Error-Driven Incremental Learning in Deep Convolutional Neural Network for Large-Scale Image Classification

Tianjun Xiao[†], Jiaxing Zhang[‡], Kuiyuan Yang[‡], Yuxin Peng[†], and Zheng Zhang[‡]
[†]Institute of Computer Science and Technology, Peking University, Beijing 100871, China
[‡]Microsoft Research Asia, Beijing, P.R.China
{xiaotianjun,pengyuxin}@pku.edu.cn; {jiaxz,kuyang,Zheng.Zhang}@microsoft.com

ABSTRACT

Supervised learning using deep convolutional neural network has shown its promise in large-scale image classification task. As a building block, it is now well positioned to be part of a larger system that tackles real-life multimedia tasks. An unresolved issue is that such model is trained on a static snapshot of data, while data is always gradually collected in real application scenario. Instead, this paper positions the training as a continuous learning process as new classes of data arrive. A system with such capability is useful in practical scenarios, as it gradually expands its capacity to predict increasing number of new classes. It is also our attempt to address the more fundamental issue: a good learning system must deal with new knowledge that it is exposed to, much as how human do.

We developed a training algorithm that grows a network not only incrementally but also hierarchically. Classes are grouped according to similarities, and self-organized into levels. The newly added capacities are divided into component models that predict coarse-grained superclasses and those return final prediction within a superclass. Importantly, all models are cloned from existing ones and can be trained in parallel. These models inherit features from existing ones and thus further speed up the learning. Our experience points out advantages of this approach, and also yields a few important open questions.

Categories and Subject Descriptors

I.5.1 [**Pattern Recognition**]: Models—*Neural nets*

General Terms

Algorithms, Experimentation, Performance

Keywords

Incremental Learning; Deep Convolutional Neural Network; Large-scale Image Classification

MM'14, November 03 - 07 2014, Orlando, FL, USA.

http://dx.doi.org/10.1145/2647868.2654926.

1. INTRODUCTION

This paper focuses on incremental learning of deep convolutional neural network (DCNN) [14] in image classification task. By *incremental*, we mean that batches of labeled data of new classes are made available gradually. Our objective is to train a deep neural network that performs well at each of such steps. Figure 1 illustrates incremental learning in a multiclass classification model.

Three aspects motivate this study. The first is from the perspective of end applications. The net trained by image classification task can perform as a critical building block in many multimedia tasks. A DCNN model can map images to text, in tasks such as image tagging and annotation, the tags can be further used in image retrieval tasks [18]. Other tasks applies DCNN indirectly. For example, researchers have used a fine-tuned DCNN trained by image classification using ImageNet data to extract features in the detection task of TRECVID [20]. In all such real-world scenarios, classes and their associated labeled data are always collected in an incremental manner. As such, incremental learning plays a critical role.

The second is a performance one. Applying deep learning to image classification has made rapid progress. In a short span of one year, DCNN has improved the top-5 error rate on the challenge of ImageNet 1K-category classification from 26.2% to 15.3% [14]. Yet, the same network performs poorly on the more general 22K-category classification. One obvious culprit is the relatively limited network capacity. Thus, one option is to substantially increase the network capacity. The difficulties are two-folds. First, large models are inherently difficult to train, probably exponentially so. Second, it is not clear at all how and where new capacities should be allocated. It is more prudent to incrementally evolve the network capacity onwards, which is one principle behind the model proposed in this study.

The third motivation is more fundamental: we believe that this is a more general pattern of learning. As we are exposed to new and more data, we don't start learning from a blank slate. Rather, we leverage what's been learned and absorb new knowledge in a continuous process. Unlike other transfer-learning tasks where the features extracted in one domain is applied to a different but similar one (e.g. applying ImageNet feature sets to PASCAL VOL data [6]), the problem here is to transfer the existing features and learn new one in the *same* task with an ever expanding scope. This process can be emulated with the setting we outlined above, and the goal is to understand what alternatives would work well.

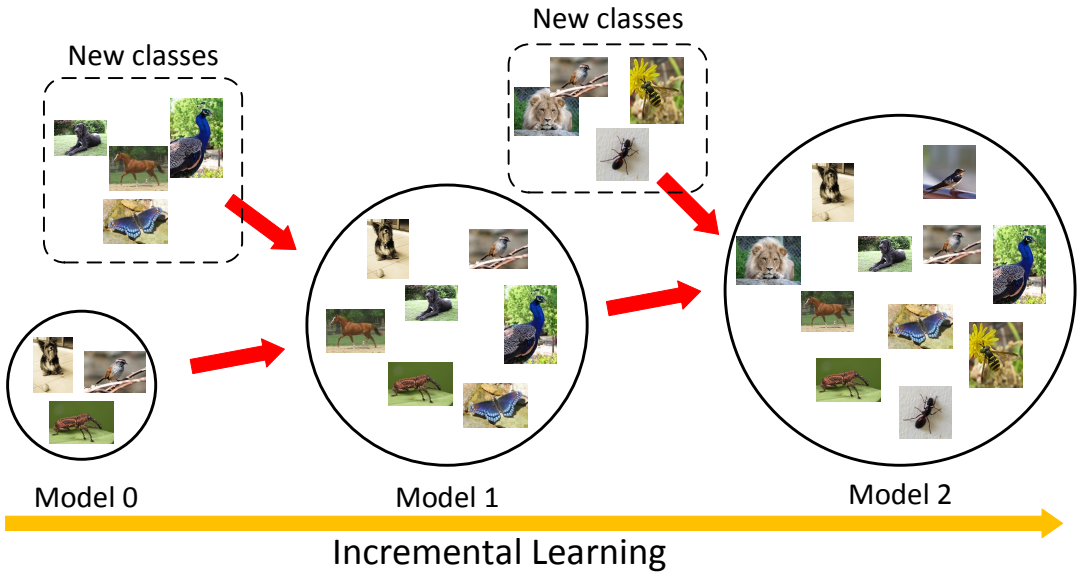

Figure 1: Incremental Learning in Multiclass Classification: the model needs to evolve with arrival of data of new classes.

There have been relatively little study in putting learning in such a dynamic and evolving context. One reason is that large amount of labeled data is only available more recently. Second, doing incremental learning using deep neural network is a new problem and faces inherent technical challenges. Unlike SVM-based approaches, neural networks embed feature extraction and classification in one coherent architecture and within the same model. A consequence is that the modification of parts of the parameter space immediately affects the rest of the model globally. For instance, the so-called "catastrophic forgetting" problem [10] refers to the destruction of existing features learned from earlier data, when the model is exclusively trained with data of new classes. A related difficulty is the allocation of new parameters, as mentioned earlier.

