Compositional Models and Complexity of Vision
Confidential Draft: Not to be circulated

A.L. Yuille and R.Mottaghi

Abstract

This paper describes serial and parallel compositional models of multiple objects with part sharing. Objects are built by part-subpart compositions and expressed in terms of a hierarchical dictionary of object parts. These parts are represented on lattices of decreasing sizes which yield an executive summary description. We describe inference and learning algorithms for these models. Conceptually this theory is like a hierarchically distributed content addressable memory. This may serve as a "toy model" of the visual cortex since it enables both bottom-up and top-down processing. We analyze the complexity of this model in terms of computation time (for serial computers) and numbers of nodes (e.g., "neurons") for parallel computers. In particular, we compute the complexity gains by part sharing and its dependence on how the dictionary scales with the level of the hierarchy. We explore three regimes of scaling behavior where the dictionary size (i) increases exponentially with the level, (ii) decreases exponentially with scale, (iii) is determined by an unsupervised compositional learning algorithm applied to real data. This analysis shows that in some regimes the use of shared parts enables algorithms which can perform inference in time linear in the number of levels for an exponential number of objects. In other regimes part sharing has little advantage for serial computers but can give linear processing on parallel computers.

I. Introduction

Comment: need a figure which gives the big picture and introduces the players – the main concepts and ideas. Main ideas – compositional models built recursively from part-subpart compositions, dictionaries of shared parts,executive summary (abstraction) and hierarchical distributed representation, models are generative but enable rapid bottom-up inference. Inference and learning by log-likelihood tests which compare the local evidence for parts to default background models. Local evidence – purely bottom-up – will be supplemented by top-down context. Serial and parallel implementation. Neuronal plausibility. Complexity results.

A fundamental problem of vision is how to deal with the enormous complexity of images and visual scenes. The total number of possible images is almost infinitely large (Kersten 1987). The number of visual scenes and objects is also enormous. In light of this complexity, how can vision systems, artificial or biological, be designed so that that they can rapidly infer the visual scene from the input image?

A promising way to address this complexity is by making the assumption of compositionality. This assumes that objects and images can be built by combining elementary components to form composite parts in a hierarchical manner. The key advantage of compositionality is that it enables part-sharing by treating parts as modular components. This leads to more efficient representations, inference algorithms, and learning algorithms. Intuitively, many objects can be represented by compositions of the same elementary components (imagine the range of objects you can build using a lego kit). Inference is also more efficient because a part only needs to be detected once even if it is used for multiple objects (eg., a leg detector can be used for detecting the legs of cows, horses, and yaks). Finally, learning is also more efficient because parts learnt from one object can be used to rapidly learn other objects (eg., it is much faster to learn a yak if you have already learnt a cow).

Biological visual system have found ways to deal with this complexity. In particular, humans can obtain a rough interpretation of an image in less than one hundred and fifty milliseconds. In addition, humans can also learn novel objects extremely rapidly if they are similar to ones already seen. But it is less clear how the human visual system performs inference and learning. There is rough understanding of the architecture of the human visual system and some general principles have been discovered. It has a hierarchical structure where neurons at the lowest levels are responsive to simple spatially localized

stimuli, while neurons at the higher levels respond to more complex stimuli but care less about the spatial location. There is general consensus that the rapid ability of humans to interpret images is performed by bottom-up/feedforward processing up the hierarchy. But there is growing evidence of top-down processing, particularly when visual attention is involved.

The purpose of this paper is to develop a computational model which can be used to investigate these issues by mathematical analysis and computer simulations to quantify the advantages of compositionality for representation, inference, and learning. The model is generative, in the sense that we can sample from it to obtain an image, but we can also perform inference rapidly on it by bottom-up/feedforward processing. In addition, we derive a parallelized version of this model which we show has several properties of the human visual cortex (these are derived from the model rather than being imposed by hand). The models we study are restrictive, which seems inevitable if we want to perform mathematical analysis, but nevertheless they have been applied to real images and we have obtained good results on challenging computer vision tasks [**?**],[**?**].

A useful metaphor for this computational model is a hierarchical organization where each layer is related only to the level directly below and above it. Within this organization knowledge is represented hierarchically. The bosses, at the top node, have high level executive summary descriptions of the image (e.g., a cow in a field of grass) while lower level executives represent the positions of the cow's legs, regions in the image where the grass is long, and so on. Employees at the lowest level encode the positions of the edges in the images and other low level image properties. In this metaphor, inference is performed by propagating hypotheses up the hierarchy. Images are typically highly ambiguous locally so the low-level employees, who only have access to part of the image, cannot give unambiguous interpretations of the image (e.g., decide definitely if there is an edge or not) so instead they propagate hypotheses (i.e. possible interpretations) to the higher levels which access larger parts of the image, hence have greater context, and also encode higher-level knowledge. Hence hypotheses are resolved as we proceed up the hierarchy and the high-level interpretation is unambiguous. These high-level interpretations are propagated down the hierarchy and resolve the low-level ambiguities.

The background section (II) describes related material in the literature. The compositional section (III) introduces the key ideas. Section (IV) describes the inference algorithms for serial and parallel implementations. Section (V) performs a complexity analysis and shows potential exponential gains by using compositional models. Section (VI) summarizes the learning algorithm for the hierarchical dictionaries. Section (IX) describes the possible relationship to neuroscience and related issues.

II. Background

This work builds on the idea of compositionality [**?**], where objects are built by composing elementary components, and relates to probabilistic grammars of patterns [**?**]. The advantage of this approach is that it enables us to address the complexity problem of vision by representing an extremely large number of objects in terms of a limited number of components, while also enabling efficient inference and learning. The ideas are roughly consistent with some theories of human perception [**?**] and [**?**] (note: wrong Ullman reference). Some practical models for real world images have been built using these ideas, see [**?**], [**?**] [**?**] JMIV review!! Methods for learning these models are described in [**?**],[**?**], and [**?**] which discusses the relations to minimum length encoding [**?**].

Refer to Le Cun, Lee and Mumford, Thorpe, Poggio, others!! Murray et al. Also mention Valiant's work. Also Mottaghi and Yuille (ICCV Workshop 2011). Also Lennie's review. Di Carlo's review. Pearl's modularity. Zhu and Mumford. Also Dominguez, Williams, Fergus.

