Pacific Graphics 2017
J. Barbic and O. Sorkine-Hornung
(Guest Editors)

Volume 36 (2017), Number 7

Patch2Vec: Globally Consistent Image Patch Representation

O. Fried[1] S. Avidan[2] D. Cohen-Or[2]

[1]Princeton University [2]Tel-Aviv University

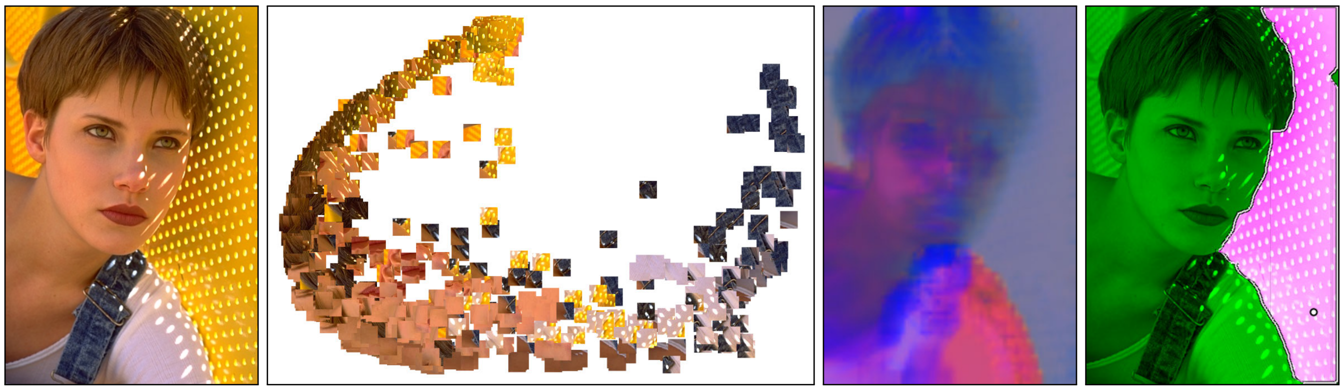


Figure 1: *(a) Input image. (b) A 2D embedding of patches in the image using* Patch2Vec. *Observe how patches with similar texture are clustered together. (c) Projecting the embedding to 3D for visualization purposes. (d) Single Click Segmentation. Given a single click (white circle) we automatically segment the fence.*

Abstract

Many image editing applications rely on the analysis of image patches. In this paper, we present a method to analyze patches by embedding them to a vector space, in which the Euclidean distance reflects patch similarity. Inspired by Word2Vec, we term our approach Patch2Vec. *However, there is a significant difference between words and patches. Words have a fairly small and well defined dictionary. Image patches, on the other hand, have no such dictionary and the number of different patch types is not well defined. The problem is aggravated by the fact that each patch might contain several objects and textures. Moreover, Patch2Vec should be universal because it must be able to map never-seen-before texture to the vector space. The mapping is learned by analyzing the distribution of all natural patches. We use Convolutional Neural Networks (CNN) to learn Patch2Vec. In particular, we train a CNN on labeled images with a triplet-loss objective function. The trained network encodes a given patch to a* 128*D vector. Patch2Vec is evaluated visually, qualitatively, and quantitatively. We then use several variants of an interactive single-click image segmentation algorithm to demonstrate the power of our method.*

Categories and Subject Descriptors (according to ACM CCS): I.4.10 [Image Processing and Computer Vision]: Image Representation—Multidimensional

1. Introduction

Patch representation is a central theme in the analysis and synthesis of images. Consider, e.g., image segmentation. At its core, segmentation requires a distance measure between different image regions. Similarly, retrieving patches for hole-filling, grouping image regions for faster computation and image based search queries are all tasks that require a robust way to represent patches.

Patch representation is not trivial, especially due to the prevalence of (non-regular) texture regions. It is quite simple to segment an image made of piece-wise constant colors, but much more challenging to do so when the image consists of textured regions.

Patches and textures can be represented in a number of different ways. Early attempts use a filter bank response; similar texture should have similar response. Alternatively, texture can be represented as a histogram or a Gaussian mixture model. Nowadays it is common to use raw pixel values as patch representation. This non-parametric representation leads to impressive results in applications such as texture synthesis or image denoising. Since patches often include textured regions, from here on we will use the terms "patch" and "texture" interchangeably, explicitly acknowledging the complexity of image patches.

The goal of the different representations is to map patches to some vector space where it is easy to compute meaningful distances. The distance measure between similar patches should hopefully be small to capture our perception of texture similarity. The representation in the cases mentioned above is fixed ahead of time, regardless of the data. Therefore it is difficult to ensure that the distance measure indeed captures texture similarity.

We are inspired by the work on Word2Vec that maps words with similar meaning to vectors with small distances between them. In our case, we wish to create an embedding space where the Euclidean distance between patches of similar texture is small. We term our approach *Patch2Vec*. However, there is a significant difference between words and image patches. Words have a fairly well defined dictionary. For our task, on the other hand, there is no such dictionary and the number of different textures and objects in photos is not well defined. Moreover, we aim to map never-seen-before texture to the vector space.

Consider Figure 1. The image on the left depicts a woman against a challenging background. Yet, with a single click we are able to segment the background based on distances between pixels in the embedding space (Figure 1(d)). This, we argue, is a strong indication to the quality of the embedding. In addition, we show an embedding of patches from the image to the 2D plane (Figure 1(b)). Observe how patches of similar texture are clustered together, regardless of texture shifts and illumination changes. In Figure 1(c) we use PCA to project the embedding vectors on the three largest principal components. The entire fence, that exhibits strong texture as well as shadows, is mapped to a single pseudo-RGB gray color, yet another indication to the power of our method.

The Patch2Vec mapping is universal. The mapping is learned by analyzing the distribution of all natural patches in all the images of our training set. This is in contrast to spectral methods, for example, that typically learn an image dependent embedding - changing an image will change the embedding. Another difference between spectral methods and Patch2Vec is that spectral methods are unsupervised by nature. As a result, at their core they still rely on some distance function between the raw pixel values of two patches. Patch2Vec, on the other hand, is defined in a supervised setting where the goal is to map pairs of patches that are deemed similar by *humans* to the same code in embedding space. This way the Euclidean distance in the embedding space reflects a perceptual similarity. Once we learn a universal mapping we can use it to process multiple images simultaneously or operate on images that change dynamically, say during editing.

We use Convolutional Neural Networks (CNN) to learn Patch2Vec. In particular, we train a CNN on a large training set of labeled images with a triplet-loss objective function. This objective function takes as input three patches. Two of them from the same object and another one from a different object. During training, the CNN learns to map patches of the same texture to nearby points in the Euclidean embedding space, while mapping other textures as far away as possible.

Once trained, patches are mapped to vector representation in a Euclidean space. Measuring perceptual similarity between patches now amounts to measuring the Euclidean distance between their corresponding embedded vectors. We evaluate Patch2Vec on some variants of a single-click image segmentation application. In this application, the user clicks on a single pixel and the application automatically extracts the appropriate segment. We then show that this basic functionality can be used in a multi-image single-click segmentation, as well as a super-pixel application. These extensions indicate the potential power of Patch2Vec.

The main contributions of this work are:

- A learning framework for image patch embedding, including a novel training regime, and the insight that a segmentation dataset can be used to supervise patch embedding (Sections 3 and 4).
- New single-click selection and super-pixel creation algorithms (Section 5).
- A new mask stability measure (Section 5.1).

2. Related Work

We deal with Representation Learning, Texture, and Image Segmentation. The literature on each of these topics is quite extensive. Here we highlight only research directly relevant to our work.

Representation Learning Word2Vec [MYZ13] reignited the interest in semantic embedding of words in vector spaces. It uses a Neural Network that was trained on a large dataset with billions of words and millions of words in the vocabulary.

Learning similarity measures between image patches has been actively investigated in the past. For example, Žbontar and LeCun [vL16] learn to do stereo matching by training a convolutional neural network to compare image patches. Observe that here the goal is to learn a similarity measure of the *same* 3D world under slightly different viewing directions.

Simo-Serra *et al.* [SSTF*15] learn feature point descriptors using a Siamese network. The output of their algorithm is 128D feature vector that can be used as a drop-in replacement for any task involving SIFT. They are similar to us in that they too learn a universal code for image patches. However, they focus on learning a code that is invariant to changes in the viewpoint, whereas we wish to learn a code that is invariant to fluctuations within a texture.

