21^{st} Computer Vision Winter Workshop
Luka Čehovin, Rok Mandeljc, Vitomir Štruc (eds.)
Rimske Toplice, Slovenia, February 3–5, 2016

BaCoN: Building a Classifier from only N Samples

Georg Waltner Michael Opitz Horst Bischof
Institute for Computer Graphics and Vision
Graz University of Technology, Austria
{waltner, opitz, bischof}@icg.tugraz.at

Abstract. *We propose a model able to learn new object classes with a very limited amount of training samples (*i.e. 1 *to* 5*), while requiring near zero runtime cost for learning new object classes. After extracting Convolutional Neural Network (CNN) features, we discriminatively learn embeddings to separate the classes in feature space. The proposed method is especially useful for applications such as dish or logo recognition, where users typically add object classes comprising a wide variety of representations. Another benefit of our method is the low demand for computing power and memory, making it applicable for object classification on embedded devices. We demonstrate on the Food-101 dataset that even one single training example is sufficient to recognize new object classes and considerably improve results over the probabilistic Nearest Class Means (NCM) formulation.*

1. Introduction

With recent advances in object recognition [7], off-the-shelf features which are learned from a large number of annotated images have become freely available. As datasets grow, it will become increasingly computationally demanding to extend models built from these features to new object classes. Consider images of different food types - Japanese and European users will have quite different imaginations of an average lunch meal. Adapting a pretrained classifier to recognize meals that have not been seen during the training procedure is desirable. Similar to cognitive capabilities of humans, who can learn new classes from only very few samples, we aim for a computer vision system where new classes can be added incrementally. In this work, we consider classification methods which can integrate previously unseen classes from a single (one-shot) or a small number (n-shot) of training samples. The main purpose of these methods is to recognize new object classes from a very limited number of training samples. This is especially useful for open-ended recognition scenarios, such as logo detection or food recognition, where the number of object classes steadily grows during the life cycle of an object recognition system. However, integrating new classes in a classifier pretrained on different classes is not straightforward. On the one hand a new class often exhibits large variations, on the other hand the classifier trained on seen classes may not be capable to generalize to new classes. Retraining state-of-the-art classifiers such as CNN every time after such class additions leads to

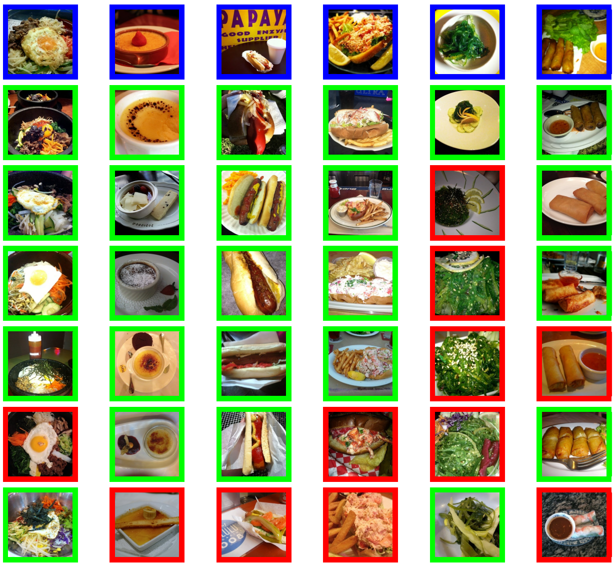

Table 1. One-shot learning results: The top row shows the used training sample (blue), the other rows are the first 6 results where the our proposed method yields different results than the probabilistic NCM version. Green framing indicates our improved NCM version is correct, while red stands for the opposite. From left to right: bibimap, creme brulee, hot dog, lobster roll sandwich, seaweed salad, spring rolls.

