Interpolating Convolutional Neural Networks Using Batch Normalization

Gratianus Wesley Putra Data(✉), Kirjon Ngu, David William Murray, and Victor Adrian Prisacariu

Active Vision Laboratory, Department of Engineering Science, University of Oxford, Oxford, UK
gwpd@robots.ox.ac.uk

Abstract. Perceiving a visual concept as a mixture of learned ones is natural for humans, aiding them to grasp new concepts and strengthening old ones. For all their power and recent success, deep convolutional networks do not have this ability. Inspired by recent work on universal representations for neural networks, we propose a simple emulation of this mechanism by purposing batch normalization layers to discriminate visual classes, and formulating a way to combine them to solve new tasks. We show that this can be applied for 2-way few-shot learning where we obtain between 4% and 17% better accuracy compared to straightforward full fine-tuning, and demonstrate that it can also be extended to the orthogonal application of style transfer.

Keywords: Neural network interpolation · Batch normalization
Few-shot learning · Style transfer

1 Introduction

Human visual cognition is remarkable. One of the many things humans do naturally is linking visual concepts to a combination of other concepts. For example, after being shown images of a dog, a cat, and a fox, a child could say that the fox looks like a cross between a cat and a dog (Fig. 1). Furthermore, the child will understand much about the concept of a fox given prior knowledge of what cats and dogs look like. A loose mathematical analogy can be expressed as follows: if visual representations of cats and dogs can be encapsulated in the form of functions ϕ_{cat} and ϕ_{dog}, respectively, it should also be possible to build from them a representation for foxes $\phi_{\text{fox}} = f(\phi_{\text{cat}}, \phi_{\text{dog}}, \alpha)$, where f represents how the functions should be combined as parameterized by α. Additionaly, it should be easier to deduce the value of α than ϕ_{fox} directly.

In this paper we ask if the same ideas can be adapted to deep convolutional neural networks to enable more efficient learning. This is desirable as, despite generally being powerful state-of-the-art models [11,16,22], deep networks require a tremendous amount of data to tune millions of parameters that

V. Ferrari et al. (Eds.): ECCV 2018, LNCS 11217, pp. 591–606, 2018.
https://doi.org/10.1007/978-3-030-01261-8_35

allow it to work so well, limiting its application to tasks where data is plenty or inexpensive.

Recent works [3,20] have reported that it is possible to prepare a single network that is able to perform visual recognition in multiple domains. This is achieved by training the network to produce universal image representations, relying on (i) the convolutional kernels to extract domain-agnostic information about the world and on (ii) the batch normalization (BN) layers to transform the internal representations to the relevant target domains. Analogously, within the application domain of style transfer, [6] shows that a single network can be equipped with multiple distinct styles by encoding the style information in the network's instance normalization (IN) layers, after which each style can selectively be applied to a target image. These discoveries seem to provide evidence for the ability of normalization layers to encode transforms that can be used to express visual concepts.

In line with our opening exposition, we propose and wish to test the following intuition in this paper: given that normalization layers (e.g. BN) can be trained to discriminate specific visual classes, it should be possible to combine these normalization layers and interpolate within them to efficiently learn new, unseen classes. In particular, since we will only be manipulating the normalization layers within a network, the number of parameters that we need to tweak will be much lower than full fine-tuning. Fewer parameters also means less tendency to overfit, enabling training with smaller amounts of data. Focusing on binary classification tasks, we summarize our contributions in this paper as follows:

1. Defining a procedure that specifies how component networks that discriminate specific classes are generated and interpolated to discriminate new, unseen classes.
2. Demonstrating how interpolation of component BN layers can be applied to the problem of few-shot visual recognition.
3. Showing that the same interpolation process (using IN) can be adapted to the orthogonal task of style transfer.

Fig. 1. The fox can be seen as a mix between this cat and this dog.

The remainder of the paper proceeds as follows: we first mention several works that are related to our method in Sect. 2. Afterwards, we describe and elaborate procedures for creating and interpolating between component BNs in

Sect. 3. We validate the key idea of interpolating BNs on CIFAR10 [15], apply the same procedures to few-shot learn ImageNet32 [5] using CIFAR10-trained kernels, and also apply it to the orthogonal task of style transfer (by replacing BNs with INs) in Sect. 4. Finally, we conclude our paper in Sect. 5.

2 Related Work

To our knowledge, we are the first to tackle the problem of neural network interpolation. Our main reason for attempting this is to reduce the number of parameters required to train the neural network and to achieve faster convergence with fewer images. We therefore believe our approach is most related to other works that (i) try to reduce the number of trained/tested parameters, (ii) dictionary learning, and (iii) few-shot learning/meta-learning approaches.

