Impatient DNNs – Deep Neural Networks with Dynamic Time Budgets

Manuel Amthor
manuel.amthor@uni-jena.de
Erik Rodner
erik.rodner@uni-jena.de
Joachim Denzler
joachim.denzler@uni-jena.de

Computer Vision Group
Friedrich Schiller University Jena
Germany
www.inf-cv.uni-jena.de

Abstract

We propose Impatient Deep Neural Networks (DNNs) which deal with dynamic time budgets during application. They allow for individual budgets given a priori for each test example and for anytime prediction, *i.e.* a possible interruption at multiple stages during inference while still providing output estimates. Our approach can therefore tackle the computational costs and energy demands of DNNs in an adaptive manner, a property essential for real-time applications.

Our Impatient DNNs are based on a new general framework of learning dynamic budget predictors using risk minimization, which can be applied to current DNN architectures by adding early prediction and additional loss layers. A key aspect of our method is that all of the intermediate predictors are learned jointly. In experiments, we evaluate our approach for different budget distributions, architectures, and datasets. Our results show a significant gain in expected accuracy compared to common baselines.

1 Introduction

Deep and especially convolutional neural networks are the current base for the majority of state-of-the-art approaches in vision. Their ability to learn very effective representations of visual data has led to several breakthroughs in important applications, such as scene understanding for autonomous driving [1], object detection [6], and robotics [4]. The main obstacle for their application is still the computational cost during prediction for a new test image. Many previous works have focused on speeding up DNN inference in general achieving constant speed-ups for a certain loss in prediction accuracy [10, 16].

In contrast, we focus on inference with dynamic time budgets. Our networks provide a series of predictions with increasing computational cost and accuracy. This allows for (1) dynamic interruption of the prediction in time-critical applications (anytime ability, Figure 1 left), or for (2) predictions with a dynamic time budget individually given for each test image a-priori (Figure 1 right). Dynamic budget approaches can for example deal with varying energy resources, a property especially useful for real-time visual inference in robotics [17]. Furthermore, early predictions allow for immediate action selection in reinforcement learning scenarios [21].

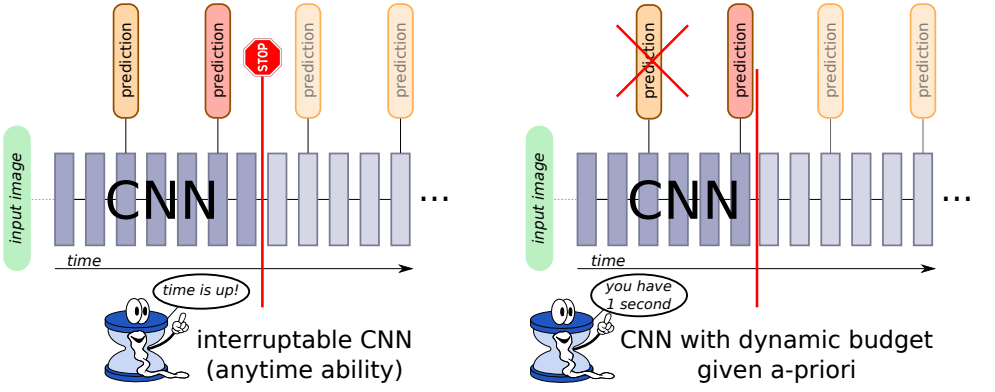

Figure 1: Illustration of convolutional neural network prediction in dynamic budget scenarios: (left) prediction can be interrupted at **any time** or (right) the budget is given **before** each prediction.

The main idea of our approach is to formulate the learning of dynamic budget predictors as a generalized risk minimization that involves the distribution of budgets provided for the application. The distribution of possible budgets has been either previously neglected or assumed to be uniform [12]. However, we show that such an easily available prior information can significantly help to improve the expected accuracy.

Our formulation leads to a straight-forward modification of convolutional neural network (CNN) architectures and their training. In particular, we add several early prediction and loss layers along the standard processing pipeline of a DNN (Figure 1 and Figure 2). According to our risk minimization framework for dynamic budget predictors, all of these layers need to be learned jointly with a weighted combination derived from a time-budget distribution. Whereas this strategy is directly related to DNN learning strategies, such as deep supervision [24] and inception architectures [23], we demonstrate its usefulness for adapting to varying resources during testing.

