EUROGRAPHICS 2016 / J. Jorge and M. Lin
(Guest Editors)

Volume 35 (2016), Number 2

DeepProp: Extracting Deep Features from a Single Image for Edit Propagation

Yuki Endo[1] Satoshi Iizuka[2] Yoshihiro Kanamori[1] Jun Mitani[1]

[1] University of Tsukuba [2] Waseda University / JST CREST

Abstract

Edit propagation is a technique that can propagate various image edits (e.g., colorization and recoloring) performed via user strokes to the entire image based on similarity of image features. In most previous work, users must manually determine the importance of each image feature (e.g., color, coordinates, and textures) in accordance with their needs and target images. We focus on representation learning that automatically learns feature representations only from user strokes in a single image instead of tuning existing features manually. To this end, this paper proposes an edit propagation method using a deep neural network (DNN). Our DNN, which consists of several layers such as convolutional layers and a feature combiner, extracts stroke-adapted visual features and spatial features, and then adjusts the importance of them. We also develop a learning algorithm for our DNN that does not suffer from the vanishing gradient problem, and hence avoids falling into undesirable locally optimal solutions. We demonstrate that edit propagation with deep features, without manual feature tuning, can achieve better results than previous work.

Categories and Subject Descriptors (according to ACM CCS): I.4.0 [Image Processing And Computer Vision]: General—

1. Introduction

Edit propagation enables us to easily edit images based on simple user input. Various applications of edit propagation exist, such as grayscale image colorization, color image recoloring, segmentation and tone adjustment. Many efforts have been made to attack the edit propagation problem [LLW04, LWCO*07, LAA08, XLJ*09, XWT*09, LJH10, CZZT12, XYJ13, CZL*14, YY15]. Specifying sparse image edits, users can propagate the image edits to the entire image according to the propagation principle based on pixel similarity (e.g., proximities of positions, colors or textures).

We can regard edit propagation as a sort of multi-class classification problem. From image feature vector $\mathbf{x}_i$ at pixel i, a typical edit propagation pipeline estimates multi-class probability vector $\mathbf{y}_i \in [0,1]^n$ that represents how likely it is that a pixel i belongs to each of n types of strokes (class labels). A model for estimating $\mathbf{y}_i$ is constructed from user-specified strokes and $\mathbf{x}_i$ on the strokes. A number of techniques have been proposed for edit propagation using various models, such as optimization-based methods [LLW04, AP08], gentle boost classifier [LAA08], function interpolation with radial basis functions (RBFs) [LJH10], manifold learning [CZZT12], and a probabilistic model [XYJ13]. Most of these previous studies use a combination of multiple features with different meanings as input. For example, they use concatenated features $\mathbf{x} = [\sigma_c \mathbf{c}^T, \sigma_t \mathbf{t}^T, \sigma_s \mathbf{s}^T]^T$ where $\mathbf{c}$ denotes pixel color, $\mathbf{t}$ texture feature, and $\mathbf{s}$ spatial coordinate whereas σ_c, σ_t, and σ_s denote parameters that determine the importance of each feature.

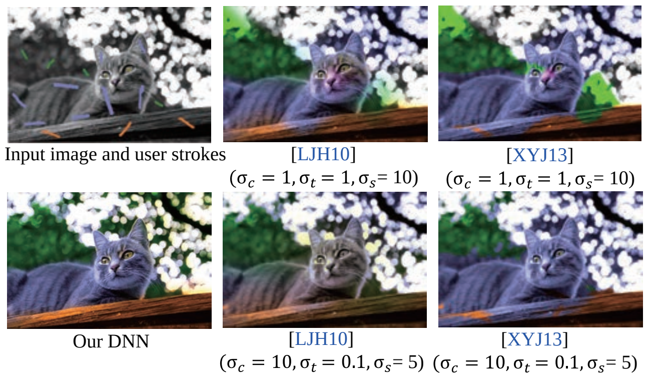
Input image and user strokes
[LJH10]
($\sigma_c = 1, \sigma_t = 1, \sigma_s = 10$)
[XYJ13]
($\sigma_c = 1, \sigma_t = 1, \sigma_s = 10$)
Our DNN
[LJH10]
($\sigma_c = 10, \sigma_t = 0.1, \sigma_s = 5$)
[XYJ13]
($\sigma_c = 10, \sigma_t = 0.1, \sigma_s = 5$)

Figure 1: *Image colorization using edit propagation. While existing methods require manual parameter tuning for each image feature, our DNN-based method automatically extracts stroke-adapted features.*

In most previous work, users must heuristically select the image features that they use and adjust parameters for the features in accordance with their needs and target images. As shown in Figure 1, the editing results can be drastically changed depending on the parameters. For example, edits on strokes with a given intensity prop-

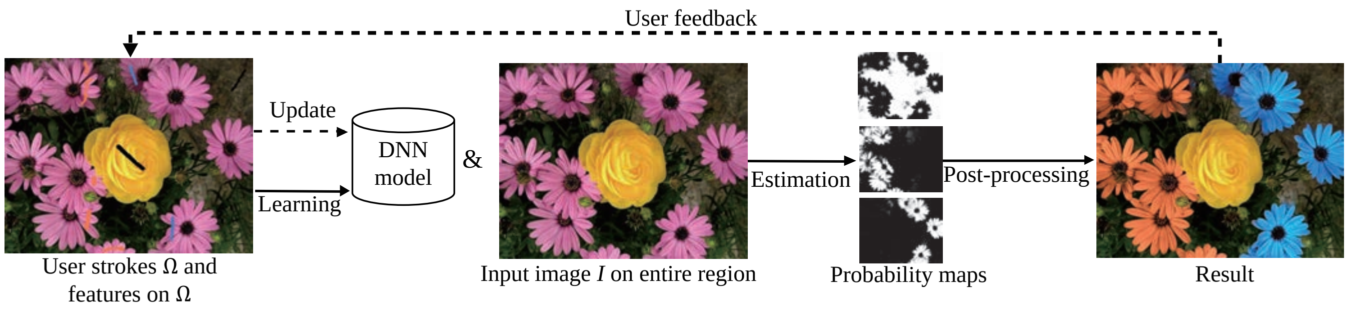

Figure 2: *System overview. The system first learns a DNN model from an input image and user strokes. Next, stroke probabilities on all pixels are estimated using the DNN model, and probability maps are obtained. Finally, the probability maps are refined by post-processing. Every time the user updates strokes, the system updates the DNN model efficiently using previously learned parameters.*

agate to pixels that have similar intensity when σ_c is large. Furthermore, using too many visual features might limit the range/diversity of input images on which the propagation succeeds. For example, the Gabor feature, which is typically used as a texture feature, has multiple parameters such as a kernel size and rotation angle. Using various patterns of the parameters as well as other image features causes an overfitting problem of the estimated model to training data on strokes due to the increased dimension of input vectors. Consequently, unneeded visual features degenerate estimated results [LAA08]. On the other hand, manually selecting appropriate visual features from many candidates is labor intensive.

Considering the above, we conducted a study to address the following research question:

Can we automatically extract effective features for edit propagation without selecting image features manually?

To this end, we adopt deep learning [Ben09], which is a technique for learning deep neural network (DNN) models. Recently, this technique dramatically improved the accuracy of image recognition and speech recognition [KSH12, DLH*13]. While conventional approaches manually engineer features for classification, DNNs can be used for representation learning, which automatically extracts effective high-level features for a task from low-level features (e.g., image pixels and audio spectrum).

In this paper, we propose *DeepProp* for extracting Deep features from a single image for edit Propagation. Our method uses low-level visual patches and spatial pixel coordinates as input of a DNN that automatically extracts features adapted to user-specified strokes from a single image. In contrast to most previous work, we do not need to adjust the importance of the input features. Then, we use the DNN as a classifier that estimates user stroke probabilities, which represent how likely it is that each pixel belongs to each stroke, from extracted features on the entire image. Figure 1 demonstrates that our method can, without tuning parameters for image features, generate a better result than previous work.

For edit propagation with deep features, this paper makes the following contributions:

- *Edit propagation system using deep learning*: We propose a system design of a framework that consists of the following three main modules (Figure 2): learning, estimation, and post-processing. First, a DNN model is learned using user strokes and pixels on the strokes as input. Then, using the learned DNN model, user stroke probabilities are estimated on the entire image (on pixels or, for more efficiency, superpixels), and probability maps are obtained. Finally, probability maps or application results (e.g., colorization) obtained from the probability maps are refined by post-processing. Every time the user updates strokes, the system re-learns the DNN model efficiently using previously learned parameters.
- *DNN architecture for edit propagation*: We propose a DNN architecture that extracts stroke-adapted visual features and spatial features using convolutional and fully-connected network structures. The importance of the extracted two features is also automatically determined using a feature combiner layer, and stroke probability vectors are computed from the combined features using a soft-max layer.
- *Learning algorithm for our DNN*: A number of local minima can be found during optimization of parameters of DNN models, especially when DNNs have complicated structures with deep layers and many neurons. Optimizing the entire network of our DNN tends to fall into such undesirable local minima due to the error function's vanishing gradients. This is because our DNN contains two types of the networks with different depths. We develop a learning algorithm that efficiently propagates the gradients to these networks and finds more desirable solutions, and can generate better results than a naïve learning algorithm.

2. Related Work

2.1. Edit propagation

The first studies related to edit propagation are colorizations using optimization [LLW04] and chrominance blending [YS06]. In addition to these approaches based on pixel colors, colorizations using texture features in manga [QWH06] and natural images [LWCO*07] have been also proposed. Furthermore, we can regard tone adjustment [LFUS06], material editing [PL07], intrinsic images [BPD09] and editing of bidirectional texture functions (BTFs) [XWT*09] as applications of edit propagation because they all propagate user edits based on similarities between on-stroke pixels and the rest of the image. Such stroke-based approaches are also used in matting [LRAL07, RRW*09], edge-preserving smoothing [XLXJ11, CLKL14], and temporally consistent propagation for video editing [CZL*14].

