Incremental Learning of Random Forests for Large-Scale Image Classification

Shuan Guo saguo@umich.edu

Ruobai Feng rbfeng@umich.edu

Takeshi Kondoh takekon@umich.edu

Zhaoming Zhang zmzhang@umich.edu

1 Introduction

Billions of images being produced every day, it is impossible to tag each image by hand. Image classification aims to solve this problem automatically. Offline classification methods assume a static setting where the number of training images is fixed as well as the number of classes. However, these methods are not suitable for large-scale data sets or increasing data sets. The rapid expansion of the number of images over social networks calls for new dynamic classification methods suitable for adding new classes of images. M Ristin, M Guillaumin and J Gall [1] proposed a new method that learns dynamically on an increasing input data set with feasible performance. They first build random forest classifier with initial input data, then they use different incremental learning strategies to update the random forest with new inputs. The updating process is incremental in a sense that we can actually reuse the previous training results, that the training time can be significantly reduced.

Classification method aiming at dynamic large-scale scenario, where the number of classes as well as the number of images gradually increase and reach large numbers, have been proposed. In [2] Nearest Class Mean (NCM) classification is applied on the initial set of classes, and new classes can be added using their feature means. An alternative multi-class incremental approach based on least-squares SVM has been proposed in [3] where for each class a decision hyperplane is learned. However, every time a class is added, the whole set of the hyperplanes has to be updated, which is expensive as the number of classes grow.

Random forests (RF), as Ristin et al. suggested, is also suitable for the task of learning incrementally new classes. RFs are intrinsically multiclass and hierarchical classifiers, 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[1]. As an result, one can update the classifiers by keeping the higher level untouched and update the deeper levels, and the more local features and less data can be handled very efficiently. Based on this idea, Ristin et al studied two variants of RFs with different classifiers as their building blocks, i.e. NCM and SVM, and compared a few strategies to update the trees for incremental learning.

2 Problem Statement and Methods

Our group is aiming at the large-scale incremental learning problem. Incremental means that the size of total data actually increase over time, which is often the case in real life. Data is collected progressively, and new classes may also appear as the total data size grows. Considering the increasing size of data, retrain the whole classifier could become forbiddingly time consuming. In this project, we studied two variants of Random Forests, NCM forest and SVM forest, and evaluate how random forests perform with three strategies to incorporate new classes while avoiding to retrain the Random Forests from scratch.

2.1 SVM Forest and NCM Forest

Random Forest is an ensemble of many decision trees where each tree is trained independently with certain randomness, and the output of a random forest is the majority vote of all the decision trees in the forest. Random forests outperform a single decision tree in a sense that it is more stable, less likely to overfit, and have smoother classifications. The tree takes input data in a form of d-dimensional vectors $\boldsymbol{x} \in \mathbb{R}$, and at each node n the training data $\boldsymbol{S}_n$ arriving to that node is divided into two subsets and passed to child nodes. For conventional random forest, a simple split function is often used, where the data is randomly split by one the features. The performance of RF highly dependent on the choice of splitting. Usually a random set of splitting functions is generated at each node but only the best one is selected according to the information gain.

$$\begin{aligned} \text{Gain} &= i(n) - [p(n_1)i(n_1) + p(n_2)i(n_2)] \\ i(n) &= -[q \log q + (1 - q \log(1-q))] \end{aligned}$$

Another decision to make when growing the tree is when to stop growing. One of the method is mentioned in class that we can first fully grow the tree until each leave node consists of only one class, and apply pruning method to avoid overfitting. The method used by the paper is that they stop growing the tree when there is less data element in a single node than a threshold. At every leaf node, a probability distribution of labels is derived from the member elements of that node, and the forest sum that distribution from each trees and assign the label to the one with largest probability.

2.1.1 Combine SVM with random forest

SVMs are well suited binary classifiers to serve as splitting functions in RFs. Each node n of the forest will be associated with its own hyperplane $\boldsymbol{\omega}$ and we can simply use $\langle \omega, \boldsymbol{x} \rangle$ to decide whether a sample is passed to the left child node or to the right child node, which is fast in term of testing time. To train a node n, all $\boldsymbol{x}_i$ from a class k observed at node n are assigned the same random label $\boldsymbol{y}_i = \boldsymbol{y}_k \in \{-1, 1\}$. An SVM is then trained by the labels to learn a single splitting function. The assignments of classes to the two labels serves as the source of randomness. A number (20 in the paper) of splitting functions is generated and the optimal one is picked by information gain. Each time the label is reassigned, the SVM must be retrained. This becomes the major penalty for the training time of SVM random forest (SVMF).

