Incremental Learning of Random Forests for Large-Scale Image Classification

Marko Ristin, Matthieu Guillaumin, Juergen Gall, *Member, IEEE* and Luc Van Gool, *Member, IEEE*

Abstract—Large image datasets such as *ImageNet* or open-ended photo websites like *Flickr* are revealing new challenges to image classification that were not apparent in smaller, fixed sets. In particular, the efficient handling of dynamically growing datasets, where not only the amount of training data but also the number of classes increases over time, is a relatively unexplored problem. In this challenging setting, we study how two variants of Random Forests (RF) perform under four strategies to incorporate new classes while avoiding to retrain the RFs from scratch. The various strategies account for different trade-offs between classification accuracy and computational efficiency. In our extensive experiments, we show that both RF variants, one based on Nearest Class Mean classifiers and the other on SVMs, outperform conventional RFs and are well suited for incrementally learning new classes. In particular, we show that RFs initially trained with just 10 classes can be extended to 1000 classes with an acceptable loss of accuracy compared to training from the full data and with great computational savings compared to retraining for each new batch of classes.

Index Terms—Incremental learning, random forests, large-scale image classification

1 Introduction

With the ease of capturing and sharing pictures, the digital representation of our rich visual world grows and so does the need for efficient image classification algorithms that scale with the vast digitized visual knowledge. In fact, there has been a shift towards large-scale image classification problems in the last few years. Datasets with fewer images and classes, such as PASCAL VOC [1], give way to more complex and voluminous datasets, such as "ImageNet" [2] or "80 Million Tiny Images" [3], which comprise millions of images and thousands of classes. Larger datasets do not only pose quantitative problems that need to be addressed, they also introduce challenges of new quality: the classes become finer and are semantically and visually more similar. For example, while conventional one-vs-all classifiers performed well on small-scale datasets, they are now outperformed on large-scale datasets both in accuracy and in training time by nearest neighbor or multiclass approaches [4], [5], [6], [7].

Offline classification methods, such as multiclass SVMs [4], assume a *static* setting where the number of training images is fixed as well as the number of classes that a model can handle. However, the virtual representation, for example due to the rapid expansion of the shared visual data in social networks, is very dynamic. It is not only the number of the images that increases, but also the semantics becomes more complex with the emergence of new semantic classes. To add a single class to an existing system, static approaches need to retrain the whole model, which becomes too expensive for large datasets.

In this work, we consider a *dynamic* large-scale scenario where the number of classes as well as the number of images gradually increase and reach large numbers, *i.e.*, thousand of classes and million of images. This scenario is relevant for many applications where the number of classes is a-priori unknown. During the development, one typically focuses on a few classes that are most relevant. Over time the demands of the users evolve, including the classification of additional classes [8]. For applications that involve open-ended learning, the number of classes even grows continuously. Although one-vs-all classifiers are already a basic framework for incrementally learning a dynamically growing number of image classes where adding a new class is achieved by training a new one-vs-all classifier, the computational cost of training a new classifier is high [5]. There are only a few recent approaches [6], [9] that explicitly address the problem of incrementally learning new image classes. As the new data is collected over time, the classifiers evolve and adapt to the new situation without the need of retraining from scratch. Fig. 1 gives an illustration of incremental learning of new classes as it is considered in this work. The multiclass classifier is first trained with training data for a certain number of classes, which results in an initial model that can successfully recognize the initial set of classes. Additional classes can be added at any point by providing training data for novel classes. The model is then updated and classifies the initial and new classes.

In [6] a discriminative metric is learned for Nearest Class Mean classification on the initial set of classes and new classes are added using their feature means. The approach, however, assumes that the number of initial classes is relatively large. An alternative multiclass incremental approach based on least-squares SVM has been proposed in [9] where for each class a decision hyperplane is learned. Every time a class is added, the whole set of the hyperplanes is updated, which is expensive as the number of classes grow. In this work, we investigate random forests (RF) [10] for the task of learning incrementally new classes. RFs are intrinsically multiclass and hierarchical

- *Marko Ristin, Matthieu Guillaumin and Luc Van Gool are with the Computer Vision Laboratory, ETH Zurich, Switzerland. E-mail: {ristin, guillaumin, vangool}@vision.ee.ethz.ch*
- *Juergen Gall is with the Computer Vision Group, Univ. of Bonn, Germany. E-mail: gall@iai.uni-bonn.de*

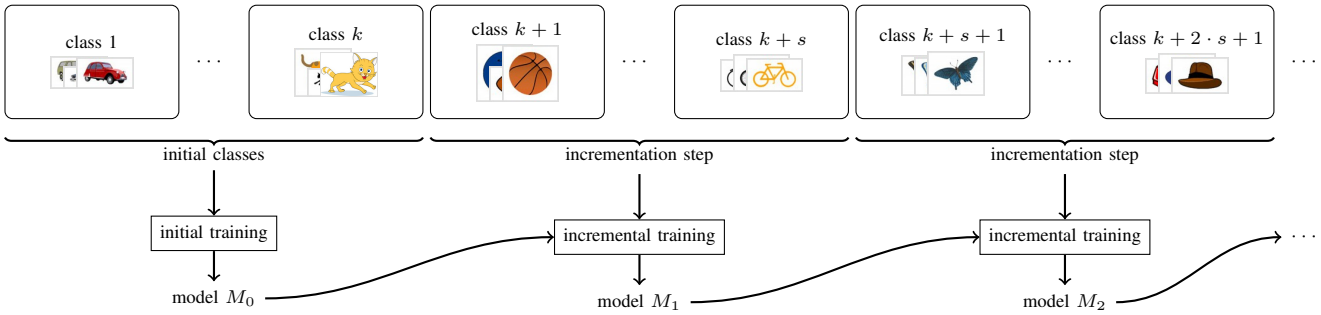


Fig. 1: The training starts with k initial classes and the corresponding initial model M_0 that can classify these k classes. When a batch of s new classes arrives, the model is incremented to M_1 which is now able to discriminate $k+s$ classes. The incremental learning scenario is open-ended and training continues as new classes become available.

classifiers, properties which make them suitable for large-scale classification problems. Since each tree imposes a hierarchy on the feature space, the changes at the deeper levels of the tree are more local in the feature space and depend on less data. This allows us to update the classifiers very efficiently. In this work, we study two variants of RFs with different classifiers as their building blocks.

The first one, inspired by Nearest Class Mean (NCM) classification [6] and introduced in [11], implements the decisions at each node based on the Voronoi cells formed by a small random subset of the class means observed at that node, the *centroids*. The centroids partition the feature space and assign a sample either to the left or the right subtree. We refer to these forests as *Nearest Class Mean Forests* (NCMF). Their application is depicted in Fig. 2. As second RF variant, we examine linear SVMs as binary classifiers at nodes. To integrate SVMs to RFs, we follow the approach proposed in [12] and denote them as *SVM Forests* (SVMF). While the method proposed in [12] focuses on offline, fine-grained classification, our aim is to examine how SVMFs behave in the setting of large-scale image classification and incremental learning. For both RF variants, we propose and evaluate efficient updating strategies to integrate new classes so as to maintain high accuracy at the lowest possible cost for training. Our experiments show that both RF variants outperform conventional RFs and match state-of-the-art classifiers on the challenging large-scale ImageNet dataset [13]. In the context of incrementally learning new classes, NCMFs and SVMFs outperform [6], [9].

A preliminary version of this work appeared in [11]. The present work extends incremental learning to SVMFs and proposes a novel scheme for updating nodes in a tree based on the classification quality of the subtrees. The experimental evaluation has been substantially extended and includes the impact of the number of initial classes, the order of incrementally added classes, the batch size of added classes, and the dimensionality of feature space.

The paper is organized as follows. Related work is discussed in Section 2. Section 3 introduces the variants of RFs based on NCM and SVM classifiers. Approaches to train them incrementally are discussed in Section 4. Section 5 presents the experimental evaluation and comparison to other approaches on the large-scale ImageNet dataset [13].

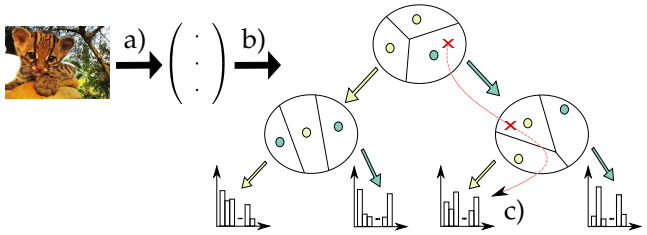


Fig. 2: Classification of an image (illustrated by the red cross) by a single tree of Nearest Class Mean forest (NCMF). (a) The feature vector is extracted, (b) the image is assigned to the closest centroid (colors indicate further direction), (c) the image is assigned the class probability found at the leaf.

2 Related Work

Image classification on large datasets is a challenging problem [14], [15], with issues that are not apparent in smaller ones [5]. To address these challenges, the state of the art consists in carefully designing a deep Convolutional Network [16] or using advanced high-dimensional features, such as Fisher Vectors [15], [17]. Below we discuss various work related to dealing with large number of classes, large amount of data and various ways in which data gradually become available.