PatchNet [HZW*13] introduces a compact and hierarchical representation of image regions. It uses raw L*a*b* pixel values to represent patches, and Euclidean distance between these representations; we compare against using raw pixel values in Table 1. The goal of PatchNet is interactive library-driven image editing, and it includes other components besides patch representation (e.g. image graph representation). Our method can be used, as is, for patch representation within PatchNet. PatchTable [BZL*15] proposes an efficient way to perform approximate nearest neighbor (ANN) queries in large datasets. ANN is an orthogonal and complementary task to patch representation.

Moving beyond image patches, Schroff *et al.* [SKP15] proposed a network that learns how to embed face images. The network, termed FaceNet, learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity. Standard techniques for recognition, clustering and verification can then be used on the FaceNet feature vectors. We take inspiration from the FaceNet architecture in Section 3.

Recently, Ponjou Tasse and Dodgson [TD16] proposed a network that learns semantically meaningful shape descriptors. These descriptors are embedded in a vector space of words which leads to a cross-modal retrieval system.

All of the above are for specific types of images, for example faces or stereo pairs. We were inspired by these works, and aim to create a system that works for arbitrary image patches.

Texture Textons are an early representation of texture [Jul81] that encode second order statistics of small patches. This motivated extensive research on filter banks for texture classification [RH99]. Among the filters proposed are Gabor filters, wavelets and Discrete Cosine Transform. These filters are fixed and are not learned from data. Filter response in pyramids was used with great success for texture synthesis [HB95,BV98]. It was later reported that raw pixel values are as informative as filter bank response [VZ03].

Patch based methods are used with great success in various applications such as texture synthesis [EL99], image denoising [BCM05] and hole filling [BSFG09]. For a survey on patch-based synthesis see Barnes and Zhang [BZ17].

Interactive image segmentation also rely on texture analysis. For example, GrabCut [RKB04] represents the foreground and background regions of the image using a Gaussian Mixture Model (GMM). Pixel label is based on its distance from each GMM. Similarly, geodesic matting [BS09] computes geodesic distance on a probability image that is based on the distance of each pixel to the GMM of the foreground or background. Multi-scale analysis was used for texture synthesis [LSA*16].

More recently, deep networks have been used to represent and segment textures [CMK*14, CMV15, LM16]. The method of Cimpoi et al. [CMK*14] explore images of textures (i.e., a single texture fills the entire image), while our approach is geared towards natural images. In a later work Cimpoi et al. [CMV15] use the last convolution layer of a convolutional neural network (CNN) as an image region descriptor. Using their method to describe each image patch will result in a 65K dimensional vector per patch, as opposed to our 128 dimensional representation. Fully Convolutional Networks (FCNs) [LSD15] produce impressive results for pixel-accurate tasks such as image segmentation. We compare to FCNs in Section 4. Lin et al. [LM16] aim for invariance to categorical variations, which is the opposite of our goal.

Image Segmentation We use the Berkley Segmentation Dataset (BSDS500) [MFM04] to train our CNN. We demonstrate its power on an interactive single-click image segmentation application.

Interactive image segmentation has been investigated extensively in the past and two prime examples are GrabCut [RKB04] and Geodesic Matting [BS09]. Both these algorithms use a Gaussian Mixture Model to model the distribution of RGB colors in the object and the background. In addition, they require user input in the form of scribbles or a bounding box, similarly to other methods (e.g., [BPK*13]). In our application, on the other hand, we use our features and require just a single point click from the user. We compare to scribble-based methods in Section 4.

Single click interactive segmentation was also proposed by Bagon *et al.* [BBI08] in the context of "Segmentation by Composition". They define an image segment as one that can be easily composed from its own pieces, but difficult to compose from other pieces in the image. They use this definition to extract a segment using a single user click. (Single-click segmentation should not be confused with point supervision for *training* [BRFFF16].) Xu et al. [XPC*16] train on input images and generated click-locations for an end-to-end supervised segmentation network. Their implementation is not available online for comparison.

3. Patch Embedding

Our goal is to embed image patches into a low-dimensional representation, such that l_2 distances in the representation space correspond to some notion of patch similarity, with a focus on textured patches. Specifically, we would like to learn a universal embedding operator $f(p)$ such that for two given patches p_1 and p_2, the distance $\|f(p_1) - f(p_2)\|_2$ is small if p_1 and p_2 are similar textures, and large otherwise.

We use labeled images to learn the embedding in a supervised manner. A key observation is that humans tend to "understand" textures, thus a segmentation dataset is suitable as a guide for texture-aware patch embedding. We use a neural network to learn the patch embedding space. It should be noted that we aim for the network to be applicable to all natural image patches, and not be domain specific (e.g., face embedding [SKP15]).

During training, we deem patches that were annotated as part of the same segment as "positive pairs" and pairs from different segments as "negative pairs". We use a triplet loss for training: given an anchor patch p_a which makes a positive pair with p_p and a negative pair with p_n, the loss for a single triplet is defined as:

$$L(p_a, p_p, p_n) = [\|f(p_a) - f(p_p)\|_2^2 - \|f(p_a) - f(p_n)\|_2^2 + m]_+, \quad (1)$$

where m is a margin value (set empirically to 0.2) and $[x]_+$ is defined as $\max\{0, x\}$. The triplet loss is a sum over all anchor-positive-negative triplets in the dataset $\mathcal{D}$:

$$L = \sum_{(p_a, p_p, p_n) \in \mathcal{D}} L(p_a, p_p, p_n). \quad (2)$$

Notice that while p_a and p_p are by definition from the same photo, the negative example p_n can be from either a different segment in the same photo or from a different photo altogether. However, it is crucial to select p_n from the same photo, thus producing a triplet which is closer to the separation margin, as patches from the same photo are more likely to be correlated. Moreover, selecting the entire triplet from the same photo produces a context-aware learning mechanism. In this way, we are effectively learning an embedding that separates patches from different segments *which co-occur in* *the same natural scene*. This context-aware exemplar selection separates us from previous works. In Schroff et al. [SKP15], e.g., all face identities are incorporated in each training batch.

A careful triplet selection is crucial for good convergence. In each mini-batch, we use only "hard" examples for training. Specifically, for each anchor-positive pair (p_a, p_p) we find the set $\mathcal{N}$ of all negative patches p_n such that the loss falls within margin m:

$$\mathcal{N}(p_a, p_p) = \{p_n \mid \|f(p_a) - f(p_n)\|_2^2 - \|f(p_a) - f(p_p)\|_2^2 < m\}. \tag{3}$$

If multiple such patches exist, we select one at random.

For ground-truth labels, we use the Berkeley Segmentation Dataset (BSDS500) [AMFM11]. We randomly sample 50,000 32×32 patches from each image in the BSDS500 training set. If patches extend beyond image boundaries, we pad the original image, repeating the boundary pixel values. We have experimented with other patch sizes and found 32×32 to perform well while keeping memory overhead manageable. Figure 4 Shows results using patches of size 16×16.

We use a similar architecture to FaceNet [SKP15] as implemented by the OpenFace project [ALS16], with changes to accommodate the difference in patch sizes. Appendix A contains the neural network specification in Torch. More importantly, we changed the training regime as explained above. After training most values of $f(p)$ are in the range $[-0.3, 0.3]$. Over the whole dataset $f(p)$ follows a Gaussian distribution ($\mu = 0.012, \sigma = 0.088$).

The network was trained on a (shared) Linux machine with an Intel Xeon Processor E5-2699 v3 and a Tesla K40 GPU, for 700 epochs. Training took approximately 50 hours.

4. Evaluation

We evaluate Patch2Vec on images that were not part of the training set and report both qualitative and quantitative results. Later on we use Patch2Vec for single-click image segmentation. We believe that this application is a good test bed to evaluate the quality of the embedding because it addresses a real-world problem while using the minimal amount of user input possible, namely a single point click. This puts a heavy burden on the representation, which is precisely our goal. Then we show a couple of possible extensions to highlight the potential power of Patch2Vec.

4.1. Qualitative Evaluation

A simple way to visualize the embedding is to project the $128D$ codes on the three leading principal components, producing a pseudo-RGB image. Figure 2 shows results on a variety of images. In particular, observe how Patch2Vec maps various textures to uniform pseudo-RGB colors.

When a deep neural network (DNN) is used to process images, each layer or combination of layers can be considered a form of patch embedding. Figure 3 compares our embedding to other DNN methods (UVRL [DGE15], FCN [LSD15]). We use the same visualization as in Figure 2. Note how our method produces similar embeddings (similar visualized color) for pixels of the same region, unlike the other methods. When comparing to Doersch et al. [DGE15] we used their pre-trained weights, and also retrained on BSDS. We use their `fc6` layer which performed the best in our tests. Each `fc6` descriptor contains 4096 values; embedding a 5MP image requires more than 20 *billion* values. We require $128/4096 = 3\%$ the amount of storage. Note that UVRL is a self-supervised method.