2.1.2 Combine NCM with random forest

Nearest Class Mean 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. To combine NCM with RF, a few modifications are made. First, at each node, we only use a subset of the classes to train NCM, speeding up the training process. Second, the labeled data is further assigned to $\boldsymbol{y}_i = \boldsymbol{y}_k \in \{-1, 1\}$ randomly for a binary split. The randomness occurs only for the label reassignment that we can reuse the classification result from NCM. This greatly save time compared to SVMF. From all the label assignments (1024 in the paper), the optimal one is chosen according to the information gain.

2.2 Incremental strategy

Many researchers have studied on-line learning of Random Forests for applications such as object detection (Andreas Opelt, et al.)[4], or segmentation (Olga Barinova, et al.)[5]. However, many works assume that the number of classes is known. Our project focuses on incrementally add new classes to the forest for large-scale image classification. With a proper incremental learning method, one does not need to retrain Random Forests from scratch when new training data with new classes are added. This would save a lot of computation time especially the amount of data increases. We use three strategies for incremental learning which are proposed in [1]. In this section, we will explain those three strategies and the concept of node sampling which is used for the last strategy.

2.2.1 Update Leaf Statistics (ULS)

Every incremental strategy assume that a multiclass RF (we call this baseline) has been already trained for the set of $\mathcal{K}$ of classes. When training data of a new class κ' are added, Update Leaf Statistics (ULS) passes the training data of the new class through the initial RF and updates the class probabilities stored at the leaves. This strategy does not change the splitting function nor the size of the initial RF. The illustration of ULS is drawn in Fig.1.a). Since the structure of the initial RF is not changed, this strategy is useful only for situations where the initial RF is already complex enough to deal with new class. If the initial RF was obtained with fewer initial training data, the splitting functions would overfit to the initial training data and the resulting new RF would perform poorly because the initial RF could not produce a meaningful hierarchy for the new class. We tried different size of initial classes and compare the accuracy and computation time.

2.2.2 Incrementally Grow Tree (IGT)

Incrementally Grow Tree (IGT) also passes new training data of a new class through the initial RF. However unlike ULS, if enough data of the new class reach a leaf, the RF grows again and new splitting nodes can be added. The newly added splitting functions are trained using the data from $\mathcal{K} \cup \kappa'$. The illustration of IGT is drawn in Fig.1.b). Like ULS, we tried different size of initial classes.

2.2.3 Retrain Subtree (RTST)

ULS and IGT never produce a RF trained on the all data from $\mathcal{K} \cup \kappa'$ because the tree structure trained on the initial data from $\mathcal{K}$ is not changed. Retrain Subtree (RTST) updates some of the splitting function of the initial RF. To this end, subsets of subtree in the RF trained on the initial data from $\mathcal{K}$ are converted into leaves with all of their children removed. Then the RF grows again from these leaves, using the data from $\mathcal{K} \cup \kappa'$ stored in these leaves in the same way IGT does. The illustration of RTST is drawn in Fig.1.c). The next subsection further explains the distribution of the probability that a node is converted into a leave for retraining. We implemented two kinds of probability distributions for RTST.

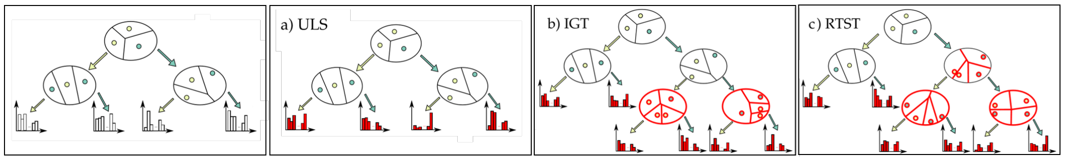
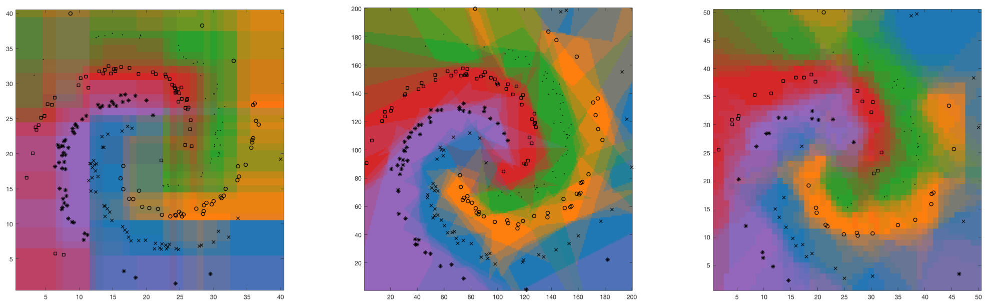