The paper is structured as follows. After discussing related work, we define dynamic budget predictors and derive a new learning framework based on risk minimization with budget distributions (Sect. 2). Our framework can be directly applied to deep and especially convolutional neural networks as described in Sect. 3. Experiments in Sect. 4 show the advantages of our approach for different architectures, datasets, and budget distributions.

Related work on anytime prediction The work of Karayev *et al.* [12] presented an approach that iteratively and dynamically selects feature representations to maximize the area above an entropy vs. cost curve. Our approach however focuses on a static order of predictors and is able to incorporate budget distributions expected for the application. Fröhlich *et al.* [8] proposed a semantic segmentation approach with anytime classification capability. Their method is based on random decision forests learned in a layer-wise fashion. Xu *et al.* [26] considers anytime classification with unknown budgets by combining a cost-sensitive support vector machine with feature learning. Similar to [8], their predictors are learned in a greedy fashion and not learned jointly as in our case. Learning all of the predictors with shared parameters jointly allows us to share computations while directly optimizing with respect to expected accuracy during training. The paper of [25] presents an algorithm for

learning tree ensembles with a constrained time budget available during training. In our case, the whole distribution budgets is given during training.

Related work on deep supervision and DNNs with multiple losses There are multiple methods that use a similar architecture of deep neural networks than ours characterized by multiple loss layers and joint training of them. For example, [24] refers to such a training strategy as "deep supervision" and shows that it allows for training deeper networks in a robust fashion. A very similar technique has been used in [7] for improved scene recognition. Furthermore, multiple loss layers are often used for multi-task learning, where the goal is to jointly predict various outputs [27].

In contrast to these works, our paper focuses on the impact of such an architecture on the ability of DNNs to deal with dynamic time budgets during inference. Furthermore, we show that such an architectural design can be directly derived from a very general risk minimization framework for predictors with dynamic budgets.

Related work on speeding up convolutional neural networks There are multiple works that focus on speeding up DNNs and the special case of convolutional neural networks (CNNs). Applied and adapted techniques range from low-rank approximations [2, 6, 10] to FFT computations of the involved convolutions [19]. The Fast R-CNN method of [6] speeds up fully-connected layers by simple SVD approximation. Similar techniques have been presented by [2] and [10]. The paper of [8] provides an empirical study of the effects of CNN architectural design choices on the computation time and the achieved recognition performance. A straightforward technique to speed up convolutions with large filter sizes uses Fast Fourier Transforms as studied by [19]. Furthermore, efficient filtering techniques, such as the Winograd transformation [14], are applicable as well.

Our approach also tries to speed up inference of deep neural networks, *i.e.* a forward pass. However, instead of approximating different operations performed in single layers, we achieve a significant speed-up by allowing the algorithm to deal with dynamic time budgets. Therefore, our research is orthogonal to the one briefly described and combining them is straightforward.

2 Learning Dynamic Budget Predictors

In this section, we derive a simple yet powerful learning scheme for dynamic budget predictors. Without loss of generality, we focus on time budgets in the following.

Specification of dynamic budgets An important challenge for dynamic budget approaches is that the budget available for inference during testing is not known during training and for anytime scenarios even not known during inference itself. For anytime tasks, we need to learn algorithms that can be interrupted at several time steps and balance the trade-off between calculating direct predictions of an output y for an example $\boldsymbol{x}$ or performing calculating intermediate outputs that help later on for further refinements of the predictions.

This trade-off is without any further specification, ill-posed. However, in many applications, we know something about the distribution $p(t \mid \boldsymbol{x}, y)$ of time budgets t available to the algorithm for a given input-output pair $(\boldsymbol{x}, y)$. In the following, we assume that this distribution is either given or can be modeled for an application.

Risk minimization with budget distributions In the following, we develop a framework for learning dynamic budget predictors using risk minimization. We consider inference algorithms f that provide predictions $y \in \mathcal{Y}$ for input examples $\boldsymbol{x} \in \mathcal{X}$ at different times $t \in \mathbb{R}$, *i.e.* we have $f : \mathcal{X} \times \mathbb{R} \rightarrow \mathcal{Y}$.

