

Classification-Driven Dynamic Image Enhancement

Vivek Sharma¹, Ali Diba², Davy Neven², Michael S. Brown³, Luc Van Gool^{2,4}, and Rainer Stiefelhagen¹

¹CV:HCI, KIT, Karlsruhe, ²ESAT-PSI, KU Leuven, ³York University, Toronto, and ⁴CVL, ETH Zürich

Ffi rstname.l astnameG@ki t.edu, Ffi rstname.l astnameG@esat.kul euven.be, mbrown@eecs.yorku.ca

Abstract

Convolutional neural networks rely on image texture and structure to serve as discriminative features to classify the image content. Image enhancement techniques can be used as preprocessing steps to help improve the overall image quality and in turn improve the overall effectiveness of a CNN. Existing image enhancement methods, however, are designed to improve the perceptual quality of an image for a human observer. In this paper, we are interested in learning CNNs that can emulate image enhancement and restoration, but with the overall goal to improve image classification and not necessarily human perception. To this end, we present a unified CNN architecture that uses a range of enhancement filters that can enhance image-specific details via end-to-end dynamic filter learning. We demonstrate the effectiveness of this strategy on four challenging benchmark datasets for fine-grained, object, scene, and texture classification: CUB-200-2011, PASCAL-VOC2007, MIT-Indoor, and DTD. Experiments using our proposed enhancement show promising results on all the datasets. In addition, our approach is capable of improving the performance of all generic CNN architectures.

1. Introduction

Image enhancement methods are commonly used as pre-processing steps that are applied to improve the visual quality of an image before higher level-vision tasks, such as classification and object recognition [28, 29]. Examples include enhancement to remove the effects of blur, noise, poor contrast, and compression artifacts – or to boost image details. Examples of such enhancement methods include Gaussian smoothing, anisotropic diffusion, weighted least squares (WLS), and bilateral filtering. Such enhancement methods are not simple filter operations (e.g., 3×3 Sobel filter), but often involve complex optimization. In practice, the run time for these methods is expensive and can take seconds or even minutes for high-resolution images.

Several recent works have shown that convolutional neural networks (CNN) [2, 3, 23, 27, 39, 40] can successfully

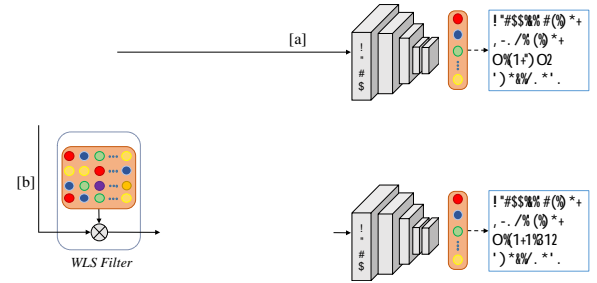


Figure 1: Overview of the proposed unified CNN architecture using enhancement filters to improve classification tasks. Given an input RGB image, instead of directly applying the CNN on this image ([a]), we first enhance the image details by convolving the input image with a WLS filter (see Sec. 3.1), resulting in improved classification with high confidence ([b]).

emulate a wide range of image enhancement by training on input and target output image pairs. These CNNs often have a significant advantage in terms of run-time performance. The current strategy, however, is to train these CNN-based image filters to approximate the output of their non-CNN counterparts.

In this paper, we propose to extend the training of CNN-based image enhancement to incorporate the high-level goal of image classification. Our contribution is a method that jointly optimizes a CNN for enhancement and image classification. We achieve this by adaptively enhancing the features on an image basis via dynamic convolutions, which enables the enhancement CNN to selectively enhance only those features that lead to improved image classification.

Since we understand the critical role of selective feature enhancement, we propose to use the dynamic convolutional layer (or dynamic filter) [7] to dynamically enhance the *image-specific* features with a classification objective (see Fig. 1). Our work is inspired by [7]. However, while [7] applies the dynamic convolutional module to transform an angle into a filter (steerable filter) using input-output image pairs, we used the same terminology as in [7]. The dynamic filters are a function of the input and therefore vary from

one sample to another during train/test time, which means when the image enhancement is done in an *image-specific* way to enhance the texture patterns or sharpen edges for discrimination. Specifically, our network learns the amount of various enhancement filters that should be applied to an input image, such that the enhanced representation provides better performance in terms of classification accuracy. Our proposed approach is evaluated on four challenging benchmark datasets for fine-grained, object, scene, and texture classification respectively: CUB-200-2011 [37], PASCAL-VOC2007 [12], MIT-Indoor [26], and DTD [4]. We experimentally show that when CNNs are combined with the proposed dynamic enhancement technique (Sec. 3.1 and 3.3), one can consistently improve the classification performance of vanilla CNN architectures on all the datasets. In addition, our experiments demonstrate the full capability of the proposed method, and show promising results in comparison to the state-of-the-art.

The remainder of this paper is organized as follows. Section 2 overviews related work. Section 3 describes our proposed enhancement architecture. Experimental results and their analysis are presented in Sections 4. Finally, the paper is concluded in Section 5.

2. Background and Related Work

Considerable progress has been seen in the development for removing the effects of blur [2], noise [27], and compression artifacts [38] using CNN architectures. Reversing the effect of these degradations in order to obtain sharp images is currently an active area of research [2, 22, 39]. The investigated CNN frameworks [2, 3, 15, 22, 23, 27, 39, 40] are typically built on simple strategies to train the networks by minimizing a global objective function using input-output image pairs. These frameworks encourage the output to have a similar structure with the target image. After training the CNN, a similar approach to transfer details to new images has been proposed [39]. These frameworks act as a filter that are specialized for a specific enhancement method.