Parameter Reduction. Within the realm of parameter reduction, there have been attempts to compress and distill knowledge in neural networks [12], and novel designs for efficient architectures which reduce the number of parameters during inference [4,9]. These assume the neural network is trained in a traditional way and provide methods by which the post-training parameters can be reduced, e.g. through some form of sparsity. Reduction of training parameters is however much less studied, with the traditional approach looking at training only a subset of the complete network, e.g. the last few layers. However, recent work has shown an alternative strategy of training a neural network (rather than just retuning the last layers): to adapt the network's batch normalization parameters. This proved to be effective when training and adapting domain-agnostic neural networks in [3,20] and, relatedly, when aiming to adapt existing neural networks to new types of style transfer as in [6].

Dictionary Learning. The aim of dictionary learning [21] is to learn fundamental representations from data that can be combined linearly to construct sparse codings of the data. A collection of these fundamental representations (*atoms*) form a *dictionary*. A few well-known algorithms that perform dictionary learning include the method of optimal directions [7] and K-SVD [1].

Few-Shot Learning/Meta-Learning. The application of deep convolutional networks for few-shot learning has recently seen a resurgence. Naming just a few methods, Koch *et al.* [14] used Siamese networks to quantify distances between samples, and then used a non-parametric classifier such as k-nearest neighbours to perform one-shot learning. Bertinetto *et al.* [2] modified the Siamese architecture to enable the first network to predict suitable weights for the other in the one-shot regime. Hariharan and Girschick [10] proposed the SGM loss and hallucination as data augmentation to perform n-way few-shot learning where n is large. Luo *et al.* [18] suggested a network framework that is able to learn transferrable representations in a label-efficient manner.

Within the framework of meta-learning, Vinyals *et al.* [25] introduced the concept of episodic training to ensure few-shot training and testing conditions match, and used a cosine similarity metric on network embeddings to peform the classification. Ravi and Larochelle [19] used an LSTM meta-learner to directly perform episodic weight updates on a few-shot learner network, made possible by the similarity between LSTM and gradient descent update formulation. Snell *et al.* [23] utilized a network to learn embeddings which cluster classes around prototypes, which then classifies new examples by proximity to learned prototypes. Finn *et al.* [8] proposed a simple meta-learning training algorithm that aims to generate good initialization parameters for a classifier network, which was then able to achieve good performance after a single parameter update step.

3 Method

First, we briefly review the batch normalizing (BN) transform [13]. Let x_i be the activations of a single example i inside a mini-batch of size m. The BN transform is defined as the operation

$$\mathrm{BN}(x_i) \equiv \gamma \widehat{x}_i + \beta, \tag{1}$$

given the mean $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$, variance $\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$, and the normalized input $\widehat{x}_i = (x_i - \mu_{\mathcal{B}})/\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}$. The scale γ and shift β parameters are learnable, while ϵ is a small positive constant to prevent division by zero.

Next, we will show how component BNs are constructed, and detail two ways of combining them for the purpose of interpolating new classes. This section focuses on the binary classification scenario, but the principles presented can translate to other application domains, as we show later in Sect. 4.

3.1 Component Generation

Given that component BNs are purposed to be discriminative towards a particular object class, a straightforward way to generate them for that object class would be to extract BNs from a network trained on a corresponding binary classification task.

More specifically, we start from a base pretrained network which we refer to as the template network. To create BN layers that detect the concept of e.g. cat, we fine-tune the network on a dataset containing examples of cats and non-cats by adjusting only the BN and last classification layer parameters. This fine-tuned network is now a component network that detects cats. We repeat this procedure for other object classes, always starting from the same template network, until the desired number of component networks is obtained.

The number of component networks is a function of the task at hand and the quantity of available data and classes. For example, for our experiments, we create 9 component networks for CIFAR10 and 200 for ImageNet.

3.2 Binary Dataset Creation

It is beneficial to generate a number of component BNs so that a good coverage of the target task is achievable. Large multiclass datasets are naturally a suitable source. However, they need to be binarized before they can be used for component generation.