We draw intuition from our everyday experience of learning. For natural objects, there is an inherent ontology hierarchy due to the process of evolution. Artifacts, too, borrow elements from existing ones as one key factor of innovation. As such, it is highly unlikely that there is one flat space where we assign probability to predictions (i.e. with one softmax output layer for the entire 22K ImageNet classes). Rather, we first make a reasonable guess, and zoom into a sub-space where classes are similar. This process is inherently iterative and may involve multiple levels. If that's the case, the model should similarly have a hierarchical structure and the inference should allow iterative refinement.

In this paper, we propose a model that grows organically and hierarchically, as new classes become available. We make the following contributions. First, the model is hierarchical, and the total classes are split gradually towards the leaf of the tree, where similar classes are grouped. Importantly, this process is guided by a pragmatic, error-driven preview process. Second, we carefully control the growth of the network capacity, allocating them according to the responsibility of the components (i.e. coarse-grained routing versus fine-grained prediction). Third, we initialize the new parameters using cloning, thereby maximally retain the learned features. Finally, we perform a detailed study and reveal how learning propagates, with and without cloning.

Our preliminary results show that this approach is promising. Cloning, rather than starting from scratch, can learn faster or better, and often both. Comparing with the existing approaches that always train from scratch, we can reach similar performance by starting from an existing model, but with up to 25% few samples. The results of hierarchical model are mixed, in the sense that the performance gain is limited, due to the errors of superclass prediction. We have gained a few important lessons and insights to guide our future work.

The rest of the paper is organized as follows. We cover related work in Section 2 and then describe our overall architecture in Section 3. We then explain the training algorithm in Section 4, which also depicts how the network grows. Detailed performance study is covered in Section 5. We discuss what we learned, future work and conclude in Section 6.

2. RELATED WORK

Our work attempts to bring the promise of deep learning to the practical and more general paradigm of incremental learning. We organize the discussion of related work according to researches in these areas.

Multi-class image classification and deep learning . The last decade has witnessed a great progress in multi-class image categorization, both in terms of the number of categories and accuracy. For instance, ImageNet has about 22K categories [4], compared to hundreds in the Caltech series [9, 11], and the collection is growing. When the dataset grows bigger, hierarchy pattern in the class set becomes more obvious. Jia *et al.* [3] researched on the classification result in ImageNet_10K dataset and report the phenomenon where misclassification information has some correlation with the semantic hierarchy of ImageNet. Griffin *et al.* [12] utilized the

taxonomy hierarchy in the class set to do hierarchical classification to pursue greater speedup. Besides those two works, others mainly focused on the classification accuracy on different dataset. The accuracy has been improving steadily with new developed image features over the years [17, 23, 19], and achieved a great leap with the renewed Deep Convolutional Neural Network [14]. The superiority of DCNN comes from its ability in simultaneously learning the feature extractor and classifier via the network with many layers. Besides image classification task, the DCNN model has been applied to some multimedia tasks directly or indirectly [4, 20].

Incremental learning. A crude definition of incremental learning is that learning is a continuous process with new data. Prior work has addressed two major scenarios, out of which the second one is relevant to this study. The first is concept drift in the training dataflow, and therefore the classifier learns in a non-stationary environment [5, 24]. The second is when there are existing classifiers that are related to the new classes to be learned [21, 1, 22, 8, 7, 16, 15].

Thrun [21] proposed the question as whether leaning the n-th classifier is easier (than learning the first), and since then researchers began to tackle this problem using transfer learning intuition, with techniques using fewer samples to get new models or better generalization. Two of the hot research topics derived from this problem is one-shot learning and zero-shot learning. Fei-Fei *et al.* [8] proposed a Bayesian transfer learning method to avoid learning new categories from scratch and instead using very few training samples. Tommasi *et al.* [22] proposed a multi model knowledge transfer method where source classifiers were weighted by learned coefficient. Lampert *et al.* [16] achieved zero-shot learning by introducing attribute-based classification. Kuzborskij *et al.* [15] pointed out that prior work mostly focused on binary classification problem (object detection), and proposed a discriminative method in the One-Versus-All multi-class classification task by transferring knowledge to a new class while preserving what has already been learned. Those works rely on shallow models instead of DCNN, and the category size is small in comparison.

The particular challenge with DCNN in the context of incremental learning (and in same sense transfer learning as well) is that it mingles feature extractor and classifier in one architecture. Goodfellow *et al.* [10] investigated the catastrophic forgetting problem in gradient-based neural networks. The study is however on the small MNIST dataset with small network, and the proposal of using of dropout is already in the default DCNN configuration. Nevertheless it qualitatively re-affirms the difficulty of achieving good performance on old and new tasks simultaneously. Many works have focused on performing domain adaptation, one such example is the one-shot learning by adapting features learned form ImageNet_1K dataset to other datasets [13]. The recent work of zero-shot learning assumes the availability of a semantic embedding space to which outputs of DCNN are projected. Our work differs in the goal, as we want to transfer the learning *within*the same task with larger dataset.

While many incremental learning works paid more attention on efficiency than accuracy, Bengio [2] offered a deeper insight that speed and quality can be obtained simultaneously if increments are made properly, noted as curriculum learning. Its key idea is to start learning on easier aspects of tasks and then gradually increase the difficulty. This setting may have positive effect on both the convergence speed and the quality of local minimum obtained. This work can help analyzing our results.

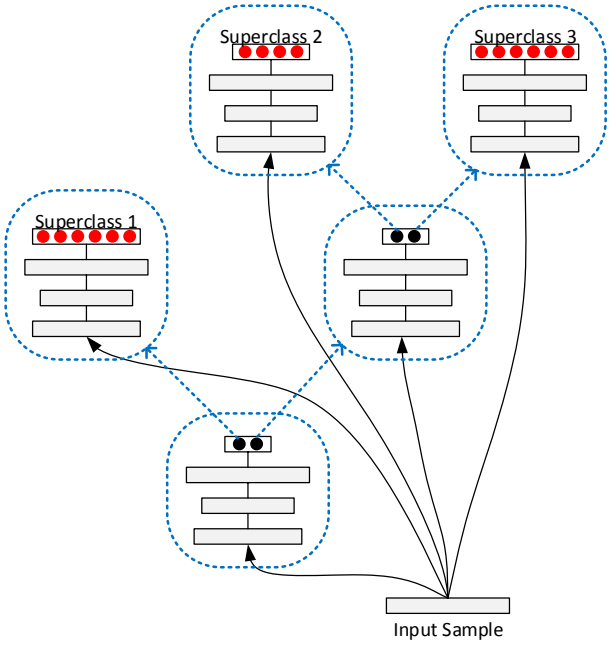

Figure 2: A hierarchy of models: branch models predict superclasses, leaf models return final predictions.

