PHOTO MANIPULATION, THE EASY WAY

OHAD FRIED

A DISSERTATION

PRESENTED TO THE FACULTY

OF PRINCETON UNIVERSITY

IN CANDIDACY FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE

BY THE DEPARTMENT OF

COMPUTER SCIENCE

ADVISER: PROFESSOR ADAM FINKELSTEIN

JUNE 2017

Abstract

The typical smartphone user has many thousands of photos in their personal collection. Photo acquisition is effortless, and the next challenge is in devising methods to easily edit such large collections. Specifically, we need manipulation algorithms that are powerful enough for experts, yet simple for novices to master.

We identified three key directions to empower novice users with expert-level editing capabilities while maintaining an overall simplicity in the process. Those directions are (1) better selection masks, (2) high-level goal-centric algorithms and (3) domain specific algorithms. In this thesis we give examples from each category.

Given a photo, a novice user will typically either not edit it at all, or apply a simple global operation such as exposure correction. In contrast, a professional photo editor might perform local edits, specifying a selection mask to limit the operation to specific photo regions, or combining regions from several photos into a single composition. To ease **selection mask creation** we present a new patch embedding technique that allows for single-click selection masks.

Novices often think in terms of goals (e.g. improve lighting, de-clutter photo) and less in technical terms such as color spaces and image layers. One example of a **high-level goal** is the removal of distracting elements from photos. The task is motivated by the way professional photographers operate. They carefully frame the scene and might move objects around in order to stage the perfect photo. We define "photo distractors" as the elements that, if removed, would improve the photo. Using a simple slider interaction we allow users to automatically remove such distractors from photos.

It is at times useful to **tailor solutions to specific photo types**. As an example we show that, specifically for human heads, simple controls can induce sophisticated edits. Given a single portrait photo as input, we can change the pose of the head and the camera distance. This allows users to correct the "selfie effect", i.e. big noses and small ears or to transform distant photos into selfies.

We conclude by discussing how each of these directions can be further explored to enable better image editing tools for novices.

Acknowledgements

I would like to thank my family and especially my wife for understanding and accepting the weird hours and exhausting deadlines. I would like to thank Adam, my adviser, who is the best adviser anyone could wish for. I will forever try to mimic your insightful comments and relaxed attitude. I would like to thank my thesis committee for taking the time to listen to me rambling about my research, and all my collaborators for helping me produce such research. I would be nothing without you. Specifically, I would like to acknowledge Daniel Cohen-Or and Shai Avidan for their invaluable contribution to Chapter 2, and Eli Shechtman and Dan Goldman for their invaluable contribution to Chapters 3 and 4. Lastly, I would like to thank all the amazing friends and colleagues in Princeton and especially in the graphics group. You are a fantastic group of people and you played a big part in making the past five years of my life a pleasurable adventure.

This research was funded partly by a Google PhD Fellowship and a generous gift from Adobe.

To N.F.

Contents

Abstract iii

Acknowledgements v

List of Tables x

List of Figures xi

1 Introduction **1**

2 Texture-aware Selection Masks **6**

2.1 Related Work 9

2.2 Patch Embedding 11

2.3 Evaluation 13

2.3.1 Quantitative Evaluation 13

2.3.2 Single-Click Segment Selection 16

2.3.3 Extensions 20

2.4 Discussion 25

3 Detection And Removal Of Distracting Photo Elements **26**

3.1 Related Work 28

3.2 Datasets 30

3.2.1 Mechanical Turk Dataset (MTurk) 30

3.2.2 Mobile App Dataset (MApp) 31

3.2.3 Data Analysis 32

3.3 Distractor Prediction . . . 35
3.3.1 Segmentation . . . 35
3.3.2 Features . . . 35
3.3.3 Learning . . . 38
3.3.4 Feature Ranking . . . 38
3.4 Evaluation . . . 39
3.4.1 Inter-Dataset Validation . . . 40
3.5 Applications . . . 41
3.5.1 Distractor Removal . . . 42
3.5.2 Image Retargeting . . . 43
3.6 What's Next For Distractor Prediction? . . . 43

4 Content-specific Photo Editing **48**
4.1 Related Work . . . 51
4.2 Our Method . . . 53
4.2.1 Tensor Model . . . 53
4.2.2 Fiducial Detection . . . 56
4.2.3 Fitting . . . 56
4.2.4 Changing Distance and Pose . . . 60
4.2.5 Warping . . . 60
4.3 Evaluation . . . 61
4.3.1 Pipeline Evaluation . . . 61
4.3.2 Synthetic Heads . . . 62
4.3.3 Real Heads . . . 62
4.3.4 Background Preservation . . . 63
4.3.5 Runtime . . . 64
4.4 Applications . . . 64
4.4.1 Distance Correction . . . 64

4.4.2 Headshot Stereoscopy 65
4.4.3 Other Applications 65
4.5 Limitations and Future Work 66

5 Conclusion **78**

A Code Snippets **80**
A.1 Torch Implementation 80

Bibliography **82**

List of Tables

2.1 Patch embedding evaluation 15

3.1 Distractor prediction: dataset comparison 30

3.2 Distractor prediction: feature selection 39

3.3 Distractor prediction: results 40

3.4 Distractor prediction: feature subset results 41

3.5 Distractor prediction: inter-dataset results 41

List of Figures

2.1 Texture2Vec overview 6

2.2 Visualizing Texture2Vec 14

2.3 Single-click selection results 17

2.4 Single-click stability examples 18

2.5 Single-click selection stability 19

2.6 Masking algorithm comparison 21

2.7 Single-click multi-frame selection 22

2.8 Texture aware superpixels 23

2.9 Texture aware superpixels on NYU Depth Dataset 24

3.1 User annotated distractors 28

3.2 Distractor data collection interfaces 32

3.3 Average distractor annotation 33

3.4 Distractor types 34

3.5 Algorithm stages for distractor removal 36

3.6 User interface for distractor removal 42

3.7 Distractor removal results 44

3.8 Distractor-aware image retargeting 46

3.9 Distractor removal failure modes 47

4.1 Overview of portrait manipulation results 49

4.2 Effects of camera distance 54
4.3 Head fitting procedure 55
4.4 Generating warp fields for portrait manipulation 68
4.5 Comparison to a single-mesh model 69
4.6 Fiducial point importance 69
4.7 Portrait warp comparison 69
4.8 Ground truth evaluation of portrait manipulation 70
4.9 Portrait manipulation numeric comparison 71
4.10 Portrait manipulation visual comparison 72
4.11 Fixing/generating selfies 73
4.12 In-the-wild selfie correction 74
4.13 Manipulating distances for expressive faces 75
4.14 3D anaglyphs from a single portrait photo 76
4.15 Interactive portrait editing 77

Chapter 1

Introduction

"Consequently, in the chain of reactions accompanying the creative act, a link is missing. This gap which represents the inability of the artist to express fully his intention, this difference between what he intended to realize and did realize, is the personal 'art coefficient', contained in the work."

— Marcel Duchamp, *The Creative Act (1957)*

The earliest known surviving photograph made in a camera, called View from the Window at Le Gras, was taken by Joseph Nicéphore Niépce in 1826 or 1827 [77]. Niépce created it by exposing a chemically coated pewter plate for over eight hours. It is not much to look at. Over the intervening decades, photo quality has improved, while exposure time decreased and cameras became portable. Photography was no longer just a technical wonder, it was becoming an expressive tool and an art form, with its own lingo, techniques and community. Numerous photography books teach the photographer what to do *when capturing the photo* in order to improve the results [2, 13, 25, 29]. In the days of film photography, post-processing of photos was much less common and the scene inscribed on film was often the final product.

One of the first digital images was scanned by Russell Kirsch in 1957 at the National Bureau of Standards. The first digital camera was created by Steven Sasson in 1975 while working for Eastman Kodak [63]. Today there are more than 5 billion cameras in the world,

most of which are camera phones. Moving from film to pixels opened up new possibilities for digital manipulation of photos. Computational photography [64, 84] uses computation instead of (or alongside) traditional optics to create better photos or to produce photos which are unattainable under real-world physical constraints. Computational photography methods such as high dynamic range imaging [85], gigapixel mosaicing [59], light field photography [76], coded apertures [60], image re-targeting [8], recoloring [26] and image-based rendering [70] expand the expressive range of photos. With the prevalence of digital cameras and as smartphones couple cameras with substantial local processing, computational techniques are playing an increasingly important role in photography.

The typical smartphone user has tens or even hundreds of thousands of photos in their *personal* photo collection. Given all those photos, most people seldom give their photos a second glance after acquiring them. We have reached a point where photo acquisition is trivial, and the next challenge is in devising methods to easily edit large photo collections. Specifically, we need manipulation algorithms which are powerful enough for experts, yet simple for novices to master.

Given a photo, a novice user will typically either not edit it at all, or apply a simple global operation such as exposure correction. Such corrections can either be fully automatic ("auto-enhance" features are common in modern photo editing suites) or require the user to move a slider or two. In contrast, a professional photo editor will also perform *local* edits, specifying a selection mask to limit the operation to specific photo regions, or combining regions from several photos into a single composition. If we want to push novices towards sophisticated edits, the first step is to supply them with a simple selection mechanism. In Chapter 2 we describe a new patch embedding technique which allows single-click mask selection. The user clicks on a single pixel within a region they are interested in, and the full region is automatically selected. The selection is texture-aware, meaning that even for highly textured regions such as a plaid shirt, the user can click a single pixel on the shirt and it will be selected, despite large differences in color values and shading.

Another strategy for empowering novices with sophisticated photo editing capabilities is to fully automate high level goals. Automatic enhancement features in tools such as Apple Photos or Google Photos are used due to the simplicity of a single-click operation. However, they are mostly limited to the same global editing operations that fix exposure and color, or add an interesting photo filter. We would like to keep the simple single-click or single-slider interaction, but use it to achieve higher level goals. In Chapter 3 we give one example of such a goal — removing distracting elements from photos. The task is motivated by the way professional photographers operate. They usually carefully control the scene, either by using a studio or by a deliberate selection of the photo's backdrop, viewing angle and frustum. In many cases a professional will move objects around in order to stage the perfect scene for a photo. We define "photo distractors" as the elements that, if removed, would improve the photo. Using a simple slider interaction we allow users to automatically remove such distractors from photos.

In order to achieve both simplicity and sophistication of results, it is at times useful to tailor solutions to a specific category of photos. For example, in Chapter 4 we show that tailoring a solution for human heads (arguably one of the most important object classes) allows for simple control over sophisticated edits. Given a single portrait photo as input, we allow a user to change the pose of the head and the camera distance. This allows us to correct the "selfie effect", i.e. big noses and small ears or to transform distant photos into selfies.

The bulk of the material in Chapters 3 and 4 were presented publicly prior to the compilation of this thesis [39, 40].

Contributions

This thesis presents different strategies to empower novice users and help them achieve professional-level photo editing skill via interaction modalities which are simple to understand and easy to master. Here we explicitly state the contributions according to chapter order:

Chapter 1: Introduction

- Defining three directions that can lead to better photo editing tools for novices.

Chapter 2: Texture-aware Selection Masks

- A new neural network architecture and training regime for image patch embedding.
- Single-click and texture-aware selection of image regions.
- Texture-aware super-pixel creation algorithm.

Chapter 3: Detection And Removal Of Distracting Photo Elements

- Defining a new task called distractor prediction.
- Collecting a large-scale database with annotations of distractors.
- Training a prediction model that can produce distractor maps for arbitrary images.
- Using our prediction model to automatically remove distractors from images.

Chapter 4: Content-specific Photo Editing

- The ability to edit perceived camera distance in portraits.
- A robust head fitting method that estimates camera distance.
- A new image warping approach that approximates changes in head or camera pose.
- A method to create stereo pairs from an input portrait.

- Evaluation of our approach using an existing dataset and a new dataset captured for this purpose.

As a whole, the goal of this work is to empower novice photo editors and allow people from all skill levels to make the most of their photos. We hope to inspire further exploration into the democratization of sophisticated photo editing techniques.

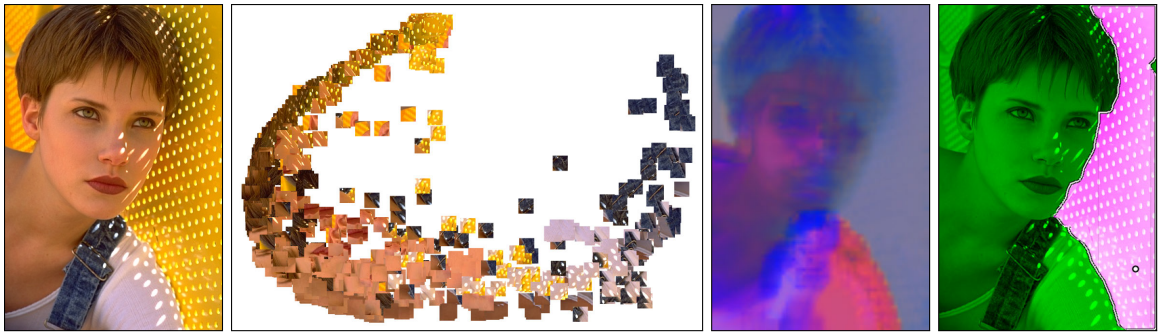
Chapter 2

Texture-aware Selection Masks

"For me the future of the image is going to be in electronic form. You will see perfectly beautiful images on an electronic screen. And I'd say that would be very handsome. They would be almost as close as the best reproductions."

— Ansel Adams, *Dialogue with Photography (1979)*

Texture representation is a central theme in the analysis and synthesis of images. It is quite simple to segment an image made of piece-wise constant colors. It is much more challenging to do so when the image consists of textured regions. Likewise, it is easy to



(a) (b) (c) (d)

Figure 2.1: (a) The input image. (b) A 2D embedding of patches in the image using *Texture2Vec*. Observe how patches with similar texture are clustered together. (c) Projecting the texture code to 3D for visaulization purposes. (d) Single Click Segmentation. Given a single click (the white circle on the fence) we automatically segment the entire fence.

fill a region with a constant color. It is much more challenging to fill a region with a given texture.

Texture can be represented in a number of different ways. Early attempts represent texture as a response to a filter bank. Similar texture should have similar response. Alternatively, texture can be represented as a histogram or a Gaussian mixture model. Today it is common to represent texture as patches of raw pixel values. Analyzing or synthesizing texture amounts to working in patch space. This non-parametric representation leads to impressive results in applications such as texture synthesis or image denoising.

The goal of the different representations is to map texture to some vector space where it is easy to compute distances between textures. The distance measure between similar textures 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 distance 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 *Texture2Vec*. However, there is a significant difference between words and textures. Words have a fairly well defined dictionary. In texture, on the other hand, there is no such dictionary and the number of different textures is not well defined. Moreover, with texture we map never-seen-before texture to the vector space.

Consider Figure 2.1. 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 2.1(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 2.1(b)). Observe how patches of similar texture are clustered together, regardless of texture shifts and illumination changes. In

Figure 2.1(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 Texture2Vec 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 Texture2Vec 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. Texture2Vec, 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 Texture2Vec. 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 texture and another one from a different texture. During training, the CNN learns to map patches of the same texture to nearby points in the Euclidean embedding space, while mapping the patch of the other texture as far away as possible.

Once trained, patches are mapped to vector representation in a Euclidean space. Measuring perceptual similarity between textures now amounts to measuring the Euclidean distance between their corresponding embedded vectors. We evaluate Texture2Vec 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 segmenta-

tion, as well as a super-pixel application. These extensions indicate the potential power of Texture2Vec.

2.1 Related Work

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

Texture Textons are an early representation of texture [52]. They encode second order statistics of small patches. This motivated extensive research on the use of filter banks for texture classification. See review in [83]. Among the filters proposed are Gabor filters, wavelets and Discrete Cosine Transform. These filters are fixed and are not learned from the data. Filter response in pyramids was used with great success for texture synthesis [46, 15]. It was later reported that raw pixel values are as informative as filter bank response [100].

Patch based methods are used with great success in various applications such as texture synthesis [34] and image denoising [20].

Interactive image segmentation also rely on texture analysis. For example, GrabCut [86] 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 [11] computes geodesic distance on a probability image that is based on the distance of each pixel to the GMM of the foreground or background.

Representation Learning Word2Vec [71] 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 [103] 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.* [90] learn feature point descriptors using a Siamese network, where 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.

Moving beyond image patches, Schroff *et al.* [89] 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.

Recently, Ponjou Tasse and Dodgson [95] 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.

Image Segmentation We use the Berkley Segmentation Dataset (BSDS500) [69] 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 [86] and Geodesic Matting [11]. 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.

In our application, on the other hand, we use our features and require just a single point click from the user.

Single click interactive segmentation was also proposed by Bagon *et al.* [10] 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.

2.2 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 a large set of 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 [89]).

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:

$$L(p_a, p_p, p_n) = [\|p_a - p_p\|_2^2 - \|p_a - p_n\|_2^2 + m]_+, \tag{2.1}$$

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 $\mathcal{D}$:

$$L = \sum_{(p_a, p_p, p_n) \in \mathcal{D}} L(p_a, p_p, p_n). \tag{2.2}$$

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. [89], 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 prefer using 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:

$$\mathcal{N}(p_a, p_p) = \{p_n \mid \|p_n - p_a\|_2^2 - \|p_p - p_a\|_2^2 < m\}. \tag{2.3}$$

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

For ground-truth labels, we use the Berkeley Segmentation Dataset (BSDS500) [7]. We randomly sample 50,000 32×32 patches from each of the 200 images in the BSDS500 training set. If patches extend beyond image boundaries, we pad the original image, repeating the boundary pixel values.

We use a similar architecture to the one introduced in FaceNet [89] as implemented by the OpenFace project [5], with changes to accommodate the difference in patch sizes.

Appendix A.1 contains the neural network specification in Torch. More importantly, we changed the training regime as explained above.

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.

2.3 Evaluation

We evaluate Texture2Vec on images that were not part of the training set and report both qualitative and quantitative results. Later on we use Texture2Vec 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 Texture2Vec.

A simple way to visualize the embedding is to project the $128D$ texture codes on the leading three principal components. Ideally, regions with the same texture will be mapped to the same pseudo-RGB color. Figure 2.2 demonstrates that on a wide variety of images. In particular, observe how Texture2Vec maps various textures to uniform pseudo-RGB colors.

2.3.1 Quantitative Evaluation

The first experiment we report measures how good is the embedding in determining whether a pair of patches come from 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

$$d(p_1, p_2) = \|f(p_1) - f(p_2)\|_2. \tag{2.4}$$

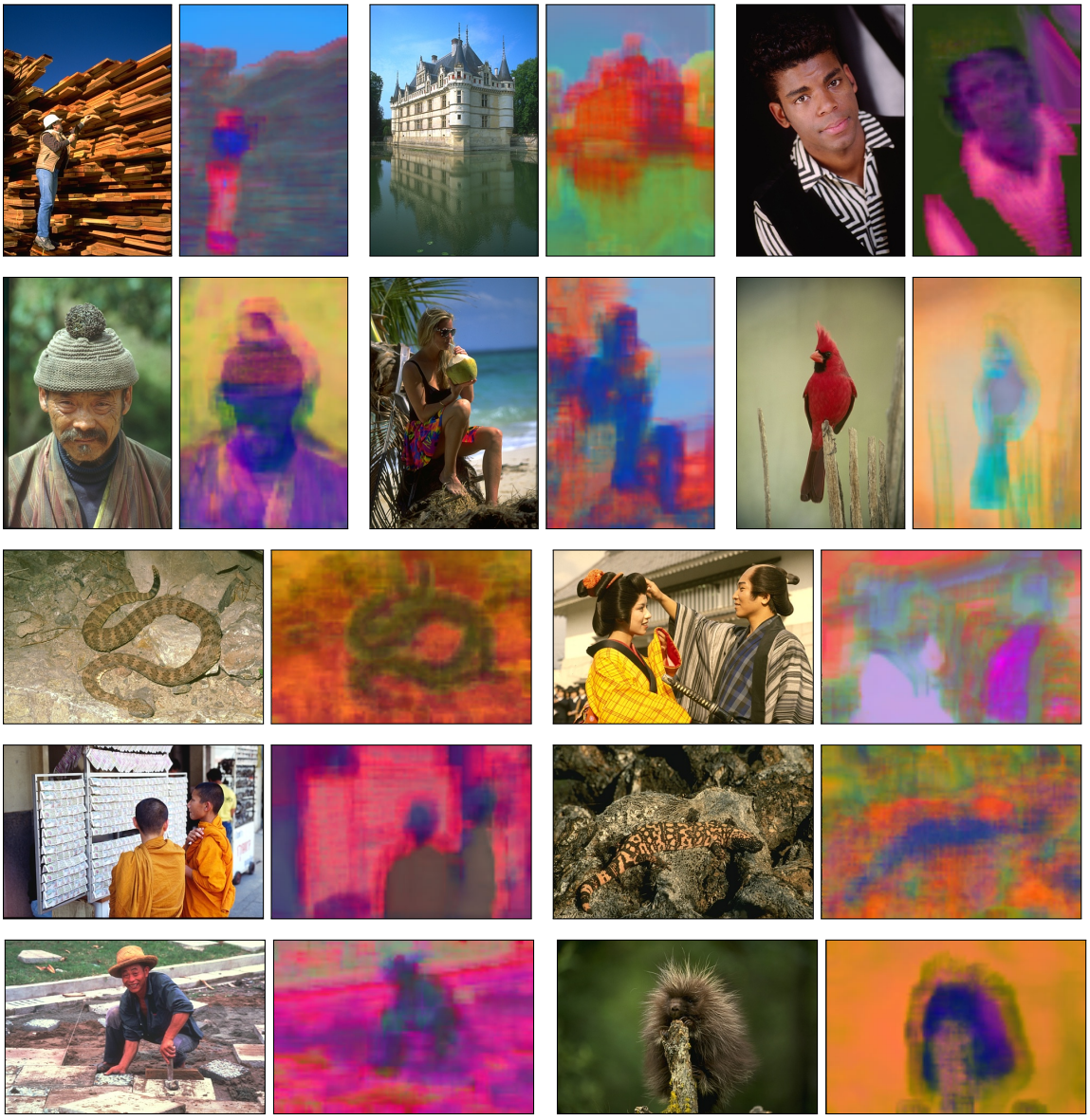


Figure 2.2: Visualizing Texture2Vec. We project the $128D$ embedding vectors on a 3D space and visualize it as pseudo RGB colors. Observe how, for example, the shirt of the person at the top row on the right is mapped to a nearly constant color. We are also able to assign different pseudo colors to the building and its reflection (top row) even though they are very similar in appearance.