For example, Xu et al. [39] learn a CNN architecture to approximate existing edge-aware filters from input-output image pairs. Chen et al. [3] learn a CNN that approximates end-to-end several image processing operations using a parameterization that is deeper and more context-aware. Yan et al. [40] learn a CNN to approximate image transformations for image adjustment. Fu et al. [15] learn a CNN architecture to remove rain streaks from an image. For CNN training, the authors use rainy and clean image detail layer pairs rather than the regular RGB images. Li et al. [22] propose a learning-based joint filter using three CNN architectures. In Li et al.'s work, two sub-networks take target and guidance images, while the third-network selectively transfers the main content structure and reconstructs the desired output. Remez et al. [27] propose a fully convolutional

CNN architecture to do image denoising using image prior—that is, class-aware information. The closest work to ours is by Chakrabarty et al. [2] and Liu et al. [23]. Chakrabarty et al. propose a CNN architecture to predict the complex Fourier coefficients of a deconvolution filter which is applied to individual image patches for restoration. Liu et al. use CNN+RNNs to learn enhancement filters; here we use CNNs only for learning filters. Our methods produce one representative filter per method, while they produce four-way directional propagation filters per method. Like others, their work is meant for low-level vision tasks similar to [2, 3], while our goal is enhancement for classification. In contrast to these prior works, our work differs substantially in scope and technical approach. Our goal is to approximate different image enhancement filters with a classification objective in order to selectively extract informative features from enhancement techniques to improve classification, not necessarily approximating the enhancement methods.

Similar to our goal are the works [6, 9, 19, 25, 35, 36], where the authors also seek to ameliorate the degradation effects for accurate classification. Dodge and Karam [9] analyzed how blur, noise, contrast, and compression hamper the performance of ConvNet architectures for image classification. Their findings showed that: (1) ConvNets are very sensitive to blur because blur removes textures in the images; (2) noise affects the performance negatively, though deeper architectures' performance falls off slower; and (3) deep networks are resilient to compression distortions and contrast changes. A study by Karahan et al. [19] reports similar results for a face-recognition task. Ullman et al. [35] showed that minor changes to the image, which are barely perceptible to humans, can have drastic effects on computational recognition accuracy. Szegedy et al. [32] showed that applying an imperceptible non-random perturbation can cause ConvNets to produce erroneous prediction.

To help to mitigate these problems, Costa et al. [6] designed separate models specialized for each noisy version of an augmented training set. This improved the classification results for noisy data to some extent. Peng et al. [25] explored the potential of jointly training on low-resolution and high-resolution images in order to boost performance on low-resolution inputs. Similar to [25] is Vasišević et al.'s [36] work, where the authors augment the training set with degradations and fine-tune the network with a diverse mix of different types of degraded and high-quality images to regain much of the lost accuracy on degraded images. In fact, with this approach the authors were able to learn to generate a degradation (particularly blur) invariant representation in their hidden layers.

In contrast to previous works, we use high-quality images that are free of artifacts, and jointly learn ConvNet to enhance the image for the purpose of improving classification.

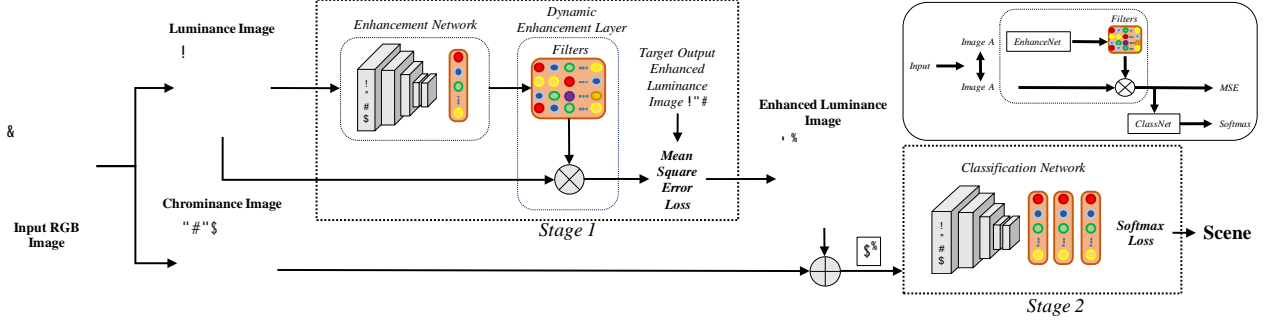


Figure 2: **Dynamic enhancement filters.** Input to the network are input-output image pairs, as well as image class labels for training. In this architecture, we **learn a single enhancement filter** for each enhancement method individually. The model operates on the luminance component of RGB color space. The enhancement network (i.e., filter-generating network) generates dynamic filter parameters that are *sample-specific* and conditioned on the input of the enhancement network, with the overall goal to improve image classification. The figure in the upper-right corner shows the whole pipeline workflow.

3. Proposed Method

As previously mentioned, our aim is to learn a dynamic image enhancement network with the overall goal to improve classification, and not necessarily approximating the enhancement methods specifically. To this end, we propose three CNN architectures described in this section.