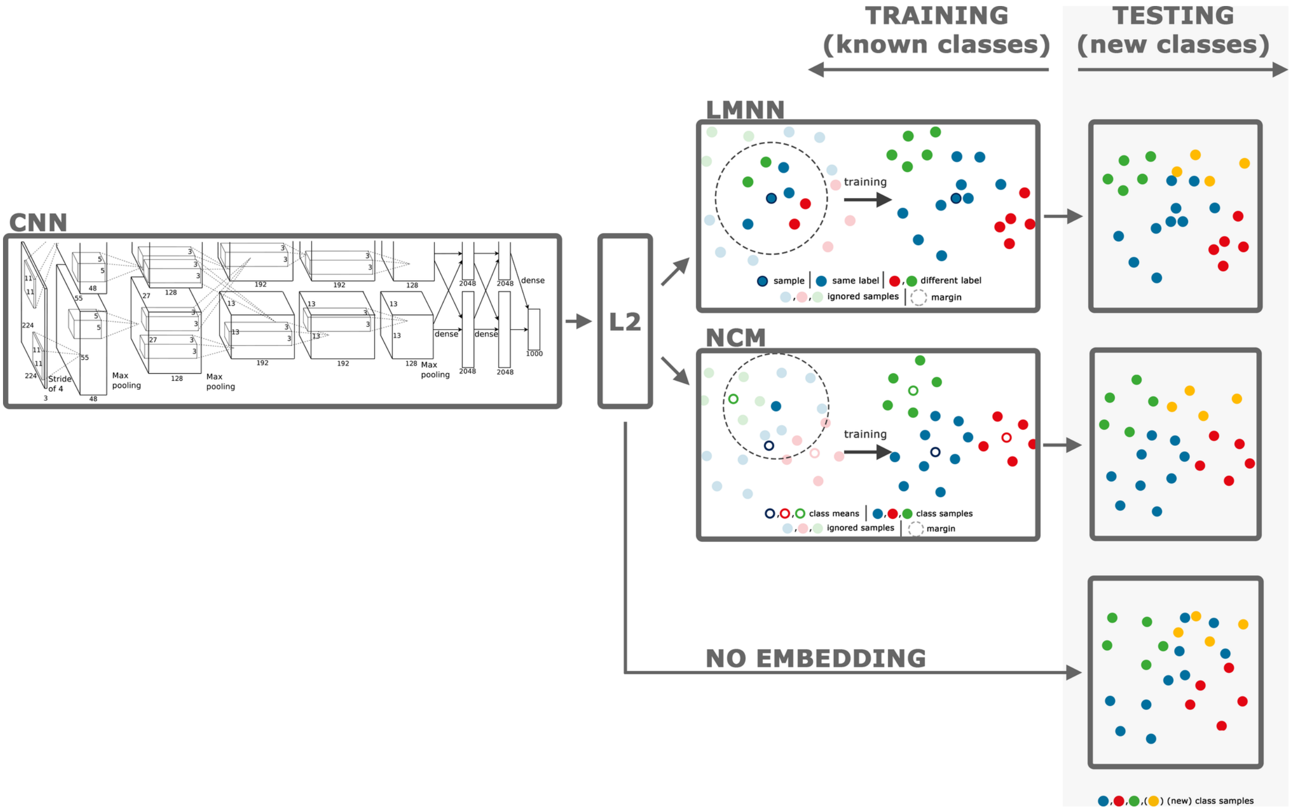

Figure 1. Overview of our method: After extracting CNN features from a fine-tuned CNN, we normalize the feature vectors and learn NCM or LMNN embeddings. These embeddings separate the classes in a way, so that newly added classes can be inserted without much loss in overall accuracy compared to insertion without a trained embedding. We leave the CNN and the trained embeddings fixed when adding new classes.

high accuracy, but is computationally inefficient and requires a significant amount of memory. For practical application this is often prohibitive, especially for embedded systems. An ideal system should therefore be able to integrate new object classes from few very seen samples on the fly; without the need for time- and memory-consuming retraining of the classifier and negligible performance loss.