Learning the parameters $\boldsymbol{\theta}$ of f is done by minimizing the following regularized risk:

$$\operatorname{argmin}_{\boldsymbol{\theta}} \int_{t\in\mathbb{R}} \int_{y\in\mathcal{Y}} \int_{\boldsymbol{x}\in\mathcal{X}} \mathcal{L}(f(\boldsymbol{x},t;\boldsymbol{\theta}),y) \cdot p(\boldsymbol{x},y,t)\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}y\,\mathrm{d}t + \mathcal{R}(\boldsymbol{\theta}) \quad , \tag{1}$$

with $\mathcal{L}$ being a suitable loss function, $\mathcal{R}(\boldsymbol{\theta})$ being a regularization term, and $p(\boldsymbol{x},y,t)$ being the joint distribution of an input-output pair $(\boldsymbol{x},y)$ and the available time t. This formulation does not require any differentiation between a-priori given budget or anytime scenarios.

We further assume that the time available is independent of the actual example and it's label. This is a reasonable assumption, since the available time is in most applications just based on a limitation of hardware or data transfer resources. Since we are given a training set $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^n$, learning is based on minimizing the empirical risk:

$$\operatorname{argmin}_{\boldsymbol{\theta}} \int_{t\in\mathbb{R}} \sum_{i=1}^{n} \mathcal{L}(f(\boldsymbol{x}_i,t;\boldsymbol{\theta}),y_i) \cdot p(t)\mathrm{d}t + \mathcal{R}(\boldsymbol{\theta}) \quad . \tag{2}$$

The predictor f is an algorithm performing a finite sequence of atomic operations. Therefore, the prediction output will be only changing at discrete time steps $t_1,\ldots,t_K$:

$$\forall \boldsymbol{x}\, \forall 1 \leq k < K\, \forall t_k \leq t < t_{k+1} : \quad f(\boldsymbol{x},t;\boldsymbol{\theta}) = f(\boldsymbol{x},t_k;\boldsymbol{\theta}) \stackrel{\text{def.}}{=} f_k(\boldsymbol{x};\boldsymbol{\theta}_k), \tag{3}$$

$$\forall \boldsymbol{x}\, \forall t \geq t_K : \quad f(\boldsymbol{x},t;\boldsymbol{\theta}) = f_K(\boldsymbol{x};\boldsymbol{\theta}_K) \quad . \tag{4}$$

Furthermore, before t_1, no output estimate is available. Since this leads to a constant additive term independent of $\boldsymbol{\theta}$, we can ignore this aspect in the following. In total, Eq. (2) simplifies as follows:

$$\operatorname{argmin}_{\boldsymbol{\theta}} \sum_{k=1}^{K} w_k \cdot \left(\sum_{i=1}^{n} \mathcal{L}(f_k(\boldsymbol{x}_i;\boldsymbol{\theta}_k),y_i) \right) + \mathcal{R}(\boldsymbol{\theta}) \quad , \tag{5}$$

with weights $w_k = \int_{t_k}^{t_{k+1}} p(t)\mathrm{d}t$ for $1 \leq k < K$ and $w_K = \int_{t_K}^{\infty} p(t)\mathrm{d}t$. As can be seen we have a simple learning objective, which is a weighted combination of the learning objectives of each of the individual predictors f_k. If some of the parameters are shared between the predictors, which is the case for our approach presented in Sect. 3, each term in the objective can not be optimized independently and joint optimization is necessary. Sharing parameters is essential for optimizing shared computations towards maximizing the expected accuracy of the complete model.

The information about the time-budget distribution defines the weights of the loss terms in an intuitive manner: if there is a high probability of the time budget being between t_k and t_{k+1}, the loss of f_k has a strong impact on the overall learning objective and the parameters $\boldsymbol{\theta}_k$ including the shared ones should be tuned towards reducing the loss of f_k rather than contributing significantly to other predictors.

3 Learning Impatient DNNs with Early Prediction Layers

In this section, we show how a single deep neural network with additional prediction layers is well suited for providing a series of prediction models.