Hierarchical classification. When the number of classes is large, various authors propose to exploit a hierarchy either over classes or over input space to improve classification performance or time complexity of training and testing. An explicit class hierarchy is used in [18] in order to predict not only fine-grained classes at the lowest level of the hierarchy, but falls back to higher coarser levels when fine-grained classification is uncertain. In this scenario, a trade-off between accuracy and specificity determines the output of the classifiers. Instead of performing one-vs-all classification, classifiers can also be stacked in a hierarchy as a decision tree. At each node a sample is compared to a small number of SVM decision boundaries and assigned to one of the child nodes, thus leading to a logarithmic rather than linear complexity [19]. Several works have built on this idea and proposed alternative parameter training methods [20], [21], [22].

Random Forests [10] are the archetype of hierarchical classifiers. They are ensemble classifiers composed of Random Decision Trees, which are independently trained on random

subsets of the data and then combined by averaging their classification scores. The decision trees are themselves randomized in the selection of weak classifiers at each node of the hierarchy. When the trees are balanced, the RFs are very efficient, as the time complexity of classification is logarithmic in the number of nodes. RFs have been successfully used in many tasks such as image classification [23], vocabulary generation through vocabulary trees [24], as feature representation for image segmentation [25], object detection [26], and fine-grained image classification [12], [27]. Our RF variants employ either Near Class Mean classifiers [6] (NCM Forest) or linear SVMs (SVM Forest) as node classifiers. This is in contrast to axis-aligned tests proposed in conventional RFs [10], random linear splitting functions proposed in [23] and unsupervised cluster centers that disregard class information proposed in [24]. SVM Forests have been proposed in [12] for fine-grained classification where each node of the trees classifies a single or a pair of rectangular image regions by a binary SVM, where each class is randomly assigned to one of the binary classes. Although the SVM Forests slightly differ in our context (the trees do not combine SVMs for various image regions, and the nodes classify entire images), we show how SVM Forests can be incrementally learned.

Big data. To efficiently handle massive amounts of data, there has been a wide development of *online learning* methods, such as stochastic gradient descent [14], [15]. These methods iteratively learn from a limited batch of data instances at a time and hence remain frugal in terms of memory. The main assumption in online learning is that samples are provided in a uniformly random sequence, and, as a matter of fact, most methods start by randomly permuting the data [28]. This *i.i.d.* assumption allows authors to ignore typical problems of sequential data such as stochastic drift and birth or death of classes. As a consequence, they typically assume that the classes are known and fixed beforehand.

In particular, online learning has been studied in the context of RFs. This is typically done by extending the trees as more samples become available. The authors of [29] propose to initialize the trees in an extremely random fashion. Statistics at the leaves are then updated as the new samples arrive. Various methods convert leaves to a splitting node and proceed with the training recursively. In [30], an analytically derived upper bound is computed in order to select leaves for further training. In [31], a simple alternative with a fixed threshold on the number of samples is used, and [32] shows empirically that such a threshold suffices to select leaves for recursive training and obtain good classification accuracy. In [33] the splitting nodes are not trained directly by optimizing an objective function, but they are sampled instead from a Mondrian process (*i.e.* a distribution over KD-trees), which allows efficient incremental learning. In contrast to [31], [32], the Mondrian forests not only update the leaves, but also introduce new splitting function within the tree structure. In the context of unsupervised vocabulary trees, the number of samples has also been proposed in [34] as a criterion to select which leaves need to be refined in order to adapt the forest to newly observed data. [31] also discards trees based on the out-of-bag error in order to progressively adapt to the new data and forget the old one. In [35], a Hough Forest is trained by incrementally growing the leaves at each step with user feedback, in an *active learning* scenario. Like most existing classifiers (*e.g.*, SVM or [12]), active or online learning methods do not consider observing new classes in the data stream, and are typically not straightforward to adapt to this scenario.

Transfer learning. The large-scale nature of the datasets such as *ImageNet* implies uneven distribution of training samples across the classes [2]. Some classes may lack sufficient data for robust learning. Transfer learning can be used to address this problem by sharing knowledge between related classes. In [36], the decision hyperplane of a class is regularized towards a subset of the decision hyperplanes of the other classes. For a large dataset with few annotated objects, the localization [37] and segmentation [38] of classes can be propagated by sharing appearance, location distribution and the context of the already labeled objects towards related classes as defined by the class hierarchy of *ImageNet*. In [39], knowledge is being transfered among classes using similarities based on attributes, textual web queries and semantic relatedness. Although the hierarchical nature of RFs implicitly entails knowledge sharing in higher nodes, our study focuses on the efficient integration of new classes rather than trying to model knowledge sharing explicitly so as to reduce the amount of training data needed for any particular class.

Incremental learning, as defined in [6], [9], [40] and our previous work [11], is the scenario where training data is not provided uniformly at random, but where classes are provided in sequence. Typically, a few classes are available to start with and new classes are added afterwards (*cf.* Fig. 1), in an open-ended fashion. In such a setting, the authors of [6] propose to train a discriminative metric for Nearest Class Mean classification on the initial set of classes and then integrate new classes simply using their data means. This leads to a near-zero cost for integrating new classes and good performance is reported provided that enough initial training data is present to learn a robust metric. In their work [6], the authors initialized the training with as many as 800 classes and experimented with adding 200 new ones. As we show in our experiments in Section 5, such a system struggles when the number of initial classes is relatively small and the amount of new classes increases since the initial metric remains fixed. In contrast, our RFs are designed so that they can be gracefully updated. As we show, the forests can be initially trained with as few as 10 classes and the complexity of their structure can be easily increased, if necessary, in order to successfully integrate any number of new classes.

An alternative multiclass incremental approach based on least-squares SVM has been proposed in [9]. Building on top of the transfer learning method introduced in [36], the model for a new class, *i.e.*, a decision hyperplane, is constrained to differ as little as possible from a subset of previously trained models. Each incremental step is formulated as an optimization problem where the whole set of the hyperplanes is updated, which is potentially expensive as the number of classes grows. In our case, the update is significantly more efficient, as the update of nodes in a decision tree only depends on a fraction of the data and of the classes. Furthermore, each independent

tree can be learned and updated in parallel. In the case of NCMF, the weak classifiers themselves allow for update with little computational effort.

Besides large-scale image classification, which is the focus of this work, incremental learning has been recently also applied for activity modeling in streaming videos [40]. The authors of [40] introduce a system based on the ensemble of SVMs and use active user responses to annotate samples of the new classes. Once there are enough samples, the old and the new models are finally combined by adjusting the model weights accordingly. Since the old models do not change, the method suffers from similar issues as the method described in [6]. With the increasing number of classes, the old models will eventually generalize poorly to the new data. Our forests, in contrast, are specifically designed to address the issue of changing data.

Compared to previous work, our experiments also put much more strain on the systems so as to push the limits of incremental learning beyond what has been studied before [6], [9]. Unlike [6], where 800 classes are available at initialization, we start with a much smaller number of classes (10 or 20) and study the influence of the order in which classes are added. The experimental evaluation used in [9] considered only a sequence of up to 48 classes, where only a single class had to be integrated at a time. Instead, we perform the evaluation with up to 1000 classes and batches of one or more classes.

3 Random Forests

Random Forests (RF) [10] consist of T randomized decision trees. Each tree and each node at the same depth is trained and classifies independently, which makes RFs very efficient at training and test time. The trees operate on data instances given as d-dimensional vectors $\vec{x} \in \mathbb{R}^d$. At each node n of each tree, the training data S^n arriving at that node is divided by a *splitting function* $f^n : \mathbb{R}^d \mapsto \{-1, 1\}$ into two subsets $S^n_{f^n=-1}$ and $S^n_{f^n=1}$. The performance of RFs heavily depends on the choice of splitting functions, and commonly used ones are axis-aligned [10] or linear splitting functions [23]. For training, a random set of splitting functions $\mathcal{F}^n$ is generated and the best one, f^n, is selected according to the information gain U:

$$\begin{aligned} f^n &= \operatorname*{argmax}_{f \in \mathcal{F}^n} U(f) \\ U(f) &= H\left(S^n\right) - \sum_{i \in \{-1,1\}} \frac{|S^n_{f=i}|}{|S^n|} H(S^n_{f=i}) \\ H(S^n) &= -\sum_{\kappa \in \mathcal{K}} P(\kappa|S^n) \ln P(\kappa|S^n) \end{aligned} \tag{1}$$

where H denotes class entropy and $P(\kappa|S^n)$ the fraction of S^n belonging to the class κ. The left and right children nodes are then trained on $S^n_{f^n=-1}$ and $S^n_{f^n=1}$, respectively, and the training continues recursively.

Given a pre-defined constant μ, the splitting stops when no $f \in \mathcal{F}^n$ satisfies $\left|S^n_{f=-1}\right| > \mu$ and $\left|S^n_{f=1}\right| > \mu$. At each leaf node l of a tree t, we store the distribution over classes observed during the training, *i.e.*, $P^t_l(\kappa)$. For classification, the feature vector of the image is extracted and passed through each tree until it arrives at leaf $l(\vec{x})$. The class probabilities of all trees are averaged and classification is defined by:

$$\kappa^*(\vec{x}) = \operatorname*{argmax}_{\kappa} \frac{1}{T} \sum_t P^t_{l(\vec{x})}(\kappa)\,. \tag{2}$$

In the following, we describe how different classifiers can be used as splitting functions f in a random forest framework. Namely, we look into classification based on support vector machines (SVM) and a nearest class mean classifier (NCM).

3.1 Linear support vector machine (SVM)

A linear SVM classifies images represented by a d-dimensional feature vector $\vec{x} \in \mathbb{R}^d$ using a decision hyperplane defined by its normal vector $\vec{w} \in \mathbb{R}^d$. The samples are classified in two classes with label $y \in \{-1, 1\}$ depending on which side of the hyperplane they reside:

$$y(\vec{x}) = \operatorname{sgn}\langle \vec{w}, \vec{x} \rangle, \tag{3}$$

where sgn is the sign function and $\langle \cdot, \cdot \rangle$ is the inner-product.