2. Related Work

One approach in related works considering n-shot settings is the use of Bayesian learning, where probabilistic estimates are used to extend the algorithms to new classes. For example, [3] fit probabilistic density functions as category models and use them as prior knowledge for new classes, while using one or more samples for generation of the posterior model of the new class. Hierarchical Bayesian models are used by [12], where super-categories are automatically discovered based on available classes and serve as prior information to incorporate new classes. In [8], authors investigate one-shot learning of characters using Hierarchical Bayesian Program Learning (HBPL), where characters are modeled as a composition of primitives under certain causality constraints. Another approach for extension to new classes or categories is the use of attributes [9, 11], that can be seen as semantic descriptors which are shared by multiple classes. New classes are then added by generating a semantic description of the new class via attributes. Attribute-based approaches have been used for animal categorization and recognition [9] or for human-nameable visual attributes [11]. Similar to our idea of learning an optimal embedding, [6] statistically infers a Mahalanobis distance metric on similar and dissimilar feature pairs. In [16], a classifier is trained discriminatively for nearest prototype classification. Another distance metric learning approach was presented in [15], where the authors propose Large Margin Nearest Neighbor (LMNN) for classification. The LMNN classifier learns a Mahalanobis metric, so that same class samples are contracted and samples from different classes are pushed apart from each other. These methods are able to generalize to previously unseen samples, but in contrast to our approach they do not regard insertion of previously unseen object classes.

We employ a Nearest Class Mean (NCM) classifier [10, 14] for classification. Object classes in NCM classifiers are represented by the mean feature vector of the corresponding class samples and can be easily extended to new classes by computing the mean over newly added training samples. Our method learns discriminative embeddings to better separate the classes in feature space. Other than the probabilistic NCM approach of [10], we use CNN features. We propose the hinge loss for optimization and show that this improves overall accuracy. Additionally, we show how to robustify the learned NCM embeddings. In contrast to other approaches for one-shot and n-shot learning, our method does not need access to the full dataset as we do not employ classifier retraining, enabling the use of our system for embedded platforms like smartphones, where computing power and storage is limited. Furthermore, the most one-shot algorithms are Bayesian methods and use prior knowledge from the training data to generate posterior probabilities for new classes. We do not model such probabilities, but rely on the learned feature embeddings only. Figure 1 gives an overview of our method: We use l_2-normalized CNN features and learn additional layers that embed the features in an optimal way. After that we add new classes to evaluate the incremental learning capability of our classifier. Table 1 shows one-shot learning results for some classes of the Food-101 dataset [2].

3. One-Shot and N-Shot Classification

In the n-shot classification setting the classifier extends to new classes from a very limited number of samples (*i.e.* 1 to 5). NCM classifiers store a mean vector for each object class they recognize. This has the advantage that recognition of new classes can be incorporated by simply computing mean vectors for these classes. Mean vectors can be efficiently computed online, eliminating the need of explicitly storing feature vectors of all training samples that the class mean originates from. For one-shot learning, the class mean corresponds to one added class sample, for n-shot learning the mean is calculated from n samples of a new class. More formally, let $\boldsymbol{\mu}_c$ be the mean vector for the c-th class from the set $\mathcal{C}$ of available classes, defined as

$$\boldsymbol{\mu}_c = \frac{1}{N_c} \sum_{i=1}^{N_c} f(\mathbf{x}_i; \boldsymbol{\theta}), \tag{1}$$

where N_c is the number of samples $\mathbf{x}$ for class c, f is a feature extraction function and $\boldsymbol{\theta}$ are model-parameters. To predict the object class of a sample we seek the minimum distance to all class means by computing

$$\underset{c=1,\dots,C}{\arg\min} \left\| f(\mathbf{x}; \boldsymbol{\theta}) - \boldsymbol{\mu}_c \right\|_2 , \tag{2}$$

where $C = |\mathcal{C}|$ is the total number of classes.