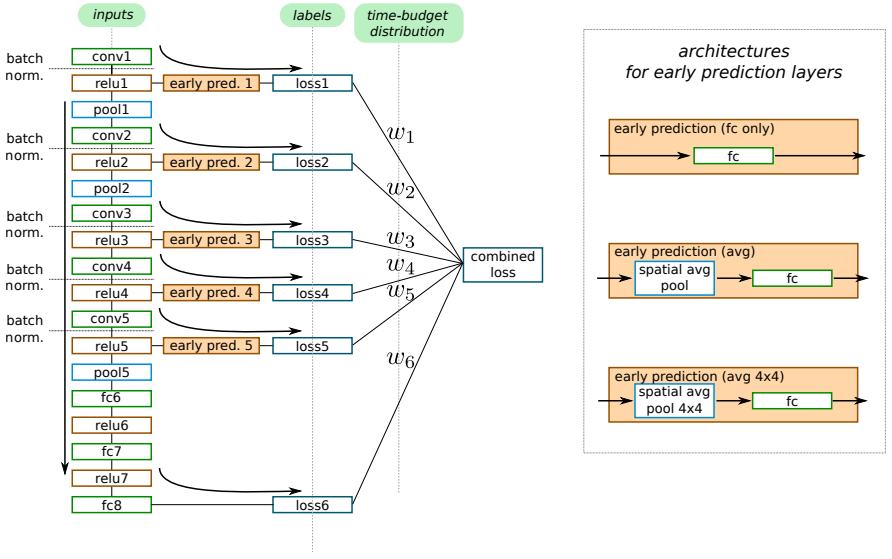

Figure 2: (Left) Modification of the AlexNet architecture for dynamic budgets and early predictions. (Right) Possible architectures for early prediction.

Early prediction layers To obtain a series of predictions, we add K additional layers to a common DNN architecture as illustrated in Figure 2. We refer to these layers as early prediction (EP) layers in the following. The output $f_k(\boldsymbol{x})$ of these layers has as many dimensions as y. Already after the first layers, our approach is able to perform predictions with only a very few number of computational operations. The layered architecture of a DNN has an important advantage, since all f_k naturally share a large set of their parameters and also a large number of computations. Anytime approaches require a forward pass to go through all early prediction layers that can be processed until interruption. In case of non-parallel computation, the computational overhead of the early prediction layers should therefore be reduced as much as possible.

The right part of Figure 2 shows different choices for EP layers we experimented with: (1) FC only, which is a simple single fully-connected (FC) layer followed by a softmax layer, (2) AVG, which performs average pooling across the spatial dimensions of previous layer before a fully-connected layer, which leads to a significantly reduced number of parameters for the EP layers, and (3) AVG 4×4, which allows for preserving rough spatial information by performing average pooling in $4 \times 4 = 16$ uniformly-sized regions.

Learning with weighted losses For learning, each of the EP layers is connected to a loss layer. The overall loss during training is exactly the weighted combination we derived in the previous section in Eq. (2).

In theory, training our Impatient DNNs does not require any further modifications and learning can be done with standard back-propagation and gradient-descent. However, we observed in experiments that batch normalization [9] leads to a significantly more robust training and is even required to achieve convergence at all in most cases.

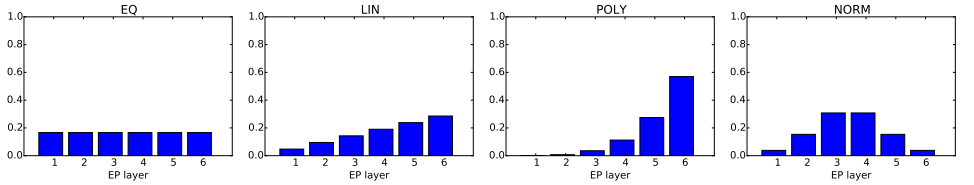

Figure 3: Types of time-budget distributions we consider in our paper.

Weighting schemes In our experiments, we are interested in the effect of different time-budget distributions provided during learning. To simulate them, we consider the following schemes for early prediction layer weights $w_1, \ldots, w_K$: (STD) standard DNN training, *i.e.* only the last prediction matters: $w_K = 1$ and $w_k = 0$ otherwise, (EQ) uniform weights for uniform time-budget distributions: $w_k = \frac{1}{K}$, (LIN) linearly increasing weights, *i.e.* small time budgets are unlikely: $w_k \propto k$, (POLY) polynomially increasing weights: $w_k \propto k^\gamma$ with $\gamma > 1$, (ILIN, IPOLY) decreasing weights, *i.e.* small time budgets are likely: $w_k = w'_{K+1-k}$ for weights w'_k of the former schemes, and (NORM) small and large time budgets are rare and layers in the middle of the architecture are given a high weight: $w_k \propto \exp(-\beta \cdot (k - \frac{K+1}{2})^2)$ with $\beta = 0.34$. All of these schemes are simulating different budget specifications of an application. An illustration of several instances is given in Figure 3.

