The University of New South Wales
School of Computer Science and Engineering

SENG4920 (Thesis Part B) Report

Applications of Graphical Models:

Image Inpainting using Belief-Propagation

Julian McAuley
jjmc046@cse.unsw.edu.au

February 26, 2007

Contents

List of Figures

List of Tables

Abstract

In this thesis, we seek to solve image processing problems using graphical models. The main problem we shall deal with is known as image 'inpainting' – that is, trying to restore 'scratched', or otherwise corrupted regions of an image. We will show that techniques known as *nonparametric belief-propagation* can be used in this setting – an approach that we believe has not previously been applied to this problem. By using such techniques, we will attempt to inpaint images faster than any existing techniques.

Problems in the field of image processing are often approached using graphical models. Consequently, many models, and many inference algorithms have been developed. Two such classes of algorithms are *belief-propagation* and *gradient-ascent*. Yet while both of these algorithms seek to solve the same optimisation problem, the conditions under which they can be applied are very different.

Gradient-based approaches are typically preferred when dealing with images. This occurs due to the fact that images are typically represented using high-order models, for which belief-propagation techniques tend to be very expensive. Gradient-based approaches are often faster, in spite of the fact that they may require several thousand iterations to converge, compared to belief-propagation approaches which converge in very few iterations.

In this thesis, we will try to address the shortcomings of belief-propagation to render it applicable to the problem of image inpainting. We will present results to demonstrate that by using such techniques, we are able to inpaint images significantly faster than existing gradient-based approaches.

Acknowledgements

I would like to thank my supervisor for his advice, and for his comments on the first draft. I would also like to thank August Dvorak for his keyboard layout, and Donald Knuth for his typesetting system. Without their work, the physical realisation of this document would likely have been impossible.

Chapter 1

Introduction

As we mentioned in our abstract, the purpose of this thesis is to solve the problem of image inpainting using graphical models. In order to do so, we will need to build upon and modify a great deal of work in the fields of graphical models, image processing, as well as statistics. Since this document is intended to be relatively self contained, a large part of this thesis shall be spent explaining the mathematical concepts involved.

To begin with, however, it is worth briefly defining the ideas of image inpainting, and graphical models, in order to motivate the material that follows.

1.1 Image Inpainting

Image inpainting [BSCB00] refers to the process of restoring 'hidden' regions in an image. The most common example of such a region might be a 'scratch' – inpainting allows us to infer what is 'behind' this scratch, thereby producing a scratch-free version of the image.

Two examples of this approach are shown in figure 1.1.

In principle, the model we develop for inpainting images could easily be used for other tasks, such as image denoising (i.e. filtering noise such as electrical interference from an image), or super-resolution (i.e. producing a high-resolution version of an image using a low-resolution one). Image inpainting should therefore be thought of as a verification for our method, rather than as the sole purpose of this thesis. Image inpainting also happens to be an application for which the techniques we shall describe result in a particularly fast solution.

As we suggested in our abstract, the purpose of this thesis will be to develop *fast* algorithms for image inpainting. While neither image inpainting, nor nonparametric belief-propagation

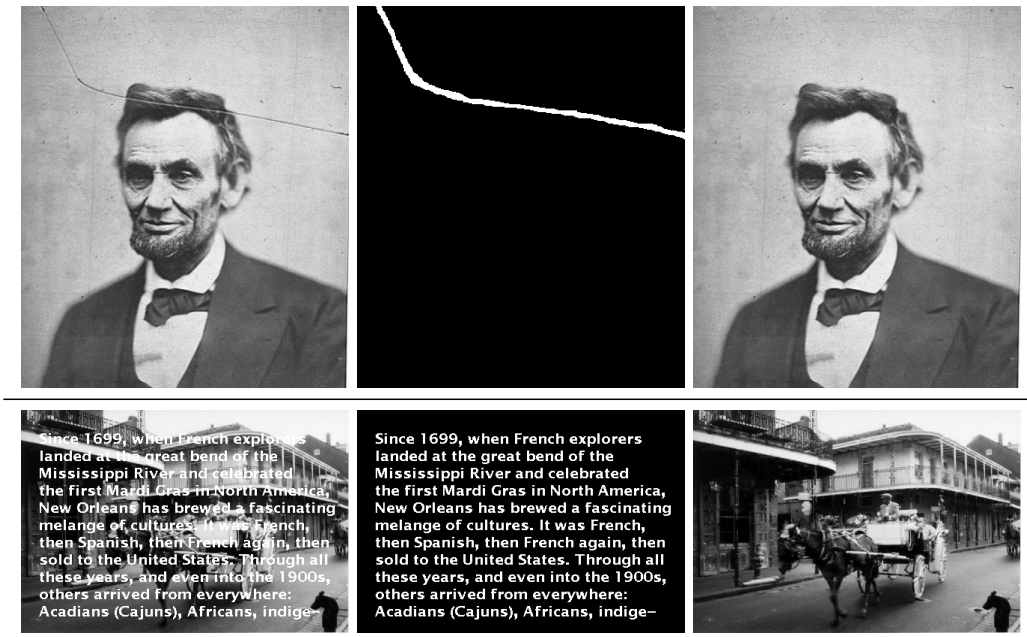


Figure 1.1: Left to right (top and bottom): the image to be inpainted; the region corresponding to the 'corrupted' part of the image we wish to restore (in white); the image restored using the technique to be described.

are new fields, we believe that the latter has never before been applied to the former. While our technique will not inpaint images at the state-of-the-art level (in terms of signal-to-noise ratio), it will produce good results significantly faster than other methods. This speed increase may give rise to new inpainting applications (such as removing the watermarks from frames in a movie). It can also be seen as a guide to develop similar algorithms to solve other image-processing problems.

To make the purpose of this thesis more intuitively obvious, the graph in figure 1.2 shows how existing techniques compare to ours. We aim at a method which exhibits reasonably fast performance, yet is still complex enough to inpaint images well.

1.2 Graphical Models

Graphical models [Bis06] are nothing more than tools which can be used to solve a variety of optimisation problems. Graphical models are beneficial whenever the variables over which we are trying to optimise exhibit certain conditional independence properties.

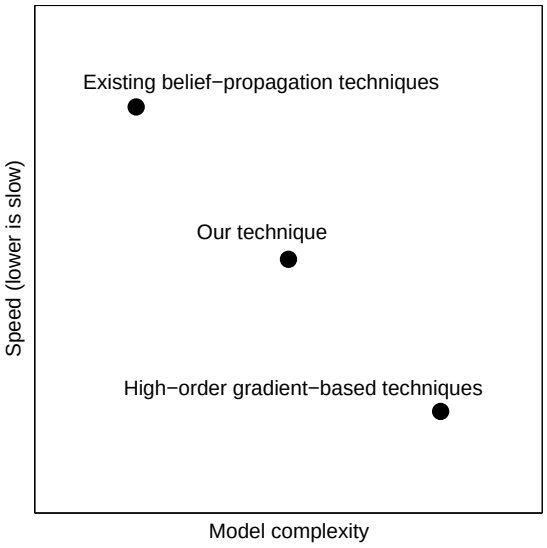


Figure 1.2: The above plot shows that there already exist low-complexity belief-propagation techniques (top left). In comparison, high-complexity gradient-based techniques (bottom right) tend to be much better performing, but also far slower. The purpose of this thesis is to develop fast algorithms, that still exhibit reasonable performance (centre).

1.3 Since Our Thesis Proposal

In the original proposal for this thesis, we stated that our intentions were to explore the possible problems which can be solved using graphical models, and to implement the algorithms required to solve them. At that stage, we were not sure about exactly *which* applications we might explore, nor about which algorithms these problems may require us to implement – image processing was only one of the many possibilities we suggested.

At this stage, we have decided that image processing shall comprise the core material in this thesis. This decision was made since image processing is an area with which we have some familiarity, meaning that less time needed to be spent exploring a new field. We also thought that it would be better to explore this one topic at a reasonable level of depth, rather than to treat many topics shallowly.

Of course, for all of the material we present, we shall make an effort to provide references to other possible applications of the same techniques.

1.4 This Document

Here we shall present a brief overview of the remaining sections of this document.

In **Chapter 2**, we shall present a background of the topics studied in this thesis. Before we are able to present the topic of graphical models, we will need to introduce notions such as conditional independence, and explain how these notions relate to images. In this chapter, we will also give a brief overview of some of the inference algorithms commonly used for image processing. Many of the ideas in this section have already been covered in our proposal – the main difference being that this time we will be discussing them in terms of their relationship to image processing.

In order to apply graphical models to an image processing problem, we need to define an *image prior*. This will be done in **Chapter 3**. We will define the state-of-the-art priors that are currently being used to solve similar problems, and develop a model of this form.

To use nonparametric belief-propagation, we will have to make a number of modifications to the prior presented in Chapter 3. In **Chapter 4**, we will present the modifications required.

In **Chapter 5**, we will further describe the inference algorithms we have used for image processing. We will give a full description of nonparametric belief-propagation, and explain how the prior defined in Chapter 4 can be used in this setting.

In **Chapter 6**, we will present the main results of our thesis. These will be further discussed in **Chapter 7**, in which we shall also look at possible extentions, and analyse possible shortcomings of our technique.

Finally, Appendix A will cover various proofs and further details that are not included in previous chapters. In Appendix B we will include a conference version of this thesis, which has

been submitted to the International Conference of Machine Learning (ICML2007).

Further Reading

The first paper to formally introduce the idea of image inpainting is the paper by the same title [BSCB00]. The current 'state-of-the-art' work in this area is appears to be the *Field of Experts* model developed in [RB05] (this is essentially the model that we shall be adapting).

Image denoising (another application we mentioned) is also covered in [RB05], and more recently (by the author) in [MCSF06]. Super-resolution (among other topics) is covered in [FPC00].

As for the topic of graphical models (which are a common theme in the above references), [Bis06] provides a fairly complete discussion of the topic. Two other important texts (though currently unpublished) are [Jor] and [KF].

Chapter 2

Background

In this chapter, we will present the mathematical background required to introduce the topic of graphical models (sometimes referred to as Markov Random Fields). As we have mentioned, graphical models are nothing more than tools to aid us in performing statistical inference; when dealing with image inpainting, this simply means to determine the correct values of hidden or corrupted pixels.

A significant proportion of this chapter will cover material already presented in our thesis proposal. The main difference here is that we shall make more of an effort to relate these ideas to images.

2.1 Conditional Independence

Before we are able to properly introduce the topic of graphical models, we must establish a clear notion of conditional independence. In mathematical terms, we say that two random variables, A and B, are *independent*, if $P(A = a, B = b) = P(A = a)P(B = b)$ (for all a, b). Alternatively, we would say that A and B are *conditionally independent* given C, if $P(A = a, B = b|C = c) = P(A = a|C = c)P(B = b|C = c)$ (for all a, b, c). The difference between these two notions is very important – the first is simply saying that knowing the value of A tells us nothing about the value of B, while the second is saying that *if we know* C, then knowing A tells us nothing *more* about B – that is, even though A and B may be dependent, their dependence is entirely 'explained' by C. These two ideas are summarised below.

A and B are independent: $P(A = a, B = b) = P(A = a)P(B = b)$ (for all a, b).

A and B are conditionally independent, given C: $P(A = a, B = b|C = c) = P(A = a|C = c)P(B = b|C = c)$ (for all a, b, c).

To make this distinction more clear, consider the following example involving umbrellas: obviously, if I carry an umbrella, it is highly likely that you will carry one also (assuming that we live in the same city). Therefore, we would assert that these two random variables are *not* independent. However, if we know that it is raining, then knowing if I carry an umbrella probably tells us nothing about whether or not you do. Therefore we would assert that these two random variables are *conditionally* independent, if we know the weather.

In the above case, we would say that the relationship between us carrying an umbrella and the weather is *causal* – it makes sense to say that I am carrying an umbrella *because* it is raining, but not the other way around. In some cases, however, no such relationship exists – for example, the weather in all Australian cities is related, but it is not very meaningful to say that the weather in one city 'caused' the weather in another.

Further examples, which will relate these ideas to images, shall be presented in section 2.1.1.

2.1.1 Graphical Representation of Conditional Independence

In fact, our notion of conditional independence lends itself very naturally to a graphical representation. We can simply use nodes to denote each of the random variables in our graph, and use edges between two nodes to indicate that there is some direct relationship between them. Using such a representation, we present the above two examples as graphs in figure 2.1.

The graph on the left in figure 2.1 is *directed*; such a graph can be used to form a *directed graphical model*. In this case, the directed arrows indicate a causal relationship between variables, but this needn't be the case; this type of graphical model may be used in other instances in which the relationship between variables in not symmetric. Alternately, the graph on the right is *undirected*, resulting in an *undirected graphical model* – indicating a mutual relationship between the variables.

For images, it makes sense to say that two nearby pixels are *mutually* related to each other; it does not make sense to say that one pixel 'caused' another's value. Therefore, the remainder of this section will be concerned only with *undirected* graphical models. However, the results we shall present are nearly the same in either case. Further detail about directed graphical models may be found in [Jor].

Relation to Images

It is possible to establish similar notions of conditional independence for images. For example, we would certainly say that the colours (or the grey-levels of) two nearby pixels are highly dependent upon each other [GG84, MCSF06]. That is (for example), a white pixel will probably be surrounded by other (nearly) white pixels. However, if we already know the grey-levels of a pixel's neighbours, then knowing the grey-levels of other nearby pixels will probably tell us very little. Hence we might conclude that the grey-level of a certain pixel only directly depends on the grey-levels of its neighbours.

The graph in figure 2.2 represents one possible notion of the dependencies between pixels in

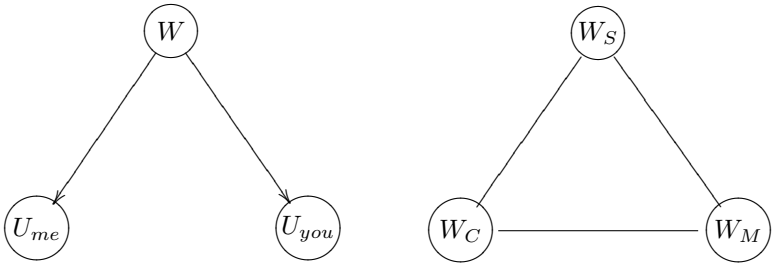


Figure 2.1: The graph on the left is a *directed* graphical model – the node W represents the weather; U_{me} and U_{you} represent whether or not we carry umbrellas. In this case, the directed arrows indicate a causal relationship between these variables (although other types of relationship are possible). The graph on the right is an *undirected* graphical model, in which W_S, W_C, and W_M may represent the weather in Sydney, Canberra, and Melbourne. In this case, no such causal relationship exists.

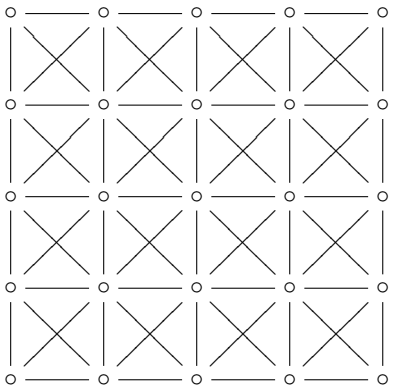


Figure 2.2: The above graph represents dependencies between pixels in an image. This graph asserts that a pixel only has a *direct* relationship to its nearest neighbours, and its diagonal neighbours.

an image. While this is not the only model possible (indeed, it may be reasonable to say that a pixel is *directly* related to its neighbours two or more units away [MCSF06, RB05]), it is the model we shall be considering throughout most of this document.

2.1.2 Interpreting a Graph

In order to make the conditional independencies implied by the graphs presented more explicit, we must first introduce some notation for graphs. The notation we use is standard, and is given as follows:

$G = (\mathbf{x}, E)$: A graph.

$\mathbf{x} = (x_1 \dots x_N)$: The set of vertices/nodes in our graph (where each vertex/node

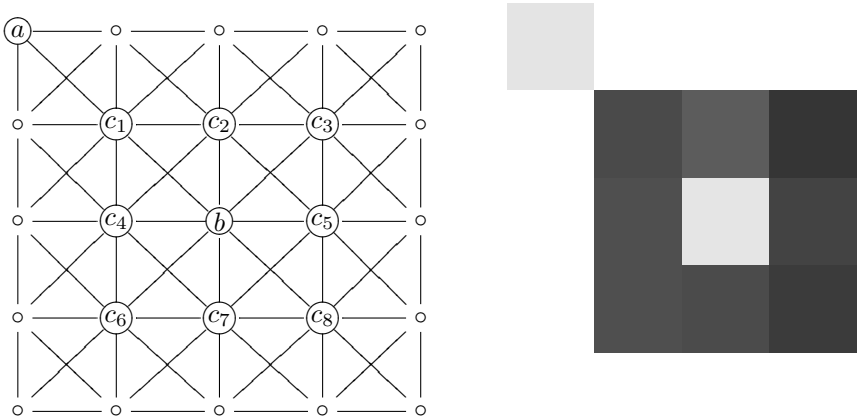


Figure 2.3: In the graph on the left, we can see that there is no path from the node a to the node b which doesn't pass through one of the nodes in $\mathbf{x}_c = (c_1 \dots c_8)$. Hence we conclude that a and b are conditionally independent given C. On the right, we show 'observed' pixels in a real image – the two light-gray pixels represent the unknowns (a and b).

represents a random variable). Although this is usually denoted by V, we use $\mathbf{x}$ here to denote the fact that each of our vertices is actually a random variable.

$E = (e_1 \dots e_M) \subseteq (\mathbf{x} \times \mathbf{x})$: The set of edges in our graph.

With this notation in mind, we can rephrase 'conditional independence' as follows: for three sets of vertices $\mathbf{x}_A, \mathbf{x}_B$, and $\mathbf{x}_C$ ($\subset \mathbf{x}$),[1] we say that $\mathbf{x}_A$ and $\mathbf{x}_B$ are conditionally independent, given $\mathbf{x}_C$, if there is no path from the vertices in $\mathbf{x}_A$ to $\mathbf{x}_B$ which doesn't pass through $\mathbf{x}_C$. Another way of putting this is to say that if we were to *remove* the vertices in $\mathbf{x}_C$ (and the corresponding incident edges) from G, then we would form (at least) two disjoint graphs – one of which contained all of the vertices in $\mathbf{x}_A$, and another of which contained all of the vertices in $\mathbf{x}_B$. This is seen in figure 2.3.

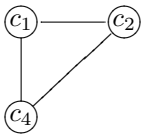
2.2 Graphical Models

These notions of conditional independence underpin the topic graphical models. However, the graph itself is only half of the story – before we are able to talk meaningfully about an optimisation problem, we must first discuss how probabilities are actually defined in this setting.

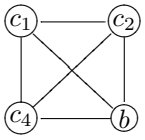
To do so, we must first introduce some more notation. Firstly, for a subset of the vertices in $\mathbf{x}$ (say $\mathbf{x}_c$), we say that these vertices form a *clique*, if for all $x_i, x_j \in \mathbf{x}_c (i \neq j)$, we have that

[1] Here we have adopted the convention that a normal-face (e.g. x_i) denotes a single vertex, whereas a boldface (e.g. $\mathbf{x}_A$) denotes a *set* of vertices (specifically the set $\{x_a | a \in A\}$).

(x_i, x_j) (or (x_j, x_i)) $\in E$. That is, $\mathbf{x}_c$ is a set of vertices, all of which are connected to each other (except for self-loops). Furthermore, we say that a clique is *maximal*, if for any node $x_k \notin \mathbf{x}_c$, we have that $\mathbf{x}_c \cup \{x_k\}$ does *not* form a clique. For example, for the graph in figure 2.3, we would say that the set $\{c_1, c_2, c_4\}$ forms a clique:



However, it is not a *maximal* clique, since we may obtain a larger clique by adding b (and its associated edges):



This clique *is* now maximal, since it it is impossible to add any more vertices and still form a clique.

It is now possible to separate any graph into a set of maximal cliques $\mathcal{C} = (c_1, \dots c_l)$, such that each $c_i \subseteq \mathbf{x}$ is a maximal clique (and, of course, $\bigcup_{i=1}^{l} c_i = \mathbf{x}$). The example in figure 2.3 has a particularly straightforward decomposition – each clique is simply a 2×2 region (i.e. the complete graph K_4), much like the one presented above.

2.2.1 The Hammersley-Clifford Theorem

The *Hammersley-Clifford* theorem [HC71] now states that the probability for any given configuration of a Markov Random Field can be defined entirely in terms of its maximal cliques. Specifically, if our field has cliques $\mathcal{C}$, then the probability for a given configuration of $\mathbf{x}$ is given by

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c). \tag{2.1}$$

Here, each ϕ_c is a *potential function* which acts on the clique c. Although we will define the concept of a potential function more clearly in chapter 3, for the moment we shall simply say that each potential function expresses the 'likelihood' for any assignment to the clique's variables, by returning a (strictly) positive potential.