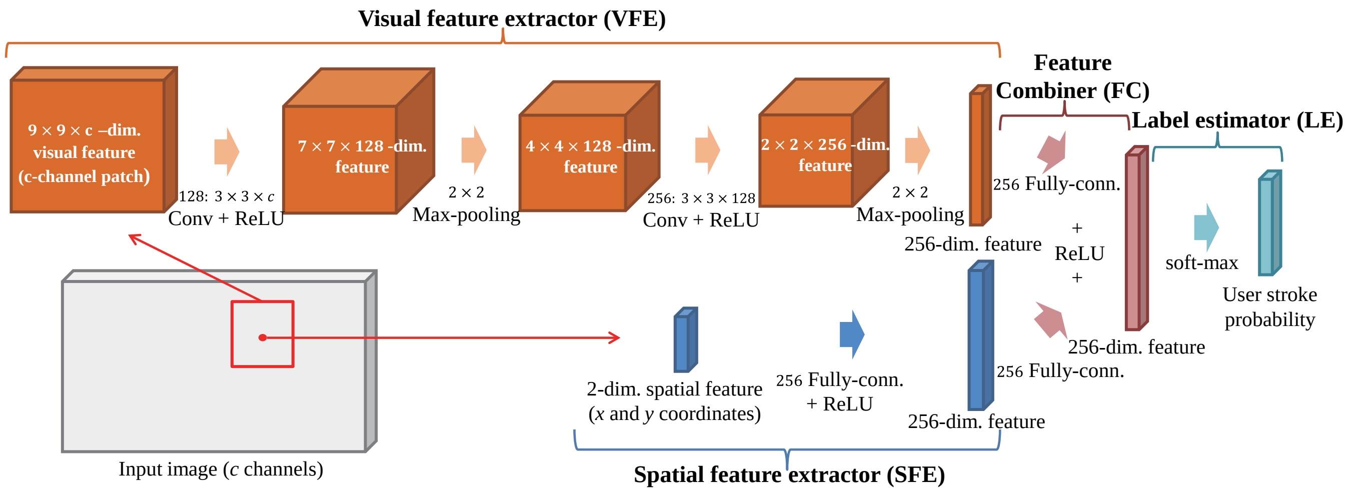

Figure 3: *Our deep neural network architecture.*

Other than the various applications, advances have been also made in the fundamental algorithms for edit propagation. Li et al. proposed an approach using a gentle boost classifier by formulating edit propagation as a pixel classification problem [LAA08]. *AppProp* [AP08] yields better propagation by optimizing color differences for optimizing color differences not only between nearby pixels, but also between non-neighboring ones. Computational efficiency has been also improved by using a kd-tree [XLJ*09], continuously approximating feature space using *radial basis functions* (RBFs) [LJH10], manifold learning [MCY*13], efficient stroke sampling [BHW11], and sparse pixel sampling [YY15]. Most of the previous approaches share a common issue, namely that halo artifacts occur across object boundaries [LWA*12]. Chen et al. [LWA*12] achieved graceful color blending along object boundaries based on *locally linear embedding* (LLE), which was further improved in terms of efficiency using dictionary learning [CZL*14]. Xu et al. proposed a stochastic modeling of the similarities between on-stroke and off-stroke pixels, based on iterative feature discrimination and sparse sampling of on-stroke pixels [XYJ13]. Also, there are approaches using specific distance metrics such as geodesic distance [CSRP10] and diffusion distance [FFL10]. Whereas these previous approaches use various image features, effective features vary depending on input images and user-specified strokes. Therefore, if multiple image features are considered simultaneously, we have to tune the importance that enhances or suppresses each feature for each image. Even worse, the more image features are used, the more over-fitting occurs.

2.2. Deep learning

Recently, *deep learning* has gained significant attention in computer science. Deep learning is a generic term for techniques using deep neural network (DNN), and has achieved outstanding results in various fields. An advantage of DNN is that it can be used for representation learning, where low-level features (e.g., pixel values) are directly used as input and automatically converted to higher-level features through intermediate layers, without conventional manual design of discriminative features. A commonly occurring problem is that of vanishing gradients of the energy functions, which hinders deep layers from sufficiently learning effective features, especially in deeper neural networks. Recent progress including the unsupervised pretraining with deep belief network (DBN) [BLP*07] and the use of rectified linear unit (ReLU) [GBB11] as well as performance improvement of computers in the last few years have enabled efficient learning using DNN.

With regards to the types of neural networks used for deep learning, fully-connected neural network, convolutional neural network (CNN) and recurrent neural network (RNN) are commonly chosen [Ben09]. Fully-connected neural networks such as multilayer perceptron (MLP) are typical examples in DNN and have been applied to classification and regression problems in several research fields (e.g., speech recognition [KSH12, DLH*13], activity recognition [LGQ15], and exemplar-based photo adjustment [YZW*15]). On the other hand, CNNs are often used if the inputs are images. Unlike general fully-connected networks, a CNN is a network consisting of convolutional layers with multiple filters and pooling layers, which enables CNN to achieve invariant features (e.g., shift invariance) under various deformations during feature extraction. CNNs have been applied to many image processing tasks, such as image classification [XXY*15], contour detection [SWW*15], motion deblurring [SCXP15], saliency detection [LY15], and depth estimation [LSL15]. We adopt a combination of a CNN and a fully-connected network for edit propagation for the first time, and demonstrate its effectiveness through several applications.

3. DNN Architecture

The basic idea for designing a DNN is to extract high-level features for edit propagation from low-level visual and spatial features. In edit propagation, we can assume that the lowest-level features are color (or grayscale) values and coordinates. For visual features, it is known that effective high-level features can be extracted using convolutional layers, as explained in Section 2. Instead of using single pixels, we use color (or grayscale) patches as low-level visual features for convolution. This is because a patch enables implicitly handling pixel gradients and capturing more various patterns than

Table 1: *Definitions of main symbols.*

Symbols	Descriptions
I	input image.
$\mathbf{x}_i$	image feature vector at pixel i.
$\mathbf{g}_i$	binary vector for user strokes at pixel i.
Ω	user stroke region.
X_Ω	set of input features $\mathbf{x}_i$ in stroke region Ω.
$\mathcal{G}_\Omega$	set of binary vectors $\mathbf{g}_i$ in stroke region Ω.
$\mathbf{y}_i$	probability vector at pixel i.
Z	probability map or colorized result.
U	result obtained by post-processing.
$\theta_v, \theta_s, \theta_c, \theta_l$	parameters of DNN model.
G_v, G_s, G_c, G_l	layer functions of DNN model.

a single pixel. Additionally, we use pixel coordinates as low-level spatial features. To extract the corresponding high-level spatial features, we apply a non-linear transformation, since a convolutional layer cannot be applied in this case. Both of the extracted high-level features are then combined and used for edit propagation. We empirically experimented with several structures of the DNN, varying in, e.g., the sizes of convolutional kernels and the dimensions of high-level features.

As illustrated in Figure 3, our DNN is a feedforward neural network that has four structures: visual feature extractor (VFE), spatial feature extractor (SFE), feature combiner (FC), and label estimator (LE). The main notations used in this paper are summarized in Table 1. The DNN estimates probability vector $\mathbf{y}_i$ of user strokes from feature vector $\mathbf{x}_i = [\mathbf{p}_i^T, \mathbf{s}_i^T]^T$ at pixel i in input image I. Here, $\mathbf{p}_i \in \mathbb{R}^{9\times9\times c}$ is a 9×9 patch feature centered at pixel i with c color channels (e.g., three-channel RGB or one-channel grayscale), and $\mathbf{s}_i \in \mathbb{R}^2$ is a pixel coordinate. Each element of input feature vectors is normalized from 0 to 1. In contrast to most previous work, determining importance of each image feature is not required. In the following sections, we describe the role of each structure of our DNN.

3.1. Visual feature extractor (VFE)

The VFE extracts a high-level visual feature, such as complicated texture patterns or simple colors adapted to user strokes, from a low-level visual patch feature $\mathbf{p}$. Specifically, the VFE computes high-level visual features $\mathbf{f}_v \in \mathbb{R}^{256}$ from $\mathbf{p}$ using function G_v with model parameter θ_v. G_v consists of two convolutional functions f_{conv1} and f_{conv2}, a max-pooling function f_{mp}, and an activation function of the rectifier linear unit (ReLU) :

$$\mathbf{f}_v = G_v(\mathbf{p};\theta_v), \tag{1}$$

$$G_v(\mathbf{p};\theta_v) = f_{mp}(f_{ReLU}(f_{conv2}(f_{mp}(f_{ReLU}(f_{conv1}(\mathbf{p};\theta_{conv1})));\theta_{conv2}))) \tag{2}$$

where f_{conv1} applies 128 types of $3\times3\times c$ filter kernels k with biases b to input patch feature $\mathbf{p}$. That is, f_{conv1} obtains 128 filtered maps $(k*\mathbf{p}+b)$ by convolving input patch features $\mathbf{p}$ with different filter kernels k and biases b. Then, the ReLU activation function $f_{ReLU}(\mathbf{v}) = [\max(0,v_1), \max(0,v_2), ..., \max(0,v_m)]^T$ is applied to the convolved patch (where the subscripts of v denote element indices of the vector). Next, we use a 2×2 max-pooling function f_{mp} with stride 2, which yields position invariance over the patches. We additionally apply convolution f_{conv2} with a $3\times3\times128$ kernel and the activation function f_{ReLU}. Finally, a 2×2 max-pooling with stride 2 is used again and a 256-dimensional feature vector $\mathbf{f}_v$ is obtained. θ_v denotes filter parameter (i.e., filter kernels k and bias terms b) and is learned using user strokes as training data, as explained in Section 4.