In Figure 3 we also show results using the `relu3-4` layer of FCN [LSD15]. Using other layers produced worse results. We observe inferior embeddings when using FCN, which suggests that optimizing for, e.g., label prediction, does not imply that similar patches would have similar embeddings. On the contrary – they can have very different embeddings, and combining the information leads to a successful labeling. Another disadvantage of convolutional approaches is shown in Figure 5. When stitching several images, the different context changes the patch embeddings. In our method, an input patch will always produce the same embedding, even when stitched to other images or after being used in a hole-filling operation. It is important to note that FCN is faster than our method due to the convolutional nature of their approach. Finding a good way to rephrase our method in a convolutional setting is a promising direction for future work.

4.2. Quantitative Evaluation

The first experiment we report measures the capability of the embedding to determine whether a pair of patches is part of the same object or not. For each test image we sample 100,000 positive pixel pairs (belonging to the same segment) and 100,000 negative pixel pairs (belonging to different segments). For each pair (p_1, p_2) we measure the distance d between the two patch embeddings

$$d(p_1, p_2) = \|f(p_1) - f(p_2)\|_2. \tag{4}$$

where $f(p)$ is our patch embedding operator. Given a threshold t we can define a binary classifier $C(p_1, p_2)$ that determines whether the patches belong to the same segment or not:

$$C(p_1, p_2) = \begin{cases} \text{same} & \text{if } d(p_1, p_2) < t \\ \text{different} & \text{if } d(p_1, p_2) \geq t \end{cases}. \tag{5}$$

We calculate the area under the curve (AUC) of the receiver operating characteristic curve for all t values. Higher AUC implies a better classifier. We apply the above procedure on the training, validation and test sets of the BSDS500 dataset [AMFM11]; only the training set was used in Section 3 to create our embedding.

Instead of using $f(p)$ we can define other operators and repeat the above procedure. In particular, we compare to:

- *Raw pixels (RGB)*: RGB values of a given patch.
- *Raw pixels (L*a*b*)*: CIELAB values of a given patch.
- *Mean color*: average patch color.
- *Gabor*: response of a filter bank consisting of multiple Gabor filters at different orientations and scales.
- UVRL: `fc6` layer from Doersch et al. [DGE15]. Other layers produced lower scores.
- FCN: `relu3-4` and output labels from Long et al. [LSD15]. Other layers produced lower scores. Note that l_2 is not suitable for output labels. Instead we use distance = 0 if labels match, otherwise distance = 1.

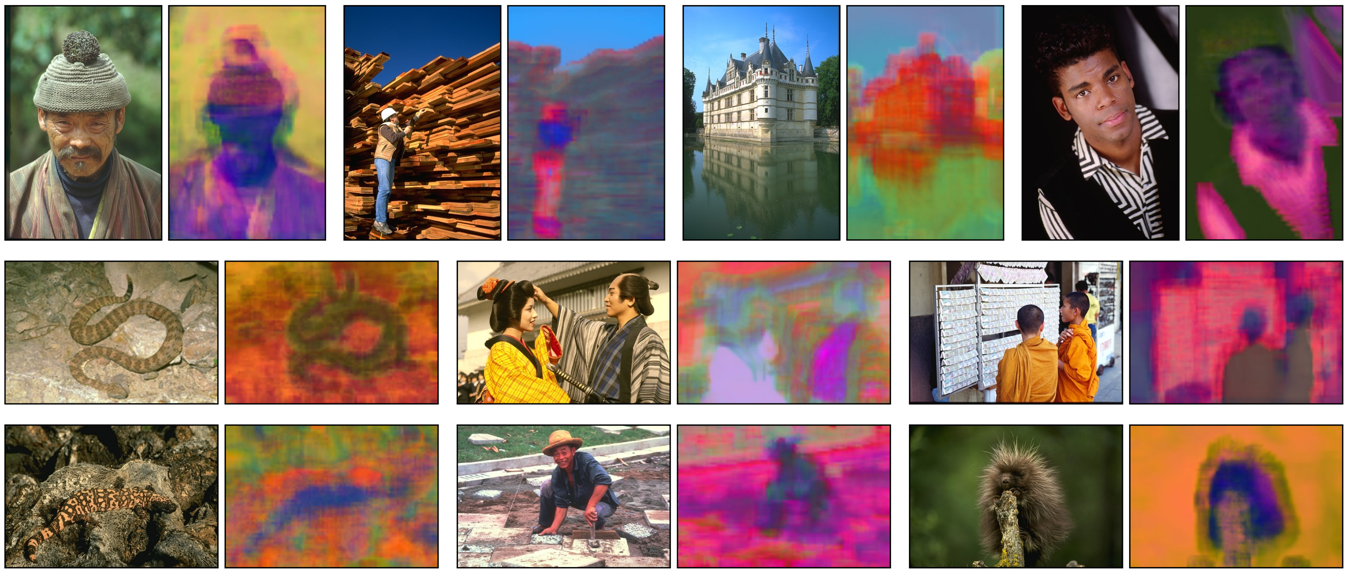


Figure 2: *Visualizing Patch2Vec. We project the* 128*D embedding vectors on a 3D space and visualize it as pseudo RGB colors. Observe how, for example, the shirt of the person at the top row on the right is mapped to a nearly constant color. We are also able to assign different pseudo colors to the building and its reflection (top row) even though they are very similar in appearance.*

Method	Mean AUC
Random	0.50
FCN (relu3-4)	0.60
FCN (labels)	0.65
Gabor	0.66
Raw pixels (RGB)	0.69
Mean color	0.70
UVRL	0.70
Raw pixels (L*a*b*)	0.73
Our method (validation)	0.75
Our method (testing)	0.76
Our method (training)	0.78
Human	0.86

Table 1: *Same-Not-Same Evaluation. We measure the capability to predict if a pair of patches comes from the same segment or not, by calculating AUC scores using l_2 distance between patch representation for prediction. We compare to established patch representation methods (e.g., Gabor) and to representative deep neural methods (UVRL [DGE15], FCN [LSD15]). Higher score is better, our method outperforms the others by 0.03, which is 8% of the total span of values between random and human prediction.*

Table 1 summarizes all AUC scores. Values range between a random selection (0.50) and human performance (0.86), which is calculated by predicting one annotator using another (BSDS500 contains several annotations per image). Our method scores 0.76, compared to the next best method that scored 0.73. Interestingly, FCN produces the lowest scores, despite the success of fully convolutional networks in tasks such as image segmentation. This shows that optimizing for correct pixel labels is inherently different from our goal of finding good patch embeddings.

5. Applications

We start by showing single-click segment selection and use it to demonstrate the strength of our embedding (Section 5.1). Such a narrow information channel between the user and the algorithm is best suited to investigate the properties of Patch2Vec. We show selection results, investigate selection stability given different click locations, and compare to other selection methods. Note that we are not claiming single-click selection to be the optimal interaction method. On the contrary, in many cases it is preferable to supply more clicks for better segmentation results. However, a single-click is a good *evaluation* method for patch embeddings.

In Section 5.2 we extend selection to multiple images, and introduce a super-pixel creation algorithm that uses our embedding.

5.1. Single-Click Segment Selection

Given a photo $\mathcal{I}$, we calculate patch embeddings $f(p)$ for each 32×32 image patch in a preprocessing step. We allow patch overlap and pad $\mathcal{I}$ such that the number of patches and pixels is identical. At runtime, the user clicks a single pixel location that corresponds to patch p_c. For all other patches, we calculate the embedding distance

$$d_p := d(p, p_c) = \|f(p) - f(p_c)\|_2 \quad \forall p \in \mathcal{I}, \tag{6}$$

which yields a per-pixel distance value. Next, we threshold the distances using Otsu's method [Ots79] to produce a binary selection