Method	Mean AUC
Random	0.50
Gabor	0.66
Raw pixels	0.69
Mean color	0.70
Our method (validation)	0.75
Our method (testing)	0.76
Our method (training)	0.78
Human	0.86

Table 2.1: Same-Not-Same Evaluation: We measure how well we can predict if a pair of patches comes from the same segment or not. The evaluation is made by calculating the distance between the embedding representation of both patches. A number of representations is evaluated. (Higher is better). Our method outperforms the others by a large margin.

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:

$$C(p_1, p_2) = \begin{cases} \text{same} & \text{if } d(p_1, p_2) < t \\ \text{different} & \text{if } d(p_1, p_2) \geq t \end{cases} . \tag{2.5}$$

We use the common receiver operating characteristic curve for all t values and calculate the area under the curve (AUC). Higher AUC implies a better classifier. We apply the above procedure on the training, validation and test sets of the BSDS500 dataset [7]. Note that only the training set was used in Section 2.2 to create our embedding.

Instead of using our patch embedding operator $f(p)$ we can define other operators and repeat the above procedure. In particular, we consider:

- *Raw pixels*: use the RGB values of a given patch.
- *Mean color*: use the average patch color.
- *Gabor*: use the response of a filter bank consisting of multiple Gabor filters at different orientations and scales.

Table 2.1 summarizes all AUC scores. Reasonable values should 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 only 0.70.

2.3.2 Single-Click Segment Selection

We use single-click selection to demonstrate the strength of our embedding. Such a narrow information channel between the user and the algorithm is best suited to investigate the properties of Texture2Vec.

Given a photo $\mathcal{I}$, we calculate patch embeddings $f(p)$ for each 32×32 image patch in a preprocessing step. At runtime, the user clicks a single pixel location that corresponds to patch p_{c}. For all other patches, we calculate the embedding distance

$$d_p := d(p, p_{\mathrm{c}}) = \|f(p) - f(p_{\mathrm{c}})\|_2 \quad \forall p \in \mathcal{I}, \tag{2.6}$$

which yields a per-pixel distance value. Next, we threshold the distances using Otsu's method [80] to produce a binary selection mask. As an optional step, the mask can be refined using snakes [53] 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 the results shown here use $K = 5$. Pre-processing a 320x480 RGB image takes 40 seconds on standard hardware.

Figure 2.3 shows single-click selection results. Importantly, the click locations were randomly selected and not hand picked. Examining Figure 2.3, 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 select it all. Another important property is that training on pixel-accurate masks allows a click near the segment

Figure 2.3: 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.

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 2.4 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 $M_1, ...M_n$ 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

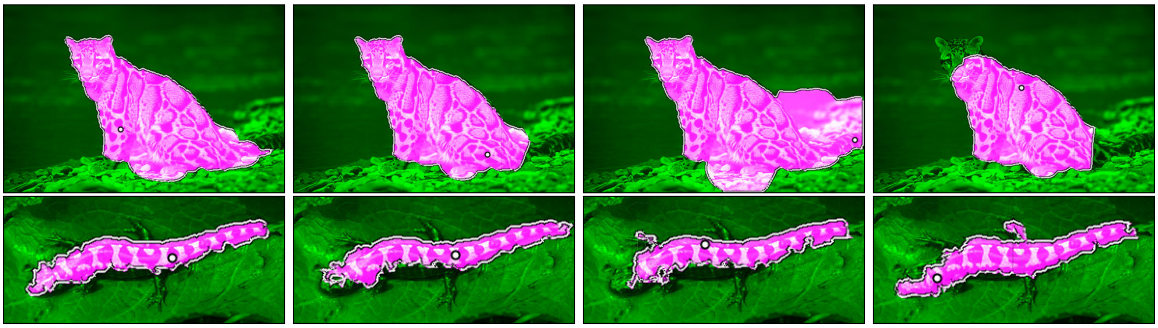


Figure 2.4: Single-click selection stability. Randomly chosen click locations (white dots) for each segment produce similar selection masks. We show results including edge snapping (top) and without snapping (bottom). Notice how, e.g., click locations on the lizard texture include dark and bright spots, yet the selection masks remain stable.

mask. Define:

$$\text{IoU}_i = \frac{\sum_x M(x) M_i(x)}{\sum_x M(x)}, \tag{2.7}$$

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:

$$\text{stability} = \frac{1}{n} \sum_{i=1}^{n} \text{IoU}_i \tag{2.8}$$

If all masks are exactly the same, the score is 1.

Equipped with this stability measure, we conducted 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 computed stability per segment (Equation 2.8) and averaged over 866 image segments. For comparison, we repeated this protocol with Diffusion Map embedding as well [37]. Specifically, for a given image, we compute a 128D embedding using Diffusion Maps and then apply the protocol. Results are reported in Figure 2.5. 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.

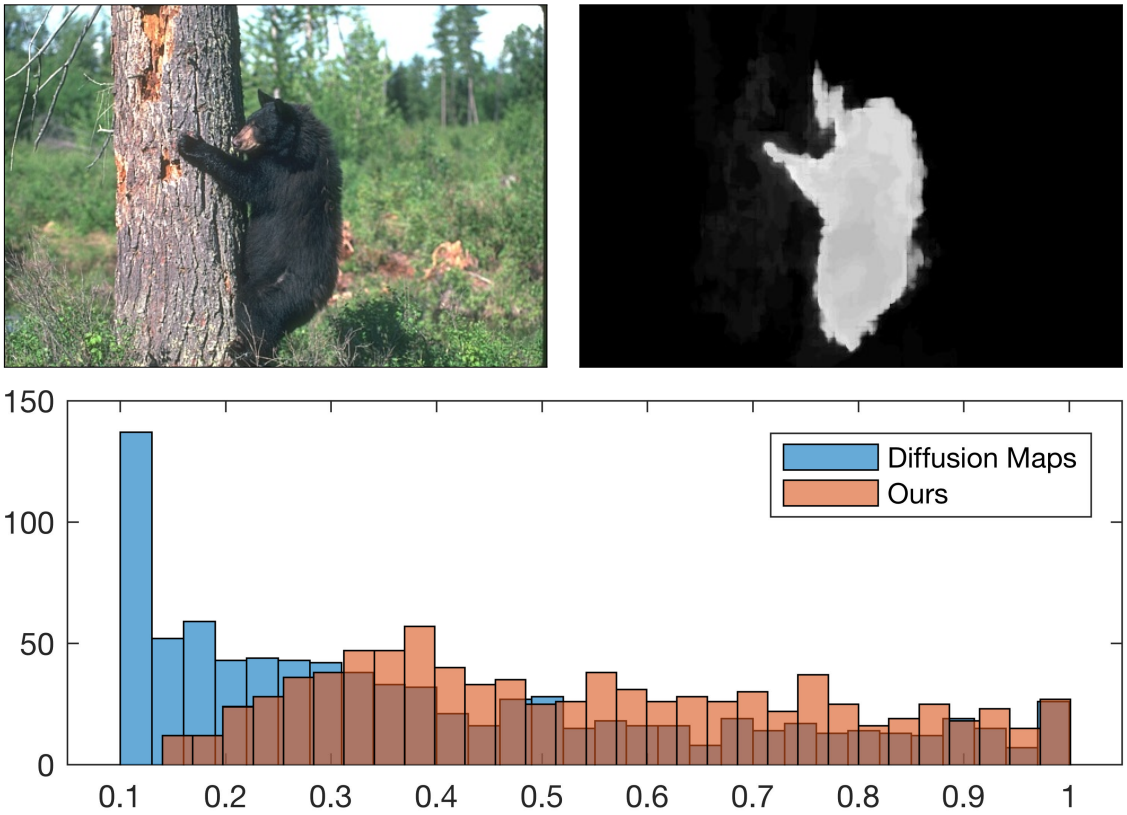


Figure 2.5: Single-Click Selection Stability: (Top) A typical image (left) and its average mask for a particular segment (right). The stability score for this segment is 0.65 (see text for details). (Bottom) Histogram of mask stability scores on a data set of 866 image segments using Texture2Vec and Diffusion Maps. The average stability score of masks generated using our method is 0.54, which is higher than the 0.40 score obtained by Diffusion Maps.

The average stability score, across the entire set, of Texture2Vec is 0.54, compared to 0.40 for the Diffusion Maps.

Figure 2.6 compares our single-click selection to several other masking methods. A scribble based method [61] 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 [86] also does not produce the expected result, even when given a *tight* bounding box. Classic descriptors such as Gabor filter banks, which histori-

cally were used to describe textures [49] 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).

2.3.3 Extensions

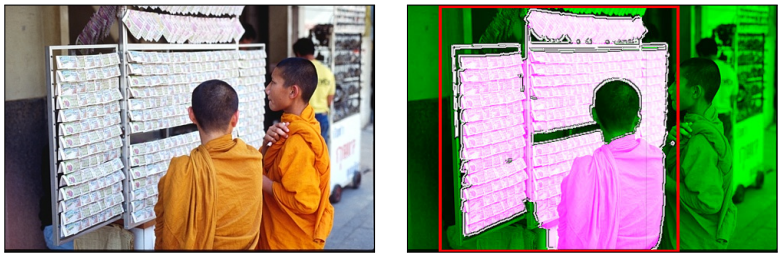
Single-Click Image Segmentation is a core algorithm that can be extended in a number of ways.

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 2.3.2. Results are shown in Figure 2.7. 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* texture representation such as Texture2Vec.

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

We compare Texture2Vec to two popular super-pixels methods. The first is SLIC [1], 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 [33]. Our algorithm is similar to [33] but replaces La*b* values with the first three principle components of Texture2Vec.

Figure 2.8 shows a comparison between the methods. Both Dollár et al. [33] and our method adhere to edges better than SLIC. Our approach is the only one that can distinguish



Input

(a) GrabCut

(b) Gabor

(c) Diffusion (best t)

(d) Spectral (unsupervised)

(e) Spectral (light supervision)

(f) Spectral (supervised)

(g) Ours

Figure 2.6: Masking algorithm comparison. (a) GrabCut [86] does not correctly separate the person. (b) Replacing our patch embedding with a Gabor filter bank response produces a partial segment. (c) Diffusion maps [37] using the best possible time value t. (d) Unsupervised spectral matting [61] does not accurately separate the texture from the rest of the image, 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 rectangle), for (b) (c) and (g) is a *single* pixel location (white circle), and for (e) and (f) is several scribbles.

Figure 2.7: Single-click multi-frame selection. Given an input video (showing frames 0, 33, 50, 81), the user clicks on a single seed point (white dot, top-right 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.

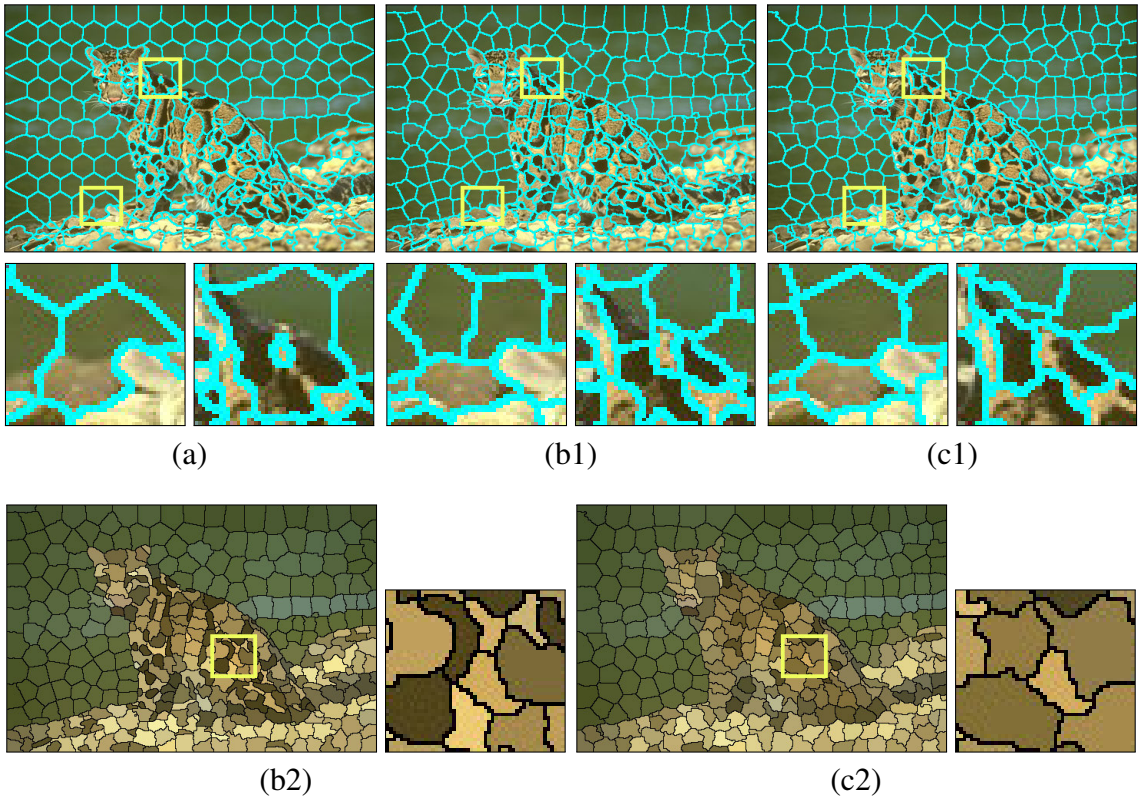


Figure 2.8: Texture Aware Superpixels. We compare (a) SLIC [1], (b1) a SLIC variant guided by edge detection [33] and (c1) our method. Both (b1) and (c1) follow meaningful edges better than vanilla SLIC (see, e.g., zoomed regions). However, our method does *not* follow intra-texture edges, as can be 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.

between inter-segment edges, that separate the target object from the background, 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). This way it will be easier for higher level algorithms, such as image segmentation, to cluster them together.

Figure 2.9 shows results of our super-pixel algorithm on the NYU Depth Dataset [74]. 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.

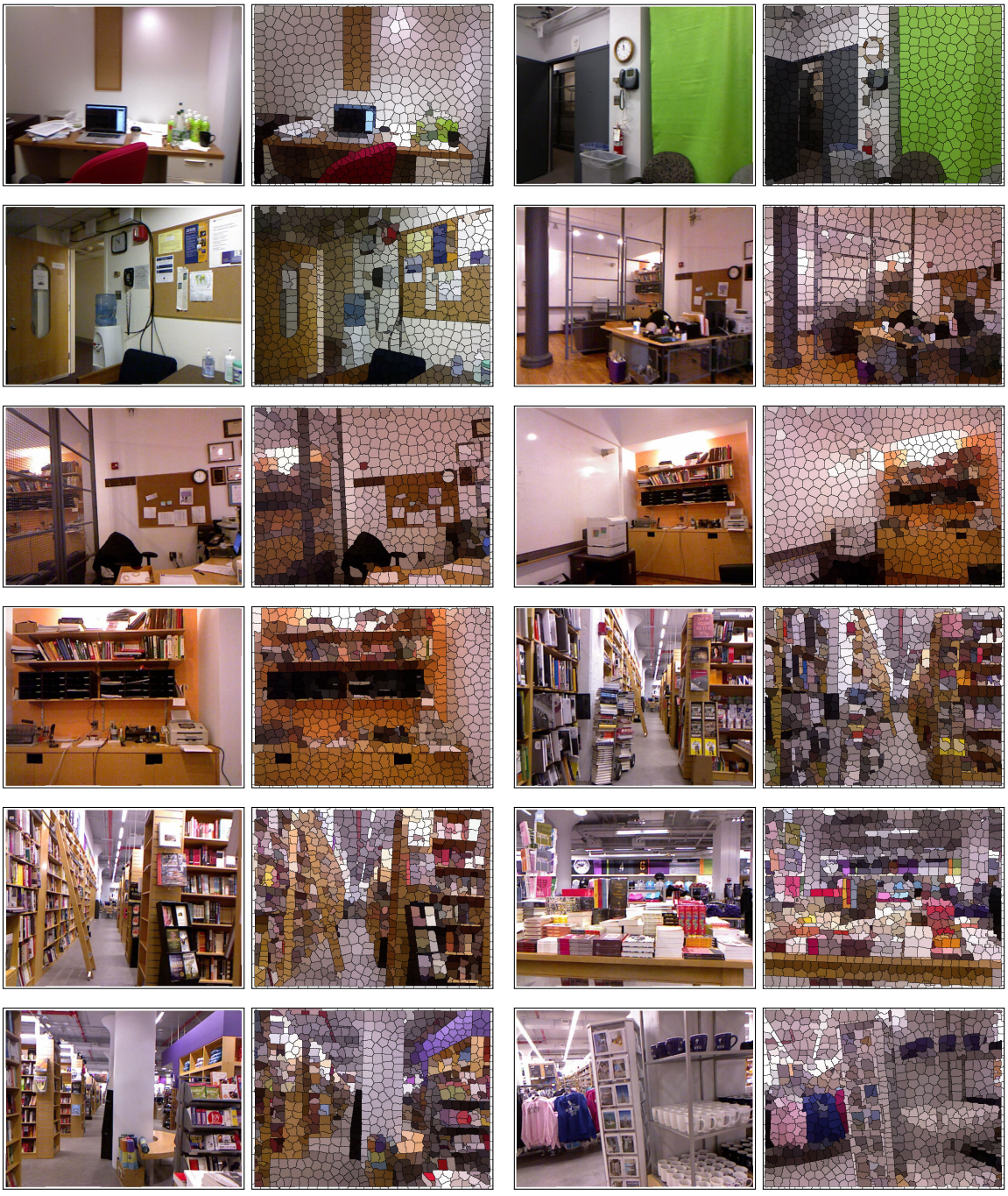


Figure 2.9: Super-pixel creation on the NYU Depth Dataset [74]. We produce useful super-pixels, despite the fact that the dataset statistics are different from BSDS (on which we trained).

2.4 Discussion

Texture2Vec offers a universal embedding of texture 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 some 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 quality of the embedding because it is bounded by the size and quality of the training set. Hopefully, 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 probably be addressed by collecting more data geared towards Texture2Vec.

We believe that Texture2Vec 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 this scenario, 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 Texture2Vec representation for super pixel generation, by replacing pixel RGB values with our embedding vectors.

In the future, we would like to develop tools that will let us 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.

Chapter 3

Detection And Removal Of Distracting Photo Elements

"Photographers deal in things which are continually vanishing and when they have vanished there is no contrivance on earth which can make them come back again.'

— Henri Cartier-Bresson, *The Mind's Eye: Writings on Photography and Photographers (1999)*

Taking pictures is easy, but editing them is not. Professional photographers expend great care and effort to compose aesthetically-pleasing, high-impact imagery. Image editing software like Adobe Photoshop empowers photographers to achieve this impact by manipulating pictures with tremendous control and flexibility – allowing them to carefully post-process good photos and turn them into great photos. However, for most casual photographers this effort is neither possible nor warranted. Last year Facebook reported that people were uploading photos at an average rate of 4,000 images *per second.* The overwhelming majority of these pictures are casual – they effectively chronicle a moment, but without much work on the part of the photographer. Such cases may benefit from semi- or fully-automatic enhancement methods.

Features like "Enhance" in Apple's iPhoto or "Auto Tone" in Photoshop supply one-click image enhancement, but they mainly manipulate *global* properties such as exposure

and tone. Likewise, Instagram allows novices to quickly and easily apply eye-catching filters to their images. Although they have some more localized effects like edge darkening, they apply the same recipe to any image. However, local, image-specific enhancements like removing distracting areas are not handled well by automatic methods. There are many examples of such *distractors* – trash on the ground, the backs of tourists visiting a monument, a car driven partially out of frame, etc. Removing distractors demands a time-consuming editing session in which the user manually selects the target area and then applies features like iPhoto's "Retouch Tool" or Photoshop's "Content Aware Fill" to swap that area with pixels copied from elsewhere in the image.

