

Sequential Analysis with Specified Confidence Level and Adaptive Convolutional Neural Networks in Image Recognition

Andrey Savchenko*[†]

*Samsung-PDMI Joint AI Center

St. Petersburg Department of Steklov Institute of Mathematics

[†]Laboratory of Algorithms and Technologies for Network Analysis

National Research University Higher School of Economics

Nizhny Novgorod, Russia

Email: avsavchenko@hse.ru

Abstract—In this paper the problem of high computational complexity of deep convolutional nets in image recognition is considered. An existing framework of adaptive neural networks is extended by appending the separate classifier to intermediate layers. The hierarchical representations of the input image are sequentially analyzed. If the first classifier returns rather high confidence score, the inference process will be terminated. Otherwise, the inference to the next intermediate layer with attached classifier is continued until the reliable solution is obtained or the penultimate layer is reached. The thresholds for classifier scores at each layer are automatically chosen based on the Benjamini-Hochberg multiple comparisons for a specified confidence level. Experimental study for both pre-trained and fine-tuned deep convolutional neural networks demonstrates that the proposed approach reduces the running time by up to 1.7 times without significant accuracy degradation. Moreover, the larger is the training sample, the more noticeable is the gain in performance.

Index Terms—adaptive convolutional neural network, dynamic neural network, sequential analysis, multiple comparisons

I. INTRODUCTION

The task of the closed-set multi-category image recognition is to assign an observed image X represented by a tensor of RGB values to one of $C > 1$ classes (categories). The classes are specified by the training set of $N \geq C$ reference images X_n . I focus on the supervised learning case, when the class label $c(n) \in \{1, \dots, C\}$ of the n -th image is known. In particular, I deal with a few-shot learning problem [1], in which the training sample is rather small to train a complex classifier, e.g. a deep convolutional neural networks (CNN) [2], [3], from scratch. In such a case, two possible techniques can be applied for the deep CNN, which was preliminarily trained on an external very large dataset, e.g. ImageNet. The first one uses this CNN as a feature extractor: input image and all training images from the limited sample of instances

are fed to the pre-trained CNN in order to extract *off-the-shelf* features (embeddings) [4] at the output of one of the last layers. Such deep learning-based feature extractors allow training of a general classifier, e.g., SVM (support vector machine), RF (random forest), gradient boosting or simple k-NN (nearest neighbor) rule, that performs nearly as well as if a large dataset of images from these C classes is available [5]. However, the most popular is the second variant, namely, *fine-tuning* of the CNN on the given training set with replacement of several last layers by new fully connected layer, i.e. LR (logistic regression) classifier.

In both techniques the recognition process includes feeding an observed image X to the the CNN with large depth and performing an inference (forward pass) in this network. Various recent studies have shown that the increase in the number of layers (depth) lead to much more accurate solutions [2], [6]. The increase of the number of layers leads to increase of the the running time of inference in contemporary deep models. As a result, this time may be too high for real-time processing [7]. In this paper I address the challenging problem of increasing the speed of image recognition without significant degradation in accuracy by using sequential analysis [8], which was proved to improve performance in many classification tasks [9]. In particular, the framework of adaptive neural networks [10] is explored to terminate the inference if the reliable solution of image recognition can be obtained during an analysis of the outputs of intermediate layers.

The main contribution of this paper includes the novel approach for speeding-up CNNs by adaptively adjusting the running time depending upon the difficulty of the input image. In contrast to known techniques [11], [12], the proposed algorithm does not require to learn the weights of the CNN from scratch using large training datasets and can be applied with *existing* neural models. In particular, separate classifiers are learned for the hierarchical representations of the input image obtained as the outputs of small number of intermediate layers. The decisions made by these classifiers are sequentially analyzed. It is proposed to automatically choose the thresholds

This research is based on the work supported by Samsung Research, Samsung Electronics. The work in the Section IV was prepared within the framework of the Basic Research Program at the National Research University Higher School of Economics (HSE).

for classifier scores at each layer by using the multiple comparisons (multiple hypothesis testing) [13] if a confidence level is specified for the whole decision-making process. It is demonstrated that the most accurate and fast results are achieved for linear classifiers trained in one-versus-all fashion in order to maximize the squared hinge loss.

The rest of the paper is organized as follows. Section II reviews existing works devoted to adaptive CNNs in image recognition. In Section III, the proposed algorithm is presented. In Section IV, its potential to decrease the running time is shown in experiments with several datasets and CNN architectures. Concluding comments are given in Section V.

II. LITERATURE SURVEY

The need for improving the speed of CNNs [7] has become evident when it was shown that the deeper is the model, the more accurate are its results. The CNN compression with structural pruning [14] removes entire convolutional channels in order to reduce the model size speed-up the inference. Matrix decomposition represents weight matrices (or tensors) in each layer as a sequence of operations with lower order matrices and vectors [15]. However, all of these techniques deal mainly with CNN compression and are not supposed to obtain the highest accuracy of very deep models.