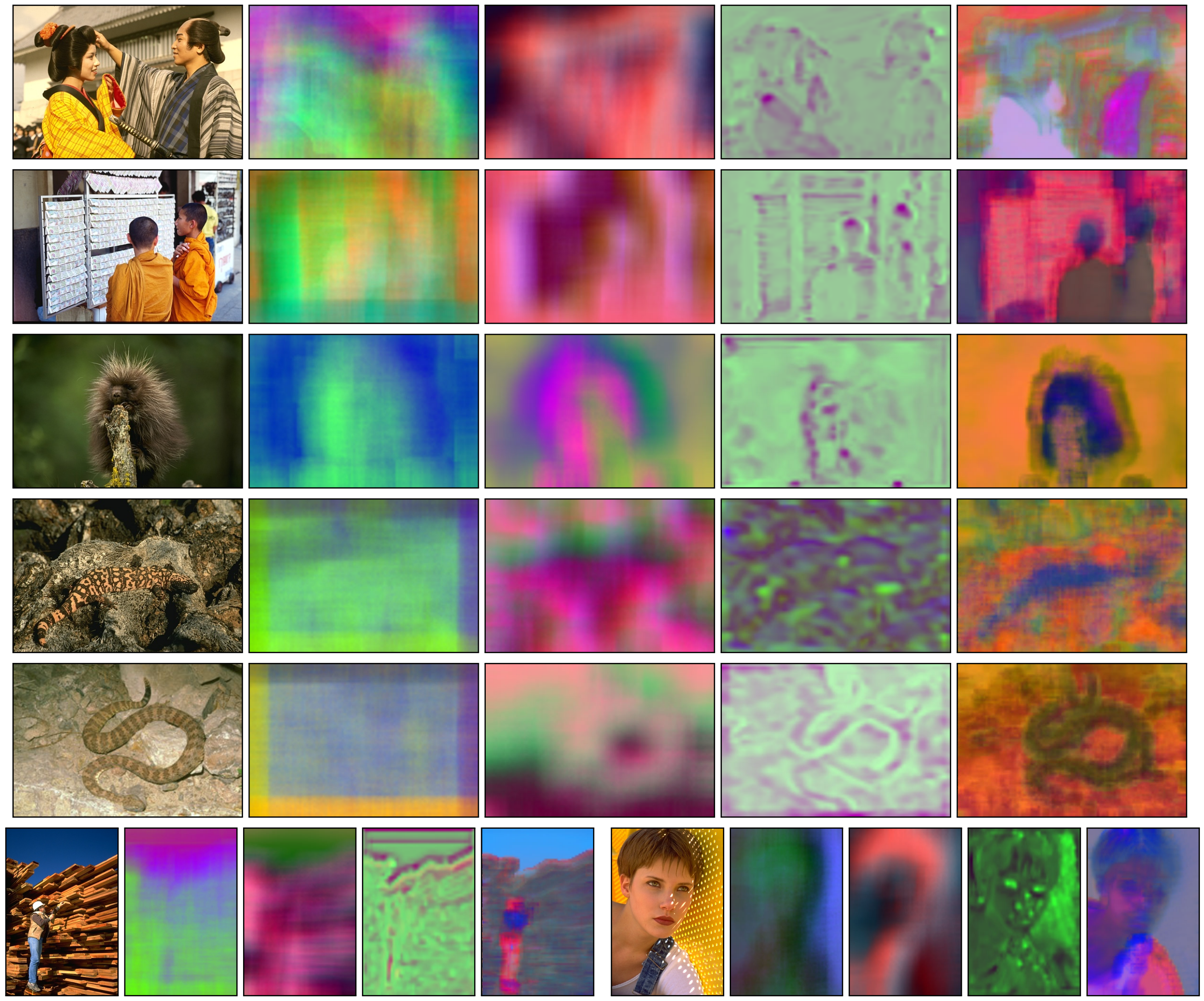


Figure 3: *Embedding comparison. Left to right: input image, the method of Doersch et al. [DGE15] with their pre-trained network weights, the method of Doersch et al. re-trained on BSDS [AMFM11], Fully Convolutional Networks for Semantic Segmentation [LSD15], and our embedding result. Visualization method is the same as in Figure 2. See text for details.*

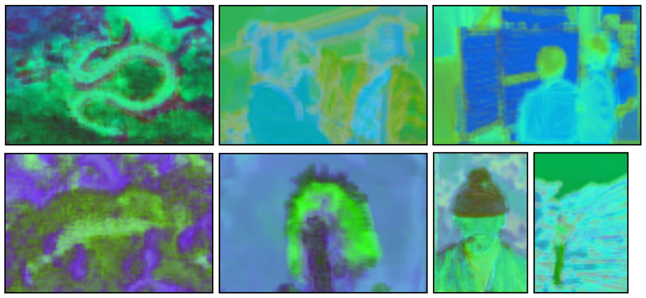


Figure 4: *Embeddings using* 16×16 *patches. Some results are comperable to* 32×32 *patches (e.g. top-right), others are inferior (e.g. bottom-left). Input images in Figure 2.*

mask. As an optional step, the mask can be refined using snakes [KWT88] to better snap the selection to image edges.

If preprocessing runtime is a constraint, we compute an embedding for pixels in K-pixel strides (horizontally and vertically), and interpolate for the rest of the pixels. This reduces computation time by a factor of K^2. All results shown use $K = 5$; runtime is 2.6×10^{-4} seconds per pixel on standard hardware. Pre-processing a 320x480 RGB image takes 40 seconds and runtime scales linearly with the number of pixels.

Figure 6 shows single-click selection results. Importantly, the click locations were randomly selected and not hand picked. Examining Figure 6, we notice a few interesting points. Notice how the mask distinguishes between "real" edges and intra-texture edges. A single click on a textured shirt or a spotted animal is enough to

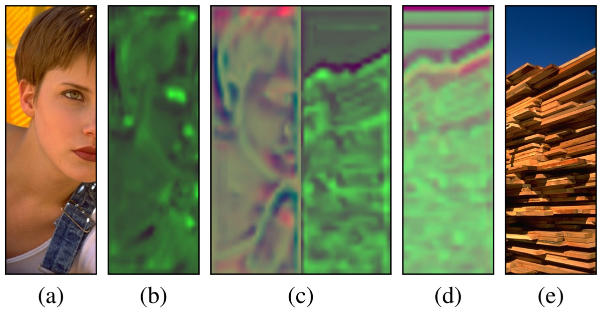


Figure 5: *We demonstrate the context dependency of FCNs [LSD15]. (a) and (e) are concatenated to create a new photo. FCN result (c) on the concatenated photo is different from original results (b) and (d). Context dependency is useful, but can also be a drawback. E.g., consistent embedding may be required after image stitching, hole-filling or when operating on multiple photos. For comparison, see our results in Figure 3 which remain identical after stitching the two image halves, by construction.*

select it all. Another important property is that training on pixel-accurate masks allows a click near the segment boundary to produce a successful mask. Notice, e.g., the white statue on the third row. The click is adjacent to a segment boundary, which in turn implies that the surrounding patch includes elements from several segments. Despite that, we managed to learn that the center pixel of the patch is a part of the statue segment.

We next want to verify that our method is robust to variations in click locations. Figure 7 shows some qualitative examples. Observe how different seed points lead to visually similar masks. To quantify the notion of mask stability, we propose the following mask stability measure. Let $M_1, \dots M_n$ be n masks, where mask M_i is a binary image that equals 1 for segment pixels and 0 anywhere else. Let $M = \frac{1}{n}\sum_{i=1}^{n} M_i$ be the average mask. Define:

$$\text{IoU}_i = \frac{\sum_x M(x) M_i(x)}{\sum_x M(x)}, \tag{7}$$

where x is pixel coordinate. In words, IoU_i sums the pixels in M that belong to the segment according to mask M_i, normalized by the sum of all pixels in M. The final stability score is:

$$\text{stability} = \frac{1}{n}\sum_{i=1}^{n} \text{IoU}_i \tag{8}$$

If all masks are exactly the same the score is 1. If most masks contain the same k pixels, while a small group of outlier masks $n_o \ll n$ contain a different disjoint set of k pixels, stability score is $1 - {}^{2n_0}/_{n} + {}^{2n_0^2}/_{n^2} \to 1$. If half the masks agree on a set of k pixels and half agree on a disjoint set of k pixels, stability score is 0.5. Thus, our stability score measures mask agreement, and is robust against outlier masks.

Equipped with this stability measure, we conduct the following experiment. For a given image and a non-trivial ground truth segment (larger than 5% of the size of the image), we randomly sample $n = 10$ seed points within the segment and use each one independently to construct a mask, giving us a total of 10 masks per segment. We then compute stability per segment (Equation 8) and average over 866 image segments. For comparison, we repeat this protocol with Diffusion Map embedding [FFL10]. Specifically, for a given image, we compute a 128D embedding using Diffusion Maps and then apply the protocol. Results are reported in Figure 8. The figure shows a histogram of stability scores using both methods, as well as a typical example (top row) of an average mask for one image. The mask stability score for the example shown in the figure is 0.65. The average stability score, across the entire set, of Patch2Vec is 0.54, compared to 0.40 for Diffusion Maps.