In this work we take the first steps towards semi-automatic distractor removal from images. The main challenge towards achieving this goal is to automatically identify what types of image regions a person might want to remove, and to detect such regions in arbitrary images. To address this challenge we conduct several studies in which people mark distracting regions in a large collection of images, and then we use this dataset to train a model based on image features.

Our main contributions are: (1) defining a new task called "distractor prediction", (2) collecting a large-scale database with annotations of distractors, (3) training a prediction model that can produce distractor maps for arbitrary images, and (4) using our prediction model to automatically remove distractors from images.

In the following sections we describe related work (Section 3.1), describe and analyze our datasets (Section 3.2), explain our distractor prediction model (Section 3.3), evaluate our predictor (Section 3.4) and present applications of distractor prediction (Section 3.5). With this publication we also make available an annotated dataset containing images with distractors as well as code for both analyzing the dataset and computing distractor maps.

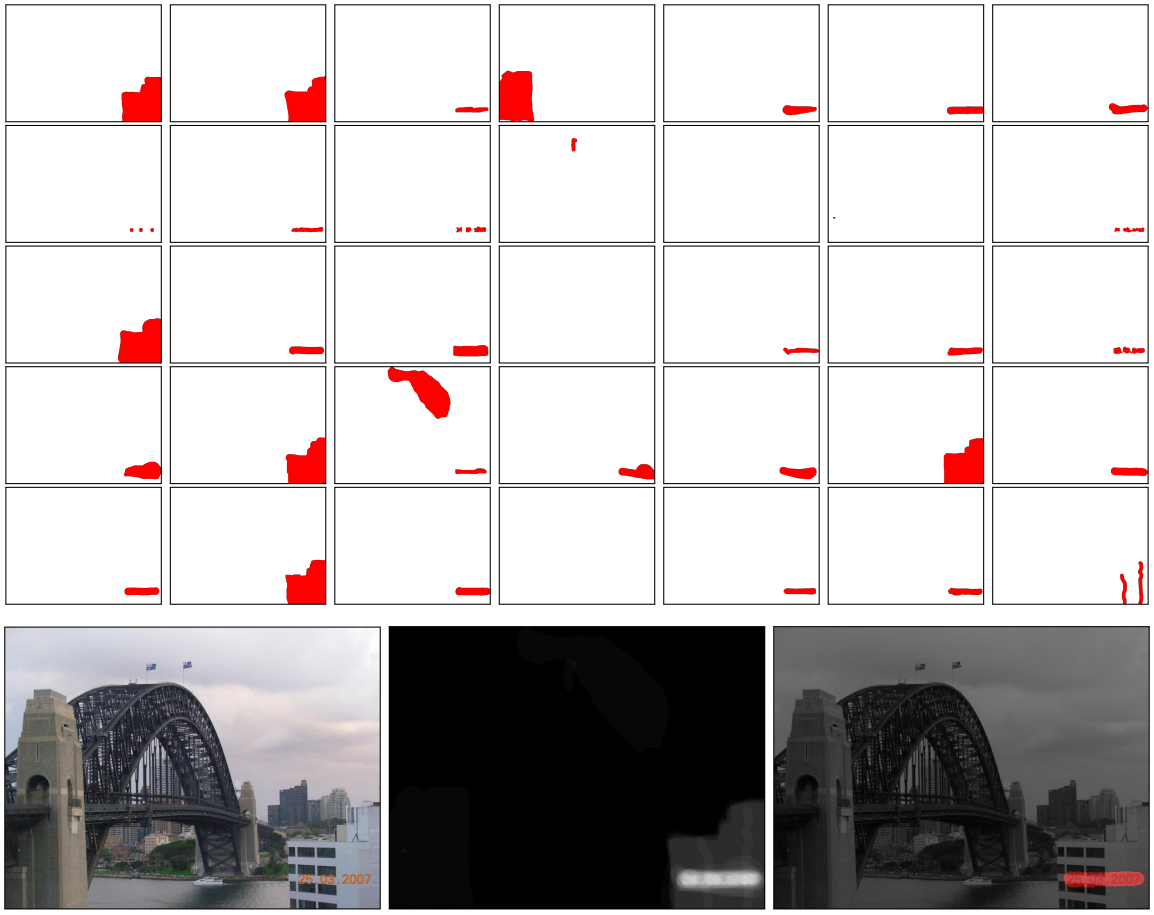


Figure 3.1: User annotations. Top: 35 user annotations for one input image. Bottom, from left to right: input image, average annotation, overlay of thresholded annotation with input image. We collected 11244 such annotations for 1073 images.

3.1 Related Work

A primary characteristic of distractors is that they attract our visual attention, so they are likely to be somewhat correlated with models of visual saliency. Computational saliency methods can be roughly divided into two groups: human fixation detection [48, 42, 51] and salient object detection [28, 27, 68, 66]. Most of these methods used ground-truth gaze data collected in the first 3-5 seconds of viewing (a few get up to 10 seconds) [51]. Although we found some correlation between distractor locations and these early-viewing gaze fixations, it was not high. Our hypothesis is that humans start looking at distractors after longer periods of time, and perhaps only look directly at them when following different viewing

instructions. Existing computational saliency methods are thus insufficient to define visual distractors, because the main subject in a photo where people look first usually has a high saliency value. Moreover, many of these methods (especially in the second category) include components that attenuate the saliency response away from the center of the image or from the highest peaks - exactly in the places we found distractors to be most prevalent.

Another line of related work focuses on automatic image cropping [93, 67, 105]. While cropping can often remove some visual distractors, it might also remove important content. For instance, many methods just try to crop around the most salient object. Advanced cropping methods [105] also attempt to optimize the layout of the image, which might not be desired by the user and is not directly related to detecting distractors. Removing distractors is also related to the visual aesthetics literature [54, 65, 72, 94] as distractors can clutter the composition of an image, or disrupt its lines of symmetry. In particular, aesthetics principles like simplicity [54] are related to our task. However, the computational methods involved in measuring these properties don't directly detect distractors and don't propose ways to remove them.

Image and video enhancement methods have been proposed to detect dirt spots, sparkles [58], line scratches [50] and rain drops [35]. In addition, a plethora of popular commercial tools have been developed for face retouching: These typically offer manual tools for removing or attenuating blemishes, birth marks, wrinkles etc. There have been also a few attempts to automate this process (e.g., [97]) that require face-specific techniques. Another interesting work [92] focused on detecting and de-emphasizing distracting texture regions that might be more salient than the main object. All of the above methods are limited to a certain type of distractor or image content, but in this work we are interested in a more general-purpose solution.

	Mechanical Turk	Mobile App
Number of images	403	376
Annotations per image	27.8 on average	1
User initiated	No	Yes
Image source	Previous datasets	App users

Table 3.1: Dataset comparison.

3.2 Datasets

We created two datasets with complementary properties. The first consists of user annotations gathered via Amazon Mechanical Turk. The second includes real-world use cases gathered via a dedicated mobile app. The Mechanical Turk dataset is freely available, including all annotations, but the second dataset is unavailable to the public due to the app's privacy policy. We use it for cross-database validation of our results (Section 3.4). Table 3.1 and the following subsections describe the two datasets.

3.2.1 Mechanical Turk Dataset (MTurk)

For this dataset we combined several previous datasets from the saliency literature [3, 51, 62] for a total of 1073 images. We created a Mechanical Turk task in which users were shown 10 of these images at random and instructed as follows:

> *For each image consider what regions of the image are disturbing or distracting from the main subject. Please mark the areas you might point out to a professional photo editor to remove or tone-down to improve the image. Some images might not have anything distracting so it is ok to skip them.*

The users were given basic draw and erase tools for image annotation (Figure 3.2). We collected initial data for the entire dataset (average of 7 annotations per image) and used it to select images containing distractors by consensus: An image passed the consensus test if more than half of the distractor annotations agree at one or more pixels in the image.

403 images passed this test and were used in a second experiment. We collected a total of 11244 annotations, averaging 27.8 annotations per image in the consensus set (figure 3.1).

3.2.2 Mobile App Dataset (MApp)

Although the Mechanical Turk dataset is easy to collect, one might argue that it is biased: Because Mechanical Turk workers do not have any particular expectations about image enhancement and cannot see the outcome of the intended distractor removal, their annotations may be inconsistent with those of real users who wish to remove distractors from images. In order to address this, we also created a second dataset with such images: We created a free mobile app (Figure 3.2) that enables users to mark and remove unwanted areas in images. The app uses a patch-based hole filling method [12] to produce a new image with the marked area removed. The user can choose to discard the changes, save or share them. Users can opt-in to share their images for limited research purposes, and over 25% of users chose to do so.

Using this app, we collected over 5,000 images and over 44,000 fill actions (user strokes that mark areas to remove from the image). We then picked only images that were exported, shared or saved by the users to their camera roll (i.e.not discarded), under the assumption that users only save or share images with which they are satisfied.

Users had a variety of reasons for using the app. Many users removed attribution or other text to repurpose images from others found on the internet. Others were simply experimenting with the app (e.g. removing large body parts to comical effect), or clearing large regions of an image for the purpose of image composites and collages. We manually coded the dataset to select only those images with distracting objects removed. Despite their popularity, we also excluded face and skin retouching examples, as these require special tools and our work focused on more general images. After this coding, we used the 376 images with distractors as our dataset for learning.

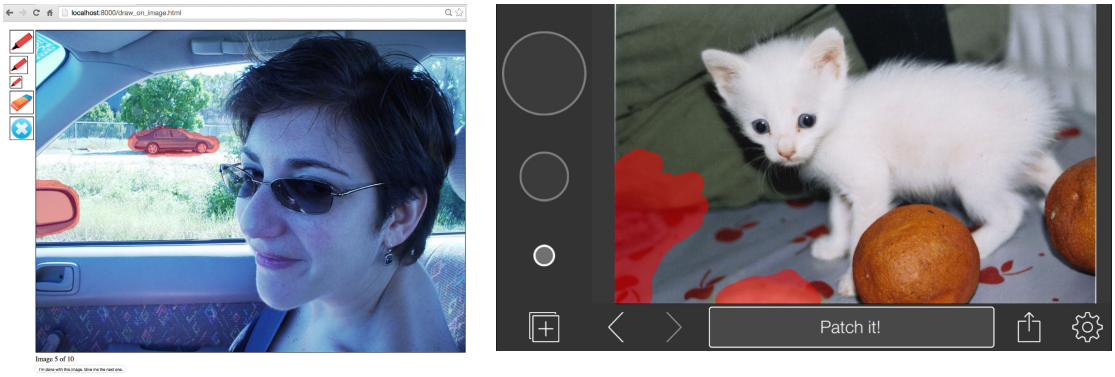


Figure 3.2: Data collection interfaces. Left: MTurk interface with basic tools for marking and removal. Right: Mobile app using inpainting with a variable brush size, zoom level and undo/redos.

3.2.3 Data Analysis

Our datasets afford an opportunity to learn what are the common locations for distractors. Figure 3.3 shows the average of all collected annotations. It is clear that distractors tend to appear near the boundaries of the image, with some bias towards the left and right edges. We use this observation later in Section 3.3.2.

We can also investigate which visual elements are the most common distractors. We created a taxonomy for the following objects that appeared repeatedly as distractors in both datasets: **spot** (dust or dirt on the lens or scene), **highlight** (saturated pixels from light sources or reflections), **face**, **head** (back or side), **person** (body parts other than head or face), **wire** (power or telephone), **pole** (telephone or fence), **line** (straight lines other than wires or poles), **can** (soda or beer), **car**, **crane**, **sign**, **text** (portion, not a complete sign), **camera**, **drawing**, **reflection** (e.g.in images taken through windows), **trash** (garbage, typically on the ground), **trashcan**, **hole** (e.g.in the ground or a wall).

Treating each annotated pixel as a score of 1, we thresholded the average annotation value of each image in the MTurk dataset at the top 5% value, which corresponds to 0.18, and segmented the results using connected components (Figure 3.1). For each connected component we manually coded one or more tags from the list above. The tag **object** was

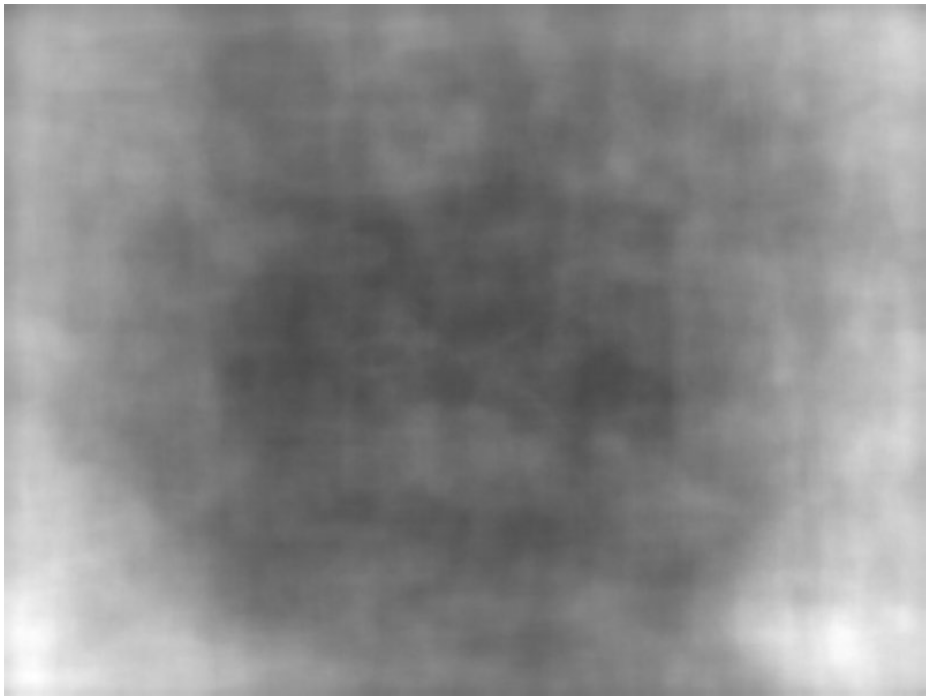


Figure 3.3: An average of all collected annotations. Distractors tend to appear near the image boundary.

used for all objects which are not one of the other categories, whereas **unknown** was used to indicate regions that do not correspond to discrete semantic objects. We also included three optional modifiers for each tag: **boundary** (a region touching the image frame), **partial** (usually an occluded object) and **blurry**. Figure 3.4 shows the histograms of distractor types and modifiers. Notice that several distractor types are quite common. This insight leads to a potential strategy for image distractor removal: training task-specific detectors for the top distractor categories. In this work we chose several features based on the findings of figure 3.4 (e.g.a text detector, a car detector, a person detector, an object proposal method). An interesting direction for future work would be to implement other detectors, such as electric wires and poles.

All distractor annotations for the MTurk dataset are freely available for future research as part of our dataset.

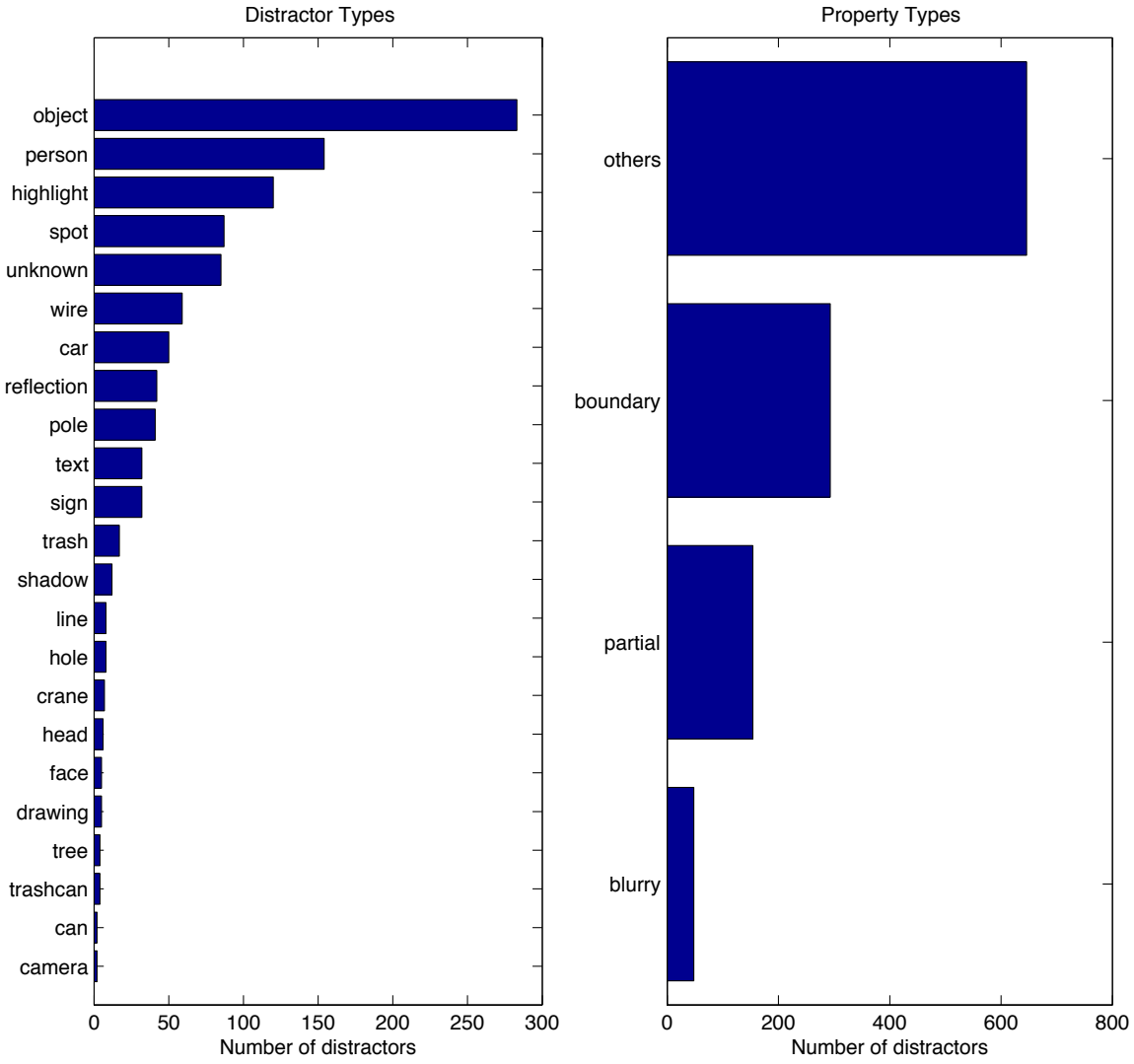


Figure 3.4: Distractor types. Left: histogram of distractor types described in Section 3.2.3. Right: histogram of distractor properties, indicating whether the distractor is close to the image boundary, occluded/cropped or blurry.

3.3 Distractor Prediction

Given input images and user annotations, we can construct a model for learning distractor maps. We first segment each image (section 3.3.1). Next we calculate features for each segment (section 3.3.2). Lastly, we use LASSO [96] to train a mapping from segment features to a distractor score — the average number of distractor annotations per pixel (section 3.3.3). The various stages of our algorithm are shown in figure 3.5.

3.3.1 Segmentation

Per-pixel features can capture the properties of a single pixel or some neighborhood around it, but they usually do not capture region characteristics for large regions. Thus, we first segment the image and later use that segmentation to aggregate measurements across regions. For the segmentation we use multi-scale combinatorial grouping (MCG) [6]. The output of MCG is in the range of $[0, 1]$ and we use a threshold value of 0.4 to create a hard segmentation. This threshold maximizes the mean distractor score per segment over the entire dataset. We found empirically that the maximum of this objective segments distracting objects accurately in many images.

3.3.2 Features

Our datasets and annotations give us clues about the properties of distractors and how to detect them. The distractors are, by definition, salient, but not all salient regions are distractors. Thus, previous features used for saliency prediction are good candidate features for our predictor (but not sufficient). We also detect features that distinguish main subjects from salient distractors that might be less important, such as objects near the image boundary. Also, we found several types of common objects appeared frequently, and included specific detectors for these objects.