Our first architecture is proposed to learn a single enhancement filter for each enhancement method in an end-to-end fashion (Sec. 3.1) and by end-to-end we mean each image will be enhanced and recognized in one unique deep network with dynamic filters. Our second architecture uses pre-learned enhancement filters from the first architecture and combines them in a weighted way in the CNN. There is no adaptation of weights of the filters (Sec. 3.2). In our third architecture, we show end-to-end joint learning of multiple enhancement filters (Sec. 3.3). We also combine them in a weighted way in the CNN. All these setups are jointly optimized with a classification objective to selectively enhance the image feature quality for improved classification. In the network training, image-level class labels are used, while for testing the input image can have multiple labels.

3.1. Dynamic Enhancement Filters

In this section we describe our model to learn representative enhancement filters for different enhancement methods from input and target output enhanced image pairs in the end-to-end learning approach with a goal to improve classification performance. Given an input RGB image I , we first transform it into the luminance-chrominance $YCbCr$ color space. Our enhancement method operates on the luminance component [14] of the RGB image. This allows our filter to modify the overall tonal properties and sharpness of the image without affecting the color. The luminance image $Y \in \mathbb{R}^{h \times w}$ is then convolved with an image enhancement method E : $Y \rightarrow T$, resulting in an enhanced target output luminance image $T \in \mathbb{R}^{h \times w}$, where h , and w denote the height and width in the input Y respectively. We generate

target images for a range of enhancement methods E as a preprocessing step (see Section. 4.2 for more details). For filter generation, we explicitly use a dataset of only one enhancement method at a time for learning the transformation. The scheme is illustrated in Figure 2.

First stage (enhancement stage): The enhancement network (EnhanceNet) is inspired by [7, 18, 20], and is composed of convolutional and fully-connected layers. The EnhanceNet maps the input to the filter. The enhancement network takes the one channel luminance image Y and outputs filters f , $\mathbb{R}^{s \times s \times n}$, where s is the parameters of the transformation generated dynamically by the enhancement network, s is the filter size, and n is the number of filters, being equal to 1 for a single generated filter meant for one channel luminance image. The generated filter is applied to the input image $Y(i, j)$ at every spatial position (i, j) to output predicted image $\hat{Y}(i, j) = f(Y(i, j))$ with $\hat{Y} \in \mathbb{R}^{h \times w}$. The filters are *image-specific*, and are conditioned on Y . For generating the enhancement filter parameters, the network is trained using mean squared error (MSE) between the target image T and the network’s predicted output image \hat{Y} . Note that, the parameters of the filter are obtained as the output of a EnhanceNet that maps the input to a filter and therefore vary from one sample to another. To compare the reconstruction image \hat{Y} with the ideal T , we use MSE loss as a measure of image quality, although we note that more complex loss functions could be used [10].

The chrominance component is then recombined, and the image is transformed back into RGB, \hat{I} . We found that the filters learned the expected transformation and applied the correct enhancement to the image. Figure 5 shows qualitative results with dynamically enhanced image textures.

Second stage (classification stage): The predicted output image \hat{I} from Stage 1 is fed as an input to the classification network (ClassNet). As the classification network (e.g., Alexnet [21]) has fully-connected layers between the last convolutional layer and the classification layer, the param-

eters of the fully-connected layer and C-way classification layer are learned when fine-tuning a pre-trained network.

End-to-end learning: The Stage 1-2 cascade with two loss functions - MSE (enhancement) and softmax-loss L (classification) - enables joint optimization by end-to-end propagation of gradients in both ClassNet and EnhanceNet using SGD optimizer. The total loss function of the whole pipeline is given by:

$$\text{Loss}_{\text{Filters}} = \text{MSE}(T, Y) + L(P, y)$$

$$P_q = \frac{\exp(a_q)}{\sum_{r=1}^C \exp(a_r)}, L(P, y) = - \sum_{q=1}^C y_q \log(P_q) \quad (1)$$

where a is the output of the last fully-connected layer of ClassNet that is fed to a C-way softmax function, y is the vector of true labels for image I , and C is the number of classes.

We fine-tune the whole pipeline until convergence, thus leading to learned enhancement filters in the dynamic enhancement layer. The joint optimization allows the loss gradients from the ClassNet to also back-propagate through the EnhanceNet, making the filter parameters also optimized for classification.

3.2. Static Filters for Classification

Here, we show how to integrate the pre-learned enhancement filters obtained from the first approach. For each image in the train set, we obtain a dynamic filter using our first approach. The static filter is computed by taking a mean of all these dynamic filters. The extracted static filters are convolved with the input luminance Y component of the RGB image I , and the chrominance component is added and then the image is transformed back to RGB I' , which is then fed into the classification network. Figure 3 shows the schematic layout of the whole architecture.

First stage (enhancement stage): We begin by extracting the pre-trained filters for K image enhancement methods learned from the first approach. Given an input luminance image Y , these $f_{\cdot,k}$ filters are convolved with the input image to generate Y_k enhanced images as $Y_k = f_{\cdot,k}(Y)$, $k = 1 \dots K$. We also include an identity filter ($K+1$) to generate the original image, as some learned enhancements may perform worse than the original image itself. We then investigate two different strategies to weight W_k the enhancement methods: (1) giving equal weights with value equal to $1/K$, and (2) giving weights on the basis of MSE, as discussed in Sec. 3.3.

The output of this stage is a set of enhanced luminance images and their corresponding weights indicating the potential importance for pushing to the next stage of the classification pipeline. Chrominance is then recombined, and the images are transformed back to RGB, I_k .

Second stage (classification stage): The enhanced images I_k for K image enhancement methods and original image are fed as an input to the classification network one by one sequentially, with class labels and their weights W_k indicating the importance of the enhancement method for the input image. Similar to the last approach, the network parameters of the fully-connected layer and C-way classification layer are fine-tuned using a pre-trained network in an end-to-end learning approach.