3. INCREMENTAL LEARNING MODEL

In this section, we will introduce our proposal of incremental learning model and its inference process. Without loss of generality, we assume there is a model M_0 that is already trained on N_0 classes. The goal of incremental learning is to evolve from M_{i-1} to M_i, to train N_i classes, in which $N_i - N_{i-1}$ are *new* classes. For example, we might have a DCNN classifier for 500 animal classes out of the 22K categories in ImageNet, we want to grow the network to classify 1000 and then the complete collection of animal classes.

Obviously, the model must increase its capacity to accommodate more classes. The simplest way to grow is widening the top softmax layer to output the extra probabilities on new classes. In other words, M_i share the same structure as M_{i-1} except it has N_i softmax units at the output. One obvious drawback is that the capacity increment is small: if the width of the layer before softmax is H then the additional parameters amounts to $H \times (N_i - N_{i-1})$. To put it more concretely, if we use the default configuration of DCNN [14] for the 500-class model, when we increase it to 1000 classes, the number of parameter increment is merely 1%.

We can make the model bigger by injecting units in the fully-connected layers or having more feature maps in the convolutional layers. However, it is not clear how this should be done. We could end up putting many untrained new parameters into the model everywhere, and their initialization becomes a black magic: too big or too small a random value will either ruin the existing model or making training tediously long.

A better approach must maximize the transfer of learned features, and still conservatively grow the capacity. The architecture we explore in this paper is a hierarchy of models, as shown in Figure 2. In our default setting, all the models (in each blue box) share the same topology and the amount of parameters, except the top output layers. All of them receive the same input sample, and have a softmax output. They logically make up a tree during the inference process. Models at different positions play different roles. Each leaf model performs the final prediction among a non-overlapping subset of the total classes. The branch models, on the other hand, have each of their output unit points to a child model, directing the prediction (eventually) to the correct leaf model. More specifically, the total classes are partitioned into *superclasses*, and each superclass is assigned to a leaf model (the red units in Figure 2). An input sample triggers the root model which outputs the probabilities of which branch the sample belongs to, and then the child model (can be a leaf or branch model) with the highest probability is chosen. This recursion continues until a leaf model is selected, and we take the output in this superclass as the final prediction.

The intuition in this tree-style prediction is that each branch model, being constrained by capacity, is only optimized for predicting a coarse collection of classes. A leaf model, on the other hand, concentrates on a more accurate fine-grained classification on a smaller set of classes. This architecture mirrors the hierarchical and iterative inference process that human brain seems to be adept at, as we do a crude prediction first, and then refine it further by discriminating against close competitors. It also allows the total model capacities to be divided according to responsibilities (coarse vs fine-grained classification). In such a hierarchy of models, the capacity growth is done by cloning new models from the exiting ones and expanding the tree. The cloning reduces the difficulty of initializing the added capacity. We will discuss the details of this incremental learning in the next section.

4. ALGORITHM

For ease of disposition, we will describe the algorithm when there is only one single model, and then generalize the procedure.

4.1 Starting from a Single Superclass

In the starting point of the training, all N_0 classes are in one single superclass and predicted by one model L_0. Thus, L_0 is a leaf model by itself. When new classes come and the superclass size increases to N_1, we have two choices to make the model bigger. One choice is simply extending L_0 to L_0' by inserting more output units, which conservatively increases a small amount of capacity. The second choice that substantially scales up the capacity is partitioning the superclass into K superclasses, and clone L_0 into several new leaf models $L_1, L_2, \ldots, L_K$ to predict within each of these new superclasses. A branch model B with K final output units is also cloned from L_0 to direct the prediction to the correct leaf model on a given input sample. These two choices are illustrated in Figure 3. We call the former *flat increment*, and the latter *clone increment*. There are three problems we need to address: 1) how to partition a superclass; 2) how to re-train these models with changed

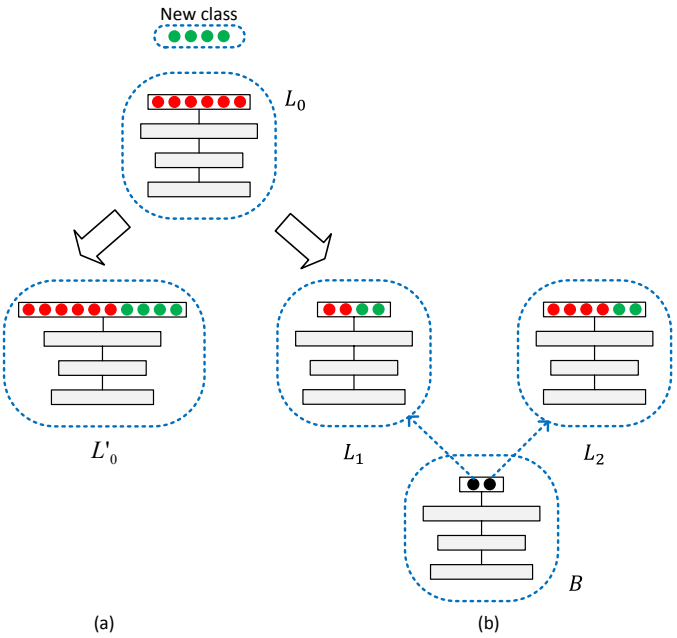

Figure 3: Two choices of capacity increment. (a) Flat increment. The output units is increased to hold more classes. (b) Clone increment. A single leaf model is duplicated to clones for more classes. A branch model is also cloned to direct the correct prediction of superclass.

data and objectives; 3) most importantly, how to decide the best strategy out of the two alternatives.