3.1. Feature Extraction

Motivated by their recent success in image recognition tasks, we utilize CNNs for feature extraction. Instead of training a deep network from scratch, we take a CNN model trained for the ImageNet Challenge [7] and fine-tune on our task-specific training data. This can be seen as domain transfer from one task to another and has proven to be superior to hand-crafted features [4]. As the later layers of the network correspond to high-level features, we use the last fully connected layer as 4096-dimensional feature representation and normalize each feature vector by dividing by its l_2-norm.

3.2. Embedding

After fine-tuning the CNN, we employ several distance metric learning methods to learn a discriminative linear embedding matrix $\mathbf{W} \in \mathbb{R}^{d \times 4096}$, with $d \in \{1024, 4096\}$. This embedding projects samples from the same object class next to each other in a high dimensional feature space, while simultanously pushing samples from different object classes far away from each other. Using the embedding $\mathbf{W}$, the class prediction from Equation (2) becomes

$$\underset{c=1,\dots,C}{\arg\min} \left\| \mathbf{W} \cdot f(\mathbf{x}; \boldsymbol{\theta}) - \mathbf{W} \cdot \boldsymbol{\mu}_c \right\|_2 , \tag{3}$$

In this work, we consider optimizing NCM loss functions and the LMNN loss with respect to $\mathbf{W}$ to learn our embedding. In the remainder of this section, we will formally explain the different methods.

NCM. As proposed in [10], embeddings for NCM classifiers are usually learned by minimizing the negative log-likelihood. The posterior probability $p(c|\mathbf{x})$ of class c given a sample $\mathbf{x}$ is defined as

$$p(c|\mathbf{x}) = \frac{e^{-\delta(\mathbf{x}, \boldsymbol{\mu}_c; \boldsymbol{\theta})^2}}{\sum_{i=1}^{C} e^{-\delta(\mathbf{x}, \boldsymbol{\mu}_i; \boldsymbol{\theta})^2}}, \tag{4}$$

where δ is defined as

$$\delta(\mathbf{x}, \boldsymbol{\mu}; \boldsymbol{\theta}) = \left\| \mathbf{W} \cdot f(\mathbf{x}; \boldsymbol{\theta}) - \mathbf{W} \cdot \boldsymbol{\mu} \right\|_2 . \tag{5}$$

To learn the embedding $\mathbf{W}$, minimize the negative log-likelihood

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \ln p(y_i|\mathbf{x}_i) \tag{6}$$

of sample $\mathbf{x}_i$ and its corresponding class label y_i. In subsequent sections we refer to this method as probabilistic NCM (NCM_P), since we are optimizing a negative log-likelihood function.

LMNN. The loss function of the LMNN embedding [15] consists of two terms. One adds a penalty for samples that share the same class label but exceed a certain distance (margin), while the other penalizes samples with different class labels that are close in feature space. The loss is calculated on triplets instead of pairs, where a sample is complemented by a sample of the same and a sample of a different class. The set of triplets is given by

$$D = \{(i,j,k) : y_i = y_j, y_i \neq y_k\} \tag{7}$$

and $1 \leq i < j < k \leq N$, the LMNN loss function over the triplet set is then defined as

$$L(D) = \sum_{(i,j,k)\in D} d_{ij}^2 + l_{ijk}. \tag{8}$$

The distance function d of two samples $\mathbf{x}_i$ and $\mathbf{x}_j$ is

$$d_{ij} = \|\mathbf{W}\cdot f(\mathbf{x}_i;\boldsymbol{\theta}) - \mathbf{W}\cdot f(\mathbf{x}_j;\boldsymbol{\theta})\|_2 \tag{9}$$

and the triplet loss function l_{ijk} is defined as

$$l_{ijk} = \max(0, 1 + d_{ij} - d_{ik}). \tag{10}$$

This embedding maximizes the distance in feature space between samples of different classes $(\mathbf{x}_i, \mathbf{x}_k)$, while concentrating samples that belong to the same class $(\mathbf{x}_i, \mathbf{x}_j)$. Following [13], during training we perform "hard" negative mining of triplets which violate the margin constraint imposed by l_{ijk}. Opposite to "soft" negatives, which do not violate the margin or violate the margin by only a small amount, "hard" negatives impose a high loss and therefore lead to faster training of the model and increased performance.