End-to-end training: The loss of the network training is the weighted W_k sum of the individual softmax losses L_k term. The weighted loss is given as:

$$\text{Loss}_{\text{Stat}} = \sum_{k=1}^{K+1} W_k L_k(P, y) \quad (2)$$

where W is the weight indicating the importance of the K enhancement method, where $W_{K+1} = 1$ for original RGB image, contributing to the total loss of the whole pipeline.

3.3. Multiple Dynamic Filters for Classification

Here, we recycle the architectures from Sections 3.1-3.2. Figure 4 shows the schematic layout of the whole architecture. Our architecture uses the similar architecture proposed in Sec. 3.1; we dynamically generate K filters using K enhancement networks, one for each enhancement method. In this proposed architecture, the loss associated with Stage 1 is the MSE between the predicted output images Y_k and the target output images T_k .

For computing the weights of each enhancement method, the MSE for the enhanced images are transformed to weights W_k by comparing their relative strengths as: $W_k = \text{MSE}_k / \sum_{m=1}^K \text{MSE}_m$, followed by $W_k = (W_k - \max(W)) / (\min(W) - \max(W))$. Since now the $\min(W)$ is zero, in order to avoid giving zero-weight to one of the enhancement methods, we subtract the *second-min*(W)/2 from W and add the *second-min*(W) to the W_k with $\min(W)$. Finally, we scale the weights $W_k = W_k / \sum_{m=1}^K W_m$ with the constraint that the sum of the weights for all K methods should be equal to 1. The enhanced images with the smallest errors obtain the highest weight, and vice versa. In addition, we also compare against giving equal weights to all enhancement methods. Of both weighting strategies, MSE-based weighting yielded the best results, and was therefore selected as the default. Note that we also include the original image by simply convolving it with an identity filter ($K+1$) similar to approach 2: the weight for the RGB image is set to 1, i.e. $W_{K+1} = 1$. During training, the weights are estimated by cross-validation on the train/validation set, while for the testing phase, we use these pre-computed weights. Further, we observed that training the network without regularization of weights has prevented the model from converging throughout the learn-

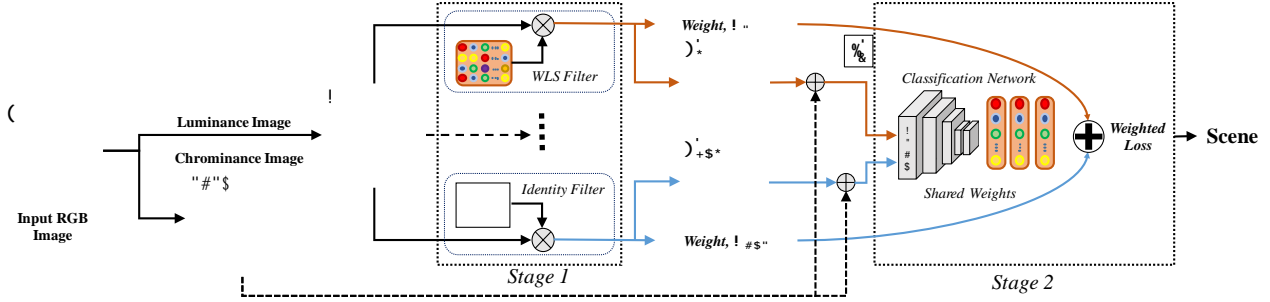


Figure 3: **(Stat-CNN)** In this architecture, we use pre-learned filters from Sec. 3.1 (Figure 2) for image enhancement and the original image. The individual softmax scores are combined in a weighted way in the CNN. There is **no adaptation** of weights of the filters.

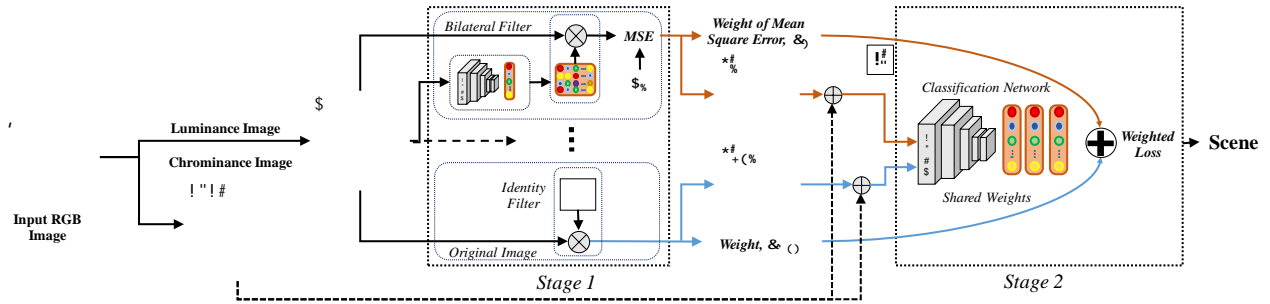


Figure 4: **(Dyn-CNN)** In this architecture, similar to Sec. 3.1 (Figure 2), we show end-to-end joint **learning of multiple filters**. The individual softmax scores are combined in a weighted way in the CNN. There is **adaptation** of weights of the filters.

ing, and led to overfitting with significant drop in performance.