In equation (2.1), the term Z is simply a normalisation constant, ensuring that the sum of the probabilities for all possible configuration sums to 1. That is, Z may be computed according to

$$Z = \sum_{\mathbf{x}} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c) \quad \text{(here we are summing over } \textit{all possible} \text{ assignments to } \mathbf{x}). \tag{2.2}$$

Thus, we are (in principle) able to assign a non-zero probability to every possible configuration of our field. This theorem forms the basis of both of the inference algorithms to be presented. In general, however, the exact value of Z may be very difficult (if not impossible) to compute – since it may involve summing over *every* possible assignment to the field, which may be a prohibitively large number. As we shall see, however, we are not usually concerned with actually *computing* the probability, but only *maximising* it, meaning that we will never need to compute this normalisation constant explicitly.

2.2.2 Inference in MRFs

Unlike our proposal, in which we presented a variety of inference algorithms which may be relevant to topics in graphical models, in this section we will focus specifically on those that are relevant to the topic of image inpainting. The two classes of algorithms we shall deal with are *gradient-ascent* and *belief-propagation*.

As an aside, it is worth making a brief mention of why the other inference algorithms we mentioned in our proposal are no longer relevant to this topic. Consider, for example, Gibbs sampling (we will not explain Gibbs sampling in this document – see [CG92] for more detail): for each pixel, Gibbs sampling requires us to compute the potential function (ϕ) for every clique containing that pixel, for every possible assignment to that pixel. Even if we assume a clique size of 2×2, and if we limit ourselves to grayscale images with 256 distinct gray-levels, we will still have to evaluate the potential $256 \times 4 = 1024$ times[2] *just to update a single pixel*. Since a Gibbs sampler may require several *thousand* iterations to converge, such algorithms are often simply too expensive when dealing with real images.

Alternately, gradient-ascent algorithms allow us to avoid the need to deal with a large number of discrete gray-levels – indeed, such algorithms simply treat each variable as continuous.[3]

As we shall see, the belief-propagation algorithm we shall present actually suffers from many of the same problems as the Gibbs sampler mentioned above. For this reason, high-order belief-propagation algorithms are typically not used when dealing with images (although some exceptions exist [FPC00, LRHB06]).[4] Therefore, we will need to use a variant on this algorithm, known as *nonparametric* belief-propagation [SIFW03], in order to render these algorithms practical for our purposes.

Gradient-Ascent

Gradient Ascent algorithms are commonly used in many areas of mathematics and statistics when maximum-likelihood solutions cannot be computed exactly. They are also commonly used in graphical models applications [MCSF06, RB05].

[2] This number arises since we have 256 gray levels, and each pixel is contained by 4 distinct cliques.

[3] Actually, it is not impossible to deal with continuous variables when using a Gibbs sampler, except that doing so would require us to be able integrate the potential function, as opposed to differentiating it. Unfortunately, the families of potential functions we shall deal with do not have a simple integral.

[4] Of course, *low*-order belief propagation (e.g. with cliques of size 2) may still be possible, but is unlikely to inpaint images well.

Algorithm 1: Gradient-ascent

choose an initial estimate $\mathbf{x}^0$ (i.e. choose an x_i for each $v_i \in V$)
choose a 'learning rate', δ.
for $i \in (1 \dots I)$ **do**
  $\mathbf{x}^i = \mathbf{x}^{i-1} + \delta \frac{\partial}{\partial \mathbf{v}} \log(P(V = V'))$
end
return $\mathbf{x}^I$

Figure 2.4: Pseudocode for the gradient-ascent algorithm

In our setting, gradient ascent consists of computing the derivative of the probability function in equation (2.1) (or the derivative of its logarithm), with respect to each of our variables. Here we have that

$$\frac{\partial}{\partial x_i} \log(p(\mathbf{x})) = \frac{\partial}{\partial x_i} \log(\frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c)) \tag{2.3}$$

$$= \frac{\partial}{\partial x_i} (\sum_{(c \in \mathcal{C} | x_i \in c)} \log(\phi_c(\mathbf{x}_c)) - \log(Z)) \tag{2.4}$$

$$= \sum_{(c \in \mathcal{C} | x_i \in c)} \frac{\partial}{\partial x_i} \log(\phi_c(\mathbf{x}_c)) \tag{2.5}$$

(where in the second line we have restricted the set of cliques we are summing over to be only those which contain x_i). Further simplification may be possible – in fact, the form of the ϕ_c's is often chosen to come from an exponential family, thus allowing us to eliminate the logarithm from the above expression [GG84]. It should also be noted that the normalisation constant (Z), which we suggested may be intractable to compute (see section 2.2.1) disappears from the above expression after differentiation.

Pseudocode for the gradient ascent algorithm is given in figure 2.4 (note that here we use $\frac{\partial}{\partial \mathbf{v}}$ to denote a *vector* of partial derivatives). There may be many variations on the specific implementation – for example the 'for' loop may be replaced by a 'while' loop, which would exit once the gradient was sufficiently small; similarly, the learning rate, δ could be changed after each iteration.

Belief-Propagation – Exact Inference

Our problem of inference can also be solved using message passing techniques. The mathematics behind some of these techniques is fairly detailed, so we shall only cover them superficially now, and defer a full explanation until chapter 5. For further information, a good summary paper on these results is [AM01].

We have already discussed the idea that a set of connected vertices may form a clique. Of course, in a fully-connected graph, there will also be nodes that are shared between several cliques. The set of nodes shared between two cliques is usually called their *separator set* (we will use $S_{i,j}$ to denote the separator set $\mathbf{x}_i \cap \mathbf{x}_j$) – an example is shown in figure 2.5.

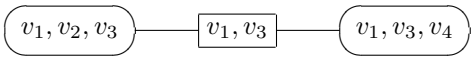


Figure 2.5: Two cliques (left, right), and the separator set (centre) corresponding to their intersection.

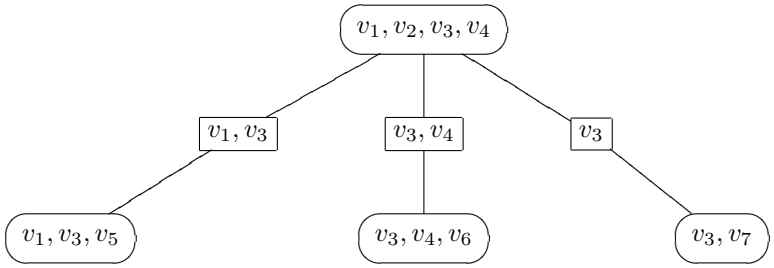


Figure 2.6: One possible spanning tree for our set of cliques.

Now consider a new graph in which each of our cliques becomes a node. It is possible to form a spanning tree for this graph, in which edges are formed between those cliques with non-empty intersection. For example, consider a graph containing vertices $\{v_1 \dots v_7\}$ and cliques $\{\{v_1, v_2, v_3, v_4\}, \{v_1, v_3, v_5\}, \{v_3, v_4, v_6\}, \{v_3, v_7\}\}$ – one possible spanning tree for this new graph graph (together with its separator sets) is shown in figure 2.6.

Of course, it should be noted that we have not connected *all* cliques with non-empty intersection – only enough to form a spanning tree. However, the graph in figure 2.6 has a very important property – for any cliques that *aren't* connected (i.e. $\mathbf{x}_i$ and $\mathbf{x}_j$, with $\mathbf{x}_i \cap \mathbf{x}_j = S_{i,j} \neq \emptyset$), any clique which lies on a path between them also contains their intersection (i.e. if $(\mathbf{x}_i, \mathbf{x}_{p_1} \dots \mathbf{x}_{p_n}, \mathbf{x}_j)$ is a path between C_i and C_j, then $S_{i,j} \subset \mathbf{x}_{p_i}$, $i \in \{1 \dots n\}$).[5] Such a spanning tree is known as a *junction-tree*, which gives rise to the *junction-tree property* – a property required in order to perform exact inference using message passing techniques [AM00, AM01]. It is important to note that it is not always possible to form a junction-tree – for example, it is impossible to form a junction-tree using the graph in figure 2.7 (we will deal with this situation later).

However, assuming that a junction-tree exists, it is possible to perform exact inference using

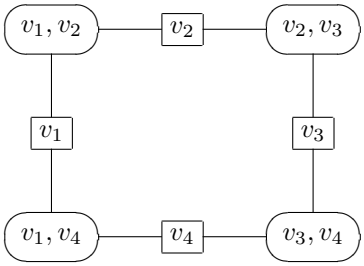


Figure 2.7: It is impossible to form a junction-tree from the set of cliques above.

[5]The subset here is indeed proper (i.e $\subset$ rather than $\subseteq$), since we are only concerned with *maximal* cliques.

Algorithm 2: The junction-tree algorithm

```
while some cliques have not received messages from all of their neighbours do
    for c ∈ C do
        if x_c has received messages from all of its neighbours then
            for x_k ∈ Γ_{x_c} do
                compute and send M_{c→k} ;        /* if it hasn't been sent
                already */
            end
        end
        else if x_c has received messages from all neighbours except one x_k then
            compute and send M_{c→k}
        end
    end
    for c ∈ C do
        compute D_c(x_c)
    end
end
return D ;          /* the vector of all marginal probabilities */
```

Figure 2.8: Pseudocode for the junction-tree algorithm

a message passing algorithm. We assume that we have already defined a potential function ϕ_i for each clique $\mathbf{x}_i$ (which we will call that clique's *local* distribution). Now, the message that $\mathbf{x}_i$ sends to $\mathbf{x}_j$ (denoted $M_{i\to j}(S_{i,j})$, a function of the variables in $\mathbf{x}_i \cap \mathbf{x}_j$) is defined as

$$M_{i\to j}(S_{i,j}) = \sum_{\mathbf{x}_i \setminus \mathbf{x}_j} \phi_i(\mathbf{x}_i) \prod_{\mathbf{x}_k \in (\Gamma_{\mathbf{x}_i} \setminus \{\mathbf{x}_j\})} M_{k\to i}(S_{k,i}) \quad (\Gamma_{\mathbf{x}_i} \text{ denotes all neighbours of } \mathbf{x}_i). \tag{2.6}$$

Once all messages have been sent, the final distribution for the clique $\mathbf{x}_i$ is defined as

$$D_i(\mathbf{x}_i) = \phi_i(\mathbf{x}_i) \prod_{\mathbf{x}_k \in \Gamma_{\mathbf{x}_i}} M_{k\to i}(S_{k,i}). \tag{2.7}$$

The distribution of each D_i now corresponds to the true marginal for that clique.[6] The exact order in which messages are passed is still slightly unclear from this informal specification – so we present the algorithm in pseudocode in figure 2.8. Although it is not stated explicitly, it can be seen from the pseudocode that messages will propagate inwards from the leaves to the 'root' of the tree, and then proceed outwards toward the leaves.

It may already be clear that this algorithm gives us a more powerful result than the gradient-ascent algorithm – while gradient-ascent only computes an estimate for each node, this algorithm is able to compute a full marginal density (and moreover, do so exactly).

[6] Proving this fact is beyond the scope of this document – see [AM00].

Algorithm 3: Loopy belief-propagation

```
for i ∈ {1...I} ;                                   /* for each iteration */
do
    for c, k ∈ C|k ∈ Γ_c ;        /* for all pairs of neighbouring cliques
    (usually ordered randomly) */
    do
        compute and send M_{c→k}
    end
end
for c ∈ C do
    compute D_c(x_c)
end
return D ;            /* the vector of all marginal probabilities */
```

Figure 2.9: Pseudocode for the loopy belief-propagation algorithm

Belief-Propagation – Approximate Inference

As we have already mentioned, it is not always possible to form a junction-tree from any set of cliques. For example, the cliques in figure 2.3 do not obey this property, although such a structure is used in many applications [GG84]. In these cases, an *approximate* method exists, known as *Loopy Belief-Propagation* [AM00, YFW00]. This algorithm is very similar to the junction-tree algorithm, except that we connect *every* pair of cliques with non-empty intersection. Also, messages may need to be passed for several iterations before they converge.

Pseudocode for loopy belief-propagation is presented in figure 2.9. The main differences here are that the message-passing order is *random*, and that we are required to choose a number of iterations for which to run the algorithm.

Unfortunately, very few theoretical results exist for loopy belief-propagation [IFW04] – it is often not known whether the algorithm will converge to produce the correct marginals, or worse, whether it will converge at all. Despite this shortcoming, there have been many successful applications of loopy belief-propagation [YFW00]. In order to perform image inpainting, we shall require both forms of belief-propagation.

Nonparametric Belief-Propagation

As we have already suggested, belief-propagation algorithms are highly desirable due to the fact that they typically converge in far fewer iterations than similar gradient-based approaches. However, although it may not be immediately apparent from the pseudocode shown, the belief-propagation algorithms we have presented may be so computationally expensive as to negate this benefit.

For example, when dealing with images, we typically deal with cliques of size at least 2×2; in order to compute the messages in the pseudocode in figures 2.8 and 2.9, we would first need

to compute the local distribution ($\phi_i(\mathbf{x}_i)$). Even if we restrict ourselves to grayscale images, with only 256 possible gray-levels, we would still need to consider 256^4 possible configurations of each clique in order to compute $\phi_i(\mathbf{x}_i)$. Such an approach is clearly impractical.

Nonparamateric belief-propagation [SIFW03] tries to deal with these problems by expressing the potential function in some nonparametric form. That is, rather than compute the response for all 256^4 possible configurations, we may say (for example) that the response takes the form of a Gaussian distribution – the probability for every configuration is now captured by the mean vector and covariance matrix for this Gaussian.

This allows us to replace the sum in equation (2.6) with an integral. That is, our new messages take the form

$$M_{i\to j}(S_{i,j}) = \int_{\mathbf{x}_i \setminus \mathbf{x}_j} \phi_i(\mathbf{x}_i) \prod_{\mathbf{x}_k \in (\Gamma_{\mathbf{x}_i} \setminus \{\mathbf{x}_j\})} M_{k\to i}(S_{k,i}). \tag{2.8}$$

Usually, in order to apply this approach, we assume that the potential functions take the form of a *Gaussian mixture* – i.e. a weighted sum of several Gaussian density functions [SIFW03]. This form is assumed because both the product and the integral in equation (2.8), when applied to a Gaussian mixture, result in a new Gaussian mixture.

Assuming that we are able to approximate our potential function using such a Gaussian mixture, it is now possible to perform very fast inference in this setting. Actually *finding* this approximation may be difficult – especially if the potential function is high dimensional; the approach used in [SIFW03] is only used to learn at most three-dimensional potential functions – smaller than those typically required when dealing with images.

However, we will show that it *is* possible to use this technique when dealing with images. We shall present the nonparametric belief-propagation algorithm, as well as the specific approximation we derive in more detail in chapter 3.

2.3 Probability Distributions

Although it is somewhat difficult to motivate at this stage, we will need to deal with some theory about probability distributions.

We have already suggested that our potential function may take the form of a Gaussian mixture. However, what we have not yet mentioned is that the potential function we shall be approximating using a Gaussian mixture will take the form of a Student's T-distribution [Gos42].

It is therefore important that we highlight the difference between these two distributions. The Student's T-distribution was originally suggested when we wish to approximate a distribution in the presence of outliers [Gos42].

In figure 2.10 we show a histogram of a synthesised data set. This data set appears to roughly follow the shape of a normal distribution, except for a few outlying points. If we try to approximate this distribution using a Gaussian, the variance is overestimated in order to compensate

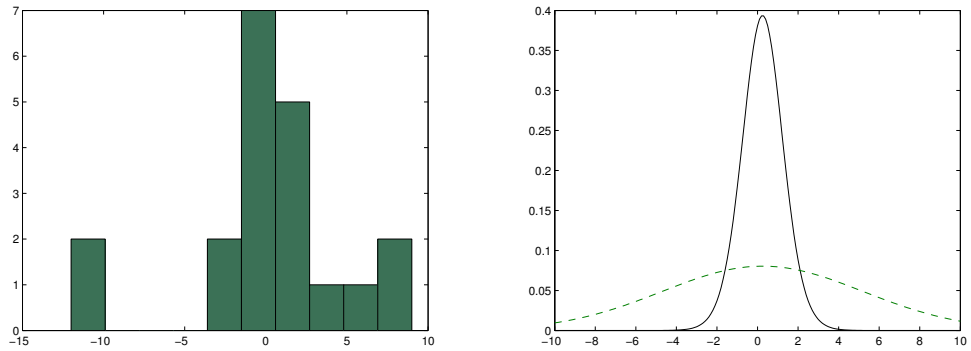


Figure 2.10: Left: a histogram of 20 (synthesised) data-points. Right: an approximation of this data using a Gaussian distribution (dotted line), and using a Student's T-distribution (solid line). The Student's T-distribution seems to do a better job at capturing the 'important' part of the curve.

for these outliers (figure 2.10, dotted line). Alternately, when we try to estimate the data using a Student's T-distribution, the outliers have very little effect (figure 2.10, solid line). Therefore, if these points are indeed outliers, then the Student's T-distribution has done a better job of capturing the 'important' part of the curve.

Specifically, the Student's T-distribution is proportional to

$$(1 + \frac{1}{\nu}x^2)^{-(\nu+1)/2}. \tag{2.9}$$

Although this distribution will prove important in later sections of this document, it is important to note that this distribution *cannot* be used directly with the nonparametric belief-propagation framework we have outlined. Specifically, we no longer have the useful property that the shape of the distribution is maintained upon multiplication and integration. Therefore, we shall be forced to use an approximation of the Student's T-distribution, which we shall introduce in chapter 3.

Further Reading

The original paper in which the Student's T-distribution was suggested is reproduced in [Gos42]. The paper was originally written by Willeam Sealy Gosset (under the pen-name 'Student'), when he was working for Guinness in Dublin.

An outline of the proof of the Hammersley-Clifford theorem was originally given in Hammersley and Clifford's (unpublished) manuscript [HC71]. A more detailed proof is given in [Bes74].

A short introduction to the belief-propagation algorithms presented above is presented in the tutorial paper [AM00] (or [AM01]). Additional texts which cover these ideas in more depth include [Bis06, Jor, KF].

A pioneer paper using nonparametric belief-propagation is [SIFW03]. Similar ideas are also considered in [Bis06] and [RH05].

A number of papers utilise gradient-ascent algorithms for image restoration. One paper which performs inpainting using such an approach is [RB05]. Others include [FPC00, MCSF06].

As we mentioned, the belief-propagation techniques presented are typically unsuitable for image restoration problems. However, similar belief-propagation techniques have seen some success in the field of image restoration – some example papers include [FPC00, LRHB06].

Chapter 3

Image Priors

In order to apply any of the inference algorithms mentioned in chapter 2, we must first deal with the problem of selecting an appropriate prior. That is, we must select the potential functions (ϕ_is) for each maximal clique in our field. The prior should give us information about how pixels in an image 'should behave'.

Specifically, we shall adapt the *Field of Experts* image prior presented in [RB05]. As we have already suggested, this prior is not in a form which renders it suitable for belief-propagation techniques. We will therefore use an approximation of this prior which renders inference tractable in our setting.

Most of the results presented in this section cover well-established work on image priors. It is not until section 4.1 that we shall begin to derive new results in this area.

3.1 The 'Field of Experts' Image Prior

In [RB05], the authors presented the Field of Experts image prior, which appears to produce state-of-the-art results for image restoration problems. In this setting, each potential function is assumed to take the form of a *product of experts* [Hin99], in which each 'expert' is the response of some image patch ($\mathbf{x}_c$) to a particular filter (J_f). That is, we have potential functions of the form

$$\phi(\mathbf{x}_c; J, \alpha) = \prod_{f=1}^{F} \phi'_f(\mathbf{x}_c, J_f, \alpha_f). \tag{3.1}$$

This formulation probably requires further explanation. Here, the 'image patches' ($\mathbf{x}_c$s) are just the cliques in our field. Each filter (J_f) is just a particular patch that captures some

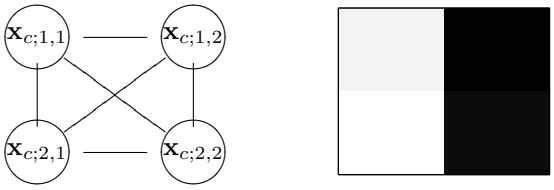


Figure 3.1: Left: an image patch – here each $x_{i,j}$ is an unknown. Right: a filter. In this case the filter corresponds to a vertical edge. Image patches which are 'similar' to vertical edges will have a very large (in magnitude) inner-product.

property about 'natural' images. The 'response' of the patch to this filter is typically assumed to be a function of their inner-product. That is, assuming the cliques in our image correspond to $n \times n$ regions, the response of the clique $\mathbf{x}_c$ to the filter J_f takes the form[1]

$$\phi'_f(\mathbf{x}_c, J_f) = \psi(\langle \mathbf{x}_c, J_f \rangle) = \psi(\sum_{i=1}^{n}\sum_{j=1}^{n}(\mathbf{x}_{c;i,j} \times J_{f;i,j})). \tag{3.2}$$

Essentially, the inner-product tells us *how similar* the image patch is to a particular filter. For example, consider the case in which our image patches correspond to 2×2 cliques. One possible filter is shown in figure 3.1.