4 Experiments

In the following, we evaluate our approach with respect to different dynamic budget schemes and compare with standard DNN training and other relevant baselines.

Experimental setup and datasets For evaluation, we conducted experiments on two object classification datasets. The **15-Scenes** [15] dataset comprises a total of 4,485 images covering categories from kitchen and living room to suburban and industrial. Each category contains between 200 and 400 images each, from which we took 100 images for training, as suggested by [15], and the remaining ones for testing. The training set is further divided into 90 images for actual training and 10 images for validation. The **MIT-67** [20] indoor scenes database is comprised of 67 categories. We follow the procedure of [20] and take 80 images for training and 20 for testing. Again, the training set is split in order to obtain a validation set of 8 images per class.

Since our datasets are too small for DNN training from scratch, we perform fine-tuning of different models pre-trained on ImageNet, *e.g.* AlexNet [13] and VGG19 [22]. The positions of EP layers for AlexNet are given in Figure 2. For VGG19, we add EP layers after each block of convolutional layers. Please note that the last "early" prediction layer is always the output layer of the original CNN architecture.

Analysis of learning Impatient DNNs In the following, we show that for learning Impatient DNNs care has to be taken to ensure convergence. For example, an adequate learning rate has to be determined to ensure convergence of the network while avoiding saturation at low accuracy. This becomes much more important when dealing with losses of multiple branches, since the gradients at shared layers accumulate leading to the network training

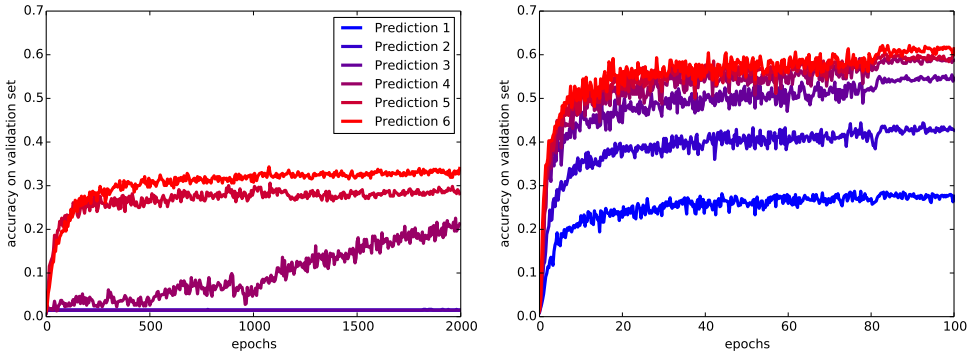

Figure 4: Convergence during learning an Impatient AlexNet trained on MIT-67 with (right) and without (left) batch normalization: Different colors indicate individual early prediction layers and it can be clearly seen that batch normalization significantly improves stability during training.

being more fragile. Especially in the case of deeper network architectures, *e.g.* VGG, we observed that convergence can not be achieved at all without proper normalization.

Therefore, we made use of batch normalization [9] which rectifies the covariate shift in the input data distribution of each convolution layer. This technique allows for training with much higher learning rates ensuring faster convergence and in our case convergence at all. In Figure 4 (left), an example of optimizing an Impatient AlexNet is shown where the validation accuracy for early prediction layers saturates very slow at a low value caused by a highly decreased learning rate of 10^{-4}. Even no convergence is achieved for very early layers after running 2000 epochs of training. In contrast, adding batch normalization (right-hand side) allows for a $100\times$ higher learning rate resulting in very fast convergence at a high level of validation accuracy for all prediction layers.

Evaluation of early prediction architectures As presented in Sect. 3, several architectures are possible for early prediction. The straightforward approach of connecting FC layers directly to each convolutional layer leads to a huge amount of additional parameters to be optimized. These layers are prone to overfitting. This can be seen in the learning statistics for MIT67 with a VGG19 base architecture shown in Figure 5. The training loss is near zero together with a moderate validation accuracy for early layers. We also experimented with multiple FC layers. However, learning of these architectures failed to converge in all cases independently from the choice of hyperparameters. By applying spatial pooling layers, validation accuracy is substantially improved, which can be seen in Figure 5 (AVG and AVG4x4). Especially AVG4x4 provides rough spatial information which helps to improve performance even further. Therefore, we use this architecture in the following experiments.