End-to-end training: Finally, we now extend the loss of approach 2, by adding an MSE term for joint optimization of K enhancement networks with a classification objective. We learn all parameters of the network jointly in an end-to-end fashion. The weighted loss is *sample-specific*, and is given as:

$$\text{Loss}_{\text{Dyn}} = \sum_{k=1}^K \text{MSE}_k(T_k, Y_k) + \sum_{k=1}^{K+1} W_k L_k(P, y) \quad (3)$$

We believe training our network in this manner, offers a natural way to encourage the filters to apply a transformation that enhances the image structures for an accurate classification, as the classification network is regularized via enhancement networks. Moreover, joint optimization helps minimize the overall cost function of the whole architecture, hence leading to better results.

4. Experiments

In this section, we demonstrate the use of our enhancement filtering technique on four very different image classification tasks. First, we introduce the dataset, target output data generation, and implementation details, and explore the design choices of the proposed methods. Finally, we test and compare our proposed method with baseline methods and other current ConvNet architectures. Note that, the

purpose of this paper is to improve the baseline performance of generic CNN architectures using an *add-on* enhancement filters, and not to compete against the state-of-the-art methods.

4.1. Datasets

We evaluate our proposed method on four visual recognition tasks: fine-grained classification using CUB-200-2011 CUB) [37], object classification using PASCAL-VOC2007 (PascalVOC) [12], scene recognition using MIT-Indoor-Scene (MIT) [26], and texture classification using Describable Textures Dataset (DTD) [4]. Table 1 shows the details of the datasets. For all of these datasets, we use the standard training/validation/testing protocols provided as the original evaluation scheme and report the classification accuracy.

4.2. Target Output Data

We generate target output T images for five (i.e., $K=5$) enhancement methods E : (1) weighted least squares (WLS) filter [13], (2) bilateral filter (BF) [11, 34], (3) image sharp-

Data-set	# train img	# test img	# classes
CUB [37]	5994	5794	200
PascalVOC [12]	5011	4952	20
MIT [26]	4017	1339	67
DTD [4]	1880	3760	47

Table 1: Details of the training and test set for datasets.

ening filter (Imsharp), (4) guided filter (GF) [16], and (5) histogram equalization (HistEq). Given an input image, we first transform the RGB color space into a luminance-chrominance color space, and then apply these enhancement methods on the luminance image to obtain an enhanced luminance image. This enhanced luminance image is then used as the target image for training. We used default parameters for WLS and Imsharp, and for BF, GF and HistEq parameters are adapted to each image, thus requiring no parameter setting. For comprehensive discussion, we refer the readers to [11, 13, 16]. The source code for fast BF [11], WLS [13] is publicly available, and others are available in the Matlab framework.

4.3. Implementation Details

We use the MatCovNet and Torch frameworks, and all the ConvNets are trained on a TitanX GPU. Here we discuss the implementation details for ConvNet training (1) with dynamic enhancement filter networks, (2) with static enhancement filters, and (3) without enhancement filters as a classic ConvNet training scenario.

We evaluate our design on AlexNet [21], GoogleNet [31], VGG-VD [30], VGG-16 [30], and BN-Inception [17]. In each case, the models are pre-trained on the ImageNet [8] and then fine-tuned on the target datasets. To fine-tune the network, we replace the 1000-way classification layer with a C-way softmax layer, where C is the number of classes in the target dataset. For fine-tuning the different architectures depending on the dataset about 60-90 epochs (batch size 32) were used, with a scheduled learning rate decrease, starting with a small learning rate 0.01. All ConvNet architectures are trained with identical optimization schemes, using SGD optimizer with a fixed weight decay of 5×10^{-4} and a scheduled learning rate decrease. We follow two steps to fine-tune the whole network. First, we fine-tune (last two fc layers) the ConvNet architecture using RGB images, and then embed it in Stat/Dyn-CNN for fine-tuning the whole network with enhancement filters, by setting a small learning rate for all layers except the last two fc layers, which have a high learning rate. Specifically, for example, in BN-Inception the network requires a fixed input size of 224×224 . The images are mean-subtracted before network training. We apply data augmentation [21, 30] by cropping the four corners, center, and their x-axis flips, along with color jittering (and the cropping procedure repeated for each of these) for network training. Ahead we provide more details for ConvNet training using BN-Inception.

–**Dynamic enhancement filters (Dyn-CNN):** The enhancement network consists of 570k learnable model parameters, with the last fully-connected layer (i.e., dynamic filter parameters) containing 36 neurons - that is, filter-size 6×6 . We initialize the enhancement networks' model pa-

rameters randomly, except for the last fully-connected layer, which is initialized to regress the identity transform (zero weights, and identity transform bias), suggested in [18]. We initialize the learning rate with 0.01 and decrease it by a factor of 10 after every 15k iterations. The maximum number of iterations is set to 90k. In terms of computation speed, the training enhancement network along with BN-Inception takes approx. 7% more training time for network convergence in comparison to BN-Inception for approach 1 (Sec. 3.1). We use five enhancement networks for generating five enhancement filters (one for each method) for approach 3 (Sec. 3.3). We also include original RGB image too.

–**Without enhancement filters (FC-CNN):** Similar to classical ConvNets' fine-tuning scenario, we replace the last classification layer of a pre-trained model with a C-way classification layer before fine-tuning. The fully connected layers and the classification layer are fine-tuned. We initialize the learning rate with 0.01 and decrease it by a factor of 10 after every 15k iterations. The maximum number of iterations is set to 45k.

–**Static enhancement filters (Stat-CNN):** Similar to FC-CNN, here, we have five enhanced images for five static filters and original RGB image as the sixth input that are fed as input to the ConvNets for network training. In practice, the static filters for image enhancement are very low-complex operations. The optimization scheme used here is the same as FC-CNN. We use all the five static learned filters for approach 2 (Sec. 3.2).