We formalize this as follows. Suppose we have a set of N labelled images $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ where $y_n \in \{1, \ldots, K\}$ are the labels. Binarizing $\mathcal{D}$ for class k means randomly copying $S/2$ elements of $\mathcal{D}$ where $y_n = k$ and another $S/2$ where $y_n \neq k$ to form a new set $\mathcal{D}_k = \{(x_1, y_1), \ldots, (x_S, y_S)\}$ where $y_s \in \{0, 1\}$ after applying the binary label transform

$$\phi(y_n) = y_s = \begin{cases} 1 & \text{if } y_n = k, \\ 0 & \text{if } y_n \neq k. \end{cases} \tag{2}$$

3.3 Component Selection

Once a number of component BNs have been generated, it is important to select the components that will be relevant for the task at hand (e.g. airplanes are probably not a good component to include when trying to detect foxes). Although there may exist sophisticated selection methods, we propose two straightforward criteria which we demonstrate in more detail in Sect. 4.2. The first criterion involves selecting m component networks with the lowest cross-entropy loss on the target binary task. The second criterion does the same thing, except that it ranks based on highest accuracy on the target binary task. Naturally, the first criterion is more amenable to tasks with few examples to evaluate.

3.4 Interpolating Component Networks

After the BN components have been computed and selected for a specific novel target class, we propose two approaches for interpolation:

1. Composite Batch Normalization (ComBN), providing a linear combination of BN components.
2. Principal Component Batch Normalization (PCBN), providing a PCA-based latent space interpolation.

The interpolation weights for both approaches are learned through standard neural network optimization techniques, i.e. backpropagation and stochastic gradient descent (SGD).

Other methods could be used to interpolate the BN components, such as more complex non-linear dimensionality reduction techniques like Gaussian Process Latent Variable Models [17], but, as we show in the results section, the simpler linear models already achieve very good results.

Composite Batch Normalization (ComBN). Using a similar notation to the one above, we propose the ComBN transform as a linear combination of generated BN components,

$$\mathrm{ComBN}_{\alpha}(x_i) \equiv \sum_{j=1}^{J} \alpha_j \mathrm{BN}_j(x_i), \tag{3}$$

where J is the number of BN components that make up a ComBN, and α_j are learnable scalar coefficients that represent the interpolation weights, all initialized to $1/J$.

In practice, after the component networks have been generated, each BN layer in the original template network is replaced with a ComBN, which is constructed from BN layers of selected component networks originating from the same depth-wise layer position. Afterwards, we train the ComBN network by optimizing α_j and the last layer to the target task using standard techniques (i.e. backpropagation and SGD).

Note that the component BNs in the ComBN network are always utilized in inference mode; i.e. their γ, β, running mean, and running variance are frozen, and the running mean and variance are used in place of the mini-batch mean $\mu_{\mathcal{B}}$ and variance $\sigma^2_{\mathcal{B}}$ when evaluating Eq. 1.

Additionally, this formulation typically enables a large reduction of the number of parameters, which is helpful in reducing overfitting when the training data is scarce.

Principal Component Batch Normalization (PCBN). An alternative way to exploit information contained in BN components is to first use them to learn a latent space mapping for its parameters, and then perform optimization in the latent space.

To achieve this using PCA, we first stack row vectors of γ and β parameters that originate from each BN component j to form $J \times C$ matrices $\mathbf{\Gamma}$ and $\mathbf{B}$, respectively, where J is the number of components and C is the number of channels in each component BN layer. We then mean-center $\mathbf{\Gamma}$ and $\mathbf{B}$ by subtracting from them their column-wise mean vectors μ_γ and μ_β, resulting in $\mathbf{X}_\gamma = \mathbf{\Gamma} - \mu_\gamma$ and $\mathbf{X}_\beta = \mathbf{B} - \mu_\beta$. Afterwards, we apply singular value decomposition to obtain principal axes matrices $\mathbf{V}_\gamma^\top$ and $\mathbf{V}_\beta^\top$,

$$\mathbf{U}_\gamma \mathbf{S}_\gamma \mathbf{V}_\gamma^\top \leftarrow \mathbf{X}_\gamma, \tag{4}$$

$$\mathbf{U}_\beta \mathbf{S}_\beta \mathbf{V}_\beta^\top \leftarrow \mathbf{X}_\beta. \tag{5}$$

The number of dimensions of our latent space is set to the maximum possible, i.e. $\min(J, C)$. We then train latent space parameter vectors g and b (initialized by transforming existing BN weights of the template network to latent space), and transform these back to parameter space using the principal axes matrices,

$$\gamma = g\mathbf{V}_{\gamma}^{\top} + \mu_{\gamma}, \tag{6}$$

$$\beta = b\mathbf{V}_{\beta}^{\top} + \mu_{\beta}. \tag{7}$$

This is then finally applied in a similar fashion to ComBN by replacing BN layers in the original template network with PCBN (*i.e.* substituting Eqs. 6 and 7 into Eq. 1). In essence, this is like using standard BN except for the optimization of parameters in latent space.

In contrast to ComBN where interpolation is directly performed in the parameter space of the original component class (in the form of frozen BN components), here we attempt to first distill the concepts of class into principal classes in latent space before optimizing them.