This filter could be described as a 'vertical edge'. Assuming that this filter is appropriately normalised (e.g. the values are scaled to lie between -0.5 and 0.5), the inner-product will have a large magnitude exactly when the image patch ($\mathbf{x}_c$) is also a vertical edge. For example, if the filter in figure 3.1 is given by

$$\begin{bmatrix} -0.5 & 0.5 \\ -0.5 & 0.5 \end{bmatrix}$$

(rounding to the nearest decimal place) and we have image patches

$$p_1 = \begin{bmatrix} 0 & 255 \\ 0 & 255 \end{bmatrix}, \quad p_2 = \begin{bmatrix} 255 & 0 \\ 255 & 0 \end{bmatrix}, \quad p_3 = \begin{bmatrix} 255 & 255 \\ 0 & 0 \end{bmatrix}, \quad p_4 = \begin{bmatrix} 255 & 255 \\ 255 & 255 \end{bmatrix},$$

then p_1 and p_2 have very large (in magnitude) inner products with our filter, whereas p_3 and p_4 have inner products of 0. This matches our intuition, since both p_1 and p_2 correspond to vertical edges. Thus, each of our 'experts' measures how similar a particular image patch is to a given filter. Of course, the 'vertical edge' filter we have shown is used by only one expert – we may have another for horizontal edges, another for uniform patches, etc.

The expert is not given by the inner product, but rather by some function of the inner product. In order for this function to be sensible, it makes sense that it should be consistent with the actual distribution of inner products that we would observe in real images [Hin99].

[1] Here we define the inner product to act on *matrices*, rather than *vectors*, but the definition is otherwise the same.

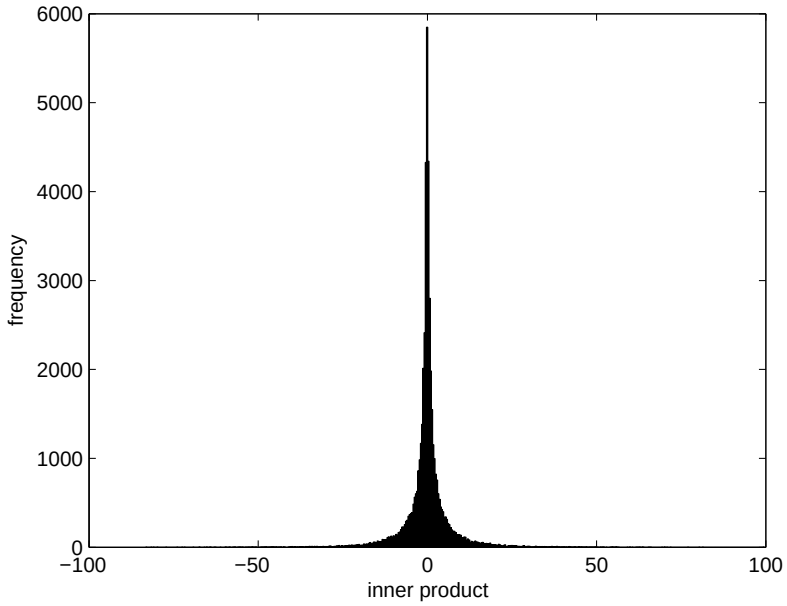


Figure 3.2: The above histogram shows the distribution of inner products observed in real 2×2 patches taken from real images in the Berkeley Segmentation Database [MFTM01]. A total of $50,000$ inner products were used to generate this histogram, which has been separated into 500 bins.

The plot in figure 3.2 shows a histogram of the inner products of real image patches against the filter shown in figure 3.1. In order to generate these inner products, we randomly cropped $50,000$ 2×2 patches from images in the Berkeley Segmentation Database [MFTM01], and computed their inner products against the filter shown.

Two important observations should be made from the histogram in figure 3.2. Firstly, the mode of the distribution is clearly 0. This is not surprising: 'constant' patches are the most common in natural images [RB05], and these are completely orthogonal to this particular filter. Other common patches, such as horizontal edges also have an inner product of zero, meaning that this distribution is very 'peaked' about its mode. However, vertical edges are also reasonably common in natural images, meaning that large inner products are not uncommon. Hence, the distribution is quite heavily tailed.

In fact, the distribution is substantially more heavily tailed than would be suggested by a normal distribution. Although this is difficult to see from the histogram alone, it is more clearly evidenced by the *normal probability plot* shown in figure 3.3. A normal probability plot is constructed as follows: consider random data generated by a hypothetical normal distribution. If the data are sorted and plotted, they should correspond to a smooth monotonic curve. By appropriately scaling the vertical axis, the data can be made to follow a straight line. Now, if we are assessing whether or not *new* data follow a normal distribution, we can simply plot the

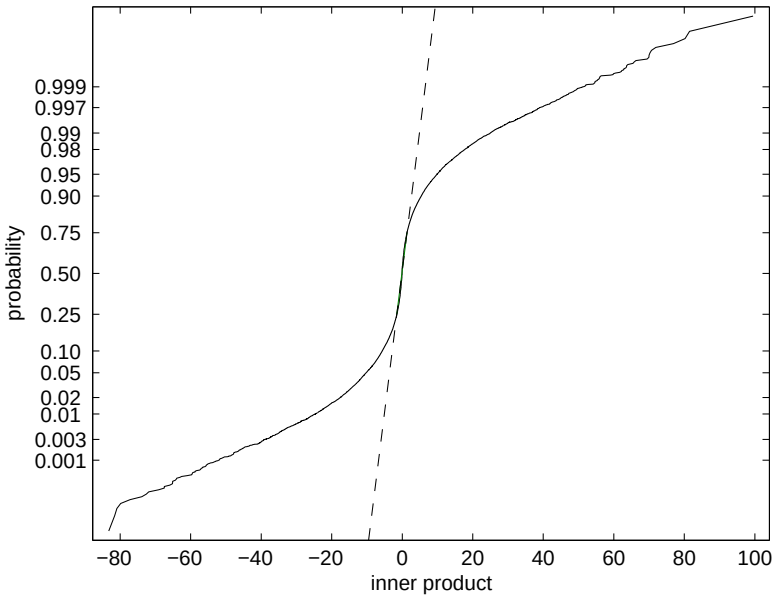


Figure 3.3: The above normal probability plot shows that the inner products (horizontal axis) are more heavily tailed than would be suggested by a normal distribution.

sorted data on this scale. If it is normally distributed, it should approximately follow this line. If it does not follow this line, the plot is able to tell us whether it is more or less heavily tailed than a normal distribution.

In figure 3.3, the dotted line corresponds to the theoretical normal distribution mentioned above. It is very clear from this plot that the inner products (shown by the solid line) do *not* follow a normal distribution. Specifically, the plot reveals that the inner products are more heavily tailed (on both sides) than would be suggested by a normal distribution.

In light of this information, it seems that a Student's T-distribution [Gos42] may be a reasonable choice. This confirms the conclusion obtained in [WHO02], in which such a distribution is suggested for problems of this form.

This leads to the previously mentioned Field of Experts prior [RB05], in which each expert takes the form of a Student's T-distribution. Specifically, we have each expert taking the form

$$\phi'_f(\mathbf{x}_c; J_f, \alpha_f) = (1 + \frac{1}{2} \langle J_f, \mathbf{x}_c \rangle^2)^{-\alpha_f}, \tag{3.3}$$

leading to a potential function of the form

$$\phi(\mathbf{x}_c; J, \alpha) = \prod_{f=1}^{F} (1 + \frac{1}{2} \langle J_f, \mathbf{x}_c \rangle^2)^{-\alpha_f}. \tag{3.4}$$

The α_f in the above expression is simply a transformation of the degrees of freedom term (ν) given in equation (2.9). These parameters simply control the 'shape' of the Student's T-distribution, and in doing so, control the relative importance of each filter.

The shape of this prior should be considered carefully – the mode of each distribution actually occurs when the inner product is zero. The inner product is only zero when the image patch is orthogonal to the filter – i.e. when the two are extremely *dissimilar*. In this sense, the image prior actually tells us which filters we should *not* look like. Hence (as we shall see), the 'most important' filter shall actually correspond to the *least* likely configuration of an image patch.

3.1.1 Learning the Field of Experts Prior

By now we have motivated the appropriateness of the Field of Experts model as an image prior. What remains to be shown is how we actually choose the filters (J_fs) and the importances (α_fs) given in the above equations.

It is typical to choose a set of filters that forms a spanning set for our space of image patches. This means that the number of filters required should be the same as the number of variables in our maximal cliques. Although we have not yet mentioned it explicitly, it is natural to assume that each maximal clique in our image will share the same potential function, since regions in natural images tend to be homogeneous [BM87]. It should be noted that there are a number of applications in which more filters are used than are required to form a spanning set (in such a case we are said to have an *overcomplete basis* [OF97]), but we will not consider such cases here.

Hence, when dealing with $n \times n$ image patches, we are required to select $n \times n$ filters, each of whose size is $n \times n$, as well as their corresponding importances. This results in a total of $n^4 + n^2$ parameters which must be learned (e.g. 20 in the 2×2 case; 90 in the 3×3 case).

When dealing with such a large number of parameters, it is not possible to simply select them by maximum likelihood. [RB05] use an approach known as *contrastive divergence learning* [Hin01, CPnH05]. While this approach has proven to be effective, it has been shown that comparable results can be achieved simply by performing a *principal component analysis* on the filters [MCSF06].

Principal Component Analysis

A principal component analysis (or PCA) can be performed on a set of vectors in order to determine the axes upon which these vectors *vary the most* [Smi02]. The result of the PCA is a transformation matrix; this allows us to express our data-points using a new set of basis vectors, such that the largest amounts of variance now occur in a direction parallel to the basis.

The PCA consists of two steps: given an $n \times k$ data-matrix, D (in which each column is a vector corresponding to a single observation), we first find the covariance matrix of D^T. The principal components are simply given by the eigenvectors of the covariance matrix, which are typically found using a *singular-value decomposition*. Explicitly, the PCA is performed as follows:

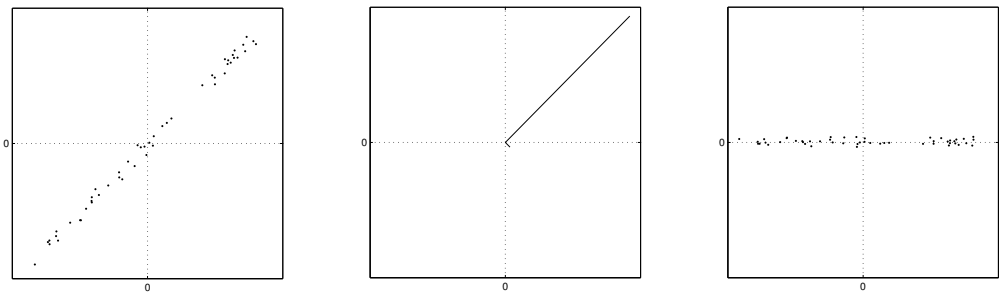


Figure 3.4: Left to right: the original data; the principal components of the data; the transformed data.

- Given a data-matrix $D \in M_{n \times k}$, find

$$\Sigma = cov(D^T) = E\left[(D^T - E(D^T))(D^T - E(D^T))^T\right] = E\left[(D - E(D))^T(D - E(D))\right]$$

- Perform a singular-value decomposition to find $\Sigma = USU^T$.[2]
- The principal components are now given by the columns of U, which correspond to the eigenvalues of Σ. The entries of S (a diagonal matrix) are the variances for the corresponding columns of U

Furthermore, the amount of variance along each axis is given by the *eigenvalues* of the covariance matrix. These are just given by S in the above formula.

Despite the fairly complex formulation of the PCA, it actually has a very intuitive explanation. In figure 3.4 (on the left), we show a scatterplot of some (synthesised) data. The data points *almost* correspond to a straight line, despite some random error. It is clear that *most* of the variance occurs along the line $y = x$. In the centre of figure 3.4, we show the two vectors found by performing PCA, scaled according to their eigenvalues. This confirms that *most* of the variance occurs along $y = x$, and a much smaller amount occurs along the line $y = -x$. Finally, the data are transformed using the transformation found from the PCA. It is clear that for the transformed data, most of the variance occurs along the line $y = 0$ (or, equivalently, parallel to the vector $(1, 0)$); the rest occurs along the line $x = 0$ (or parallel to the vector $(0, 1)$).

The Learned Image Patches

We applied exactly the above approach to select the filters for our Field of Experts model. Of course, the above definition only applies when the data-points are *vectors*, whereas the image patches we are dealing with are two-dimensional. This is not a problem, however – we simply

[2]This is sometimes known as the *eigenvalue decomposition* – a special case of the singular-value decomposition for cases in which Σ is a square, symmetric matrix (as is always the case with covariance matrices).

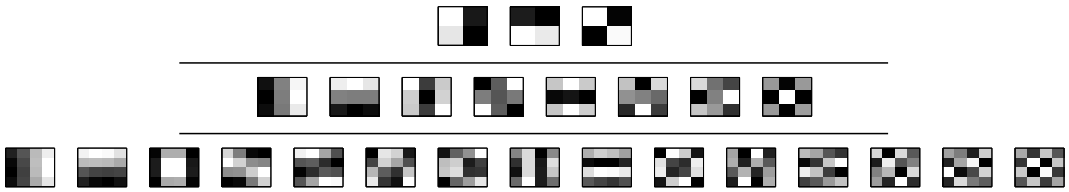


Figure 3.5: Top to bottom: the (last three) filters for a 2×2 model; the filters for a 3×3 model; the filters for a 4×4 model.

let each component of the matrix correspond to a component of a vector.[3] As long as we *reverse* this process after the PCA has been performed, we will arrive at a result in terms of matrices.

As our data-matrix (D), we randomly selected $50,000$ image patches from the Berkeley Segmentation Database [MFTM01] of the size corresponding to our filters. By performing a principal component analysis, it was easy to learn 2×2, 3×3, or even larger models. Some of the learned models are shown in figure 3.5.

In 3.5, we have not shown the *first* filter – in every case, this filter simply corresponded to a uniform gray patch. We can interpret such a patch as capturing variation due to intensity. This is indeed true of the responses it would generate – the inner product of an image patch with such a filter would be directly proportional to that patch's overall intensity. With some thought, it can be seen that we actually should *not* include such a patch in our model – we do not want to favour a patch with lower or higher intensity as being more or less correct than another. For this reason, such a patch should typically not be included – this was indeed the approach taken by [RB05].[4]

Despite having learned our filters, we have yet to learn their corresponding importances. In [RB05] the importances were also learned using contrastive divergence learning, and in [MCSF06], they were learned using gradient-ascent. Another obvious approach would be simply to select them according to a maximum-likelihood estimate of a Student's T-distribution on the set of inner products of these filters with the image patches used in the PCA.

However, we will not adopt any of these approaches – as we shall present in section 4.1, the Student's T-distribution will not be used by our model. The importances of each filter shall be implicitly learned by the Gaussian approximation we shall present. Hence we shall defer this important learning step until the next chapter.

[3] e.g. $\begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix}$ becomes $(m_{1,1}, m_{1,2}, m_{2,1}, m_{2,2})^T$

[4] Even though [RB05] used contrastive divergence learning to select their filters instead of PCA, their process still produced a filter corresponding to a uniform gray patch.

3.2 Extending the Approach to Colour Images

So far, all of the results we have presented have been for *grayscale* images. Fortunately, these approaches can be very easily applied to colour images.

The simplest approach is simply to apply the techniques developed for grayscale images to each of the red, green, and blue channels of an RGB image. Unfortunately, this will produce fairly poor performance, as the channels in a multiband image appear to be highly correlated [MCSF06].

This very fact is actually the motivation for the YCbCr colour space – it is simply a transformation of the original RGB colour space in which the bands become less correlated [The94]. Assuming that RGB values range from 0–255, then the transformation is defined as:

$$\begin{aligned} Y &= 16 + \frac{1}{256}(65.738R + 129.057G + 25.064B) \\ Cb &= 128 + \frac{1}{256}(-37.945R - 74.494G + 112.439B) \\ Cr &= 128 + \frac{1}{256}(112.439R - 94.154G - 18.285B) \end{aligned}$$

The inverse transformation is then defined as:

$$\begin{aligned} R &= 1.164(Y - 16) + 1.596(Cr - 128) \\ G &= 1.164(Y - 16) - 0.813(Cr - 128) - 0.392 * (Cb - 128) \\ B &= 1.164(Y - 16) + 2.017(Cb - 128) \end{aligned}$$

To better explain what is happening here, Y, Cb, and Cr, or R, G, and B can simply be thought of as three basis vectors that can be used to describe a colour. The above transformation simply takes some linear combination of red, green, and blue components, and expresses it in terms of Y, Cb, and Cr. Saying that Y, Cb, and Cr are less correlated than R, G, and B is the same as saying that the basis vectors in YCbCr are more *orthogonal* than those in RGB (i.e. their dot-products are smaller).

Now, if we apply our inference algorithms to each channel in this transformed colourspace, we are able to achieve very good performance on the multiband image. This is the approach used by [RB05].

Of course, even in the YCbCr colourspace, there is still *some* correlation between the channels (although the YCbCr colourspace is defined in such a way that there should be no *first* order correlations, there may still be high-order correlations). Therefore, it is still possible to produce superior results if the channels are treated as being correlated. Indeed, similar techniques to those already described can be used to perform a principal component analysis on multiband images in this way – our 'image patches' now become $2 \times 2 \times 3$, or $3 \times 3 \times 3$ regions (corresponding to the three channels for each pixel), instead of 2×2 or 3×3 [MCSF06].

However, dealing with multiband images in this way also has the effect of increasing the size of our maximal cliques by a factor of three. Since the purpose of this thesis is to develop *fast*

algorithms, as opposed to algorithms which produce state-of-the-art results, we would simply use the standard technique of converting the colourspace to YCbCr.

Further Reading

[RB05] introduce the Field of Experts image prior for use in image processing (and specifically inpainting). They show that their model produces results at the current state-of-the-art level.

The Product of Experts model upon which the Field of Experts model is built is described in [Hin99]. Subsequent papers which demonstrate how contrastive divergence learning can be used in this setting are [Hin01] and [CPnH05].

The original motivation for the use of the Student's T-distribution in image priors is given in [WHO02].

Finally, [MCSF06] explores the improvement that can be achieved by treating the channels in a multiband image as being correlated objects.

Chapter 4

The Gaussian Approximation

As we already suggested in chapter 3, the form of the Student's T-distribution is not appropriate for nonparametric belief-propagation techniques. This is due to the fact that neither the integral nor the product of Student's T-distributions results in another distribution of the same form.

Hence, in this chapter, we will develop an approximation of the Student's T-distribution which takes the form of a mixture of Gaussian density functions. Using such an approximation, we will develop a potential function which is easy to integrate, multiply, and marginalise, while maintaining the shape of a mixture of Gaussian densities.

4.1 The Gaussian Image Prior

In a Gaussian mixture model, our potential functions take the form of a sum of weighted Gaussian probability densities. Specifically, ϕ takes the form

$$\phi(\mathbf{x}_c) = \sum_{i=1}^{N} \beta_i e^{(\mathbf{x}_c - \mu_i)^T \Sigma_i^{-1} (\mathbf{x}_c - \mu_i)}. \tag{4.1}$$

Here, each β_i is simply a weighting coefficient – hence we generally assume that $\sum_{i=1}^{N} \beta_i = 1$. The μ_is and Σ_is are simply the means and covariances of the corresponding Gaussians.

We will show that by approximating each of our experts using a Gaussian mixture, we are able to approximate our potential function using such a mixture also.[1] In order to approximate arbitrary data using a mixture of Gaussians, we will use an algorithm known as the *expectation-maximisation algorithm*, presented below.

[1] Although this approximation produces a *product* of *sums* of Gaussians, this can be expressed as simply a product, as we shall demonstrate later.

4.1.1 The Expectation-Maximisation Algorithm

The aim of our approximation is essentially as follows – given a collection of inner products (similar to those found in section 3.1),[2] we want to estimate their distribution using a mixture of N weighted Gaussians. Obviously, we want to choose those Gaussians that result in the closest fit to the true distribution of the inner products. This requires us not only to compute the means and variances of the N Gaussians, but also to determine for each data-point how responsible each of the N Gaussians was in generating it.

The parameters we wish to estimate are $\beta = (\beta_1 \dots \beta_N)$, $\mu = (\mu_1 \dots \mu_N)$, and $\Sigma = (\Sigma_1 \dots \Sigma_N)$.[3] For brevity, we will use Θ to denote our set of unknowns, i.e. $\Theta = (\beta, \mu, \sigma)$.

However, in order to estimate these parameters, we will also need to determine how responsible each of our Gaussians was in generating each data-point. That is, supposing that we have a data-matrix $\mathbf{U} \in M_{K\times D}$ (here we have K data-vectors, each of which is D dimensional), we want to compute the matrix $\mathbf{J} \in M_{K\times N}$, where each $J_{i,j}$ represents the probability that the ith data-point was generated by the jth Gaussian. The matrix $\mathbf{J}$ is referred to as the 'hidden' data in our model.