In this paper I consider another approach based on dynamic computational graph that exploits the observation that features learned at an early layer of a network may often be sufficient for the classification of many data points [12]. Conditional Deep Learning (CDL) network [11] introduces linear classifiers, e.g., logistic regression (LR), to each convolutional layer and monitors their outputs to decide whether an inference can be exited early. Only results for MNIST dataset and simple CNNs have been provided due to the difficulty of the training methodology. The BranchyNet [12] introduced early exits (branches) along the CNN architecture. It is trained in end-to-end fashion and usually requires large training datasets of images with low resolution, so that its authors provide the results for MNIST and CIFAR10 datasets. The adaptive computational time model was extended for application in image classification and object detection [16]. Here sigmoidal halting units were added to the residual blocks, so that this approach is applicable only for the ResNet architectures.

Adaptive neural networks for efficient inference have been proposed in [17] with automatic selection of proper pre-trained deep model based on the outputs of AlexNet. Adaptive inference graphs decreased the number of floating-point operations by using gated inference in the ResNet models [18]. The results of end-to-end learning of this network for CIFAR-10 and ImageNet datasets proved its faster speed when compared to baseline ResNet-50/101 models. Cascading of different ResNet-based modules [19] caused a noticeable speedup of decision-making for end-to-end learning on CIFAR-10, CIFAR-100 and SVHN datasets. The HydraNets transform the state-of-the-art architectures into adaptive architectures which exploit conditional gating mechanism in order to reduce the

inference cost by dynamically choosing components of the network to evaluate at runtime [10].

Thus, one can conclude that though there exist several papers devoted to dynamic (adaptive) CNNs, which can decrease the running time of image recognition when the large training set is available, the few-shot learning case is completely ignored in these studies. All such papers contain experimental results for learning from scratch on such large datasets as MNIST, CIFAR and ImageNet, and their techniques cannot be used in transfer learning and domain adaptation frameworks. As a matter of fact, most of these papers contain only the estimates of the number of operations (e.g., FLOPs or MACs) rather than the measurements of inference time. However, it was experimentally noticed that the absolute time of condition checks on CPU and, especially, GPU, is much higher when compared to the time for matrix multiplication. Hence, the real inference speed-up of existing methods can be less significant. Finally, the parameters of termination conditions, e.g., thresholds for entropy [12] or maximal posterior probability [11], are arbitrarily chosen without any attempt to specify the overall confidence level of decision-making procedure. In this paper the possibility to overcome the above-mentioned drawbacks of the known adaptive CNNs is studied by using sequential analysis and the theory of multiple comparisons [13].

III. PROPOSED APPROACH

Inspired by sequential statistical analysis [8], $M > 1$ intermediate layers (exit branches [12]) are arbitrarily chosen. Let me assume that the architecture of a CNN allows to split the whole computational graph into M sequentially connected parts so that the inference process is recursive:

$$\mathbf{z}_m = f_{exit_m}(\mathbf{z}_{m-1}; \boldsymbol{\theta}), m \in \{1, \dots, M\}, \quad (1)$$

where \mathbf{z}_0 is initialized by the RGB matrix of an input image X , $\boldsymbol{\theta}$ is a vector of all weights of the CNN, and f_{exit_m} is the output of the m -th part (exit branch). These outputs are used to identify the variability in the difficulty of input images and conditionally activate the deeper layers of the CNN.

The output of typical convolutional layers is a multi-dimensional tensor. Hence, it is necessary to add conventional global average pooling, reshape and (optionally) L_2 normalization layers to every exit branch in order to transform activations \mathbf{z}_m into a convolutional layer feature vector \mathbf{x}_m . As such transformation has no parameters, it can be added even to existing pre-trained CNN. However, the best efficiency for rather large training sample is achieved by fine-tuning of the CNN, i.e. adding fully connected layer (LR classifier) to every exit branch and training the whole network using a weighted sum of losses from all exits [12].

In any case, the input image is represented as a hierarchy of feature vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M$. To speed-up the image recognition, I will use the sequential decision-making [9], which assign three decision regions at each representation level and examine the next levels with more detailed information only when there is a need for doing so. The known

implementations of adaptive neural networks use the multi-class LR classifier either simultaneously trained during fine-tuning [12] or separately trained on a given training set [11]. In order to measure of how confident the classifier at an exit point is about an observed image, these networks compute some statistics, e.g., the entropy [12], based on a set of the class posterior probabilities at the output of the LR layer. If this statistics (confidence) for the n -th layer is a greater than a fixed threshold, the decision at this layer is returned. Otherwise, inference in the CNN is continued and the outputs at the $(n + 1)$ exit are analyzed.