4 Experiments

In this section, we show results for two application domains: visual classification and style transfer.

We choose to constrain our experiments to binary classification so that the same protocol can be used for both component generation and evaluation. To highlight the contribution of BN layers, we utilize a template network trained on a different dataset to the one we are testing with; first using ImageNet for the template and CIFAR10 for the testing of our approach, and second using CIFAR10 to train the template network and ImageNet32 for testing.

We chose style transfer because (i) we view this as an orthogonal (i.e. related but highly distinct) task to binary classification since it requires utilization of full encoder-decoder networks and replacement of batch normalization with instance normalization, and (ii) it allows us to produce qualitative results.

4.1 Learning CIFAR10 from ImageNet Template

Here we validate the idea of using BN components for training networks on new, unseen tasks. Lastly, for the experiments in this section, we base our template network on an ILSVRC2012-pretrained ResNet34 [11], and use binarized CIFAR10 datasets to generate our BN components and evaluate the performance of ComBN and PCBN networks.

We begin by creating master training/validation/test splits. These master training/validation splits are created by partitioning the original CIFAR10 training set 40,000/10,000, while the master test split is the same as the original CIFAR10 test set. Afterwards, we generate binarized splits by applying the method in Sect. 3.2 to each of the master splits. Specifically, for each target class, the binary training split is formed by sequentially sampling 1000 positive and 1000 negative examples from the master split, while the binary validation/test splits are formed by sequentially and exhaustively sampling all available examples from their respective master splits such that a balanced dataset is obtained.

Table 1. Percentage test accuracies on binary CIFAR10 datasets. Asterix (*) indicates results that were based on a random template network. Best results are in bold.

Positive class	Last	Full	BN	ComBN	PCBN	BN*	ComBN*	PCBN*
airplane	90.1	**96.1**	95.8	81.9	93.8	76.8	77.2	95.4
car	92.0	**98.0**	97.3	97.7	97.4	77.7	79.8	97.1
bird	87.5	**95.5**	94.5	93.8	91.4	66.7	68.3	93.5
cat	84.3	91.7	89.6	**91.9**	90.3	70.6	69.8	88.9
deer	86.0	**96.6**	94.8	94.3	94.0	71.7	73.1	93.5
dog	89.1	93.1	93.3	**93.9**	93.7	71.4	70.9	93.1
frog	92.3	97.3	95.9	**97.4**	96.0	77.6	77.8	97.0
horse	91.2	**96.8**	96.0	96.2	95.3	67.4	68.5	95.1
ship	91.2	**97.3**	96.9	97.0	95.8	77.8	78.7	96.0
truck	92.9	**97.3**	96.5	90.1	96.2	76.4	73.7	95.9

All experiments pertaining to a particular target class use identical binary training/validation/test splits.

All networks (except the template, which naturally follows [11]) are trained on random batches of size of 8 using SGD with a momentum of 0.9 for 30 epochs. The learning rate is set initially to 10^{-3} and decayed by 0.1 after epoch 20. Images fed into the networks are upscaled to 224 by 224 using bilinear interpolation and normalized without any additional augmentation. We did not perform hyperparameter tuning and while this choice of hyperparameters is arbitrary, we believe it suffices for the tasks at hand as all networks were able to converge.

To generate and evaluate a component network pertaining to a target class, we fine-tune the BN and last layers (BN) of the template network on its binary training set. We then select the network that has the best validation accuracy and finally report on the test set accuracy. This is repeated for each target class in CIFAR10, resulting in 10 component networks. To generate and evaluate the ComBN/PCBN networks, we replace the BN layers in the template network with ComBN/PCBN layers built from the BNs of the 9 component networks that were not trained on the target class, and then apply the same evaluation procedure. One might object to the fact that the 9 component networks might have seen a few examples of the target class as negative examples during their original training, but we find the difference to be negligible even after carefully omitting them. Finally, for completeness, we also report on results of performing full (Full) and last layer (Last) fine-tuning on the same tasks. The results are summarized in Table 1.

For this initial set of experiments, we find that in comparison with other methods, fine-tuning just the last layer results in the lowest accuracy. We also find that results obtained by ComBN and PCBN were comparable, if not just slightly worse, with BN and full fine-tuning, suggesting that ComBN and PCBN are potentially valid methods for training networks.

To better understand the role of BN components, we also performed the Comp, ComBN and PCBN experiments using a template network that has randomized (according to [11]) convolutional layer weights. Surprisingly, this configuration still manages to achieve respectable accuracies, suggesting that random weights can still manage to map the inputs to representations which can be discriminated by the BN tranformations, while also attesting the representational power of BN layers. Even more surprising is the performance that random PCBN obtained, which we leave for future investigation.