In the last two columns of Table 1, average computation times according to the particular weighting schemes and budget distributions are presented for a single image. If inference is performed up to a particular prediction layer known in advance, previous prediction layers do not have to be assessed and we achieve low prediction times t_{B} without additional overhead. Interruptable prediction in anytime scenario A (t_{A}) requires inference of all intermediate prediction layers caused by the potential sudden interruption. In the worst case, *i.e.* the forward pass includes all prediction layers, average computation time increases compared to

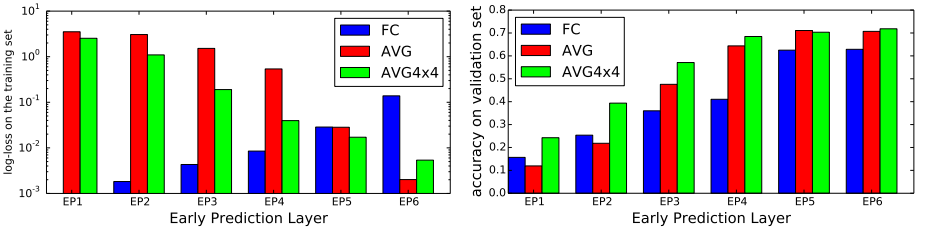

Figure 5: Comparison of different early prediction architectures of an Impatient VGG19 trained on MIT-67. Replacing fully-connected layers (FC) by spatial average pooling (AVG & AVG4x4) reduces the effect of overfitting resulting in higher validation accuracy.

the scenario with a-priori given budgets. All experiments were performed on an NVIDIA GeForce GTX 970 GPU.

Does joint training of EP layers help? The most interesting question, however, is whether our joint training scheme motivated in Sect. 2 provides superior results compared to learning predictors independently. To answer this question, we compared our approach with different baselines that learn several SVM classifiers based on extracted CNN features [3] at each early prediction layer. We optimize SVM hyperparameters on the validation set to allow fair comparison. The underlying networks, on the contrary, differ in the sense that we made use of an original CNN pre-trained on ImageNet and a pre-trained CNN fine-tuned on the current dataset.

In Table 1, the evaluation for different time-budget distributions is presented where each result shows the expected accuracy according to the particular weighting scheme and budget distribution. It can be clearly seen that the original CNN (ORIG) without the adaptation to the current dataset performs worst. By applying fine-tuning (FT), however, accuracy can be noticeably increased for all early prediction SVMs.

Our joint learning of the EP layers provides superior results in almost all scenarios. Especially in the case of small time budgets our method benefits from taking the budget distribution during learning into account resulting in an improvement of almost 10% on MIT-67 and 6% on 15-Scenes for an Impatient VGG19 compared to the best performing baseline. For extreme weighting schemes with high priority on later predictions (**POLY** $\smile$), fine-tuning of the original networks provides slightly better results compared to our approach. This is not surprising since in this case training is very similar to that of standard DNNs with only one final loss layer.

In Table 2, we compared our approach to state-of-the-art results for MIT-67 and 15-Scenes. Although the focus of this paper is rather on anytime capability while running the risk of dropping accuracy at final layers, we achieved superior results. It should be noted that only the last layer is used to obtain predictions, since we assume to have no budget restrictions. Especially for the jointly trained Impatient VGG19 on MIT-67, it was even possible to outperform the standard fine-tuned CNN, which supports the idea of "deep supervision" [24].

Cascaded prediction Apart from both scenarios presented in Figure 1, efficient classification constitutes another interesting application of our approach. The task here is–for a given set of examples–to reach a desired accuracy within a minimal but not fixed amount of time.