In this paper there is no restriction to use the LR classifiers. During the training procedure, all instances X_r are fed into the input of the CNN in order to be represented by feature vectors $\mathbf{x}_{n;1}, \mathbf{x}_{n;2}, \dots, \mathbf{x}_{n;M}$. As a result, it is possible to train M appropriate classifiers so that the parameters of the m -th classifier are learned using a set of examples $\{(\mathbf{x}_{n;m}, c(n))\}, n \in \{1, \dots, N\}$. Without a lack of generality, let us assume that each classifier represents a decision function $\mathbf{s}_m(\mathbf{x}_m) = [s_m^{(1)}(\mathbf{x}_m), \dots, s_m^{(C)}(\mathbf{x}_m)]$, where $s_m^{(c)}(\mathbf{x}_m) \geq 0$ is a confidence score of the classifier. For example, the confidence score in the one-versus-all SVM for an observation is the signed distance of that observation to the hyperplane, which separates the c -th class with all other classes. The decision is made in favor of the class with the maximal confidence

$$c_m^*(\mathbf{x}_m) = \operatorname{argmax}_{c \in \{1, \dots, C\}} s_m^{(c)}(\mathbf{x}_m). \quad (2)$$

Hence, the termination condition (acceptance decision in the three-way decisions) at the n -th branch can be written by simple matching of the maximal score with a certain threshold:

$$\max_{c \in \{1, \dots, C\}} s_m^{(c)}(\mathbf{x}_m) > s_m. \quad (3)$$

In this paper I propose to choose the threshold s_m by fixing the false acceptance rate (FAR) α_m for decisions at the m -th level. This procedure is theoretically well-studied for several specific cases, e.g., the 1-NN classifier with the Kullback-Leibler (KL) divergence between (positive) feature vectors \mathbf{x}_m and $\mathbf{x}_{n;m}$. If the input image corresponds to class c , then the KL distance between this image and each other class $i \neq c$ has asymptotic non-central chi-squared distribution with the non-centrality parameter proportional to the distance between classes c and i . As a result, if the confidence score is an inverse of the KL distance, then the threshold s_m is defined as an inverse α_m -quantile of the non-central chi-squared distribution [9], [20].

However, thresholds in (3) should be estimated empirically for *arbitrary* classifiers. Let us extract $0 < K < N$ training examples with indices $\{n_1, \dots, n_K\}$ in a stratified fashion from the whole set $\{(\mathbf{x}_{n;m}, c(n))\}, n \in \{1, \dots, N\}$ and use it to initially fit the m -th classifier. Next, the outputs of this classifier for the rest $(N - K)$ examples are computed, and

threshold s_m is estimated from the following equation:

$$\sum_{n \notin \{n_1, \dots, n_K\}} H \left[\max_{c \neq c(n)} s_m^{(c)}(\mathbf{x}_{n;m}) - s_m \right] = \lfloor \alpha_m(N - K) \rfloor, \quad (4)$$

where $H[s]$ is the Heaviside step function. This equation is solved by evaluating a α_m -quantile of the maximal scores of other classes $\left\{ \max_{c \neq c(n)} s_m^{(c)}(\mathbf{x}_{n;m}) \mid n \notin \{n_1, \dots, n_K\} \right\}$. Next, the m -th classifier is retrained using complete training set $\{(\mathbf{x}_{n;m}, c(n))\}, n \in \{1, \dots, N\}$.

As the number of steps in decision process adaptively depend on the input image, it is not obvious how to choose concrete values $\alpha_1, \dots, \alpha_M$ if a confidence level α for the whole image recognition procedure is specified only. Here the parameters α_m do not stand for the FAR in the described sequential analysis because making a decision at the m -th layer depends on decisions from the previous levels. As a matter of fact, it is a multiple-testing problem, so appropriate correction should be used. One can simply choose the Bonferroni correction and assign $\alpha_m = \alpha/M$, but such correction is too strict [13]. If one would like to control the false discovery rate, the Benjamini-Hochberg test [21] can be applied. Unfortunately, this test involves the sorting of p-values of all hypothesis, so it cannot be directly applied in the proposed sequential procedure, when the results for $(m+1), \dots, M$ layers are not available at the m -th step. Hence, in this paper I assume that the reliability of decision-making increases with an increase of the branch index m . In such case, the Benjamini-Hochberg correction is straightforwardly implemented: $\alpha_m = \alpha \cdot m/M$.

At this point complete Algorithm (Fig. 1 and Fig. 2) can be introduced. Their parameters include M exit layers, the specified confidence level α and the train/validation split ratio $\delta \in (0; 1)$. Location of intermediate layers is studied in [11], where the classifiers are added to *all* layers and the exits are chosen from layers, that provide better trade-off between speed and validation accuracy. This procedure is too expensive for deep networks, so several ‘‘bottleneck’’ layers are randomly chosen. They split a CNN into sequential blocks (i.e., layers in the middle of residual or inception blocks are inappropriate), and repeat the layer selection procedure from [11].

Here the decision-making speed is increased by cascading classifiers of outputs for several intermediate layers and monitoring the output of the classifier (3) to decide whether classification can be terminated at the current branch or not. If the linear SVM mentioned in Fig. 1 is applied, then the confidence score is computed as a signed distance of an observed image to the hyperplane. If condition (3) holds, then the classifier output $c_m^*(\mathbf{x}_m)$ (2) is accepted as the final decision. Otherwise, the next step of inference (1) is executed to examine the input image at a finer representation level $m + 1$ with more detailed information [9], [20]. The multiple comparison theory is used to define thresholds given a confidence level α of the whole sequential procedure. If the