In clone increment, the network grows by one level, and the total N_1 classes are first clustered to superclasses by similarity. This allows the branch model B and the leaf models to focus on inter- and intra-superclass discriminative features, respectively. This is done using *error-driven preview*, which essentially *error-driven preview* observes the error distribution using current model L_0 on samples drawn from all the data, including both new and old. A validation set of N_0 are tested through L_0, and calculating a confusion matrix $C \in \mathbb{R}^{N_0 \times N_0}$ from the output. The entry C_{ij} denotes the probability that the i-th class is predicted to j-th class, which also measures the similarity between class i and j. We then use spectral clustering partition to split N_0 classes into K clusters based on the confusion matrix. The classes that are easy to confuse with each other are grouped into the same cluster as a superclass, with the desired side effects to minimize the confusion between superclasses. Next, $N_1 - N_0$ new classes are assigned to superclasses based on their confusion rates among the superclasses.

After the superclass partitioning, there are a total of K leaf models to train. Each of them has the same topology as L_0 except the output layer. Compared with L_0 which is only trained to predict N_0 old classes, each new leaf model L_i inherits a portion of old classes plus with some new classes. This change of data and objective requires each leaf model to be retrained. Instead of training from scratch, we train these new leaf models incrementally by copying L_0's parameters as initialization and using random initialization on the remaining new parameters (i.e. the weights connecting the

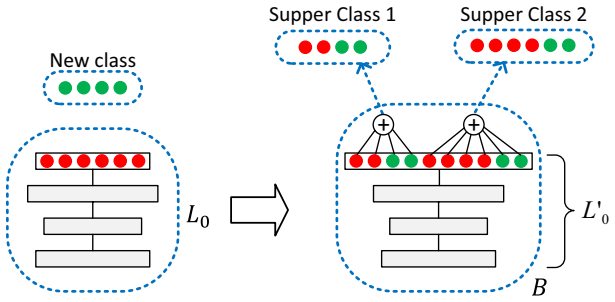

Figure 4: Cloning a branch model. We first do the flat increment and train the new leaf model L'_0. The units in L'_0 for each superclass are summed up for a branch model output unit.

units of the last hidden layer to the newly added output layer units). This training is much more efficient than from scratch.

We also evolve a new leaf model L'_0 to have N_1 output units, trained with flat incremental with data of N_1 classes. Finally, we clone the branch model B by copying parameters from L'_0. As illustrated in Figure 4, and simply sum up the softmax units belonging to each superclass as the predicted probability of that superclass.

At this point, we have built two separate models, the first is L'_0, and the second is the branch model B and its leaf models L_1 to L_K. Strictly speaking, choosing one over the other depends on the estimation of extra model capacity demanded by the new classes. We adopt a simpler strategy to simply let the two compete. This is illustrated in Algorithm 1.

Algorithm 1: ExtendLeafModel

input $(\mathbf{L_0}, \mathbf{S})$: leaf model L_0, superclass S
output $(\mathbf{L'_0}, \mathbf{B}, \mathbf{L_1}, \mathbf{L_2}, \ldots, \mathbf{L_K})$: leaf model L'_0 by flat increment, leaf models $\{L_1, L_2, \ldots, L_K\}$ and branch model B by clone increment
`/* flat increment*/`
incrementally train L_0 to L'_0
`/* clone increment */`
error-driven preview of S
partition S into $\{S_1, S_2, \ldots, S_K\}$
clone L_0 into new leaf models $\{L_1, L_2, \ldots, L_K\}$
for $i = 1$ to K **do**
  incrementally train L_i
end for
clone branch model $B \leftarrow L'_0$
`/* use clones or not */`
if prediction accuracy $\{B, L_1, L_2, \ldots, L_K\} > L'_0$ **then**
  $L'_0 \leftarrow \varnothing$
else
  $B \leftarrow \varnothing$
end if
return $(L'_0, B, L_1, L_2, \ldots, L_K)$

Note that all these models can be trained in parallel. More importantly, the process of cloning and incremental leaf model training automatically transfers features learned from the old model L_0 to the new leaf and branch models. Comparing with training such new models from scratch, the hope is that we can reach better accuracy with the same amount of time budget, or retain the accuracy with shorter training time, or both. Therefore, this is a building block whose performance is critical, and is the subject of much deeper analysis in later part of this paper.

4.2 Incremental Learning

Incremental learning from a single superclass can be generalized to a deeper hierarchy, at each step receiving new batches of classes (Algorithm 2). The first step is to distribute all the new classes into the existing superclasses by error-driven preview in a coarse granularity of superclass (line 1 to 6). Next, for each leaf model l, we grow it using flat increment or clone increment by Algorithm 1, discussed previously. Note that if the clone increment is selected, l is itself expanded to a subtree.

This algorithm can be applied repeatedly with new batches of classes, and naturally support more levels of hierarchy, independent of how many leaf models and how many levels of hierarchy exists in the current model. As the hierarchy deepens, the original trained branch models, especially those near the root, might lose accuracy due to update in their subtrees. We can choose to retrain each branch model by simple flat increment training.

Algorithm 2: IncrementalLearning

input $(\mathcal{S}, \mathcal{L}, \mathcal{B}, \mathcal{S}_{\mathbf{new}})$: superclass set $\mathcal{S}$, leaf model set $\mathcal{L}$ (each $l \in \mathcal{L}$ is corresponding to a $s \in \mathcal{S}$), branch model set $\mathcal{B}$, new class set $\mathcal{S}_{new}$
output $(\mathcal{S}, \mathcal{L}, \mathcal{B})$: updated superclass set $\mathcal{S}$, leaf model set $\mathcal{L}$, branch model set $\mathcal{B}$
`/* ditribute new classes to superclasses* /`
calculate the confusion matrix Φ with entry $\Phi(c, s)$ for probability of predicting $c \in S_{new}$ to $s \in \mathcal{S}$
for all $c \in \mathcal{S}_{new}$ **do**
  select $s \in \mathcal{S}$ with maximum $\Phi(c, s)$
  $s = s \cup \{c\}$
end for
`/* incremental training */`
for all $s \in \mathcal{S}$ and the corresponding $l \in \mathcal{L}$ **do**
  $(l', b, l_1, l_2, \ldots, l_K) =$ **ExtendLeafModel**(s, l)
  if $b \neq \varnothing$ **then**
    insert b to $\mathcal{B}$, replace l by $\{l_1, l_2, \ldots, l_K\}$ in $\mathcal{L}$
  else
    replace l by l' in $\mathcal{L}$
  end if
end for
`/* refine brach models (optional) */`
for all $b \in \mathcal{B}$ **do**
  incrementally train b according to updated subtrees
end for
return $(\mathcal{S}, \mathcal{L}, \mathcal{B})$

5. EXPERIMENTS

In this section, we will evaluate our framework on two datasets, using convergence speed and classification accuracy as criterion to show how to learn faster and better incrementally.