Using a set of training images $\{x_i\}$ and their corresponding labels $\{y_i\}$, the hyperplane $\vec{w}$ is set by solving the following convex optimization problem.

$$\operatorname*{argmin}_{\vec{w}} \frac{\lambda}{2} \|\vec{w}\|^2 + \frac{1}{|S|} \sum_{i=1}^{|S|} \max\left(0, 1 - y_i \cdot \langle \vec{w}, \vec{x}_i \rangle\right), \tag{4}$$

where λ is a cross-validated constant and $|S|$ the number of training samples. Equation (4) is optimized by stochastic gradient descent [28].

3.2 Combining SVM and Random Forests

As binary classifiers, SVMs are well suited to serve as splitting functions in RFs. Each node n of the forest is associated with its own hyperplane $\vec{w_n}$ and uses $f^n(\vec{x}) = \operatorname{sgn}\langle \vec{w_n}, \vec{x} \rangle$ to decide whether a sample is passed to the left child node or to the right child node. Such a system is described in [12], where randomness comes from window extraction in the images. Here, we adopt a slightly different approach.

To train a node n, all instances $\{x_i\}$ of a class κ observed at n are assigned the same random meta-label $y_i = y_\kappa \in \{-1, 1\}$. An SVM is then trained by solving (4) with all the training instances reaching the node n and corresponding meta-labels to learn a single splitting function f. The random assignments of classes to meta-labels mitigate class imbalance problems and gives us a pool of splitting functions from which we sample a fixed number (20 in our case) and pick the optimal one, f^n, following (1).

3.3 Nearest Class Mean classifier (NCM)

Nearest class mean classifiers (NCM) have shown promising results in large-scale image classification (*cf.* Section 5, [6]). Based on a simple 1-nearest neighbor classifier, NCM assigns to a sample the label of the class with the closest mean. Since class means are very efficiently computed, the training of NCM has low computational cost. Below we provide a more formal definition of NCM classification.

With an image I being represented by a d-dimensional feature vector $\vec{x} \in \mathbb{R}^d$, we first compute the class centroid c_κ for each class $\kappa \in \mathcal{K}$:

$$c_\kappa = \frac{1}{|\mathcal{I}_\kappa|} \sum_{i \in \mathcal{I}_\kappa} \vec{x}_i, \tag{5}$$

where $\mathcal{I}_\kappa$ is the set of images labeled with class κ. Since there is a centroid for each class, the set of centroids $\mathcal{C} = \{c_\kappa\}$ has cardinality $|\mathcal{C}| = |\mathcal{K}|$.

NCM classification of an image I represented by $\vec{x}$ is then formulated as searching for the closest centroid in feature space:

$$\kappa^*(\vec{x}) = \underset{\kappa \in \mathcal{K}}{\operatorname{argmin}} \|\vec{x} - c_\kappa\| . \tag{6}$$

Without additional refinements, the classification of one image implies $|\mathcal{K}|$ comparisons in $\mathbb{R}^d$. A crucial contribution of [6] to improve classification accuracy is to replace the Euclidean distance in (6) with a low-rank Mahalanobis distance optimized on training data.

3.4 Combining NCM and Random Forests

In this section, we propose to use a variation of NCM classifiers as splitting functions and we name the resulting forests *NCM Forests*. To use them as node classifiers, NCM classifiers are modified in two aspects. First, at any particular node, only a fraction of the classes will be used, hence speeding up (6) and obtaining weak classifiers. Second, the multiclass output of NCM is translated into a binary output (left vs. right child) by assigning the classes to either side.

The benefit of such an NCM Forest compared to NCM classification is that only a few comparisons are required at each node, implicitly encoding a hierarchical structure of classes. This results in an improved classifier accuracy that alleviates the need for expensive metric learning. Compared to the most common variants of Random Forests, NCM Forests also offer non-linear classification at the node level.

More specifically, we perform the following procedure to train a node n with its corresponding data S^n. First, we denote by $\mathcal{K}^n$ a random subset of the classes observed in S^n, and by S^n_κ the subset of S^n of class $\kappa \in \mathcal{K}^n$. Then, for each $\kappa \in \mathcal{K}^n$, we compute the corresponding centroids as in Section 3.3:

$$c^n_\kappa = \frac{1}{|S^n_\kappa|} \sum_{i \in S^n_\kappa} \vec{x}_i. \tag{7}$$

Then, each centroid c^n_κ is assigned randomly to a left or right child node symbolized by a binary value $e_\kappa \in \{-1, 1\}$. The corresponding splitting function f is then defined by:

$$f(\vec{x}) = e_{\kappa^*(\vec{x})} \quad \text{where} \quad \kappa^*(\vec{x}) = \underset{\kappa \in \mathcal{K}^n}{\operatorname{argmin}} \|\vec{x} - c^n_\kappa\| . \tag{8}$$

We use (1) to select the optimal f^n from the pool of splitting functions corresponding to random centroids assignments $\{e_\kappa\}$. We do not optimize over random choices of $\mathcal{K}^n$ for two reasons. First, this would force us to compute all class means at all nodes. Second, we can exploit reservoir sampling to add new classes to $\mathcal{K}^n$ in a principled manner. With $|\mathcal{K}^n| \ll |\mathcal{K}|$, the forests will perform a low number of the comparisons.

Our experiments in Section 5 show that the proposed NCM splitting functions outperform standard ones for the task of large-scale image classification. We also show that the classification accuracy of NCMF without metric learning is comparable to the performance of NCM with metric learning (MET+NCM), but the training of the RF is intrinsically parallelizable and thus faster than MET+NCM. Moreover, the main benefit of the approach is the ease of incrementally adding new classes to an already trained multiclass classifier as we discuss in the next section. Classification using a tree of an NCM Forest is illustrated in Fig. 2.

4 Strategies for Incremental Learning

As discussed in Section 2, online learning of Random Forests has been studied for vision applications such as tracking, object detection, or segmentation [29], [31], [41], [35]. However, these works focus on problems where the number of classes is known beforehand. In this work, we focus on incrementally adding new classes to the forest in the context of large-scale image classification. Without a proper incremental learning mechanism, a multiclass classifier would need to be retrained from scratch every time a new class is added. This makes it potentially very expensive to add new classes, especially as the dataset grows. Below, we devise four strategies for incremental learning applicable for both NCM Forests (NCMF) and SVM Forests (SVMF). These approaches exploit the hierarchical nature of the forests for efficient updating and present gradual trade-offs between the computational efficiency of the update and the accuracy of the resulting classifier.

4.1 Update leaf statistics (ULS)

Assuming that a multiclass RF has been already trained for the set $\mathcal{K}$ of classes, a new class κ' is added by passing the training images of the new class through the trees and updating the class probabilities $P_l(\kappa)$ stored at the leaves. Notably, this approach updates only the leaves but does not change the splitting functions or size of the trees. Since the structure of the tree does not change, it is only applicable to situations where the tree is already complex enough to cope with new data. Therefore, it needs enough training data at the initialization that cover well the distribution of all the data. Otherwise, the splitting functions overfit to the initial training data and result in poor performance since the tree does not produce a meaningful hierarchy for the new data. While [29] use extremely randomized trees as initialization point for a tracking application, we train our initial forest on the initially available classes and observe how the approach behaves in image classification.

4.2 Incrementally grow tree (IGT)

Unlike ULS, Incrementally Grow Tree (IGT) continues growing the trees if enough samples of the new class arrive at a leaf. The previously learned splitting functions remain unchanged, but new splitting nodes can be added. While the newly added splitting functions were trained on samples from $\mathcal{K} \cup \kappa'$, the old splitting functions are based on samples from $\mathcal{K}$. The approach

presented in [35] refined the leaves based on the user feedback. Hence it can be assumed that the new training samples are drawn from a finer distribution than the coarse initial one. In our case, we consider a scenario where new classes appear in random order and test how the approach behaves when the new data is not necessarily related to the previously observed training samples.

4.3 Retrain subtree (RTST)

In contrast to ULS and IGT, which do not converge to a forest trained on $\mathcal{K} \cup \kappa'$ classes since the tree structure learned for $\mathcal{K}$ classes is not changed, RTST updates also previously learned splitting functions. To this end, a subset of nodes in the trees trained on $\mathcal{K}$ classes are marked and converted into leaves by removing all of their children. By storing references to the training samples in leaves, it is efficient to reuse the training of the $\mathcal{K}$ classes together with the new classes for the newly created leaf node and update statistics. As for IGT, the cut trees are then grown again, which, in essence, corresponds to retraining subtrees with samples from all classes. The distribution $p(n)$ which defines the probability that a node n be marked for retraining will be further explained in Section 4.5.

To control the amount of retraining, only a fraction $\pi \in [0, 1]$ of the subtrees is selected by randomly sampling without replacement. If $\pi = 1$, the trees are completely retrained and the training is not incremental anymore. For $\pi = 0$, RTST is the same as IGT.

4.4 Reuse subtree (RUST)

While RTST retrains subtrees entirely, we also propose a fourth approach that reuses subtrees to reduce the training time. Instead of marking full subtrees, RUST updates single splitting nodes. The nodes are selected for update as in RTST. The incremental training is then performed breadth-first.