4.2 Few-Shot Learning ImageNet32 from CIFAR10 Template

Motivated by the results in Sect. 4.1, we now attempt the more challenging task of evaluating ComBN and PCBN on ImageNet using a CIFAR10-pretrained template network. Owing to constraints on computational resources, we switch to testing on ImageNet32 [5] (which is ImageNet downsampled to 32×32 images) and use ResNet32 [11] as the template network. Additionally, to align ourselves with the original goal, we will perform the evaluations in terms of 2-way few-shot tasks.

Unlike the previous experiments, we forego the creation of a test split and will instead report on validation accuracy. We use the original ImageNet training/validation sets as our master training/validation splits, and then create binarized splits from the master splits using the same procedure outlined in Sect. 3.2. In order to ensure evaluation is performed on unseen classes, we first randomly sample 200 target classes to construct our component networks and reserve the remaining 800 for few-shot tuning and evaluation. The training/validation splits of these 200 binary datasets exhaust all available positive and negative examples from the master splits that result in a balanced dataset (i.e. about 2000 examples in the training split, and exactly 100 in the validation split). The remaining 800 binary datasets will have training/validation splits that respectively possess $2n/100$ examples per split, where n refers to a particular n-shot task.

Unless stated otherwise, all results are trained on random batches of size 128 ($2n$ during few-shot) using SGD with a momentum of 0.9 for 60 epochs and weight decay of 10^{-4}. The learning rate is set initially to 0.01, and is decayed by 0.1 after epochs 30 and 45. The model with the best validation accuracy during the training procedure is selected. No data augmentation is performed, and this choice of hyperparameters is again not optimized. The template network was trained following [11].

The following discussions refer to Table 2, where we report the mean validation accuracy of various networks trained in the few-shot training regime on the 800 binary datasets (except for Max, which uses all training data and is therefore not few-shot; we include this to illustrate a peformance upper bound). As in Sect. 4.1, we also report on the results of fine-tuning the last layer (Last), all layers (Full), and only BN + last layers (BN) of the template network to serve as baselines.

Table 2. Mean percentage validation accuracies μ on the 800 binary ImageNet32 datasets and their differences relative to full fine-tuning $\Delta = \mu - \mu_{\text{Full}}$. Asterix (*) indicates evaluation towards a subset (of about 400) of the 800 binary datasets that fulfill the 75% threshold criterion.

Setup	Component selection	No. of components	1-shot		5-shot	
			μ	Δ	μ	Δ
Max	—	—	87.6	25.1	87.6	17.6
Last	—	—	63.0	0.5	69.3	−0.7
Full	—	—	62.5	—	70.1	—
BN	—	—	62.9	0.4	69.6	−0.5
PCBN	—	200	58.2	−4.3	65.3	−4.8
ComBN	Few-shot loss	3	66.3	3.8	73.6	3.5
ComBN	Few-shot loss	5	65.8	3.3	73.3	3.2
ComBN	Few-shot loss	10	65.7	3.2	71.5	1.4
ComBN	Max-shot accuracy	3	77.2	14.7	78.3	8.3
ComBN	Max-shot accuracy	5	76.3	13.8	78.0	7.9
ComBN	Max-shot accuracy	10	72.2	9.7	75.0	4.9
ComBN	Max-shot accuracy	75% threshold*	80.0	17.5	81.6	11.5
PCBN	Few-shot loss	3	64.5	2.0	71.3	1.3
PCBN	Few-shot loss	5	63.3	0.7	70.6	0.5
PCBN	Few-shot loss	10	62.8	2.6	68.4	−1.7
PCBN	Max-shot accuracy	3	71.2	8.7	74.4	4.4
PCBN	Max-shot accuracy	5	70.8	8.3	74.5	4.4
PCBN	Max-shot accuracy	10	67.5	5.0	71.4	1.3
PCBN	Max-shot accuracy	75% threshold*	73.0	10.5	77.5	7.4
SGM	—	—	64.8	2.3	70.6	0.5
L2	—	—	57.0	−5.5	59.7	−10.4

Afterwards, we proceed to generating the 200 component networks by fine-tuning the BN and last layers of the template network using the aforementioned 200 binary datasets, and use all 200 components to construct a PCBN network. However, as we can see from the results, this construction performs worse than the baselines in both 1-shot and 5-shot tasks. This drop in performance might be attributed to the presence of components that are irrelelvant for the target task. Furthermore, it is not feasible to create a ComBN network using 200 components due to memory constraints, making it apparent that we need to selectively reduce the number of components used in ComBN/PCBN.