Dataset.

We prepare two datasets for our study. The first smaller dataset include all the 398 animal classes in ImageNet_1K, in which 501K and 18K images are used as the training set and validation set, respectively. We call it ImageNet_1K_Animal. It is used primarily to understand the performance of initialization with existing model parameters. We perform a fuller experiment growing a model incrementally with the second larger dataset selected from the 3998 animal synsets in ImageNet_22K. Samples of these synsets have a few problems:

- Not all of them have image samples. For those that do, there are severe unbalance in the number of samples; some of them contain fewer than 10 images.
- The image samples of an inner node in the ImageNet hierarchy can also belong to a leaf node. For such a image, there are multiple labels.

We therefore take all the leaf nodes in the ImageNet_22K hierarchy. We also remove classes with fewer than 100 images, results in 2282 classes. The training, validation and testing set contains 85%, 5%, 10% of the data respectively, with 1.6M, 91K and 183K images each. We call it ImageNet_22K_Animal.

To create an incremental training process, the small dataset is incremented from 195 randomly drawn classes to 398 classes, whereas the big dataset is incremented from 1000 to the full 2282 classes.

Model.

The baseline model is the one 8-layer convolutional network as described in [14], with configuration shown in Table 1. As discussed in Section 4.1, we have two ways to extend the capacity, *flat increment* by increasing the softmax output layer size, and *clone increment* by duplicating more models. In either case, the configurations stay the same except the top output layer, and network parameters are initialized by copying from an existing model, which we call *incremental learning*. We also compare the results against a baseline where the training starts from scratch, which is called *from-scratch learning*.

Training Details.

One of the "black magics" in training a deep network is the scheduling of learning rate adjustment. As we are interested in possible training time savings, we apply an adaptive learning rate tuning strategy to avoid tuning by hand. After training every 30K samples, we test with 1K samples randomly drawn from the whole validation set. We compute and monitor the average validation error for each epoch, and if it doesn't drop by more than 0.1% for three consecutive epochs, we stop the training or cut the learning rate by 10-folds. For one complete training, the learning rate is adjusted twice, starting from 0.01.

5.1 Result Overview

Results on ImageNet_1K_Animal.

With the model well trained on 195 animals, we can train it for 398 animals incrementally or from scratch. Table 2 summarizes the comparison between these two choices, by the error rates trained for different epochs. We hand-tuned the learning rate of the incremental runs for we want them to train for certain epochs. We observe that we can achieve better performance if we incrementally train from the existing 195-animal-model (36.8% versus 38.6%), taking slightly less time than the scratch run. On the other hand, with 25% few data (stopping with 30 epochs), incremental learning already beating the from-scratch one (37.9% versus 38.6%).

Table 2: Incremental/from-scratch learning results in ImageNet_1K_Animal from 195 animals to 398 animals

Training	Epochs	Error Rate
from-scratch	41	38.6%
incremental	10	41.6%
incremental	20	39.2%
incremental	30	37.9%
incremental	40	**36.8%**

The scenario we consider here is one where a model of smaller net has already been trained. From-scratch training simply throws that model away, whereas ours leverages it. The simple flat-incremental method actually creates a curriculum learning [2] scenario. Training a net classifying 195 animals is an easier task compared with a net for 398 animals. We observed positive effects both on convergence speed and the quality of local minimum obtained. More analysis is needed because curriculum learning study is currently empirical.

Results on ImageNet_22K_Animal.

When class number grows from 1000 to 2282, we split the superclass and do a clone increment from the 1000-animal-model. The branch model and the two leaf models are all incrementally trained by copying exiting parameters as initialization. Figure 5 shows the confusion matrices of the existing 1000 and the new 1282 classes before clone increment. All these classes are partitioned into two superclasses with 1350 and 932 classes, respectively (shown in red and blue boxes in Figure 5). One can observe the similar error distribution on the old and new classes.

Table 3: Incremental training result in ImageNet_22K_Animal dataset

Increment	Training	Error Rate	Examples
flat	from-scratch	49.46%	74.74M
flat	incremental	49.15%	**61.45M**
clone	incremental	**48.52%**	117.79M

Table 3 shows advantage of the incremental learning over other training strategies. Column `Increment` shows how the model capacity are extended, flat increment or clone increment. Column `Training` tells how the new models are trained, from-scratch or incremental. Column `Examples` accounts the training examples to reach that accuracy. Note that if the model has several parts, the examples of each part are summed up to fill the entry. Compared with the baseline (the first row) with flat increment and from-scratch learning, the flat increment combining incremental learning (the second row) performs slightly better, with 0.3% accuracy gain and 18% fewer training examples. Finally, our increment (the last row), with more extra capacity from clone, gains some more (48.52% versus 49.46% of baseline) but with

Table 1: Architecture of the baseline model

Layer	1	2	3	4	5	6	7	8
Type	conv+max+norm	conv+max+norm	conv	conv	conv+max	full	full	full
Channels	96	256	384	384	256	4096	4096	Output Dim
Filter Size	11*11	5*5	3*3	3*3	3*3	-	-	-
Convolution Stride	4*4	1*1	1*1	1*1	1*1	-	-	-
Pooling Size	3*3	3*3	-	-	3*3	-	-	-
Pooling Stride	2*2	2*2	-	-	2*2	-	-	-
Padding Size	2*2	1*1	1*1	1*1	1*1	-	-	-

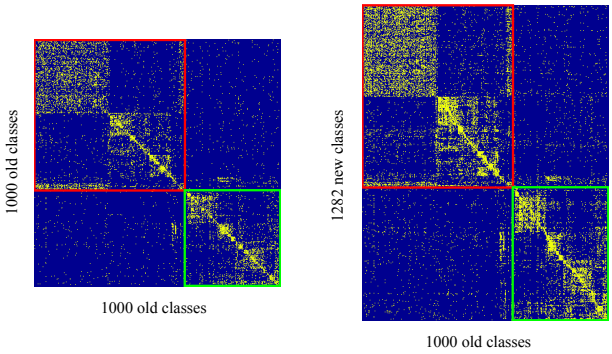

Figure 5: Confusion matrix among 1000 original classes (left), and the confusion matrix between new 1282 classes and original 1000 classes (right). The partitioning has been done (in red and blue boxes, respectively).

substantially more data training the leaf models. The total amount of time needed won't increase if trained in parallel.