Of course, we are not interested in the values of our hidden variables – we only want to maximise the probability of our estimate (say, Θ') given the data ($\mathbf{U}$). That is, we want to compute the value of Θ which maximises

$$l(\Theta; \mathbf{U}) = \log P(\mathbf{U}|\Theta) = \log \sum_{\mathbf{J}\in\mathcal{J}} P(\mathbf{U}, \mathbf{J}|\Theta) \tag{4.2}$$

(This is known as the 'incomplete' log-likelihood, since it does not contain $\mathbf{J}$). Here we are summing over all possible assignments to $\mathbf{J}$ in order to compute the marginal probability with respect to Θ, given $\mathbf{U}$. Ultimately, we want to compute

$$\Theta' = \underset{\Theta}{\operatorname{argmax}} \sum_{\mathbf{J}\in\mathcal{J}} P(\mathbf{U}, \mathbf{J}|\Theta). \tag{4.3}$$

In general, it is not possible to compute this assignment by maximum likelihood directly, due to the exponential number of possibilies for $\mathbf{J}$. Instead, we introduce an 'averaging coefficient' ($q(\mathbf{J}|\mathbf{U}, \Theta)$), which allows us to compute the expected *complete* log-likelihood [RKC05]. An expression for this is given by

$$\langle l(\Theta; \mathbf{U}, \mathbf{J})\rangle_q = \sum_{\mathbf{J}\in\mathcal{J}} P(\mathbf{U}, \mathbf{J}|\mathbf{\Theta}). \tag{4.4}$$

Now, we note that while it may not be possible to maximise $l(\Theta; \mathbf{U})$ directly, q gives us a

[2] As the expectation-maximisation algorithm is far slower than performing a singular value decomposition (section 3.1), we only used 5,000 random patches here, rather than the 50,000 used in section 3.1.

[3] In general, each μ_i is actually the *vector* of means for the ith gaussian, and each Σ_i is a covariance matrix – in our case we are actually dealing with one-dimensional Gaussians, meaning that both of these may simply be regarded as scalars.

method of computing a *lower bound* to l. This is given by

$$\begin{aligned} l(\Theta;\mathbf{U}) &= \log \sum_{\mathbf{J}\in\mathcal{J}} P(\mathbf{U},\mathbf{J}|\Theta) && (4.5)\\ &= \log \sum_{\mathbf{J}\in\mathcal{J}} q(\mathbf{J}|\mathbf{U}) \frac{P(\mathbf{U},\mathbf{J}|\Theta)}{q(\mathbf{J}|\mathbf{U})} && (4.6)\\ &\geq \sum_{\mathbf{J}\in\mathcal{J}} q(\mathbf{J}|\mathbf{U}) \log\left(\frac{P(\mathbf{U},\mathbf{J}|\Theta)}{q(\mathbf{J}|\mathbf{U})}\right) && (4.7)\\ &\stackrel{\text{def.}}{=} \mathcal{L}(q,\Theta) && (4.8) \end{aligned}$$

(the above result can be derived using *Jensen's inequality* [Del02, Bil98, Bor04]).

This allows us to express the EM algorithm as follows: given an initial estimate of Θ (say, Θ^0), we repeatedly perform two steps (known as the 'E', or 'expectation', and the 'M', or 'maximisation' steps), until our estimates of Θ converge. These two steps are specified as follows:

E-step Compute $q^{t+1} = \operatorname{argmax}_q \mathcal{L}(q,\Theta^t)$

M-step Compute $\Theta^{t+1} = \operatorname{argmax}_\Theta \mathcal{L}(q^{t+1},\Theta)$.

It can be proved that successive estimates of Θ taken in this way will eventually converge to a local maximum of $\log P(\mathbf{U},\Theta)$ [DLR77, MK97, Del02].

The M-step is now equivalent to maximising the expected complete log-likelihood (equation (4.4)). This can be seen as follows:

$$\begin{aligned} \mathcal{L}(q,\Theta) &= \sum_{\mathbf{J}\in\mathcal{J}} q(\mathbf{J}|\mathbf{U}) \log\left(\frac{P(\mathbf{U},\mathbf{J}|\Theta)}{q(\mathbf{J}|\mathbf{U})}\right) && (4.9)\\ &= \sum_{\mathbf{J}\in\mathcal{J}} q(\mathbf{J}|\mathbf{U}) \log P(\mathbf{U},\mathbf{J}|\Theta) - \sum_{\mathbf{J}\in\mathcal{J}} q(\mathbf{J}|\mathbf{U}) \log q(\mathbf{J}|\mathbf{U}) && (4.10)\\ &= \langle l(\Theta;\mathbf{U},\mathbf{J})\rangle_q - \sum_{\mathbf{J}\in\mathcal{J}} q(\mathbf{J}|\mathbf{U}) \log q(\mathbf{J}|\mathbf{U}). && (4.11) \end{aligned}$$

The right-hand-side of the above expression is now independent of Θ, so it can be ignored when maximising.

Finally, the specific solution for $q^{t+1}(\mathbf{J}|\mathbf{U})$ is just $P(\mathbf{J}|\mathbf{U},\Theta^t)$ [RKC05], since

$$\begin{aligned} \mathcal{L}(P(\mathbf{J}|\mathbf{U},\Theta^t),\Theta^t) &= \sum_{\mathbf{J}\in\mathcal{J}} P(\mathbf{J}|\mathbf{U},\Theta^t) \log\left(\frac{P(\mathbf{U},\mathbf{J}|\Theta)}{P(\mathbf{J}|\mathbf{U},\Theta^t)}\right) && (4.12)\\ &= \sum_{\mathbf{J}\in\mathcal{J}} P(\mathbf{J}|\mathbf{U},\Theta^t) \log P(\mathbf{U}|\Theta^t) && (4.13)\\ &= \log P(\mathbf{U}|\Theta^t) && (4.14)\\ &= l(\Theta^t;\mathbf{U}). && (4.15) \end{aligned}$$

4.1.2 The EM Algorithm for Gaussian Mixture Models

Although we have tried to present the most general forumaltion of the EM algorithm above, it is worth specifically stating how this applies when the parameters we are trying to estimate are those which define a Gaussian mixture.

All of the formulas presented below can be derived from the above equations. Since this is probably the most common application of the EM algorithm, there are many texts relating to exactly this specific form. See [Bil98] for further details.

Assuming that we have determined a suitable initial estimate of Θ ($\Theta^0 = (\beta^0, \mu^0, \sigma^0)$) (we will deal with the details of this initial estimate later), then the E-step and the M-step of our algorithm are presented in figure 4.1.

Of course, since the Gaussians we are trying to estimate are all one dimensional, our mean vectors (μ_is), and our covariance matrices (Σ_is) are simply scalars.

K-Means Clustering

While the algorithm presented in figure 4.1 will produce an approximation of the data in the desired form, the quality of this approximation tends to be highly sensitive to its initialisation [MA93]. Possibly the simplest approach for initialisation would be simply to select the the means and covariances to be the component means and covariances for random subsets of the data.

A slightly more principled approach, which we will use, is that of *K-means clustering*. This algorithm is very similar to the EM algorithm in its purpose. Here, we are trying to cluster our data into K groups, such that the distances of our data-points to the means of their groups are minimised. Hence, we are trying to minimise the error function

$$V = \sum_{c=1}^{C} \sum_{U_i \in \mathbf{U}_c} (U_i - \mu_c)^2 \quad \text{(Or equivalently } \sum_{c=1}^{C} \sum_{U_i \in \mathbf{U}_c} |U_i - \mu_c|) \tag{4.16}$$

(Here our data are separated into C clusters, with means ($\mu_1 \ldots \mu_C$) – minimising the mean-squared error is equivalent to minimising the distance).

Again, this is done using an iterative approach. Initially, we simply use a random selection of C of our data-points as the means of the clusters. In order to cluster the remaining data-points, we simply assign them to whichever cluster has the closest mean.

Having done this, we now compute the *new* means of the clusters, and the process is repeated. Pseudocode for the EM algorithm is presented in figure 4.2.

Furthermore, the βs and Σs may simply be computed by computing the covariances of the clusters and the size of the clusters respectively.

As an aside, it is worth mentioning that the K-means clustering algorithm is *also* highly

Algorithm 4: E-Step

Data: U, $\Theta^t = (\beta^t, \mu^t, \Sigma^t)$
$\mathbf{J} = \mathbf{0}$
for $i \in \{1 \dots K\}$; /* K = number of data-points */
do
  for $j \in \{1 \dots N\}$; /* N = number of Gaussians */
  do
    $z = \frac{1}{\sqrt{2*\pi}^N \sqrt{\det(\Sigma_j^t)}} exp(-\frac{1}{2}(\mathbf{U}_i - \mu_j^t)^T (\Sigma_j^t)^{-1}(\mathbf{U}_i - \mu_j^t))$
    numerator $= \beta_j^t \times z$
    denominator $= 0$
    for $r \in \{1 \dots N\}$ **do**
      $z = \frac{1}{\sqrt{2*\pi}^N \sqrt{\det(\Sigma_r^t)}} exp(-\frac{1}{2}(\mathbf{U}_i - \mu_r^t)^T (\Sigma_r^t)^{-1}(\mathbf{U}_i - \mu_j^t))$
      denominator = denominator $+\beta_r^t \times z$
    end
    $\mathbf{J}_{i,j} = \frac{\text{numerator}}{\text{denominator}}$
  end
end
return J

Algorithm 5: M-Step

Data: $\mathbf{U}$, $\mathbf{J}$
for $s \in \{1 \dots N\}$ **do**
  $\beta_s^{t+1} = \sum_{j=1}^K \mathbf{J}_{j,s}$
  $\mu_s^{t+1} = \frac{\sum_{j=1}^K \mathbf{U}_j \mathbf{J}_{j,s}}{\sum_{j=1}^K \mathbf{J}_{j,s}}$
  $\Sigma_s^{t+1} = \frac{\sum_{j=1}^K (\mathbf{U}_j - \mu_s^{t+1})(\mathbf{U}_j - \mu_s^{t+1})^T \mathbf{J}_{j,s}}{\sum_{j=1}^K \mathbf{J}_{j,s}}$
end
return $\Theta^{t+1} = (\beta^{t+1}, \mu^{t+1}, \Sigma^{t+1})$

Algorithm 6: The EM algorithm

Data: $\mathbf{U}$, Θ^0
for $i \in \{1 \dots I\}$; /* for each iteration */
do
  $\mathbf{J}$ =E-Step($\mathbf{U}$, Θ^{i-1})
  Θ^i =M-step($\mathbf{U}$, $\mathbf{J}$)
end
return $\Theta' = \Theta^I$

Figure 4.1: Pseudocode for the EM algorithm

Algorithm 7: The K-means clustering algorithm

Initially, choose C random data-points as the means of our clusters ($\mu_1 \dots \mu_C$).
for $i \in \{1 \dots I\}$; `/* for each iteration */`
do
  for $U_j \in$ **U do**
    assign U_j to the cth cluster such that $(U_j - \mu_c)$ is minimised.
  end
  compute the new cluster means from the assignments just generated.
end
return μ

Figure 4.2: Pseudocode for the K-means clustering algorithm

sensitive to its initialisation. For this reason, several more sophisticated initialisation algorithms have been developed [MA93]. In our case, we found such approaches to be unnecessary.

Further Reading

Many ideas relating to the EM algorithm are covered in [Har58]. The algorithm was first fully specified in [DLR77].

This algorithm is covered in more detail in [MK97]. Many tutorial papers have also been valuable when learning this material, especially [Del02, Bil98, Bor04] and [Min98]. Many of the derivations presented above were adapted from those in [RKC05]. Furthermore, [Bil98] deals specifically with how the EM algorithm relates to estimating Gaussian mixture models, meaning that it is probably the most relevant to this work.

Issues related to the initialisation of the EM algorithm, which motivate our use of K-means clustering, are covered in [MA93].

Finally, as we have already mentioned in chapters 1 and 2, the use of the Gaussian distribution in the MRF setting is described in [Bis06] and [RH05].

Chapter 5

Belief-Propagation

In this chapter, we will give a more detailed discussion of the belief-propagation algorithms presented in chapter 2. This will allow us to see how these algorithms may be applied when the messages being passed are (the parameters of) multivariate Gaussian distributions.

Having done this, we will see how the Gaussian image prior we developed in chapter 4 can be expressed in the form required by these algorithms.

5.1 Nonparametric Belief-Propagation Continued

We have already introduced the important equations for nonparametric belief-propagation in chapter 2. Specifically, we have that messages are computed according to

$$M_{i\to j}(S_{i,j}) = \int_{\mathbf{x}_i \setminus \mathbf{x}_j} \phi_i(\mathbf{x}_i) \prod_{\mathbf{x}_k \in (\Gamma_{\mathbf{x}_i} \setminus \{\mathbf{x}_j\})} M_{k\to i}(S_{k,i}), \tag{5.1}$$

and that the final distribution is computed according to

$$D_i(\mathbf{x}_i) = \phi_i(\mathbf{x}_i) \prod_{\mathbf{x}_k \in \Gamma_{\mathbf{x}_i}} M_{k\to i}(S_{k,i}). \tag{5.2}$$

We have also explained that our image prior takes the form of a *mixture of Gaussians*. That is, it takes the form

$$\phi(\mathbf{x}_c) = \sum_{i=1}^{N} \beta_i e^{(\mathbf{x}_c - \mu_i)^T \Sigma_i^{-1} (\mathbf{x}_c - \mu_i)}. \tag{5.3}$$

What we must now demonstrate is that when our potential functions take such a form, the equations (5.1) and (5.2) have a particularly simple solution.

In order to do so, we will first need to develop some additional theory about Gaussian probability distributions.

5.1.1 The Multivariate Gaussian Probability Density Function

In order to compute equations (5.1) and (5.2), we will need to show how to compute the product of several Gaussians, and the marginal distribution of a Gaussian. Furthermore, since we are trying to solve inpainting problems, many of the variables (pixels) in our field are observed, i.e. those pixels on the edge of an inpainting region. Hence we shall also need to show how to compute the *conditional* distribution of a multivariate Gaussian.

Fortunately, all three of these equations have a simple form [Bis06, RH05]. We shall simply state each of these forms without proof in this section – each of these results shall be proved in the Appendix.

Products of Gaussian Distributions

Given K Gaussians (with means $\mu_1 \dots \mu_K$, and covariances $\Sigma_1 \dots \Sigma_K$), the covariance of the product (Σ') is given by

$$\Sigma' = (\sum_{i=1}^{K} \Sigma_i^{-1})^{-1} \tag{5.4}$$

(It should be noted that although each Σ_i^{-1} is often singular, their sum, in general, is not). The mean of the product (μ') is given by

$$\mu' = \Sigma'(\sum_{i=1}^{K} \Sigma_i^{-1}\mu_i). \tag{5.5}$$

The corresponding importance term for the product is just $\beta' = \prod_{i=1}^{K} \beta_k$.

Marginal Distribution

Suppose that we want to marginalise a certain Gaussian distribution in $\mathbf{x}$ in terms of some subset of its variables $\mathbf{x}_u$. Then without loss of generality (by reordering the variables), we can write $\mathbf{x}$ as $\mathbf{x} = (\mathbf{x}_{(u)}^T \mid \mathbf{x}_{(m)})^T$ (here $\mathbf{x}_{(m)}$ corresponds to the variables we want to integrate out). With this ordering of the variables, we can partition the mean and the covariance matrix as

$$\mu = \begin{pmatrix} \mu_{(u)} \\ \mu_{(m)} \end{pmatrix} \tag{5.6}$$

and

$$\Sigma = \begin{bmatrix} \Sigma_{(u,u)} & \Sigma_{(u,m)} \\ \Sigma_{(m,u)} & \Sigma_{(m,m)} \end{bmatrix}. \tag{5.7}$$

The mean of the marginal distribution is now simply given by the corresponding subset of the means of the full distribution – i.e. the new mean is given by $\mu_{(u)}$. Similarly, the new covariance matrix is given by $\Sigma_{(u,u)}$.

Conditional Distribution

Finally, suppose that certain variables in a Gaussian distribution have been observed. Again, we write $\mathbf{x}$ as $\mathbf{x} = (\mathbf{x}_{(u)}^T \mid \mathbf{x}_{(o)})^T$, where in this case $\mathbf{x}_{(o)}$ corresponds to our set of observed values (a constant vector). Again, we partition the mean and the covariance matrix as

$$\mu = \begin{pmatrix} \mu_{(u)} \\ \mu_{(o)} \end{pmatrix} \tag{5.8}$$

and

$$\Sigma = \begin{bmatrix} \Sigma_{(u,u)} & \Sigma_{(u,o)} \\ \Sigma_{(o,u)} & \Sigma_{(o,o)} \end{bmatrix}. \tag{5.9}$$

The mean of the conditional distribution ($\mu_{(u;o)}$) is now given by

$$\mu_{(u;o)} = \mu_{(u)} + \Sigma_{(u,o)}\Sigma_{(o,o)}^{-1}(\mathbf{x}_{(o)} - \mu_{(o)}), \tag{5.10}$$

and the covariance matrix ($\Sigma_{(u;o)}$) is given by

$$\Sigma_{(u;o)} = \Sigma_{(u,u)} - \Sigma_{(u,o)}\Sigma_{(o,o)}^{-1}\Sigma_{(u,o)}^T \tag{5.11}$$

(see the Appendix for more details).

5.1.2 Gaussian Mixture Models

It is easy to apply the above results when we are dealing with not one, but several multivariate Gaussians. The marginal and conditional distribution formulas can simply be applied to each of the Gaussians in the mixture, with their individual weights (β_is) remaining unchanged.

Products of several Gaussian mixtures are computed in much the same way that polynomials are multiplied – if we wish to multiply together P different mixture models, each consisting of K Gaussians (say, $G_{1,1} \ldots G_{P,K}$), then we would obtain a new mixture containing one Gaussian for each element in the cartesian product $G_1 \times G_2 \times \ldots \times G_P$. Each of these products is computed using the equations presented above. The new mixture now contains K^P Gaussians.

It is not difficult to see that this poses a certain problem for our algorithm – the number of Gaussians in our mixtures will grow exponentially as we pass messages (equation (5.1)). Since we are interested in producing *fast* algorithms, it will be necessary to decrease this number to a reasonable level.

When passing messages, [SIFW03] use a Gibbs sampler to estimate a new mixture model which contains a smaller number of Gaussians. However, they are only able to apply the algorithms they develop to small cliques (the largest they show contain three nodes). Since the smallest model we are reasonably able to work with contains cliques of size four (see chapter 3), their

approach is not suitable in our case. Hence we opt for the simpler (if suboptimal) approach of including only the most important Gaussians in our new model (i.e. those with the highest weighting coefficient, β_i), up to some predefined limit.

5.2 Our Model as a Mixture of Gaussians

In chapter 3 we developed an image prior which took the form of a product of experts, in which each expert took the form of a Student's T-distribution. In chapter 4, we showed that each of these experts can be approximated using a mixture of Gaussians. However, this approximation is still not in the form required by our model. Specifically, each of our experts has been approximated in the form

$$\phi'_f(\mathbf{x}_c;\Theta,J) \simeq \sum_{i=1}^{K} \beta_{f,i} exp\left(\frac{(\langle J_f,\mathbf{x}_c\rangle - \mu_{f,i})^2}{2\sigma_{f,i}^2}\right) \tag{5.12}$$

(i.e. as a mixture of K *one*-dimensional Gaussians in terms of the inner product of J_f and $\mathbf{x}_c$). Instead, we would like each Gaussian in our mixture to be expressed in terms of $\mathbf{x}_c$. Hence, we need to solve the system

$$exp\left(\frac{(\langle J,\mathbf{x}\rangle - \mu)^2}{2\sigma^2}\right) = exp\left((\mathbf{x}-\underline{\mu})^T\Sigma^{-1}(\mathbf{x}-\underline{\mu})\right). \tag{5.13}$$

That is, we are trying to solve for Σ^{-1} (a matrix) and $\underline{\mu}$ (a vector), in terms of J (a vector) and μ (a constant). It is not difficult to see (by expanding $(\langle J,\mathbf{x}\rangle - \mu)^2$, and looking at the coefficients of the quadratic terms) that the only solution for Σ^{-1} is

$$\Sigma^{-1} = \frac{1}{2\sigma^2} J\cdot J^T = \frac{1}{2\sigma^2}\begin{bmatrix} J_1^2 & J_1J_2 & \cdots & J_1J_n \\ J_2J_1 & J_2^2 & & J_2J_n \\ \vdots & & \ddots & \vdots \\ J_nJ_1 & J_nJ_2 & \cdots & J_n^2 \end{bmatrix} \tag{5.14}$$

(where n is the size of the filter J – in our case, we will typically have $n = 4$).