Figure 1: Illustration of baseline and three incremental strategies [1]. a) ULS does not change the splitting function of the baseline, but changes the probability distributions of the terminal nodes. b) IGT does not change the splitting function of the baseline, either, but IGT grows trees from the terminal nodes of the baseline, using both initial and newly added training data. c) RTST converts some subsets of subtrees of the baseline to the leaves and then grow trees, using both initial and newly added training data. Each subtree has a chance to be converted with certain probability.

2.2.4 Node Sampling for Partial Tree Update

Updating a splitting node during RTST could update the whole subtree. Therefore we adapt two different probability distributions [1] that are used to select a node or subtree for retraining.

- *Uniform.* Equal probability: $\frac{1}{N}$, is assigned to each splitting node, where N is the number of splitting nodes of the initial RF. We let RTST denote RTST using this probability.
- *Quality.* We calculate the quality of a subtree with root node n and its leaves by information gain from its root to the leaves. Let S^n denote the data of classes $\mathcal{K} \cup \kappa'$



(a) split (b) SVMF (c) NCMF

Figure 2: Random forest classifier with different random split functions

reached at the node n and S^l denote the data reached at a leaf l. The quality is defined as:

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

where H denotes the entropy. Then a node is selected only if the information gain is greater than zero. The probability is then inversely proportional to the quality, which means that subtrees which perform poorly are more likely to be selected for retraining. We let RTSTQ denote RTST using this probability.

3 Experiments

In this section, the implemented random forests are tested against sample spiral data for sanity check, and tested on real features extracted from images.

For the usual incremental learning problem, the size of data is tremendous and the number of classes can be as large as several hundred. However, our computation power is limited to desktop machines and is not capable of handling very large data set, so we managed to reduce the size of data while preserving the general idea. All of these algorithms were implemented in Matlab 2016a.

3.1 Experiment with simulated data

We first apply the implemented random forest on simulated data (as presented in Fig.2) - spiral data. The data was generated with equation

$$\begin{aligned} r &= \text{random number} \\ y &= \text{random label} \\ \varphi &= 2\pi y/\text{total label} + 0.5 * r + \text{fluctuation}; \end{aligned}$$

The whole dataset is randomly permutated and 70% of the data is set as training data while the remain 30% test data. After training, a colored map is generated to illustrate the classifier (different splitting functions), as is shown in Fig.2. For each pixel with coordinate (x, y), the color is decided by its probabilities of being assigned to every label. The edge of colors suggests characteristics of splitting functions of the decision tree, i.e. simple split, SVM and NCM. SVM and NCM methods outperform simple split as they have much smoother classification boundaries.

3.1.1 SVMF vs. NCMF

In order to compare the performance of SVMF and NCMF. We generated a dataset with 20 classes, including 500 training samples and 100 test samples per class. We stop the growing of trees when the best gain is less than 0.001 (determined through efforts). Since the SVM hyperplanes require relative long time for calculation, the training time of SVMF is extremely long (1762.05s). While the training of NCMF is much more faster (82.48s). For NCMF, the complex calculations for different centroids were done before repeats for finding best gain and only need once per node. However, the complex calculations while refreshing the parameters of SVM are required at each iteration in order to obtain a best splitting strategy. The test time for SVMF (1.92s) is also longer than the test time of NCMF (0.36s). Making a prediction from different SVM parameters may be a slower process for Matlab than the calculations of different distances to each centroid. As for the accuracy, the training error of SVMF (0.0177) and NCMF (0.0098) are close. Both are low enough. The test error of SVMF (0.036) and NCMF (0.0215) are close, too, though NCMF is slightly better. Overall, NCMF is more preferable in both performance and time efficiency in such a case.