Since updating the splitting function f^n might result in a redistribution of the training samples from the classes $\mathcal{K}$ within the subtree of the node, the samples with $f'^n(\vec{x}) \neq f^n(\vec{x})$ are removed from the leaves and passed through the subtree again, where f'^n is the splitting function after the update. As this might create leaves without samples, the trees are cut such that each leaf contains a minimum number μ of samples. The impact of π and μ is evaluated in Section 5.

While ULS, IGT and RTST are general approaches that work with any type of splitting functions, RUST needs to be tailored to NCM Forest (NCMF) and SVM Forest (SVMF).

4.4.1 *RUST for NCMF*

Each splitting node n already stores a function f^n where the $|\mathcal{K}^n|$ centroids have been sampled from $\mathcal{K}$. The splitting functions for $\mathcal{K} \cup \kappa'$ classes, however, would have been sampled from centroids from the larger set of classes. We therefore use reservoir sampling [42] to decide if the centroid $c^n_{\kappa'}$ is ignored, added or replaces an element of $\mathcal{K}^n$ to form $\mathcal{K}'^n$, in which case the splitting function is updated as well:

$$f'^n(\vec{x}) = e^n_{\kappa'(\vec{x})} \text{ with } \kappa'(\vec{x}) = \underset{\kappa \in \mathcal{K}'^n}{\operatorname{argmin}} \|\vec{x} - c^n_\kappa\|, \quad (9)$$

where $e^n_{\kappa'} \in \{-1, 1\}$ is selected based on (1).

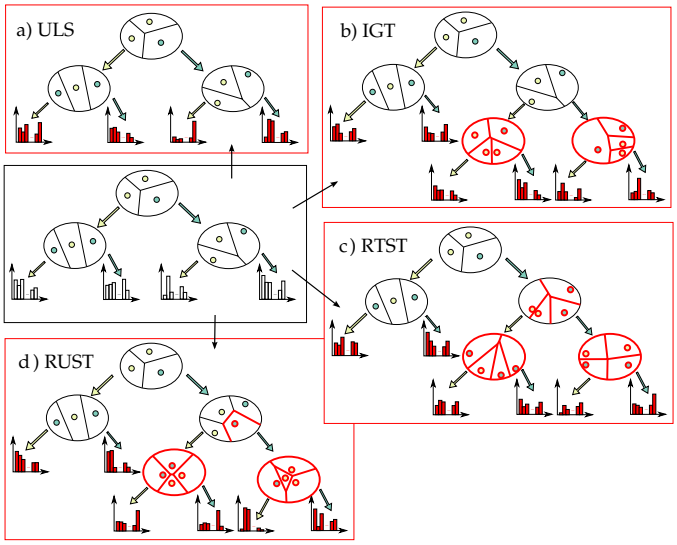


Fig. 3: Illustration of our incremental approaches with NCM forest: **a)** Update leaf statistics (ULS), **b)** Incrementally grow tree (IGT), **c)** Retrain subtree (RTST), **d)** Reuse subtree (RUST). The colors of the centroids (yellow, green) indicate the directions associated with the Voronoi cells. The elements marked in red are modifications to the structure of the tree. In c), the centroids of the root's right child are re-computed, while in d) only a new centroid is added.

4.4.2 *RUST for SVMF*

Each splitting node n stores a function $f^n(\vec{x}) = \text{sgn}\langle \vec{w_n}, \vec{x} \rangle$. The splitting function is updated by training two SVMs using the previous meta-labels for classes $\mathcal{K}$ and assigning samples of the new class κ' to -1 or 1, respectively. Each SVM is initialized with $\vec{w_n}$. The updated function $f'^n(\vec{x}) = \text{sgn}\langle \vec{w'_n}, \vec{x} \rangle$ is given by the SVM with the highest information gain (1).

Fig. 3 illustrates the four approaches for incremental learning with NCM forests.

4.5 Node sampling for partial tree update

Updating a splitting node during RTST and RUST implies updating the whole subtree, but updating all N splitting nodes equals the inefficient retraining of the tree from scratch. We therefore investigate three different distributions $p(n)$ that are used to select a node or subtree for updating:

a) Uniform. Each splitting node is assigned equal probability: $p(n) = \frac{1}{N}$, where N denotes the number of splitting nodes.

b) Subtree size. The computational cost of retraining depends on the size of the subtrees. Thus we set the probability of a node n to be updated as inversely proportional to the cardinality of the subtree T_n with n as root: $p(n) \propto (|T_n| + 1)^{-1}$ where $\sum_n p(n) = 1$.

c) Quality. We measure the quality of a subtree with root node n and corresponding leaves $\{l \in \text{leaves}(n)\}$ by the information gain from its root to the leaves. Let S^n be the samples of classes $\mathcal{K} \cup \kappa'$ observed at the node n and S^l the

samples observed at a leaf l. The quality Q is then computed:

$$Q(n) = H\left(S^n\right) - \sum_{l \in \text{leaves}(n)} \frac{|S^l|}{|S^n|} H\left(S^l\right), \qquad (10)$$

where H denotes the class entropy as in (1). Since a splitting function is only selected if the information gain is larger than zero, $Q(n) > 0$. The probability of a node n to be updated is then inversely proportional to the quality: $p(n) \propto Q(n)^{-1}$ where $\sum_n p(n) = 1$. Hence we prefer to update subtrees which perform poorly, rather than focusing on computational effort.

5 Experiments

We evaluate the NCMF and SVMF Forests and the corresponding incremental learning approaches discussed in Section 4 on a challenging large-scale dataset for image classification, namely "ImageNet Large Scale Visual Recognition 2010 challenge benchmark (ILSVRC10)" [13]. It consists of 1k categories, between 668 and 3047 training images per class and 1.2M in total, and 150 testing images per category. The dataset is organized in a class hierarchy based on WordNet.

In Section 5.1, the impact of the parameters is evaluated in detail on a subset with up to 200 categories. As image features, we use a bag-of-words (BoW) representation. To this end, we use densely sampled SIFT features clustered into 1k visual words provided by the benchmark [13]. We normalized the BoW features by whitening the features by their mean and standard deviation computed over the starting training subset. When metric learning [6] is used for comparison, whitening is not performed in addition to metric learning. Section 5.2 compares the approaches to other methods on the entire large-scale datasets with all 1k categories. Finally, the impact of the dimensionality of the features is evaluated in Section 5.3 where we use 4k-dimensional features based on Fisher vectors.

As measure of performance, we use top-1 average accuracy. The training time without feature extraction and test time per image are measured per tree by wall clock in seconds and microseconds, respectively.

To evaluate incremental learning, we fixed a random permutation of all categories and used it throughout all experiments[1].

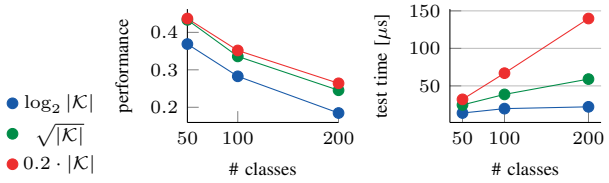
5.1 Parameters

5.1.1 Offline learning

We first evaluate parameters of the forests in an offline setting, *i.e.*, when all categories are presented at once. We always trained 50 trees and, if not stated otherwise, used $\mu = 10$ minimum training samples at a leaf as stopping criterion. For NCMF and SVMF, we sampled 1024 and 20 splitting functions without replacement at each node, respectively. For SVMF, we optimized the parameters for a linear SVM on the first 50 categories.

Splitting function for NCMF. The size of sampled classes $\mathcal{K}^n$ out of all classes $\mathcal{K}$ is an important parameter for NCMF.

1. We provide the fixed random order at http://www.vision.ee.ethz.ch/datasets_extra/mristin/ilsvrc_meta_2010.zip



a) performance b) test time

Fig. 4: Comparison of **a)** average classification accuracy and **b)** test time for different sizes of $\mathcal{K}^n \subset \mathcal{K}$. While setting $|\mathcal{K}^n|$ linear to the number of classes performs better than a sublinear growth, it takes much longer at the test time.

We compared $|\mathcal{K}^n| \in \left\{\log|\mathcal{K}|, \sqrt{|\mathcal{K}|}, 0.2|\mathcal{K}|\right\}$ and present the results in Fig. 4. The results show that $|\mathcal{K}_n| = \sqrt{|\mathcal{K}|}$ gives a good trade-off between accuracy and test time and is used for the rest of the paper.

Stopping criterion. The minimum number of samples at a leaf μ defines the stopping criterion for growing the trees. The smaller the number, the deeper the trees grow. Fig. 5a) shows that a small number increases the accuracy, but induces more computation at the test time. For the rest of the experiments, we use $\mu = 10$ and compare different forest variants trained and evaluated on 50 classes.

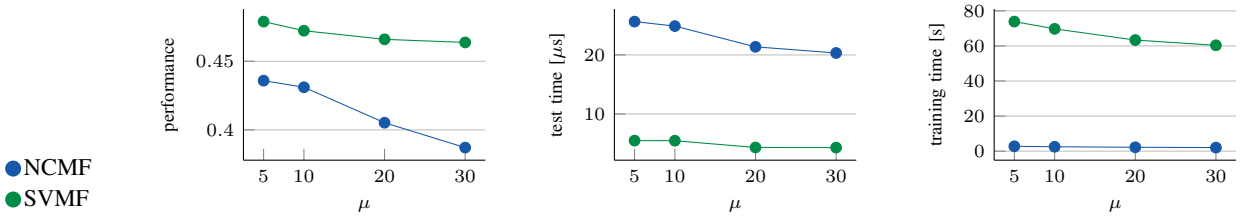
NCMF vs. SVMF. While SVMF is most accurate (0.47) as well as fast to evaluate (5.5μs) since only one inner product is performed per node, the SVM hyperplanes required longest training times (70s). NCMF is a good compromise between accuracy (0.43) and training times (2.5s) though it is slower at test time (24.9μs) due to computation of distances to the centroids at a node. Hence, NCMF is more suitable for a system which has to cope with a dynamic environment where new classes arrive frequently, while SVMF is better fit for a more static setting where the decisions about the learned categories are often evaluated.