Alternately, there are infinitely many solutions for $\underline{\mu}$. One obvious solution is

$$\underline{\mu} = \begin{pmatrix} \frac{\mu}{\sum_{i=1}^n J_i} \\ \vdots \\ \frac{\mu}{\sum_{i=1}^n J_i} \end{pmatrix}. \tag{5.15}$$

However, we found for all of our filters that $\sum_{i=1}^n J_i \simeq 0$, meaning that this solution would be highly unstable. A more stable solution (which we used) is given by

$$\underline{\mu} = \begin{pmatrix} \mu/J_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{5.16}$$

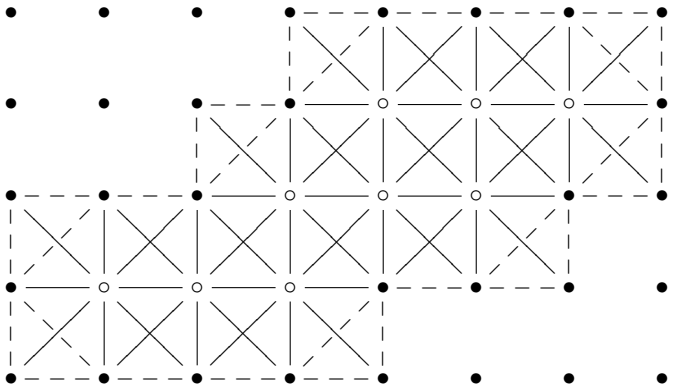


Figure 5.1: The topology of a scratched region. In the above graph, the empty circles correspond to the scratched regions in our image, whereas the filled circles correspond to the 'observed' regions.

(in all cases we had that $J_1 \sim \pm 0.5$). Of course, there are plenty of other solutions of the same (or a similar) form, but the particular solution chosen should make no difference in practice (as long as it is stable).

Using the above solution, each of our experts can now be expressed in the required form. Furthermore, by using the equations already introduced for the product of several Gaussian mixtures (equation (5.4) and (5.5)), we are able to express our *entire* potential function as a single Gaussian mixture.

5.2.1 The topology of our model

When dealing with image inpainting, the topology of the graphical model being used will depend on the shape of the region being inpainted. Fortunately, by using tools in *Elefant* (see section 5.3), we were able to perform belief-propagation for graphical models of any topology.

In chapter 2, we presented several figures to describe common topologies of graphical models that may be encountered when dealing with images. The situation is slightly different when dealing with inpainting, since most of the nodes in our graph are *observed* (i.e. the pixels which have *not* been corrupted). As a result, the only cliques which participate in our graphical model are those cliques that contain some unknown values (i.e. the scratches).

Now, any clique which is fully contained within a scratch simply uses the Gaussian model described in chapter 4 as its potential function. For other cliques which are only partially contained within the scratch, this Gaussian model is conditioned upon those variables which have been observed (see section 5.1.1). An example of such a graph (using a 2×2 model) is presented in figure 5.1.

In figure 5.1, the empty circles correspond to the scratched regions in our image, whereas the filled circles correspond to the 'observed' regions. Since we are using a 2×2 model, only the

observed regions which are adjacent to our scratch are considered by our model. The other pixels are simply ignored (indicated by the absence of an edge in the graph). Although some edges (indicated by a dotted line) only connect two observed variables, these edges are still used to form the maximal cliques. Consequently, any maximal clique containing one or more of these observed nodes will have its potential function conditioned upon their observed values. In figure 5.1, there is actually only one maximal clique which uses the full, unconditional, potential function (in the top right section of the graph). It is also worth noting that it *is* possible to form a junction tree from the cliques in figure 5.1 (see section 6.3).

5.2.2 Estimating the Results After Propagation

After all messages have been propagated throughout our field, we still have the problem of selecting the correction of each pixel from the distributions contained in each clique.

Using the equation for the marginal distribution described above, it is easy to determine the (one-dimensional) marginal distribution for any given pixel.

Unfortunately, computing the mode of, or sampling from, a mixture of Gaussian density functions is actually very difficult.[1] Hence, to select the mode of such a distribution, we simply consider all 256 possible pixel values. Although this may appear to make our algorithm linear in the number of gray-levels (the very problem we were trying to avoid), it should be noted that this is performed only as a final step, after the last iteration, and that measuring the response of a one-dimensional Gaussian is a very inexpensive procedure. More complicated mode-finding techniques do exist [CPn00], but we found these to be unnecessary in our case.

Of course, when dealing with the special case in which our mixture contains only a single Gaussian, we can simply use the mean of this distribution as its mode, and avoid this problem altogether (see chapter 6).

5.3 Our Implementation

Having now fully described the model and the algorithms required to apply nonparametric belief-propagation to image inpainting problems, it is worth briefly describing a few details of our implementation.

Learning/Approximating the Prior

In order to select our filters, we randomly cropped $50,000$ 2×2 image patches from images in the Berkeley Segmentation Database [MFTM01, RB05, MCSF06]. These regions were selected, and the principal component analysis was performed using MATLAB code that we developed.

[1] If we were dealing with the sum of several Gaussian *random variables* this would be easy – but since we are actually dealing with a *single* distribution, this problem is hard.

We implemented the EM algorithm and K-means clustering (as specified in chapter 4) using MATLAB.

Belief Propagation

All of our belief-propagation routines were written using a high-level Python implementation. Much of this code used libraries from *Elefant*[2] – an open-source machine-learning toolkit which is currently being developed by NICTA (which I played a small role in developing). With this code, we are able to produce loopy belief-propagation and junction-tree algorithms for graphs of any topology. Additionally, we used the *numpy* matrix library (see `http://numpy.scipy.org/` for more detail) for all message-passing code, allowing us to develop reasonably fast belief-propagation algorithms, in spite of our high-level implementation.

Ultimately, once the prior model has been learned (and the corresponding importances, covariance matrices, and means have been saved in a file), our algorithm requires the following parameters:

A corrupted image – the image whose pixels we want to restore.

An inpainting mask – a two colour (usually black/white) image indicating which pixels are corrupted.

The original image – the 'uncorrupted' image. This is only used to gauge inpainting performance, and can be ignored if it is unknown (for example, for the picture of Abraham Lincoln presented in chapter 1, we do not have access to an uncorrupted version).

I – the number of iterations to perform belief-propagation.

Our algorithm simply produces a new image, which is just the corrupted image, with the region specified by the mask restored.

We have also developed a concurrent (MPI) version of loopy belief-propagation (using *pypar* – a Python binding for MPI, available at `http://datamining.anu.edu.au/~ole/pypar/`). However, for the sake of running-time comparisons, we have restricted ourselves to using a single processor.

[2]Elefant can be downloaded at `https://rubis.rsise.anu.edu.au/elefant`.

Chapter 6

Results

In this section, we will present the inpainting experiments we performed using our implementation. As we have mentioned, the main purpose of our method is to inpaint images *quickly*, rather than to achieve optimal inpainting performance. Therefore, running-times shall be the main focus of this section. Fortunately, we shall also demonstrate that in spite of our approximate technique, our method is able to produce results that are visually comparable to high-order (gradient-based) techniques.

6.1 Reference Implementation

Currently, the state-of-the-art (in terms of signal-to-noise ratio) inpainting implementation is probably [RB05]. This implementation is available online, at

`http://www.cs.brown.edu/people/roth/research/software.html`

This implementation uses gradient-based techniques, and appears to achieve optimal inpainting performance (in terms of PSNR). Unfortunately, the code provided only has trained filters for a 3×3 model, meaning that it is not easily comparable to our 2×2 model. Therefore, we shall present some of these results using a 3×3 model, as an indication of what 'optimal' inpainting performance looks like. In order to measure running-times, we shall simply use a *random* set of filters with this implementation; while these filters will not be able to inpaint images successfully, they will indicate the running-time of this method if it were properly trained on 2×2 filters (the actual choice of filters should have no effect on the running time).

6.2 Performance measures

Other than the running time of our inpainting algorithm, we will assess two measures of its performance – *peak signal-to-noise ratio* (PSNR), and *structured similarity* (SSIM) [WBSS04]. A brief description of the meaning of these two measures is as follows:

PSNR

The peak signal-to-noise ratio is defined in terms of the mean-squared difference between the original (uncorrupted) image, and the inpainted image.[1] For a set of pixels in the original image (say X), and a set of inpainted pixels (say Y), the mean-squared error between X and Y is defined as

$$\mathrm{MSE}(X,Y) = \frac{1}{|X|}\sum_{i=1}^{|X|}(X_i - Y_i)^2. \tag{6.1}$$

The PSNR is now computed as

$$\mathrm{PSNR}(X,Y) = 20\log_{10}\left(\frac{255}{\sqrt{\mathrm{MSE}(X,Y)}}\right). \tag{6.2}$$

Here, 255 is the maximum pixel value that we observe in images (assuming that pixels are stored as integers in the range 0-255). This value simply appears in the numerator of the above expression in order to ensure that the results are appropriately scaled, meaning that the PSNR is independent on the specific representation of the image. Ultimately, the PSNR increases without bound as X and Y are made more similar.

SSIM

[WBSS04] reported that even though two results may differ in terms of their PSNR, they may appear to be *visually* very similar. For this reason, they produced a similarity measure which is intended to better coincide with people's real perception of the difference between images, as opposed to the less interpretable mean-squared error. Since the interpretation of this method is quite involved, we shall only present the computational formula.

Given two images, X and Y, we define

$$\mu_X = \frac{1}{|X|}\sum_{i=1}^{|X|} X_i, \tag{6.3}$$

$$\sigma_X = \sqrt{\frac{1}{|X|-1}\sum_{i=1}^{|X|}(X_i - \mu_X)^2}, \tag{6.4}$$

$$\sigma_{XY} = \frac{1}{|X|-1}\sum_{i=1}^{|X|}((X_i - \mu_X)(Y_i - \mu_Y)) \tag{6.5}$$

[1] For the purpose of testing, we shall generally assume that we have access to the uncorrupted image, which would not be the case in practice.

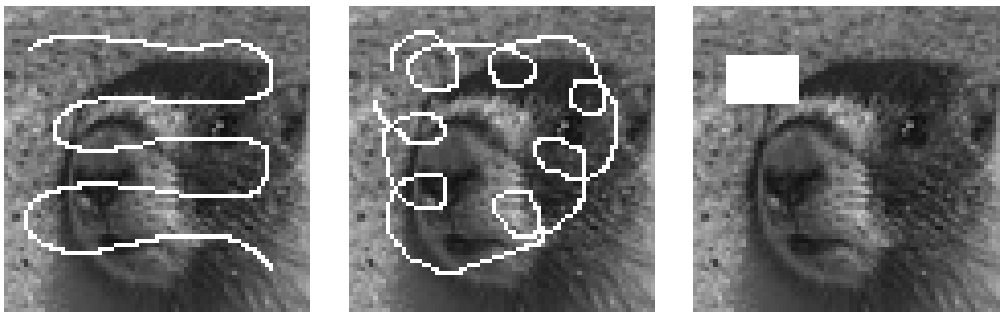


Figure 6.1: The graph formed from the white pixels in the left image forms a junction-tree (assuming a 2×2 model). The graphs formed from the white pixels in the other two images do not.

(which is well defined since $|X| = |Y|$). μ_Y and σ_Y are defined analogously. The structured similarity (SSIM) is now computed by

$$\text{SSIM}(X, Y) = \frac{(2 \cdot \mu_X \cdot \mu_Y + c_1)(2 \cdot \sigma_{XY} + c_2)}{(\mu_X^2 + \mu_Y^2 + c_1)(\sigma_X^2 + \sigma_Y^2 + c_2)}. \tag{6.6}$$

Here c_1 and c_2 are both constants which must be chosen; [WBSS04] define these constants in terms of two different values, namely $(c_1, c_2) = ((255K_1)^2, (255K_2)^2)$, where $(K_1, K_2) = (0.01, 0.03)$ (the specific values of these constants were chosen by [WBSS04] in order to design a similarity measure that closely matches our visual perception). Again, the number 255 is used to scale the result appropriately.

This measure now corresponds to a value between 0 and 1, with 1 indicating that the two images are identical.

6.3 Junction-Trees vs. Loopy Belief-Propagation

As we have already suggested, it is sometimes possible to form a junction-tree from the cliques in the region being inpainted (such an example was shown in section 5.2.1). This may be highly advantageous, as it would allow us to inpaint an image using only a single iteration, whereas loopy belief-propagation may take multiple iterations in order to converge.

While it is difficult to characterise exactly for which graphs it is possible to form a junction-tree, it should be sufficient to say that they roughly correspond to those which contain no 'loops'. As an example, consider the three images in figure 6.1: it is easy to form a junction-tree from cliques in the leftmost image (assuming that we are using a 2×2 model) – in fact, the resulting tree will simply be a chain. However, no junction-tree exists for the cliques in the other two (again assuming a 2×2 model).[2]

[2]It should be noted that we *could* form a junction tree, if we were willing to deal with larger cliques. Of course, since we are interested in developing fast algorithms, we shall only consider cases in which 2×2 cliques are used.

Unfortunately, determining whether or not a graph obeys the junction-tree property or not is a very expensive procedure (see chapter 2). Therefore, in all of our examples, we have simply used the loopy belief-propagation algorithm. However, it should be noted that if the user is able to visually determine that this property is obeyed (as it often will be when removing common forms of corruption, such as scratches), it will be possible for them to use the junction-tree algorithm, and produce results even faster.

6.4 Inpainting Results

In figure 6.2 we show an image which has had text overlayed which we want to remove (this particular image was originally used in [BSCB00]). Even after only a single iteration of inpainting, the text has been almost completely removed. This is in contrast to gradient-based methods, which typically take several thousand iterations to inpaint an image.

Alternately, figure 6.3 compares our results to the current state-of-the-art. Although it is true that there are regions of the image in which a difference between the two is perceptible, the quality of both images is quite high. The 'state-of-the-art' technique uses a 3×3 model, and 2750 iterations of gradient-ascent. Consequently, it takes several times longer to produce a result.

In the images presented in figure 6.2, we trained a model using three Gaussians, and then selected the most important Gaussian in order to perform inpainting. Interestingly, this approach outperformed the more obvious approach of training a model using only a single Gaussian. It is possible that in the former case, the 'most important' Gaussian is capturing only the relevant information about the shape of the curve, whereas in the latter case, the variance of our Gaussian is overestimated to compensate for the heavy tails of our distribution (see chapter 2).

In figure 6.4, we compare the performance of several models, in which training and inference is performed using different numbers of Gaussians. These results are presented in detail in table 6.1.

6.4.1 A Colour Image

To demonstrate that this approach *can* be applied to colour images, it is worth giving one example of a colour image being inpainted using this technique. in figure 6.5, we show a corrupted image, converted to the YCbCr colour-space (see chapter 3), and inpainted in each channel separately. The three inpainted channels are then combined to form a single colour image.

Figure 6.2: Above, top-left to bottom-right: the original image; the image containing the text to be removed; inpainting after a single iteration, using a single Gaussian (PSNR = 22.74, SSIM = 0.962); inpainting after two iterations (PSNR = 22.82, SSIM = 0.962). Below: close-ups of all images.

Figure 6.3: Left: our inpainted image (PSNR = 22.82); right: inpainted image from [RB05] (PSNR = 31.4232).

Gaussians	Max	Iteration	PSNR	SSIM
1	1	1	22.57	0.927
		2	22.67	0.928
		3	22.68	0.928
3	1	1	22.81	0.927
		2	22.87	0.928
		3	22.88	0.928
3	3	1	22.82	0.927
		2	22.87	0.928
		3	22.88	0.928
3	9	1	22.80	0.927
		2	22.86	0.928
		3	22.87	0.928

Table 6.1: Comparison of inpainting performance for several models. Here we vary the number of Gaussians used to compute the initial mixture, as well as the maximum number of Gaussians allowed during propagation.

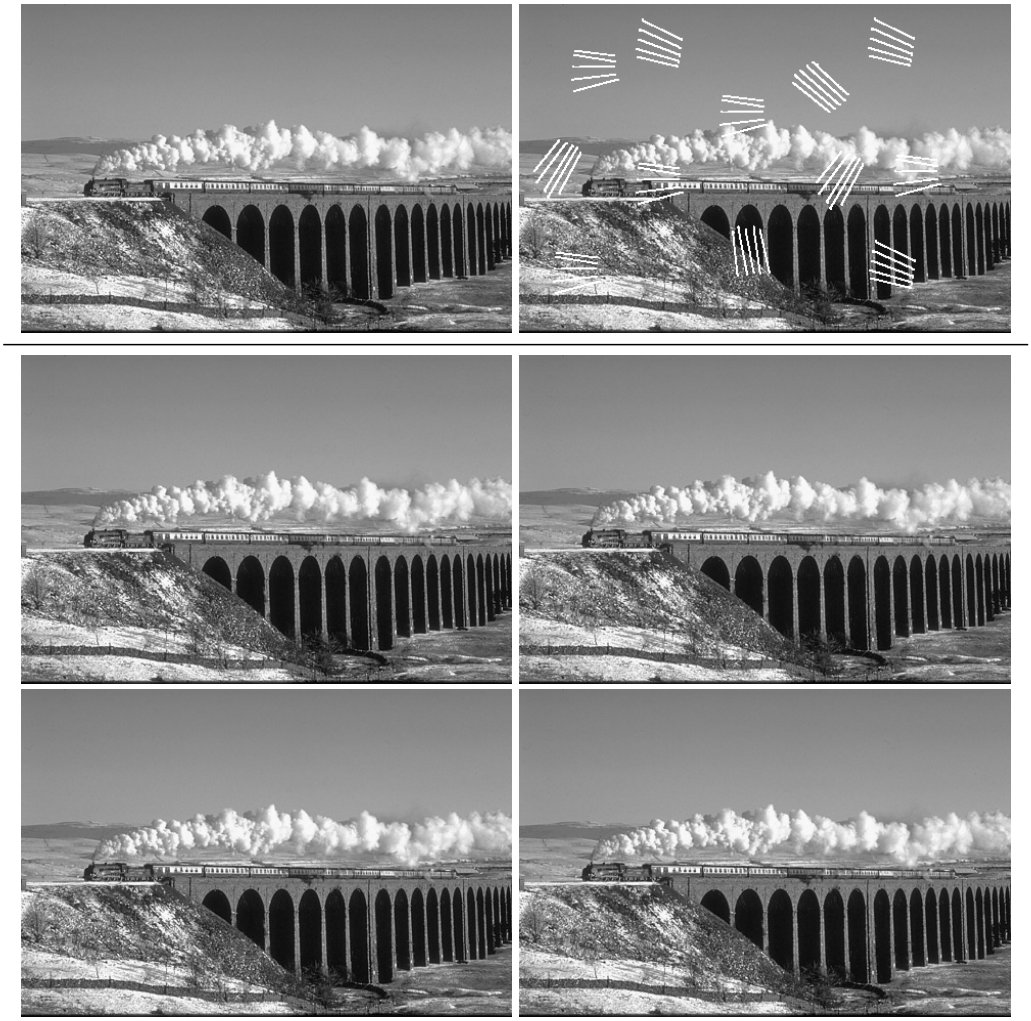


Figure 6.4: Above: the original image; the corrupted image containing 'scratches'. Below, top-left to bottom-right: mixture contains 1 Gaussian, propagation is performed with 1 Gaussian; mixture contains 3 Gaussians, propagation is performed with 1 Gaussian; propagation is performed with 3 Gaussians; propagation is performed with 9 Gaussians. All results are shown using the 2×2 model, after three iterations. See table 6.1 for more detail.