3.1.2 Compare the incremental strategies

We tested the incremental learning algorithms with spiral data. Initially the forest is trained on 4 classes, and we continued to add 2 new classes each time until the number of classes reaches 20. The relative performance, test time, and training time are compared for NCMF with three incremental strategies and baseline (retrain the whole forest). Here performance is defined as the number of correctly classified data divided by the total number of data, and the relative performance is the the performance divided by baseline performance. Fig.3 shows the comparison result.

It is observed that the relative performance of IGT, RTST and RTSTQ are only slightly degenerated from baseline. Test time and training time of these three methods are similar: there is little difference between these three methods as for efficiency. All of them have obvious higher time efficiency than the baseline. Although the performance of ULS significantly decreases as new classes are added, because the initial RF were trained on only 4 classes and the ULS method did not update the forest. It could not produce a meaningful hierarchy for the new added class.

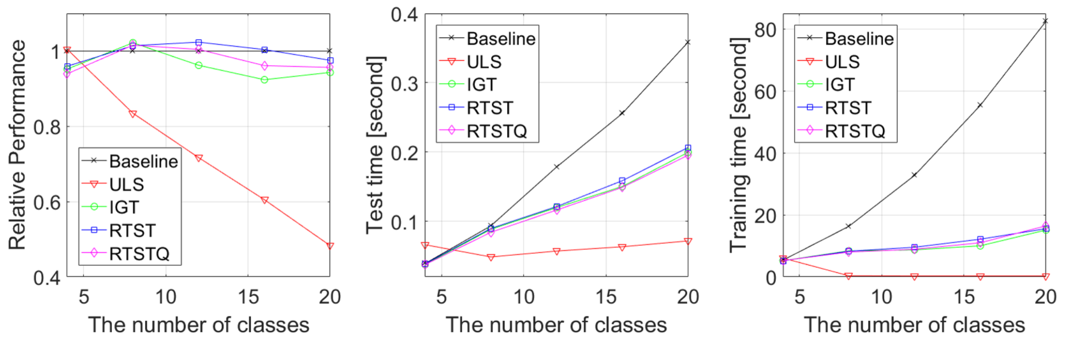


Figure 3: Measurement at variable number of classes for an incremental training of NCMF. The relative performance of IGT,RTST and RTSTQ are similar and close to the performance of baseline (the relative performance of baseline is 1, which is not shown in the plot), while the relative performance of ULS decreases as the number of classes grows. Both the test time and training time of IGT, RTST and RTSTQ behave similarly, while those of ULS changes very little because it does not change or add splitting function, but just change the probability distribution of the terminal nodes.

The incremental learning strategies are applied to SVMF, too. The results for the comparison of both variants show in Fig.4. These measurements are all performed at the final 20 classes. Except for the invariant relative poor performance for ULS in both SVMF and NCMF, baseline and the rest of incremental strategies show consistency in similarly high accuracy. For IGT, RTST, and RTSTQ, test time and training time are stable across the forest variants. ULS still exhibits the highest time efficiency in both NCMF and SVMF.

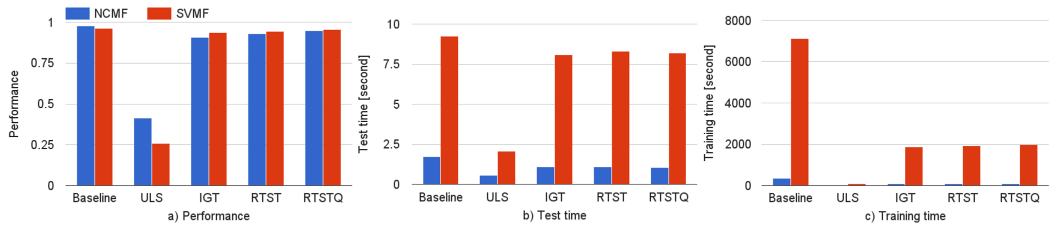


Figure 4: Measurement at 20 classes starting with 4 initial classes for various incremental learning methods and forest variants. The performance of both NCMF and SVMF are similar and close to the baseline except when ULS is used for the incremental strategy. Test time of IGT, RTST and RTSTQ (both NCMF and SVMF) are a little shorter than the baseline, while ULS can significantly reduce test time. On the other hand, training time of all incremental methods are less than 30% of that of the baseline.