We calculate per-pixel and per-region features. The 60 per-pixel features are:

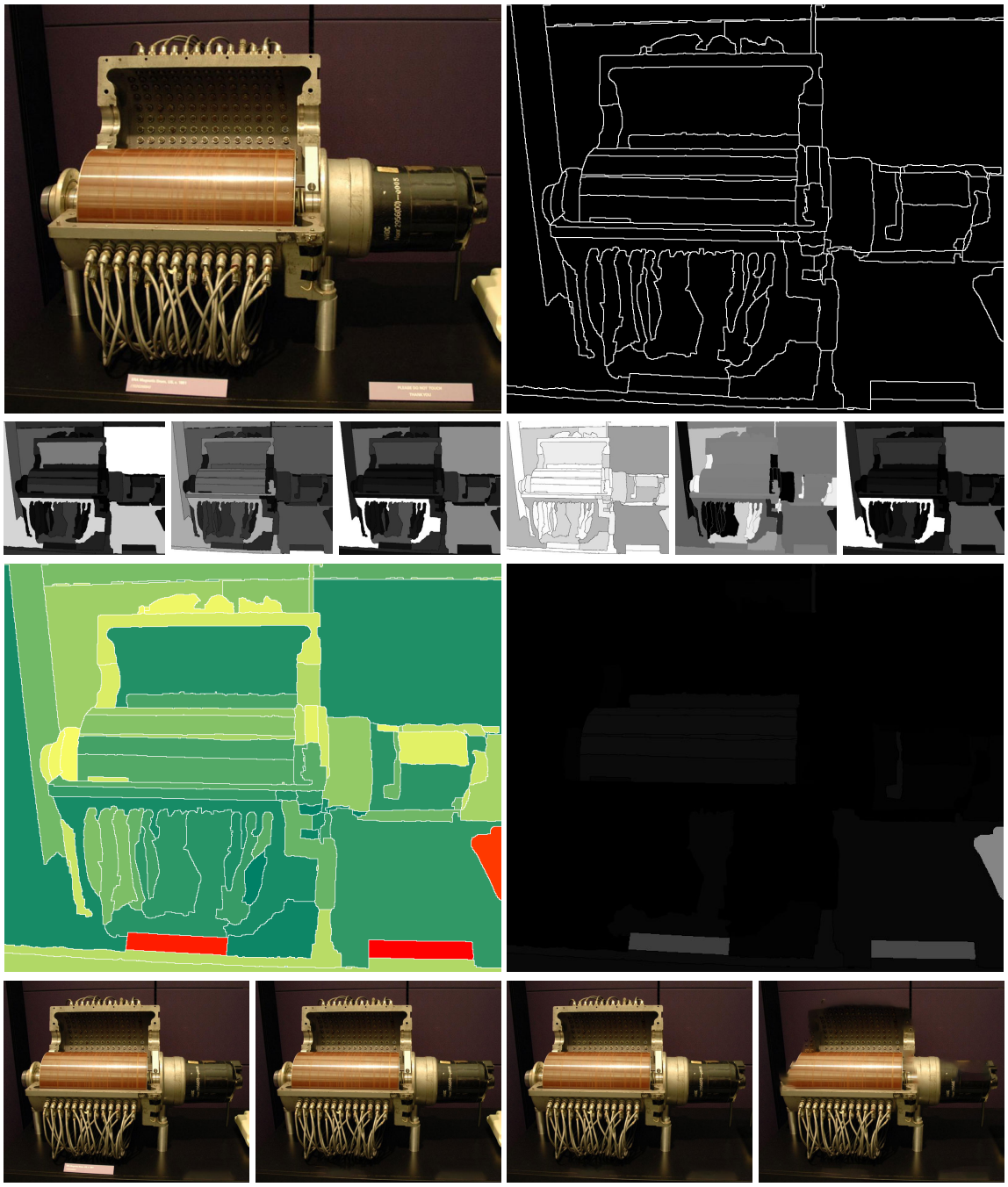


Figure 3.5: Various stages of our algorithm. Top left: original image. Top right: MCG segmentation. 2nd row: examples of 6 of our 192 features. 3rd row left: our prediction, red (top three), yellow (high score) to green (low score). 3rd row right: ground truth. Bottom row: distractor removal results, with threshold values 1, 2, 3, 20 from left to right. The 3 distractors are gradually removed (3 left images), but when the threshold is too high, artifacts start appearing (far right).

- (3) Red, green and blue channels.
- (3) Red, green and blue probabilities (as in [51]).
- (5) Color triplet probabilities, for five median filter window sizes (as in [51]).
- (13) Steerable pyramids [91].
- (5) Detectors: cars [38], people [38], faces [101], text [75], horizon [78, 51].
- (2) Distance to the image center and to the closest image boundary.
- (7)* Saliency prediction methods [47, 48, 66, 68, 78].
- (2)* Object proposals [109]: all objects, top 20 objects. We sum the scores to create a single map.
- (2) Object proposals [109]: all objects contained in the outer 25% of the image. We create 2 maps: one by summing all scores and another by picking the maximal value per pixel.
- (9) All features marked with $*$, attenuated by $\mathcal{F} = 1 - \mathcal{G}/\max(\mathcal{G})$. $\mathcal{G}$ is a Gaussian the size of the image with a standard deviation of $0.7 * \sqrt{d_1 * d_2}$ (d_1 and d_2 are the height and width of the image) and centered at the image center.
- (9) All features marked with $*$, attenuated by $\mathcal{F}$ as defined above, but with the Gaussian centered at the pixel with the maximal feature value.

All the per-pixel features are normalized to have zero mean and unit variance.

To use these per-pixel features as per-segment features, we aggregate them in various ways: For each image segment, we calculate the mean, median and max for each of the 60 pixel features, resulting in 180 pixel-based features per segment.

Lastly, we add a few segment-specific features: area, major and minor axis lengths, eccentricity, orientation, convex area, filled area, Euler number, equivalent diameter, solidity, extent and perimeter (all as defined by the Matlab function `regionprops`). All features are concatenated, creating a vector with 192 values per image segment.

3.3.3 Learning

For each image in our dataset we now have a segmentation and a feature vector per segment. We also have user markings. Given all the markings for a specific image, we calculate the average marking (over all users and all segment pixels) for each segment. The calculated mean is the ground truth distractor score for the segment.

We use Least Absolute Selection and Shrinkage Operator (LASSO) [96] to learn a mapping between segment features and a segment's distractor score. All results in this paper are using LASSO with 3-fold cross validation. Using LASSO allows us to learn the importance of various features and perform feature selection (section 3.3.4).

Besides LASSO, we also tried linear and kernel SVM [36] and random forests with bootstrap aggregating [17]. Kernel SVM and random forests failed to produce good results, possibly due to overfitting on our relatively small dataset. Linear SVM produced results almost as good as LASSO, but we chose LASSO for its ability to rank features and remove unnecessary features. Nonetheless, linear SVM is much faster and may be a viable alternative when training time is critical.

3.3.4 Feature Ranking

Table 3.2 shows the highest scoring features for each of the datasets. We collect the feature weights from all leave-one-out experiments using LASSO. Next we calculate the mean of the absolute value of weights, providing a measure for feature importance. The table shows the features with the largest mean value.

Using our feature ranking we can select a subset of the features, allowing a trade-off between computation and accuracy. Using the label Fk to denote the set of k highest scoring features when trained using all features, Table 3.4 shows results with only F5 and F10 features. Although these results are less accurate than the full model, they can be calculated much faster. (But note that feature sets Fk as defined above are not necessarily the optimal set to use for training with k features.)

MTurk	MApp
Torralba saliency[1†]	Torralba saliency[2†]
RGB Green[3]	Coxel saliency[3]
Torralba saliency[2†]	RGB Probability (W=0)[1]
Itti saliency[1†]	RGB Probability (W=2)[3]
RGB Blue probability[3]	Text detector[1]
RGB Probability (W=4)[1]	RGB Green probability[1]
RGB Probability (W=2)[1]	Boundary object proposals[1]
Object proposals[3‡]	Hou saliency[3‡]
RGB Probability (W=16)[1]	Hou saliency[1†]
Distance to boundary[3]	Steerable Pyramids[2]

[1]mean [2]median [3]max
[†]attenuated with an inverted Gaussian around image center
[‡]attenuated with an inverted Gaussian around maximal value

Table 3.2: Feature selection. For each dataset we list the features with the highest mean feature weight across all experiments.

3.4 Evaluation

For evaluation, we plot a receiver operating characteristic (ROC) curve (true positive rate vs. false positive rate) and calculate the area under the curve (AUC) of the plot. We use a leave-one-out scheme, where all images but one are used for training and the one is used for testing. We repeat the leave-one-out experiment for each of our images, calculating the mean of all AUC values to produce a score.

Table 3.3 summarizes our results. A random baseline achieves a score of 0.5. We achieve an average AUC of 0.81 for the MTurk dataset and 0.84 for the MApp dataset. The LASSO algorithm allows us to learn which of the features are important (section 3.3.4) and we also report results using subsets of our features (Table 3.4). As expected, these results are not as good as the full model, but they are still useful when dealing with space or time constraints (less features directly translate to less memory and less computation time). We also report results for previous saliency methods as well as a simple adaptation to these methods where we multiply the saliency maps by an inverted Gaussian (as described in Sec. 3.3.2). This comparison is rather unfair since saliency methods try to predict all

Method	MTurk AUC	MApp AUC
Random	0.50	0.50
Saliency IV‡ [78]	0.55	0.56
Saliency I [66]	0.57	0.53
Saliency I‡ [66]	0.58	0.59
Saliency II [68]	0.59	0.57
Saliency II‡ [68]	0.59	0.57
Saliency III‡ [47]	0.62	0.59
Saliency III [47]	0.65	0.69
Saliency I† [66]	0.67	0.65
Saliency II† [68]	0.68	0.68
Saliency III† [47]	0.70	0.75
Saliency IV† [78]	0.72	0.72
Saliency IV [78]	0.74	0.76
Our method	**0.81**	**0.84**
Average Human	0.89	-

†attenuated with an inverted Gaussian around image center
‡attenuated with an inverted Gaussian around maximal value

Table 3.3: AUC scores. We compare against saliency prediction methods as published, and the same methods attenuated with an inverted Gaussian, as described in Section 3.3.2. Our method outperforms all others. As an upper bound we report average human score, which takes the average annotation as a predictor (per image, using a leave-one-out scheme). We also compared against individual features from Table 3.2 (not shown): all scores were lower than our method with a mean score of 0.59.

salient regions and not just distractors. However, the low scores for these methods show that indeed distractor prediction requires a new model based on new features.

3.4.1 Inter-Dataset Validation

Using Mechanical Turk has the advantage of allowing us to get a lot of data quickly, but might be biased away from real-world scenarios. We wanted to make sure that our images and annotation procedure indeed match distractor removal "in the wild". For that purpose we also created dataset collected using our mobile app (MApp dataset), which contains real world examples of images with distractors that were actually removed by users. We performed inter-dataset tests: training on one dataset and testing on the other, the results

Method	Average # of used features	MTurk AUC
Ours (F-5 features)	3.40	0.72
Ours (F-10 features)	7.43	0.80
Ours (all features)	28.06	**0.81**

Table 3.4: AUC scores. Results using all 192 features and subsets F5 and F10 described in Section 3.3.4. Column 2 is the mean (over all experiments) of the number of features that were not zeroed out by the LASSO optimization. F10 produces a score similar to the full model, while using 5% of the features.

Train Dataset	Test Dataset	# of features	# of used features	AUC
MTurk	MApp	192	37	0.86
MApp	MTurk	192	25	0.78

Table 3.5: Inter-dataset AUC scores. We train on one dataset and test on the other, in order to validate that our MTurk dataset is similar enough to the real-world use cases in MApp to use for learning.

are summarized in table 3.5. We show good results for training on MTurk and testing on MApp (0.86) and vice versa (0.78). The MApp dataset contains a single annotation per image (vs. 27.8 on average for the MTurk one). We therefore believe that the value 0.78 can be improved as the MApp dataset grows.

3.5 Applications

We propose a few different applications of distractor prediction. The most obvious application is automatic in-painting (Section 3.5.1), but the ability to identify distracting regions of an image can also be applied to down-weight the importance of regions for image retargeting (Section 3.5.2). We also posit that image aesthetics and automatic cropping can benefit from our method (Section 3.6).

3.5.1 Distractor Removal

The goal of distractor removal is to attenuate the distracting qualities of an image, to improve compositional clarity and focus on the main subject. For example, distracting regions can simply be inpainted with surrounding contents. To illustrate this application we created a simple interface with a single slider that allows the user to select a distractor threshold (figure 3.6). All segments are sorted according to their score and the selected threshold determines the number of segments being inpainted (figure 3.5). For a full demo of this system please see our supplementary video. Some before and after examples are shown in figure 3.7. We chose a rank order-based user interface as it is hard to find one threshold

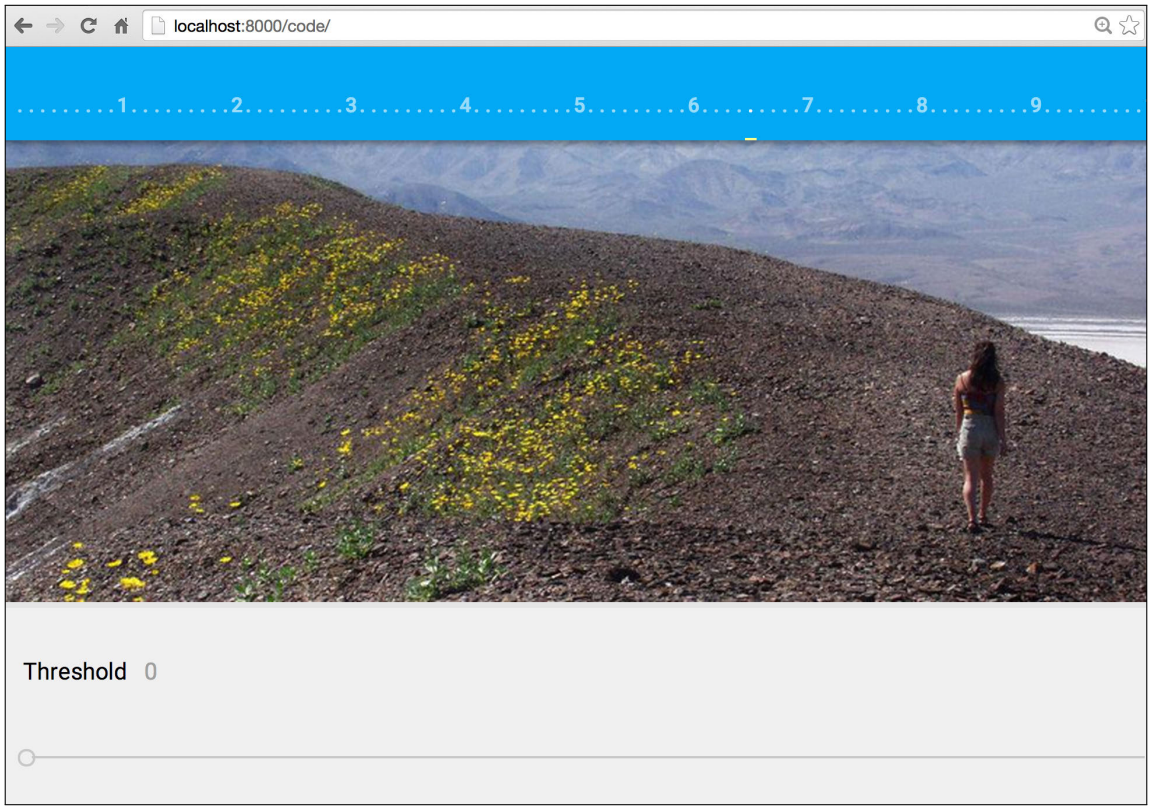


Figure 3.6: User interface for distractor removal. The user selects the amount of distracting segments they wish to remove. We refer the reader to the accompanying video for a more lively demonstration.

that would work well on all images, however we found that if distractors exist in an image they will correspond to the first few segments with the highest score.

3.5.2 Image Retargeting

Image retargeting is the task of changing the dimensions of an image, to be used in a new layout or on a different device. Many such methods have been developed in the past years [9, 87]. In addition to the input image and the desired output size, many of them can take an importance mask, which may be derived (based on image gradients, saliency prediction and gaze data) or provided by the user.

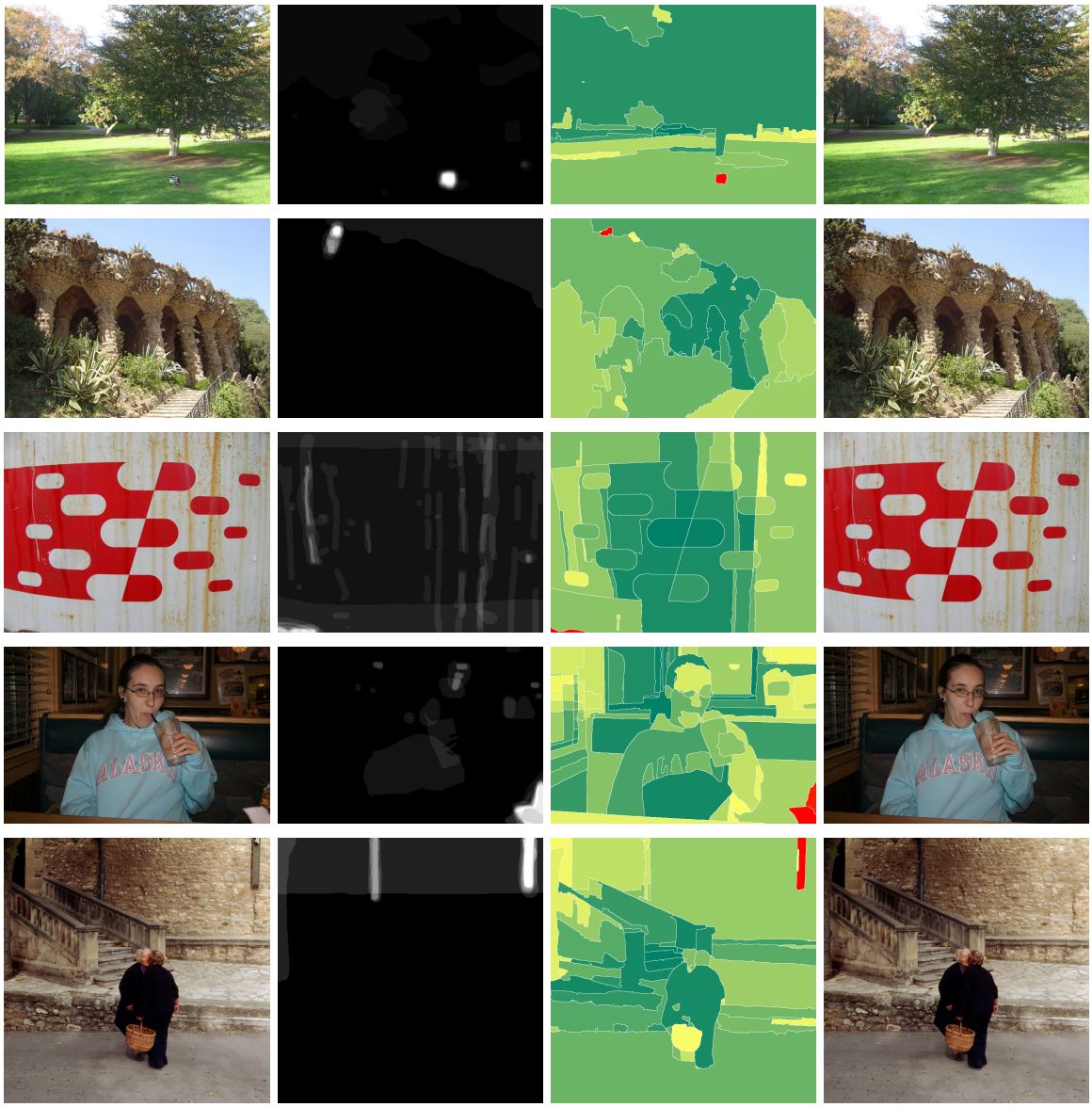
We can thus use our distractor prediction model to enhance a retargeting technique such as seam-carving [9]. For this application we view the distractor map as a complement to a saliency map: Whereas saliency maps give information regarding areas we would like to keep in the output image, a distractor map gives information regarding areas we would like to remove from the output. We thus calculate the gradient magnitudes of the image (G) and our distractor prediction map (D). Next, we invert the map ($D' = 1 - D$) and normalize for zero mean and unit standard deviation ($\hat{G}, \hat{D'}$). Our final map is $\hat{G} + \alpha\hat{D'}$. We use $\alpha = 1$.

Even this simple scheme produces good results in many cases. In figure 3.8, notice how the top-right image does not contain the red distractor and the bottom-right image does not contain the sign on the grass. (See figure 3.7 for the full distractor maps for these images.) However, we believe that a model which combines saliency and distractor maps will produce superior results. The creation of such a model is left for future work.

3.6 What's Next For Distractor Prediction?

We have acquired a dataset of distracting elements in images, used it to train a learning algorithm to predict such regions in novel images, and applied our predictor to a novel

Figure 3.7: Examples of distractor removal results. Each quadruplet shows (from left to right): (1) Original image. (2) Normalized average ground-truth annotation. (3) Order of segments as predicted by our algorithm. (4) Distractor removal result. We urge the reader to zoom in or to look at the full resolution images available as supplementary material. The number of segments to remove was manually selected for each image. Segments are shown on a green-to-yellow scale, green being a lower score. Segment selected for removal are shown on an orange-to-red scale, red being a higher score. Notice how the red segments correlate with the ground-truth annotation. Also notice that we manage to detect a variety of distracting elements (a sign, a person, an abstract distractor in the corner, etc.)