Also Yuille on when high level models help.Researchers in artificial intelligence have exploited redundancies, like part sharing, in graphical models to improve inference – see Darwiche and Choi).

Kersten paper on the number of possible images. (Kersten 1987).

Barlow paper on suspicious coincidences (Vision Research). Morel on meaningful alignments.

Mel's bart-lets? Check with him.

Robustness issue – e.g. 2 out of 3 rule.

Discussion of the need for scaling, rotations, and other transformations. Say that Leo's models did have these properties. Mention why they are useful – e.g., transfer learning and the need to deal with complex objects.

Mention the "where does high-level help" study by Yuille at al. Compositional Models, in particular the learning, require that we can get "partial credit" by using a model that is simpler than the correct model (i.e. the model that generated the data). Formally the performance can be characterized by the Bayes risk. Yuille et al studied the task of detecting roads from aerial images by analyzing a model proposed by Geman and Jedynak. Yuille et al showed that under some circumstances almost optimal performance could be obtained using a simple model (e.g., a model that used only first order relations between parts of the road which ignores the second order relations).

Background model. As an example, use derivative filter responses as inputs. Then we can use the standard model for the statistics of derivatives as the default model. Refer to this model here. Give a figure to describe this at the right part of the paper.

III. The Compositional Models

Compositional models are based on the idea that objects are built by compositions of parts which, in turn, are compositions of more elementary parts. These are built by part-subpart compositions.

A. Compositional Part-Subparts

We formulate part-subpart compositions by probabilistic graphical model which specifies how a part is composed of its subparts. The parent node ν of the graph represents the part by its type τ_ν and a state variable x_ν (e.g., x_ν could indicate the position of the part). The r child nodes $Ch(\nu) = (\nu_1, ..., \nu_r)$ represent the parts by their types $\tau_{\nu_1}, ..., \tau_{\nu_r}$ and state variables $\vec{x}_{Ch(\nu)} = (x_{\nu_1}, ..., x_{\nu_r})$. The type of the parent node is specified by $\tau_\nu = (\tau_{\nu_1}, ..., \tau_{\nu_r}, \lambda_\nu)$. Here $(\tau_{\nu_1}, ..., \tau_{\nu_r})$ are the types of the child nodes, and here λ_ν specifies a distribution over the states of the subparts (e.g.,over their relative spatial positions). Hence the type τ_ν of the parent specifies the part-subpart compositional model.

The probability distribution for the part-subpart model relates the states of the part and the subparts by:

$$P(\vec{x}_{Ch(\nu)}|x_\nu; \tau_\nu) = \delta(x_\nu - f(\vec{x}_{Ch(\nu)}))h(\vec{x}_{Ch(\nu)}; \lambda_\nu). \tag{1}$$

Here $f(.)$ is a deterministic function, so the state of the parent node is determined uniquely by the state of the child nodes ("child-parent determinism"). The function $h(.)$ specifies a distribution on the relative states of the child nodes.

The state variables x_ν of the parent are intended to provide an *executive summary* description of the part and hence are restricted to take a smaller set of values than the state variables x_{ν_i} of the subparts. Intuitively, the state x_ν of the parent offers summary information (e.g., there is a cow in a field) while the child states $\vec{x}_{Ch(\nu)}$ offer more detailed information (e.g., the position of the cow in the field, and the position of the parts of the cow). Hence information is represented at all levels of the graph.

We require the distribution $P(\vec{x}_{Ch(\nu)}|x_\nu; \tau_\nu)$ to obey a *locality principle*, so that $P(\vec{x}_{Ch(\nu)}|x_\nu; \tau_\nu) = 0$, unless $|x_{\nu_i} - x_\nu|$ is smaller than a threshold for all $i = 1, .., r$. A requirement of this form is fairly natural (subparts of a part are usually close together) and we will specify it detail later in this section.

Dan asks "why is locality a hard constraint". Answer is that it enables the analysis but relaxing it shouldn't cause too many problems.

B. T's and L's as examples of part-subpart compositions

We show examples of part-subpart compositions in figure (1). Here the compositions represent the letters T and L. The types of the child nodes are horizontal and vertical bars, indicated by $\tau_1 = H,\ \tau_2 = V$. The child state variables x_1, x_2 indicate the positions of the horizontal and vertical bars in the image.

The state variable x of the parent node gives a summary description of the position of the letters T and L. The compositional models for letters T and L differ by their λ parameter which species the relative positions of the horizontal and vertical bars. In this example, we choose $h(.;\lambda)$ to a Gaussian distribution, so $\lambda = (\mu, \sigma)$ where μ is the mean relative positions between the bars and σ is the covariance. We set $f(x_1, x_2) = (1/2)(x_1 + x_2)$, so the state of the parent node specifies the average positions of the child nodes (i.e. the positions of the two bars). Hence the two compositional models for the T and L have types $\tau_T = (H, V, \lambda_T)$ and $\tau_L = (H, V, \lambda_L)$.

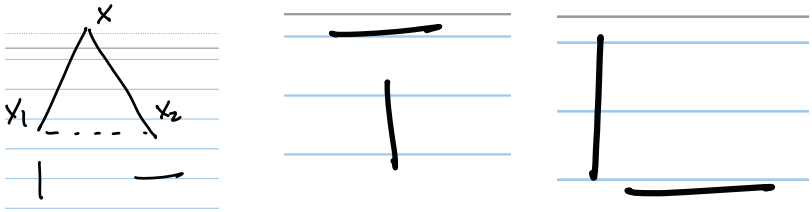

Fig. 1. Compositional part-subpart models for T and L are constructed from the same elementary components τ_1, τ_2, horizontal and vertical bar using different spatial relations $\lambda = (\mu, \sigma)$, which impose *locality*. The state x of the parent node gives the summary position of the object, the *executive summary*, while the positions x_1, x_2 of the components give details about its components.

C. Building Object Models by Compositions of Part-Subparts

We can build models of objects by combining part-subparts compositions in at least two ways. One is a bottom-up strategy, illustrated in figure (2), which builds bigger parts by compositions of subparts. This leads naturally to a compositional learning algorithm, see section (VI). Another is a top-down strategy which is similar to probabilistic grammar models [**?**] where an object is generated by applying a set of production rules recursively starting at a root node. But we will not investigate the top-down strategy in this paper.