3.2. Spatial feature extractor (SFE)

The SFE extracts an abstracted feature $\mathbf{f}_s \in \mathbb{R}^{256}$ from a spatial pixel coordinate $\mathbf{s}$:

$$\mathbf{f}_s = G_s(\mathbf{s};\theta_s), \tag{3}$$

$$G_s(\mathbf{s};\theta_s) = f_{ReLU}(\mathbf{W}_s\mathbf{s}+\mathbf{b}_s), \tag{4}$$

where $\theta_s = \{\mathbf{W}_s, \mathbf{b}_s\}$ is a model parameter of the SFE, and $\mathbf{W}_s \in \mathbb{R}^{256\times2}$ and $\mathbf{b}_s \in \mathbb{R}^{256}$ are a weight matrix and bias term. The dimension of the output feature vectors is set to 256 so that they have the same dimension as the visual features. In the same way, the feature combiner (explained in the next section) can handle both the visual features and spatial features fairly.

3.3. Feature combiner (FC)

The function G_c of the FC converts $\mathbf{f}_v$ and $\mathbf{f}_s$ extracted by the VFE and the SFE into a single feature vector $\mathbf{f}_c \in \mathbb{R}^{256}$:

$$\mathbf{f}_c = G_c(\mathbf{f}_v,\mathbf{f}_s;\theta_c), \tag{5}$$

$$G_c(\mathbf{f}_v,\mathbf{f}_s;\theta_c) = f_{ReLU}(G_{cv}(\mathbf{f}_v;\theta_{cv}) + G_{cs}(\mathbf{f}_s;\theta_{cs})), \tag{6}$$

$$G_{cv}(\mathbf{f}_v;\theta_{cv}) = \mathbf{W}_{cv}\mathbf{f}_v + \mathbf{b}_{cv}, \tag{7}$$

$$G_{cs}(\mathbf{f}_s;\theta_{cs}) = \mathbf{W}_{cs}\mathbf{f}_s + \mathbf{b}_{cs}, \tag{8}$$

where $\theta_c = \{\theta_{cv}, \theta_{cs}\}$ is a model parameter of the FC (where $\theta_{cv} = \{\mathbf{W}_{cv} \in \mathbb{R}^{256\times256}, \mathbf{b}_{cv} \in \mathbb{R}^{256}\}$ and $\theta_{cs} = \{\mathbf{W}_{cs} \in \mathbb{R}^{256\times256}, \mathbf{b}_{cs} \in \mathbb{R}^{256}\}$). Network structures similar to the FC have also been introduced for combining multimodal features, such as image and audio [NKK*11, WJW*14]. Such a network structure can determine the importance of two features by capturing correlations across the two modalities. In this paper, we utilize this structure for combining the visual features and spatial features.

3.4. Label estimator (LE)

The function G_l of the LE estimates user stroke probability vectors $\mathbf{y}$ from feature vectors $\mathbf{f}_c$ extracted by the FC:

$$\mathbf{y} = G_l(\mathbf{f}_c;\theta_l), \tag{9}$$

$$G_l(\mathbf{f}_c;\theta_l) = f_{softmax}(\mathbf{W}_l\mathbf{f}_c + \mathbf{b}_l), \tag{10}$$

where $f_{softmax}(\mathbf{v}) = [\frac{\exp(v_1)}{\sum_i^n \exp(v_i)}, \frac{\exp(v_2)}{\sum_i^n \exp(v_i)}, ..., \frac{\exp(v_n)}{\sum_i^n \exp(v_i)}]^T$ is the softmax function and $\theta_l = \{\mathbf{W}_l \in \mathbb{R}^{n\times256}, \mathbf{b}_l \in \mathbb{R}^n\}$ is a model parameter of the LE. The LE determines, for each off-stroke pixel, a fractional weight for each stroke, based on the off-stroke pixel's similarity to pixels on the stroke. This is done using the soft-max layer, which acts as a standard regression function for probabilistic multi-class classification.

4. Learning DNN from User Strokes

This section explains how to learn the model parameters $(\theta_v, \theta_s, \theta_c, \theta_l)$ of our DNN using training data: user strokes $\mathcal{G}_\Omega$ and input features $\mathbf{x} \in X_\Omega$ on stroke region Ω.

A straightforward way for learning the model parameters is to minimize the error function E between probability vectors estimated by the DNN and binary vectors $\mathbf{g}_i \in \mathcal{G}_\Omega$ obtained from user strokes:

$$(\hat{\theta}_v, \hat{\theta}_s, \hat{\theta}_c, \hat{\theta}_l) = \underset{\theta_v,\theta_s,\theta_c,\theta_l}{\operatorname{argmin}}\ E(\theta_v, \theta_s, \theta_c, \theta_l), \tag{11}$$

$$E(\theta_v, \theta_s, \theta_c, \theta_l) = \sum_{i \in \Omega} L(G_l(G_c(G_v(\mathbf{p}_i; \theta_v), G_s(\mathbf{s}_i; \theta_s); \theta_c); \theta_l), \mathbf{g}_i), \tag{12}$$

where L indicates the cross-entropy loss function defined as $L(\mathbf{y}, \mathbf{g}) = -\sum_{k=1}^{n}\{g_k \ln y_k + (1-g_k)\ln(1-y_k)\}$.

Unfortunately, this naïve approach does not work in practice. Because of the structural complexity of our DNN, we need to consider the fact that many local minima can be found in an optimization process such as the one described above. In general, model parameters of feedforward neural networks are learned using the backpropagation of error with a gradient-based optimization algorithm, such as the stochastic gradient descent (SGD) [KB14]. However, in our case, when the model parameters of the entire network are optimized simultaneously using the backpropagation, the algorithm falls into local minima before the gradient error propagates sufficiently into the deep layers of the VFE, due to the vanishing gradient problem. That is, the learning speed of the SFE is much faster than that of the VFE because the SFE is shallow whereas the VFE is much deeper.

To address this issue, we develop a learning algorithm for optimizing the model parameters efficiently in two steps. Figure 4 illustrates this learning strategy. In the first step, we omit the SFE from the DNN and optimize the deep network consisting of the VFE, FC, and LE. Specifically, we pre-train visual features by minimizing a new error function E_v:

$$(\hat{\theta}_v, \hat{\theta}_{cv}, \hat{\theta}_l) = \underset{\theta_v,\theta_{cv},\theta_l}{\operatorname{argmin}} E_v(\theta_v, \theta_{cv}, \theta_l), \tag{13}$$

$$E_v(\theta_v, \theta_{cv}, \theta_l) = \sum_{i \in \Omega} L_v(G_l(G_{cv}(G_v(\mathbf{p}_i; \theta_v); \theta_{cv}); \theta_l), \mathbf{g}_i), \tag{14}$$

where L_v is the cross-entropy loss of the above network. We optimize the parameters based on gradients $\frac{\partial L_v}{\partial \theta_v}$, $\frac{\partial L_v}{\partial \theta_{cv}}$, $\frac{\partial L_v}{\partial \theta_l}$ using the backpropagation. We initialize the model parameters with random numbers following a normal distribution with zero mean and standard deviation equal to the inverse of the dimension of each model parameter. For optimization, we use the mini-batch Adam algorithm [KB14] because of its fast convergence. To determine convergence, we check if the error in all training data is smaller than $\varepsilon|\Omega|$ or the difference between the current error and previous error is larger than $\gamma|\Omega|$ (where ε and γ are coefficients, and $|\Omega|$ is the number of pixels in region Ω). Note that the latter condition is used to abort optimization if it does not converge completely. In the second step, we optimize the entire network with the SFE by initializing the DNN with the model parameters $\hat{\theta}_v$, $\hat{\theta}_{cv}$, and $\hat{\theta}_l$ learned in the first step. The model parameters θ_s and θ_{cs} not learned in the first step are randomly initialized and optimized using the mini-batch Adam algorithm in the same way as above. For optimization, mini-batch size is empirically set to 10, and both ε and γ are set to 0.01.

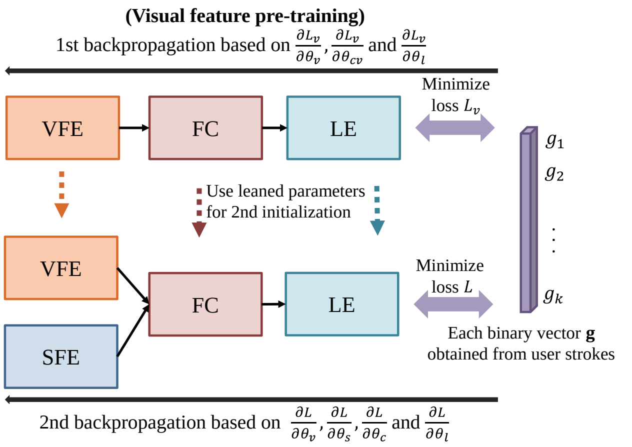
(Visual feature pre-training)
1st backpropagation based on $\frac{\partial L_v}{\partial \theta_v}, \frac{\partial L_v}{\partial \theta_{cv}}$ and $\frac{\partial L_v}{\partial \theta_l}$
VFE
FC
LE
Minimize loss L_v
Use leaned parameters for 2nd initialization
VFE
SFE
FC
LE
Minimize loss L
g_1
g_2
g_k
Each binary vector **g** obtained from user strokes
2nd backpropagation based on $\frac{\partial L}{\partial \theta_v}, \frac{\partial L}{\partial \theta_s}, \frac{\partial L}{\partial \theta_c}$ and $\frac{\partial L}{\partial \theta_l}$

Figure 4: *Our strategy of DNN learning. We first pre-train visual features on the network consisting of the VFE, FC, and LE using backpropagation, and then learn the entire network together with the SFE.*

As can be seen in Equations (12) and (14), the computational complexity of the optimization w.r.t. the number of training data $|\Omega|$ is $O(|\Omega|)$. To accelerate the learning, we subsample pixels Ω' by 10% of Ω for optimization. Even with subsampling the learned result is similar to the result with all data due to inherent redundancy, as demonstrated in Section 6.1. Algorithm 1 summarizes our DNN learning algorithm.