Visualization. At the end of the paper, we visualize a single NCMF tree trained on 50 classes of *ILSVRC 2010*. Table 15 a)-c) illustrate the centroids stored at three different nodes and Table 15 d) and e) illustrate the path of two test images in the tree where one is correctly classified and the other misclassified.

5.1.2 Incremental learning

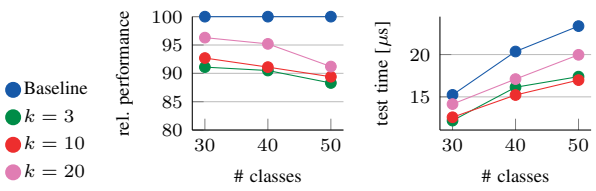
To evaluate the incremental learning approaches presented in Section 4, we train a forest on a pre-defined number k of initial classes and then incrementally add the other classes one by one. The performance is measured whenever the method has learned to recognize a certain number of classes. Since the goal is to match the performance of the forest re-trained at every new class, we measure the performance relatively to that baseline.

Comparison of node sampling. While ULS and IGT do not have any extra parameters, RUST and RTST depend on the sampling distribution (uniform, subtree size or quality) as well as on the ratio π of the splitting nodes sampled for update as discussed in Section 4.5. In Fig. 7, we compare



a) performance b) test time c) training time

Fig. 5: Measurements at 50 classes of **a)** performance, **b)** test time and **c)** training time of NCMF and SVMF baselines with variable constraints of μ minimal number of training samples at a leaf node. SVMF is much faster at test time and outperforms NCMF, but takes much longer (28x) to train.



a) performance b) test time

Fig. 6: Comparison of **a)** relative performance and **b)** test time of RUST applied to a NCMF with nodes sampled by quality and $\pi = 0.05$ starting with k initial classes measured at 30, 40 and 50 classes. Increasing the number of initial classes to 20 is beneficial, but has limited impact.

measurements of the different sampling schemes with RTST applied on NCMF and measured when the forest was trained to classify 50 classes, starting with 3 initial classes. Sampling by quality picks more relevant nodes and hence allows better updates with smaller portion π. With updating as little as 5% of the nodes sampled according to quality, we achieve a relative performance of 96.0% and a short training time of 16.7s. Using a uniform distribution, the training time is increased. The sampling based on the subtree size as proposed in [11], achieves a relative performance of 95.6% for $\pi = 0.5$ and takes 24.0s to train.

Further experiments with RUST confirmed these relations. While sampling by quality achieved 88.3% relative performance and took 10.3s to train, sampling by subtree size achieved 90.7% relative accuracy and needed 15.2s for training. In the following, nodes were sampled by quality with $\pi = 0.05$.

Comparison of the update strategies. Fig. 8 plots the relative performance, test and training time for the baseline NCMF and our approaches 'Update leaf statistics' (ULS), 'Incrementally grow tree' (IGT), 'Retrain subtree' (RTST) and 'Reuse subtree' (RUST) trained from $k = 3$ initial classes up to 50 classes. Splitting nodes were sampled by quality for RTST and RUST with $\pi = 0.05$. As ULS does not grow the trees, its test time is constant and the training time very low, but the final relative performance at 50 classes is poor (27.0%). IGT extends the trees, yielding higher test and training times, but achieves 83.5% relative performance while reducing training time of the baseline by a factor of 17 and test time by 2. IGT achieves 36.2% average accuracy, outperforming NCM, KNN, RF and MF [33], *cf.* Table 1a). RTST re-trains the nodes and achieves the best relative performance (96.0%), but takes longest to train. RUST outperforms IGT (relative performance 88.3%), suggesting that reusing the subtrees is indeed beneficial. It also speeds up the training of the baseline by a factor of 5 and is 1.6× faster to train than RTST. The gap in training times between baseline, RTST and RUST widens with the number of classes.

The incremental learning strategies can be applied to NCMF and SVMF. The results for both variants are plotted in Fig. 9. The measurements are performed at the final 50 classes and the results demonstrate not only that the relations between our incremental approaches in relative performance, test and training times are stable across the forest variants, but also reflect the results in Fig. 5. Baseline SVMF outperformed NCMF baseline by roughly 3% (47.2% compared to 43.3%), *i.e.* it was 1.09× better. The ratios are also similar for performance of RTST and RUST incremental approaches when they are performed on SVMF and NCMF, respectively: RTST on SVMF 44.6% *versus* RTST on NCMF 41.6% (1.07× better) and RUST on SVMF 40.0% *versus* RUST on NCMF 38.3% (1.05× better).

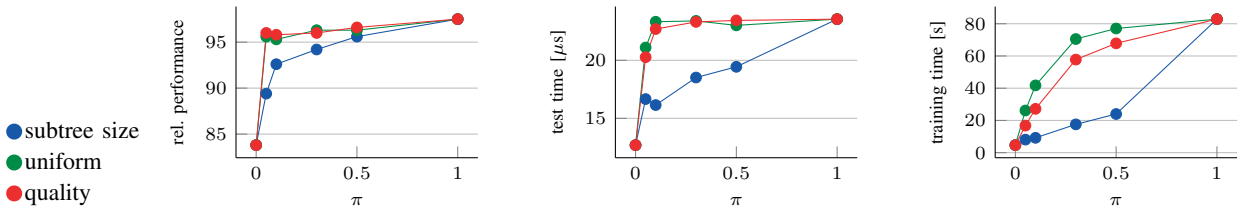
Initial classes. The influence of the number of initial classes on RUST of NCMF is shown in Fig. 6. The method is quite insensitive to the number of initial classes and already achieves good performance with only a few. Starting with 3 and 20 initial classes gives us the relative performance of 88.3% and 91.2%, respectively, a difference of only 2.9%.

So far we have used a single random permutation of the classes for the experiments. To evaluate the impact of the initial classes, we evaluate ten random permutations of the previously used 50 classes. The results are plotted in Fig. 10. The standard deviation never exceeded 10% of the mean values of the measurements indicating little impact of the order of the classes, which is desirable for incremental learning.

5.2 Large-scale

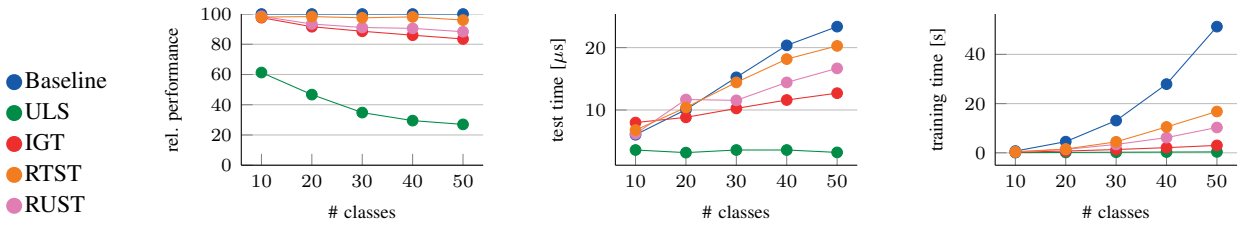
In the following section, we examine the behavior of our forests in experiments involving all 1k classes of *ILSVRC10*.

Before comparing our methods with other approaches, we study how our approaches cope with the batch size. Since in



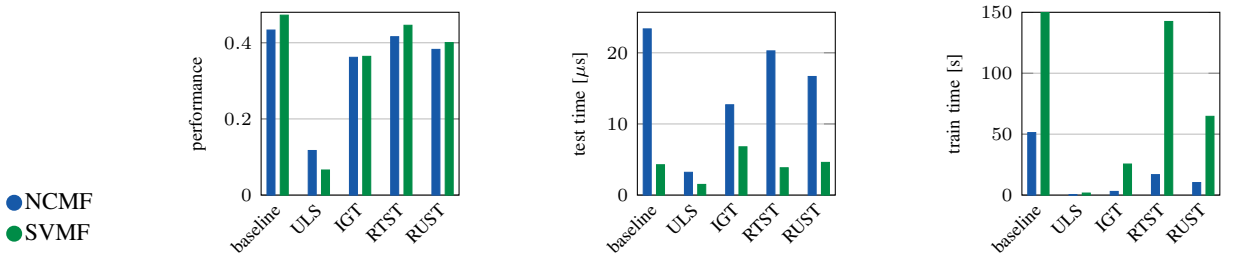
a) relative performance b) test time c) training time

Fig. 7: Starting with 3 initial classes, additional classes are incrementally learned until 50 classes are reached. RTST incremental training of NCMF with different schemes used for node sampling is evaluated. Using uniform sampling or subtree quality instead of subtree size as measure, a smaller number of nodes needs to be updated to achieve a good relative performance. Only a small portion of nodes (π) needs to be updated to achieve a relative performance of over 95%. Using the quality criterion in comparison to a uniform distribution results in lower training times.



a) relative performance b) test time c) training time

Fig. 8: Measurements at variable number of classes for an incremental training of NCMF starting with 3 initial classes. For RTST and RUST we used quality weighting with $\pi = 0.05$. 'Update leaf statistics' (ULS) is faster to train and test, but has inferior performance. 'Incrementally grow tree' (IGT) is slower than ULS both at train and test time, but achieves 83.5% of the baseline's performance at 50 classes. 'Retrain subtree' achieves the best performance (96.0% at 50 classes), but takes longest to train. 'Reuse subtree' (RUST) is a good trade-off between training time and relative performance (88.3% at 50 classes). The relative differences in training time increase with the growing number of classes.