3.1.3 Concerning initial class number

Considering the possible influence of the initial class size on the different incremental learning methods. We compared the performance of both SVMF and NCMF combined with different incremental learning approaches training starting from different number of initial classes in Table 1. As we can see, ULS is the special case, which shows increasing accuracy with the increase of the initial class size.(e.g. NCMF+ULS, the performance are respectively: 0.41, 0.56, 0.68 from 4, 6, 10 to 20 classes; SVMF+ULS are 0.26, 0.39, 0.57) Other incremental approaches are quite insensitive to the number of initial classes and all achieve good performance. This illustrates the high initial classes dependency property of ULS. Since during the incremental process, the structure of the initial random forest of ULS does not change, the majority of bias should come from the already trained initial classes. Also, since this method maintains poor performance, the slightly changes of errors could be more obvious than other methods.

Table 1: Comparison of baselines and different incremental learning methods measured at various initial classes, all starting with the k initial classes. The classification accuracy is reported in the cell, while relative performance to the corresponding baseline is given as percentage in brackets. The forests were learned on k initial classes in b), c) & d). Incremental methods were trained with batches of 2 classes. NCMFs and SVMFs with RTST consistently outperform ULS for incremental learning.

	method \ # of classes	12	16	20
a) baseline	SVMF	0.97	0.96	0.96
	NCMF	0.95	0.95	0.98

	method \ # of classes	12	16	20
b) $k = 4$	SVM+ULS$_2$	0.38(39.1%)	0.31(32.4%)	0.26(26.7%)
	SVM+IGT$_2$	0.96(98.9%)	0.92(95.6%)	0.94(97.4%)
	SVM+RTST$_2$	0.89(92.3%)	0.91(94.1%)	0.95(98.1%)
	SVM+RTSTQ$_2$	0.93(95.9%)	0.93(97.2%)	0.96(99.3%)
	NCMF+ULS$_2$	0.58(61.2%)	0.48(50.3%)	0.41(42.4%)
	NCMF+IGT$_2$	0.89(94.3%)	0.88(92.3%)	0.91(92.9%)
	NCMF+RTST$_2$	0.94(98.8%)	0.90(94.6%)	0.93(94.9%)
	NCMF+RTSTQ$_2$	0.92(96.7%)	0.92(97.0%)	0.95(97.0%)

	method \ # of classes	12	16	20
c) $k = 6$	SVM+ULS$_2$	0.58(60.4%)	0.47(49.0%)	0.39(40.1%)
	SVM+IGT$_2$	0.93(96.4%)	0.92(96.1%)	0.94(97.9%)
	SVM+RTST$_2$	0.97(100.3%)	0.96(100.2%)	0.95(99.0%)
	SVM+RTSTQ$_2$	0.95(98.5%)	0.95(98.8%)	0.96(99.4%)
	NCMF+ULS$_2$	0.72(75.5%)	0.67(70.2%)	0.56(57.2%)
	NCMF+IGT$_2$	0.97(102.6%)	0.92(96.7%)	0.92(93.6%)
	NCMF+RTST$_2$	0.94(99.6%)	0.93(97.5%)	0.94(95.7%)
	NCMF+RTSTQ$_2$	0.97(102.0%)	0.92(96.5%)	0.94(96.3%)
d) $k = 10$	SVM+ULS$_2$	0.83(85.9%)	0.69(71.3%)	0.57(58.7%)
	SVM+IGT$_2$	0.90(92.5%)	0.92(95.4%)	0.94(97.9%)
	SVM+RTST$_2$	0.97(100.1%)	0.96(100.3%)	0.96(100.1%)
	SVM+RTSTQ$_2$	0.97(100.0%)	0.96(100.3%)	0.96(99.8%)
	NCMF+ULS$_2$	0.90(94.7%)	0.76(79.9%)	0.68(69.7%)
	NCMF+IGT$_2$	0.96(101.6%)	0.96(100.3%)	0.95(97.0%)
	NCMF+RTST$_2$	0.93(98.5%)	0.96(100.3%)	0.96(97.7%)
	NCMF+RTSTQ$_2$	0.98(102.9%)	0.97(101.4%)	0.96(97.7%)

3.1.4 Batch size

We studied how the incremental learning strategies are able to cope with a different batch size. This is of great interest because in practice multiple classes would be added at once. We trained the initial RF with 4 classes and incrementally updated it with batches of 2,4 and 8 classes. As shown in Fig.5, both test time and training time reduce by training several classes at once, while relative performance does not depend on chunk size.