VGG19 Budget Scheme	MIT-67 Orig	FT	Ours	15-Scenes Orig	FT	Ours	$\varnothing t_B$ [ms]	$\varnothing t_A$ [ms]
EQ−	46.65	48.07	**53.93**	83.37	84.28	**85.63**	1.11	1.19
LIN⟋	54.19	56.52	**60.55**	85.87	87.47	**88.02**	1.37	1.47
POLY⌒	62.82	67.07	**69.66**	88.71	**91.71**	90.88	1.72	1.84
ILIN⟍	37.25	37.71	**45.62**	77.56	77.73	**80.87**	0.82	0.86
IPOLY⌣	25.63	25.65	**35.11**	70.14	69.85	**75.93**	0.50	0.51
NORM∧	47.53	47.90	**55.38**	84.46	84.74	**86.67**	1.07	1.15
AlexNet Budget Scheme	**MIT-67 Orig**	**FT**	**Ours**	**15-Scenes Orig**	**FT**	**Ours**	$\varnothing t_B$ [ms]	$\varnothing t_A$ [ms]
EQ−	41.75	46.19	**48.40**	82.56	84.28	**85.11**	0.68	0.75
LIN⟋	45.19	50.96	**52.13**	83.73	**86.19**	85.94	0.79	0.89
POLY⌒	48.50	**56.29**	55.76	85.56	**88.98**	87.38	0.96	1.09
ILIN⟍	36.64	39.59	**42.91**	78.10	79.03	**81.87**	0.54	0.59
IPOLY⌣	28.69	30.17	**36.14**	72.38	72.48	**77.85**	0.40	0.42
NORM∧	43.97	47.80	**49.93**	83.25	84.82	**84.89**	0.65	0.72

Table 1: Comparison of Impatient AlexNet (top) and VGG19 (bottom) CNNs with several baselines. Performance is measured by expected accuracy in % based on the particular budget distribution.

Dataset	Orig	FT	Ours (eq)	Ours (poly)	PlacesCNN [28]	[18]*
MIT-67	65.0%	71.04%	67.23%	**71.71**%	68.24%	71.5%
15-Scenes	88.30%	**92.83**%	92.13%	91.45%	90.19%	-

Table 2: How good are our VGG19 Impatient Networks when there are no budget restrictions during testing? The table shows the accuracy of the last prediction layer also compared to state-of-the-art results. * The method of [18] requires more than 4*s* per image.

In particular, interrupting the network at a certain depth might already provide the correct decision which renders further computation unnecessary. To implement the idea of efficient inference, an adequate stopping criterion has to be defined. Since each early prediction layer provides probabilistic outputs, we applied uncertainty-based decision making by calculating the ratio between the two highest class probabilities, which is known as 1-vs-2 strategy [11]. If the current prediction of class probabilities is characterized by a high ratio, inference can be interrupted.

The analysis of the proposed criterion can be seen in Figure 6 showing time-accuracy plots. Thereby, one point on the red graph is obtained by a fixed ratio threshold which determines whether an early layer prediction already reaches sufficient certainty and thus provides the final decision. The blue graph, however, represents classification results of each early prediction layer itself, *i.e.*, the final decision is made at always the same depth, independently of the underlying ratio. As can be seen, by using uncertainty-based predictions, accuracy can be increased substantially in a lot of cases with the same computational efforts. For example, by interrupting the AlexNet network at the fifth prediction layer consistently takes $\sim$ 1 ms per image for MIT-67 (second-last plot in Figure 6). In contrast, using the proposed criterion, accuracy can be increased from 53% up to 57% while still requiring exactly the same computation time on average. An entropy-based criterion achieved inferior performance in our experiments.

Qualitative results In Figure 7, qualitative results for the task of scene recognition (class "bathroom" from MIT-67) are shown. Different numbers in each image indicate the early

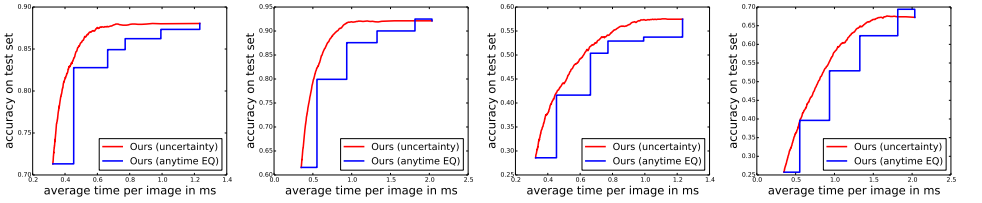

Figure 6: Evaluation of uncertainty-based predictions compared to early layer predictions. From left to right: Impatient AlexNet on 15-Scenes, Impatient VGG19 on 15-Scenes, Impatient AlexNet on MIT-67, and Impatient VGG19 on MIT-67.