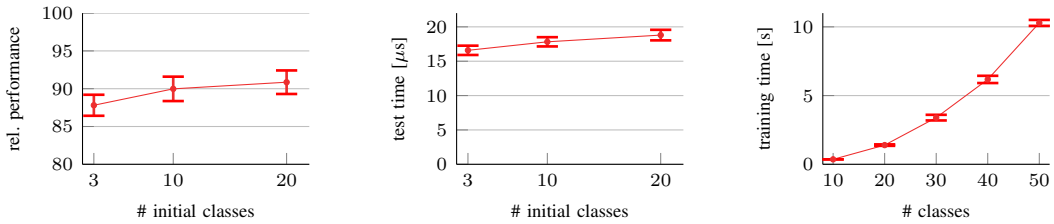
a) performance b) test time c) training time

Fig. 9: Measurements at 50 classes starting with 3 initial classes for various incremental learning approaches and forest variants. NCMF is much faster to train, but achieves a lower accuracy than SVMF and takes longer at the test time. The advantages and disadvantages of the forest variants for offline learning (*baseline*) are the same for incremental learning.

practice multiple classes can and do appear simultaneously, it is highly relevant that an incremental approach can handle such a setting. We trained our initial forests with 20 classes and incrementally updated it with chunks of 1, 10, 20 and 40 classes. The measurements were carried out whenever the forests integrated 100, 500 and 1000 classes. As shown in Fig. 11, the training time reduces by training several classes at a time. The batch size has a low impact on the relative classification accuracy of NCMF whereas SVMF performs better when adding only one class for each update.

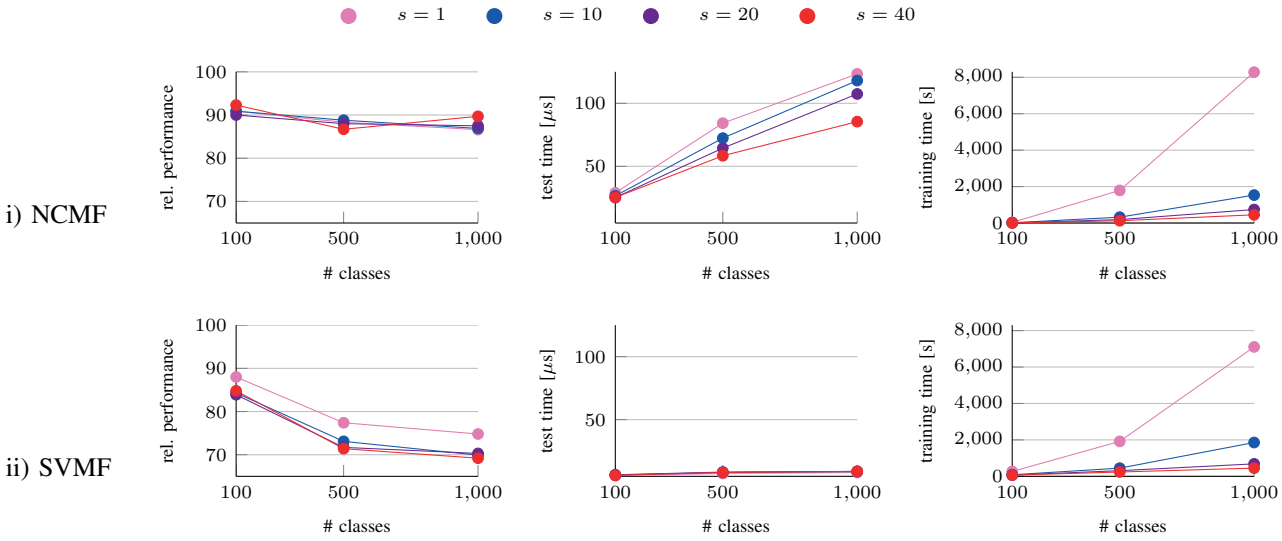
We compared NCMF and SVMF with other multi-class classifiers using the same features on all 1k classes of the *ILSVRC10* dataset. For comparison, we used nearest class mean classifier (NCM), NCM with metric learning [6] (MET+NCM), structured-output multi-class SVM [4] (MC SVM), k-nearest neighbors (KNN), Mondrian Forest [33] (MF) and Random Forest with axis-aligned splitting functions [10] (RF), which in our case outperformed RF with linear splitting functions. The method parameters were optimized by cross-validation for the first 50 classes.

The results in Table 1a) show that NCMF and SVMF perform comparable to NCM with metric learning [6]. In par-



a) relative performance b) test time c) training time

Fig. 10: Comparison of **a)** relative performance and **b)** test time of NCMF RUST. Nodes were sampled by quality with $\pi = 0.05$. Different number of classes were used for initialization and we measured at 50 classes and 10 random permutations of the classes. **c)** Training time for 3 initial classes over 10 random permutations of the classes. The small standard deviations indicate the limited impact of the order of the classes.



a) relative performance b) test time c) training time

Fig. 11: Comparison of **a)** relative performance, **b)** test time and **c)** training time for RUST based on **i)** NCMF and **ii)** SVMF, respectively, when several classes (s indicates the chunk size) are added simultaneously, starting from 20 initial classes. Training with multiple classes in a batch can reduce the training time substantially.

ticular, both NCMF and SVMF outperform NCM, MC SVM, Mondrian Forest (MF) and conventional Random Forest (RF) by a significant margin.

While we compared different forest variants with other approaches in Table 1a), we now compare the incremental learning approaches of NCMF and SVMF on all 1k classes in Table 1b) and c). Since IGT of NCMF and SVMF already outperforms NCM, KNN, MF and RF, we focus on NCM with metric learning [6] for incremental learning, which performed comparable to SVMF, *cf.* Table 1a). We start with $k = 10$ and $k = 20$ initial classes. The setting for the incremental learning of our forests remains the same, *i.e.*, the whitening is estimated on the initial k classes. For MET$_k$+NCM, the metric is only learned on the initial classes, and the model is updated with projected centroids of the new classes. According to Table 1, RUST outperforms IGT indicating that updating the trees is beneficial. While it was shown in [6] that a metric learned on 800 classes is applicable to the other 200 classes, the learned metric on up to 20 classes does not generalize well, making the method unsuitable for a small initial training set. In this case, the three approaches IGT, RUST and RTST applied to either NCMF or SVMF outperform MET$_k$+NCM. The relations between incremental training methods on NCMF presented in Fig. 8 are also corroborated in Table 1b) and c). However, the differences between SVMF and NCMF at 1k classes are smaller for RUST than for RTST. The improvement of the SVMF baseline by factor 1.21 over the NCMF baseline is preserved by RTST. At 1k classes, RTST with SVMF is $1.20\times$ better than RTST with NCMF.

The training and test times of our approaches across forest variants trained from the initial 20 up to 1k classes are given in Table 2. For the same setting, we also plot the absolute and relative performance with respect to training time for all approaches in Fig. 12. Although the baseline SVMF achieves a better accuracy than NCMF (*cf.* Table 1), NCMF achieves a better relative performance for incremental learning and compensates partially for the differences of the baselines. The plots also show that the presented approaches offer various trade-offs between training time and classification accuracy and the right choice of the approach depends on the

	method \ # of classes	50	500	1000
a) baseline	NCM	0.31	0.11	0.07
	MET+NCM [6]	0.44	**0.19**	**0.14**
	MC SVM [4]	0.42	0.10	0.05
	KNN	0.28	-	-
	MF [33]	0.28	0.08	-
	RF [10]	0.30	0.09	0.06
	NCMF	0.43	0.16	0.11
	SVMF	**0.47**	**0.19**	**0.14**

	method \ # of classes	50	500	1000
b) $k=10$	MET_{10}+NCM [6]	0.28 (63.0%)	0.08 (42.8%)	0.05 (39.1%)
	NCMF+ULS_{10}	0.25 (58.1%)	0.05 (32.9%)	0.03 (28.6%)
	NCMF+IGT_{10}	0.38 (88.7%)	0.12 (74.7%)	0.08 (74.7%)
	NCMF+RTST_{10}	0.41 (94.2%)	0.16 (96.6%)	0.11 (97.2%)
	NCMF+RUST_{10}	0.39 (90.6%)	0.14 (86.0%)	0.10 (84.9%)
	SVMF+ULS_{10}	0.19 (41.1%)	0.04 (19.9%)	0.02 (16.6%)
	SVMF+IGT_{10}	0.41 (85.9%)	0.13 (66.2%)	0.09 (65.2%)
	SVMF+RTST_{10}	**0.43** (91.8%)	**0.19** (95.6%)	**0.13** (95.9%)
	SVMF+RUST_{10}	0.42 (88.1%)	0.14 (71.2%)	0.09 (68.5%)
c) $k=20$	MET_{20}+NCM [6]	0.32 (68.2%)	0.09 (50.0%)	0.06 (46.2%)
	NCMF+ULS_{20}	0.30 (70.0%)	0.07 (43.2%)	0.04 (38.6%)
	NCMF+IGT_{20}	0.39 (90.0%)	0.12 (76.6%)	0.09 (75.9%)
	NCMF+RTST_{20}	0.41 (95.0%)	0.16 (97.9%)	0.11 (100.1%)
	NCMF+RUST_{20}	0.40 (92.1%)	0.14 (88.8%)	0.10 (86.9%)
	SVMF+ULS_{20}	0.29 (61.0%)	0.05 (28.3%)	0.03 (24.9%)
	SVMF+IGT_{20}	0.43 (90.8%)	0.13 (69.0%)	0.09 (67.1%)
	SVMF+RTST_{20}	**0.45** (94.5%)	**0.19** (97.9%)	**0.14** (99.5%)
	SVMF+RUST_{20}	0.43 (91.9%)	0.14 (73.1%)	0.10 (69.9%)