The bottom-up strategy proceeds by introducing new part-subpart compositions, whose subparts are modeled, in turn, by part-subpart compositions (of more elementary subparts). This enables us to build a graphical model with an arbitrary number h of layers. Note than in our conventions, the type of the root node of the compositional model specifies (deterministically) the types of its child nodes and hence, by recursion, the type nodes of the entire graph. We can illustrate the bottom-up procedure by building a model which combines letters T's and L's. To do this we use new part-subpart models where the types of the subparts are τ_T and τ_L. These new models would differ by the choice of subparts (e.g., two T's, two L's, or one T and one L) and by the values of the λ parameters specifying the relative states of the subparts. This is illustrated in figure (2).

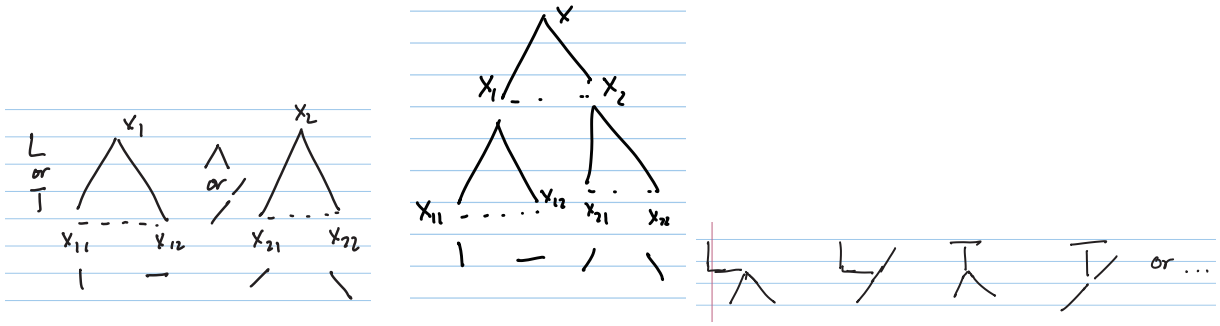

Fig. 2. Left Panel: Two part-subpart models. Center Panel: Combining two part-subpart models by composition to make a higher level model. Right Panel: Some examples of the shapes that can be generated by different parameters settings θ of the distribution.

D. Multiple Object Categories and Image Lattices

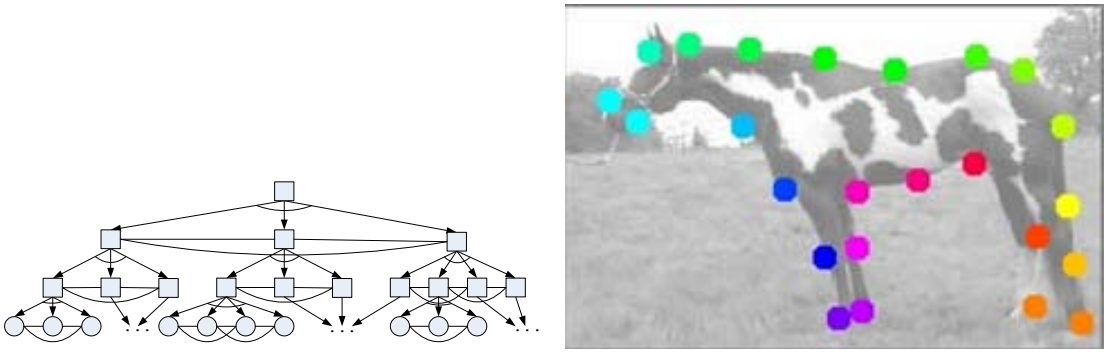

Fig. 3. Left: An example of a compositional model applied to a natural image. Square Nodes at different levels of the model represent parts and subparts of the object. The circle nodes are the leaf nodes which interact directly with the image. Right Panel: the leaf nodes are shown on the horse.

We now describe a model for multiple object categories where the state variables describe spatial position only (we will relax this condition later on). These state variables take values on a set of image lattices at different levels of resolution (see below). The probability models are built from a hierarchical dictionary of part-subpart compositions (see next subsection).

An object category is represented by a probability distribution defined over a graph $\mathcal{V}$. This graph has a hierarchical structure with levels $h \in \{0, ..., \mathcal{H}\}$, where $\mathcal{V} = \bigcup_{h=0}^{\mathcal{H}} \mathcal{V}_h$. Each object has a single, root node, at level-$\mathcal{H}$ (i.e. $\mathcal{V}_{\mathcal{H}}$ contains a single node). Any node $\nu \in \mathcal{V}_h$ (for $h > 0$) has r children nodes $Ch(\nu)$ in $\mathcal{V}_{h-1}$ indexed by $(\nu_1, ..., \nu_r)$. Hence there are $r^{\mathcal{H}-h}$ nodes at level-h (i.e. $|\mathcal{V}_h| = r^{\mathcal{H}-h}$). (In this paper, we restrict to the case where each parent has a fixed number of children r, but this restriction can easily be relaxed. At each node ν there is a state variable x_ν which indicates spatial position and type τ_ν. These position variables x_ν take values in a *set of lattices* $\{\mathcal{D}_h : h = 0, ..., \mathcal{H}\}$, so that a level-h node, $\nu \in \mathcal{V}_h$, takes position $x_\nu \in \mathcal{D}_h$. The leaf nodes $\mathcal{V}_0$ of the graph take values on the image lattice $\mathcal{D}_0$. The lattices are evenly spaced and the number of lattice points decreases by a factor of $q < 1$ for each level, so $|\mathcal{D}_h| = q^h |\mathcal{D}_0|$, see figure (4). This decrease in number of lattice points is a way to impose the *executive summary principle*. Intuitively, the lattice spacing is designed so that parts do not overlap. At higher levels of the hierarchy the parts cover larger regions of the image and so the lattice spacing must larger, and hence the number of lattice points smaller, to prevent overlapping. (Our pervious work [**?**],[**?**] was not formulated on lattices and used non-maximal suppression to achieve the same effect).

The probability model for an object category of type $\tau_{\mathcal{H}}$, see figure (3), can be expressed as a directed graphical model:

$$P(\vec{x}; \tau_{\mathcal{H}}) = \prod_{\nu \in \mathcal{V}/\mathcal{V}_0} P(\vec{x}_{Ch(\nu)} | x_\nu; \tau_\nu) U(x_{\mathcal{H}}). \tag{2}$$

State that $U(x_{\mathcal{H}})$ is the uniform distribution. Mention prior propagation and the intuition that if an object is equally likely to be anywhere in the image, then any subpart of the object is equally likely to be anywhere.