3.3. Large Margin Nearest Class Mean Classifiers

Inspired by LMNN we propose a large margin loss function for NCM classifiers. Ideally, samples from the same class are close to their own mean vector and are far away from other mean vectors in feature space. Let $\mathbf{x}$ be a data sample, y the class label and $\boldsymbol{\theta}$ the parameters of the feature extraction function f. We propose to train a NCM layer on top of the CNN features with the following NCM loss function

$$L(\mathbf{x}, y;\boldsymbol{\theta}) = \lambda\cdot\delta_y^2 + \sum_{c\in\mathcal{C}\setminus\{y\}} \max(0, 1+\delta_y-\delta_c)^2, \tag{11}$$

where $\delta_y = \delta(\mathbf{x}, y;\boldsymbol{\theta})$ and $\delta_c = \delta(\mathbf{x}, c;\boldsymbol{\theta})$ are distance functions as defined in Equation (5) and λ is a weighting parameter. The first part enforces the samples of one class to be embedded near the class mean of the data sample, while the second term penalizes if samples are within the margin of other class means. This large margin version of the NCM classifier will be referred to as NCM_{LM}

3.4. Robust NCM

Due to variations in shape, illumination and appearance, feature vectors from an object class usually exhibit intra-class variance. We model this uncertainty by assuming that a feature vector for a sample $\mathbf{x}$ associated with class c is generated by a normal distribution $\mathcal{N}(\boldsymbol{\mu}_c, \sigma_c)$. We incorporate this variation in our model by computing the standard deviation σ_c for all classes $c \in \mathcal{C}$ over the training set. During optimization we add random noise ϵ to our feature vectors to account for this uncertainty. More formally, the loss function we minimize is

$$L(\mathbf{x}, y;\boldsymbol{\theta}) = \lambda\cdot\hat{\delta}_y^2 + \sum_{c\in C\setminus\{y\}} \max(0, 1+\hat{\delta}_y-\hat{\delta}_c)^2, \tag{12}$$

where

$$\hat{\delta}(\mathbf{x}, \boldsymbol{\mu};\boldsymbol{\theta}) = \left\|\mathbf{W}\cdot\hat{f}(\mathbf{x};\boldsymbol{\theta}) - \mathbf{W}\cdot\boldsymbol{\mu}\right\|_2. \tag{13}$$

and

$$\hat{f} = f(\mathbf{x}) + \boldsymbol{\Sigma}_y^{\frac{1}{2}}\epsilon\cdot\gamma. \tag{14}$$

$\boldsymbol{\Sigma}_y$ is the diagonal covariance matrix of class y, $\epsilon \in \mathbb{R}^{4096} \sim \mathcal{N}(0,1)$ is a random vector drawn from a normal distribution and γ is a hyper-parameter, which defines the impact of the distortions. In our experiments we fix λ to 0.01 and γ is set to 0.5. During training we first compute the standard deviation of each feature per object class. We then add the noise to our feature vectors, to make the embedding $\mathbf{W}$ more robust against inter-class variations. This robustification is done in real time during training and can be seen as data augmentation, making the impact of outliers on the means smaller. We refer to this method as $NCM_{LM\text{-}R}$.