Figure 9 compares our single-click selection to several other masking methods. A scribble based method [LRL08] can produce plausible results, but requires several scribbles (this holds for other scribble-based methods). Diffusion Maps does not produce good masks, even when using the best possible time value t, measured by comparing to ground-truth segmentation and finding the optimal operating point of the Receiver Operator Characteristic (ROC) curve . GrabCut [RKB04] also does not produce the expected result, even when given a *tight* bounding box. Classic descriptors such as Gabor filter banks, which historically were used to describe textures [JF91] cannot mask the full segment via a single click. We use 8 orientations and 16 wavelengths to produce a 128D Gabor descriptor (same length as our embedding).

5.2. Extensions

Single-Click Image Segmentation is a core algorithm that can be extended in a number of ways.

The first extension is to multi-image single click segmentation. This scenario is applicable in case we have multiple images, or a video, of some event and we want to segment the same texture across multiple images. Given the seed pixel selected by the user, we compute the distance, in embedding space, between the seed pixel and all the pixels in all the images. We then proceed as in 5.1. Results are shown in Figure 10. The sequence consists of 82 frames and the user clicks on just a single point in the first frame. The method can handle occlusions and appearance changes without resorting to tracking or higher level computer vision algorithms. This demonstrates the potential power of a *universal* patch representation such as Patch2Vec.

The second extension is a super-pixel application. Super-pixels are a mid-level image representation that is often the first step in many image processing and computer vision algorithms.

We compare Patch2Vec to two popular super-pixels methods. The first is SLIC [ASS*12], that uses K-means clustering in x-y-l-a-b space to create the segmentation. SLIC is simple to code and quite fast, however some important edges are occasionally missed, creating super-pixels that span two or more image segments. The second is based on edge maps to guide the algorithm [DZ13]. Our algorithm is similar to [DZ13] but replaces La*b* values with the first three principle components of Patch2Vec.

Figure 11 shows a comparison between the methods. Both Dollár et al. [DZ13] and our method adhere to edges better than SLIC. Our approach is the only one that can distinguish between inter-segment edges, that separate the target object from the background,

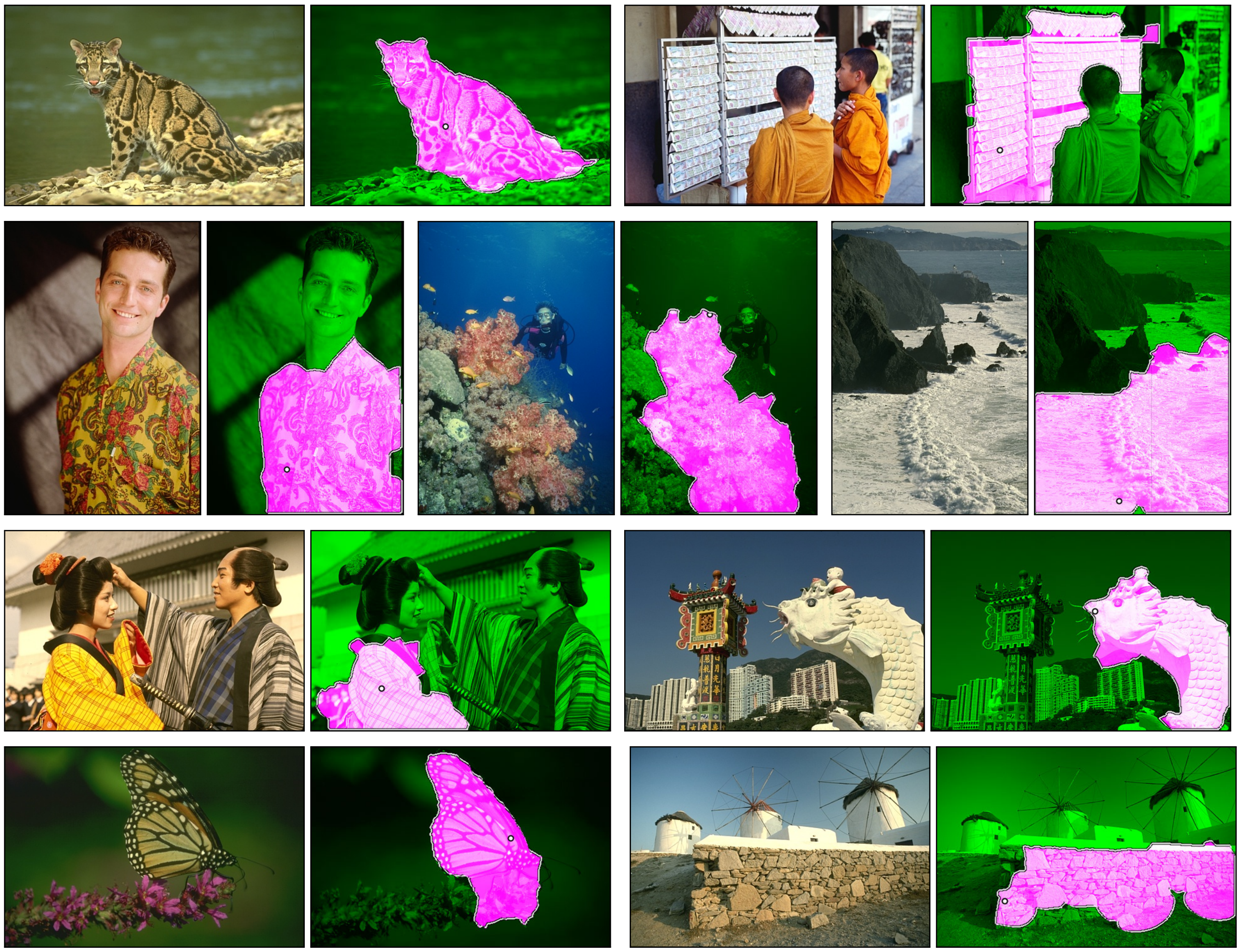


Figure 6: *Single-click selection results. The user clicks on a single pixel (white dot), which in turn produces a selection mask. Notice how textured regions such as plaid clothing or butterfly wings are selected as a single region. The user can fix errors (e.g. part of a branch selected with the butterfly) with consecutive clicks.*

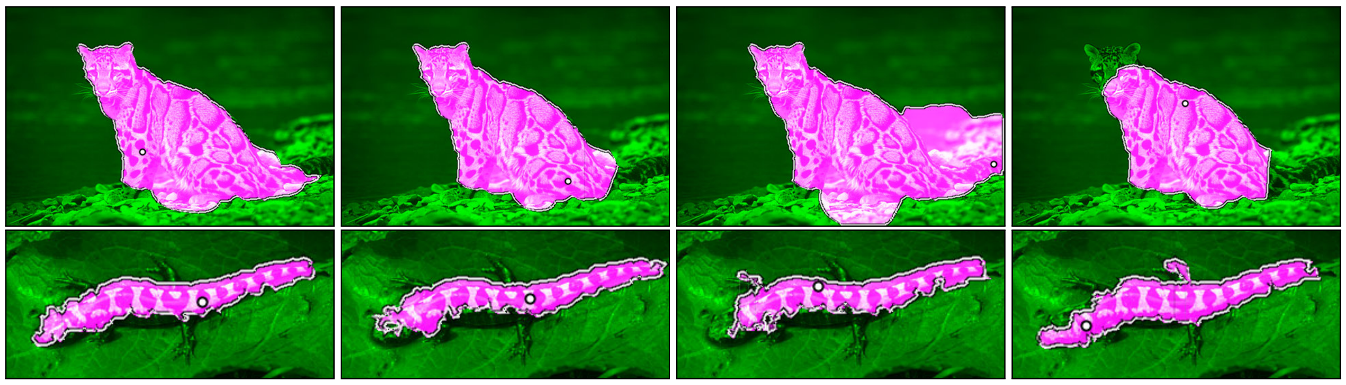


Figure 7: *Single-click selection stability. Randomly chosen click locations (white dots) for each segment produce similar selection masks. We show results including edge snapping (top) and without snapping (bottom). Notice how, e.g., click locations on the lizard texture include dark and bright spots, yet the selection masks remain stable.*

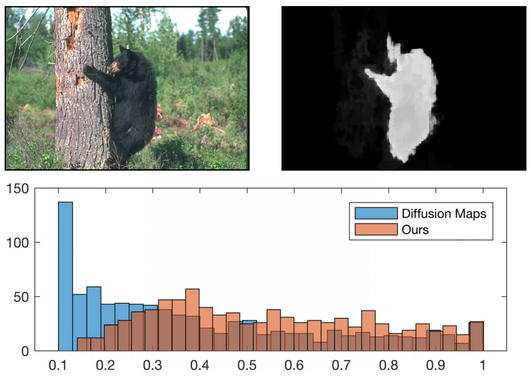


Figure 8: *Single-Click Selection Stability: (Top) A typical image (left) and its average mask for a particular segment (right). The stability score for this segment is* 0.65 *(see text for details). (Bottom) Histogram of mask stability scores on a data set of 866 image segments using Patch2Vec and Diffusion Maps. The average stability score of masks generated using our method is* 0.54*, compared to* 0.40 *obtained by Diffusion Maps.*

and intra-segment edges (e.g. the notable edges on the textured fur). Each super pixel in the figure is assigned the average RGB color of all its pixels. Ideally, we want all super pixels of a texture to have similar properties (e.g. same color), which will make it easier for higher level algorithms to cluster them together.