To do so, and as previously hinted in Sect. 3.3, we propose to evalute the cross-entropy loss of each component network on the training split of one of the 800 datasets that correspond to the unknown target class. We then rank and

select m component networks with the lowest loss and use these m networks to construct our ComBN/PCBN networks (i.e. few-shot loss component selection). The results from Table 2 seem to indicate that this strategy works, with both ComBN and PCBN outpeforming the baselines by about 4%, and $m = 3$ leading to the best results overall.

Furthermore, we also considered the case of having an ideal selection of components by assuming we were able to binarize all available training data from the master split for component selection (i.e. max-shot accuracy component selection). From here, we (i) selected m component networks with the highest validation accuracy on the target task and (ii) selected 3–10 component networks that perform above 75% in terms of validation accuracy on the target task. This resulted in a marked performance increase of about 14–17% when compared to the baseline, suggesting that component selection is an important procedure that warrants further study. Again, $m = 3$ seems to lead to the best results overall. An illustration of (i) for 3-component ComBN in the 1-shot regime for other as a function of dataset size is plotted in Fig. 2.

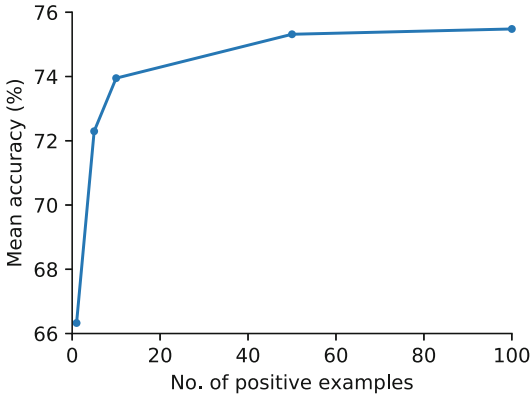
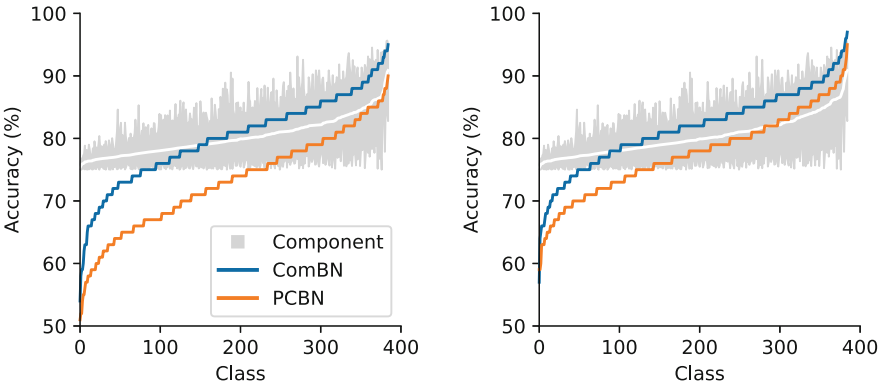


Fig. 2. Mean 1-shot validation accuracy of 3-component ComBN as a function of number of positive examples in training split used in 1-shot loss component selection.

Additionally if we plot the validation accuracies of 75% threshold components and ComBN/PCBN networks constructed from them (Fig. 3), we notice that for some classes ComBN/PCBN does worse than any of the components. This seems odd given that we know for ComBN, a solution consisting of $\alpha_j = 1$ if $j = 1$, and $\alpha_j = 0$ if $j \neq 1$ should result in an accuracy above 75%. This anomaly might be caused by the tendency of SGD to avoid this particular solution, although further work is necessary to better understand this.

As an extra benchmark, we also attempted to compare our results to [10], which is one of the few works which attempted to tackle few-shot tasks without resorting to meta-learning. We do this by training two additional template networks using SGM and L2 loss functions as described in that work, which are

theorized to create features better suited for few-shot learning. As before, we then subjected the two template networks to fine-tuning in the few-shot regime, and reported their mean validation accuracies. Set up this way, our best-performing method (3-component ComBN) outperformed theirs (SGM) by a margin of 2–3% with the few-shot loss selection, which could go up to 8–13% assuming ideal component selection.



(a) 1-shot (b) 5-shot

Fig. 3. 75% threshold component, ComBN, and PCBN network validation accuracies. The filled plot represents the minimum, mean, and maximum accuracies for each set of component networks. Each plot has been independently sorted by mean accuracy to aid visualization.

4.3 Style Transfer

To demonstrate a completely orthogonal application of our framework and generate qualitative results, we took the network proposed by [6] and used it as a template. As the network uses instance normalization (IN) layers [24], we will need to replace BNs in our method formulation with INs, resulting in composite instance normalization (ComIN) and principal component instance normalization (PCIN). The bases are formed by 32 IN parameters that were already present in the original network, and are used to learn new styles, some of which are shown in Fig. 4.