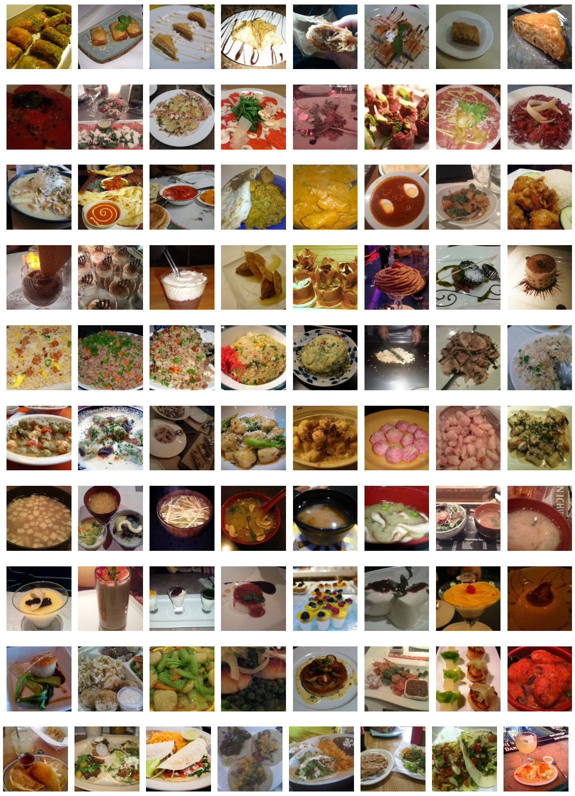

Figure 2. First 8 samples of randomly chosen classes from the Food-101 dataset [2]. From top to bottom: baklava, beef carpaccio, chicken curry, chocolate mousse, fried rice, gnocchi, miso soup, panna cotta, scallops, tacos.

4. Experiments

For evaluation of our method we use the publicly available Food-101 dataset [2]. It consists of 101 food classes with 1000 images per class. The images were taken in real world environments, exhibiting a lot of variation in illumination conditions or food arrangement (see Figure 2 for some examples) and are well suited for the targeted application case where users add data continuously.

Following the protocol in [2], we randomly split the 1000 samples of each class into 750 for training and 250 for testing. For training of the CNN, we then apply a 80%/20% split for training and validation (600 and 150 samples respectively). This results in a training-, validation- and test-set with 60.600 (60%), 15.150 (15%) and 25.250 (25%) samples, respectively. Further from the 101 classes we randomly select 50 training classes on which we train our classifiers and 51 classes on which we evaluate the generalization capability of our method to novel classes. For the sake of completeness, we also evaluate the embeddings on the 50 training classes only.

For fine-tuning CaffeNet on the Food-101 dataset, we train our network with Stochastic Gradient Descent (SGD) and momentum. We follow standard fine-tuning protocols [7] and use a low initial learning rate of 0.001 and a momentum of 0.9. We anneal the learning rate by a factor of 10 after each 20.000 iterations. To determine convergence, we measure the accuracy on a validation set after 500 gradient updates.

We optimize our embeddings with SGD and momentum. For training the embeddings, when not otherwise stated, we fix the weights of the CNN and train just the last embedding layer. This allows us to use large learning rates of 0.25-0.5 with a momentum term of 0.9. Further, we use large minibatch sizes of 1024 and train for about 20 epochs. We exponentially anneal the learning rate at epoch 15 and 18. To determine convergence, we measure the accuracy on our validation set after each training epoch.

In our experiments we use Caffe [5] for fine-tuning, while the evaluations on the embedding methods are implemented in Python utilizing the Theano library [1].

4.1. Experiments with Known Classes

To obtain feature representations and a softmax baseline, we fine-tune the pretrained ImageNet CaffeNet model from [7] on the 50 training classes from the Food-101 dataset as described above. In the following, we compare our methods to the softmax classifier ($CNN_{softmax}$) and to a probabilistic NCM (NCM_P) version related to the work of [10]. The first results are obtained by nearest class mean classification, using euclidean (CNN_{euc}) and cosine (CNN_{cos}) distance measures between the class means of all trainings samples and the test samples. Subsequently we train our NCM and LMNN embedding layers on top of the fine-tuned network while leaving the net weights fixed (NCM_{LM}, $NCM_{LM\text{-}R}$, $LMNN$). A summary of the results is depicted in Table 2.