To understand the benefit of clone increment, we do a breakdown analysis in Table 4 and Table 5. Flat-incremental model achieved 49.15% overall error rate, in which the superclass 1 and 2 have 45.79% and 56.02% error rates, respectively. If the branch model is perfect, the two leaf models obtain 44.43% and 54.79% error rates, gaining more than 1.2% accuracy each. Accounting the prediction error of the branch model, the 2-level model achieves 45.03% and 55.63%.

Table 4: Breakdown comparison between cloned models and the flat-incremented model

Model	Superclass 1	Superclass 2
cloned leaf_1	44.43%	N/A
cloned leaf_2	N/A	54.79%
cloned branch	* 2.20%	* 4.11%
cloned leafs+branch	45.03%	55.63%
flat-incremented model	45.79%	56.02%

The end performance on a given superclass can *not* be derived from the average accuracy of a leaf model (i.e. 55.56% superclass 1) and the branch accuracy (i.e. 97.80%). For superclass 1, this will yield the error rate to be $1 - 0.5556 \cdot 0.9780$, which is 45.66%. This is higher than we actually get (45.03%). Why is that?

The reason is that the images misclassified by the branch are also more difficult to be classified correctly by the leaf model. Table 5 shows the error rate if we divide the data according to the branch decision. It is clear that there exists such a strong correlation. This is both subtle and intuitive, and explains the apparent discrepancy described earlier. The correct way to compute the performance on a given superclass is:

Table 5: Conditional Leaf Model Error

SuperClass	Branch Correct	Branch Miss
SuperClass_1	43.79%	72.89%
SuperClass_2	53.72%	79.76%

$$E_{whole} = 1 - (1 - E_{L|B_{Correct}}) * (1 - E_B) \quad (1)$$

The conditional probability that an image is correctly classified given that it is correctly predicted by the branch $(1 - E_{L|B_{Correct}})$ is not equal to and higher than its average $(1 - E_L)$. Improving the branch model will help, but with a diminishing return.

5.2 Leaf Model Learning Analysis

In our architecture, each leaf model is cloned to deal with one superclass, which includes part of the old classes and some extra new classes. In this section we perform a detailed analysis, showing why initialization with existing parameters is advantageous, and furthermore how new features fight a continuous "tug-of-war" with the old ones. The study is on the 1350 superclass partition, which includes 602 old classes and 748 new ones.

5.2.1 *Advantage of Incremental Learning*

In Figure 6, we compare the error curves of incremental learning versus one that is trained from scratch. It is obvious that the error rate of incremental learning drops much faster. Such speedup benefits are the results of bigger gradients on weights, which subsequently relate to activations in the forward pass, and derivatives in the backward pass.

Figure 7 shows the ℓ^2 norms of the weight gradients between different layers as a function of training epochs. To save space, only the gradient between layer 7 and 8, and that between layer 1 and 2 are shown, as representatives of classification layers and low feature extraction layers, respectively. The gradient curves between other layers have similar trends. In the first 150 batches, the incremental learning produces much bigger gradient than the from-scratch learning in both high and low layers. This explains why error rate of the incremental learning drops quickly in the same period, as it is able to descend the error surface faster. As training progresses to later epochs, the differences between incremental and from-scratch gradually disappears.

What leads to the bigger gradients? In back-propagation, the gradient of the weights between layer $i-1$ and i, $\mathbf{g}_{i-1,i} \equiv$

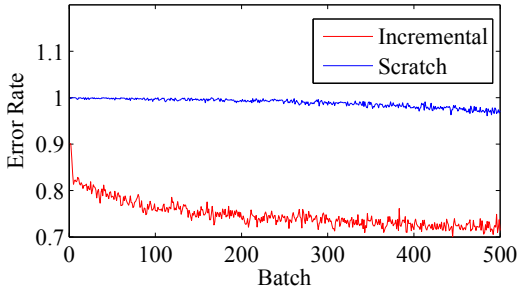

Figure 6: Learning curves of incremental run and scratch run in the first 500 batches

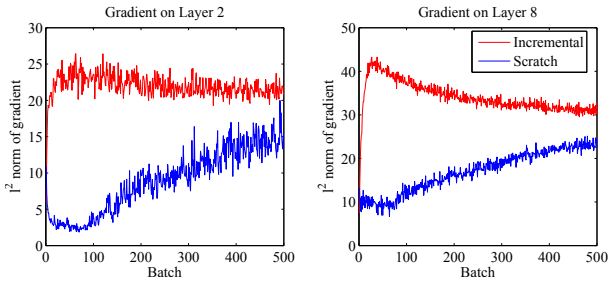

Figure 7: l^2 norm of the gradient in layer 2 and layer 8 in the incremental run and the scratch run

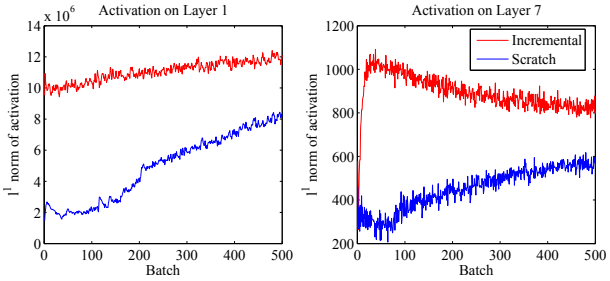

Figure 8: l^1 norm of the activation in layer 1 and layer 7 in the incremental run and the scratch run

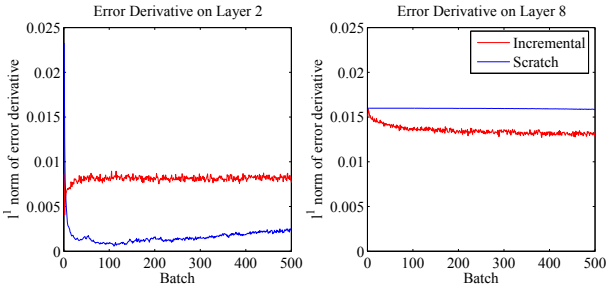

Figure 9: l^1 norm of the error derivative in layer 2 and layer 8 in the incremental run and the scratch run

$\frac{\partial E}{\partial W_{i-1,i}}$, is the outer product of $\mathbf{x}_{i-1}$, the activation of layer $i-1$ computed in the forward pass, and $\mathbf{e}_i$, the error derivatives of layer i computed in the back propagation pass. In other words, $\mathbf{g}_{i-1,i} = \mathbf{e}_i \cdot \mathbf{x}_{i-1}^T$. For example, $\mathbf{g}_{7,8} = \mathbf{e}_8 \cdot \mathbf{x}_7^T$.