- 1: (Optional) Add fully-connected LR classifiers to all M branches and fine-tune the CNN on the given training set
- 2: **for** each intermediate layer $m \in \{1, \dots, M\}$ **do**
- 3: **for** each training instance $n \in \{1, \dots, N\}$ **do**
- 4: Feed the n -th image into a CNN and compute the outputs $\mathbf{x}_{n;m}$ at the m -th layer
- 5: **end for**
- 6: Split N instances in a stratified fashion using train/test split ratio δ to get indices $\{n_1, \dots, n_K\}$, $K = \lceil N \cdot \delta \rceil$
- 7: Train the m -th classifier, e.g., the linear one-versus-all SVM in order to maximize the squared hinge loss, using feature vectors $\{(\mathbf{x}_{n_k;m}, c(n))\}$, $k \in \{1, \dots, K\}$
- 8: Initialize a list of maximal intra-class scores $S = \square$
- 9: **for** each validation instance $n \notin \{n_1, \dots, n_K\}$ **do**
- 10: Append the maximal inter-class confidence score $\max_{c \neq c(n)} s_m^{(c)}(\mathbf{x}_{n;m})$ to the list S
- 11: **end for**
- 12: Assign the $\lfloor \alpha \cdot m(N - K)/M \rfloor$ -th largest element from S to the threshold s_m (4) using the Benjamini-Hochberg correction
- 13: Retrain the m -th classifier using all feature vectors $\{(\mathbf{x}_{n;m}, c(n))\}$, $n \in \{1, \dots, N\}$
- 14: **end for**
- 15: **return** M classifiers and their thresholds $\{s_m\}$

Fig. 1. Proposed training procedure

Require: observed image

Ensure: class label

- 1: Initialize \mathbf{z}_0 by the RGB matrix of an input image X
- 2: **for** each intermediate layer $m \in \{1, \dots, M\}$ **do**
- 3: Compute the output of the m -th layer $\mathbf{z}_m = f_{exit_m}(\mathbf{z}_{m-1}; \boldsymbol{\theta})$ (1)
- 4: Transform activations \mathbf{z}_m into feature vector \mathbf{x}_m
- 5: Predict the confidence scores $\mathbf{s}_m(\mathbf{x}_m)$ using the m -th classifier
- 6: **if** $m = M$ OR condition (3) holds **then**
- 7: **return** class label $c_m^*(\mathbf{x}_m)$ (2)
- 8: **end if**
- 9: **end for**

Fig. 2. Proposed prediction procedure

last, M -th, layer is reached, but the maximal confidence score is not higher than s_M , the output of the last classifier will be returned or such unreliable input image will be rejected.

Let us analyze the run-time complexity of the Algorithm (Fig. 2). Let t_m and ϵ_m be the inference time and validation error rate for the m -th layer. I assume that the time for additional computations in each early exit is identical and equal to t_{exit} . If the procedure is terminated at the first exit layer, its best-case running time is equal to $t_1 + t_{exit}$. The worst-case running time $t_M + (M - 1)t_{exit}$ is slightly higher than the inference time t_M in conventional CNN. The average inference time is estimated as follows:
$$\sum_{m=1}^M p_m(t_m + mt_{exit}),$$

where the conditional probability to exit at the m -th layer if it is reached is equal to $p_m = (1 - \sum_{i=1}^{m-1} p_i)(1 - \epsilon_m \cdot \alpha \cdot m/M)$. Here ϵ_M is explicitly assigned to 0. Hence, the m -th layer should be considered as an early exit only if $t_{exit} < t_m - t_{m-1}$.

The proposed approach is implemented in a special Python script (<https://github.com/HSE-asavchenko/fast-image-recognition>) based on the deep learning framework Keras with the TensorFlow backend. The sequence of functions (“keras.backend.function”) was used to implement the sequential processing during the inference. The preprocessed input (RGB) image is fed into the input of the first function in this sequence. Each function returns a tuple of: 1) the outputs of an intermediate convolutional layer \mathbf{z}_m (1); and 2) transformation \mathbf{x}_m of the first output in an average global pooling layer. The first output is fed to the input of the next function in a sequence (1). The second output is classified by an appropriate method. In addition, my script transfers the learned parameters of the linear SVM into the Keras Dense layer in order to perform the inference on the resulted network on GPU.

IV. EXPERIMENTAL RESULTS

In the experimental study I have not chosen traditional for adaptive nets MNIST and CIFAR datasets, in which the images have small resolutions (28x28, 32x32), because my most important result, performance improvements on deep pre-trained CNNs, cannot be demonstrated on these datasets. Hence, the following datasets are used:

- 1) Caltech-101 Object Category dataset, which contains 8677 images of $C = 102$ classes including distractor background category.
- 2) Caltech-256 dataset with 29780 images of $C = 257$ classes including the clutter category.
- 3) Stanford dogs dataset that contains 20580 images and $C = 120$ classes.

As all the datasets are imbalanced, I estimate the accuracy of classifying images from each category individually and computed the mean (macro) accuracy. The testing protocol for these datasets implements the following variant of the random subsampling cross-validation. Every dataset is randomly divided 3 times into the training and testing sets. I split each class independently in a stratified fashion to provide approximately identical ratio of the number of images in the training and testing set. The size R of the training set was chosen as follows: from 2 to 30 images per class for Caltech-101, 60 images per class for Caltech-256 and 110 images per class for Stanford Dogs. The train/validation split ratio δ in the Algorithm (Fig. 1) is equal to 0.5.