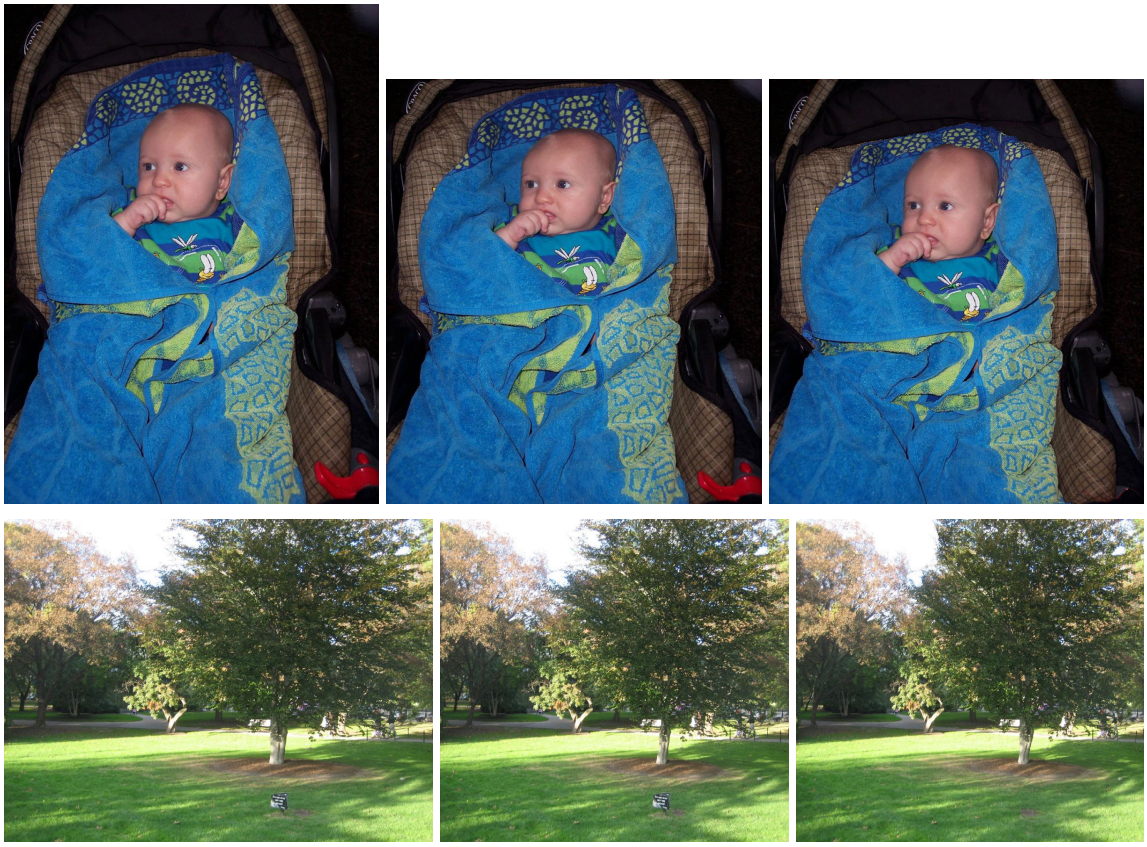


Figure 3.8: Image retargeting using seam carving [9]. From left to right: original image, retargeted image using a standard cost function (gradient magnitude), retargeted image using distractor cost function described in Section 3.5.2.

system that can in-paint distractors, removing them from an image with little or no user input.

Although our system shows great promise, there is plenty of room for improvement. Figure 3.9 illustrates several cases where our approach produces unsatisfactory results: The first two cases on the left illustrate a failure of our learned model. We predict the patch on the jeans of the main subject, and an entire person, even though they are critical parts of the main subject or the composition. The third example shows a segmentation failure, where only part of the arm at the lower right corner is removed. The last shows a removal-method failure, in which the sign behind the right person is correctly detected as distracting, but

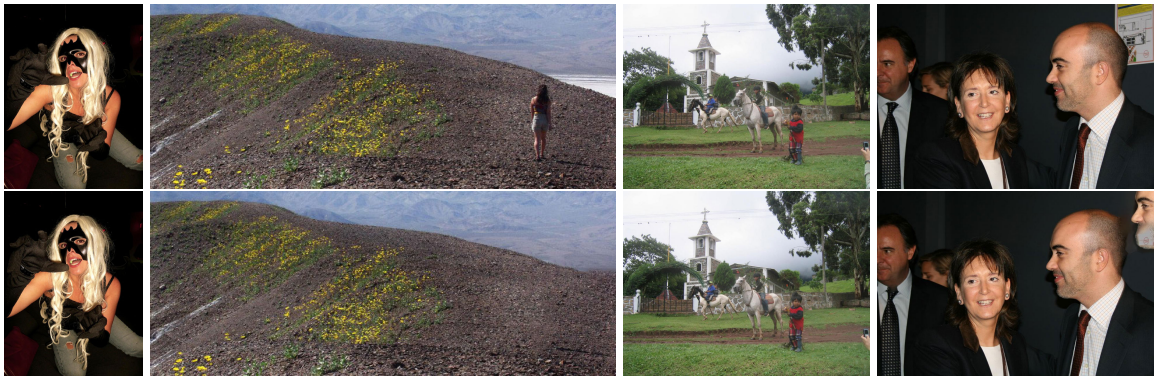


Figure 3.9: Two model failures, a segmentation failure, and an in-painting failure (see Section 3.6). Top row: original images. Bottom row: output images.

our patch-based hole filling method failed to remove it properly and instead duplicated the person.

Each of these failures suggests directions for future work. The first two cases suggest our model could be improved by using features related to image composition, a main subject detector, or relations between different elements in the image. The segmentation failure suggests focusing on improving the segmentation using the detected saliency. And of course, other image manipulations beyond patch-based hole filling [12] could be used to attenuate distractors like the last example: Since color saturation and contrast are key components of distractors, we can also consider removing them via de-saturation, exposure and contrast attenuation, blurring and various other methods. Implementing several removal methods and learning a model to automatically select the best one for a given distractor is an interesting direction for future work.

There are also additional applications of distractor prediction that we have not fully explored. For example, in addition to retargeting and inpainting, automatic cropping [105] could make use of distractor maps. However, since objects cut off at the edge of frame are often highly distracting, one would have to take into account the change in prediction that occurs as a result of the crop itself. One could also consider the use of distractor prediction as a cue for computational image aesthetics methods.

Chapter 4

Content-specific Photo Editing

"There are no bad pictures; that's just how your face looks sometimes."

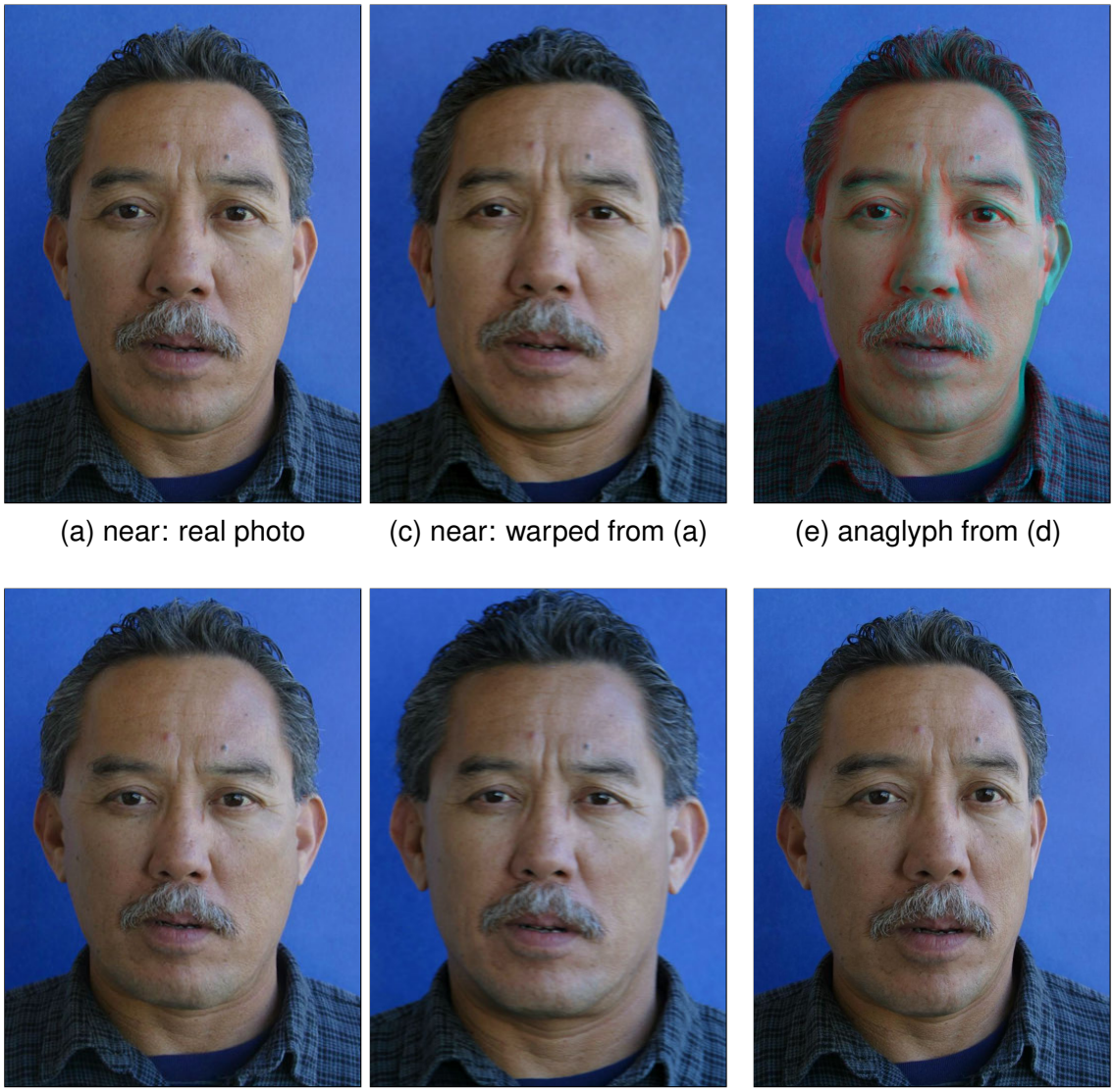
— Unknown, (falsely attributed to Abraham Lincoln)

"I've never met a person I couldn't call a beauty."

— Andy Warhol, *The Philosophy of Andy Warhol (1975)*

In more than a century since the invention of the daguerreotype, photographers have developed a set of conventions for effective composition of a photo. For example, the combination of subject pose, camera angle, and lighting can help define a jawline. Even the *camera distance* to the subject impacts perception; the literature shows that portraits taken up close are associated with terms such as "peaceful" and "approachable", whereas headshots taken from further away are perceived as "attractive", "smart" and "strong" [19, 81, 82].

This paper introduces a method that can subtly alter apparent camera distance and head pose *after* a portrait has been taken (Figure 4.1). This system fits a virtual camera and a parametric 3D head model to the photo, then models changes to the scene in the virtual camera, and finally approximates those changes using a 2D warp in the image plane. Similar frameworks have been used for a variety of applications including changing pose and



(a) near: real photo (c) near: warped from (a) (e) anaglyph from (d)

(b) far: warped from (d) (d) far: real photo (f) rotated from (d)

Figure 4.1: Comparing real photos taken with a near (a) or far (d) camera, one can observe the subtle effect of perspective on portrait photos. We simulate this effect by warping (d) $\rightarrow$ (c) to match the apparent distance of (a); and also (a) $\rightarrow$ (b) to match the distance of (d). These warps are guided by an underlying 3D head model. This framework can also generate stereo anaglyphs (e) and apparent head rotation (f).

gender [14], face transfer [102], and expression transfer [108]. Our work specifically builds on the FaceWarehouse approach of Chen et al. [24]. These prior methods all use a weak perspective camera model, which is a reasonable approximation only when scene points are all at a similar distance to the camera. In contrast, our approach uses a full perspective camera model, which allows us to modify camera distance and handle scenes that come very close to the camera. In a full perspective camera model, the distance and field of view parameters are *nearly* interchangeable, which makes optimization challenging. Nevertheless, this model is necessary for several of the effects that we show, especially treatment of "selfies."

Today most photos are taken using mobile devices with fixed focal length. This trend accounts for the sudden explosion of the "selfie" – 2013 word of the year in the Oxford Dictionary – meaning a portrait taken of oneself, often with a smartphone. Selfies are typically shot at arm's length, leading to visible distortions similar to the fisheye effect but with their own characteristics, most notably an enlarged nose. In some cases this selfie effect may be desired, but professional portrait photographers often prefer to position the camera several meters from the subject, using a telephoto lens to fill the frame with the subject [99]. Figure 4.2 shows two photos of the same subject, revealing the effects of this tradeoff [79]. Our framework allows one to simulate a distant camera when the original shot was a selfie, and vice versa, in order to achieve various artistic goals – reducing distortion, making a subject more approachable, or adapting a portrait such that it may be composited into a group shot taken at a different distance.

We show that our framework can also create convincing stereo pairs from input portraits or videos, rendered as anaglyphs. The approach relies on the full perspective camera available in our 3D model. Finally, our method is also capable of other applications shown in previous work using a weak perspective model, such as simulating small rotations of the subject's head. Our main contributions are:

- The ability to edit perceived camera distance in portraits.

- A robust head fitting method that estimates camera distance.
- A new image warping approach that approximates changes in head or camera pose.
- A method to create stereo pairs from an input portrait.
- Evaluation of our approach using an existing dataset and a new dataset captured for this purpose.

4.1 Related Work

Despite a large body of work on face modeling, 3D face shape estimation from a single image is still considered challenging, especially when the subject is captured under unconstrained conditions (varying expressions, lighting, viewpoint, makeup, facial hair). High quality face reconstruction methods often require the subject to be scanned under controlled laboratory conditions with special equipment such as lighting rigs and laser scanners [32, 104, 4, 16]. Kemelmacher and Seitz [56] showed it is possible to reconstruct a face shape from a large Internet collection of a person's photos using ideas from shape from shading. These methods are not applicable in a single photo scenario.

In their seminal work, Blanz and Vetter [14] fit a 3D face morphable model to a single input image, texture-map a face image onto a 3D mesh, and parametrically change its pose and identity. Vlasic et al. [102] extended their work using a multilinear model to handle expressions and visemes. FaceWarehouse [24] extended the model from the face region to an entire head shape. Other single-image reconstruction methods include an approach based on patch-based depth synthesis from a 3D dataset [44], photometric stereo with a 3D template prior [55] and a 3D template corrected with a flow optimization [43]. Unlike morphable models, the latter do not allow changing the identity and expression of the subject.

In order to edit 3D face properties in a photograph using any of the above methods, the face has to be accurately segmented from the background, texture-mapped onto the face mesh, and then projected back to the image after the mesh is edited. The background, the

rest of the head, and the eyes and teeth must be adjusted – often manually – to fit the pose change. This complex pipeline can result in an unrealistic appearance due to artifacts of segmentation, color interpolation, and inpainting.

An alternative approach uses the 3D model to generate a 2D warp field induced from a change in 3D, and apply this warp directly on the photograph [108, 106]. This approach doesn't support extreme edits, but it can be fully automated and often leads to more realistic results. We adopt this approach, driving our warp field with a multilinear morphable model with parametrized pose, identity, and expression.

Existing morphable model methods typically have two main drawbacks: First, the camera distance is given as input (or assumed to be infinite) and remains fixed; and second, there are no annotations near the top of the head, which we show poses a major problem for fitting and altering the apparent camera distance. We extend a multilinear model to incorporate camera distance, and present an optimization algorithm for the more challenging fitting problem that results. We also add a few annotations in some key locations and show in Section 4.3 that these are critical for our application.

The methods of Cao et al. [23, 22] and Hassner et al. [45] estimate a perspective camera model similar to our approach. Cao et al. drive a real-time animation with an input head video, but their system uses multiple frames for accurate estimation of model parameters, whereas our goal is to use a single input image. We tested some of their underlying assumptions and found them inapplicable to the case of single-image input (Section 4.2.3). Also, Cao et al. reduce the intrinsic matrix to a single parameter estimation (focal length), fixing the principal point offset to zero. In order to support, for example, cropped images, our model estimates this offset as well (2 extra parameters). Hassner et al. frontalize a face given an input image. They estimate the intrinsic camera matrix given a fixed 3D template model, since an accurate fit is not required for their task. In contrast, our method addresses the harder problem of jointly estimating camera and model parameters. Nonetheless, some features of the method proposed by Hassner et al. are complementary to ours, for exam-

ple "borrowing" features from one side of the face to complete the other could be used to augment our system, in the case where there are occlusions in the input.

The perceptual literature draws a direct connection between camera distance, lens focal length, and the way we perceive people in photos: Portraits taken from up close are associated with terms such as "peaceful" and "approachable", while those taken from further away are "attractive", "smart" and "strong" [81, 19, 82]. Cooper et al. [31] further showed that portraits taken using a 50-mm lens are most likely to be viewed from a distance from which the percept will be undistorted.

The Caltech Multi-Distance Portraits Dataset [21] contains portraits of different subjects taken from various distances. In their paper, the authors created a way to estimate the camera distance from an input portrait photo. We use their dataset to evaluate our method.

No previous method suggests changing the apparent camera distance in a photo. As far as we know, we present the first work to address the task of fixing portrait distortions due to camera distance.

4.2 Our Method

To perform perspective-aware manipulation, first we formulate a parameterized 3D model for a head and camera (Section 4.2.1), automatically detect fiducials in a photo (Section 4.2.2), and fit the model to the observed fiducials (Section 4.2.3). Next, we can alter the parameters of the model (e.g., move the camera or head pose, Section 4.2.4) and then approximate the resulting 3D changes as a 2D warp to the input image (Section 4.2.5).

4.2.1 Tensor Model

Our head model builds on the dataset collected by Chen et al. [24]. This dataset contains scans of 150 individual heads, each in 20 poses. Each head has 11,510 vertices in 3D (34,530 DOF). Expressions are represented by a blendshape model using 47 shapes.

©Anton Orlov, used with permission.

Figure 4.2: Compare focal lengths. Left: close-up using 90mm wide angle lens with a large format camera (29mm equivalent on 35mm film). Right: distant shot with 265mm telephoto lens (84mm equiv.)

Let us denote the average of all heads in the dataset as $A \in \mathbb{R}_{34530\times 1}$. We calculate the difference of each head from the average and arrange the data in a tensor $Z \in \mathbb{R}_{34530\times 150\times 47}$, with dimensions corresponding to vertices, identities and expressions, respectively. We use high order SVD (HOSVD) [98] to calculate a core tensor $C \in \mathbb{R}_{40\times 50\times 25}$. Here our approach differs from that of Chen et al. [24], who do not perform SVD on the vertex dimension. We find that our compact representation still produces good results. Given the core tensor we use an identity vector $\beta \in \mathbb{R}_{1\times 50}$ and an expression vector $\gamma \in \mathbb{R}_{1\times 25}$, together with the original vector expansion calculated by HOSVD $v \in \mathbb{R}_{34530\times 40}$ to generate a head with a specific expression and identity F' via:

$$F' = (C \otimes_1 v \otimes_2 \beta \otimes_3 \gamma) + A \tag{4.1}$$

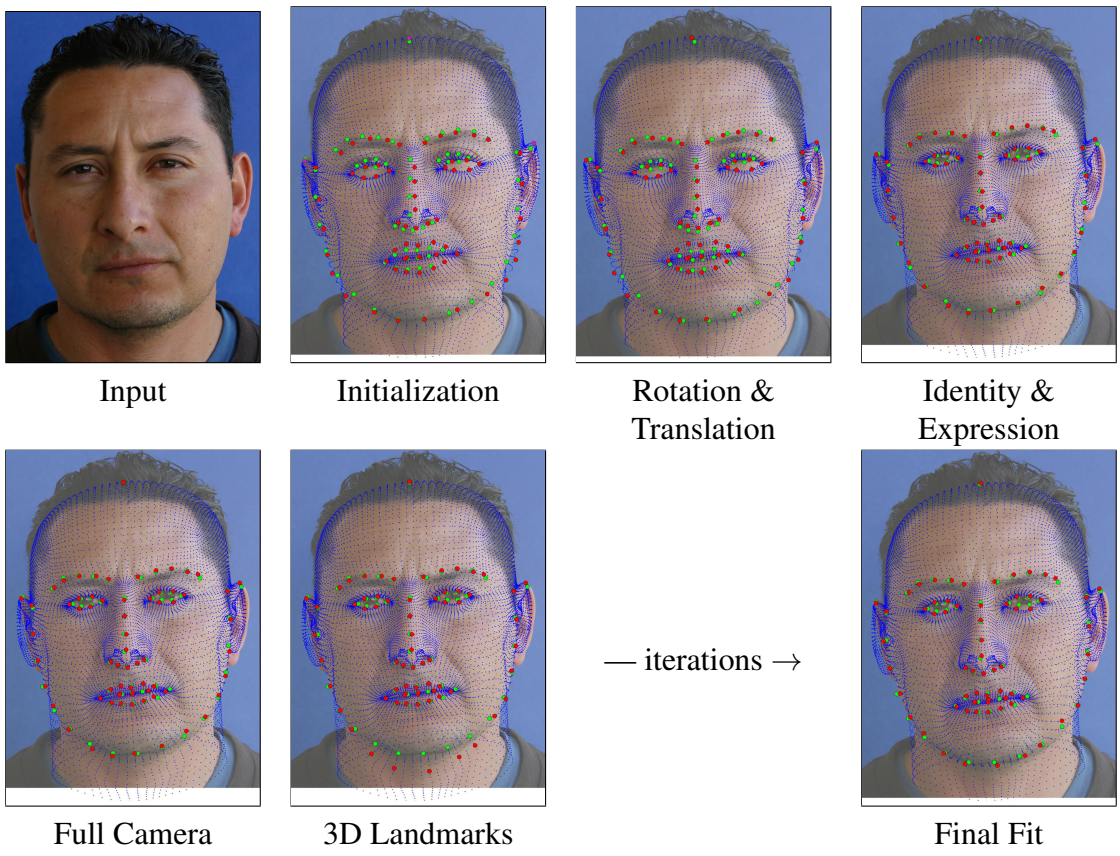


Figure 4.3: Fitting procedure. Green dots are 2D fiducial points. Red dots are corresponding points on 3D mesh (shown in blue). Purple dots in initialization image are three manually annotated fiducials for top of head and ears. *Images, from left to right:* Input. Initialization gives a rough fit, but with some misalignments (e.g. eyes). Solving rotation and translation improves the silhouette fit (e.g. chin). Solving identity and expression fixes the eye and mouth misalignment. Solving the full camera model improves, in this case, the top of the head. After 3D landmark update the alignment is worse, but landmark locations on the 3D mesh are more accurate. Repeating the process produces a good final fit.