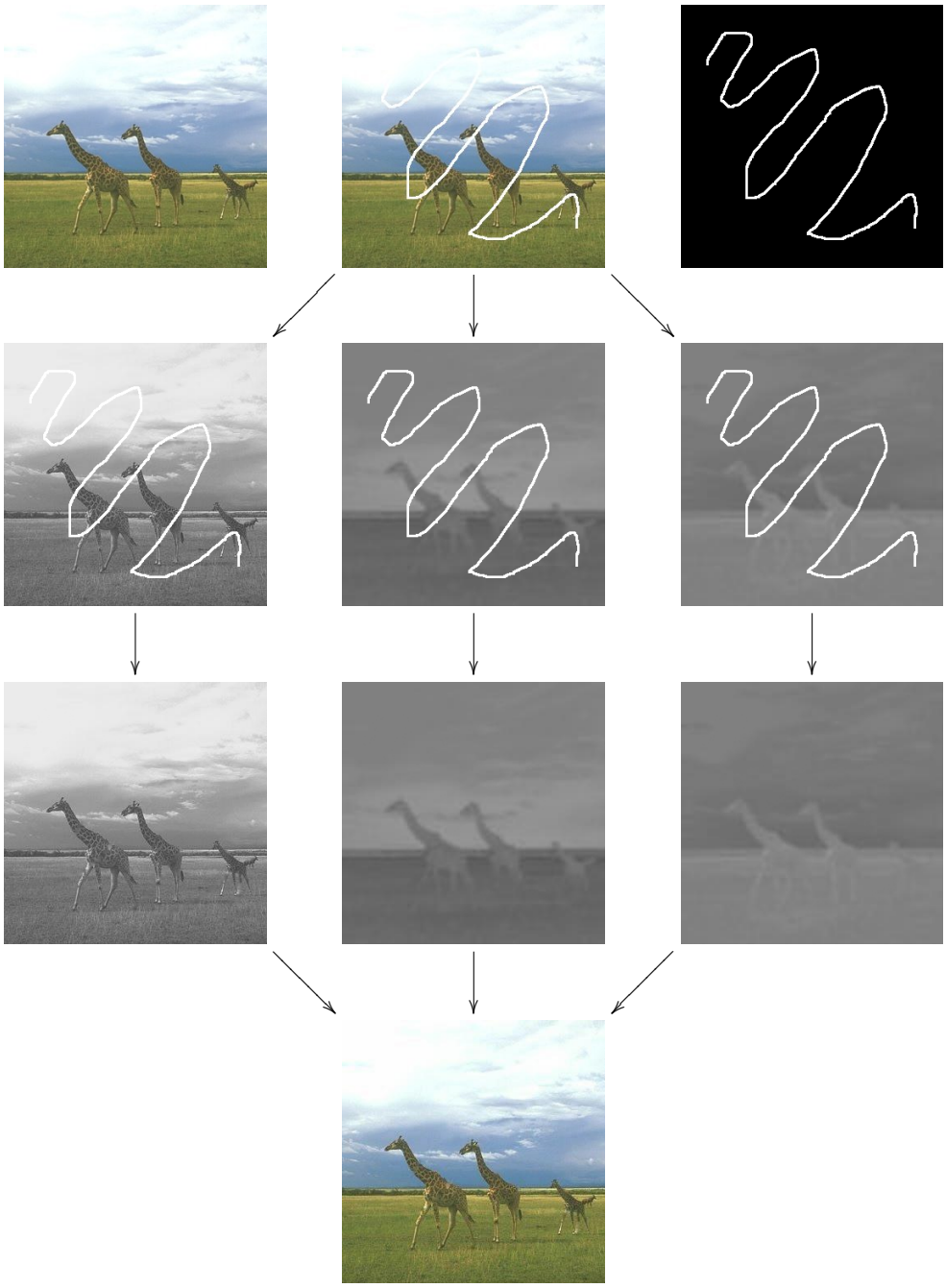


Figure 6.5: Top to bottom, left to right: the original image, the corrupted image, and the inpainting mask; the corrupted image is separated into its Y, Cb, and Cr channels; each of the channels is inpainted seperately; the three channels are combined to form the inpainted colour image.

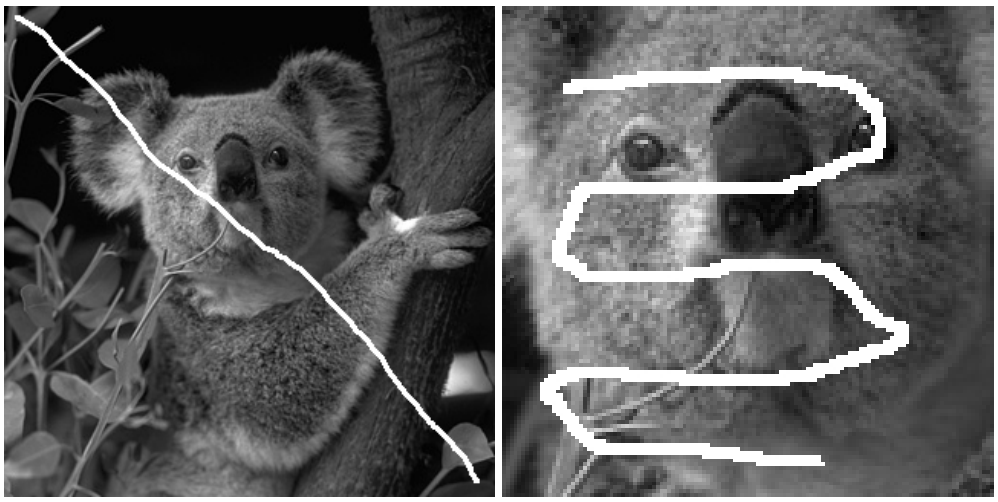


Figure 6.6: Two equally large regions to be inpainted, in two differently sized images. While the gradient-based implementation we used for comparison inpaints the smaller (right) image much faster, our algorithm inpaints both at the same speed, rendering a fair comparison more difficult.

6.5 Execution Times

Unfortunately, it proved very difficult to compare the execution times of our model with existing gradient-ascent techniques. For example, the inpainting algorithm used in [RB05] computes the gradient for all pixels using a 2-dimensional matrix convolution over the *entire* image, and then selects only the region within the inpainting mask. While this results in very fast performance when a reasonable proportion of an image is being inpainted, it results in very slow performance when the inpainting region is very sparse (as is often the case with scratches). It is easy to produce results which favour either algorithm, but such a comparison will likely be unfair.

To make explicit this difficulty, consider the images in figure 6.6. The image on the left is significantly larger than the image on the right, yet the corrupted regions are of the same size ($\sim$1500 pixels). As a result, our algorithm exhibited the same running time on both images ($\sim$10 seconds per iteration), whereas the gradient-ascent algorithm from [RB05] was approximately 6 times faster on the smaller image ($\sim$0.06 seconds compared to $\sim$0.01 seconds).

As a more representative example, when inpainting the image in figure 6.4 (using a single Gaussian), the first iteration took $\sim$33.6 seconds on our test machine. The second iteration took $\sim$39.0 (as did subsequent iterations – the first is slightly faster due to many messages being empty at this stage). The running time of this algorithm increases linearly with the number of Gaussians (for example, when using three Gaussians, the first iteration took $\sim$87.0 seconds).

Alternately, a single iteration of inpainting using the gradient-ascent algorithm from [RB05] took $\sim$0.1 seconds (using a 2×2 model). However, their code was run for 2,500 iterations,

n	Multiplications	Inverses
1	14800	44648
2	42072	37386
3	25760	12880
4	43308	21654

Table 6.2: Number of operations required by our algorithm. Multiplications are of $(n \times n) \times (n \times 1)$, and inverses are of $(n \times n)$ matrices, for various values of n.

meaning that our code is still in the order of 2 to 3 times faster. This is a pleasing result, given that we used a high-level language for our implementation.

In an attempt to provide a more 'fair' comparison, we have tried to analyse the computations required by both algorithms. It can be seen from the equations presented in chapter 5 that our algorithm consists (almost) entirely of matrix multiplications and inverses.[3] Although it is very difficult to express exactly the number of such operations required by our algorithm in general, we have calculated this number for a specific case.

The corrupted image in figure 6.4 requires us to inpaint a total of 5829 pixels. The number of operations required by our algorithm to inpaint this image (during the *second* iteration) is shown in table 6.2.

Alternately, the gradient-ascent approach in [RB05] is dominated by the time taken to compute the inner products in (the derivative of the logarithm of) equation (3.4). Each pixel is contained by four cliques, and we must compute the inner product against each of our three filters. Therefore we must compute a total of $4 \times 3 \times 5829 = 69948$ inner products per iteration.

As a simple experiment, we timed these operations in MATLAB (using random matrices and vectors). We found that computing 69948 inner products was approximately 10 times faster than computing the matrix operations shown in table 6.2. This leads us to believe that a low-level implementation of our belief-propagation algorithms may be significantly faster than the results we have shown.

[3] Other operations, such as additions and permutations are typically much faster than these.

Chapter 7

Discussion and Conclusion

7.1 Discussion

Our results have shown than even a 2×2 model is able to produce very satisfactory inpainting performance. We believe that even this small model is able to capture much of the important information about natural images. While higher-order models exist [RB05], the improvements appear to be quite incremental, despite a significant increase in their execution time. While it is certainly the case that our results fall short of the state-of-the-art in terms of PSNR, the differences are difficult to distinguish visually. It is therefore pleasing that we are able to produce competitive results within only a short period of time.

While it is difficult to shed further light on the examples we have presented, it is still possible to discuss the shortcomings of our approach, as well as different applications and extensions. These ideas will be discussed in this section.

7.1.1 Limitations of Our Approach

One possibility which we have not considered is that the corrupted pixels may contain some information about the original image. As an example for which this would be the case, consider a coffee stain – although the region behind the coffee stain has been corrupted, it still contains important information about the correct values of the pixels. In such a case, our graphical model would look more like the one presented in figure 7.1. In this case, we add an additional edge between the original (observed) pixels, and their corrected (inferred) values. Since this edge is also a maximal clique in our model, the relationships between these two nodes would also be modelled using a Gaussian mixture.

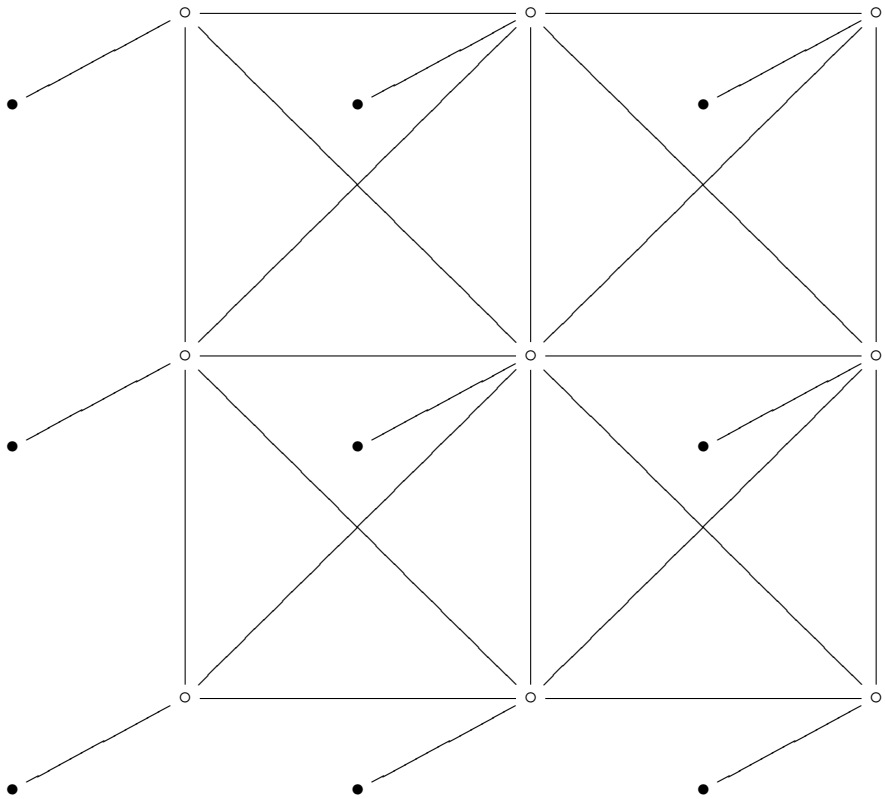


Figure 7.1: A corrupted region in which the original pixels (filled circles) contain important information about the corrupted pixels whose correct values we want to infer (empty circles).

Many gradient-ascent approaches implicitly exploit this possibility by initialising their algorithms using the corrupted pixels. If the restored image is 'close to' the corrupted image, this can result in faster convergence.

Our approach *can* deal with this case, but it is not quite as simple – we have to explicitly specify the relationship between the original pixels and their corrected values. For instance, in the case of a coffee stain, we might suggest that the Gaussian distance between the original pixels and their corrected values (i.e. $exp(-\frac{1}{2\sigma^2}(\text{original} - \text{corrected})^2)$) should be small, to prevent us from deviating too far from the observed values. However, since learning the nature of such a relationship may be quite difficult, we have not yet dealt with such problems at this stage.

Furthermore, it is clear that our algorithm will exhibit the best performance when the regions being inpainted are small, such as the scratches we have presented. In these cases, we found that loopy belief-propagation tended to converge in very few iterations. While we believe it helped that the regions we inpainted appeared to be fairly 'tree-like', there is very little theory to support this claim. On the other hand, loopy belief-propagation often converges far slower when dealing with large regions, meaning that we can inpaint a 'scratch' much faster than the coffee stain presented above.

7.1.2 Extensions

In this thesis, we have not only solved inpainting problems, but rather defined a prior model for natural images. While inpainting is an application in which our model is highly beneficial, it can potentially be applied in many cases in which we are making inferences using image priors. One such example is *image denoising*, which is in fact very similar to the example presented in figure 7.1. In an image denoising application, we assume that every pixel in the image has been corrupted by some random noise – we then want to find a correction of this image which is not only consistent with our image prior, but is also consistent with our noise model (e.g. if we are dealing with Gaussian noise, we would expect that the corrected pixel values have a small Gaussian distance to the original values, as was the case presented above).

We have not dealt with this case, simply because it would do little to highlight the advantages of our method. Gradient-based approaches to image denoising can be very fast, due to the fact that the gradient can be computed using a full matrix convolution over the whole image [RB05]. Since this matrix convolution is very costly in an inpainting scenario, our methods are better demonstrated in an inpainting setting.

Furthermore, we have not yet fully explored the possibility of using the junction-tree algorithm to inpaint images (we have only used loopy belief-propagation). Unfortunately, determining whether a graph obeys the junction-tree property (see chapter 2) is very expensive, meaning we simply used loopy belief-propagation in all cases, without even performing this test. However, there are many cases in which we can be *sure* that a junction-tree exists – for example, if the inpainting region is a scratch which is only one or two pixels wide. In such cases, optimal results can be produced after only a single iteration, which would render our algorithm several times faster again.

7.2 Conclusion

In this thesis, we have developed a model for inpainting images quickly using belief-propagation. While image inpainting has previously been performed using low-order models by belief-propagation, and high-order models by gradient-ascent, we have presented new methods which manage to exploit the benefits of both, while avoiding their shortcomings.

We have shown these algorithms to be faster than existing gradient-based techniques, even in spite of our high-level implementation. We believe that this represents an important step towards developing a variety of fast algorithms for image processing.

Appendix A

Proofs and Further Details

A.1 Proofs for Multivariate Gaussians

Here, we shall derive the formulae for marginals, conditionals, and products of multivariate Gaussians. We will only consider the case in which we have *two* Gaussians – these derivations can easily be extended to many Gaussians. Some of these ideas have been taken from [Bis06, RH05], as well as [Ahr05, Mar06]. Others have been derived independently.

Firstly, it shall help us to think of a Gaussian in terms of eigenvectors and eigenvalues, rather than its covariance matrix. We already mentioned in chapter 3 that a covariance matrix can be decomposed in terms of its eigenvectors and eigenvalues, by using an *eigenvalue decomposition*. Namely, we have that

$$\Sigma = V\Lambda V^{-1} = V\Lambda V^T \tag{A.1}$$

(where V is a matrix of eigenvectors, Λ is a diagonal matrix of eigenvalues).

In this setting, it makes sense that a Gaussian distribution may be transformed simply by transforming its eigenvectors, and its mean.

That is, for a Gaussian $\mathcal{N}(\mu, \Sigma)$, and a transformation matrix B, we have that $B\mathcal{N}(\mu, \Sigma)$ is given by

$$\begin{aligned} B\mathcal{N}(\mu, \Sigma) &= B\mathcal{N}(\mu, V\Lambda V^T) && \text{(A.2)} \\ &= \mathcal{N}(B\mu, (BV)\Lambda(BV)^T) && \text{(A.3)} \\ &= \mathcal{N}(B\mu, BV\Lambda V^T B^T) && \text{(A.4)} \\ &= \mathcal{N}(B\mu, B\Sigma B^T). && \text{(A.5)} \end{aligned}$$

A demonstration of this transformation is given in figure A.1. Here, a Gaussian distribution (shown using its eigenvectors, scaled by their eigenvalues), has its eigenvalues transformed to produce a new Gaussian distribution.

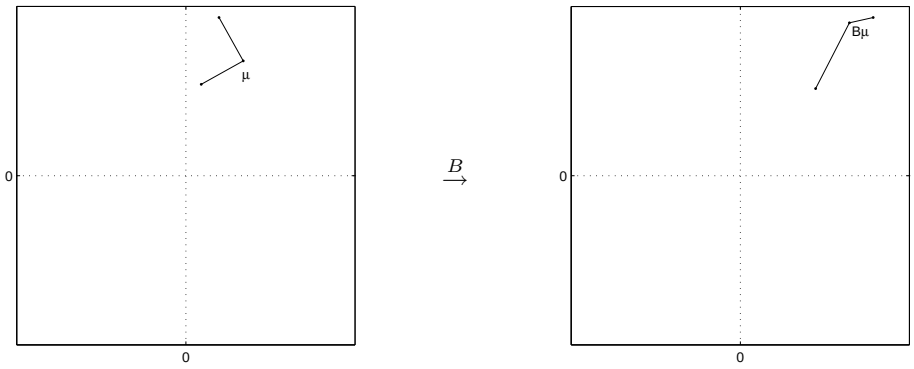


Figure A.1: The original Gaussian (left), shown using its eigenvectors (scaled by their eigenvalues); the Gaussian transformed by B (right).

From this it is clear that applying any linear transformation to a Gaussian distribution results in a new distribution which is also Gaussian.

A.1.1 Marginals

Corollary A.1.1. *Suppose* $\mathbf{x}$ *follows a multivariate Gaussian distribution, and is partitioned as* $\mathbf{x} = (\mathbf{x}_{(u)}^T \mid \mathbf{x}_{(m)})^T$*, with mean and covariance*

$$\mu = \begin{pmatrix} \mu_{(u)} \\ \mu_{(m)} \end{pmatrix} \tag{A.6}$$

and

$$\Sigma = \begin{bmatrix} \Sigma_{(u,u)} & \Sigma_{(u,m)} \\ \Sigma_{(m,u)} & \Sigma_{(m,m)} \end{bmatrix}. \tag{A.7}$$

Then the marginal $\mathbf{x}_{(u)}$ *is simply a multivariate Gaussian with mean* $\mu_{(u)}$*, and covariance* $\Sigma_{(u,u)}$*.*

Proof. Simply apply equation (A.5) with

$$B = \left[I_{|\mathbf{x}_{(u)}|} \middle| \mathbf{0}_{|(u,m)|} \right] \tag{A.8}$$

□

A.1.2 Conditionals

Theorem A.1.2. *Suppose* $\mathbf{x}$ *follows a multivariate Gaussian distribution, and is partitioned as* $\mathbf{x} = (\mathbf{x}_{(u)}^T \mid \mathbf{x}_{(o)})^T$*, with mean and covariance*

$$\mu = \begin{pmatrix} \mu_{(u)} \\ \mu_{(o)} \end{pmatrix} \tag{A.9}$$

and

$$\Sigma = \begin{bmatrix} \Sigma_{(u,u)} & \Sigma_{(u,o)} \\ \Sigma_{(o,u)} & \Sigma_{(o,o)} \end{bmatrix}. \tag{A.10}$$

The mean of the conditional distribution ($\mu_{(u;o)}$) *is now given by*

$$\mu_{(u;o)} = \mu_{(u)} + \Sigma_{(u,o)}\Sigma_{(o,o)}^{-1}(\mathbf{x}_{(o)} - \mu_{(o)}), \tag{A.11}$$

and the covariance matrix ($\Sigma_{(u;o)}$) *is given by*

$$\Sigma_{(u;o)} = \Sigma_{(u,u)} - \Sigma_{(u,o)}\Sigma_{(o,o)}^{-1}\Sigma_{(u,o)}^T. \tag{A.12}$$

Proof. Assuming that the joint distribution of $\mathbf{x}_a$ and $\mathbf{x}_b$ (which we shall denote $\pi(\mathbf{x}_a, \mathbf{x}_b)$) is Gaussian, then we have already shown (corollary A.1.1) that $\pi(\mathbf{x}_a)$ and $\pi(\mathbf{x}_b)$ also follow a Gaussian distribution.

Furthermore, it is not difficult to see that $\pi(\mathbf{x}_a|\mathbf{x}_b)$ (the conditional distribution we want to find) is also Gaussian. This follows from the fact that $\pi(\mathbf{x}_a|\mathbf{x}_b) = \frac{\pi(\mathbf{x}_a, \mathbf{x}_b)}{\pi(\mathbf{x}_b)}$, where $\pi(\mathbf{x}_b)$ is merely a constant in this case. Furthermore, by substituting specific values into the joint distribution, it is easy to solve the resulting equation in terms of a new mean and covariance. It is only the specific computational equation for the conditional distribution which is not obvious.

So, we will simply state that $\pi(\mathbf{x}_a|\mathbf{x}_b) \sim \mathcal{N}(\alpha + \beta\mathbf{x}_b, \Sigma_e)$. Our task is now to solve for α, β, and Σ_e.

In the above expression, we have essentially described the conditional distribution of $\mathbf{x}_a$ given $\mathbf{x}_b$ as a *linear model* in terms of $\mathbf{x}_b$. While we shall not delve too deeply into the theory of linear models here, we require one simple theorem – namely, that the *residual* of the model (i.e. the difference between the value predicted by the linear model and the true value), is independent of the predictor (i.e. the input value). This is true as long as our response is indeed of the form $\alpha + \beta\mathbf{x} + \mathcal{N}(\mu, \Sigma)$ (i.e. it is linear, together with some random error), which is certainly the case here.

Here, we have that the residual is given by

$$\pi(E) = \pi(\mathbf{x}_a) - (\alpha + \beta\pi(\mathbf{x}_b)). \tag{A.13}$$

(in this case, the 'random error' in our response is determined by Σ_e). Hence, our theorem from linear models simply states that $\pi(E)$ is independent of $\pi(\mathbf{x}_b)$.