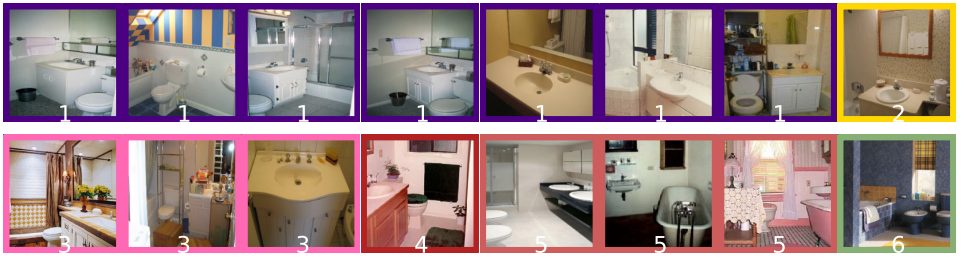

Figure 7: Images of the MIT-67 first correctly classified as "bathroom" at different early prediction layers of an Impatient VGG19 CNN. The position of the layers is highlighted as a number and a uniquely colored border.

prediction layer in which the particular example was first correctly classified. It can be clearly seen that the examples already decided at EP1 are white colored bathrooms with clearly visible toilet bowl, shower, and sink. With increasing complexity of the scene, layer depth increases as well to provide correct decisions. For example, the right most images in the second row of Figure 7 shows extraordinary bathrooms of unusual colored walls and furnishings increasing the likelihood of confusion with other classes, *e.g.* children room.

5 Conclusions

In this paper, we presented impatient deep neural networks that tackle the problem of classification with dynamic time budgets during application. Compared to standard DNNs which suffer from a high computational demand during inference, we showed that our approach allows for anytime prediction, *i.e.* a possible interruption at multiple stages while still providing output estimates which renders our method suitable even for real-time applications. We presented a novel general framework of learning dynamic budget predictors based on risk minimization, which we adapted directly to state-of-the-art convolutional neural network architectures by branching additional early prediction layers with weighted losses. Based on a set of object classification datasets and architectures, we showed that our approach provides superior results for different time budget distributions. Furthermore, we developed an uncertainty-based prediction framework allowing for reducing computational costs while still providing the same accuracy.

References

[1] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *arXiv preprint arXiv:1604.01685*, 2016.

[2] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *CoRR*, abs/1404.0736, 2014.

[3] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[4] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *ICRA*, 2016.

[5] Björn Fröhlich, Erik Rodner, and Joachim Denzler. As time goes by: Anytime semantic segmentation with iterative context forests. In *Symposium of the German Association for Pattern Recognition (DAGM)*, pages 1–10, 2012.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[7] Sheng Guo, Weilin Huang, and Yu Qiao. Locally-supervised deep hybrid model for scene recognition. *arXiv preprint arXiv:1601.07576*, 2016.

[8] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. *CoRR*, abs/1412.1710, 2014. URL http://arxiv.org/abs/1412.1710.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[10] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[11] Ajay J Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. Multi-class active learning for image classification. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2372–2379. IEEE, 2009.

[12] Sergey Karayev, Mario Fritz, and Trevor Darrell. Anytime recognition of objects and scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 572–579, 2014.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[14] Andrew Lavin. Fast algorithms for convolutional neural networks. *CoRR*, abs/1509.09308, 2015.

[15] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[16] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. *arXiv preprint arXiv:1506.02515*, 2015.

[17] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

[18] Lingqiao Liu, Chunhua Shen, and Anton van den Hengel. The treasure beneath convolutional layers: Cross-convolutional-layer pooling for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4749–4757, 2015.

[19] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

[20] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 413–420. IEEE, 2009.

[21] David Silver, J Andrew Bagnell, and Anthony Stentz. Learning autonomous driving styles and maneuvers from expert demonstration. In *Experimental Robotics*, pages 371–386. Springer, 2013.

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[24] Liwei Wang, Chen-Yu Lee, Zhuowen Tu, and Svetlana Lazebnik. Training deeper convolutional networks with deep supervision. *arXiv preprint arXiv:1505.02496*, 2015.

[25] Zhixiang Xu, Kilian Weinberger, and Olivier Chapelle. The greedy miser: Learning under test-time budgets. *arXiv preprint arXiv:1206.6451*, 2012.

[26] Zhixiang Xu, Matt Kusner, Gao Huang, and Kilian Q Weinberger. Anytime representation learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1076–1084, 2013.

[27] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Learning deep representation for face alignment with auxiliary attributes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(5):918–930, May 2016. ISSN 0162-8828. doi: 10.1109/TPAMI.2015.2469286.

[28] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Advances in neural information processing systems*, pages 487–495, 2014.