Figure 12 shows results of our super-pixel algorithm on the NYU Depth Dataset [NSF12]. We chose a dataset with different image statistics from the one we trained on, to show that our network did not overfit to a specific photo type.

6. Conclusions and discussion

Patch2Vec offers a universal embedding of image patches. The embedding maps patches with similar textures to nearby vectors in the embedded space. As a result, Euclidean distance in that space corresponds to the perceptual distance of humans. This is in contrast to common methods that define distances between patches based on low-level analysis of the raw pixel values of the patches.

However, the implementation is still limited. We use a fairly small training set (the BSDS500 data set). This affects the embedding quality because it is bounded by the size and characteristics of the training set. Working with more data, and with better and richer data augmentation, can further improve the embedding. Another problem that we currently face is that we use an image segmentation dataset to train our model. As a result we treat image segments as having the same texture, which is not always the case in practice. This can be addressed by collecting more data geared towards Patch2Vec.

We believe that Patch2Vec opens the door to a wide variety of image editing applications. In particular, we demonstrate a number of possible use cases. The first is a single click segmentation scenario, in which the user clicks a single point and the system determines the image segment automatically, based on patch similarities in the embedded space. We next presented a multi-image single click segmentation where a single click in one image is propagated to other images automatically. This is made possible by the universal property of the embedding that lets us measure distances between patches taken from different images. Finally, we have used Patch2Vec representation for super pixel generation, by replacing pixel RGB values with our embedding vectors.

In the future, we would like to reduce the dependency on labeled training data, using possibly self-supervised or unsupervised techniques. We would also like to explore ways to synthesize data that will help us refine and augment the training set.

7. Acknowledgements

This work was supported by the Israeli Science Foundation, research programs 1790/12 and 2366/16.

References

[ALS16] Amos B., Ludwiczuk B., Satyanarayanan M.: *OpenFace: A general-purpose face recognition library with mobile applications*. Tech. rep., CMU-CS-16-118, 2016. 4, 12

[AMFM11] Arbelaez P., Maire M., Fowlkes C., Malik J.: Contour detection and hierarchical image segmentation. *TPAMI 33*, 5 (May 2011), 898–916. 4, 6

[ASS*12] Achanta R., Shaji A., Smith K., Lucchi A., Fua P., Süsstrunk S.: SLIC superpixels compared to state-of-the-art superpixel methods. *TPAMI 34*, 11 (2012), 2274–2282. 7, 11

[BBI08] Bagon S., Boiman O., Irani M.: What is a good image segment? a unified approach to segment extraction. In *ECCV* (2008), pp. 30–44. 3

[BCM05] Buades A., Coll B., Morel J.-M.: A non-local algorithm for image denoising. In *CVPR* (2005), CVPR '05, IEEE Computer Society, pp. 60–65. 3

[BPK*13] Baek J., Pająk D., Kim K., Pulli K., Levoy M.: Wysiwyg computational photography via viewfinder editing. *ACM Trans. Graph. 32*, 6 (2013), 198:1–198:10. 3

[BRFFF16] Bearman A., Russakovsky O., Ferrari V., Fei-Fei L.: *What's the Point: Semantic Segmentation with Point Supervision*. Springer International Publishing, Cham, 2016, pp. 549–565. 3

[BS09] Bai X., Sapiro G.: Geodesic matting: A framework for fast interactive image and video segmentation and matting. *IJCV 82*, 2 (2009), 113–132. 3

[BSFG09] Barnes C., Shechtman E., Finkelstein A., Goldman D. B.: PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph. 28*, 3 (Aug. 2009). 3

[BV98] Bonet J. S. D., Viola P. A.: Texture recognition using a nonparametric multi-scale statistical model. In *CVPR* (1998), pp. 641–647. 3

[BZ17] Barnes C., Zhang F.-L.: A survey of the state-of-the-art in patch-based synthesis. *Computational Visual Media 3*, 1 (Mar 2017), 3–20. 3

[BZL*15] Barnes C., Zhang F.-L., Lou L., Wu X., Hu S.-M.: Patchtable: Efficient patch queries for large datasets and applications. In *ACM Trans. Graph. (Proc. SIGGRAPH)* (Aug. 2015). 2

[CMK*14] Cimpoi M., Maji S., Kokkinos I., Mohamed S., Vedaldi A.: Describing textures in the wild. In *CVPR* (2014), CVPR '14, IEEE Computer Society, pp. 3606–3613. 3

[CMV15] Cimpoi M., Maji S., Vedaldi A.: Deep filter banks for texture recognition and segmentation. In *CVPR* (June 2015), pp. 3828–3836. 3

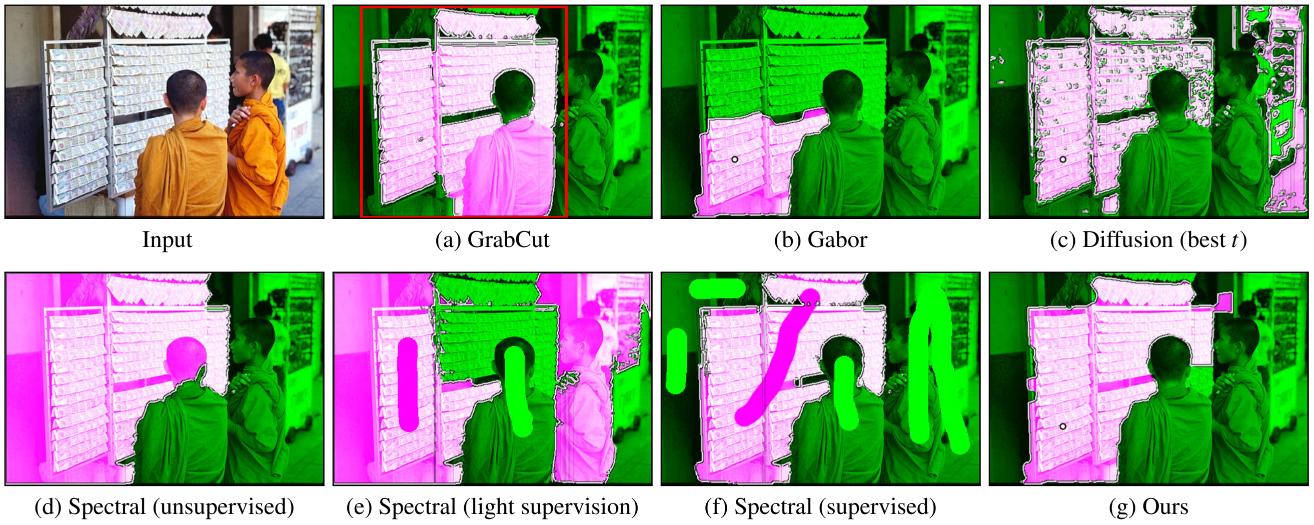

Input
(a) GrabCut
(b) Gabor
(c) Diffusion (best t)
(d) Spectral (unsupervised)
(e) Spectral (light supervision)
(f) Spectral (supervised)
(g) Ours

Figure 9: *Masking algorithm comparison. (a) GrabCut [RKB04] does not correctly separate the person. (b) Replacing our patch embedding with a Gabor filter bank response produces a partial segment. (c) Diffusion maps [FFL10] using the best possible time value t. (d) Unsupervised spectral matting [LRL08] does not accurately separate the texture, and lacks user-based control. (e) Two scribbles cannot correct the mask. (f) Fully supervised spectral matting produces an accurate result, but requires several scribbles. (g) Our single-click result. The input for (a) is a bounding box (red), for (b) (c) and (g) is a* single *pixel location (white circle), and for (e) and (f) is several scribbles.*