Algorithm 1 Learning DNN from user strokes

```
    Inputs: stroke region $\Omega$, image feature vectors $\mathbf{x}_i = \{\mathbf{p}_i, \mathbf{s}_i\} \in X_\Omega$, and stroke binary vectors $\mathbf{g}_i \in \mathcal{G}_\Omega$
    Outputs: $\theta_v, \theta_s, \theta_c, \theta_l$
 1: $\Omega' \leftarrow$ RANDOMSAMPLING($\Omega$);
 2: $\theta_v, \theta_s, \theta_c, \theta_l \leftarrow$ NORMALDISTRIBUTION();
 3: $preE_v \leftarrow 0$;
 4: while true do
 5:     $\theta_v, \theta_{cv}, \theta_l \leftarrow$ MINIBATCHADAM($X_{\Omega'}, \mathcal{G}_{\Omega'}, \frac{\partial L_v}{\partial \theta_v}, \frac{\partial L_v}{\partial \theta_{cv}}, \frac{\partial L_v}{\partial \theta_l}$);
        // Eq. (14)
 6:     if $E_v < \varepsilon|\Omega'|$ or $E_v - preE_v > \gamma|\Omega'|$ then
 7:         break;
 8:     end if
        $preE_v \leftarrow E_v$;
 9: end while
10: $preE \leftarrow 0$;
11: while true do
12:     $\theta_v, \theta_s, \theta_c, \theta_l \leftarrow$ MINIBATCHADAM($X_{\Omega'}, \mathcal{G}_{\Omega'}, \frac{\partial L}{\partial \theta_v}, \frac{\partial L}{\partial \theta_s}, \frac{\partial L}{\partial \theta_c}, \frac{\partial L}{\partial \theta_l}$);
        // Eq. (12)
13:     if $E < \varepsilon|\Omega'|$ or $E - preE > \gamma|\Omega'|$ then
14:         break;
15:     end if
        $preE \leftarrow E$;
16: end while
```

4.1. Efficient DNN update per user edit

Every time the user adds or removes a stroke in order to obtain a desired result, the system has to re-learn the model parameters from the update strokes. Naïvely re-learning from scratch is costly, and thus accelerating the re-learning is crucial for interactive editing.

To update the model parameters efficiently, we reuse the model parameters learned with the previous user strokes as the initialization of the learning algorithm (line 4 in Algorithm 1). To achieve this, we need to adapt the LE structure accordingly. As shown in Figure 5, if the total number of different types of user strokes (e.g., strokes that edit color for colorization) is reduced, corresponding rows of the model parameters (i.e., the weight matrix $\mathbf{W}_l$ and bias term $\mathbf{b}_l$) are deleted. If a new type of user stroke is introduced, we add new rows and initialize them with random numbers following the normal distribution.

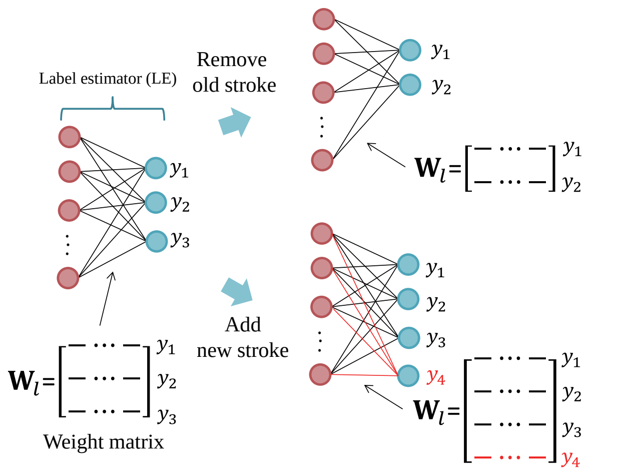

Figure 5: *Efficient model parameter update. If user strokes are added or removed, model parameters of the LE are updated efficiently using previous parameters.*

5. Edit Propagation Using DNN

5.1. Estimating probability maps

We now propagate information from the user strokes to all pixels in the input image using the learned DNN model. Given a patch feature and coordinate feature of each pixel, our feedforward neural network outputs a stroke probability vector using the learned parameters. However, processing all pixels one-by-one takes a large amount of time since the feedforward computation for one data point includes many applications of convolution filtering.

We adopt superpixel-wise propagation to reduce the computational time. For each superpixel, a center pixel is calculated, and then a user stroke probability is computed using the DNN model from features calculated only at the center pixel. Finally, the same stroke probability is assigned to all the pixels in the corresponding superpixel. For generating superpixels, we use the simple linear iterative clustering (SLIC) [ASS*12] because this algorithm can generate regularly-shaped superpixels that can reduce the gap between shapes of superpixels and patches.

5.2. Post-processing

We refine the estimated result by post-processing, which has mainly two roles, i.e., smoothing and interpolation. First, we can improve the result by smoothing it across superpixels. Second, interpolation can alleviate noise and halo artifacts along object boundaries [CZZT12, XYJ13]. Similarly to [XYJ13], we obtain a final editing result (e.g., colorized or segmented images) U from the result Z estimated with the DNN model, by solving the following optimization problem:

$$\min_{\mathbf{u}_i \in U} \sum_i \beta_i ||\mathbf{u}_i - \mathbf{z}_i||^2 + \lambda \sum_i \sum_{j \in N(i)} \phi_{ij} ||\mathbf{u}_i - \mathbf{u}_j||^2, \tag{15}$$

$$\beta_i = \ominus\{e_i = 0\}, \tag{16}$$

where $\mathbf{z}_i \in Z$ denotes the quantity being post-processed, e.g., the stroke probability vector $\mathbf{y}_i$ (in case the probability map itself is the target of the post-processing) or the result computed from $\mathbf{y}_i$ in a specific application (e.g., color in a colorization application). e_i is binary edge at pixel i in input image I, and $\ominus$ is the morphological erosion operator. λ determines the degree of smoothing, and $N(i)$ is a set of neighbor pixels of pixel i. We can use several types of weighting coefficients ϕ_{ij} that represent relations between neighbor pixels, such as the affinity function [LLW04] based on similarity of pixel values based on similarity of pixel values (when considering pixel gradient information) and matting Laplacian [LRAL07] for matting. In our experiment, λ is set to 5 for all images. To solve this linear system, we use the Gauss-Seidel method because of its simplicity of implementation. Finally, Algorithm 2 summarizes our edit propagation using the DNN model.

Algorithm 2 Edit propagation using DNN

```
    Inputs: input image $I$, image feature vectors $\mathbf{x}_i = \{\mathbf{p}_i, \mathbf{s}_i\}$,
    learned DNN parameters $\theta_v, \theta_s, \theta_c$, and $\theta_l$
    Output: final editing result $U$
1:  $\Omega_s \leftarrow$ EXTRACTSUPERPIXEL($I$);
2:  for each superpixel $S \in \Omega_s$ do
3:      $k \leftarrow$ EXTRACTCENTERPIXEL($S$);
4:      $\mathbf{y}_k \leftarrow G_l(G_c(G_v(\mathbf{p}_k;\theta_v), G_s(\mathbf{s}_k;\theta_s);\theta_c);\theta_l)$;
5:      $Z \leftarrow$ ASSIGNRESULTTOSUPERPIXEL($S, \mathbf{y}_k$);
6:  end for
7:  $U \leftarrow$ POSTPROCESSUSINGOPTIMIZATION($I, Z$);
```

6. Experiments

We implemented our prototype system with C++ and the OpenCV2.4.9 library except for deep learning. For implementing deep learning, we used Python and the chainer library. The system was run on a PC equipped with a 2.80 GHz CPU and 8 GB of memory. The image sizes we used in this paper, the number of user strokes (total pixels), and computational times of our method are summarized in Table 2. As can be seen in the table, the results of Figure 1 and the lower center of Figure 10 took longer than most of the others. This is because the convolutional layers require substantial learning to extract effective visual features such as textures from little available information (i.e., 1-channel grayscale values). We also tested with the high-resolution image in Figure 6 in our experiments. Although it took the longest, the time needed for that image is relatively short given that the number of pixels is tens of times larger than the other images. This is because only pixels on

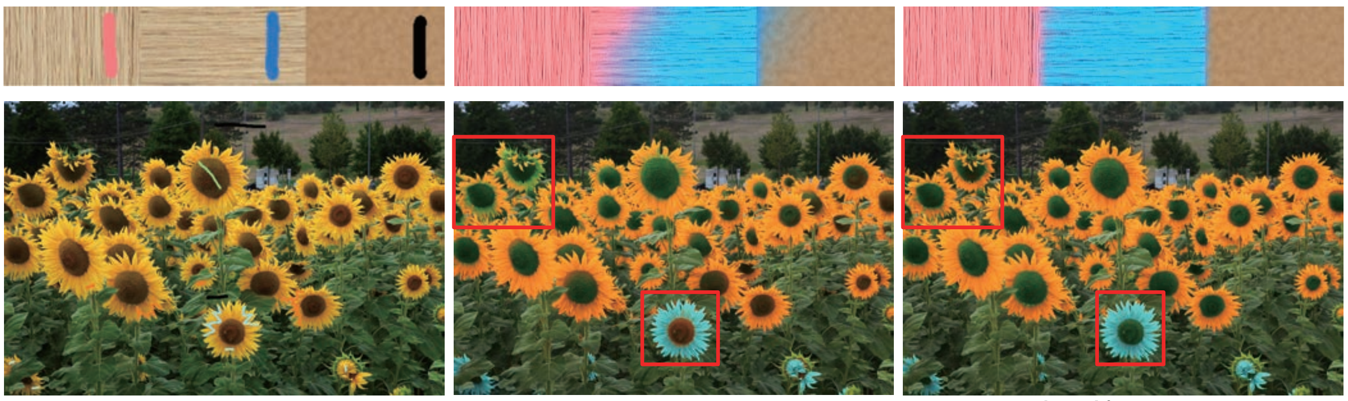
Input image and user strokes
Without visual feature pre-training
With visual feature pre-training