Here $x_{\mathcal{H}}$ is the state of the root node and $\tau_{\mathcal{H}}$ is its type. $\vec{x}$ denotes the states of all the nodes of this graph. The root node has r child nodes, whose types are specified by $\tau_{\mathcal{H}}$, and these child nodes all have r child nodes, and so on. $\vec{x}_{Ch(\nu)}$ denotes the positions of the child nodes $Ch(\nu)$ of ν, denoted by $\nu_1, ..., \nu_r$. The variable $\tau_\nu = (\tau_{\nu_1}, ..., \tau_{\nu_r}, \lambda_\nu)$ specifies the *type* of the distribution, which is determined by the types of the child nodes $\tau_{\nu_1}, ..., \tau_{\nu_r}$ and the spatial relations between them indicated by λ_ν (see the examples of T's and L's).

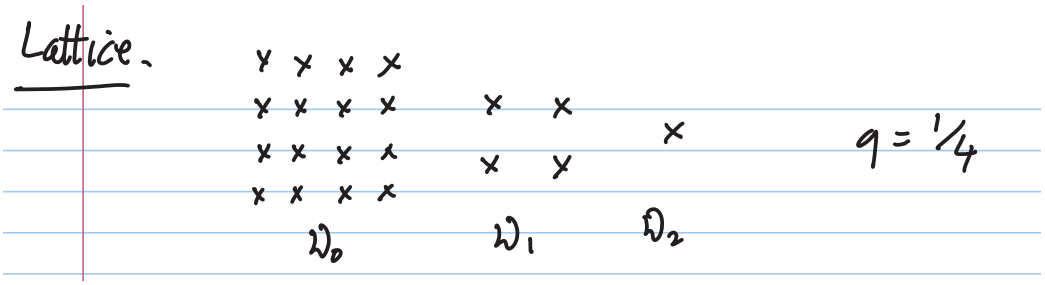

Fig. 4. The hierarchical l. The lattices decrease with scale which helps enforce executive summary and prevent having multiple hypotheses which overlap too much.

E. The Hierarchical Dictionaries

We now impose restrictions on the types of parts that can be represented in terms of part-subpart compositions. We require that they belong to a hierarchical dictionary $\{\mathcal{M}_h : h = 0, ..., \mathcal{H}\}$, where $\mathcal{M}_h$ is the dictionary of parts at level h. Hence the type variable τ_ν of an node at level h (i.e. in $\mathcal{V}_h$) indexes an element of the dictionary $\mathcal{M}_h$. The dictionary sizes $|\mathcal{M}_h|$ are finite. We impose this by restricting the size of $\mathcal{M}_0$ to be finite and by restricting the values of the parameters λ_ν to take a finite number of possible values. Object categories are expressed by distributions of the form in equation (2) with the restriction that *all parts of the objects are elements of the hierarchical dictionary.* Part sharing will occur when models for different object categories share elements of the dictionary.

An example of a hierarchical dictionary is shown in figure (5). It was obtained by a compositional learning algorithm [**?**] which is described in section (VI). Briefly, learning proceeds recursively by clustering in λ space (i.e. over the spatial relations) of instances of the lower-level dictionaries.

The hierarchical dictionaries are constructed recursively so that a dictionary element at level h is composed, by a part-subpart relation, to dictionary elements at level $h-1$. More precisely, the level-0 dictionary $\mathcal{M}_0$ consists of a set of basic elements indexed by types τ^0 (in this section, we use superscripts h to index the level in the hierarchy). In figure (5) there are six types elements corresponding to edges at different orientations. In our T and L example, the level-0 dictionary contains only horizontal and vertical bars.

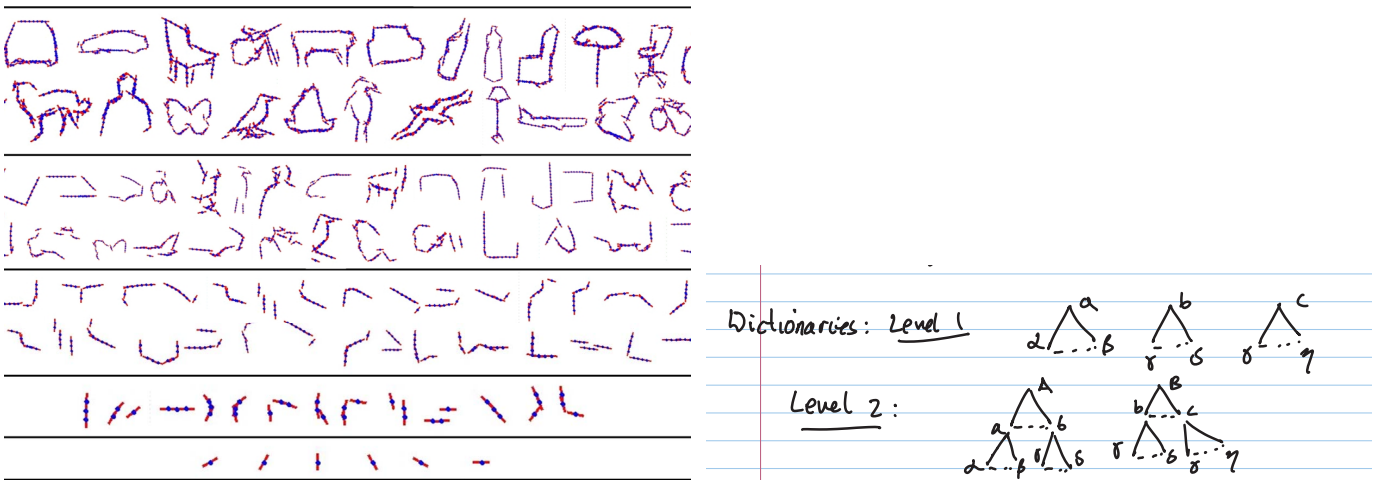

Fig. 5. Left Panel: Some of the part dictionaries learnt for 120 objects by an unsupervised compositional learning algorithm [**?**]. Only the mean shapes are shown. Right Panel: dictionary elements shown as graphical models.