Here $\otimes_i$ is the standard tensor-vector multiplication in the i-th dimension. Let us denote $F'' \in \mathbb{R}_{4\times 11510}$ as the natural reshape of F' such that each row contains x, y, and z coordinates respectively, with an added row of ones to create homogeneous coordinates. In order to generate a head in a specific location and orientation, as seen by a camera, we need to multiply the head vertices (which are in the model coordinate system) with translation $T = [\mathbb{1}_3| - t] \in \mathbb{R}_{3\times 4}$, rotation $R \in \mathbb{R}_{3\times 3}$ and the upper-triangular intrinsic matrix

$K \in \mathbb{R}_{3\times3}$. Thus, our full model (omitting the perspective divide for simplicity) is:

$$F = K \cdot R \cdot T \cdot F'' \tag{4.2}$$

We found that a general intrinsic matrix K with five parameters leads to bad shape estimation. Instead we constrain the skew to be zero and the horizontal and vertical focal length parameters to be the same – reasonable assumptions for unaltered photos from modern cameras. This intrinsic matrix constrained to three DOFs significantly improves the fit.

We contrast this full perspective model with previous work that uses weak perspective (e.g. [102, 108, 107, 24]) – essentially using orthographic projection, followed by non-uniform scaling. With weak perspective camera distance is represented by scaling, so there is no way to adjust distortions due to nearby cameras, e.g., as seen in selfies.

4.2.2 Fiducial Detection

The method of Saragih et al. [88] automatically detects 66 fiducial points on faces: chin (17), eyebrows (10), nose stem (4), below nose (5), eyes (12), and lips (18). Unfortunately, these locations (which are also common for other detectors) are not sufficient for our purposes because they lack points above the eyebrows and on the ears. Since our system manipulates perspective, such points are crucial to model the effects on apparent head shape.

Rather than invent a new fiducial detector, which we leave for future work, we use an existing detector [88], and manually annotate three extra points on top of the head and ears. We chose a small number of points to facilitate quick annotation (less than five seconds).

4.2.3 Fitting

Given an input image and the 69 fiducial point locations (Section 4.2.2) we would like to fit a head model to the image. Since all models in our dataset share the same vertex ordering,

we know the location of the corresponding fiducial points on the 3D models. Armed with Equation 4.1 and Equation 4.2 the task is now to find the best parameters β, γ, K, R, t (50 + 25 + 3 + 3 + 3 = 84 in total) such that the Euclidean distance between the fiducial points and the projection of the 3D landmarks is minimized.

Many fitting strategies are possible. We experimented with several and discuss them before describing our proposed approach. A naïve approach is to treat the problem as one large non-linear least square optimization. However, we found this approach gets stuck in local minima. Using coordinate descent, as described in Algorithm 1, obtained lower global error. Other works [108, 106, 24] also used coordinate descent. However our optimization problem is much harder due to the inherent non-linearity of the camera projection model (Algorithm 1 Line 6), which introduces ambiguity between the camera distance, focal length and the expression and identity parameters. We also tried adapting this naïve approach by using even more fiducial points, and achieved sub-par results. Our experience suggests that merely adding more points does not completely solve the problem.

We also experimented with the approach of Cao et al. [23, 22] for focal length estimation. It assumes that the fitting error taken as a function of focal length is convex. We tested this convexity assumption in the context of our global optimization, by repeating their experiments using un-cropped images from the Caltech Multi-Distance Portraits (CMDP) Dataset [21], and found that convexity does not hold when calculated using a single image. Moreover, the global optimum of the focal length was off by an average of 35% and up to 89% from the EXIF value. In contrast, Cao et al. were able to obtain errors below 2% using multiple frames.

The aforementioned experiments led to a more deliberate design of the initialization and of the gradient descent order (e.g. adding Line 2 as a precursor to the optimization loop). All the results shown in this paper use a total of 3 iterations. The following sections explain the different subparts of the optimization. Figure 4.3 contains an overview of the fitting procedure.

Algorithm 1 Fit model to image

1: Initialize camera, identity and expression parameters (§4.2.3)
2: Solve rotation and translation (§4.2.3)
3: **for** i in 1..num_iterations **do**
4: Solve identity (§4.2.3)
5: Solve expression (§4.2.3)
6: Solve camera (§4.2.3)
7: Update 3D landmark location (§4.2.3)
8: **end for**

Initialization

We extract the focal length f_E from the EXIF data of the image as an initial guess. We allow this value to change during optimization, to account for EXIF inaccuracies and the uncertainty of the exact location of the focal plane. We also use the distance t_c between camera and subject if it is known (e.g. in the dataset of Burgos-Artizzu et al. [21]). We initialize our camera parameters to be:

$$K_0 = \begin{bmatrix} f_E & 0 & 0 \\ 0 & f_E & 0 \\ 0 & 0 & 1 \end{bmatrix}, t_0 = \begin{bmatrix} 0 \\ 0 \\ t_c \end{bmatrix}, r_x = r_y = r_z = 0 \tag{4.3}$$

Here, r_x, r_y and r_z are the x, y and z rotation, respectively. If distance t_c is unknown we use a default value of 1m. Initializing β_0 and γ_0 to the average of all identity and expression vectors in our dataset, respectively, we solve for initial parameters β, γ, K, R, t using an interior-reflective Newton method [30], minimizing the Euclidean distance between 2D fiducial points and the 2D projections of corresponding 3D landmarks. Specifically, let $L = \{l_i\}$ be the 2D fiducial locations (Section 4.2.2) and $H = \{h_i\}$ be the corresponding 3D head vertices projected to the image plane by Equation 4.2. Our objective is then:

$$\min_{\beta,\gamma,K,R,t} \sum_{i=1}^{N} \|l_i - h_i\|_2^2 \tag{4.4}$$

where N is the number of fiducial points (69 throughout this paper).

Parameter Update

As introduced in Algorithm 1, we solve for rotation R and translation t once. Next holding these parameters fixed, we repeatedly solve for identity β, expression γ, and camera parameters K, R, t. As with initialization (Section 4.2.3), these optimizations use the interior-reflective Newton method to minimize Equation 4.4. We find it critical to solve first for rotation and translation only: Solving first for expression or identity results in a distorted face that overcompensates for bad pose. Solving first for the full camera matrix occasionally results in erroneous focal length.

3D Landmark Update

Some landmark locations are expected to remain fixed on the 3D model, regardless of view angle. For example, the corner of the eye should be the same vertex for any pose. However, other landmarks are pose-dependent. Specifically, the chin and the top of the head are entangled with pose. Of course, the chin doesn't actually change location; rather our fiducial detector detects contour points along the chin, and these contours are view-dependent. Thus, after initial calculation of a face shape and location, we need to recalculate the location of these "soft" landmarks. This step needs to be reasonably efficient because it is iterated many times in Algorithm 1. We follow an approach similar to that of Yang et al. [108], with two modifications: First, we add the top of the head as a movable landmark. Second, their work used a face model, rather than a full head model. Because the projected shape is nearly convex, they described an approach that iteratively projects towards the convex hull in 2D to find the contour. Since we have a full head (including protruding ears) our projected shape is far from convex. We address this problem by a one time preprocessing step in which we find a restricted set of "valid" chin and head points (omitting ears and neck, for example) and then restrict the landmark update to consider only these valid points.

4.2.4 Changing Distance and Pose

Given a good fit between the input image and the head model, we can now manipulate the model. We move the virtual camera towards or away from the subject by changing the translation t. To rotate the head we adjust both translation t and rotation R, since translation is applied before rotation. Rotation is achieved by translation in a diagonal direction (relative to the line between camera and subject), followed by a rotation to place the head back in the visible frustum. These modifications result in a new projected head shape, which will guide the warp described next.

4.2.5 Warping

After manipulating distance or pose we now have two sets of points: First, the original 3D face vertices that match the input image, and second, the manipulated vertices representing a change of distance, pose, expression or any other 3D manipulation. Given these two sets, we find a 2D image warp to produce the output image. However, some points are "occluded" for the purpose of the warp. For example, our head model includes back-facing areas, but such areas move in the direction opposite from the front-facing areas when changing camera distance. Therefore we remove occluded vertices before calculating the warp.

Given a sparse set of before and after points, we need to extrapolate the vector field to the entire image. We use triangulation-based cubic interpolation to get an initial estimate of the dense vector field. Although correct in 3D, strong discontinuities in the vector field may cause artifacts in the output. Consider, for example, an extreme rotation of the head. Cheek points that had the same x and y values (but different z values) need to be stretched to different x locations, causing a shear. Note that the vector field in this case is correct in 3D, but cannot be approximated well by a 2D warp. Therefore, we smooth out large gradients in the warp field, as follows: We first replace all values outside the face region with a smooth interpolation of the valid values, by computing the discrete Laplacian and

solving a Neumann boundary condition. We next blur the vector field by convolving with a disk with radius $\frac{1}{20}$ of the photo diagonal. Finally with the smooth warp field we use reverse mapping to calculate the origin of each pixel in the output image. We use linear interpolation since it is simple, fast, and produces satisfactory results. Figure 4.4 shows a breakdown of these steps.

4.3 Evaluation

In this section we evaluate our method. Section 4.3.1 demonstrates the importance of different parts of the pipeline by showing results with various stages disabled. Section 4.3.2 and Section 4.3.3 compare our results against ground truth photos of synthetic and real heads, respectively, taken at known distances. Section 4.3.4 discusses the impact of our warp on the background of the portrait.

4.3.1 Pipeline Evaluation

For a face with neutral expression and common proportions, a single average head model might suffice (Section 4.3.3). However, when the input image is expressive, it is important to use the full face model. Figure 4.5 shows results using an average face model, instead of optimizing a specific identity and expression to our image. Clearly a single face cannot be a catch-all solution, resulting in artifacts due to bad alignment.

Fiducial point based matching from a 3D head to a 2D image is sensitive to the choice of landmarks. Many existing works use 66 standard points spread across the chin, eyebrows, nose, eyes and lips [108, 106, 24]. This choice is motivated mostly by ease of recognition. When fitting a face model and manipulating only the face internals, such landmarks might suffice. However, we found that full head manipulation, especially one where the camera location is changed, requires more fiducial points. Adding three points (top-of-head and

ears) leads to significantly better results for our scenario. Figure 4.6 shows failure cases when these additional landmarks are not used.

Our warping procedure (Section 4.2.5) uses well established methods (such as triangulation and sampling). However, we use a specific procedure with added steps to reduce potential artifacts. Figure 4.7 compares our warping results to results obtained by standard image warping techniques.

4.3.2 Synthetic Heads

Numerically evaluating our method is hard. Photos of real people taken from different distances at different times have slight variations in illumination, expression and pose, thus the "ground truth" image does not match exactly a warped version of the input. To tackle this issue we perform evaluation on two types of data: mannequin heads and real people. The mannequin heads provide a controlled environment, for which we can get accurate ground truth.

Figure 4.8 shows several results with mannequin heads. Our input image is taken from a distance of 90cm. We warp it to simulate a range of distances between 120cm and 480cm. We compare each warped result to the ground truth by calculating the absolute difference of the gray-scale pixel values (black indicates equality; white indicates the largest difference.) Note that the method manages to simulate both the head shape and the location of internal features such as eyes and ears.

4.3.3 Real Heads

To obtain a similar evaluation of real-world cases, we use the CMDP dataset [21], which contains portraits of people captured from a few controlled distances. We evaluate the process of changing the camera distance from an image shot at 60cm to 480cm and then compare to a real image captured at that distance. This is the harder direction of manipulation, as features in the close-up image (e.g. ears) are not always visible.

However, a naïve pixel difference will not suffice here, due to slight pose, expression and illumination changes in the ground truth images. Therefore to compare two images we:

1. Register the images using a rigid transform, to avoid penalizing simple rotations or translations.

2. Use Large Displacement Optical Flow [18] to calculate optical flow between the images.

3. Mask out the background regions, since we are only interested in the head warp.

4. Calculate the median optical flow magnitude in the head region. To normalize, we multiply by 100 / image diagonal.

Figure 4.9 numerically compares our method to the following four alternatives: 1) Compute an optimal radial distortion correction given known ground truth (giving maximal advantage to this method to test its potential); 2) Use only the fiducials and the vertices of the face to drive the warp, simulating the fitting done by methods like [108, 24] and many others; 3) Fit an average head instead of the multi-linear deformable model and warp using our method, representing methods like [55, 57, 45] that use a single model; We use a mean full head model, averaged from the dataset in [24] as apposed to just a face model as was done in previous methods, to explore a full potential of this approach for our task. 4) Fit our full model. Figure 4.10 shows representative results.

Figure 4.5 and Figure 4.13 show our results for input images with a non-neutral pose and expression. We compare against a static model, showing that a deformable model is important.

4.3.4 Background Preservation

Most of the examples shown so far had a rather uniform background that might hide warp artifacts if they exist. While some works in the area are limited to these types of inputs, we

would like to have a system that works in the wild. Moreover, we cannot expect the user to mask the area around the head, since we aim for a fully automatic method.

Thus, we require minimal distortion in the background, which we achieve by using a 2D warping approach (Section 4.2.5) rather than a 3D texture mapping approach requiring perfect head segmentation. In Figure 4.12 we show several examples of our warp result on noisy backgrounds.

4.3.5 Runtime

Our method is implemented in Matlab, and can be further optimized. Typical runtime is around 5 seconds to fit the model to the input image, and less than 1 second for warp field generation and warp calculation. To support real-time interactivity, we also created a WebGL viewer that can adjust warps on the fly (Section 4.4.3). We pre-calculate warp fields for a few predefined distances or other parameters such as pitch and yaw. The pre-calculation takes 3 seconds for 4 samples of the distance parameter. After pre-computing these warp fields, the interpolated warp is rendered in the web browser in real time (more than 60 FPS).

4.4 Applications

Our primary application is to adjust camera distances (Section 4.4.1). We also discuss other applications including stereoscopic portraits (Section 4.4.2) and pose adjustment (Section 4.4.3).

4.4.1 Distance Correction

Our main motivating application is to adjust camera distance in portraits. Figure 4.11 shows distance manipulation results for seven subjects from the CMDP dataset. In each case the 60cm portrait was warped to match the 480cm one, and vice versa, so they can be compared

to ground truth. Note that the changes are subtle but noticeable. Moreover, these changes are more prominent when the subject is known (yourself, family or a friend). We refer the reader to the accompanying video as well as the interactive viewer in the supplemental materials for more examples.

All the above results are from a controlled dataset, for comparison to ground truth. However, our system also works well on images "in the wild." Figure 4.12 shows distance manipulation on real images tagged as #selfie on Twitter and Flickr. Our system works across a variety of expressions and poses despite cluttered backgrounds and complex lighting. Figure 4.13 and Figure 4.5 further illustrate the robustness of our method to exaggerated expressions and poses. More examples are in the supplementary materials.

4.4.2 Headshot Stereoscopy

We can create stereoscopic images using our framework. Given the distance from the subject and the average human interpupillary distance, we can modify the viewpoint to obtain two new images — one for each eye. Those images can then be displayed on devices such as VR headsets. Figure 4.14 shows 3D anaglyphs automatically created from 2D photos using this approach. These can be viewed using a standard pair of red/cyan glasses (red eye left).

4.4.3 Other Applications

Our 3D fitting pipeline is based on the multi-linear morphable model framework. As such, we can replicate some of the face manipulation tasks shown in previous work using similar models [102, 108, 24]. These include pose and expression manipulation, and animating a moving face from a still image (see Figure 1f and the accompanying video).

Our WebGL based user interface supports interactive photo manipulation. The user is presented with the image and sliders to manipulate camera distance and head pose (Figure 4.15). We calculate warp fields for predefined parameter values (4 distances, 5 pitch

values, 5 yaw values). When the user selects a specific parameter combination, we use trilinear interpolation to generate a warp field. Then, we use the warp field to create the output via reverse mapping. Output images are rendered at real-time rates, allowing users to adjust parameters to their liking.

4.5 Limitations and Future Work

We present a unified framework for altering the camera and subject pose in a portrait photo. This method can be used to improve selfies, make a subject look more approachable or adapt the camera distance to match a different shot for compositing. We display results for various scenarios and compare with ground truth data. Our editing operations remain in the realm of "plausible" – they do not create new people, rather they show the same people under different viewing conditions. In that sense, they are the post-processing equivalent of a portrait photographer making a different decision about the composition. Our framework also supports creating stereoscopic views from portraits and video, as well as making video with apparent camera and subject motion from a still portrait. More results, video and demos may be seen on our project page **`http://faces.cs.princeton.edu/`**.

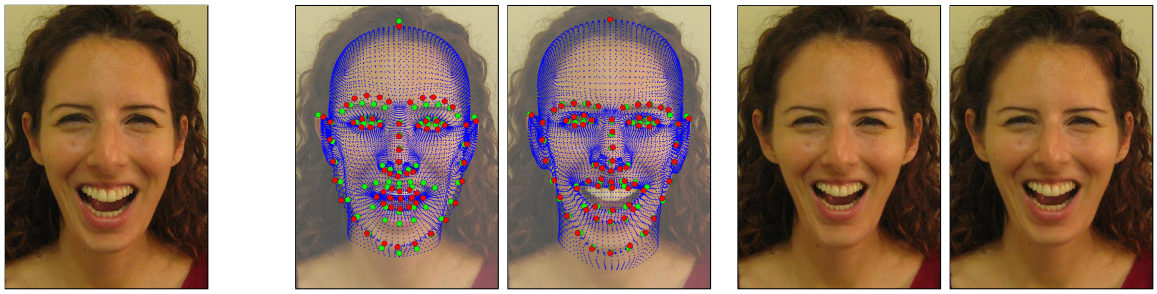
Our approach has several weaknesses that suggest opportunities for future work. First, the pipeline relies on a good fit between input and model, and if the fit fails, the results will be distorted. While our optimization has proved robust in many cases, occasional failures remain. Future approaches might build on larger, more varied head shape datasets, or rely on 2.5D sensor data emerging in new camera rigs. Second, we only warp the data that exists in the original image. This produces convincing results in many cases, but will not handle significant disocclusions such as can arise from significant head rotations. One way to address this might be by filling missing regions via, e.g., texture synthesis with a strong face prior [45]. Third, the way we currently treat hair is by a smooth extrapolation of the warp field outside of the head region. This is often insufficient, and could be improved with a specialized hair model. Fourth, our method does not handle eye gaze correction and extreme expression change which may be desired in some scenarios. One could experiment with existing techniques for editing gaze [41] and expression [108]. Finally, while the

accompanying video shows a couple speculative applications for video (stereoscopic video and a "moving portrait") a proper investigation of such applications remains for future work.

(a) *initial field* (b) *with Laplace operator*

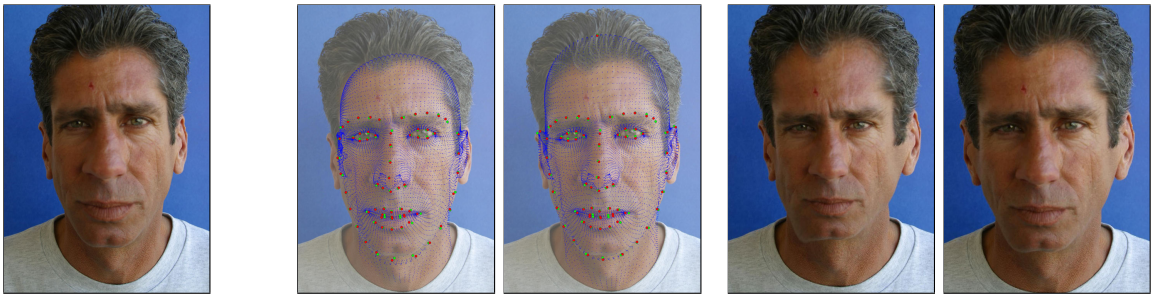
(c) *smoothed* (d) *with image*

Figure 4.4: Generating the dense warp field. (a) Initial dense field, with discontinuities in background and around face. (b) Improved background via discrete Laplace operator. (c) Smoothed using an averaging filter. (d) Overlay of the final warp field and input image.



Input Fit (single) Fit (full) Output (single) Output (full)

Figure 4.5: Using a single head model vs. our full model that allows expression and identity variation. Dot colors as in Figure 3. The single model yields a bad fit, especially near the smile, thus resulting in an unnaturally narrow chin.



Input Fit (66 fids) Fit (69 fids) Output (66 fids) Output (69 fids)

Figure 4.6: Using the standard 66 fiducial points vs. adding 3 points for top-of-ear and top-of-head. Dot colors as in Figure 3. Fitting to 66 points produces inaccurate alignment near the ears and the top of the head, thus resulting in unnatural proportions and skew.