Figure 6: *Comparisons of recolorization results without and with visual feature pre-training.*

Table 2: *Total amount of inputs and computational times (seconds) of our method on CPU.*

Images	Image sizes	# of user strokes (# of total pixels)	Times
Fig.1	620 × 413	20 (7701)	97.4
Fig.2	400 × 314	10 (1818)	23.0
Fig.6 top	392 × 70	3 (1882)	18.3
Fig.6 bottom	1800 × 1200	19 (17964)	432.4
Fig.10 top	480 × 360	4 (2457)	22.4
Fig.10 upper center	640 × 480	5 (3208)	36.5
Fig.10 lower center	500 × 375	13 (20342)	271.6
Fig.10 bottom	640 × 427	9 (3285)	40.0
Fig.12 top	400 × 285	4 (5353)	36.1
Fig.12 center	400 × 267	8 (9666)	49.9
Fig.12 bottom	400 × 280	3 (6940)	39.3

strokes are used for learning and superpixels are used for estimation.

6.1. Evaluation of proposed method

Before comparing our method to previous work, we evaluate our method itself.

Learning algorithm. We first evaluate effectiveness of the proposed learning algorithm for our DNN. Figure 6 shows comparisons of image recoloring results with and without the visual feature pre-training. Without this, the visual features (texture patterns or color) were not sufficiently learned from the image, and the image edits propagated too strongly, only relying on the spatial features. This means that the optimization process fell into the local minimum before the gradient of the error function sufficiently propagated to the network of the VFE. Specifically, in the upper-center image in the figure, the specified colors (pink and blue) are leaking across the different textures because texture features are not extracted sufficiently. Also, in the red box of the lower-center image, the color specified with the green stroke did not propagated according to the texture patterns of the center of the sunflower, due to the same reason. By contrast, we can see that using our learning algorithm with visual feature pre-training alleviates these problems in the right images.

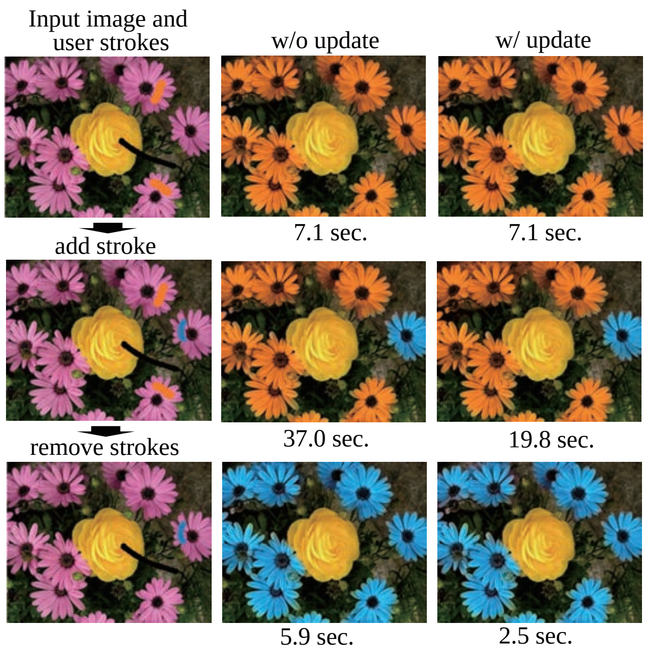

Figure 7: *Comparison of colorization results without (w/o) and with (w/) updating parameters. The caption under each result shows computational time of learning.*

Updating algorithm. We evaluate the effectiveness of the algorithm for updating the model parameters in Figure 7, where we compare the editing results and computational times of learning without and with update. When we do not use the updating algorithm, model parameters are learned from scratch every time strokes are added or removed. As can be seen in the figure, there is almost no difference between the two results, whereas the computational times with update become about half of the others.

Training data size. Figure 8 shows the editing results and computational times of learning with different amounts of training data. As shown in the figure, the results almost converged when we used equal to or more than 10% of the training data on the strokes, and computational times were linearly reduced according to the amount

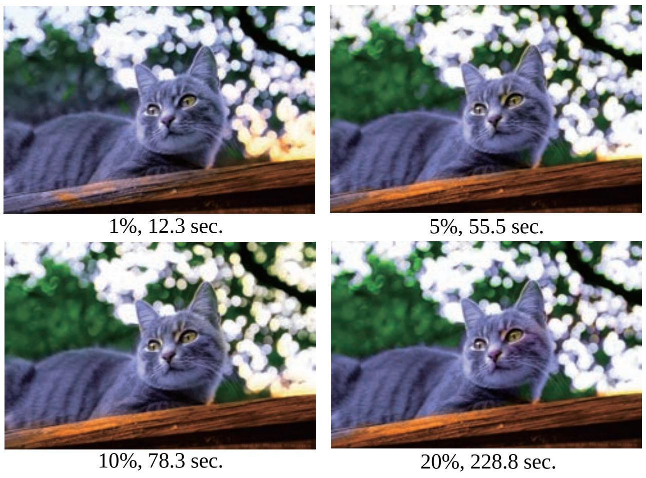

Figure 8: *Results with training data of different ratio. The input image and user strokes are the same as Figure 1. The caption under each image shows the percentage of samples of training data (left) and computational time of learning (right).*

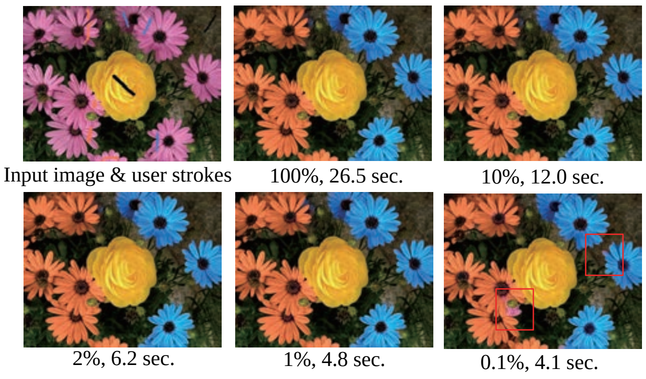

Figure 9: *Results with different number of superpixels. The caption under each image shows the percentage of the number of superpixels to that of the original image pixels (left) and computational time of estimation and post-processing (right).*

of the training data. Although it is generally known that deep learning requires a large amount of training data, in fact, a few hundreds of instances per one category are sufficient for learning models in some image recognition tasks [WYHY15]. Given this fact and the experimental results, we can confirm that our method works well with relatively little training data, and only a few types of user strokes.

Superpixel-based estimation. Figure 9 shows the editing results and computational times of estimating with different numbers of superpixels. In this example, the results are visually indistinguishable even when the number of superpixels decreases to 1% of the total image pixels. However, the computational times are significantly reduced according to the number of superpixels. Only when the number of superpixels is 0.1% of the total image pixels, the result starts deteriorating. This is because the superpixels become too large to fit the object shape. On the other hand, computational times were almost unchanged when less than 1% were used: of the total reported time, 4 seconds are due to the post-processing, which can be accelerated by using more efficient solvers.

6.2. Comparison with previous methods

To verify the effectiveness of our feature extraction method, we compare our method with the handcrafted feature-based methods.

6.2.1. Compared features

We prepared the following image features used in previous work.

- **color feature**: three (or one)-dimensional RGB (or grayscale) vectors $\mathbf{c}$.
- **spatial feature**: two-dimensional pixel coordinate vectors $\mathbf{s}$.
- **patch feature**: 243 (or 81)-dimensional vectors $\mathbf{p}$ of 9×9 RGB (or grayscale) patches.
- **texture feature**: 40-dimensional vectors $\mathbf{t}$ obtained by applying the Gabor filter [Dau85, CZL*14].
- **dense SIFT feature**: 128-dimensional vectors $\mathbf{d}$ that are scale-invariant feature transform (SIFT) features [Low99] on each pixel.

For the texture features, we selected the parameters of the Gabor filter, i.e., kernel size, wavelength of the sinusoidal factor, spatial aspect ratio, phase offset, standard deviation of the gaussian envelope, and kernel orientation as 30×30, π, 1, $\pi/2$, $\{1,2,3,4,5\}$, and $\{0,\pi/8,2\pi/8,3\pi/8,4\pi/8,5\pi/8,6\pi/8,7\pi/8\}$, respectively, and consequently 40-dimensional vectors were obtained. We concatenated these features and used $\mathbf{x} = [\sigma_c\mathbf{c}^T, \sigma_s\mathbf{s}^T, \sigma_p\mathbf{p}^T, \sigma_t\mathbf{t}^T, \sigma_d\mathbf{d}^T]^T$ in the comparisons with previous methods. The experiments were conducted by adjusting the importance of each feature $\sigma_c, \sigma_s, \sigma_p, \sigma_t$, and σ_d. As mentioned above, the proposed method uses $\mathbf{x} = [\mathbf{p}^T, \mathbf{s}^T]^T$ as input feature and we do not manually adjust the importance of them.