Furthermore, when two Gaussian distributions are independent, their joint distribution has a particularly simple form. Specifically, there is no covariance between the two distributions.

Hence, the covariance matrix is block-diagonal. In this case, the joint distribution of E and $\mathbf{x}_b$ is given by

$$\pi(E, \mathbf{x}_b) \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ \mu_{(b)} \end{pmatrix}, \begin{bmatrix} \Sigma_e & 0 \\ 0 & \Sigma_{(b,b)} \end{bmatrix}\right). \tag{A.14}$$

We can now express the joint distribution, $\pi(\mathbf{x}_a, \mathbf{x}_b)$, as an affine transformation of $\pi(E, \mathbf{x}_b)$. Specifically, we have[1]

$$\begin{aligned} \pi(\mathbf{x}_a, \mathbf{x}_b) &= \begin{pmatrix} \alpha \\ 0 \end{pmatrix} + \begin{bmatrix} I & \beta \\ 0 & I \end{bmatrix} \pi(E, \mathbf{x}_b) && \text{(A.15)} \\ &\sim \mathcal{N}\left(\begin{pmatrix} \mu_{(a)} \\ \mu_{(b)} \end{pmatrix}, \begin{bmatrix} \Sigma_{(a,a)} & \Sigma_{(a,b)} \\ \Sigma_{(b,a)} & \Sigma_{(b,b)} \end{bmatrix}\right). && \text{(A.16)} \end{aligned}$$

We can now use this fact to solve the above expression in terms of α, β, and Σ_e (again using equation (A.5)).

Here we have

$$\begin{pmatrix} \mu_{(a)} \\ \mu_{(b)} \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \end{pmatrix} + \begin{bmatrix} I & \beta \\ 0 & I \end{bmatrix} \begin{pmatrix} 0 \\ \mu_{(b)} \end{pmatrix} = \begin{pmatrix} \alpha + \beta\mu_{(b)} \\ \mu_{(b)} \end{pmatrix} \tag{A.17}$$

and

$$\begin{aligned} \begin{bmatrix} \Sigma_{(a,a)} & \Sigma_{(a,b)} \\ \Sigma_{(b,a)} & \Sigma_{(b,b)} \end{bmatrix} &= \begin{bmatrix} I & \beta \\ 0 & I \end{bmatrix} \begin{bmatrix} \Sigma_e & 0 \\ 0 & \Sigma_{(b,b)} \end{bmatrix} \begin{bmatrix} I & 0 \\ \beta^T & I \end{bmatrix} && \text{(A.18)} \\ &= \begin{bmatrix} \Sigma_e + \beta\Sigma_{(b,b)}\beta^T & \beta\Sigma_{(b,b)} \\ \Sigma_{(b,b)}\beta^T & \Sigma_{(b,b)} \end{bmatrix}. && \text{(A.19)} \end{aligned}$$

Now we can easily solve our system

$$\begin{aligned} \beta &= \Sigma_{(a,b)}\Sigma_{(b,b)}^{-1} && \text{(A.20)} \\ \alpha &= \mu_{(a)} - \beta\mu_{(b)} && \text{(A.21)} \\ \Sigma_e &= \Sigma_{(a,a)} - \Sigma_{(a,b)}\Sigma_{(b,b)}^{-1}\Sigma_{(b,a)}. && \text{(A.22)} \end{aligned}$$

Simple substitution now gives us the required result. □

A.1.3 Products

Theorem A.1.3. *Given Gaussians $\mathcal{N}(\mu_a, \Sigma_a)$ and $\mathcal{N}(\mu_b, \Sigma_b)$, the covariance of the product (Σ_c) is given by*

$$\Sigma_c = (\Sigma_a^{-1} + \Sigma_b^{-1})^{-1} \tag{A.23}$$

The mean of the product (μ_c) is given by

$$\mu' = \Sigma_c(\Sigma_a^{-1}\mu_a + \Sigma_b^{-1}\mu_b). \tag{A.24}$$

[1] Here we have ignored the dimensions of some matrices where there is no ambiguity.

Proof. Explicitly, the product is proportional to

$$exp(-\frac{1}{2}\left[(\mathbf{x}-\mu_a)^T Q_a(\mathbf{x}-\mu_a)+(\mathbf{x}-\mu_b)^T Q_b(\mathbf{x}-\mu_b)\right]) \tag{A.25}$$

(where we have used Q_a to represent Σ_a^{-1} to be more concise). Expanding the term within the exponent gives

$$\begin{aligned}
& (\mathbf{x}-\mu_a)^T Q_a(\mathbf{x}-\mu_a)+(\mathbf{x}-\mu_b)^T Q_b(\mathbf{x}-\mu_b) \\
= \;& \mathbf{x}^T Q_a\mathbf{x}-\mathbf{x}^T Q_a\mu_a-\mu_a^T Q_a\mathbf{x}+\mathbf{x}^T Q_b\mathbf{x}-\mathbf{x}^T Q_b\mu_b-\mu_b^T Q_b\mathbf{x}+K \\
= \;& \mathbf{x}^T(Q_a+Q_b)\mathbf{x}-\mathbf{x}^T(Q_a\mu_a+Q_b\mu_b)-(Q_a\mu_a+Q_a\mu_b)^T\mathbf{x}+K \\
= \;& \mathbf{x}^T(Q_a+Q_b)\mathbf{x}-\mathbf{x}^T(Q_a+Q_b)(Q_a+Q_b)^{-1}(Q_a\mu_a+Q_b\mu_b) \\
& -(Q_a\mu_a+Q_b\mu_b)^T(Q_a+Q_b)^{-1}(Q_a+Q_b)\mathbf{x}+K \\
= \;& (\mathbf{x}-(Q_a+Q_b)^{-1}(Q_a\mu_a+Q_b\mu_b))^T(Q_a+Q_b)(\mathbf{x}-(Q_a+Q_b)^{-1}(Q_a\mu_a+Q_b\mu_b))+K'
\end{aligned}$$

(the fact that $K \neq K'$ does not matter, as they only serve as scaling coefficients). This technique is similar to that of completing the square; rewriting $Q_a = \Sigma_a^{-1}$ etc. gives the required solution. □

Appendix B

ICML 2007 Paper

We decided that it would be worthwhile to submit our results to the International Conference of Machine Learning (ICML2007). ICML is well-known for publishing papers on topics such as graphical models, and learning in images. If accepted, these results will be presented in June 2007.

The full text of this paper is included below. As this paper is currently under review, it should not be distributed.

High-Order Nonparametric Belief-Propagation for Fast Image Inpainting

Abstract

In this paper, we use belief-propagation techniques to develop fast algorithms for image inpainting. Unlike traditional gradient-based approaches, which may require many iterations to converge, our techniques achieve competitive results after only a few iterations. On the other hand, while belief-propagation techniques are often unable to deal with high-order models due to the explosion in the size of messages, we avoid this problem by approximating our high-order prior model using a Gaussian mixture. By using such an approximation, we are able to inpaint images quickly while at the same time retaining good visual results.

1. Introduction

In order to restore a corrupted image, one needs a model of how uncorrupted (i.e. natural) images appear. In the Markov random field Bayesian paradigm for image restoration (Geman & Geman, 1984), natural images are modeled via an *image prior*. This is a probabilistic model that encodes how natural images behave *locally* (in the vicinity of every pixel). An inference algorithm is then used to restore the image, whose aim is to find a consensus among all vicinities on which global solution is most compatible with a natural image (while still similar to the corrupted image—a corrupted tree should be restored as an uncorrupted tree, not as an uncorrupted house). The simplest example of an image prior is perhaps the pairwise model presented in (Geman & Geman, 1984) – which simply expresses that neighboring pixels are likely to share similar gray-levels. However, it is easy to see that such a model fails to capture a great deal of important information about natural images. For example, 'edges' are highly penalized by such a prior, and it is unable to encode any information about 'texture'. Only by using higher-order priors will one be able to capture this important information.

An example of a high-order prior is the *field of experts* model (Roth & Black, 2005), which is parameterized as the product of a selection of filters (or 'experts'). Each of these filters is typically a patch of 3×3 or 5×5 pixels, resulting in a 9 or 25-dimensional prior respectively. Unfortunately, such a high-dimensional prior limits the practicality of many inference algorithms. Even though it may be possible to use smaller (for example, 2×2) patches (Lan et al., 2006), we are still limited by the number of gray-levels used to properly represent natural images (typically 256). Updating a single pixel using a Gibbs sampler (for example) requires us to consider all 256 possible gray-levels. Since Gibbs samplers may typically take hundreds (or thousands) of iterations to converge, they are simply impractical in this setting.

While belief-propagation techniques tend to converge in fewer iterations (Yedidia et al., 2000), they are often equally impractical. Since adjacent cliques may share as many as 6 nodes (using a 3×3 model), the size of the messages passed between them may be as large as 256^6. Even if we only use 2×2 cliques, the size of our messages may still be as large as 256^2, which remains impractical for many purposes.

To avoid these problems, image restoration is typically performed using gradient-ascent, thereby eliminating the need to deal with many discrete gray-levels, and avoiding expensive sampling (Roth & Black, 2005). While gradient-based approaches are generally considered to be fast, they may still require several thousand iterations in order to converge, and even then will converge only to a *local* optimum.

In this paper, we propose a method that enables one to get the best of both worlds: we manage to render belief-propagation practical using a high-order (2×2 pixels or larger) model and use it for the task of image inpainting. By using a nonparametric prior, we are able to avoid the need to discretize images, resulting in much smaller messages being passed between cliques. Our experiments show that belief-propagation

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

techniques are able to produce competitive results after only a single iteration, rendering them faster than many gradient-based approaches, while retaining similar visual quality of the restoration.

2. Background

In this section, we define the Markov random field (MRF) image prior to be used in our model. Although we shall not present any significant results in terms of *learning* the prior, we have nevertheless made a number of modifications to the 'standard' image prior in order to render inference tractable.

2.1. The 'Field of Experts' Image Prior

The Hammersley-Clifford theorem states that the joint probability distribution of a Markov random field with clique set $\mathcal{C}$ (assuming maximal cliques) is given by

$$p(\mathbf{x}) = \frac{1}{Z}\prod_{c\in\mathcal{C}}\phi_c(\mathbf{x}_c) \tag{1}$$

(where $\mathbf{x}_c$ is the set of variables in $\mathbf{x}$ belonging to the c^{th} clique; Z is a normalization constant) (Geman & Geman, 1984). When dealing with images, the ϕ_cs are often assumed to be homogeneous (Burton & Moorhead, 1987), meaning that the prior can be defined entirely in terms of a single potential function, ϕ. In the Field of Experts model (Roth & Black, 2005), this potential function is assumed to take the form of a *product of experts* (Hinton, 1999), in which each 'expert' is the response of the image patch $\{\mathbf{x}_c\}$ to a particular filter $\{J_f\}$. That is, the potential function takes the form

$$\phi(\mathbf{x}_c; J, \alpha) = \prod_{f=1}^{F}\phi'_f(\mathbf{x}_c, J_f, \alpha_f) \tag{2}$$

(where the α_f's are simply weighting coefficients controlling the importances of the filters). Specifically, each expert is assumed to take the form of a Student's T-distribution, namely

$$\phi'_f(\mathbf{x}_c; J_f, \alpha_f) = (1 + \frac{1}{2}\langle J_f, \mathbf{x}_c\rangle^2)^{-\alpha_f}. \tag{3}$$

Although (Roth & Black, 2005) use contrastive divergence learning to select the filters and alphas, it has been shown that the filters can more easily be selected using principal component analysis (PCA) (McAuley et al., 2006). This leaves only the problem of learning the alphas, which we shall deal with in section 3.

2.2. Belief-Propagation

Inference in the MRF setting can be formulated as a message passing problem. Two common message passing algorithms exist, namely the *junction-tree algorithm*, and *loopy belief-propagation* (Aji & Mceliece, 2001). In our case, which algorithm should be applied depends upon the 'shape' of the region being inpainted. We will give only a brief overview of these algorithms in order to explain why it is infeasible to apply them directly when using the above prior. A more complete specification is given in (Aji & Mceliece, 2001).

Belief-propagation algorithms work by having cliques pass 'messages' to other cliques which share one or more nodes in common. If we denote by $S_{i,j}$ the intersection of the two cliques $\mathbf{x}_i$ and $\mathbf{x}_j$, and denote by $\Gamma_{\mathbf{x}_i}$ the neighbors of $\mathbf{x}_i$ (i.e. those cliques which share one or more nodes with $\mathbf{x}_i$), then the message, $M_{i\to j}$, sent from $\mathbf{x}_i$ to $\mathbf{x}_j$ is given by

$$M_{i\to j}(S_{i,j}) = \sum_{\mathbf{x}_i\setminus\mathbf{x}_j}\phi(\mathbf{x}_i)\prod_{\mathbf{x}_k\in(\Gamma_{\mathbf{x}_i}\setminus\{\mathbf{x}_j\})}M_{k\to i}(S_{k,i}). \tag{4}$$

That is, the outgoing message from $\mathbf{x}_i$ to $\mathbf{x}_j$ is defined as the product of the local potential $\{\phi(\mathbf{x}_i)\}$ with the incoming messages from all neighbors *except* $\mathbf{x}_j$, marginalized over the variables in $\mathbf{x}_i$ but not in $\mathbf{x}_j$. Once all messages have been sent, the final distribution of $\mathbf{x}_i$ $\{D_i(\mathbf{x}_i)\}$ is given by

$$D_i(\mathbf{x}_i) = \phi(\mathbf{x}_i)\prod_{\mathbf{x}_k\in\Gamma_{\mathbf{x}_i}}M_{k\to i}(S_{k,i}). \tag{5}$$

Even when using the 2×2 model, evaluating $\phi(\mathbf{x}_i)$ requires us to consider 256^4 possible gray-level combinations. Although it may be possible to approximate the marginal being computed in equation (4) without computing $\phi(\mathbf{x}_i)$ explicitly (Lan et al., 2006), the message itself still contains 256^2 elements. This problem is dealt with in (Lan et al., 2006) by using a *factor-graph* (Kschischang et al., 2001), which requires only that one dimensional marginals are computed, however the running time of their method is still linear in the number of gray-levels, in addition to the fact that the factor-graph fails to fully capture the conditional independencies implied by the model.

As a result, we seek ϕ in such a form that the sum in equation (4) may be replaced by an integral. In (Sudderth et al., 2003), the authors defined such a model in which the potential function takes the form of a Gaussian mixture, that is, with ϕ taking the form

$$\phi(\mathbf{x}_c) = \sum_{i=1}^{N}\beta_i e^{(\mathbf{x}_c-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}_c-\mu_i)} \tag{6}$$

(this is sometimes known as a *Gaussian random field* (Bishop, 2006; Rue & Held, 2005)). Unfortunately,

the method they use to learn these mixtures appears to be applicable only to low-order models (the largest mixture models they learn are 3-dimensional).

In the remainder of this paper, we will show that the experts $\{\phi'_f\text{s}\}$ can be approximated as a Gaussian mixture, resulting in a high-order model which closely matches the one given in equations (2) and (3). We will describe belief-propagation in this setting, and show how this approach can be used for fast image inpainting.

3. Approximating the Prior

In (McAuley et al., 2006), the authors showed that the filters $\{J_f\text{s}\}$ in equation (2) can be learned by performing PCA on a collection of natural image patches. Here we follow this idea. To learn our filters (for a 2×2 model), we randomly cropped 50,000 2×2 patches from images in the Berkeley Segmentation Database (Martin et al., 2001), and used the *last three* principal components (it was found in (Roth & Black, 2005) that the first component corresponds to a uniform gray patch, which should be ignored for a model invariant to intensity). The three resulting patches appear to make sense visually, and are shown in figure 1.

Figure 1. The three filters used in our 2×2 model.

This model requires also that we learn the 'importances' $\{\alpha_f\}$ of each filter. From equation (3), it can be seen that the α_fs simply control the shape (or 'peakedness') of the Student's T-distribution. Rather than try to learn the α_fs explicitly, we will learn them implicitly through our approximation.

In order to approximate the experts $\{\phi'_f\}$, we first calculated the inner products $\{\langle J_f, \mathbf{x}_c\rangle\}$ for a random selection of 5,000 image patches (again cropped from the Berkeley segmentation database (Martin et al., 2001)). Rather than assume that this data is generated according to a Student's T-distribution, we simply tried to approximate this data directly using a mixture of Gaussians. We observed (from a normal probability plot) that the data was more heavily tailed than would be suggested by a normal distribution, indicating that the Student's T-distribution may indeed be valid.

In order to estimate the distribution governing this data, we used the expectation-maximization (EM) algorithm (Dempster et al., 1977), assuming that the set of inner products for each filter was generated by a mixture of three Gaussians. All of our parameters to be learned $\{\Theta = (\beta, \mu, \sigma)\}$ were initialized by using a K-means clustering (MacQueen, 1967) on the original inner products. We used this approach to learn a separate mixture model for each expert. This algorithm produces an approximation of the form

$$\phi'_f(\mathbf{x}_c; \Theta, J) \simeq \sum_{i=1}^{3} \beta_{f,i} \exp\left(\frac{(\langle J_f, \mathbf{x}_c\rangle - \mu_{f,i})^2}{2\sigma_{f,i}^2}\right). \tag{7}$$

The alpha terms are no longer relevant – the 'shape' of the distribution is implicitly controlled by the other parameters. However, the expression in equation (7) is not yet in the same form as equation (6). Hence we need to solve the system

$$\exp\left(\frac{(\langle J, \mathbf{x}\rangle - \mu)^2}{2\sigma^2}\right) = \exp\left((\mathbf{x} - \underline{\mu})^T \Sigma^{-1} (\mathbf{x} - \underline{\mu})\right). \tag{8}$$

That is, we are trying to solve for Σ^{-1} (a matrix) and $\underline{\mu}$ (a vector), in terms of J (a vector) and μ (a scalar). It is not difficult to see that the only solution for Σ^{-1} is

$$\Sigma^{-1} = \frac{1}{2\sigma^2} J \cdot J^T = \frac{1}{2\sigma^2} \begin{bmatrix} J_1^2 & J_1 J_2 & \cdots & J_1 J_n \\ J_2 J_1 & J_2^2 & & J_2 J_n \\ \vdots & & \ddots & \vdots \\ J_n J_1 & J_n J_2 & \cdots & J_n^2 \end{bmatrix} \tag{9}$$

(where n is the size of the filter J – in our case, $n = 4$). Alternately, there are infinite solutions for $\underline{\mu}$. One obvious solution is

$$\underline{\mu} = \begin{pmatrix} \frac{\mu}{\sum_{i=1}^{n} J_i} \\ \vdots \\ \frac{\mu}{\sum_{i=1}^{n} J_i} \end{pmatrix}. \tag{10}$$

However, we found for all of our filters that $\sum_{i=1}^{n} J_i \simeq 0$, meaning that this solution would be highly unstable. A more stable solution (which we used) is given by

$$\underline{\mu} = \begin{pmatrix} \mu/J_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{11}$$

Our potential function is now of the form

$$\phi(\mathbf{x}_c; \Theta, J) = \prod_{f=1}^{3} \sum_{i=1}^{3} \beta_{f,i} \exp\left(\frac{(\langle J_f, \mathbf{x}_c\rangle - \mu_{f,i})^2}{2\sigma_{f,i}^2}\right). \tag{12}$$

In order to expand the above product, we use the following result about the product of Gaussian distributions (Ahrendt, 2005): given K Gaussians (with means $\mu_1 \dots \mu_K$, and covariances $\Sigma_1 \dots \Sigma_K$), the covariance of the product $\{\Sigma'\}$ is given by

$$\Sigma' = (\sum_{i=1}^{K} \Sigma_i^{-1})^{-1} \tag{13}$$

(although each Σ_i^{-1} is singular in our case, their sum is not). The mean of the product $\{\mu'\}$ is given by

$$\mu' = \Sigma'(\sum_{i=1}^{K} \Sigma_i^{-1}\mu_i). \tag{14}$$

The corresponding beta term for the product is just $\beta' = \prod_{i=1}^{K} \beta_k$. In our case this results in a final approximation which is a mixture of $3^3 = 27$ Gaussians.

4. Inference

In order to perform belief-propagation, we must first be able to express equations (4) and (5) in terms of the Gaussian mixtures we have defined. In our setting, the sum in equation (4) becomes an integral, resulting in the new equation

$$M_{i\to j}(S_{i,j}) = \int_{\mathbf{x}_i \setminus \mathbf{x}_j} \phi(\mathbf{x}_i) \prod_{\mathbf{x}_k \in (\Gamma_{\mathbf{x}_i} \setminus \{\mathbf{x}_j\})} M_{k\to i}(S_{k,i}). \tag{15}$$

We have already suggested how to perform the above multiplication in equations (13) and (14). The only difference in this case is that the mixtures for each message may contain fewer variables (and smaller covariance matrices) than the local distribution $\{\phi(\mathbf{x}_i)\}$. In such a case, the inverse covariance matrices for each message $\{\Sigma^{-1}\text{s}\}$ are simply assumed to be zero in all missing variables.