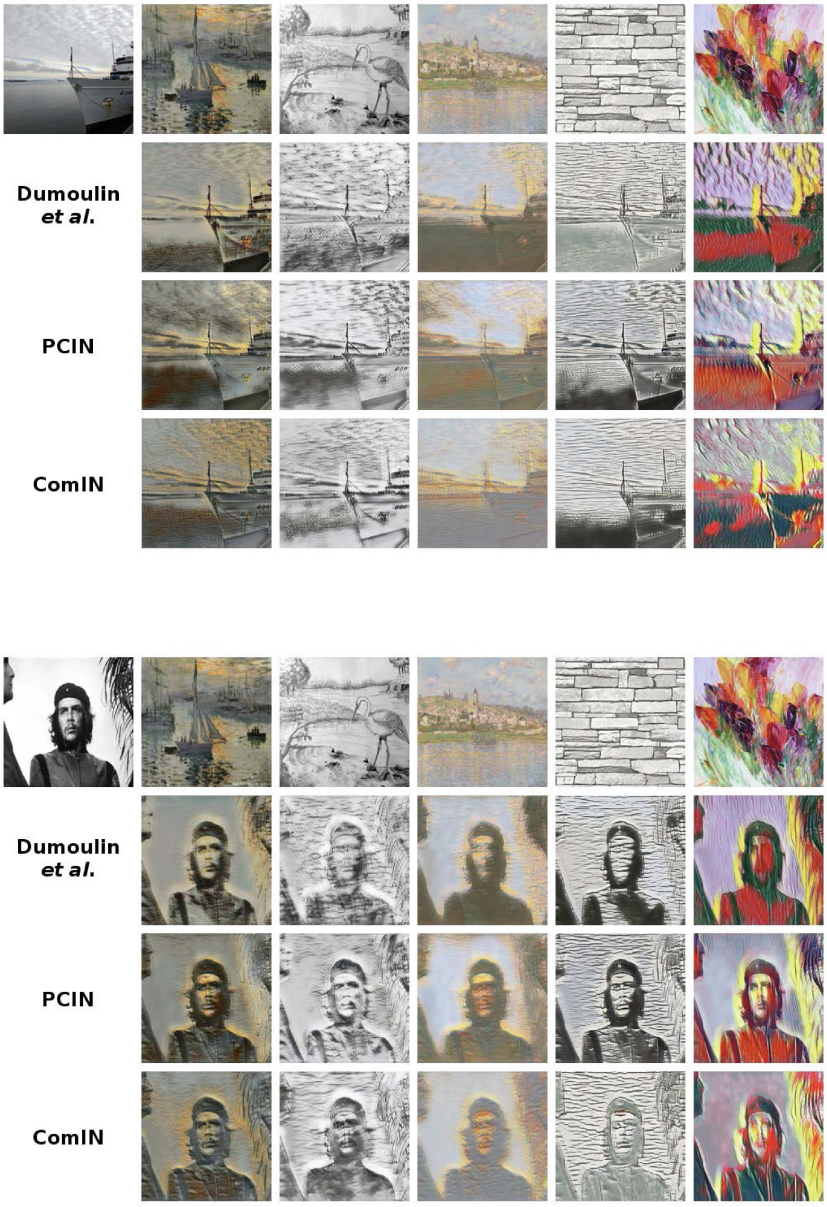


Fig. 4. Results on style transfer. Images on the top row are styles that are applied on the leftmost content image. Images on each consecutive row below are stylized images obtained from utilizing the original training procedure of Dumoulin *et al.* [6], PCIN, and ComIN, respectively.

5 Conclusions

Based on a recent idea that batch normalization modules could transform inputs to encode class-specific representations, we propose an interpolation method within learned BN layers to efficiently learn new classes. We show that this works for few-shot learning by implementing it as a linear combination of BNs (ComBN) or PCA on BN paramaters (PCBN), obtaining an accuracy between 4% to 17% over standard full fine-tuning. We have also shown that good performance is dependent on careful selection of the BN modules, and proposed a simple criterion to achieve this. Source code for the experiments can be downloaded from http://bninterp.avlcode.org/.