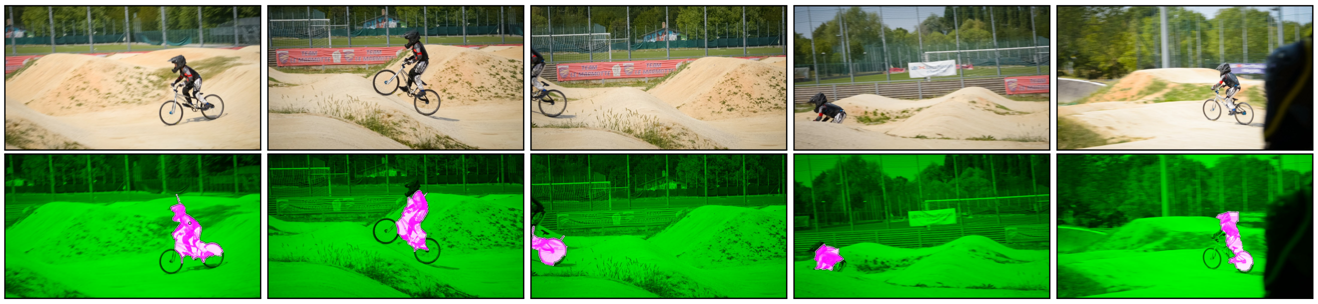


Figure 10: *Single-click multi-frame selection. Given an input video (showing frames 0, 15, 33, 50, 81), the user clicks on a single seed point (white dot, bottom-left image). We segment* all *images using distances to this single seed point. Notice how we can select the object without explicit template matching, despite appearance changes and occlusions. This demonstrates the universality property of our embedding.*

[DGE15] DOERSCH C., GUPTA A., EFROS A. A.: Unsupervised visual representation learning by context prediction. In *ICCV* (2015). 4, 5, 6

[DZ13] DOLLÁR P., ZITNICK C. L.: Structured forests for fast edge detection. In *ICCV* (2013). 7, 11

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. In *ICCV* (1999), pp. 1033–1038. 3

[FFL10] FARBMAN Z., FATTAL R., LISCHINSKI D.: Diffusion maps for edge-aware image editing. *ACM Trans. Graph. 29*, 6 (2010), 145:1–145:10. 7, 10

[HB95] HEEGER D. J., BERGEN J. R.: Pyramid-based texture analysis/synthesis. In *SIGGRAPH* (1995), pp. 229–238. 3

[HZW*13] HU S.-M., ZHANG F.-L., WANG M., MARTIN R. R., WANG J.: Patchnet: A patch-based image representation for interactive library-driven image editing. *ACM Trans. Graph. 32*, 6 (Nov. 2013). 2

[JF91] JAIN A. K., FARROKHNIA F.: Unsupervised texture segmentation using gabor filters. *Pattern Recogn. 24*, 12 (Dec. 1991), 1167–1186. 7

[Jul81] JULESZ B.: Textons, the elements of texture perception, and their interactions. *Nature 290*, 5802 (Mar. 1981), 91–97. 3

[KWT88] KASS M., WITKIN A., TERZOPOULOS D.: Snakes: Active contour models. *IJCV 1*, 4 (1988), 321–331. 6

[LM16] LIN T.-Y., MAJI S.: Visualizing and understanding deep texture representations. In *CVPR* (June 2016). 3

[LRL08] LEVIN A., RAV-ACHA A., LISCHINSKI D.: Spectral matting. *TPAMI 30*, 10 (2008), 1699–1712. 7, 10

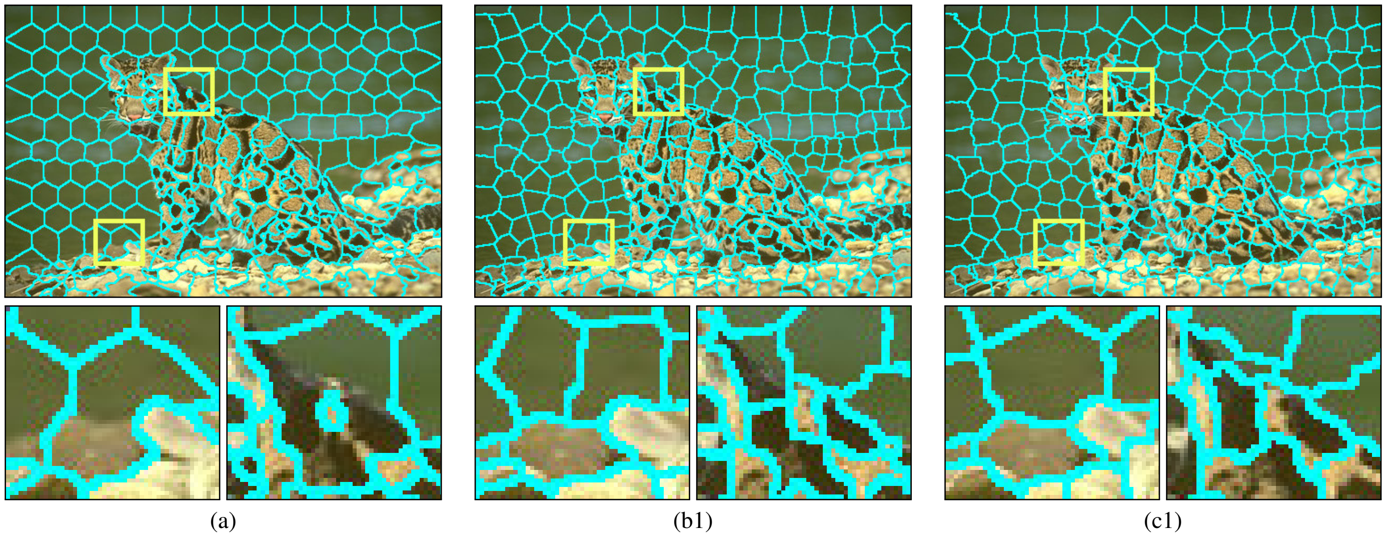
[LSA*16] LOCKERMAN Y. D., SAUVAGE B., ALLÈGRE R., DISCHLER J., DORSEY J., RUSHMEIER H.: Multi-scale label-map extraction for texture synthesis. *ACM Trans. Graph.* (07/2016 2016). 3

[LSD15] LONG J., SHELHAMER E., DARRELL T.: Fully convolutional networks for semantic segmentation. In *CVPR* (June 2015). 3, 4, 5, 6, 7

[MFM04] MARTIN D. R., FOWLKES C., MALIK J.: Learning to detect natural image boundaries using local brightness, color, and texture cues. *TPAMI 26*, 5 (2004), 530–549. 3

[MYZ13] MIKOLOV T., YIH W., ZWEIG G.: Linguistic regularities in continuous space word representations. In *Human Language Technologies* (2013), pp. 746–751. 2

[NSF12] NATHAN SILBERMAN DEREK HOIEM P. K., FERGUS R.:



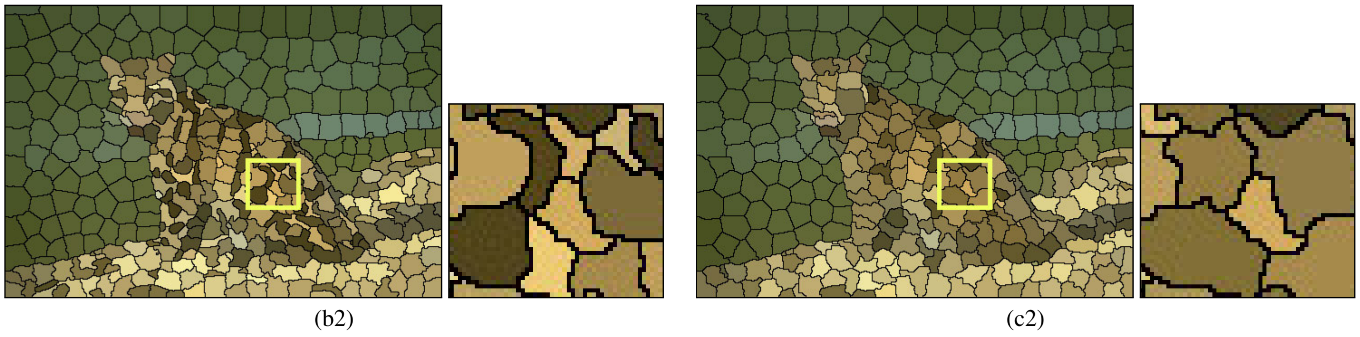


Figure 11: *Texture Aware Superpixels. We compare (a) SLIC [ASS* 12], (b1) a SLIC variant guided by edge detection [DZ13] and (c1) our method. Both (b1) and (c1) follow meaningful edges better than vanilla SLIC (see, e.g., zoomed regions). However, our method does* not *follow intra-texture edges, as can be seen when plotting the average super-pixel color in (b2) and (c2) — we get much smoother colors in the fur region. See text for more details.*