6.2.2. Results

The state-of-the-art for edit propagation is the sparse control model (SCM) [XYJ13] except for methods that focus on acceleration and memory efficiency. Given that comparisons with other edit propagation methods have been conducted in the SCM paper [XYJ13], and the fact that this paper focused mainly on feature extraction, we only compare our method to SCM and instant propagation (IP) [LJH10], which is closely related to SCM, using our own implementations.

Colorization and recoloring. Figure 10 shows comparisons of image recoloring and colorization between our method (DNN) and the previous methods with different parameters. The color of each pixel is computed as the probability-weighted average of the ab channels of each stroke in Lab color space using the probability maps obtained by our DNN. The colorized results are then post-processed. The parameters are shown at the bottom of the figure and correspond to results in each column. First, the result of our DNN obviously outperformed the leftmost results, which were obtained via the previous methods but using the same features as ours, i.e., patch and spatial features. Second, the use of all features in the previous methods tends to deteriorate the estimated results due to

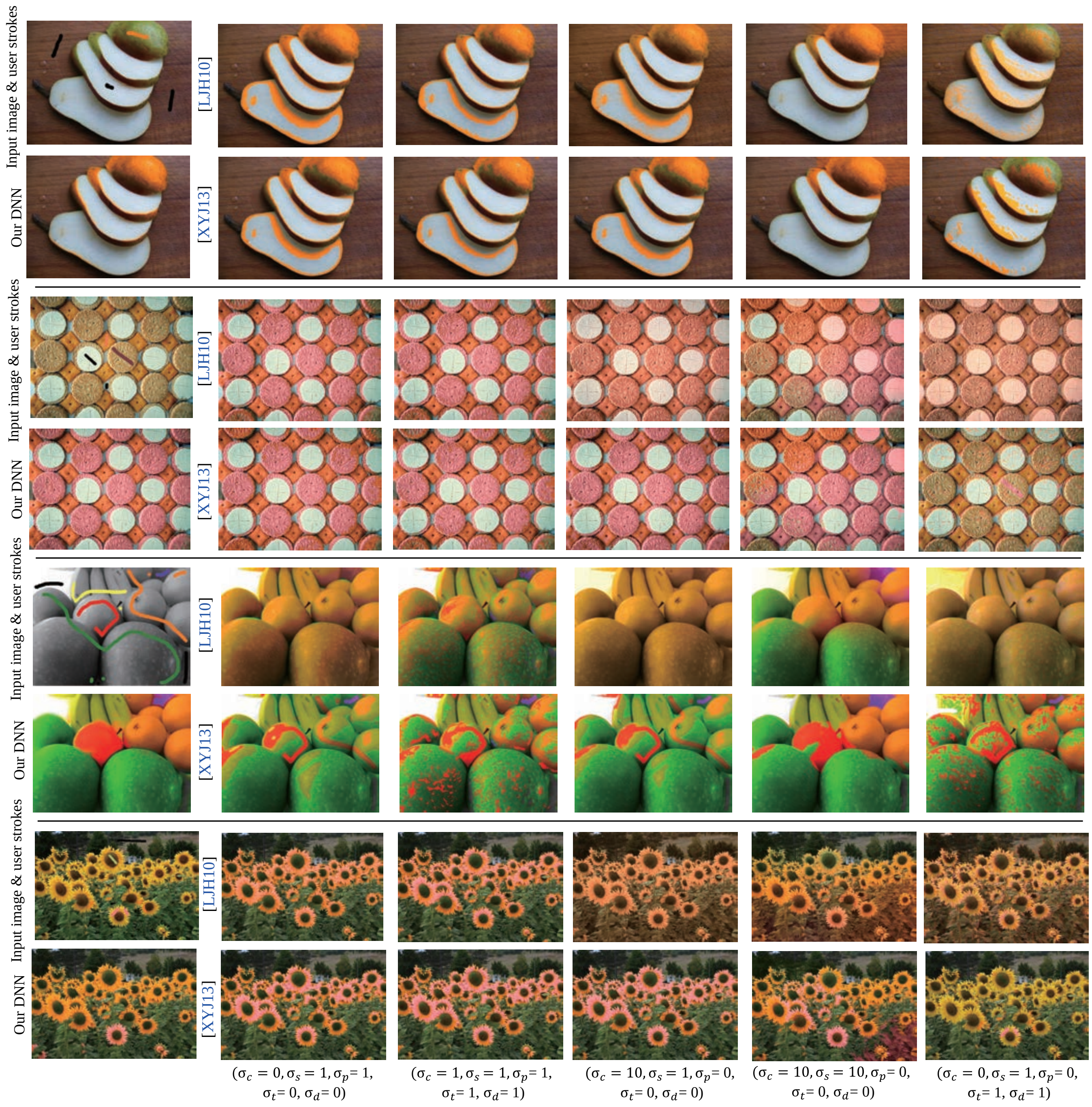
Input image & user strokes
Our DNN
[LJH10]
[XYJ13]
Input image & user strokes
Our DNN
[LJH10]
[XYJ13]
Input image & user strokes
Our DNN
[LJH10]
[XYJ13]
Input image & user strokes
Our DNN
[LJH10]
[XYJ13]
$(\sigma_c = 0, \sigma_s = 1, \sigma_p = 1, \sigma_t = 0, \sigma_d = 0)$
$(\sigma_c = 1, \sigma_s = 1, \sigma_p = 1, \sigma_t = 1, \sigma_d = 1)$
$(\sigma_c = 10, \sigma_s = 1, \sigma_p = 0, \sigma_t = 0, \sigma_d = 0)$
$(\sigma_c = 10, \sigma_s = 10, \sigma_p = 0, \sigma_t = 0, \sigma_d = 0)$
$(\sigma_c = 0, \sigma_s = 1, \sigma_p = 0, \sigma_t = 1, \sigma_d = 1)$

Figure 10: *Comparisons of color image recoloring and grayscale image colorization. For the existing methods, the feature parameters used in each column are shown in the bottom.*

the over-fitting. Third, it is difficult to handle some images even if individual features are selectively used. In contrast to these results, our method can generate better results than the previous methods overall. This means that, from low-level patch and coordinate features, our method extracted stroke-adapted high-level features that are effective for edit propagation, without manual feature selection.

Foreground segmentation. We compared results of foreground segmentation using 50 images randomly selected from MSRA 1k dataset [AHES09]. We quantitatively evaluate the methods via a precision-recall (PR) curve, which is often used for saliency detection [WLRY15]. While precision indicates the ratio of ground-truth pixels among pixels estimated as foreground, recall indicates the ratio of pixels estimated as foreground among all the ground-truth pixels. To extract foreground regions, we segmented probability maps using a threshold. PR curves were plotted with precision

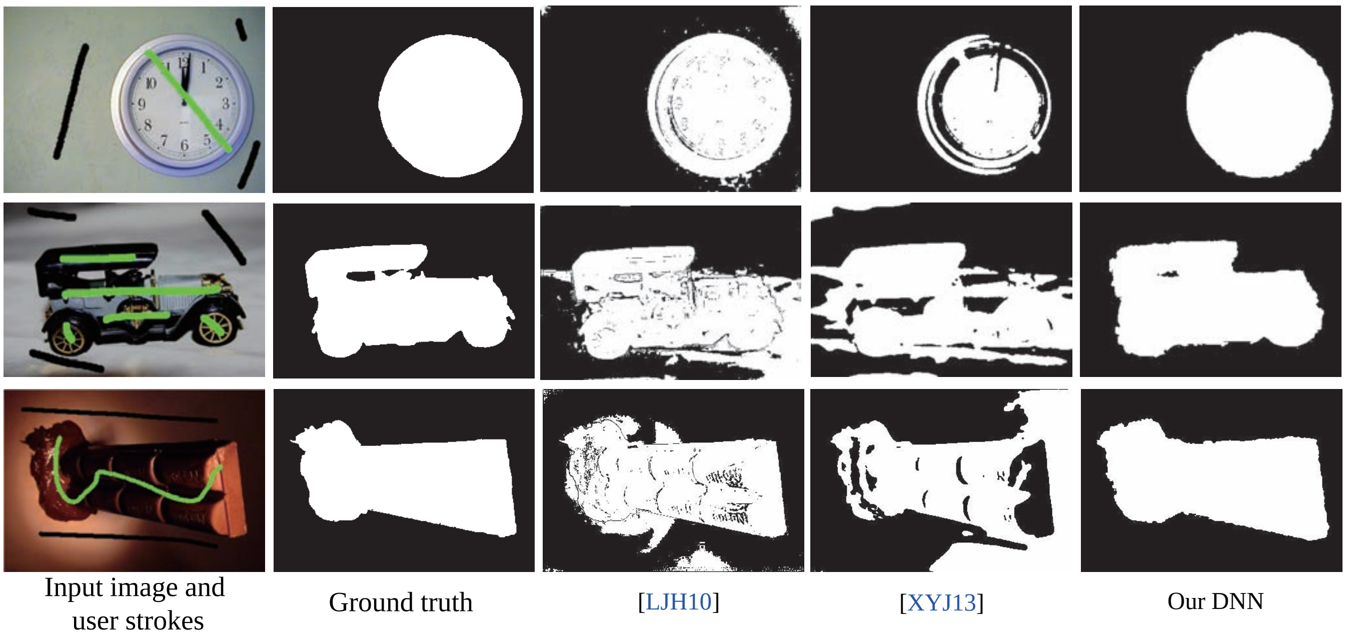

Figure 12: *Comparisons of several results of foreground segmentation selected from MSRA 1k dataset [AHES09]. Each segmented result is visualized by binarizing a probability map using a threshold (50%).*