I used InceptionResNet v2 [6] and ResNet-152 [2] from Keras framework, that were preliminarily trained on ImageNet dataset. Classification results of globally averaged outputs of several different layers were tested in order to choose the following best layers: “block17_17_ac” and “block8_5_ac” for InceptionResNet v2, and “conv4_block36_out” for ResNet-152. In addition, I included penultimate layer traditionally used for extraction of the off-the-shelf visual features [4].

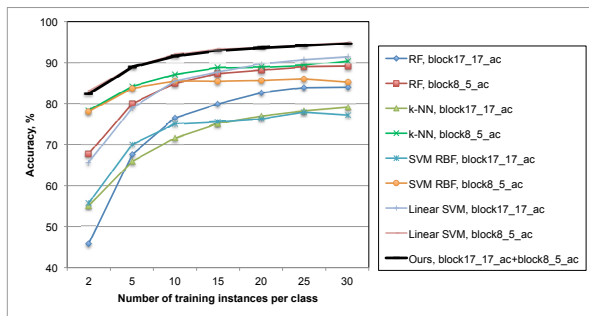


Fig. 3. Accuracy (%) depending on the number of training instances per class, Caltech-101 dataset, pre-trained InceptionResNet v2.

In addition to the pre-trained CNN, it was fine-tuned as follows. Multiple heads with LR classifier were added: one for each output of fully connected layer with softmax activations for every intermediate layer (branch). Such network is trained by using the different weights for the loss function: the closer is the layer to the input, the higher is the weight. The weights of all the layers except the new LR layers are frozen during the first 5 epochs. Next, the tuning of all weights is performed by SGD with learning rate 0.001 over 10 epochs. Here I additionally used earlier exit layers “mixed_5b” and “conv4_block1_out” for InceptionResNet v2 and ResNet-152, respectively. Though the accuracy for these layers from the *pre-trained* models are approximately equal to the random guess, the error rate for the outputs of these layers of *fine-tuned* network should be significantly lower.

One may suppose that the more layers they select, the more sufficiently is the effectiveness of adaptive CNN. As a matter of fact, more layers do not lead to inference speedup in my experiments if only small training sample is available for transfer learning. Indeed, if a classifier is attached to the lower layers of very deep CNN, the accuracy will be very small. As each new exit branch needs additional computations [11], existing adaptive nets [12], [17], [19] have only 1-3 exits, so I decided to use the same number of layers.

The globally averaged output of particular convolutional layers is classified using one of the following methods: a) one-versus-all linear SVM trained to maximize the squared hinge loss; b) SVM with radial basis function (RBF) kernel; and c) RF with 100 trees. All classifiers were implemented using Scikit-learn library.

In the first experiment I analyzed performance of the pre-trained CNNs depending on the number of training instances per class. Macro accuracy and the average time to classify one image (in ms) of the proposed approach compared to different classifiers are shown in Fig. 3 and Fig. 4, respectively. The running time is measured on the MSI laptop with CPU Intel Core i7 8750H (2.2 GHz). The thresholds for the decision function in my Algorithm (Fig. 1) were chosen to misclassify only $\alpha = 1\%$ of the validation subset.

Here the best classifier is a linear SVM from `sklearn.svm.LinearSVC` implementation based on LIBLINEAR library. In all my experiments its error

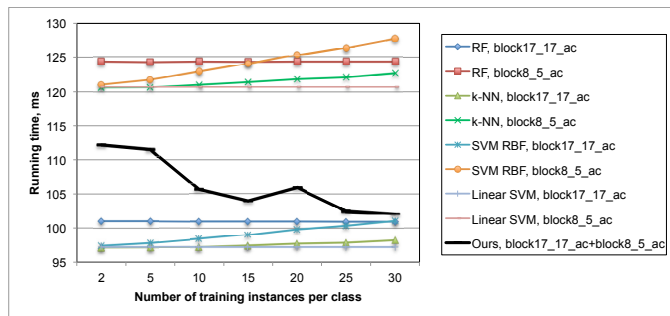


Fig. 4. The running time on CPU (ms) depending on the number of training instances per class, Caltech-101 dataset, pre-trained InceptionResNet v2.

rate is 2-5% lower than the error rate of conventional SVM (`sklearn.svm.SVC`) with linear kernel trained to optimize the hinge loss. The results of the latter classifier are approximately identical to the reported results (Fig. 3) of the SVM with RBF kernel. As the performance of the k-NN and SVM RBF is worth when compared to other classifiers, I do not present these methods in the further experiments.

The accuracy of the proposed sequential analysis is as high as the accuracy of the linear SVM for the features from one of the last layers (“block8_5_ac”). Such classification of the off-the-shelf features from very deep CNN is characterized by accuracy 94.8%, which outperformed the known state-of-the-art accuracy (93.4%) [3] for traditional protocol with 30 training samples per category. However, the proposed approach is 12-25% faster. Moreover, if the number of training images per class is increased, the overall running time of my algorithm is *decreased*. In fact, the accuracy for features from the low layer becomes higher in this case, causing an increase of the number of reliable solutions (3) obtained on this layer.