Figure 4.7: Warp comparison. L-to-R: PiecewiseLinearTransformation2D (Matlab), Local-WeightedMeanTransformation2D (Matlab), our result without smoothing, our result. Input image in Figure 4.13.

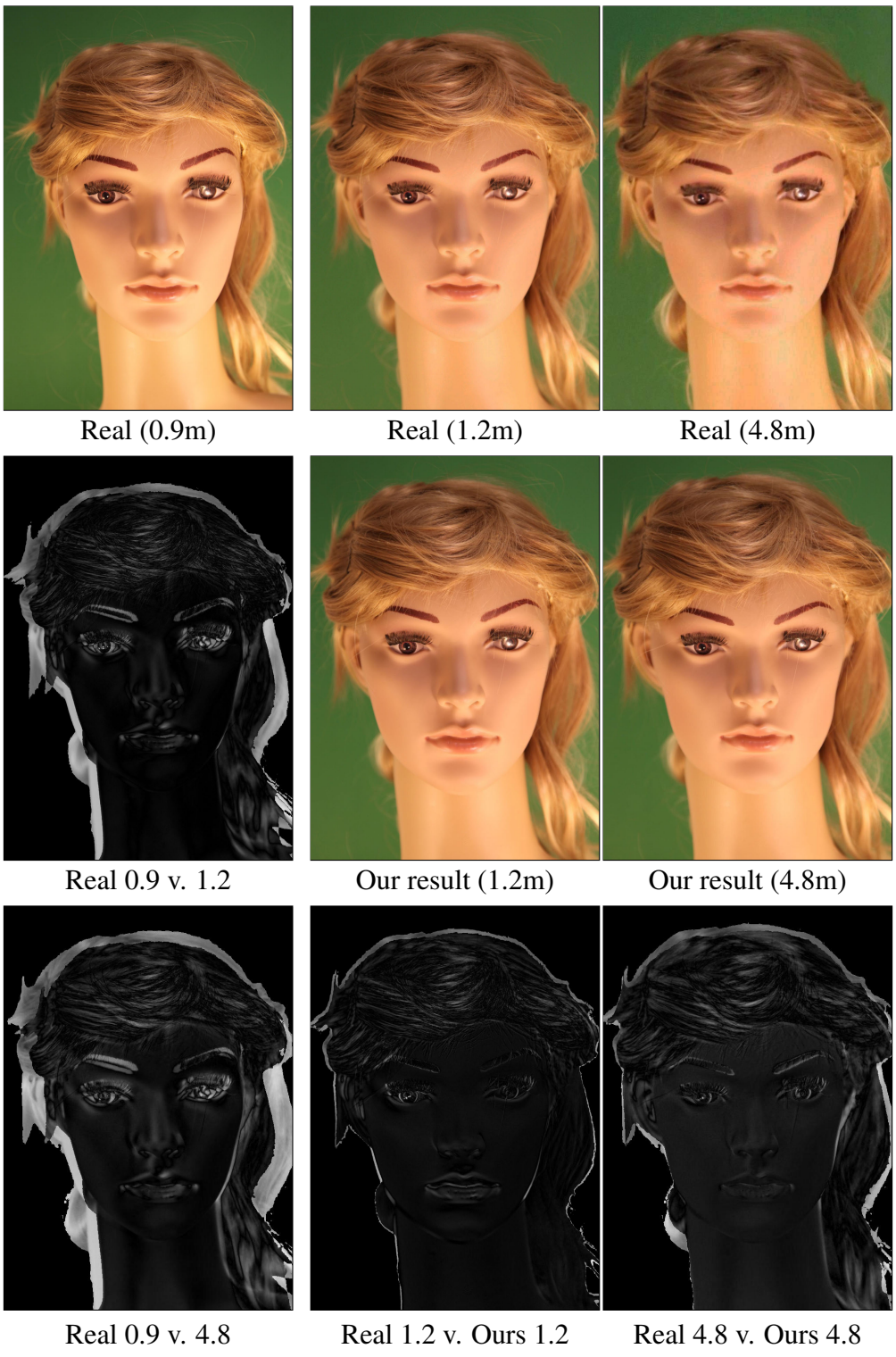


Figure 4.8: Ground truth evaluation. We use a mannequin to make sure no pose or expression changes occur in the ground truth images. Our results closely match the ground truth, both in overall head shape and the location of internal face features.

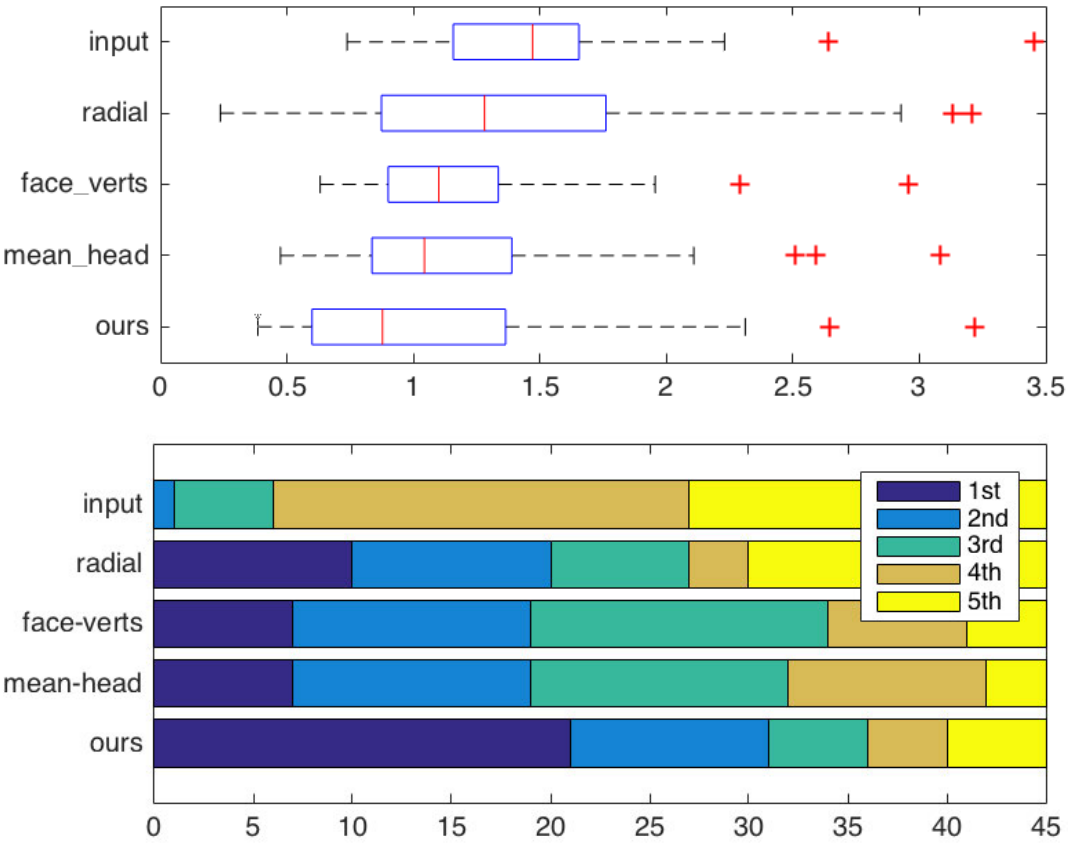


Figure 4.9: Score comparison with the 45 images from CMDP dataset that have EXIF data. We warp images taken from 60cm to appear like 480cm away, comparing to ground-truth photos from that distance. Energies are shown for: (1) input images, (2) radial distortion, (3) warping with a face only (4) warping using an average head, and (5) our full model. Top: median values of our energy function, where lower is better (Section 4.3.3). Boxes are 25th to 75th percentile and red line is median of medians. Bottom: we rank each method vs. all others, counting how often a method had each rank. Our method outperforms all others.

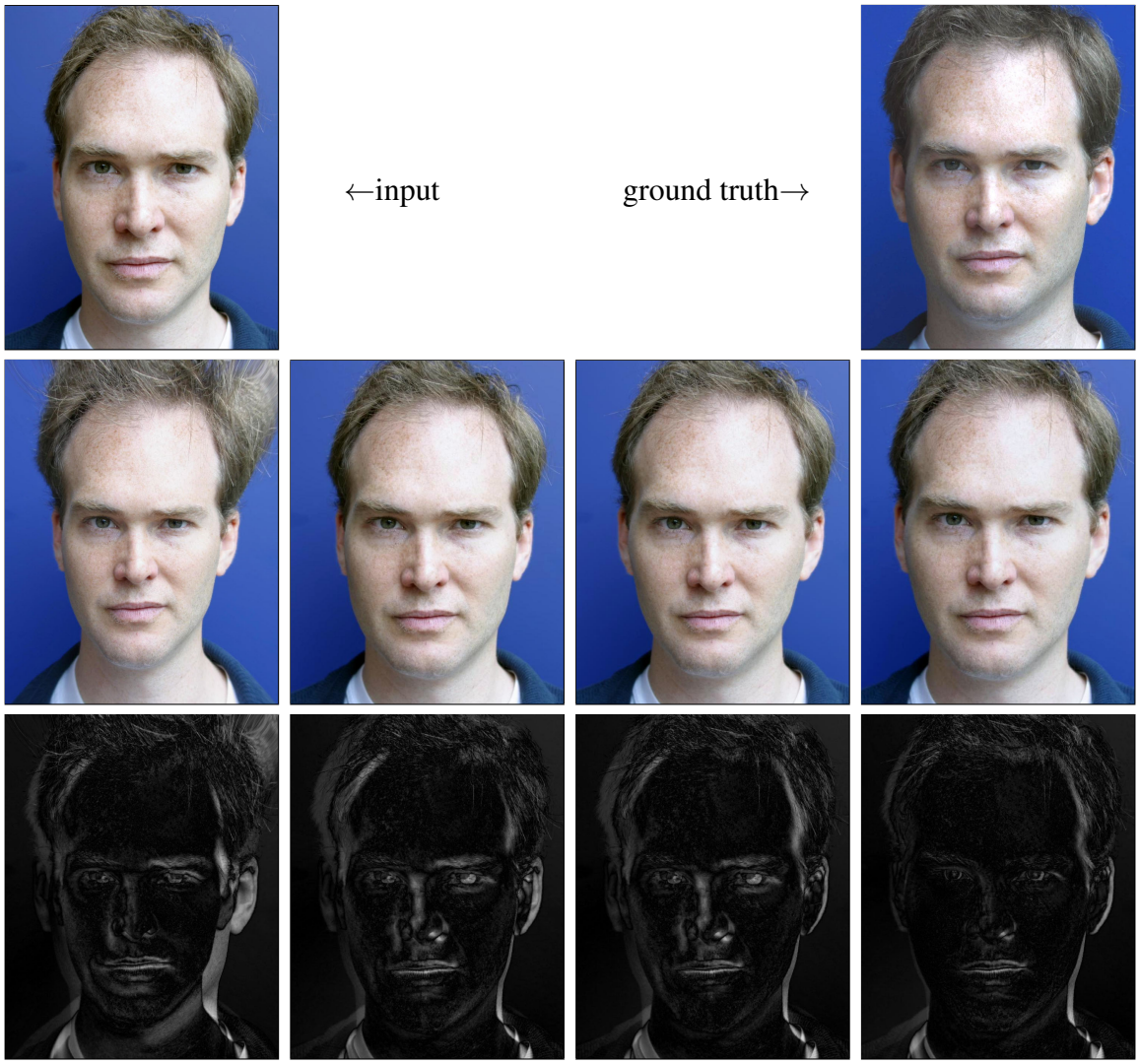


Figure 4.10: Comparing methods. Top: input and ground truth at target distance. Middle compares alternate approaches to ours: optimal radial distortion, fiducials on face only, mean head model, and ours. Bottom: visualizing error from ground truth.

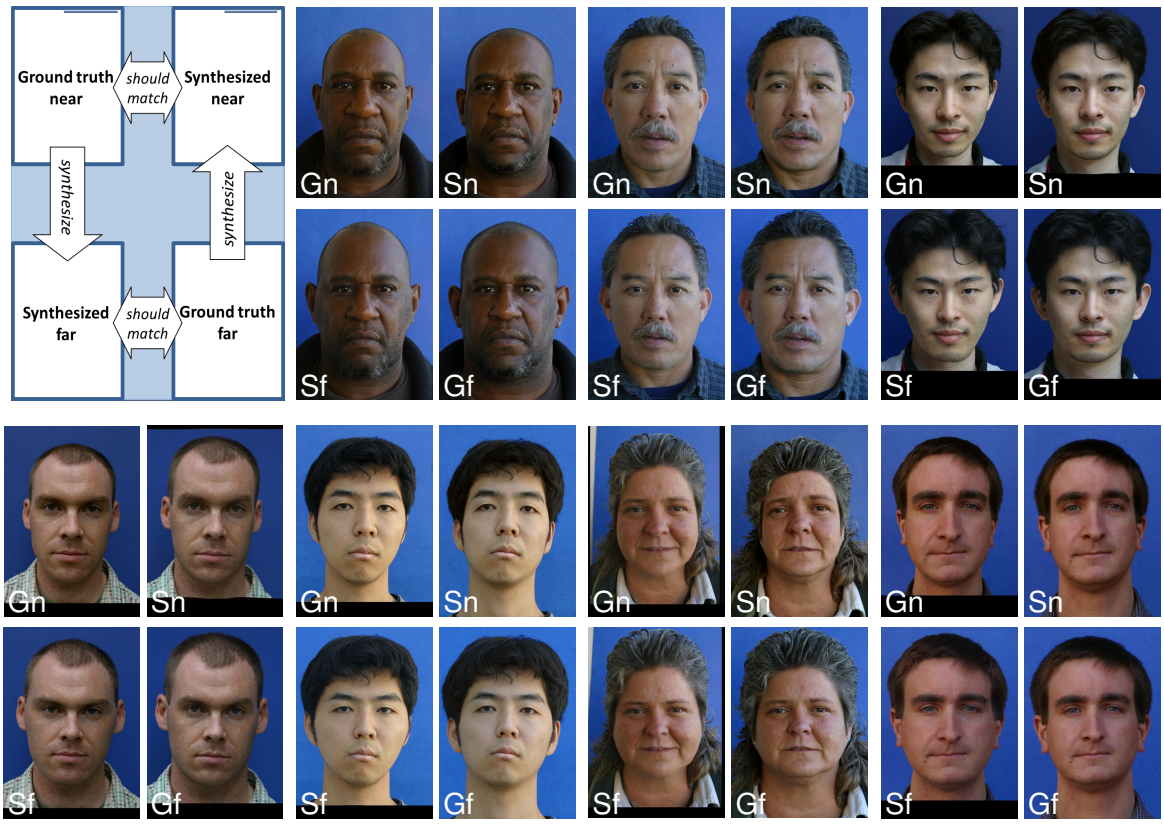


Figure 4.11: Fixing/generating selfies. Legend in upper-left corner shows arrangement of each quadruplet. Input ground truth "near" photos were taken at 60cm, whereas "far" photos were taken from 480cm (CMDP Dataset [21]). Synthetic images were warped from near to far and vice versa, and are arranged and color matched for ease of comparison. When evaluating, compare the head shape and the location of internal face features. These results are selected from a larger set available in supplemental materials.

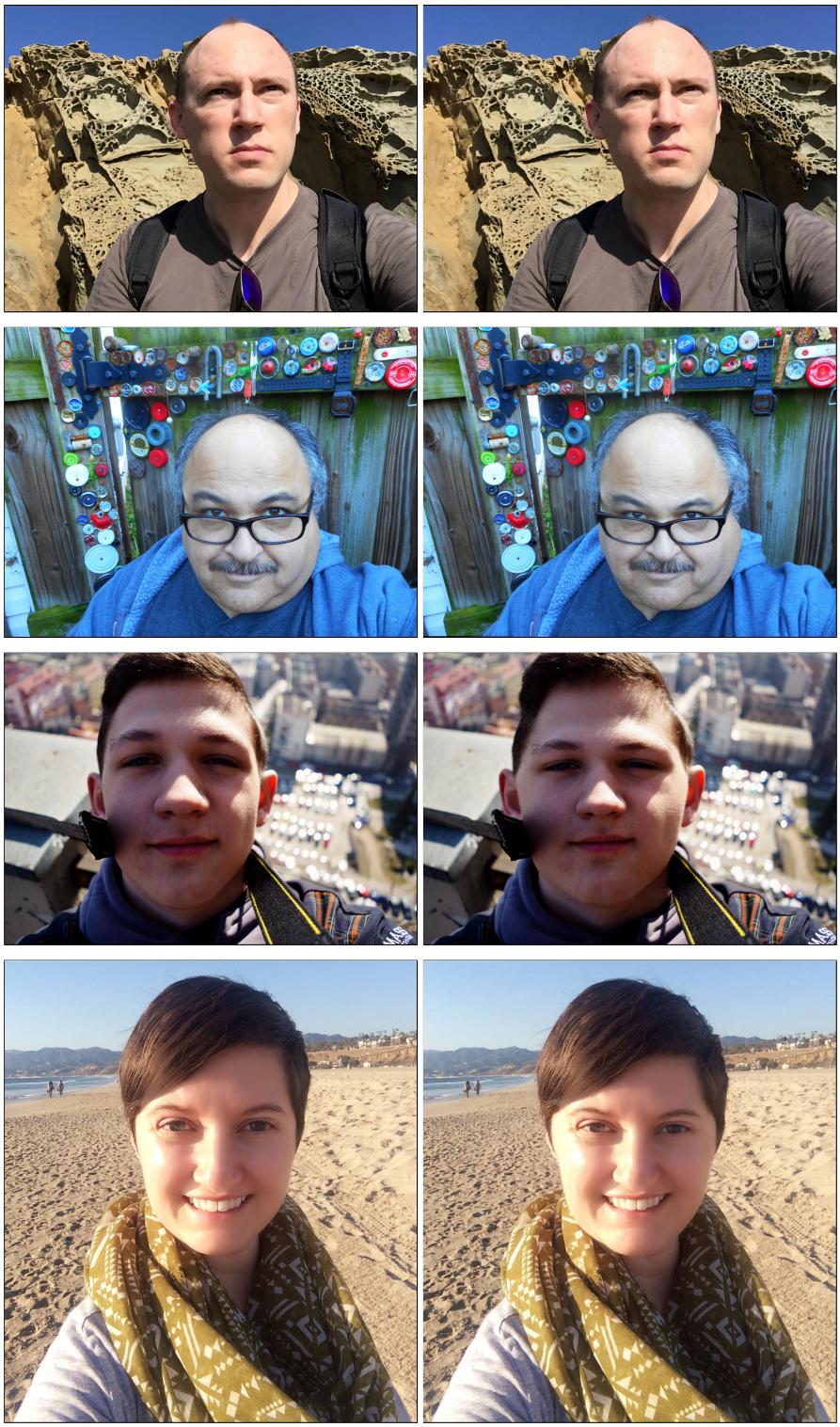


Figure 4.12: In-the-wild selfie correction. We use Twitter and Flickr images tagged as #selfie. Left: original, right: our result. Results shown for various head shapes. Background remains largely undistorted. ©Flickr users Justin Dolske, Tony Alter, Omer1r, and Christine Warner Hawks.

