TPAMI.2015.2389824.

[32] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (6) (2017) 1137–1149. doi:10.1109/TPAMI.2016.2577031.

[33] E. Shelhamer, J. Long, T. Darrell, Fully convolutional networks for semantic segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (4) (2017) 640–651. doi:10.1109/TPAMI.2016.2572683.

[34] Z. Chen, V. Badrinarayanan, G. Drozdov, A. Rabinovich, Estimating depth from rgb and sparse sensing, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018.