In the next experiments the inference of all CNNs was placed on a GPU (NVIDIA GeForce GTX1060). The proposed algorithm is compared with conventional classification of features from one layer and known implementation [19] of BranchyNet [12] and CDL [11] with new branches added to the same layers as for my approach. Original versions of both networks require the fine-tuned CNN with LR classifiers, so I do not present their results for the off-the-shelf features from pre-trained CNNs. The thresholds for entropy in the BranchyNet and maximal posterior probability estimated by the LR in the CDL was chosen in order to achieve appropriate accuracy. In addition, I modified the CDL to be more similar to the proposed approach (hereinafter “Ours, fixed threshold”) in such a way that: 1) concatenation of all features fed into the classifier at each branch is replaced to global average pooling; 2) the LR classifier was replaced to linear SVM trained with the squared hinge loss; and 3) the decision function (outputs of the LR classifier) is replaced to the signed distance of an observed image to the hyperplane. The classifier score in this modification is matched with a *fixed* threshold 0.06, which provides an appropriate trade-off between accuracy and speed. The results for the Caltech-101 dataset with InceptionResNet

TABLE I
PERFORMANCE ANALYSIS FOR INCEPTIONRESNET v2, CALTECH-101 (30 INSTANCES PER CLASS)

Classifier	Layers	block17_17_ac+block8_5_ac+penultimate				mixed_5b+block8_5_ac+penultimate	
		pre-trained		fine-tuned (all heads)		fine-tuned (all heads)	
		accuracy, %	time, ms	accuracy, %	time, ms	accuracy, %	time, ms
LR	first	3.87	26.12	10.75	25.71	9.58	4.17
LR	second	44.23	34.98	88.47	34.39	62.97	34.81
LR	last	39.45	39.56	84.41	41.20	91.35	39.90
RF	first	84.04	30.14	87.92	29.55	39.47	7.89
RF	second	89.02	38.85	91.42	38.14	90.87	38.38
RF	last	73.86	40.11	90.91	40.11	89.42	43.48
Linear SVM	first	91.35	26.46	92.82	25.82	58.66	4.28
Linear SVM	second	94.80	35.14	95.04	34.51	94.59	34.93
Linear SVM	last	90.75	39.72	91.04	41.44	89.15	40.02
Cascaded inference [19]	all	-	-	87.48	32.22	80.31	34.83
CDL [11]	all	-	-	87.12	31.79	81.31	35.18
Ours, fixed threshold	all	95.16	31.04	95.32	29.18	93.21	26.18
Ours, adaptive threshold	all	94.54	27.79	95.19	27.25	94.22	23.57

TABLE II
PERFORMANCE ANALYSIS FOR RESNET-152, CALTECH-101 (30 INSTANCES PER CLASS)

Classifier	Layers	conv4_block36_out+penultimate				conv4_block1_out+penultimate	
		pre-trained		fine-tuned (all heads)		fine-tuned (all heads)	
		accuracy, %	time, ms	accuracy, %	time, ms	accuracy, %	time, ms
LR	first	29.86	32.81	34.58	32.95	18.86	9.28
LR	second	55.16	38.35	91.05	38.42	92.19	29.65
RF	first	70.10	36.60	75.14	36.71	44.11	12.51
RF	second	87.43	42.04	90.74	42.09	91.76	42.00
Linear SVM	first	84.05	32.97	87.25	33.10	55.49	8.91
Linear SVM	second	91.19	38.46	92.47	38.57	93.09	38.44
Cascaded inference [19]	all	-	-	90.45	37.60	79.10	33.67
CDL [11]	all	-	-	90.95	38.37	78.32	32.31
Ours, fixed threshold	all	91.09	34.09	89.24	34.15	89.65	21.37
Ours, adaptive threshold	all	91.63	34.85	92.69	34.83	92.67	30.10

v2 and ResNet-152 are shown in Table I and Table II.

Here “LR” stands for classification by fine-tuned CNN with replacement of the last LR layer of the pre-trained network. It classifier in conventional fine-tuning is typically less accurate when compared to complex classifiers, especially when features from intermediate layers are processed. As a result, the known adaptive networks (CDL, Cascaded inference) are 10-50% less accurate when compared to the proposed approach. However, the modification of the CDL inspired by my approach gives performance, which is comparable to the best methods. However, in this case it is impossible to enable the reasonable trade-off between accuracy and running time in all experiments due to the difficult choice of a threshold. Thirdly, though classification of the outputs of the penultimate layer in the fine-tuned ResNet-152 leads to more accurate decisions when compared to the InceptionResNet, the features from the latter CNN are very rich, so that they provide much lower error rate even when compared to the classification of the output of penultimate layer of InceptionResNet. The

most important conclusion is that the proposed approach leads to the fastest decision-making if significant degradation in accuracy is not allowed. My algorithm is 1.1-1.6 and 1.4-1.7 times faster than the linear classifier of “block8_5_ac” and traditional layers, respectively.