Testing: As previously mentioned, the input RGB image is transformed into luminance-chrominance color space, and then the luminance image is convolved with the enhancement filter, leading to an enhanced luminance image. Chrominance is then recombined to the enhanced luminance image and the image is transformed back to RGB. For ConvNet testing, an input frame with either be an RGB image or an enhanced RGB image using the static or dynamic filters is fed into the network. In total, five enhanced images (one for each filter) and the original RGB image are fed into the network, sequentially. For final image label prediction, the predictions of all images are combined through a weighted sum, where the pre-computed weights W are obtained from Dyn-CNN.

4.4. Fine-Grained Classification

In this section, we use CUB-200-2011 [37] dataset as a test bed to explore the design choices of our proposed method, and then finally compare our method with baseline methods and the current methods.

Dataset: CUB [37] is a fine-grained bird species classification dataset. The dataset contains 20 bird species with 11,788 images. For this dataset, we measure the accuracy of predicting a class for an image.

	RGB	BF	WLS	GF	HistEq	Imsharp	LF
(a) GT-EMs	67.3 [24]	66.93	67.34	67.12	66.41	66.74	70.14
(b) RGB: GT-EMs	—	67.16	67.41	67.37	66.58	66.87	71.28
(c) Ours (Sec. 3.1)	—	68.21	68.73	68.5	67.62	67.86	72.16

Table 2: Individual accuracy (%) performance comparison of all the enhancement methods E using AlexNet on CUB, where LF is *late-fusion* as averaging of scores for the 5 enhancement methods.

Ablation study: Here we explore four aspects of our proposed method: (1) the impact of different filter size; (2) the impact of each enhancement method, separately; (3) the impact of weighting strategies; and (4) the impact of different ConvNet architectures.

– **Filter size:** In our experiment, we explore three different filter sizes. Specifically, we implement the enhancement network as a few convolutional and fully-connected layers with the last-layer containing (1) 25 neurons (f is an enhancement filter of size 5×5), (2) 36 neurons (6×6), and (3) 49 neurons (7×7). From the literature [7, 18], we exploited the insights about good filter size. The filter size determines the receptive field and is application dependent. We found that a filter size $> 7 \times 7$ produces smoother images, and thus drops the classification performance by approx. 2% (WLS: 68.73 → 66.84) in comparison to a filter size of 6×6 . Similar was the case with a filter size $< 5 \times 5$, where correct enhancement was not transferred, leading to a drop in performance by approx. 3% (WLS: 68.73 → 65.9). We found that the filter size 6×6 learned the expected transformation, and applied the correct enhancement to the input image with sharper preserved edges.

– **Enhancement method (E):** Here, we compare the performance of individual enhancement methods in three aspects: (1) We employ AlexNet [21] pre-trained on ImageNet [8] and fine-tuned (last two fc layers) on CUB for each ground-truth enhancement method separately (GT-EMs). (2) Using the pre-trained RGB AlexNet model on CUB from (1), we fine-tune the whole model with GT-EMs, by setting a small learning rate for all layers except the last two fc layers, which have a high learning rate. This slightly improves the performance of the pre-trained RGB model by a small margin. (3) Similar to (2), but here we fine-tune the whole model using approach 1 (Sec. 3.1). We can see that our dynamic enhancement approach improves the performance by a margin of 1-1.5% in comparison to a generic network when fine-tuned on RGB images only. In Table 2, we summarize the results.

In Fig. 5, as an example we show some qualitative results for the difference in textures that our enhancement method extracts from the GT-EMs, which is primarily responsible for improving the classification performance.

– **Weighting strategies:** Combining the enhancement methods in a late-fusion (LF) as an averaging of the scores gives further improvements, shown in Table 2. With this observation, we realized a more effective weighting strat-

egy should be applied, such that more importance could be given to better methods for combining. In our evaluation, we explore two weighting strategies (1) giving equal weights W_k with value equal to $1/K$ - that is, 0.2 for $K=5$, and (2) weight computed on the basis of MSE, estimated by cross-validation on the training set, shown in Table 3.

Table 4 clearly shows that weighting adds a positive regularization effect. We found that training the network with regularization of MSE loss prevents the classification objective from divergence throughout the learning. Table 3 shows that in Dyn-CNN the weight of each enhancement filter relates very well to that of its individual performance shown in Table 2. We observe that the MSE-based weighting performs the best. Therefore, we choose that as a default weighting method.

– **ConvNet architectures:** Here, we compare the different ConvNet architectures. Specifically, we compare AlexNet [21], GoogLeNet [31], and BN-Inception [17]. Among all architectures shown in Table 5, BN-Inception exhibits the best performance in terms of classification accuracy in comparison to others. Therefore, we choose BN-Inception as a default architecture for this experiment.

Results: In Table 6, we explore our static and dynamic CNNs with the current methods. We consider BN-Inception using our two-step fine-tuning scheme with Stat-CNN and Dyn-CNN. We can notice that Dyn-CNN improves the generic BN-Inception performance by **3.82%** (82.3 → 86.12) using image enhancement. Our EnhanceNet takes a constant time of only **8 ms** (GPU) to generate all enhanced images altogether, in comparison to generating ground-truth

	BF	WLS	GF	HistEq	Imsharp	RGB
W	0.23±0.05	0.25±0.04	0.24±0.03	0.13±0.03	0.17±0.05	1.0

Table 3: Relative comparison of the strength of weights W for each enhancement method estimated by cross-validation on the training set of CUB using Dyn-CNN with BN-Inception, where W for RGB image by default is set to 1.