Interestingly, the nearest class mean classification performs better than the probabilistic version of NCM, implying that the CNN features already separate the classes well. Our robust NCM version improves results over the probabilistic version by about 2% and is very competitive in comparison to the end-to-end trained softmax classifier of the network. The NCM embedding trained on the hinge loss and the LMNN embedding also reach comparable accuracy.

Method	Emb.	$n = 1$	$n = 5$	$n = 10$	$n = 20$	$n = 50$	$n = 100$
CNN_{euc}	–	44.15 ± 0.08	49.30 ± 0.31	$\mathbf{54.26 \pm 0.30}$	$\mathbf{57.66 \pm 0.20}$	$\mathbf{60.03 \pm 0.15}$	$\underline{60.83 \pm 0.13}$
CNN_{cos}	–	44.92 ± 0.24	49.82 ± 0.34	53.92 ± 0.32	$\underline{57.26 \pm 0.21}$	$\underline{59.86 \pm 0.16}$	60.76 ± 0.13
LDA	–	44.51 ± 0.01	44.92 ± 0.12	45.85 ± 0.12	48.44 ± 0.18	57.87 ± 0.17	$\mathbf{63.61 \pm 0.16}$
NCM_P	1024	45.55 ± 0.33	50.11 ± 0.32	51.89 ± 0.25	53.08 ± 0.20	54.03 ± 0.13	54.39 ± 0.09
NCM_P	4096	45.62 ± 0.34	50.23 ± 0.33	52.03 ± 0.25	53.23 ± 0.20	54.20 ± 0.14	54.57 ± 0.09
NCM_{LM}	1024	46.23 ± 0.32	51.46 ± 0.34	53.49 ± 0.26	54.88 ± 0.18	56.02 ± 0.15	56.44 ± 0.11
NCM_{LM}	4096	45.97 ± 0.28	51.43 ± 0.34	53.51 ± 0.27	54.93 ± 0.19	56.06 ± 0.13	56.50 ± 0.11
$NCM_{LM\text{-}R}$	1024	$\mathbf{46.30 \pm 0.32}$	$\mathbf{51.95 \pm 0.35}$	$\underline{54.15 \pm 0.26}$	55.67 ± 0.21	56.89 ± 0.15	57.37 ± 0.11
$NCM_{LM\text{-}R}$	4096	$\underline{46.28 \pm 0.32}$	$\underline{51.94 \pm 0.35}$	54.13 ± 0.28	55.66 ± 0.21	56.85 ± 0.15	57.31 ± 0.11
$LMNN$	1024	45.78 ± 0.25	51.60 ± 0.32	53.82 ± 0.28	55.34 ± 0.21	56.57 ± 0.16	57.05 ± 0.12
$LMNN$	4096	45.29 ± 0.18	51.58 ± 0.33	54.10 ± 0.28	55.81 ± 0.21	57.14 ± 0.14	57.63 ± 0.11
SVM	–	46.52 ± 0.40	50.02 ± 0.35	52.25 ± 0.30	55.08 ± 0.24	59.74 ± 0.19	63.38 ± 0.17

Table 3. Classification accuracy over the full Food-101 test-set (250 samples per class) after adding $n \in \{1, 5, 10, 20, 50, 100\}$ training samples for each of the 51 test-classes. Accuracy and standard deviation are calculated over 100 runs. The baseline accuracy for end-to-end training of the CNN on all classes with all available data is 66.63%. The best and the second best result in each column is shown in bold and underlined.

Method	Emb. size	Accuracy
CNN_{euc}	–	68.60
CNN_{cos}	–	68.64
NCM_P[10]	1024	67.66
NCM_P[10]	4096	67.75
NCM_{LM}	1024	69.00
NCM_{LM}	4096	69.14
$NCM_{LM\text{-}R}$	1024	**69.68**
$NCM_{LM\text{-}R}$	4096	$\underline{69.61}$
$LMNN$	1024	69.20
$LMNN$	4096	69.11
$CNN_{softmax}$	–	70.26

Table 2. Classification results of the 50 classes used for fine-tuning the CNN model for feature extraction. Our proposed robust NCM version reaches almost the same accuracy as the end-to-end trained softmax classifier while improving the results over the standard probabilistic NCM classifier by 2%. Best (bold) and second best (underlined) embeddings are marked.