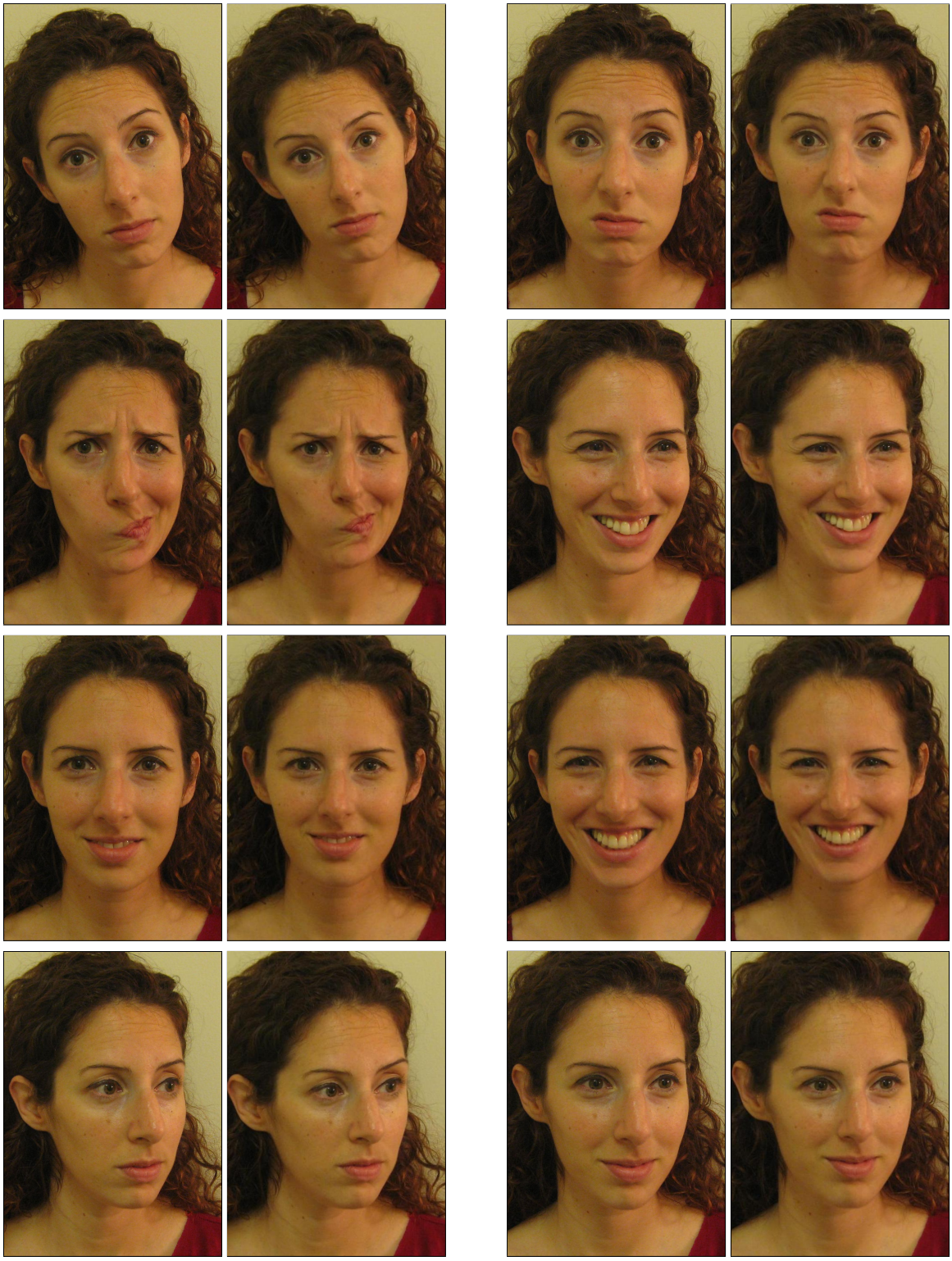


Figure 4.13: Manipulating distances for expressive faces. Each pair contains: original (60cm, left image), our output (480cm, right image).

Figure 4.14: 3D anaglyphs created from a single image. To view, wear red-cyan 3D glasses and zoom the image to fill your screen.

Figure 4.15: Interactive editing, in which sliders control the resulting warp field. (See video and demo on the project page.)

Chapter 5

Conclusion

Photos have become ubiquitous and a natural part of everyday life. Photography and image editing is not practiced in ivory towers, it is a form of self expression and is universally accessible in the age of camera equipped smartphones. Thus our goal is timely: to democratize image editing by empowering novice users to improve their photos with sophistication that matches that of professionals, coupled with simple interfaces. In this thesis we identified **three major directions** of research that can achieve the elusive goal of combining sophistication and simplicity.

We provide an example from each research direction, and yet there remain many more to explore. Single-click **smart selection** is a powerful mask selection paradigm, and the current algorithm can be further improved to make it more robust for a broader range of imagery. Also, runtime improvements that will make the algorithm real-time are required for real-world usage. While single-click interaction is minimal, there are other novice friendly selection mechanisms. In one extreme form, a user could use voice commands such as "select the yellow shirt" to perform a zero-click, voice based selection.

In the realm of distractor removal there are many pending improvements. For example, we would like to add personalization to the system, allowing the removal of distractors according to the taste and aesthetics of a specific user. Removal of distracting photo elements was presented as an example of algorithms with **"high-level" goals**. Many such algorithms were, are, and should be further created. One only needs to look at online Photoshop tutorials for inspiration. Many contain high-level goals such as *"balance lighting perfectly when compositing elements"*, *"how to*

swap heads in Photoshop" or *"design a highbrow horror-movie poster"*. These tasks typically take between a few minutes to an hour or more to complete, and almost all of them can, with effort, be turned into goal-specific algorithm.

We presented heads as one example of an object class which is important enough to warrant **domain specific algorithms**. More such classes exist, and tackling a narrower domain allows the creation of more powerful tool. Specific algorithms should be created for people, furniture, vehicles, landscapes, cityscapes, and for any other element which is common in photos, either as a physical element (e.g. cats) or a meta-element (e.g. beach). As an example, clothes are a common element in photos, thus incorporating domain-specific knowledge to model the range of possible wardrobe items could allow for better segmentation, understanding and editing of clothes. Another example is building facades, which have unique properties such as repeating patterns and large flat surfaces that can be leveraged to create powerful algorithms [73]. Domain specific algorithms extend beyond image editing, and can improve tasks which are unique to the domain. For example, better image facade algorithms will improve automatic map annotation.

This thesis narrows the gap between professional and novice photo editors. We envision a future where the two sets of tools almost converge, once the novice-centric tools are powerful enough to achieve professional grade results, via a simpler and often faster interaction.

Appendix A

Code Snippets

A.1 Torch Implementation

We present the full neural network specification used in Chapter 2. It closely resembles the OpenFace [5] implementation of FaceNet [89], with a few size changes to accommodate our smaller 32x32 patches.

```
local net = nn.Sequential()
net:add(nn.SpatialConvolutionMM(3, 64, 7, 7, 2, 2, 3, 3))
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))
```

Bibliography

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurélien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2274–2282, 2012.

[2] Ansel Adams, Robert Baker, and Alexandre Roberto de Carvalho. *The camera*. Little, Brown Boston, 1980.

[3] Hani Alers, Hantao Liu, Judith Redi, and Ingrid Heynderickx. Studying the effect of optimizing the image quality in saliency regions at the expense of background content. In *Proc. SPIE*, volume 7529, 2010.

[4] Oleg Alexander, Mike Rogers, William Lambeth, Matt Chiang, and Paul Debevec. The digital Emily project: Photoreal facial modeling and animation. In *ACM SIGGRAPH 2009 Courses*, 2009.

[5] Brandon Amos, Bartosz Ludwiczuk, and Mahadev Satyanarayanan. Openface: A general-purpose face recognition library with mobile applications. Technical report, CMU-CS-16-118, CMU School of Computer Science, 2016.

[6] P. Arbeláez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[7] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[8] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *ACM Trans. Graph.*, 26(3), July 2007.

[9] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. In *ACM Trans. on Graphics (Proc. SIGGRAPH)*, New York, NY, USA, 2007. ACM.

[10] Shai Bagon, Oren Boiman, and Michal Irani. What is a good image segment? a unified approach to segment extraction. In *European Conference on Computer Vision*, pages 30–44. Springer Berlin Heidelberg, 2008.

[11] Xue Bai and Guillermo Sapiro. Geodesic matting: A framework for fast interactive image and video segmentation and matting. *International Journal of Computer Vision*, 82(2):113–132, 2009.

[12] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3):24:1–24:11, July 2009.

[13] Hubert C Birnbaum. *Existing Light Photography (The Kodak workshop series)*. Thorsons, 1984.

[14] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 187–194, 1999.

[15] Jeremy S. De Bonet and Paul A. Viola. Texture recognition using a non-parametric multi-scale statistical model. In *1998 Conference on Computer Vision and Pattern Recognition (CVPR '98), June 23-25, 1998, Santa Barbara, CA, USA*, pages 641–647, 1998.

[16] Derek Bradley, Wolfgang Heidrich, Tiberiu Popa, and Alla Sheffer. High resolution passive facial performance capture. *ACM Trans. Graph.*, 29(4):41:1–41:10, July 2010.

[17] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[18] Thomas Brox and Jitendra Malik. Large displacement optical flow: Descriptor matching in variational motion estimation. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 33(3):500–513, March 2011.

[19] Ronnie Bryan, Pietro Perona, and Ralph Adolphs. Perspective distortion from interpersonal distance is an implicit visual cue for social judgments of faces. *PLoS ONE*, 7(9), 09 2012.

[20] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *Proceedings of the 2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 02*, CVPR '05, pages 60–65, Washington, DC, USA, 2005. IEEE Computer Society.

[21] Xavier P Burgos-Artizzu, Matteo Ruggero Ronchi, and Pietro Perona. Distance estimation of an unknown person from a portrait. In *European Conference on Computer Vision (ECCV)*, pages 313–327. Springer, 2014.

[22] Chen Cao, Qiming Hou, and Kun Zhou. Displaced dynamic expression regression for real-time facial tracking and animation. *ACM Trans. Graph.*, 33(4):43:1–43:10, July 2014.

[23] Chen Cao, Yanlin Weng, Stephen Lin, and Kun Zhou. 3d shape regression for real-time facial animation. *ACM Trans. Graph.*, 32(4):41:1–41:10, July 2013.

[24] Chen Cao, Yanlin Weng, Shun Zhou, Yiying Tong, and Kun Zhou. FaceWarehouse: A 3d facial expression database for visual computing. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):413–425, 2014.

[25] Henri Cartier-Bresson. *The Mind's Eye: Writings on Photography and Photographers*. Aperture, 2005.

[26] Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. Palette-based photo recoloring. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 34(4):139:1–139:11, July 2015.

[27] Kai-Yueh Chang, Tyng-Luh Liu, Hwann-Tzong Chen, and Shang-Hong Lai. Fusing generic objectness and visual saliency for salient object detection. In *International Conf. on Computer Vision (ICCV)*, 2011.

[28] Ming-Ming Cheng, Guo-Xin Zhang, N.J. Mitra, Xiaolei Huang, and Shi-Min Hu. Global contrast based salient region detection. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 409–416, June 2011.

[29] Eastman Kodak Co. *The Joy Of Photography*. Da Capo Press, 1991.

[30] Thomas F. Coleman and Yuying Li. An interior trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on Optimization*, 6(2):418–445, 1996.

[31] Emily A. Cooper, Elise A. Piazza, and Martin S. Banks. The perceptual basis of common photographic practice. *Journal of Vision*, 12(5):8, 2012.

[32] Douglas DeCarlo, Dimitris Metaxas, and Matthew Stone. An anthropometric face model using variational techniques. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 67–74, 1998.

[33] Piotr Dollár and C. Lawrence Zitnick. Structured forests for fast edge detection. In *ICCV*, 2013.

[34] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *ICCV*, pages 1033–1038, 1999.

[35] David Eigen, Dilip Krishnan, and Rob Fergus. Restoring an image taken through a window covered with dirt or rain. In *International Conf. on Computer Vision (ICCV)*, pages 633–640, 2013.

[36] Rong-En Fan, Pai-Hsuen Chen, and Chih-Jen Lin. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6:1889–1918, December 2005.

[37] Zeev Farbman, Raanan Fattal, and Dani Lischinski. Diffusion maps for edge-aware image editing. *ACM Trans. Graph.*, 29(6):145:1–145:10, 2010.

[38] P F Felzenszwalb, R B Girshick, D McAllester, and D Ramanan. Object Detection with Discriminatively Trained Part Based Models. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 32(9):1627–1645, 2010.

[39] Ohad Fried, Eli Shechtman, Dan B. Goldman, and Adam Finkelstein. Finding distractors in images. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 1703–1712, June 2015.

[40] Ohad Fried, Eli Shechtman, Dan B. Goldman, and Adam Finkelstein. Perspective-aware manipulation of portrait photos. *ACM Trans. Graph.*, 35(4):128:1–128:10, July 2016.

[41] Dominik Giger, Jean-Charles Bazin, Claudia Kuster, Tiberiu Popa, and Markus Gross. Gaze correction with a single webcam. *IEEE International Conference on Multimedia & Expo*, July 2014.

[42] Jonathan Harel, Christof Koch, and Pietro Perona. Graph-based visual saliency. In *Advances in Neural Information Processing Systems*, pages 545–552. MIT Press, 2007.

[43] Tal Hassner. Viewing real-world faces in 3D. In *International Conference on Computer Vision (ICCV)*, 2013.

[44] Tal Hassner and Ronen Basri. Example based 3d reconstruction from single 2d images. In *Beyond Patches Workshop at IEEE CVPR'06*, June 2006.

[45] Tal Hassner, Shai Harel, Eran Paz, and Roee Enbar. Effective face frontalization in unconstrained images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[46] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 229–238, New York, NY, USA, 1995. ACM.

[47] Xiaodi Hou and Liqing Zhang. Saliency Detection: A Spectral Residual Approach. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2007.

[48] Laurent Itti and Christof Koch. A Saliency-Based Search Mechanism for Overt and Covert Shifts of Visual Attention. *Vision Research*, 40:1489–1506, 2000.

[49] Anil K. Jain and Farshid Farrokhnia. Unsupervised texture segmentation using gabor filters. *Pattern Recogn.*, 24(12):1167–1186, December 1991.

[50] L. Joyeux, O. Buisson, B. Besserer, and S. Boukir. Detection and removal of line scratches in motion picture films. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 548–553, 1999.

[51] Tilke Judd, Krista Ehinger, Frédo Durand, and Antonio Torralba. Learning to predict where humans look. In *International Conf. on Computer Vision (ICCV)*, 2009.

[52] Bela Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290(5802):91–97, March 1981.

[53] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *INTERNATIONAL JOURNAL OF COMPUTER VISION*, 1(4):321–331, 1988.

[54] Yan Ke, Xiaoou Tang, and Feng Jing. The design of high-level features for photo quality assessment. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 419–426, 2006.

[55] I. Kemelmacher-Shlizerman and R. Basri. 3d face reconstruction from a single image using a single reference face shape. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 33(2):394–405, Feb 2011.

[56] Ira Kemelmacher-Shlizerman and Steven M. Seitz. Face reconstruction in the wild. In *International Conference on Computer Vision (ICCV)*, 2011.

[57] Ira Kemelmacher-Shlizerman, Eli Shechtman, Rahul Garg, and Steven M. Seitz. Exploring photobios. *ACM Trans. Graph.*, 30(4):61:1–61:10, July 2011.

[58] A. C. Kokaram. On missing data treatment for degraded video and film archives: A survey and a new bayesian approach. *IEEE Trans. on Image Processing*, 13(3):397–415, March 2004.

[59] Johannes Kopf, Matt Uyttendaele, Oliver Deussen, and Michael F Cohen. Capturing and viewing gigapixel images. In *aCm Transactions on Graphics (TOG)*, volume 26, page 93. ACM, 2007.

[60] Anat Levin, Rob Fergus, Frédo Durand, and William T Freeman. Image and depth from a conventional camera with a coded aperture. *ACM transactions on graphics (TOG)*, 26(3):70, 2007.

[61] Anat Levin, Alex Rav-Acha, and Dani Lischinski. Spectral matting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(10):1699–1712, 2008.

[62] H Liu and I Heynderickx. Studying the added value of visual attention in objective image quality metrics based on eye movement data. In *IEEE International Conf. on Image Processing (ICIP)*, November 2009.

[63] G.A. Lloyd and S.J. Sasson. Electronic still camera, December 26 1978. US Patent 4,131,919.

[64] Rastislav Lukac. *Computational photography: methods and applications*. CRC Press, 2010.

[65] Wei Luo, Xiaogang Wang, and Xiaoou Tang. Content-based photo quality assessment. In *International Conf. on Computer Vision (ICCV)*, Nov 2011.

[66] Rotem Mairon and Ohad Ben-Shahar. *A Closer Look at Context: From Coxels to the Contextual Emergence of Object Saliency*, pages 708–724. Springer International Publishing, Cham, 2014.

[67] L. Marchesotti, C. Cifarelli, and G. Csurka. A framework for visual saliency detection with applications to image thumbnailing. In *International Conf. on Computer Vision (ICCV)*, pages 2232–2239, 2009.

[68] Ran Margolin, Ayellet Tal, and Lihi Zelnik-Manor. What Makes a Patch Distinct? *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1139–1146, June 2013.

[69] David R. Martin, Charless Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):530–549, 2004.

[70] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46. ACM, 1995.

[71] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 746–751, 2013.

[72] Naila Murray, Luca Marchesotti, and Florent Perronnin. Ava: A large-scale database for aesthetic visual analysis. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012. (In press).

[73] P. Musialski, P. Wonka, D. G. Aliaga, M. Wimmer, L. van Gool, and W. Purgathofer. A survey of urban reconstruction. *Computer Graphics Forum*, 32(6):146–177, 2013.

[74] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[75] L Neumann and J Matas. Real-time scene text localization and recognition. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[76] Ren Ng, Marc Levoy, Mathieu Brédif, Gene Duval, Mark Horowitz, and Pat Hanrahan. Light field photography with a hand-held plenoptic camera. *Computer Science Technical Report CSTR*, 2(11):1–11, 2005.

[77] Joseph Nicéphore Niépce. View from the window at le gras. `http://www.hrc.utexas.edu/exhibitions/permanent/firstphotograph/`, 1826. Accessed: 2017-03-20.

[78] Aude Oliva and A Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001.

[79] Anton Orlov. Selecting a portrait lens with correct focal length. Accessed 2016-01-15: `http://petapixel.com/2016/01/04/selecting-a-portrait-lens-with-correct-focal-length/`, 2016.

[80] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, Jan 1979.

[81] Pietro Perona. A new perspective on portraiture. *Journal of Vision*, 7:992–992, 2007.

[82] Pietro Perona. Far and yet close: Multiple viewpoints for the perfect portrait. *Art & Perception*, 1(1-2):105–120, 2013.

[83] T. Randen and J. H. Husoy. Filtering for texture classification: a comparative study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):291–310, Apr 1999.

[84] Ramesh Raskar and Jack Tumblin. *Computational photography: mastering new techniques for lenses, lighting, and sensors*. AK Peters, Ltd., 2009.

[85] Erik Reinhard, Wolfgang Heidrich, Paul Debevec, Sumanta Pattanaik, Greg Ward, and Karol Myszkowski. *High dynamic range imaging: acquisition, display, and image-based lighting*. Morgan Kaufmann, 2010.

[86] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004.

[87] Michael Rubinstein, Diego Gutierrez, Olga Sorkine, and Ariel Shamir. A comparative study of image retargeting. *ACM Transactions on Graphics*, 29(6):160:1–160:10, December 2010.

[88] Jason M Saragih, Simon Lucey, and Jeffrey Cohn. Face alignment through subspace constrained mean-shifts. In *International Conference on Computer Vision (ICCV)*, September 2009.

[89] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, June 2015.

[90] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 118–126, Dec 2015.

[91] EP Simoncelli and WT Freeman. The Steerable Pyramid: A Flexible Architecture For Multi-Scale Derivative Computation. *IEEE International Conf. on Image Processing (ICIP)*, 1995.

[92] Sara L. Su, Frédo Durand, and Maneesh Agrawala. De-emphasis of distracting image regions using texture power maps. In *Applied Perception in Graphics & Visualization*, pages 164–164, New York, NY, USA, 2005. ACM.

[93] Bongwon Suh, Haibin Ling, Benjamin B. Bederson, and David W. Jacobs. Automatic thumbnail cropping and its effectiveness. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, UIST '03, pages 95–104, New York, NY, USA, 2003. ACM.

[94] Xiaoou Tang, Wei Luo, and Xiaogang Wang. Content-Based Photo Quality Assessment. *IEEE Transactions on Multimedia (TMM)*, 2013.

[95] Flora Ponjou Tasse and Neil Dodgson. Shape2vec: Semantic-based descriptors for 3d shapes, sketches and images. *ACM Trans. Graph.*, 35(6):208:1–208:12, November 2016.

[96] Robert Tibshirani. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[97] Wai-Shun Tong, Chi-Keung Tang, Michael S. Brown, and Ying-Qing Xu. Example-based cosmetic transfer. *Computer Graphics and Applications, Pacific Conference on*, 0:211–218, 2007.

[98] LedyardR Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[99] Erik Valind. *Portrait Photography: From Snapshots to Great Shots*. Pearson Education, 2014.

[100] Manik Varma and Andrew Zisserman. Texture classification: Are filter banks necessary? In *2003 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2003), 16-22 June 2003, Madison, WI, USA*, pages 691–698, 2003.

[101] Paul Viola and Michael Jones. Robust Real-time Object Detection. In *International Journal of Computer Vision (IJCV)*, 2001.

[102] Daniel Vlasic, Matthew Brand, Hanspeter Pfister, and Jovan Popović. Face transfer with multilinear models. *ACM Trans. Graph.*, 24(3):426–433, July 2005.

[103] Jure Žbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *J. Mach. Learn. Res.*, 17(1):2287–2318, January 2016.

[104] T. Weise, B. Leibe, and L. Van Gool. Fast 3d scanning with automatic motion compensation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 861–868, 2007.

[105] Jianzhou Yan, Stephen Lin, Sing Bing Kang, and Xiaoou Tang. Learning the change for automatic image cropping. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[106] Fei Yang, Lubomir Bourdev, Eli Shechtman, Jue Wang, and Dimitri Metaxas. Facial expression editing in video using a temporally-smooth factorization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 861–868, 2012.

[107] Fei Yang, Eli Shechtman, Jue Wang, Lubomir Bourdev, and Dimitris Metaxas. Face morphing using 3d-aware appearance optimization. In *Proceedings of Graphics Interface (GI'12)*, pages 93–99, 2012.

[108] Fei Yang, Jue Wang, Eli Shechtman, Lubomir Bourdev, and Dimitri Metaxas. Expression flow for 3d-aware face component transfer. *ACM Trans. Graph.*, 30(4):60:1–60:10, July 2011.

[109] C Lawrence Zitnick and Piotr Dollár. Edge Boxes: Locating Object Proposals from Edges. In *European Conf. on Computer Vision (ECCV)*, 2014.