TABLE 1: Comparison of baselines and different incremental learning methods measured at 50, 500 and 1k classes of [13] all starting with the same initial classes. The classification accuracy is reported in the cells, while relative performance to the corresponding baseline is given as percentage in brackets. The whitening for our methods as well as the metric MET_k were learned on k initial classes in b) & c). Incremental methods were trained with batches of 10 classes. We set $\pi = 0.05$ and sample nodes by quality for RUST and RTST. While our baseline versions of NCMF and SVMF match the state-of-the-art method MET+NCM [6], NCMFs and SVMFs with RUST and RTST consistently outperform MET+NCM [6] for incremental learning.

	training time		test time [μs]	
	NCMF	SVMF	NCMF	SVMF
baseline	3hrs	32hrs	147	7
ULS	19s	23s	57	4
IGT	2min	6min	63	9
RTST	45min	8hrs	143	7
RUST	25min	31min	118	8

TABLE 2: Training and test times for incremental approaches based on NCMF and SVMF. We initialized with $k = 20$ classes, trained with batches of $s = 10$ classes at a time and measured at 1k classes. Test times are given per image and tree without feature extraction in microseconds. Training times are given per tree. For baselines, we indicate the training time needed for re-training at each batch. In comparison, training times for 1k classes for MET_{1000}+NCM is 36hrs and MC SVM is 2.5hrs.

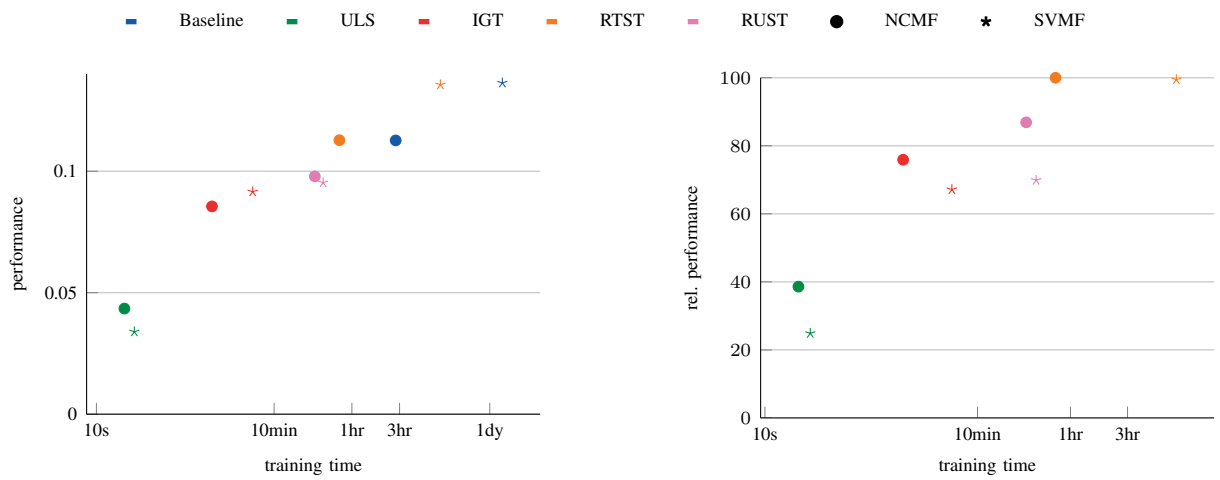
application.

We also compared our methods with MULTIpLE [9] using the publicly available code [43]. MULTIpLE is an incremental approach based on least-squares SVM [44]. The approach, however, is not suitable for large-scale problems due to its memory requirements. On a machine with 50GB RAM, we could run the approach for up to 100 classes with 100 training samples per class. The parameters of the linear SVMs were estimated by cross-validation on the first 50 categories and fixed through the experiments. Fig. 13 a) shows that LSSVM [44] is outperformed by NCMF and SVMF. The performance of LSSVM in recognizing 100 classes was 0.25, while NCMF and SVMF achieved 0.34 and 0.38.

We also present in Fig. 13 the results of an experiment performed in an incremental setting where we compared MULTIpLE with our RTST applied to NCMF and SVMF, respectively. The training was initialized with $k = 10$ classes and the models were incrementally updated by one class ($s = 1$). Due to aforementioned memory limitations, we limited the number of training samples for MULTIpLE (100 per each individual "source" and "train" set [43], respectively). For our incremental approach, we sampled nodes by quality and set $\pi = 0.05$ and did not restrict the number of training samples. Both the absolute as well as the relative performance were measured at 50 and 100 classes or when the memory limit was reached, which was the case for MULTIpLE at 70 classes. NCMF and SVMF incrementally trained by RTST outperform MULTIpLE [9] by a margin not only in absolute performance, but are also better in relative performance measured relatively to the corresponding baseline. MULTIpLE achieved only 84.0% of the performance of LSSVM, while RTST incremental training almost matched the baseline (97.9% and 98.4% for NCMF and SVMF, respectively).

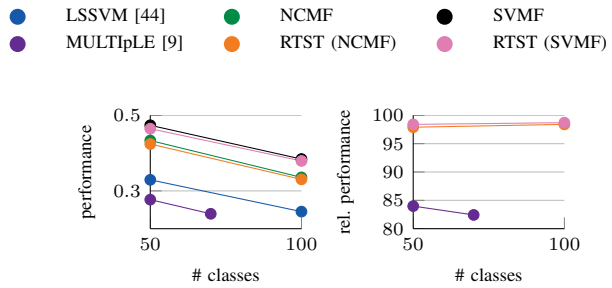
5.3 Feature dimensionality

The BoW features we used so far have a dimensionality of 1k. To investigate the impact of feature dimensionality, we employed 4k-dimensional features based on Fisher Vectors (FV), which were also used in [6]. In Fig. 14, the improvement of FV over the 1k ones is reported, measured at 1k classes. In general, all methods benefit from higher dimensional, more complex features. MET+NCM [6] which also performs a dimensionality reduction on the feature space benefits more from the high dimensional features than NCMF or SVMF. While MET+NCM achieves an average accuracy of 0.39, NCMF baseline and SVMF baseline achieve only 0.23 or 0.28, respectively. Yet, in the incremental setting, NCMF and SVMF perform better than MET+NCM. As a matter of fact, NCMF+IGT (average accuracy 0.18), NCMF+RUST (0.20) and NCMF+RTST (0.23) outperform MET_{10} + NCM (0.16),



a) abs. performance b) rel. performance

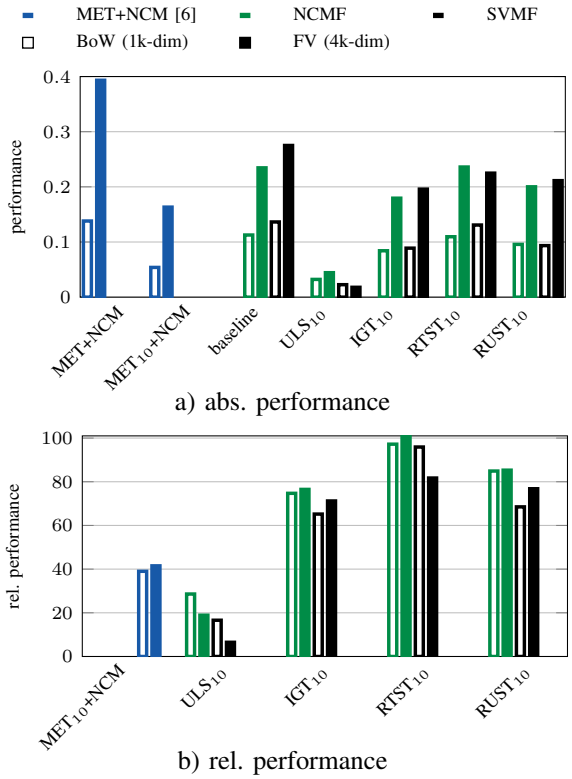
Fig. 12: Comparison of **a)** absolute and **b)** relative performance with respect to training time of our incremental methods based on NCMF and SVMF. We initialized with $k = 20$ classes, trained with batches of $s = 10$ classes at a time and measured at 1k classes. Training times are given per tree. The absolute performance shows that the presented approaches offer various trade-offs between training time and classification accuracy. In this scenario, however, there are also a few combinations that are not Pareto optimal, namely SVMF+ULS, SVMF+RUST, and NCMF. The relative performance shows that NCMFs retain the accuracy better than SVMFs for incremental learning and compensate partially for the lower absolute accuracy of NCMFs for offline learning.



a) abs. performance b) rel. performance

Fig. 13: Comparison of **a)** absolute and **b)** relative performance of MULTIpLE [9] and its baseline LSSVM [44] with our baseline forests NCMF and SVMF and their incrementally trained variants obtained by RTST. All incremental methods were initialized with $k = 10$ classes and the models were incremented by one class ($s = 1$). The measurements were performed at 50 and 100 classes. MULTIpLE runs out of memory at 70 classes. The relative performance was measured to the respective baseline. NCMF and SVMF outperform MULTIpLE [9] and LSSVM [44] both in absolute and relative performance.

and the same holds for SVMF+IGT (0.20), SVMF+RUST (0.21) and SVMF+RTST (0.22). Fig. 14 b) shows that the relative performance of the incremental learning approaches is quite stable although FV improve the absolute performance and increase the feature dimensionality. Increasing the dimensionality by a factor of 4 resulted in 2-4 times longer training times of our forests.



a) abs. performance

b) rel. performance

Fig. 14: Comparison of **a)** absolute and **b)** relative performance of our methods using 1k-dim bag-of-words and 4k-dim Fisher Vectors (FV). Incremental models started with 10 initial classes and used batches of $s = 10$ new classes. Performance is measured at 1k classes. We compare against MET+NCM [6] learned on all 1k classes and MET_{10}+NCM where the metric is learned only on the 10 initial classes.