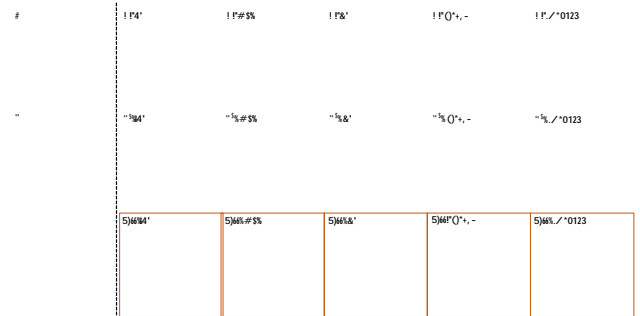


Figure 5: **Qualitative results:** CUB. Comparison between the target image T , enhanced luminance image Y , and compliment of difference image ($\text{diff}=T-Y$) obtained using approach 1 (Sec. 3.1) for all enhancement methods.

	Stat-CNN (ours)	Dyn-CNN (ours)
W: Averaging	83.19	85.58
W: MSE-based	83.74	86.12

Table 4: Accuracy (%) performance comparison of the weighting strategies using BN-Inception on CUB.

target images, which is very time-consuming and takes 1-6 seconds for each image/method: BF, WLS, and GF. Testing time for the whole model is: EnhanceNet (8 ms) plus ClassNet (inference time for the architecture used).

Further, we extend the baseline $2\times$ ST-CNN [18] to include static (Sec 3.2) and dynamic filters (Sec. 3.3) immediately following the input, with the weighted loss. In reference to ST-CNN work [18], we evaluate the methods, keeping the training and evaluation setup the same for a fair comparison. Our results indicate that Dyn-CNN improves the performance by **3.81%** (83.1 86.91). Furthermore, our Stat-CNN with static filters is competitive too, and performs 1.15% better than $2\times$ ST-CNN [18]. This means that the static filters when dropped into a network can perform explicit enhancement of features, and thus gains in accuracy are expected in any ConvNet architecture.

4.5. Object Classification

Dataset: The PASCAL-VOC2007 [12] dataset contains 20 object classes with 9,963 images that contain a total of 24,640 annotated objects. For this dataset, we report the mean average precision (mAP), averaged over all classes.

Results: In Table 7, we show the results. Dyn-CNN is **4.58/6.16%** better than Stat-CNN/FC-CNN using AlexNet, and **2.43/3.5%** using VGG-16. One can observe that for a smaller network, AlexNet shows more improvement in performance in comparison to the deeper: VGG-16 network. Also, Stat-CNN is 1.58/1.07% better than FC-CNN using AlexNet/VGG-16. Furthermore, Bilen et al. [1] with 89.7% mAP performs **3.1%** (89.7 92.8) lower than Dyn-CNN using VGG-16.

4.6. Indoor Scene Recognition

Dataset: The MIT-Indoor scene dataset (MIT) [26] contains a total of 67 indoor scene categories with 5,356 images. For this dataset, we measure the accuracy of predicting a class for an image.

Results: In Table 7, we show the results. As expected and previously observed, Dyn-CNN is **4.66/6.11%** better than Stat-CNN/FC-CNN using AlexNet, and **2.73/3.8%** using VGG-16.

	FC-CNN	Stat-CNN (ours)	Dyn-CNN (ours)
AlexNet	67.3 [24]	68.52	73.57
GoogLeNet	81.0 [24]	82.35	84.91
BN-Inception	82.3 [18]	83.74	86.12

Table 5: Accuracy (%) performance comparison of different architectures on CUB.

	FC-CNN	Stat-CNN (ours)	Dyn-CNN (ours)
$4\times$ ST-CNN [18]: 448px	84.1 [18]	—	—
BN-Inception	82.3 [18]	83.74	86.12
$2\times$ ST-CNN [18]	83.1 [18]	84.25	86.91

Table 6: **Fine-grained classification (CUB).** Accuracy (%) performance comparison of Stat-CNN and Dyn-CNN with baseline methods and previous works on CUB.

Dataset	ConvNet	FC-CNN	Stat-CNN (ours)	Dyn-CNN (ours)
PascalVOC (mAP)	AlexNet	76.9 [33]	78.48	83.06
PascalVOC (mAP)	VGG-16	89.3 [30]	90.37	92.8
MIT (Acc.)	AlexNet	56.79 [41]	58.24	62.9
MIT (Acc.)	VGG-16	64.87 [41]	65.94	68.67
DTD (mAP)	AlexNet	61.3 [5]	62.9	67.81
DTD (mAP)	VGG-VD	67.0 [5]	69.12	71.34

Table 7: **Performance comparison in %.** The table compares FC-CNN, Stat-CNN, and Dyn-CNN on AlexNet and VGG networks trained on ImageNet and fine-tuned on target datasets using the standard training and testing sets.

4.7. Texture Classification

Dataset: The Describable Texture Datasets (DTD) [4] contains 47 describable attributes with 5,640 images. For this dataset, we report the mAP, averaged over all classes.

Results: In Table 7, we show the results. The story is similar to our previous observation: Dyn-CNN outperforms Stat-CNN and FC-CNN by a significant margin. Surprisingly, it is interesting to see that Dyn-CNN shows a significant improvement of **6.51/4.34%** in comparison to FC-CNN using AlexNet/VGG-VD.