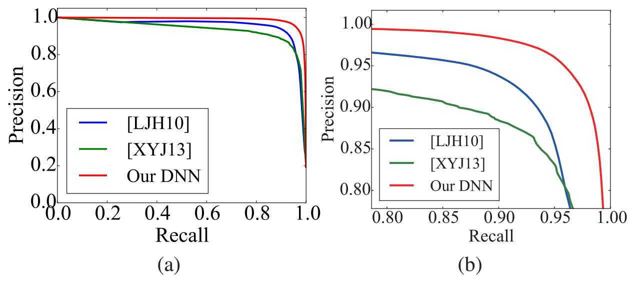

Figure 11: *(a) Micro average PR curve that shows comparisons of foreground segmentation using 50 images randomly selected from MSRA 1k dataset [AHES09]. (b) Magnified PR curve.*

and recall obtained using different thresholds. We specified foreground and background manually using strokes on each image and used the same strokes for all of the methods with the fixed feature parameters that were used in several of the results of [LJH10], i.e., $\sigma_c = 1/0.2 = 5$ and $\sigma_s = 1$. As summarized in the PR curve in Figure 11 and Table 3, our method overall outperforms the existing methods. Figure 12 also shows some results that are difficult for the existing methods to segment with only color information, that is, when the foreground and background colors are very similar. Although we tried to increase the importance of the texture features in the existing methods, over-fitting problems occurred and results degenerated in some cases. While the results of the existing methods might be improved if the feature parameters are tuned intensively, our method can generate relatively good results without this process.

As shown in Figures 1 and 10, the two existing methods propagate image edits based on different points of view. That is, IP estimates stroke probabilities smoothly, whereas SCM estimates them discriminatively. As explained in [XYJ13], SCM is based on iterative feature discrimination and associates each pixel with only a part of the control samples. For example, SCM can clearly propagate different image edits to neighbor objects that have similar color by discriminating the spatial features, even if the importance of these features is somewhat small. However, it is difficult to take advantage of this method unless appropriate features are selected and the importance of them is appropriately determined. Our focus is to address this problem, which was demonstrated in this section. It is an interesting avenue for future work to integrate deep features into the existing edit propagation methods.

Table 3: *Macro average precision, recall, and F1 ± standard deviation for binarization of probability maps using a threshold (50%) on the 50 images. Results marked with '*' show statistically significant differences between our method and the others as measured by paired t-test.*

Method	Precision	Recall	F1
[LJH10]	0.905 ± 0.113	0.900 ± 0.168	0.885 ± 0.145
[XYJ13]	0.900±0.131	0.892 ± 0.115	0.887 ± 0.099
Our DNN	0.945 ± 0.060*	0.940 ± 0.065	0.940 ± 0.048**

$*: p < 0.05$, $**: p < 0.01$

6.3. User study

We also conducted a user study to validate whether users can generate plausible results using our system in as few attempts as possible. We recruited eight users, six novice and two experienced users in image processing. The four images used in this study were Figures 1, 2, and 10 top and upper center. For each image, we showed

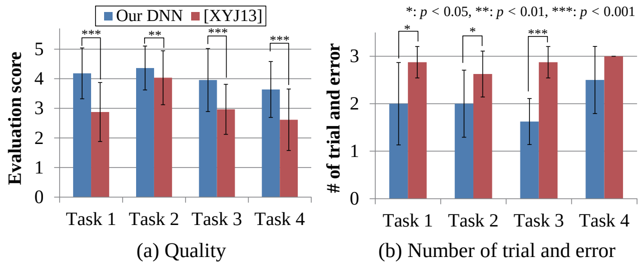

Figure 13: *Results of user study. Error bars show the standard deviation, and results marked with '*' show statistically significant differences as measured by paired t-test.*

sample results to the users and asked them to conduct the following tasks using our system and SCM [XYJ13], allowing them up to three trials until they are satisfied:

- Task 1 (Fig. 2): recolor groups of flower petals at different locations with the same colors.
- Task 2 (Fig. 10 upper center): recolor each type of cookies.
- Task 3 (Fig. 10 top): recolor only the peels of the pear.
- Task 4 (Fig. 1): colorize each object such as leaves, woods, and a cat.

Before conducting these tasks, we briefly explained how to use the systems and the image features to the users, and they practiced for about five minutes using other images such as Figure 6 top. Because the users might get used to editing the same image from the second time, we divided the users into two groups and shuffled the orders of images and methods for each group. For SCM, we limited the appropriate ranges of the feature weights from zero to five in order to avoid eccentric feature weights. The number of trials were recorded, and the resulting images were evaluated by another 11 evaluators using a subjective score ranging from one to five, as summarized in Figure 13. Thanks to the tuning-free characteristic of our system, the number of trials for the editing was typically around two, which was less than that of SCM. Additionally, the quality of resultant images using our system consistently outperforms that of the existing method.

7. Discussion and Future Work

Other possible approaches. In the context of the proposed method, it is possible to consider two other approaches: (1) data-driven pre-training and (2) network expansion.

As for (1), in several applications for computer vision, DNNs are pre-trained using datasets containing a large number of general images such as ImageNet [DDS*09] before training on data for specific tasks [LY15, LSL15]. This improves the generalization ability of models and accuracy on the tasks. In contrast, our method does not use such data-driven approaches. Nevertheless, it successfully generates several image editing results. It is not clear that data-driven pre-training would improve the task handled in this paper because learned features strongly depend on the variously defined user strokes. Additionally, applying such pre-trained models to our task is not straightforward because existing pre-trained models are designed for image-level classification, but our task is pixel-level classification. For example, Network in Network (NIN) [LCY13] and AlexNet [KSH12] require a relatively-large image as input, that is, an image of 227×227 pixels in order to classify the image as "scenery", "objects", or "animals", etc. On the other hand, our task needs to classify individual pixels in a single image, and thus we used relatively-small 9×9 image patches as input. If we were to disregard this essential difference and still integrate a pre-trained model into our framework, a naïve way would be to simply replace our VFE with a pre-trained model. We conducted such an experiment with NIN, which is a lightweight pre-trained model using ImageNet and whose performance is sometimes slightly better than AlexNet in image classification tasks. Specifically, we used NIN without its classification layer, which outputs 1000-dimensional feature vectors from input images, and fixed its model parameters during a learning session in order to leverage pre-trained features. Because NIN (and also AlexNet) requires images of 227×227 pixels, we magnified 9×9 input patches to 227×227 patches and fed these large patches to NIN. As clearly shown in Figure 14, the NIN model cannot appropriately capture visual features. Even worse, this approach took a significantly longer time (five minutes on average) despite the fact that the NIN model is lightweight one, as this network has a too deep structure for our task. Because such data-driven approaches are out of the scope of our research question, we consider such problems as future work.

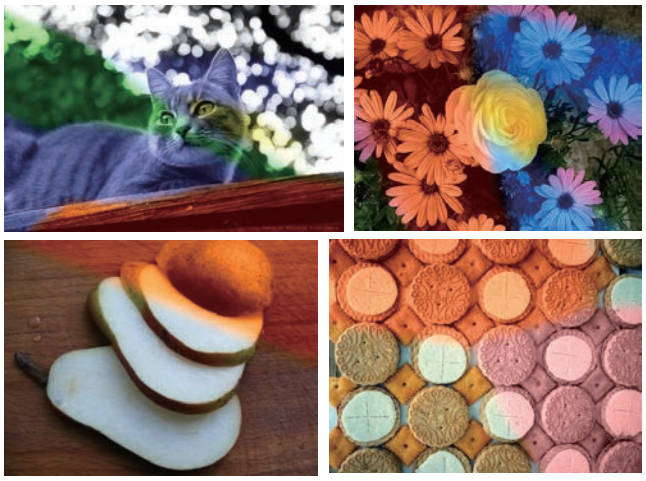

Figure 14: *Results with NIN model instead of our VFE. The same input image and user strokes are used for each image in Figures 1, 2, 10 top and upper center.*

As for (2), it is interesting to use more wide-scale networks, and we tried to use DNNs with deeper layers and larger kernels. However, there were no significant changes in results while the computational time increased when we used images with up to 2M pixels in our experiments. If we use larger images, a network that automatically changes its scale according to image sizes could possibly improve editing results.

GPU acceleration. Most of the modules in our system can be accelerated using parallel computation on GPU. To this end, the GPU-based SLIC algorithm for generating superpixels was proposed [RR11], linear solvers for the post-processing was also accelerated [JCVV09], and learning and estimation algorithms for

DNNs were parallelized [LNC*11]. The chainer library we used makes GPU implementation easy, and thus we tried to use GPU acceleration for deep learning with our method. The algorithm on PC with NVIDIA GeForce GTX 760 was about five times faster than CPU implementation. For high resolution images, our system may provide instant feedback with a low resolution result as a preview during editing sessions. We would like to accelerate our system for more interactive editing in the future.

8. Conclusion

We have proposed DeepProp, which achieves various image edits from only simple user strokes using deep leaning. As for our research question, we conclude that, without manual feature selection, effective features for edit propagation can be automatically extracted by (i) deep learning from sparse user inputs in a single image and (ii) efficiently learning the DNN in order to avoid falling into undesirable local solutions due to the vanishing gradient problem. Our edit propagation system using deep features has generated better results than previous work in several applications such as grayscale image colorization, image recoloring, and foreground segmentation. In the future, we plan to try the data-driven approaches for learning DNNs, design more flexible DNN structures to handle various images, and accelerate our system for more interactive editing.

Acknowledgements

We would like to thank Dr. Olga Diamanti for proofreading our paper. We also thank anonymous reviewers for their insightful suggestions. Yoshihiro Kanamori is funded by JSPS Postdoctoral Fellowships for Research Abroad.