Fig. 15: Visualization of a single NCMF tree trained on 50 classes of *ILSVRC 2010*. In the first three rows, we show some of the centroids stored at a node at **a)** depth 0 (root), **b)** depth 10 and **c)** depth 15. We illustrate each centroid by the three closest training images of its Voronoi cell observed at the node. If the cell contains less than three images as in **b)**, all images are shown. The assigned routing direction of the splitting function of the node is indicated by the border style; dashed means left and dotted right, respectively. Splitting at **a)** depth 0 is very general and becomes more and more specific as we move to deeper nodes **b)** and **c)**. In **d)** and **e)**, we show two test images on the left hand side with unknown ground-truth label and their paths through the tree. Only the nodes at depths 4-6 as well as the final node are displayed and they are represented by the centroid closest to the test image. Each centroid is again visualized by three images. In **d)**, the centroids become more precise along the path and very accurate at the final node where the image is correctly classified. In **e)**, the image is misclassified as Pekingese.

6 Conclusion

In this paper, we have examined how two variants of Random Forests (RF), namely Nearest Class Mean Forests (NCMF) and SVM Forests (SVMF), perform for large-scale multiclass image classification. As we have shown, both variants outperform NCM classification, multiclass SVM and conventional or Mondrian RFs in such a challenging setting. While our forests achieve competitive results in a setting where all classes are known a-priori, efficient techniques to incrementally add new classes to NCMF and SVMF are also proposed. In particular, the ability to reuse subtrees allows us to add new classes at a fraction of the cost of retraining a complete NCMF, while preserving the overall accuracy. Similarly, an incremental technique that retrains selected SVMF subtrees maintains a very high relative performance. We have performed extensive experiments in the context of image classification when the number of classes grows over time. Since NCMF and SVMF are quite insensitive to the number of initial classes and to the order in which the classes are added, they are well suited for incremental learning. For training, we assume that all previous training samples are accessible and decorrelate the features given the initial training data. This limitation can be overcome by keeping only a subset of the data at each step and including local feature decorrelation and selection in each split node.

Acknowledgments

The authors acknowledge financial support from the CTI project (15769.1 PFES-ES), DFG Emmy Noether program (GA 1927/1-1), DFG project (GA 1927/2-2 FOR 1505) and Toyota.

References

[1] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *IJCV*, 2010.
[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *CVPR*, 2009.
[3] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *TPAMI*, 2008.
[4] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid, "Good practice in large-scale learning for image classification," *TPAMI*, 2013.
[5] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei, "What does classifying more than 10,000 image categories tell us?" in *ECCV*, 2010.
[6] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka, "Distance-based image classification: Generalizing to new classes at near-zero cost," *TPAMI*, 2013.
[7] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "SUN database: Large-scale scene recognition from abbey to zoo," in *CVPR*, 2010.
[8] A. Bendale and T. Boult, "Towards open world recognition," in *CVPR*, 2015.
[9] I. Kuzborskij, F. Orabona, and B. Caputo, "From N to N+1: Multiclass transfer incremental learning," in *CVPR*, 2013.
[10] L. Breiman, "Random forests," *Machine Learning*, 2001.
[11] M. Ristin, M. Guillaumin, J. Gall, and L. Van Gool, "Incremental learning of NCM forests for large-scale image classification," in *CVPR*, 2014.
[12] B. Yao, A. Khosla, and L. Fei-fei, "Combining randomization and discrimination for fine-grained image categorization," in *CVPR*, 2011.
[13] A. Berg, J. Deng, and L. Fei-Fei, "Large scale visual recognition challenge 2010," http://www.image-net.org/challenges/LSVRC/2010, 2010, [Online; accessed 1-Nov.-2013].
[14] J. Sanchez and F. Perronnin, "High-dimensional signature compression for large-scale image classification," in *CVPR*, 2011.
[15] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang, "Large-scale image classification: Fast feature extraction and SVM training," in *CVPR*, 2011.
[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *NIPS*, 2012.
[17] J. Sanchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the Fisher vector: Theory and practice," *IJCV*, 2013.
[18] J. Deng, J. Krause, A. Berg, and L. Fei-Fei, "Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition," in *CVPR*, 2012.
[19] S. Bengio, J. Weston, and D. Grangier, "Label embedding trees for large multi-class tasks," in *NIPS*, 2010.
[20] J. Deng, S. Satheesh, A. C. Berg, and L. Fei-fei, "Fast and balanced: Efficient label tree learning for large scale object recognition," in *NIPS*, 2011.
[21] B. Liu, F. Sadeghi, M. Tappen, O. Shamir, and C. Liu, "Probabilistic label trees for efficient large scale image classification," in *CVPR*, 2013.
[22] R. Salakhutdinov, A. Torralba, and J. Tenenbaum, "Learning to share visual appearance for multiclass object detection," in *CVPR*, 2011.
[23] A. Bosch, A. Zisserman, and X. Muñoz, "Image classification using random forests and ferns," in *ICCV*, 2007.
[24] D. Nistér and H. Stewénius, "Scalable recognition with a vocabulary tree," in *CVPR*, 2006.
[25] A. Vezhnevets, V. Ferrari, and J. M. Buhmann, "Weakly supervised structured output learning for semantic segmentation," in *CVPR*, 2012.
[26] N. Razavi, J. Gall, and L. Van Gool, "Scalable multi-class object detection," in *CVPR*, 2011.
[27] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101: Mining Discriminative Components with Random Forests," in *ECCV*, 2014.
[28] A. Vedaldi and B. Fulekerson, "VLFeat: An open and portable library of computer vision algorithms," http://www.vlfeat.org/, 2008, [Online; accessed 1-Nov.-2013].
[29] M. Godec, P. Roth, and H. Bischof, "Hough-based tracking of non-rigid objects," in *ICCV*, 2011.
[30] P. Domingos and G. Hulten, "Mining high-speed data streams," in *SIGKDD*, 2000.
[31] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "On-line random forests," in *OLCV*, 2009.
[32] S. Schulter, C. Leistner, P. M. Roth, L. Van Gool, and H. Bischof, "Online Hough-forests," in *BMVC*, 2011.
[33] B. Lakshminarayanan, D. Roy, and Y. W. Teh, "Mondrian forests: Efficient online random forests," in *NIPS*, 2014.
[34] T. Yeh, J. Lee, and T. Darrell, "Adaptive vocabulary forests for dynamic indexing and category learning," in *ICCV*, 2007.
[35] A. Yao, J. Gall, C. Leistner, and L. Van Gool, "Interactive object detection," in *CVPR*, 2012.
[36] T. Tommasi, F. Orabona, and B. Caputo, "Safety in numbers: Learning categories from few examples with multi model knowledge transfer," in *CVPR*, 2010.
[37] M. Guillaumin and V. Ferrari, "Large-scale knowledge transfer for object localization in ImageNet," in *CVPR*, 2012.
[38] M. Guillaumin, D. Kuettel, and V. Ferrari, "ImageNet auto-annotation with segmentation propagation," *IJCV*, 2014.
[39] M. Rohrbach, M. Stark, and B. Schiele, "Evaluating knowledge transfer and zero-shot learning in a large-scale setting," in *CVPR*, 2011.
[40] M. Hasan and A. K. Roy-Chowdhury, "Incremental activity modeling and recognition in streaming videos," in *CVPR*, 2014.
[41] A. Wang, G. Wan, Z. Cheng, and S. Li, "An incremental extremely random forest classifier for online learning and tracking," in *ICIP*, 2009.
[42] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, 1985.
[43] I. Kuzborskij, F. Orabona, and B. Caputo, "From N to N+1: Multiclass transfer incremental learning (code)," http://idiap.ch/~ikuzbor/code/cvpr13_code.zip, 2013, [Online; accessed 23-May-2014].
[44] J. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, 1999.

Marko Ristin obtained his PhD from the ETH Zurich, Switzerland.

Matthieu Guillaumin is a post-doctoral researcher in the Computer Vision Laboratory of ETH Zurich, Switzerland.

Juergen Gall is professor at the University of Bonn and head of the Computer Vision Group.

Luc Van Gool is professor at the Katholieke Universiteit Leuven in Belgium and the ETH in Zurich, Switzerland, where he leads research in computer vision.