The level-1 dictionary $\mathcal{M}_1$ contains compositions of r basic elements together with spatial relations specified by parameters λ. This specifies a class of level-1 models $\mathcal{M}_1$ indexed by $\tau^1 = (\tau^0_1, ..., \tau^0_r, \lambda)$

(e.g. the T's and L's in the examples). The level-h dictionary $\mathcal{M}_h$ is constructed by compositions of the level-(h-1) models. Hence $\tau^h = (\tau_1^{h-1}, ..., \tau_r^{h-1}, \lambda^h)$ where $\tau_1^{h-1}, ..., \tau_r^{h-1}$ indexes the types of the level-1 models and λ^{h-1} specifies their spatial relations. This specifies a class of level-h models $\mathcal{M}_h$ indexed by τ^h.

F. The Data Terms

We described, in equation (2), probability models $P(\vec{x}; \tau_{\mathcal{H}})$ for the state configurations different object categories $\tau_{\mathcal{H}} \in \mathcal{M}_{\mathcal{H}}$.

To extend this into a *generative model of an image* $\mathbf{I} = \{\mathbf{I}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_\mathbf{0}\}$ we specify a model for generating the intensity values $I(x)$ from the states and types at leaf nodes of the graph. The prior $P(\vec{x}; \tau_{\mathcal{H}})$, see equation (2), specifies a distribution on a set of points (the leaf nodes) $\mathcal{L} = \{x_\nu : \nu \in \mathcal{V}_0\}$ which lie on the image lattice (e.g., $x_\nu \in \mathcal{D}_0$) and determines their types $\{\tau_\nu : \nu \in \mathcal{V}_0\}$. We represent this as $\{(x, \tau(x)) : x \in \mathcal{L}\}$ where $\tau(x)$ is specified in the natural manner (i.e. if $x = x_\nu$ then $\tau(x) = \tau_\nu$). The intensity at an image pixel is specified by the type of the leaf node, or by a default *background model* $P(I(x)|\tau_0)$ if there is no leaf node at that pixel. This gives an *imaging model*:

$$P(\mathbf{I}|\vec{x}) = \prod_{x \in \mathcal{L}} P(I(x)|\tau(x)) \times \prod_{x \in \mathcal{D}_0/\mathcal{L}} P(I(x)|\tau_0), \tag{3}$$

Combining equations (2,3) gives us a generative model of an image and the states $\vec{x}$ of the nodes at all levels of the hierarchy. This model can be sampled from, conditioned on the state of $x_{\mathcal{H}}$, to generate an image. Note that the imaging model, equation (3), specifies that there is only one object in the image which is embedded in background. This can be extended in the natural manner to specify that there are several objects in the image, provided they do not overlap.

The *background model* $P(I(x)|\tau_0)$ can be extended to give a default model for the entire image, or a subregion, by assuming that all image values $I(x)$ are generated independently:

$$P_B(\mathbf{I}) = \prod_{x \in \mathcal{D}_0} P(I(x)|\tau_0). \tag{4}$$

This background model will be used later during inference and learning.

Discuss the standard background model for derivative filters.

IV. Inference by Dynamic Programming

We first describe the inference tasks that we want to perform and the quantities that we wish to estimate. Then we discuss how to compute these quantities using dynamic programming for a single object category (without part sharing) using a bottom-up and top-down pass. Next we describe how to perform inference for multiple objects more efficiently by part sharing which exploits the hierarchical dictionaries. Finally, using the hierarchical dictionaries, we demonstrate a parallel algorithm for detecting multiple objects.

A. Inference Tasks: State Detection and Model Selection

Consider a graphical model for a single object category defined by equations (2,3). This a directed graphical model with closed loops of maximum size $r + 1$ (part-subpart relations).

The *first inference task* is to find the most probable configuration of an object in the image (i.e. the position of the object and of its it parts). The *second task* is to determine whether there is an object in the image at all, which can be thought of as a model selection task. The *third task* is to detect several object configurations in the image, which requires deciding how many objects are present in the image (a model selection task) and estimating their configurations. Note that the first two tasks are special cases of the third.

All these tasks can be done using dynamic programming, as we will describe in the next subsection, and involve computing quantities which we call the *local evidence* for the presence of subparts at specific locations in the image. Intuitively, the local evidence ignores the context of the part (i.e. that is a subpart of an object) and instead compares how well the part model fits the data by comparison to the fit of the default background model. At the top $\mathcal{H}$ level the local evidence at $x_{\mathcal{H}}$ becomes the *evidence* for an object at that location.

We formulate the *first inference task* – detecting the best configuration of an object $\tau_{\mathcal{H}}$ – as Maximum a Posteriori (MAP) estimation. More precisely, we need to compute:

$$\vec{x}^* = \arg\max_{\vec{x}} \{\log P(\mathbf{I}|\vec{x}) + \log P(\vec{x}; \tau_{\mathcal{H}})\}. \tag{5}$$

This can be reformulated more insightfully as:

$$\vec{x}^* = \arg\max_{\vec{x}} \{\log \frac{P(\mathbf{I}|\vec{x})}{P_B(\mathbf{I})} + \log P(\vec{x}; \tau_{\mathcal{H}})\}$$

$$= \arg\max_{\vec{x}} \{\sum_{x \in \mathcal{L}} \log \frac{P(I(x)|\tau(x))}{P(I(x)|\tau_0)} + \sum_{\nu} \log P(\vec{x}_{Ch(\nu)}|x_\nu; \tau_\nu) + \log U(x_{\mathcal{H}})\}. \tag{6}$$

This reformulation is possible since $P_B(\mathbf{I})$ is independent of $\vec{x}$. Note that $\mathcal{L}$ is the set of the states of the leaf nodes of the graph and so depends on $\vec{x}$ (i.e. it must be determined during inference).