References

1. Aharon, M., Elad, M., Bruckstein, A.: K-SVD: an algorithm for designing overcomplete dictionaries for sparse representation. IEEE Trans. Signal Process. **54**(11), 4311–4322 (2006). https://doi.org/10.1109/TSP.2006.881199
2. Bertinetto, L., Henriques, J.F., Valmadre, J., Torr, P., Vedaldi, A.: Learning feed-forward one-shot learners. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 29, pp. 523–531. Curran Associates, Inc. (2016). http://papers.nips.cc/paper/6068-learning-feed-forward-one-shot-learners.pdf
3. Bilen, H., Vedaldi, A.: Universal representations: the missing link between faces, text, planktons, and cat breeds. CoRR abs/1701.0 (2017). http://arxiv.org/abs/1701.07275
4. Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y.: Compressing Neural Networks with the Hashing Trick. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning, Proceedings of Machine Learning Research, PMLR, Lille, France, vol. 37, pp. 2285–2294 (2015). http://proceedings.mlr.press/v37/chenc15.html
5. Chrabaszcz, P., Loshchilov, I., Hutter, F.: A downsampled variant of ImageNet as an alternative to the CIFAR datasets. CoRR abs/1707.0 (2017). http://arxiv.org/abs/1707.08819
6. Dumoulin, V., Shlens, J., Kudlur, M.: A learned representation for artistic style. CoRR abs/1610.0 (2016). http://arxiv.org/abs/1610.07629
7. Engan, K., Aase, S.O., Husoy, J.H.: Method of optimal directions for frame design. In: Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP99 (Cat. No. 99CH36258). vol. 5, pp. 2443–2446 (1999). https://doi.org/10.1109/ICASSP.1999.760624
8. Finn, C., Abbeel, P., Levine, S.: model-agnostic meta-learning for fast adaptation of deep networks. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, Proceedings of Machine Learning Research, PMLR, vol. 70, pp. 1126–1135. International Convention Centre, Sydney (2017). http://proceedings.mlr.press/v70/finn17a.html
9. Gao, Y., She, Q., Ma, J., Zhao, M., Liu, W., Yuille, A.L.: NDDR-CNN: layer-wise feature fusing in multi-task CNN by neural discriminative dimensionality reduction. CoRR abs/1801.0 (2018). http://arxiv.org/abs/1801.08297

10. Hariharan, B., Girshick, R.: Low-shot visual recognition by shrinking and hallucinating features. In: The IEEE International Conference on Computer Vision (ICCV), pp. 3018–3027, October 2017
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, June 2016
12. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS Deep Learning and Representation Learning Workshop (2015)
13. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning, Proceedings of Machine Learning Research, PMLR, Lille, France, vol. 37, pp. 448–456 (2015). http://proceedings.mlr.press/v37/ioffe15.html
14. Koch, G., Zemel, R., Salakhutdinov, R.: Siamese neural networks for one-shot image recognition. In: ICML Deep Learning Workshop (2015)
15. Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical report (2009)
16. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 25, pp. 1097–1105. Curran Associates, Inc. (2012). http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
17. Lawrence, N.: Probabilistic non-linear principal component analysis with Gaussian process latent variable models. J. Mach. Learn. Res. **6**, 1783–1816 (2005). http://dl.acm.org/citation.cfm?id=1046920.1194904
18. Luo, Z., Zou, Y., Hoffman, J., Fei-Fei, L.: Label efficient learning of transferable representations acrosss domains and tasks. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 30, pp. 165–177. Curran Associates, Inc. (2017). http://papers.nips.cc/paper/6621-label-efficient-learning-of-transferable-representations-acrosss-domains-and-tasks.pdf
19. Ravi, S., Larochelle, H.: Optimization as a model for few-shot learning. In: International Conference on Learning Representations (2017)
20. Rebuffi, S.A., Bilen, H., Vedaldi, A.: Learning multiple visual domains with residual adapters. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems, vol. 30, pp. 506–516. Curran Associates, Inc. (2017). http://papers.nips.cc/paper/6654-learning-multiple-visual-domains-with-residual-adapters.pdf
21. Rubinstein, R., Bruckstein, A.M., Elad, M.: Dictionaries for sparse representation modeling. Proc. IEEE **98**(6), 1045–1057 (2010). https://doi.org/10.1109/JPROC.2010.2040551
22. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)
23. Snell, J., Swersky, K., Zemel, R.S.: Prototypical networks for few-shot learning. CoRR abs/1703.0 (2017). http://arxiv.org/abs/1703.05175

24. Ulyanov, D., Vedaldi, A., Lempitsky, V.S.: Instance normalization: the missing ingredient for fast stylization. CoRR abs/1607.0 (2016). http://arxiv.org/abs/1607.08022
25. Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., Wierstra, D.: Matching networks for one shot learning. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 29, pp. 3630–3638. Curran Associates, Inc. (2016). http://papers.nips.cc/paper/6385-matching-networks-for-one-shot-learning.pdf