To compute the marginal distribution of a Gaussian mixture with mean μ and covariance matrix Σ (i.e. the integral in equation (15)), we simply take the elements of μ and Σ corresponding to the variables whose marginals we want. The importances for each Gaussian in the mixture remain the same.

Of course, when we compute the products in equation (15), we produce a model with an exponentially increasing number of Gaussians. As a simple solution to this problem, we restrict the maximum number of Gaussians to a certain limit (see section 5), by including only those with the highest importances.

When solving an inpainting problem, we only wish to treat some of the variables in each clique as unknowns (for example, the 'scratched' sections). Hence the potential function for these cliques should be conditioned upon the 'observed' regions of the image. Suppose that for a clique c we have unknowns $\mathbf{x}_{(u)}$, and observed variables $\mathbf{x}_{(o)}$ (i.e. $\mathbf{x}_c = (\mathbf{x}_{(u)}^T; \mathbf{x}_{(o)}^T)^T$). Then we may partition the mean and covariance matrix (for a particular Gaussian in the mixture) as

$$\mu = \begin{pmatrix} \mu_{(u)} \\ \mu_{(o)} \end{pmatrix} \tag{16}$$

and

$$\Sigma = \begin{bmatrix} \Sigma_{(u,u)} & \Sigma_{(u,o)} \\ \Sigma_{(o,u)} & \Sigma_{(o,o)} \end{bmatrix}. \tag{17}$$

The mean of the conditional distribution $\{\mu_{(u;o)}\}$ is now given by

$$\mu_{(u;o)} = \mu_{(u)} + \Sigma_{(u,o)}\Sigma_{(o,o)}^{-1}(\mathbf{x}_{(o)} - \mu_{(o)}), \tag{18}$$

and the covariance matrix $\{\Sigma_{(u;o)}\}$ is given by

$$\Sigma_{(u;o)} = \Sigma_{(u,u)} - \Sigma_{(u,o)}\Sigma_{(o,o)}^{-1}\Sigma_{(u,o)}^T. \tag{19}$$

Finally, once all messages have been propagated, we are able to compute the marginal distribution for a given node (or pixel, belonging to clique c) by marginalizing $D_c(\mathbf{x}_c)$ (equation (5)) in terms of that node. In order to estimate the 'most likely' configuration for this pixel, we simply consider each of the 256 possible gray-levels.[1]

4.1. Propagation Methods

As we mentioned in section 2.2, the two propagation techniques we will deal with are the junction-tree algorithm and loopy belief-propagation. Although we will not cover these in great detail (see (Aji & Mceliece, 2001) for a more complete exposition), we will explain the differences between the two in terms of image inpainting.

Both algorithms work by passing messages between those cliques with non-empty intersection. However, when using the junction-tree algorithm, we connect only enough cliques to form a maximal spanning tree.

[1] Although this final step may appear to make the running time of our solution linear in the number of gray-levels, it should be noted that this step needs to be performed only once, after the final iteration. It should also be noted that this estimate only requires us to measure the response of a one-dimensional Gaussian, which is inexpensive. More sophisticated mode-finding techniques exist (Carreira-Perpiñán, 2000), which we considered to be unnecessary in this case. Finally, note that this step is not required when our mixture contains only a single Gaussian, in which case we simply select the mean.

Now, suppose that two cliques c_a and c_b have intersection $S_{a,b}$. If each clique along the path between them also contains $S_{a,b}$, we say that this spanning tree obeys the 'junction-tree property'. If this property holds, it can be proven that exact inference is possible (subject only to the approximations used by our Gaussian model), and requires that messages be passed only for a single iteration (Aji & Mceliece, 2001). Although it is hard to concisely characterize those graphs which obey this property (technically, *triangulated* graphs), they tend to be those which are 'tree-like'.

If this property *doesn't* hold, then we may resort to using loopy belief-propagation, in which case we simply connect *all* cliques with non-empty intersection. There is no longer any message passing order for which equation (15) is well defined (i.e. we must initially assume that some messages correspond to a uniform distribution), meaning that messages must be passed for many iterations in the hope that they will converge.

Figure 2 shows an inpainting problem for which a junction-tree exists, and two problems for which one does not (assuming (2×2)-pixel cliques). Since the regions being inpainted are usually thin lines (or 'scratches'), we will often observe graphs which do in fact obey the junction-tree property.

Figure 2. The graph formed from the white pixels in the left image forms a junction-tree (assuming a 2×2 model). The graphs formed from the white pixels in the other two images do not.

Fortunately, we found that even in those cases where no junction-tree existed, loopy belief-propagation tended to converge in very few iterations. Although there are few theoretical results to justify this behavior, loopy-belief propagation typically converges quickly in those cases where the graph *almost* forms a tree (as is usually the case for the regions being inpainted).

5. Experimental Results

In order to perform image inpainting, we used a high-level (Python) implementation of the junction-tree algorithm and loopy belief-propagation, which is capable of constructing Markov random fields with any topology.[2] Despite being written in a high-level language, our implementation is able to inpaint images within a reasonably short period of time. Since it is difficult to assess the quality of our results visually, we have reported both the peak signal-to-noise ratio (PSNR), and the structured similarity (SSIM) (Wang et al., 2004).

Table 1. Comparison of inpainting performance for several models. Here we vary the number of Gaussians used to compute the initial mixture, as well as the maximum number of Gaussians allowed during propagation.

Gaussians	Max	Iter.	PSNR	SSIM
1	1	1	22.57	0.927
		2	22.67	0.928
		3	22.68	0.928
3	1	1	22.81	0.927
		2	22.87	0.928
		3	22.88	0.928
3	3	1	22.82	0.927
		2	22.87	0.928
		3	22.88	0.928
3	9	1	22.80	0.927
		2	22.86	0.928
		3	22.87	0.928

Figure 3 shows a corrupted image from which we want to remove the text. The image has been inpainted using a model containing only a single Gaussian (although the learned mixtures contained three Gaussians – see below). After a single iteration, most of the text has been removed, and after two iterations it is almost completely gone. Although the current state-of-the-art inpainting techniques produce superior results in terms of PSNR (Roth & Black, 2005), they give similar visual results and take several thousand iterations to converge, compared to ours which takes only two (no further improvement was observed after a third iteration).

Figure 4 compares models of various sizes, varying both the number of Gaussians used to approximate each mixture, as well as the maximum number of Gaussians allowed during the inference stage. The same results are summarized in table 1.

The top-right image in figure 4 was produced using a model in which each expert was approximated using three Gaussians, yet only one Gaussian was allowed during propagation. In contrast, the model used to produce the top-left image was approximated using only a *single* Gaussian. Interestingly, the for-

[2]Our implementation is available at `http://??` (not included for blind submission).

Figure 3. Above, top-left to bottom-right: the original image; the image containing the text to be removed; inpainting after a single iteration, using a single Gaussian (PSNR = 22.74, SSIM = 0.962); inpainting after two iterations (PSNR = 22.82, SSIM = 0.962). Below: close-ups of all images.

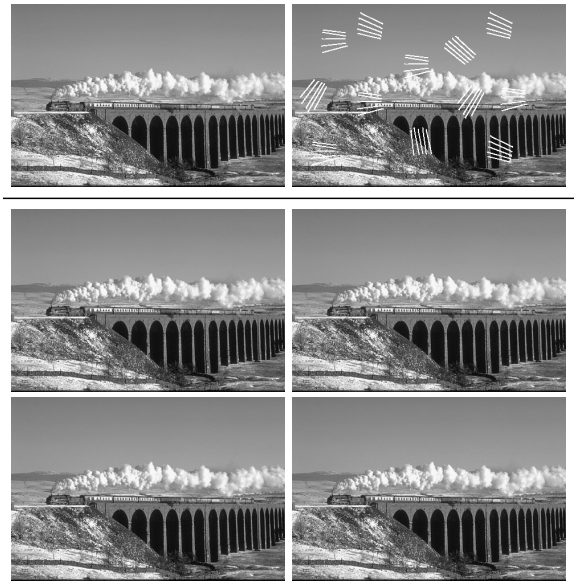


Figure 4. Above: the original image; the corrupted image containing 'scratches'. Below, top-left to bottom-right: mixture contains 1 Gaussian; mixture contains 3 Gaussians, propagation is performed with 1 Gaussian; propagation is performed with 3 Gaussians; propagation is performed with 9 Gaussians. All results are shown using the 2×2 model, after three iterations. See table 1 for more detail.

mer model actually outperformed the latter in this experiment. While this result may seem surprising, it may be explainable as follows: in the single-Gaussian model, the standard deviation is overestimated in order to compensate for the high kurtosis of the training data (Gosset, 1942). However, in the model containing three Gaussians, the most significant Gaussian (i.e. the Gaussian with the highest β term) captures only the most of the 'important' information about the distribution, and ignoring the other two is not very harmful.

Furthermore, given that increasing the maximum number of Gaussians allowed during propagation does not seem to significantly improve inpainting performance, we suggest that this single-Gaussian model may be the most practical. Even after only a single iteration, the results are visually pleasing.

5.1. Execution Times

Unfortunately, it proved very difficult to compare the execution times of our model with existing gradient-ascent techniques. For example, the inpainting algorithm used in (Roth & Black, 2005) computes the gradient for all pixels using a 2-dimensional matrix convolution over the *entire* image, and then selects only the region corresponding to the inpainting mask. While this results in very fast performance when a reasonable proportion of an image is being inpainted, it results in very slow performance when the inpainting region is very sparse (as is often the case with scratches). It is easy to produce results which favor either algorithm, but such a comparison will likely be unfair.

To make explicit this difficulty, consider the images in figure 5. The image on the left is significantly larger than the image on the right, yet the corrupted regions are of the same size ($\sim$1500 pixels). As a result, our algorithm exhibited the same running time on both images, whereas the gradient-ascent algorithm from (Roth & Black, 2005) was approximately 6 times slower on the larger image.

As a more representative example, when inpainting the

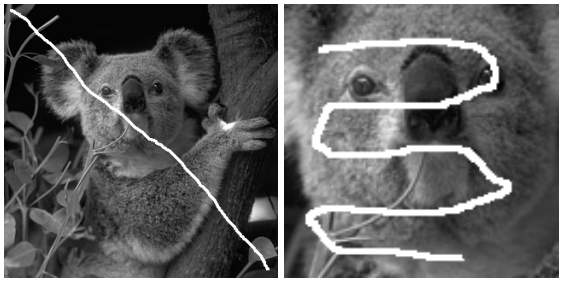


Figure 5. Two equally large regions to be inpainted, in two differently sized images.

image in figure 4 (using a single Gaussian), the first iteration took ∼33.6 seconds on our test machine. The second iteration took ∼39.0 (as did subsequent iterations – the first is slightly faster due to many messages being empty at this stage). The running time of this algorithm increases linearly with the number of Gaussians (for example, when using three Gaussians, the first iteration took ∼87.0 seconds).

Alternately, a single iteration of inpainting using the gradient-ascent algorithm from (Roth & Black, 2005) took ∼0.1 seconds (using a 2×2 model). However, their code was run for 2,500 iterations, meaning that our code is still in the order of 2 to 3 times faster. This is a pleasing result, given that we used a high-level language for our implementation.

However, in an attempt to provide a more 'fair' comparison, we have tried to analyze the computations required by both algorithms. It can be seen from the equations presented in section 4 that our algorithm consists (almost) entirely of matrix multiplications and inverses.[3] Although it is very difficult to express exactly the number of such operations required by our algorithm in general, we have calculated this number for a specific case.

The corrupted image in figure 4 requires us to inpaint a total of 5829 pixels. The number of operations required by our algorithm to inpaint this image (during the *second* iteration) is shown in table 2.

Alternately, the gradient-ascent approach in (Roth & Black, 2005) is dominated by the time taken to compute the inner products in (the derivative of the logarithm of) equation (3). Each pixel is contained by four cliques, and we must compute the inner product against each of our three filters. Therefore we must compute a total of $4 \times 3 \times 5829 = 69948$ inner products per iteration.

[3]Other operations, such as additions and permutations are typically much faster than these.

Table 2. Number of operations required by our algorithm. Multiplications are of $(n \times n) \times (n \times 1)$ matrices, and inverses are of $(n \times n)$ matrices, for various values of n.

n	Multiplications	Inverses
1	14800	44648
2	42072	37386
3	25760	12880
4	43308	21654

As a simple experiment, we timed these operations in Matlab (using random matrices and vectors). We found that computing 69948 inner products was approximately 10 times faster than computing the matrix operations shown in table 2. This leads us to believe that a low-level implementation of our belief-propagation algorithms may be significantly faster even than the results we have shown.

6. Discussion

Our results have shown than even a 2×2 model is able to produce very satisfactory inpainting performance. We believe that even this small model is able to capture much of the important information about natural images. While higher-order models exist (Roth & Black, 2005), the improvements appear to be quite incremental, despite a significant increase in their execution time. While it is certainly the case that our results fall short of the state-of-the-art in terms of PSNR, the differences are difficult to distinguish visually. It is therefore pleasing that we are able to produce competitive results within only a short period of time.

We have not yet fully explored the possibility of using the junction-tree algorithm to inpaint images. Unfortunately, determining whether a graph obeys the junction-tree property (see section 4.1) is very expensive, meaning we simply used loopy belief-propagation in all cases, without even performing this test. However, there are many cases in which we can be *sure* that a junction-tree exists – for example, if the inpainting region is a scratch which is only one or two pixels wide. In such cases, optimal results can be produced after only a single iteration, which would render our algorithm several times faster again.

In spite of this, we found that loopy belief-propagation tended to converge in very few iterations. While we believe it helped that the regions we are inpainting appear to be fairly 'tree-like', there is very little theory to support this claim. On the other hand, loopy belief-propagation often converges far slower when dealing

with large regions, meaning that we can inpaint a 'scratch' much faster than a 'coffee stain'.

We have also not considered the possibility that the corrupted pixels may contain some information about the original image. Many gradient-ascent approaches implicitly exploit this possibility by initializing their algorithms using the corrupted pixels. If the restored image is 'close to' the corrupted image, this can result in faster convergence. Our approach is also able to deal with this possibility by augmenting the graphical model with an observation layer with the respective noise model for the damaged pixels.

7. Conclusion

In this paper, we have developed a model for inpainting images quickly using belief-propagation. While image inpainting has previously been performed using low-order models by belief-propagation, and high-order models by gradient-ascent, we have presented new methods which manage to exploit the benefits of both, while avoiding their shortcomings. We have shown these algorithms to give satisfactory visual results and to be faster than existing gradient-based techniques, even in spite of our high-level implementation.

References

Ahrendt, P. (2005). The multivariate gaussian probability distribution.

Aji, S. M., & Mceliece, R. J. (2001). The generalized distributive law and free energy minimization. *Proceedings of the 39th Allerton Conference*.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Burton, G. J., & Moorhead, I. R. (1987). Color and spatial structure in natural scenes. *Applied Optics*, *26*, 157 – 170.

Carreira-Perpiñán, M. A. (2000). Mode-finding for mixtures of Gaussian distributions. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (pp. 1318–1323).

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, *39*, 1–38.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Machine Intell.*, *6*, 721–741.

Gosset, W. S. (1942). The probable error of a mean. *Student's Collected Papers* (pp. 11–34).

Hinton, G. E. (1999). Products of experts. *Ninth International Conference on Artificial Neural Networks, ICANN* (pp. 1–6).

Kschischang, Frey, & Loeliger (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, *47*, 498–519.

Lan, X., Roth, S., Huttenlocher, D. P., & Black, M. J. (2006). Efficient belief propagation with learned higher-order Markov random fields. *ECCV (2)* (pp. 269–282).

MacQueen, J. B. (1967). Some methods of classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathemtical Statistics and Probability* (pp. 281–297).

Martin, D., Fowlkes, C., Tal, D., & Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *Proc. 8th Int'l Conf. Computer Vision* (pp. 416–423).

McAuley, J. J., Caetano, T. S., Smola, A. J., & Franz, M. O. (2006). Learning high-order MRF priors of color images. *ICML '06: Proceedings of the 23rd international conference on Machine learning* (pp. 617–624). New York, NY, USA: ACM Press.

Roth, S., & Black, M. J. (2005). Fields of experts: A framework for learning image priors. *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 860–867).

Rue, H., & Held, L. (2005). *Gaussian Markov random fields: Theory and applications*, vol. 104 of *Monographs on Statistics and Applied Probability*. London: Chapman & Hall.

Sudderth, E., Ihler, A., Freeman, W., & Willsky, A. (2003). Nonparametric belief propagation. *Proceedings of the 2003 IEEE Conference on Computer Vision and Pattern Recognition*.

Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, *13*, 600–612.

Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2000). Generalized belief propagation. *NIPS* (pp. 689–695).

Bibliography

[Ahr05] Peter Ahrendt. The multivariate gaussian probability distribution, 2005.

[AM00] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. In *IEEE Transactions on Information Theory*, volume 46, pages 325–343, 2000.

[AM01] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law and free energy minimization. In *Proceedings of the 39th Allerton Conference*, 2001.

[Bes74] Julian Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society*, 36(2), 1974.

[Bil98] Jeff A. Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for gaussian mixture and hidden markov models, 1998.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, August 2006.

[BM87] G. J. Burton and I. R. Moorhead. Color and spatial structure in natural scenes. *Applied Optics*, 26:157 – 170, 1987.

[Bor04] Sean Borman. The expectation maximization algorithm, 2004.

[BSCB00] Marcelo Bertalmio, Guillermo Sapiro, Vicent Caselles, and Coloma Ballester. Image inpainting. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 417–424. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[CG92] G. Casella and E. I. George. Explaining the Gibbs sampler. *American Statistician*, 46:167–174, 1992.

[CPn00] Miguel Á. Carreira-Perpiñán. Mode-finding for mixtures of Gaussian distributions. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 22, pages 1318–1323, 2000.

[CPnH05] Miguel Á. Carreira-Perpiñán and Geoffrey E. Hinton. On contrastive divergence learning. In *10th Int. Workshop on Artificial Intelligence and Statistics (AISTATS'2005)*, 2005.

[Del02] Frank Dellaert. The expectation maximization algorithm, 2002.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[FPC00] William T. Freeman, Egon C. Pasztor, and Owen T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

[GG84] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Machine Intell.*, 6(6):721–741, Nov. 1984.

[Gos42] Willeam Sealy Gosset. The probable error of a mean. In *Student's Collected Papers*, pages 11–34, 1942.

[Har58] H. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14:174–194, 1958.

[HC71] J. M. Hammersley and P. Clifford. Markov field on finite graphs and lattices. unpublished manuscript, 1971.

[Hin99] G. E. Hinton. Products of experts. In *Ninth International Conference on Artificial Neural Networks, ICANN*, volume 1, pages 1–6, 1999.

[Hin01] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence, 2001.

[IFW04] A. T. Ihler, J. W. Fisher III, and A. S. Willsky. Message errors in belief propagation. In *Neural Information Processing Systems*, 2004.

[Jor] Michael I. Jordan. An introduction to probabilistic graphical models. In preparation.

[KF] D. Koller and N. Friedman. Structured probabilistic models. In preparation.

[LRHB06] Xiangyang Lan, Stefan Roth, Daniel P. Huttenlocher, and Michael J. Black. Efficient belief propagation with learned higher-order Markov random fields. In *ECCV (2)*, pages 269–282, 2006.

[MA93] Patricia McKenzie and Mike Alder. The EM algorithm used for gaussian mixture modelling and its initialization, 21, 1993.

[Mar06] John Marden. Multivariate statistical analysis (course notes), 2006.

[MCSF06] Julian J. McAuley, Tibério S. Caetano, Alex J. Smola, and Matthias O. Franz. Learning high-order MRF priors of color images. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 617–624, New York, NY, USA, 2006. ACM Press.

[MFTM01] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[Min98] T. Minka. Expectation-maximization as lower bound maximization, 1998.

[MK97] G.J. McLachlan and T. Krishnan. *The EM algorithm and extensions*. Wiley, New York, 1997.

[OF97] B. Olshausen and D. Field. Sparse coding with an overcomplete basis set: A strategy employed by V, 1997.

[RB05] Stefan Roth and Michael J. Black. Fields of experts: A framework for learning image priors. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 860–867, June 2005.

[RH05] H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*, volume 104 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London, 2005.

[RKC05] Antonio Robles-Kelly and Tiberio Caetano. Graph-based methods in computer vision and pattern recognition, 2005.

[SIFW03] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. Nonparametric belief propagation. In *Proceedings of the 2003 IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[Smi02] Lindsay I. Smith. A tutorial on principal components analysis, 2002.

[The94] The ITU Radiocommunication Assembly. ITU-R BT.601, 1994.

[WBSS04] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[WHO02] Max Welling, Geoffrey E. Hinton, and Simon Osindero. Learning sparse topographic representations with products of student-t distributions. In *NIPS*, pages 1359–1366, 2002.

[YFW00] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In *NIPS*, pages 689–695, 2000.