The results for much more complex Caltech-256 dataset and more accurate CNN (InceptionResNet v2) are presented in Table III. They are approximately identical to the results of the previous experiment. The off-the-shelf features from the pre-trained CNNs are rather reliable, but sequential analysis in the fine-tuned network is slightly faster, especially if bottom layer (“mixed_5b”) is used. The proposed approach is up to 1.06-1.35 times faster when compared to the inference in complete CNN. Moreover, it is much more accurate when compared to similarly engineered adaptive networks (BranchyNet, CDL). Finally, automatic choice of threshold makes it possible to provide an excellent accuracy even in few-shot learning tasks.

In the last experiment pre-trained EfficientNet v7 (Rand-aug) [22] is studied. The results of the proposed approach and






TABLE III
PERFORMANCE ANALYSIS FOR INCEPTIONRESNET V2, CALTECH-256 (60 INSTANCES PER CLASS)

Classifier	Layers	block17_17_ac+block8_5_ac+penultimate				mixed_5b+block8_5_ac+penultimate	
		pre-trained		fine-tuned (all heads)		fine-tuned (all heads)	
		accuracy, %	time, ms	accuracy, %	time, ms	accuracy, %	time, ms
LR	first	0.91	26.29	1.85	26.34	7.82	4.61
LR	second	67.59	34.80	80.20	35.05	56.23	34.97
LR	last	62.88	42.94	77.81	40.78	78.86	40.51
RF	first	65.00	30.13	66.71	30.24	22.70	8.65
RF	second	76.40	38.51	78.43	39.00	79.32	45.14
RF	last	61.05	43.47	79.22	44.86	80.36	50.96
Linear SVM	first	81.10	26.40	80.75	26.52	41.25	5.01
Linear SVM	second	87.23	34.91	86.35	35.34	86.77	35.93
Linear SVM	penultimate	83.97	43.09	82.87	40.92	83.30	41.31
Cascaded inference [19]	all	-	-	77.73	37.75	71.06	34.47
CDL [11]	all	-	-	77.79	37.96	72.67	35.14
Ours, fixed threshold	all	84.07	29.36	86.14	33.57	86.14	32.40
Ours, adaptive threshold	all	86.95	30.09	86.10	30.21	86.20	30.31

TABLE IV
PERFORMANCE ANALYSIS FOR EFFICIENTNET V7: CALTECH-101 (30 INSTANCES PER CLASS), CALTECH-256 (60 INSTANCES PER CLASS) AND STANFORD DOGS (110 INSTANCES PER CLASS)

Classifier	Layers	Caltech-101		Caltech-256		Stanford Dogs	
		block6b_add+block6f_add+penultimate		block6f_add+block7b_add+penultimate		block7b_add+penultimate	
		accuracy, %	time, ms	accuracy, %	time, ms	accuracy, %	time, ms
RF	first	60.10	176.47	46.52	185.48	86.75	205.87
RF	last	91.40	215.18	82.90	215.41	92.80	215.06
Linear SVM	first	82.12	172.72	71.18	181.76	86.82	198.03
Linear SVM	last	95.24	211.63	92.16	211.66	93.41	207.40
Ours, fixed threshold	all	92.07	178.71	83.42	191.46	90.53	200.27
Ours, adaptive threshold	all	94.49	181.91	91.13	197.82	92.99	202.18

TABLE V
EXAMPLE RESULTS OF IMAGE RECOGNITION WITH THE PROPOSED APPROACH, EFFICIENTNET V7, CALTECH-101 (30 INSTANCES PER CLASS)

						
Prediction at "block_6b_add" ($s_1 = 1.83$)	Confidence	1.14	0.81	0.97	0.34	1.64
	Class	scorpion	cannon	beaver	wrench	elephant
Prediction at "block_6f_add" ($s_2 = 1.36$)	Confidence	1.64	1.66	2.03	18.38	1.52
	Class	crab	anchor	dolphin	mayfly	kangaroo

the best classifiers for all datasets are summarized in Table IV. This CNN leads to the best known accuracy for all three datasets. For example, its accuracy is higher than 92.2% of the Weakly Supervised Data Augmentation Network [23] for the Stanford Dogs datasets, though the decision-making time of the latter approach is higher due to the need for object localization and refinement. In any case proposed sequential analysis leads to faster decision-making when compared to inference in complete CNN. Moreover, the difference in accuracy is specified by the confidence level 1%.

Let me demonstrate the qualitative results of the proposed approach for EfficientNet v7 and Caltech-101 (Table V), where the inference is terminated after the second exit, and correct class is predicted. Though the class predicted at the first exit is incorrect, the proposed algorithm makes it possible to proceed the inference, because the maximal confidence score in this case does not exceed threshold $s_1 = 1.83$. One can notice that the confidence 1.64 in the last column (kangaroo) for the first exit is higher than the confidence 1.52 for the second exit. However, correct class is returned, because confidences

are matched with different thresholds and $s_1 > s_2$. Indeed, it is more difficult to make a reliable decision taking features at the earlier layers of CNN.