First, we dive into the activations. In Figure 8, we plot the ℓ^1 norm of the activations in layer 7 and layer 1. We can see that in the feature extraction layers (e.g. layer 1), the activations all rise, signaling the learning and activation of new features. The one with incremental learning starts with bigger values (roughly five times bigger) and increase slowly, the reason is that incremental learning is able to reuse existing features and is therefore less motivated to learn new ones. In the classification layer, activations of incremental learning are also much higher, since from-scratch are gradually moving forward with random (and therefore lower) activations.

Next, we look at error derivatives to check how the error signals are propagated down by the back-propagation algorithm. The last layer's error derivative $\mathbf{e}_8$ is simply the gap between label and output. While from-scratch is essentially making random guesses and suffering from high errors, incremental learning at least are getting most of the predictions of the old classes correct. As such, higher classification layers in from-scratch learning has bigger errors to start the propagation, as shown in Figure 9. Things are different for lower layers. The back-propagation is governed by

$$\mathbf{e}_i = \frac{\partial \mathbf{x}_i}{\partial \mathbf{z}_i} \odot W_{i,i+1}^T \cdot \mathbf{e}_{i+1} \tag{2}$$

and a series of randomly initialized weights of the layers diminishes the error derivatives towards lower layer, as is the case of from-scratch learning. Weights in the incremental learning start with structures that allow the prediction of existing classes, and thus allow easier propagation of error signals. As such, in lower layers (such as layer 2) the error derivatives are larger.

The above observations can be summarized as the followings. Incremental learning has higher activation due to learned features being reused, which also allows errors to propagate more smoothly downwards. Neither is true for from-scratch learning, whose only drive is the bigger error at the output layer. As such, incremental learning has bigger gradients and descent faster.

5.2.2 *Interference between new and old classes*

By the virtue of the error-driven preview and the subsequent partitioning, classes within a superclass is similar. Since the cloned leaf model was originally trained on old classes (some of them have been moved to another superclass), at the beginning of the training all new classes are misclassified as old ones (see Figure 10). Figure 12 is an illustration of some new classes being classified into similar old classes.

The leaf model slowly learn the new classes. Meanwhile, the prediction accuracy of old classes recovers from some damages. As it turns out, such interference is the driving force for low layers to learn new features that helps separate new classes from old ones. We offer a detailed analysis below.

To make things simple, we will only concentrate on the interference between a pair of similar classes, one old and one new. As illustrated in Figure 11, y_1, t_1 and $\mathbf{w}_1$ are the output, target, output weight vector for the old class unit respectively, as are y_2, t_2 and $\mathbf{w}_2$ for the new class unit. At the beginning of incremental training, $\mathbf{w}_1$ is copied from the

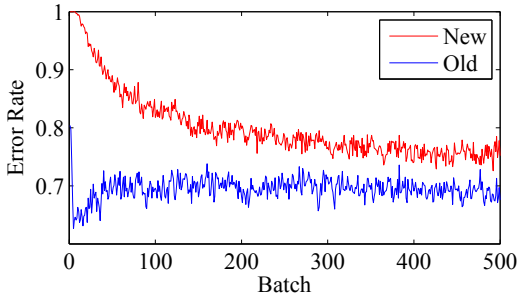

Figure 10: Learning curves of old and new classes.

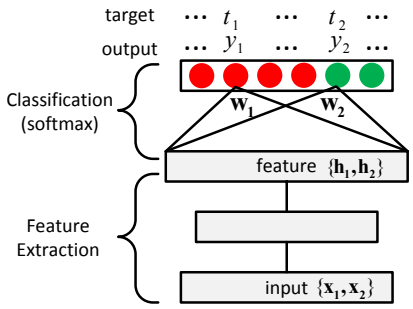

Figure 11: Inference between new and old classes. Input $\mathbf{x}_1$ with feature $\mathbf{h}_1$ belongs to old classes (target $t_1 = 1$); $\mathbf{x}_2$ with $\mathbf{h}_2$ belongs to new classes (target $t_2 = 1$).

previous training result, and $\mathbf{w}_2$ is just randomly initialized. Given an input sample vector $\mathbf{x}_1$ in old class, the last hidden layer outputs its feature vector $\mathbf{h}_1$. Since the model is well trained on old classes, $y_1 = t_1 = 1$. Because softmax output $y_1 = \frac{e^{\langle \mathbf{w}_1, \mathbf{h}_1 \rangle}}{\sum_k e^{\langle \mathbf{w}_k, \mathbf{h}_1 \rangle}}$, this correct prediction is due to a big inner product $\langle \mathbf{w}_1, \mathbf{h}_1 \rangle$. Without any error on this sample, back-propagation doesn't make any update on weight $\mathbf{w}_1$ and $\mathbf{w}_2$.

Instead, $\mathbf{x}_2$, being similar to x_1, is wrongly predicted as the old class because of the big inner product $\langle \mathbf{w}_1, \mathbf{h}_2 \rangle$. That means that, unsurprisingly, h_1 and h_2 are highly correlated, with a big inner product $\langle \mathbf{h}_1, \mathbf{h}_2 \rangle$. Back-propagation starting from the mismatch between output ($y_1 = 1$, $y_2 = 0$) and the target ($t_1 = 0$, $t_2 = 1$) makes update on both $\mathbf{w}_1$ and $\mathbf{w}_2$,

$$\begin{aligned} \mathbf{w}_1' &= \mathbf{w}_1 - \epsilon \cdot \mathbf{h}_2 (y_1 - t_1) = \mathbf{w}_1 - \epsilon \cdot \mathbf{h}_2 \\ \mathbf{w}_2' &= \mathbf{w}_2 - \epsilon \cdot \mathbf{h}_2 (y_2 - t_2) = \mathbf{w}_2 + \epsilon \cdot \mathbf{h}_2 \end{aligned} \tag{3}$$

where ϵ is the learning rate. It effectively "shaves" some weights off $\mathbf{w}_1$ and moves them to $\mathbf{w}_2$, and damages $\mathbf{w}_1$ in the sense that if the model sees $\mathbf{x}_1$ again, y_1 is a little less to be turned on. That's formulated by

$$\langle \mathbf{w}_1', \mathbf{h}_1 \rangle = \langle \mathbf{w}_1, \mathbf{h}_1 \rangle - \epsilon \cdot \langle \mathbf{h}_2, \mathbf{h}_1 \rangle \tag{4}$$

So the stronger the correlation between $\mathbf{h}_1$ and $\mathbf{h}_2$, the more damage it is. As the training proceeds by alternating between these two samples, the net result is a compromising accuracy for either one of the class. To see why this is so, consider a state when both classes are correctly predicted. This state cannot be a stable state as the correlation between $\mathbf{h}_1$ and $\mathbf{h}_2$ will cause interferences that will weaken their weights in turn.