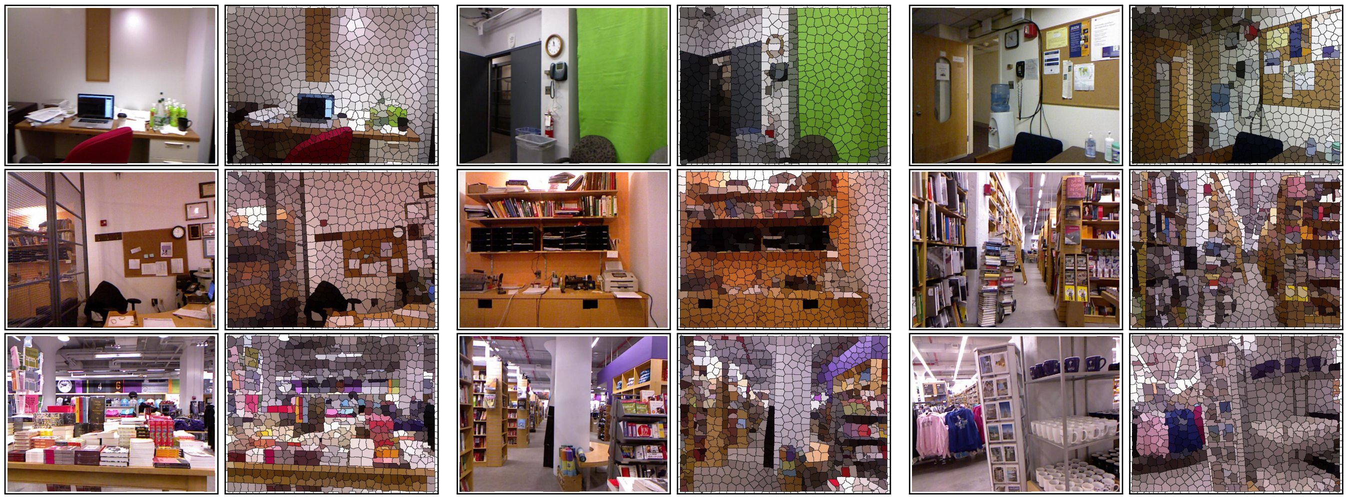


Figure 12: *Super-pixel creation on the NYU Depth Dataset [NSF12]. We produce useful super-pixels, despite the fact that the dataset statistics are different from BSDS (on which we trained).*

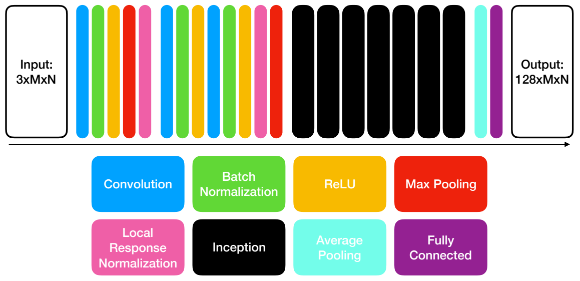


Figure 13: *Patch2Vec neural network architecture. Note that inception blocks (in black) contain many internal elements, see Szegedy et al. [SLJ*15] for details.*

Indoor segmentation and support inference from rgbd images. In *ECCV* (2012). 9, 11

[Ots79] OTSU N.: A threshold selection method from gray-level histograms. *IEEE Tran. on Sys., Man, and Cyb. 9*, 1 (1979), 62–66. 5

[RH99] RANDEN T., HUSOY J. H.: Filtering for texture classification: a comparative study. *TPAMI 21*, 4 (Apr 1999), 291–310. 3

[RKB04] ROTHER C., KOLMOGOROV V., BLAKE A.: "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph. 23*, 3 (2004), 309–314. 3, 7, 10

[SKP15] SCHROFF F., KALENICHENKO D., PHILBIN J.: Facenet: A unified embedding for face recognition and clustering. In *CVPR* (June 2015), pp. 815–823. 3, 4, 12

[SLJ*15] SZEGEDY C., LIU W., JIA Y., SERMANET P., REED S., ANGUELOV D., ERHAN D., VANHOUCKE V., RABINOVICH A.: Going deeper with convolutions. In *CVPR* (June 2015), pp. 1–9. 12

[SSTF*15] SIMO-SERRA E., TRULLS E., FERRAZ L., KOKKINOS I., FUA P., MORENO-NOGUER F.: Discriminative learning of deep convolutional feature point descriptors. In *ICCV* (2015), pp. 118–126. 2

[TD16] TASSE F. P., DODGSON N.: Shape2vec: Semantic-based descriptors for 3d shapes, sketches and images. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 208:1–208:12. 3

[vL16] ŽBONTAR J., LECUN Y.: Stereo matching by training a convolutional neural network to compare image patches. *J. Mach. Learn. Res. 17*, 1 (Jan. 2016), 2287–2318. 2

[VZ03] VARMA M., ZISSERMAN A.: Texture classification: Are filter banks necessary? In *CVPR* (2003), pp. 691–698. 3

[XPC*16] XU N., PRICE B., COHEN S., YANG J., HUANG T.: Deep interactive object selection. In *CVPR* (June 2016), pp. 373–381. 3

A. Neural Network Implementation

We present the full neural network specification in Torch syntax. It closely resembles the OpenFace [ALS16] implementation of FaceNet [SKP15], with a few size changes to accommodate our smaller 32×32 and 16×16 patches.

```
local net = nn.Sequential()
if patch_size == 32 then
  net:add(nn.SpatialConvolutionMM(3, 64, 7, 7, 2, 2, 3, 3))
elseif patch_size == 16 then
  net:add(nn.SpatialConvolutionMM(3, 64, 7, 7, 1, 1, 3, 3))
else
  error("invalid patch size")
end
net:add(nn.SpatialBatchNormalization(64))
net:add(nn.ReLU())
net:add(nn.SpatialMaxPooling(3, 3, 1, 1, 1, 1))
net:add(nn.SpatialCrossMapLRN(5, 0.0001, 0.75))
net:add(nn.SpatialConvolutionMM(64, 64, 1, 1))
net:add(nn.SpatialBatchNormalization(64))
net:add(nn.ReLU())
net:add(nn.SpatialConvolutionMM(64, 192, 3, 3, 1, 1, 1))
net:add(nn.SpatialBatchNormalization(192))
net:add(nn.ReLU())
net:add(nn.SpatialCrossMapLRN(5, 0.0001, 0.75))
net:add(nn.SpatialMaxPooling(3, 3, 1, 1, 1, 1))
net:add(nn.Inception{
  inputSize = 192,
  kernelSize = {3, 5},
  kernelStride = {1, 1},
  outputSize = {128, 32},
  reduceSize = {96, 16, 32, 64},
  pool = nn.SpatialMaxPooling(3, 3, 1, 1, 1, 1),
  batchNorm = true
})
net:add(nn.Inception{
  inputSize = 256,
  kernelSize = {3, 5},
  kernelStride = {1, 1},
  outputSize = {128, 64},
  reduceSize = {96, 32, 64, 64},
  pool = nn.SpatialLPPooling(256, 2, 3, 3, 1, 1),
  batchNorm = true
})
net:add(nn.Inception{
  inputSize = 320,
  kernelSize = {3, 5},
  kernelStride = {2, 2},
  outputSize = {256, 64},
  reduceSize = {128, 32, nil, nil},
  pool = nn.SpatialMaxPooling(3, 3, 2, 2, 1, 1),
  batchNorm = true
})
net:add(nn.Inception{
  inputSize = 640,
  kernelSize = {3, 5},
  kernelStride = {1, 1},
  outputSize = {192, 64},
  reduceSize = {96, 32, 128, 256},
  pool = nn.SpatialLPPooling(640, 2, 3, 3, 1, 1),
  batchNorm = true
})
net:add(nn.Inception{
  inputSize = 640,
  kernelSize = {3, 5},
  kernelStride = {2, 2},
  outputSize = {256, 128},
  reduceSize = {160, 64, nil, nil},
  pool = nn.SpatialMaxPooling(3, 3, 2, 2, 1, 1),
  batchNorm = true
})
net:add(nn.Inception{
  inputSize = 1024,
  kernelSize = {3},
  kernelStride = {1},
  outputSize = {384},
  reduceSize = {96, 96, 256},
  pool = nn.SpatialLPPooling(960, 2, 3, 3, 1, 1),
  batchNorm = true
})
net:add(nn.Inception{
  inputSize = 736,
  kernelSize = {3},
  kernelStride = {1},
  outputSize = {384},
  reduceSize = {96, 96, 256},
  pool = nn.SpatialMaxPooling(3, 3, 1, 1, 1, 1),
  batchNorm = true
})
net:add(nn.SpatialAveragePooling(3, 3, 2, 2))
net:add(nn.View(736))
net:add(nn.Linear(736, opt.embSize))
net:add(nn.Normalize(2))
```