V. CONCLUSION

In this paper the novel image recognition Algorithm (Fig. 2) has been proposed, which makes an inference in deep CNN faster based on sequential analysis of the outputs of intermediate layers. The thresholds for confidence scores are estimated automatically in my approach based on the given training set and the confidence level of the whole recognition process by using my implementation of the Benjamini-Hochberg test [21]. The proposed approach is less limited than existing adaptive CNNs [11], [19]: the inference speed of any pre-trained and fine-tuned CNN is increased even in few-shot learning (Fig. 4) with domain adaptation. In contrast to many existing methods [16], [18], [19] that were developed for ResNets, my algorithm can be applied with any CNN architecture (Table IV). The network is similar to BranchyNet [11], [12], though I introduced additional global average pooling and L_2 -norm layers in each branch. The most important difference is the training procedure (Fig. 1): CNN is not trained in end-to-end fashion and any shallow classifier is appropriate instead of Logistic Regression in BranchyNet. My method provides 1.06-1.7-times speed up over existing techniques on both CPU and GPU. One its limitations is the impossibility to exit from one of the parallel convolutional layers, e.g, one of the intermediate layer in the Inception block before merge point.

This study clearly highlights the main restriction of the practical application of all adaptive neural networks [17], [19] for few-shot learning [1], namely, very low recognition accuracy for the features extracted from early layers. The gain in accuracy is significant even if the feature-extraction layer becomes slightly deeper. For example, rather deep layer “conv4_block36_out” in ResNet-152 is close to the last layer, so that the inference time is only 6 ms less when compared to the inference time in all CNN (Table II). However, the accuracy for the features from early layer of the pre-trained network is 7-26% lower. In order to obtain accuracy comparable with a whole network, one should not terminate the inference even on such deep layers. Moreover, conventional fine-tuning with multiple heads cannot solve the issue as the difference in accuracy remains no less than 5% in this particular case (column 5 in Table II). The same is true for EfficientNet and Stanford dogs dataset (Tables IV). Such low accuracy for deep intermediate layers significantly restricts the potential of dynamic and adaptive inference not only in my algorithm but in any similarly engineered approach. That is the main reason why almost all methods from the literature survey (Section II) are tested only on very-large datasets (ImageNet, MNIST, CIFAR) with CNN trained from scratch. Hence, in future it is necessary to study the best ways to fine-tune CNN so that features at early layers will be reach enough.

REFERENCES

- [1] S. Qiao, C. Liu, W. Shen, and A. L. Yuille, “Few-shot image recognition by predicting parameters from activations,” in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018, pp. 7229–7238.
- [2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
- [3] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *International Conference on Learning Representations*, 2015.
- [4] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, “CNN features off-the-shelf: an astounding baseline for recognition,” in *Proceedings of Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE Computer Society, 2014, pp. 806–813.
- [5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [6] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, Inception-ResNet and the impact of residual connections on learning,” in *Proceedings of the 31 AAAI Conference on Artificial Intelligence*, 2017, pp. 4278–4284.
- [7] A. V. Savchenko, “Probabilistic neural network with complex exponential activation functions in image recognition,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 2, pp. 651–660, 2020.
- [8] A. Wald, *Sequential Analysis*. New York: Dover Publications, 2013.
- [9] A. V. Savchenko, “Sequential three-way decisions in multi-category image recognition with deep features based on distance factor,” *Information Sciences*, vol. 489, pp. 18–36, 2019.
- [10] R. Teja Mullapudi, W. R. Mark, N. Shazeer, and K. Fatahalian, “HydraNets: Specialized dynamic architectures for efficient inference,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8080–8089.
- [11] P. Panda, A. Sengupta, and K. Roy, “Conditional deep learning for energy-efficient and enhanced pattern recognition,” in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2016, pp. 475–480.
- [12] S. Teerapittayanon, B. McDanel, and H. Kung, “BranchyNet: Fast inference via early exiting from deep neural networks,” in *Proceedings of the IEEE 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.
- [13] J. Hochberg and A. Tamhane, *Multiple comparison procedures*, 2009.
- [14] D. Mittal, S. Bhardwaj, M. M. Khapra, and B. Ravindran, “Recovering from random pruning: On the plasticity of deep convolutional neural networks,” in *Proceedings of Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 848–857.
- [15] A. M. Grachev, D. I. Ignatov, and A. V. Savchenko, “Neural networks compression for language modeling,” in *Proceedings of International Conference on Pattern Recognition and Machine Intelligence (PRMI)*. Springer, 2017, pp. 351–357.
- [16] M. Figurov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, “Spatially adaptive computation time for residual networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 1039–1048.
- [17] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, “Adaptive neural networks for efficient inference,” in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, vol. 70, 2017, pp. 527–536.
- [18] A. Veit and S. Belongie, “Convolutional networks with adaptive inference graphs,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 3–18.
- [19] K. Berestizshevsky and G. Even, “Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence,” in *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. Springer, 2019, pp. 306–320.
- [20] A. V. Savchenko, “Granular computing and sequential analysis of deep embeddings in fast still-to-video face recognition,” in *Proceedings of the 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. IEEE, 2018, pp. 515–520.
- [21] Y. Benjamini and Y. Hochberg, “Controlling the false discovery rate: a practical and powerful approach to multiple testing,” *Journal of the Royal statistical society: series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.
- [22] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” *arXiv preprint arXiv:1905.11946*, 2019.
- [23] T. Hu and H. Qi, “See better before looking closer: Weakly supervised data augmentation network for fine-grained visual classification,” *arXiv preprint arXiv:1901.09891*, 2019.