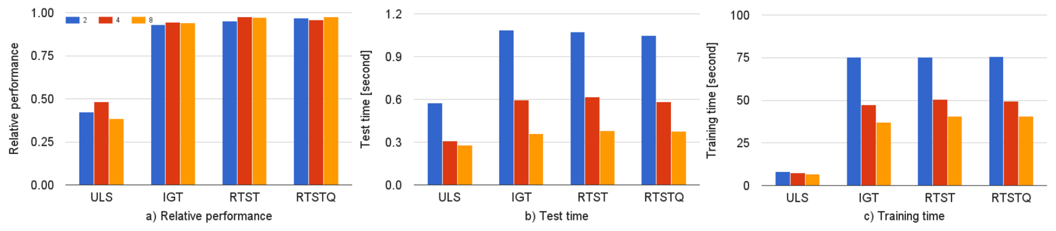


Figure 5: Measurement at variable number of classes for an incremental training of NCM forest. The initial RF is trained on 4 classes, and the batch size of 2, 4, and 8 are tested. The batch size has little impact on the relative performance. Both test and training time decrease as the batch size becomes larger.

3.2 Experiment with real data

We use a 1k visual words benchmark, namely "ImageNet Large Scale Visual Recognition 2010 challenge benchmark (ILSVRC10)" [6] for the experiment. It consists of 1k categories, between 668 and 3,047 training images per class and 1.2M in total, and 150 testing images per category. We selected 100 different categories of synset with each containing 500 data for training, 50 data for validation and 100 data for testing.

3.2.1 Feature extraction

Each data is a bag-of-words representation of a real image. To this end, first each image is resized to have a max side length of 300 pixel and then SIFT descriptors are computed on 20x20 overlapping patches with a spacing of 10 pixels. Scale-invariant feature transform (SIFT) [7] is a method in computer vision that produces descriptors of local scale-invariant features in images. After that, a visual vocabulary of 1000 visual words was formed by performing a k-means clustering on SIFT descriptors. Finally, Each SIFT descriptor is quantized into a visual word using the nearest cluster center.

3.2.2 Comparision of the update strategies

We also tested the incremental learning algorithms for real images. The experimental condition is exactly the same with the one for spiral data. As shown in Fig.6, the relative performance of every method decreases with adding new classes. Since images from different classes are more diverse than spiral data, it is harder for the initial RF trained on only 4 classes to cope with new classes. ULS performs poorer than other three methods although test time and training time are shorter. The relative performance of IGT looks slightly poorer than RTST and RTSTQ while test time and training time are shorter. RTST and RTSTQ behave similar in all aspects. Thus the trade-off relation between the accuracy and the computation time are validated. The comparison of NCMF and SVMF are shown in Fig.7, trained on 10 classes. There are observed performance differences between NCMF and SVMF when they are applied for real data. For all incremental learning methods, NCMF outperforms SVMF. The cost of this is that both training and test time of SVMF are longer than that of NCMF.

4 Conclusions

In this research, we studied how two variants of Random Forests (SVMF/NCMF) perform under three incremental learning strategies to incorporate new classes while avoiding to retrain the Random Forests from scratch. We implemented the algorithm for combining Random Forests with SVM and NCM (Nearest Class Mean Classifier). Also, we implemented three incremental algorithms based on SVMF/NCMF, namely 'Update leaf statistics' (ULS), 'Incrementally grow tree'(IGT), and 'Retrain Subtree'(RTST/RTSTQ), respectively. We evaluated these algorithms on a simulated dataset and a real image dataset (2010 challenge benchmark, ILSVRC10).[6] As we have shown, both variants of RF based on NCM classifier