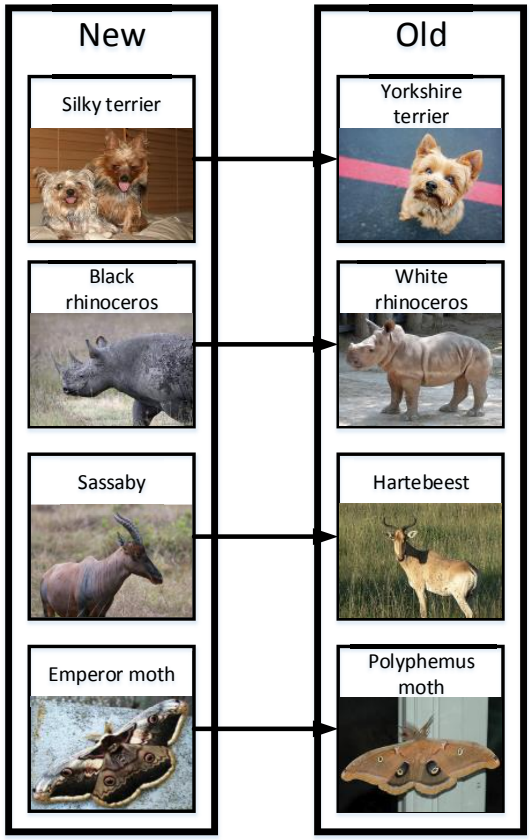

Figure 12: Examples of some new classes being misclassified into similar old classes at the beginning of training.

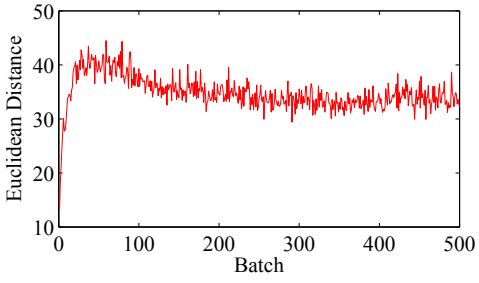

Figure 13: Changing of feature distance between new class *'silky terrier'* and old class *'yorkshire terrior'*.

That means, however, that the back-propagation, being optimum-pursuing, will eventually separate $\mathbf{h}_1$ and $\mathbf{h}_2$ by forcing the error propagates downwards. As shown in Figure 13, old class *'yorkshire terrior'* and new class *'silky terrier'* have very close features at the beginning of training, with the latter misclassfied as the former. Quickly, their feature distance widens, driven by the interference in classification layer to improve the prediction accuracy. The growth of the

capacity for this fine-grained feature learning is hopefully stolen from classes that are moved away into other superclass (and not seen by the model any more).

6. CONCLUTION

We believe that tackling large-scale image classification task can and should take a divide and conquer approach, and do so incrementally. This study establishes a framework to do so with a training procedure that grows the network capacity hierarchically. The overall performance numbers are positive, but not overwhelmingly so. We take this as a steppingstone and the experiences point out a few lessons and future directions, which we summarize as the followings:

- Cloning naturally inherits leaned features. However, there is a tug-of-war between old features and new features, in addition to removing old features that were acquired on removed classes. The current way of training with vanilla back-propagation does not seem to be efficient.
- At the time of performing superclass partition, we have a fair amount of knowledge of errors within each partition. Cloning with the exact same configuration simplifies parameter initialization, at the expense of missing needed capacity increase.
- The branch model should focus on differentiating superclasses, our current training model deprives it from such opportunities.

We are actively pursuing the above directions.

7. ACKNOWLEDGMENTS

This work was supported by National Hi-Tech Research and Development Program (863 Program) of China under Grants 2014AA015102 and 2012AA012503, National Natural Science Foundation of China under Grant 61371128, and Ph.D. Programs Foundation of Ministry of Education of China under Grant 20120001110097.

8. REFERENCES

[1] Y. Aytar and A. Zisserman. Tabula rasa: Model transfer for object category detection. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2252–2259. IEEE, 2011.

[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[3] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei. What does classifying more than 10,000 image categories tell us? In *Computer Vision–ECCV 2010*, pages 71–84. Springer, 2010.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[5] R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *Neural Networks, IEEE Transactions on*, 22(10):1517–1531, 2011.

[6] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[7] L. Fe-Fei, R. Fergus, and P. Perona. A bayesian approach to unsupervised one-shot learning of object categories. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1134–1141. IEEE, 2003.

[8] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):594–611, 2006.

[9] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[10] I. J. Goodfellow, M. Mirza, X. Da, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgeting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[11] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. 2007.

[12] G. Griffin and P. Perona. Learning and using taxonomies for fast visual categorization. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[13] J. Hoffman, E. Tzeng, J. Donahue, Y. Jia, K. Saenko, and T. Darrell. One-shot adaptation of supervised deep convolutional models. *arXiv preprint arXiv:1312.6204*, 2013.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4, 2012.

[15] I. Kuzborskij, F. Orabona, and B. Caputo. From n to n+ 1: Multiclass transfer incremental learning. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3358–3365. IEEE, 2013.

[16] C. H. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 951–958. IEEE, 2009.

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

[18] Z. Lu and Y. Peng. Image annotation by semantic sparse recoding of visual content. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 499–508. ACM, 2012.

[19] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *ECCV*, pages 143–156. Springer, 2010.

[20] C. Snoek, K. van de Sande, D. Fontijne, A. Habibian, M. Jain, S. Kordumova, Z. Li, M. Mazloom, S. Pintea, R. Tao, et al. Mediamill at trecvid 2013: Searching concepts, objects, instances and events in video. In *NIST TRECVID Workshop*, 2013.

[21] S. Thrun. Is learning the n-th thing any easier than learning the first? *Advances in neural information processing systems*, pages 640–646, 1996.

[22] T. Tommasi, F. Orabona, and B. Caputo. Safety in numbers: Learning categories from few examples with multi model knowledge transfer. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3081–3088. IEEE, 2010.

[23] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3360–3367. IEEE, 2010.

[24] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.