Equation (6) is more insightful than equation (5) for several reasons. Firstly, it shows that MAP estimation for a single object model consists of finding the configuration $\vec{x}^*$ which maximizes the log-likelihood ratio of the image by comparing the probability of the image using the object model $P(\mathbf{I}|\vec{x}^*)P(\vec{x}^*; \tau_{\mathcal{H}})$ to the probability of the image using the default background model $P_B(\mathbf{I})$. Secondly, it shows that this estimation can be performed by searching over all possible positions of the leaf nodes of the object model – i.e. over $\mathcal{L}$, see equation (6). Thirdly, after estimating $\vec{x}^*$ we obtain a log-likelihood ratio which we can use to *perform our second task* to determine whether the image contains an object or not using a model selection criterion:

$$\text{Object detected provided} \quad \max_{\vec{x}} \log\{\frac{P(\mathbf{I}|\vec{x})P(\vec{x}; \tau_{\mathcal{H}})}{P_B(\mathbf{I})}\} > T_{\mathcal{H}}. \tag{7}$$

Here $T_{\mathcal{H}}$ is a threshold which is chosen to allow for the fact that the object model has more parameters than the default model (this is similar to the AIC/BIC criteria – cite Ripley).

To perform the *third inference task* we generalize the imaging process to allow several instances of the object in the image with the requirement than they do not overlap. This non-overlap requirement can be enforced, exploiting our executive summary principle, by requiring that the different models have different values of $x_{\mathcal{H}}$ (or, in variants of the model, imposing the requirement that the values of $x_{\mathcal{H}}$ differ by more than a distance threshold). The generative model consists of first generating a random number n of objects, then generating the same number of positions $\{x_{\mathcal{H}_1}, ...x_{\mathcal{H}_n}\}$ for the root nodes of the objects (preventing them from overlapping in $\mathcal{M}_{\mathcal{H}}$), then generating a set of leaf nodes $\mathcal{L} = \bigcup_{i=1}^n \mathcal{L}_i$ by sampling from equation (2) conditioned on the $\{x_{\mathcal{H}_i}\}$. This produces a set $\{(x, \tau(x)) : x \in \mathcal{L}\}$ of types τ and positions x in the image lattice $\mathcal{D}_0$. We then apply the imaging model, equation (3), to generate the image.

The third inference task requires calculating the *evidence* $\phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$, the maximum log-likelihood ratios for each value of $x_{\mathcal{H}}$ for the models $\tau_{\mathcal{H}}$:

$$\phi(x_{\mathcal{H}}, \tau_{\mathcal{H}}) = \max_{\vec{x}/x_{\mathcal{H}}} \{\log \frac{P(\mathbf{I}|\vec{x})}{P_B(\mathbf{I})} + \log P(\vec{x}; \tau_{\mathcal{H}})\}, \tag{8}$$

where $\vec{x}/x_{\mathcal{H}}$ denotes the state of all nodes of the graphical model for $\tau_{\mathcal{H}}$ except the root node $x_{\mathcal{H}}$. We use model selection to remove those estimates whose $\phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$ values falls below threshold $T_{\mathcal{H}}$. This enables us to detect multiple instances of objects in an image, provided they do not overlap. This is an example of shared computation (we note that the first use of dynamic programming to detect objects in images

without initialization was also used to detect multiple instances of hands in an image – cite Coughlan and Yuille).

The nature of the compositional models means that computing the evidence $\phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$ can be performed efficiently by dynamic programming, as we will describe in the next section. This involves recursively computing quantities $\phi(x_h, \tau_h)$ which we will call the *local evidence* for object parts τ_h at all levels $h = 1, ..., \mathcal{H}$, where:

$$\phi(x_h, \tau_h) = \arg\max_{\vec{x}/x_h} \{\log \frac{P(\mathbf{I}|\vec{x})}{P_B(\mathbf{I})} + \log P(\vec{x}; \tau_h)\}. \tag{9}$$

The local evidence for the part τ_h ignores the higher level context of the object (i.e. the part-subpart relations at higher levels of the object model). Hence at low levels of the object model, the local evidence for parts may be weak reflecting the fact that low level vision is frequently ambiguous. The local evidence for parts becomes much stronger at higher levels of the object model because they can include more context (and the local evidence becomes the complete evidence at level $\mathcal{H}$). In addition, the local evidence is a concept that also occurs in the compositional learning algorithm described in section (VI).

B. Dynamic Programming for a Single Object

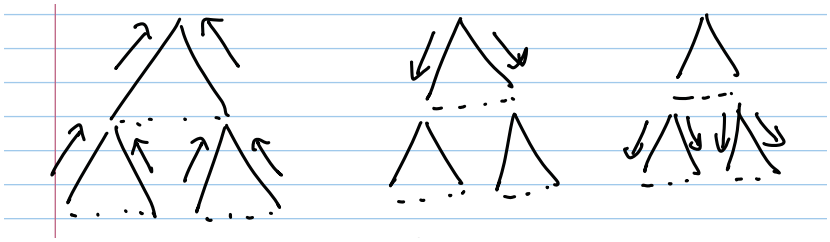

Fig. 6. Left Panel: The feedforward pass propagates hypotheses up to the highest level where the best state is selected. Center Panel: Feedback propagates information from the top node disambiguating the middle level nodes. Right Panel: Feedback from the middle level nodes propagates back to the input layer to resolve ambiguities there. This algorithm rapidly estimates the top-level executive summary description in a rapid feed-forward pass. The top-down pass is required to allow high-level context to eliminate false hypotheses at the lower levels– "high-level tells low-level to stop gossiping".

We define dynamic programming as follows (its complexity analysis is given in section (V)). It consists of a bottom-up pass followed by a top-down pass, see figure (6). The *bottom-up pass* initializes the states of the leaf nodes $\nu \in \mathcal{V}_0$ by assigning them *local evidence* $\phi(x_\nu, \tau_\nu) = \log \frac{P(I(x_\nu|\tau_\nu))}{P(I(x_\nu|\tau_0))}$ which give the local evidence for these states (e.g., this will be large for position x where the type τ is a vertical bar provided there is consistent with the image at that position. The bottom-up pass proceeds by recursively computing the local evidence, see equation (9), at all levels until we reach the top level. Note that the *local evidence* ignores the higher-level context, provided by the higher levels of the graphical model, which will be introduced by the top-down pass of dynamic programming.