5. Concluding Remarks

In this paper, we propose a unified CNN architecture that can emulate a range of enhancement filters with the overall goal to improve image classification in an end-to-end learning approach. We demonstrate our framework on four benchmark datasets: PASCAL-VOC2007, CUB-200-2011, MIT-Indoor Scene, and Describable Textures Dataset. In addition to improving the baseline performance of vanilla CNN architectures on all datasets, our method shows promising results in comparison to the state-of-the-art using our static/dynamic enhancement filters. Also, our enhancement filters can be used with any existing networks to perform explicit enhancement of image texture and structure features, giving CNNs higher-quality features to learn from, which in turn can lead to more accurate classification.

We believe our work opens many possibilities for further exploration. In future work, we plan to further investigate more enhancement methods as well as more complex loss functions which are appropriate for image enhancement tasks.

Acknowledgements: This work is supported by the DFG, a German Research Foundation - funded PLUMCOT project, the CREF VISTA programme, and an NSERC Discovery Grant.

References

- [1] H. Bilen and A. Vedaldi. Weakly supervised deep detection networks. In *CVPR*, 2016.
- [2] A. Chakrabarti. A neural approach to blind motion deblurring. In *ECCV*, 2016.
- [3] Q. Chen, J. Xu, and V. Koltun. Fast image processing with fully-convolutional networks. In *ICCV*, 2017.
- [4] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *CVPR*, 2014.
- [5] M. Cimpoi, S. Maji, I. Kokkinos, and A. Vedaldi. Deep filter banks for texture recognition, description, and segmentation. *IJCV*, 2016.
- [6] G. B. P. da Costa, W. A. Contato, T. S. Nazare, J. E. Neto, and M. Ponti. An empirical study on the effects of different types of noise in image classification tasks. *arxiv:1609.02781*, 2016.
- [7] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool. Dynamic filter networks. In *NIPS*, 2016.
- [8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.
- [9] S. Dodge and L. Karam. Understanding how image quality affects deep neural networks. In *QoMEX*, 2016.
- [10] A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In *NIPS*, 2016.
- [11] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *ACM TOG*, 2002.
- [12] M. Everingham, A. Zisserman, C. K. Williams, L. Van Gool, M. Allan, C. M. Bishop, O. Chapelle, N. Dalal, T. Deselaers, G. Dorkó, et al. The pascal visual object classes challenge 2007 (voc2007) results. 2007.
- [13] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. In *ACM TOG*, 2008.
- [14] C. Fredembach and S. Süsstrunk. Colouring the near-infrared. In *CIC*, 2008.
- [15] X. Fu, J. Huang, X. Ding, Y. Liao, and J. Paisley. Clearing the skies: A deep network architecture for single-image rain removal. *arxiv:1609.02087*, 2016.
- [16] K. He, J. Sun, and X. Tang. Guided image filtering. In *ECCV*, 2010.
- [17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.
- [18] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *NIPS*, 2015.
- [19] S. Karahan, M. K. Yildirim, K. Kirtac, F. S. Rende, G. Butun, and H. K. Ekenel. How image degradations affect deep cnn-based face recognition? In *BIOSIG*, 2016.
- [20] B. Klein, L. Wolf, and Y. Afek. A dynamic convolutional layer for short range weather prediction. In *CVPR*, 2015.
- [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.
- [22] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang. Deep joint image filtering. In *ECCV*, 2016.
- [23] S. Liu, J. Pan, and M.-H. Yang. Learning recursive filters for low-level vision via a hybrid neural network. In *ECCV*, 2016.
- [24] K. Matzen and N. Snavely. BubbleNet: Foveated imaging for visual discovery. In *ICCV*, pages 1931–1939, 2015.
- [25] X. Peng, J. Hoffman, X. Y. Stella, and K. Saenko. Fine-to-coarse knowledge transfer for low-res image classification. In *ICIP*, 2016.
- [26] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *CVPR*, 2009.
- [27] T. Remez, O. Litany, R. Giryes, and A. M. Bronstein. Deep class aware denoising. *arxiv:1701.01698*, 2017.
- [28] V. Sharma, J. Y. Hardeberg, and S. George. Rgb-nir image enhancement by fusing bilateral and weighted least squares filters. *Journal of Imaging Science and Technology*, 2017.
- [29] V. Sharma and L. Van Gool. Does v-nir based image enhancement come with better features? *arXiv preprint arXiv:1608.06521*, 2016.
- [30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.
- [31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.
- [32] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arxiv:1312.6199*, 2013.
- [33] P. Tang, X. Wang, B. Shi, X. Bai, W. Liu, and Z. Tu. Deep fishernet for object classification. *arxiv:1608.00182*, 2016.
- [34] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *ICCV*, 1998.
- [35] S. Ullman, L. Assif, E. Fetaya, and D. Harari. Atoms of recognition in human and computer vision. *National Academy of Sciences*, 2016.
- [36] I. Vasiljevic, A. Chakrabarti, and G. Shakhnarovich. Examining the impact of blur on recognition by convolutional networks. *arxiv:1611.05760*, 2016.
- [37] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.
- [38] L. Xu, J. S. Ren, C. Liu, and J. Jia. Deep convolutional neural network for image deconvolution. In *NIPS*, 2014.
- [39] L. Xu, J. S. Ren, Q. Yan, R. Liao, and J. Jia. Deep edge-aware filters. In *ICML*, 2015.
- [40] Z. Yan, H. Zhang, B. Wang, S. Paris, and Y. Yu. Automatic photo adjustment using deep neural networks. *ACM TOG*, 2016.
- [41] B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva. Places: An image database for deep scene understanding. *arxiv:1610.02055*, 2016.