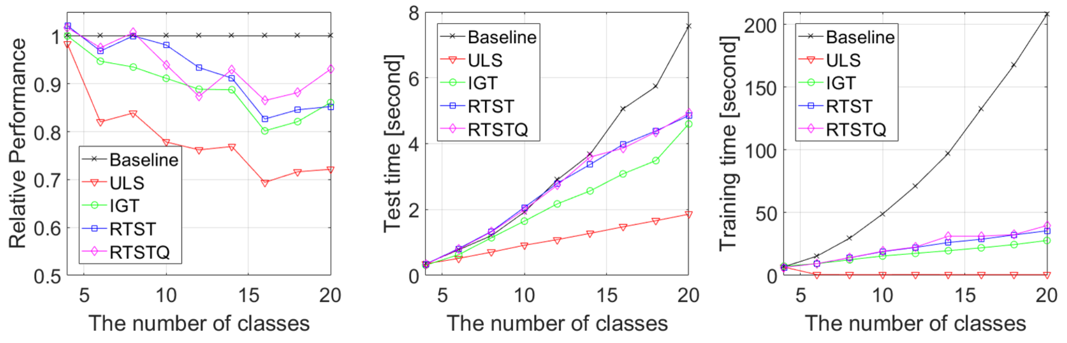


Figure 6: Measurement at variable number of classes with 4 initial classes for an incremental training of NCMF for real data. The relative performance of all methods decrease as new classes are added while both test and training time are shorter than that of baseline. The trade-off relation between the accuracy and computation time is observed.

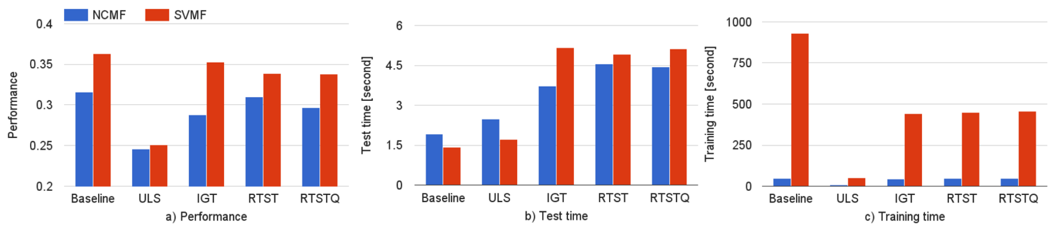


Figure 7: Measurement at 20 classes starting with 4 initial classes for various incremental learning methods and forest variants for real data. The performance of IGT, RTST and RTSTQ is lower than but relatively close to that of the baseline. It is observed that SVMF outperforms NCMF. While training time of incremental learning methods is shorter than the baseline, test time is longer than the baseline.

and SVM classifier perform well on these two datasets. Our incremental learning algorithms achieve superiority when we need adding new classes to an existing trained classifier. They significantly reduce the training time and test time, while preserving the overall accuracy, except for ULS. Incremental strategies other than ULS, both SVMF based and NCMF based, are quite insensitive to the number of initial classes and batch size, which means that they are well suited for incremental learning under various situations. Between the two variations of RF, NCMF is more recommended for incremental learning in terms of time efficiency and accuracy based on our result. Among the incremental learning algorithms, ULS is less recommended because of its poor accuracy, though time efficiency is high. On the other hand, IGT, RTST, RTSTQ have similar performances where IGT performs a bit better as a good trade-off of time efficiency and accuracy in both SVMF and NCMF. Further investigation on large-scale data is necessary, since we actually cut down the size of original large-scale data due to the limitation of computer resource. Also, more novel and high efficiency incremental learning algorithms are desired as an improvement for this research.

5 Group member roles

Ruobai was responsible for the implementation of random forest and ULS incremental method. Shuan completed the SVMF and NCMF by combining random forest with SVM and NCM split function. Zhaoming was responsible for image feature extractions, and together with Takeshi they implemented the IGT and RTST incremental learning method. Then Zhaoming combined all codes to form a consistent workflow. Finally we together tested our classifiers on both the simulated data and real data and we together wrote the report.

References

[1] Marko Ristin, Matthieu Guillaumin, Juergen Gall, and Luc Van Gool. Incremental learning of random forests for large-scale image classification. *IEEE transactions on pattern analysis and machine intelligence*, 38(3):490–503, 2016.

[2] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 35(11):2624–2637, 2013.

[3] Ilja Kuzborskij, Francesco Orabona, and Barbara Caputo. From n to n+1: Multiclass transfer incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3358–3365, 2013.

[4] Andreas Opelt, Axel Pinz, and Andrew Zisserman. Incremental learning of object detectors using a visual shape alphabet. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Volume 1*, pages 3–10. IEEE Computer Society, 2006.

[5] Olga Barinova, Roman Shapovalov, Sergey Sudakov, and Alexander Velizhev. Online random forest for interactive image segmentation. *EEML 2012–Experimental Economics in Machine Learning*, page 1, 2012.

[6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[7] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.