We propagate the local evidence up the hierarchy recursively. Let ν be a node at level h ($\nu \in \mathcal{V}_h$) with type τ_ν and child nodes $Ch(\nu) = (\nu_1, ..., \nu_r)$ with types $(\tau_1, ..., \tau_r)$. These have *local evidence* $\phi(x_{\nu_1}, \tau_{\nu_1}), ..., \phi(x_{\nu_r}, \tau_{\nu_r})$ for positions $\vec{x}_{Ch(\nu)} = (x_{\nu_1}, ..., x_{\nu_r})$ (already computed). We compute the *local evidence* for the parent node ν, which has type τ_ν, by:

$$\phi(x_\nu, \tau_\nu) = \max_{\vec{x}_{Ch(\nu)}} \{\sum_{j=1}^{r} \phi(x_{\nu_j}, \tau_{\nu_j}) + \log P(\vec{x}_{Ch(\nu)}|x_\nu, \tau_\nu)\}. \tag{10}$$

This proceeds up the hierarchy until we have computed the evidence $\phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$ for the root node at level $\mathcal{H}$ (there are no higher levels of the graph and so "local evidence" now becomes "evidence"). Computing $\hat{x}_{\mathcal{H}} = \arg\max \phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$ for the root node gives the best position for the root node – the optimal summary state of the object.

The *top-down pass* recursively computes the optimal states of the lower level nodes. This uses the high level context to resolve the ambiguities in the local evidence. It starts with the solution $x^*_{\mathcal{H}} = \arg\max \phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$ at level-$\mathcal{H}$. It proceeds recursively to estimate the child states $\vec{x}_{Ch(\nu)}$ from the parent state x^*_ν by the update formula:

$$\vec{x}^*_{Ch(\nu)} = \arg\max_{\vec{x}_{Ch(\nu)}} \{\sum_{j=1}^{r} \phi(x_{\nu_i}, \tau_{\nu_i}) + \log P(\vec{x}_{Ch(\nu)} | x^*_\nu, \tau_\nu)\}. \tag{11}$$

We now make several observations. Firstly, the algorithm estimates the top-level "executive summary" description of the image first and only later, in the top-down pass, estimates the lower-level states by using the top-down context to resolve their ambiguities. Secondly, we can extend this algorithm to perform the second and third inference tasks in the natural way. We use the bottom-up pass to compute the evidence $\phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$ for all $x_{\mathcal{H}} \in \mathcal{D}_{\mathcal{H}}$ (for all possible executive summary positions of the object). Then we eliminate those $x_{\mathcal{H}}$ for which the evidence is below threshold, see equation (7). Finally we apply the top-down pass of dynamic programming starting at all nodes $x_{\mathcal{H}}$ where the evidence is above threshold.

We emphasize that all these computations are done without any approximations despite the fact that these models involve closed loops both in the part-subpart compositions and in the AND-OR structure of the models.

C. Inference with Part Sharing using the Hierarchical Dictionaries

Now suppose we want to detect instances of many object categories $\tau_{\mathcal{H}} \in \mathcal{M}_{\mathcal{H}}$ simultaneously. We can use exploit the shared parts by performing inference using the hierarchical dictionaries.

No we introduce a *fourth inference task* which generalizes our earlier three inference tasks. The *fourth inference task* is to determine which objects from $\mathcal{M}_{\mathcal{H}}$ are present in the image, how many instances there are of each object, and the state configuration of each detected object. This requires calculating the evidence $\phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$ for all objects $\tau_{\mathcal{H}} \in \mathcal{M}_{\mathcal{H}}$ and for all possible positions $x_{\mathcal{H}} \in \mathcal{D}_{\mathcal{H}}$. At each position $x_{\mathcal{H}}$, we compute the maximum of the evidence of all models $\max_{\tau_{\mathcal{H}} \in \mathcal{M}_{\mathcal{H}}} \phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$. If this evidence is above threshold $T_{\mathcal{H}}$, we detect the object $\tau^*_{\mathcal{H}}(x_{\mathcal{H}}) = \arg\max_{\tau_{\mathcal{H}} \in \mathcal{M}_{\mathcal{H}}} \phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$, and then estimate its state configuration by a top-down pass.

More formally:

$$\begin{aligned} \text{Let } \mathcal{D}^*_{\mathcal{H}} &= \{x_{\mathcal{H}} \in \mathcal{D}_{\mathcal{H}} \text{ s.t. } \max_{\tau_{\mathcal{H}} \in \mathcal{M}_{\mathcal{H}}} \phi(x_{\mathcal{H}}, \tau_{\mathcal{H}}) > T_{\mathcal{H}}\}, \\ \text{For } x_{\mathcal{H}} \in \mathcal{D}^*_{\mathcal{H}},\ &\text{let } \tau^*_{\mathcal{H}}(x_{\mathcal{H}}) = \arg\max_{\tau_{\mathcal{H}} \in \mathcal{M}_{\mathcal{H}}} \phi(x_{\mathcal{H}}, \tau_{\mathcal{H}}), \\ \text{Detect } \vec{x}^*/x_{\mathcal{H}} = \arg\max_{\vec{x}/x_{\mathcal{H}}} &\{\log \frac{P(\mathbf{I}|\vec{x})}{P_B(\mathbf{I})} + \log P(\vec{x}; \tau_{\mathcal{H}^*(x_{\mathcal{H}})})\} \text{ for all } x_{\mathcal{H}} \in \mathcal{D}^*_{\mathcal{H}}. \end{aligned} \tag{12}$$

Observe that all these calculations can be done efficiently using the hierarchical dictionaries (except for the $\max$ and $\arg\max$ tasks at level $\mathcal{H}$). Computing the $\phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$ for each $\tau_{\mathcal{H}}$ can be done by the bottom-up pass of dynamic programming, see equation (10), but because all objects are built from the hierarchical dictionaries we only need to compute the local evidences for the dictionary element (recall that a dictionary element at level-h is composed of r dictionary elements at level-(h-1)). This is intuitive, since if part-subpart relations are shared between different objects then it is only necessary to do inference on them once. This is illustrated in figure (7). Similarly, we can compute $\vec{x}^*/x_{\mathcal{H}}$ for any $x_{\mathcal{H}}$ and $\tau_{\mathcal{H}}$ using the top-down pass of dynamic programming, see equation (11), which can also be performed efficiently by performing computation directly on the hierarchy dictionary. The states of the other objects, and their parts, are set equal to zero (although other possibilities are possible, see discussion).

The only computation which cannot be performed by dynamic programming are the $\max$ and $\arg\max$ tasks at level H, see top line of equation (12). These are simple operations and require order $M_{\mathcal{H}} \times |\mathcal{D}_{\mathcal{H}}|$ calculations. This will usually be a small number, compared to the complexity of other computations. But this will become very large if there are a large number of objects, as we will discuss in section (V).