References

[AHES09] ACHANTA R., HEMAMI S., ESTRADA F., SUSSTRUNK S.: Frequency-tuned Salient Region Detection. In *CVPR 2009* (2009), pp. 1597 – 1604. 9, 10

[AP08] AN X., PELLACINI F.: AppProp: All-pairs appearance-space edit propagation. In *ACM SIGGRAPH '08* (2008), pp. 40:1–40:9. 1, 3

[ASS*12] ACHANTA R., SHAJI A., SMITH K., LUCCHI A., FUA P., SUSSTRUNK S.: SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell. 34*, 11 (Nov. 2012), 2274–2282. 6

[Ben09] BENGIO Y.: Learning deep architectures for AI. *Found. Trends Mach. Learn. 2*, 1 (Jan. 2009), 1–127. 2, 3

[BHW11] BIE X., HUANG H., WANG W.: Real time edit propagation by efficient sampling. *Comput. Graph. Forum 30*, 7 (2011), 2041–2048. 3

[BLP*07] BENGIO Y., LAMBLIN P., POPOVICI D., LAROCHELLE H., MONTREAL U. D., QUEBEC M.: Greedy layer-wise training of deep networks. In *In NIPS* (2007), MIT Press. 3

[BPD09] BOUSSEAU A., PARIS S., DURAND F.: User-assisted intrinsic images. In *ACM SIGGRAPH Asia '09* (2009), pp. 130:1–130:10. 2

[CLKL14] CHO H., LEE H., KANG H., LEE S.: Bilateral texture filtering. *ACM Trans. Graph. 33*, 4 (July 2014), 128:1–128:8. 2

[CSRP10] CRIMINISI A., SHARP T., ROTHER C., P'EREZ P.: Geodesic image and video editing. *ACM Trans. Graph. 29*, 5 (2010), 134:1–134:15. 3

[CZL*14] CHEN X., ZOU D., LI J., CAO X., ZHAO Q., ZHANG H.: Sparse dictionary learning for edit propagation of high-resolution images. In *CVPR 2014* (2014), pp. 2854–2861. 1, 2, 3, 8

[CZZT12] CHEN X., ZOU D., ZHAO Q., TAN P.: Manifold preserving edit propagation. *ACM Trans. Graph. 31*, 6 (2012), 132:1–132:7. 1, 6

[Dau85] DAUGMAN J. G.: Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *J. Opt. Soc. Am. A 2*, 7 (1985), 1160–1169. 8

[DDS*09] DENG J., DONG W., SOCHER R., LI L.-J., LI K., FEI-FEI L.: ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR '09* (2009). 11

[DLH*13] DENG L., LI J., HUANG J., YAO K., YU D., SEIDE F., SELTZER M. L., ZWEIG G., HE X., WILLIAMS J., GONG Y., ACERO A.: Recent advances in deep learning for speech research at microsoft. In *IEEE ICASSP '13* (2013), pp. 8604–8608. 2, 3

[FFL10] FARBMAN Z., FATTAL R., LISCHINSKI D.: Diffusion maps for edge-aware image editing. *ACM Trans. Graph. 29*, 6 (Dec. 2010), 145:1–145:10. 3

[GBB11] GLOROT X., BORDES A., BENGIO Y.: Deep sparse rectifier neural networks. In *AISTATS '11* (2011), Gordon G. J., Dunson D. B., (Eds.), vol. 15, pp. 315–323. 3

[JCVV09] JOST T., CONTASSOT-VIVIER S., VIALLE S.: An efficient multi-algorithms sparse linear solver for GPUs. In *ParCo2009* (Lyon, France, Sept. 2009). 11

[KB14] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2014). 5

[KSH12] KRIZHEVSKY A., SUTSKEVER I., HINTON G. E.: ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. 2012, pp. 1097–1105. 2, 3, 11

[LAA08] LI Y., ADELSON E., AGARWALA A.: Scribbleboost: Adding classification to edge-aware interpolation of local image and video adjustments. In *EGSR '08* (2008), pp. 1255–1264. 1, 2, 3

[LCY13] LIN M., CHEN Q., YAN S.: Network in network. *CoRR abs/1312.4400* (2013). 11

[LFUS06] LISCHINSKI D., FARBMAN Z., UYTTENDAELE M., SZELISKI R.: Interactive local adjustment of tonal values. *ACM Trans. Graph. 25*, 3 (July 2006), 646–653. 2

[LGQ15] LANE N. D., GEORGIEV P., QENDRO L.: DeepEar: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *UbiComp '15* (2015), pp. 283–294. 3

[LJH10] LI Y., JU T., HU S.-M.: Instant propagation of sparse edits on images and videos. *Comput. Graph. Forum 29*, 7 (2010), 2049–2054. 1, 3, 8, 10

[LLW04] LEVIN A., LISCHINSKI D., WEISS Y.: Colorization using optimization. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 689–694. 1, 2, 6

[LNC*11] LE Q. V., NGIAM J., COATES A., LAHIRI A., PROCHNOW B., NG A. Y.: On optimization methods for deep learning. In *ICML* (2011), pp. 265–272. 12

[Low99] LOWE D. G.: Object recognition from local scale-invariant features. In *ICCV '99* (1999), pp. 1150–1157. 8

[LRAL07] LEVIN A., RAV-ACHA A., LISCHINSKI D.: Spectral matting. In *CVPR* (2007), IEEE Computer Society. 2, 6

[LSL15] LIU F., SHEN C., LIN G.: Deep convolutional neural fields for depth estimation from a single image. In *CVPR 2015* (June 2015). 3, 11

[LWA*12] LANG M., WANG O., AYDIN T., SMOLIC A., GROSS M.: Practical temporal consistency for image-based graphics applications. *ACM Trans. Graph. 31*, 4 (July 2012), 34:1–34:8. 3

[LWCO*07] LUAN Q., WEN F., COHEN-OR D., LIANG L., XU Y.-Q., SHUM H.-Y.: Natural image colorization. In *EGSR '07* (2007), pp. 309–320. 1, 2

[LY15] LI G., YU Y.: Visual saliency based on multiscale deep features. In *CVPR 2015* (June 2015). 3, 11

[MCY*13] MUSIALSKI P., CUI M., YE J., RAZDAN A., WONKA P.: A framework for interactive image color editing. *Vis. Comput. 29*, 11 (Nov. 2013), 1173–1186. 3

[NKK*11] NGIAM J., KHOSLA A., KIM M., NAM J., LEE H., NG A. Y.: Multimodal deep learning. In *ICML* (2011), pp. 689–696. 4

[PL07] PELLACINI F., LAWRENCE J.: AppWand: Editing measured materials using appearance-driven optimization. *ACM Trans. Graph. 26*, 3 (July 2007). 2

[QWH06] QU Y., WONG T.-T., HENG P.-A.: Manga colorization. *ACM Trans. Graph. 25*, 3 (July 2006), 1214–1220. 2

[RR11] REN C. Y., REID I.: *gSLIC: a real-time implementation of SLIC superpixel segmentation*. Tech. rep., University of Oxford, Department of Engineering Science, 2011. 11

[RRW*09] RHEMANN C., ROTHER C., WANG J., GELAUTZ M., KOHLI P., ROTT P.: A perceptually motivated online benchmark for image matting. In *CVPR '09* (2009), pp. 1826–1833. 2

[SCXP15] SUN J., CAO W., XU Z., PONCE J.: Learning a convolutional neural network for non-uniform motion blur removal. In *CVPR '15* (2015). 3

[SWW*15] SHEN W., WANG X., WANG Y., BAI X., ZHANG Z.: Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection. In *CVPR '15* (June 2015). 3

[WJW*14] WU Z., JIANG Y.-G., WANG J., PU J., XUE X.: Exploring inter-feature and inter-class relationships with deep neural networks for video classification. In *ACM MM '14* (2014), pp. 167–176. 4

[WLRY15] WANG L., LU H., RUAN X., YANG M.-H.: Deep networks for saliency detection via local estimation and global search. In *CVPR '15* (June 2015). 9

[WYHY15] WU J., YU Y., HUANG C., YU K.: Deep multiple instance learning for image classification and auto-annotation. In *CVPR 2015* (2015). 8

[XLJ*09] XU K., LI Y., JU T., HU S.-M., LIU T.-Q.: Efficient affinity-based edit propagation using k-d tree. In *ACM SIGGRAPH Asia '09* (New York, NY, USA, 2009), pp. 118:1–118:6. 1, 3

[XLXJ11] XU L., LU C., XU Y., JIA J.: Image smoothing via l0 gradient minimization. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 174:1–174:12. 2

[XWT*09] XU K., WANG J., TONG X., HU S.-M., GUO B.: Edit propagation on bidirectional texture functions. *Computer Graphics Forum 28*, 7 (2009), 1871–1877. 1, 2

[XXY*15] XIAO T., XU Y., YANG K., ZHANG J., PENG Y., ZHANG Z.: The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In *CVPR '15* (2015). 3

[XYJ13] XU L., YAN Q., JIA J.: A sparse control model for image and video editing. *ACM Trans. Graph. 32*, 6 (Nov. 2013), 197:1–197:10. 1, 3, 6, 8, 10, 11

[YS06] YATZIV L., SAPIRO G.: Fast image and video colorization using chrominance blending. *IEEE Transactions on Image Processing 15*, 5 (2006), 1120–1129. 2

[YY15] YATAGAWA T., YAMAGUCHI Y.: Sparse pixel sampling for appearance edit propagation. *The Visual Computer 31*, 6-8 (2015), 1101–1111. 1, 3

[YZW*15] YAN Z., ZHANG H., WANG B., PARIS S., YU Y.: Automatic photo adjustment using deep neural networks. *ACM Transactions on Graphics* (2015). 3