4.2. Introducing Unseen Classes

To assess how our method generalizes to new classes from only a limited number of samples, we use n random samples from the training set of the remaining 51 classes to compute the mean vectors from the output of the embeddings. The embeddings and the CNN remain fixed and are not retrained, hence the addition of new classes reduces to storing the new class means. We choose $n \in \{1, 5, 10, 20, 50, 100\}$ and report the accuracy on the full Food-101 test-set, where every class is represented by 250 samples. Since for small values of n the results might have a large standard deviation, we repeat these experiments 100 times using different training samples to compute the class means that represent the new classes. Table 3 shows, that fine-tuning the network in the training phase (known classes) with metric learning methods generally improves accuracy in the testing phase for smaller values of n. Training the CNN with Caffe on the full dataset of 101 classes converges after approximately 100.000 iterations to 66.63%.

We also trained two more standard classifiers on the CNN features, namely SVM and LDA. It is remarkable, that although the SVM has access to the full dataset, the performance compared to our proposed methods is inferior for $n \in \{5, 10, 20\}$. The same applies for utilizing a LDA classifier, where only a big number of new samples achieves a performance improvement compared to our proposed methods.

5. Conclusion

We introduced embedding methods for one-shot and n-shot object class recognition. Our proposed extensions to NCM classifiers consistently improve the accuracy over the standard NCM training formulation in a scenario where the amount of classes to be recognized by the classifier doubles. Our methods perform best for settings where only very few new samples ($n \leq 10$) per class are available. The extension of the classifier to new object classes is independent of the old training data and is efficient in terms of computational expense and memory. This is especially useful for recognition systems running on embedded devices, where CPU power and memory is limited.

Acknowledgements

This work was supported by the Austrian Research Promotion Agency (FFG) under the projects MANGO (836488) and DIANGO (840824).

References

[1] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU Math Expression Compiler. In *Proceedings of the Scientific Computing with Python Conference*, June 2010.

[2] L. Bossard, M. Guillaumin, and L. Van Gool. Food-101 – Mining Discriminative Components with Random Forests. In *European Conference on Computer Vision*, 2014.

[3] L. Fei-Fei, R. Fergus, and P. Perona. One-Shot Learning of Object Categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[6] M. Köstinger, P. Wohlhart, P. M. Roth, and H. Bischof. Joint Learning of Discriminative Prototypes and Large Margin Nearest Neighbor Classifiers. In *IEEE International Conference on Computer Vision*, 2013.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, 2012.

[8] B. M. Lake, R. R. Salakhutdinov, and J. Tenenbaum. One-Shot Learning by Inverting a Compositional Causal Process. In *Advances in Neural Information Processing Systems*, 2013.

[9] C. H. Lampert, H. Nickisch, and S. Harmeling. Learning to Detect Unseen Object Classes by Between-class Attribute Transfer. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[10] 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 and Machine Intelligence*, 35(11):2624–2637, 2013.

[11] D. Parikh and K. Grauman. Relative Attributes. In *IEEE International Conference on Computer Vision*, 2011.

[12] R. Salakhutdinov, J. B. Tenenbaum, and A. Torralba. One-Shot Learning with a Hierarchical Nonparametric Bayesian Model. In *Workshop on Unsupervised and Transfer Learning in conjunction with the International Conference on Machine Learning*, 2012.

[13] F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[14] A. R. Webb and K. D. Copsey. *Statistical Pattern Recognition*. Wiley, 3rd edition, 2011.

[15] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. In *Advances in Neural Information Processing Systems*, 2005.

[16] P. Wohlhart, M. Köstinger, M. Donoser, P. M. Roth, and H. Bischof. Optimizing 1-Nearest Prototype Classifiers. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.