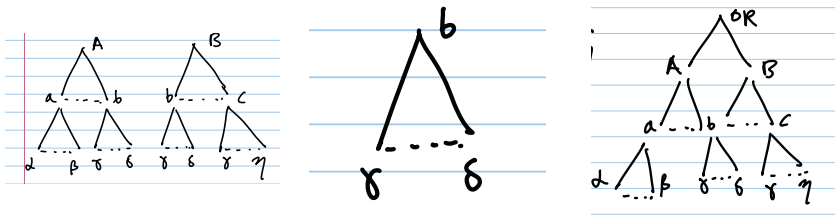

Fig. 7. Sharing. Left Panel: Two Level-2 models A and B which share Level-1 model b as a subpart. Center Panel: Level-1 model b. Inference computation only requires us to do inference over model b once, and then it can be used to computer the optimal states for models A and B. Right Panel: Note that is we combine models A and B by a root OR node then we obtain a graphical model for both objects. This model has a closed loop which would seem to make inference more challenging. But by exploiting the shape part we can do inference optimally despite the closed loop. Inference can be done on the dictionaries, far right.

There can be a great gain in speed of computation by doing inference on the dictionaries, provided a large number of parts are shared. This is quantified in section (V).

We emphasize that we are still doing exact inference, even though we are using the hierarchical dictionaries. This may seem slightly paradoxical because we can reformulate these probability models into a graph with closed loops, see figure (7). In other words, we are able to do exact inference on a graph with closed loops because of the specific form of the graph (i.e. because it can be expressed in terms of shared parts).

D. Parallel Formulation and Inference Algorithm

We now describe a parallel formulation of compositional models which could be suitable for implementation on certain types of computers (e.g., GPUs) and also serves as a "toy model" for the visual cortex of humans and monkeys.

Once again, we need to perform the computations of the fourth inference task, see equation (12). The parallel formulation exploits the fact that, in the bottom-up pass of dynamic programming, calculations are done separately for each position x, see equation (10). So we compute the local evidence for all parts in the hierarchical dictionary recursively in parallel for each position, and hence compute the $\phi(x_{\mathcal{H}}, \tau_{\mathcal{H}})$ for all $x_{\mathcal{H}} \in \mathcal{D}_{\mathcal{H}}$ and $\tau_{\mathcal{H}} \in \mathcal{M}_{\mathcal{H}}$. The $\max$ and $\arg\max$ operations at level $\mathcal{H}$ can also be done in parallel for each position $x_{\mathcal{H}} \in \mathcal{D}_{\mathcal{H}}$. Similarly we can perform the top-down pass of dynamic programming, see equation (11), in parallel to compute the best configurations of the detected objects.

The parallel formulation can be visualized by making copies of the elements of the hierarchical dictionary elements (the parts), so that a model at level-h has $|\mathcal{D}_h|$ copies, with one copy at each lattice point. Hence at level-h, we have m_h "receptive fields" at each lattice point in $\mathcal{D}_h$ with each one tuned to a different part $\tau_h \in \mathcal{M}_h$, see figure (8). At level-0, these receptive fields are tuned to specific image properties (e.g., horizontal or vertical bars).

Note that these "receptive fields" are highly non-linear in several respects. They do not obey the superposition principle and they are influenced by top-down context during the top-down pass. Nevertheless they are broadly speaking, tuned to image stimuli which have the mean shape of the corresponding part τ_h. In agreement, with findings about mammalian cortex, the receptive fields become more sensitive to image structure (e.g., from bars, to more complex shapes) at increasing levels. Moreover, their sensitivity to spatial position decreases because at higher levels the models only encode the executive summary descriptions, on coarser lattices, while the finer details of the object are represented more precisely at the lower levels.

We illustrate, in figure (9), the computations required for the the bottom-up and top-down pass between two adjacent levels of the hierarchy. The bottom-up pass is performed by a two-layer network where the first layer performs an AND operation (to compute the local evidence for a specific configuration of the child nodes) and the second layer performs an OR, or $\max$ operation, to determine the local evidence (by max-ing over the possible child configurations). The top-down pass only has to perform an $\arg\max$

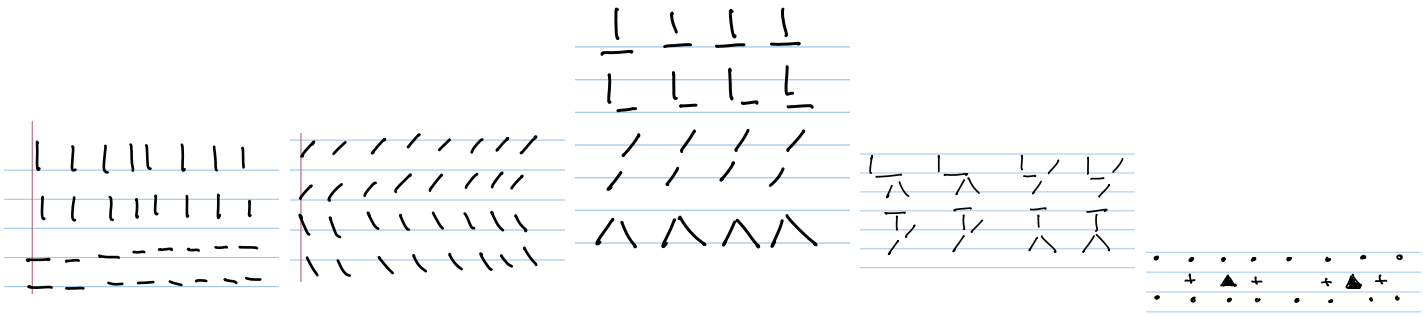

Fig. 8. Parallel Hierarchical Implementation. Far Left Panel: Two Level-0 models have spaced densely in the image (here an 8×2 grid). Left Panel: the other two Level-0 models are also densely sampled on the image. Center Panel: the four Level-1 models are sampled at a lower rate, and each have 4×1 copies. Right Panel: the four Level-2 models are sampled less frequently. Far Right Panel: a bird's eye view of the parallel hierarchy. The dots represent a "column" of four Level-0 models. The crosses represent columns containing four Level-1 models. The triangles represent a column of the Level-2 models.

computation to determine which child configuration gave the best local evidence. This two-layer model might be biologically plausible, as we discuss in a later section.

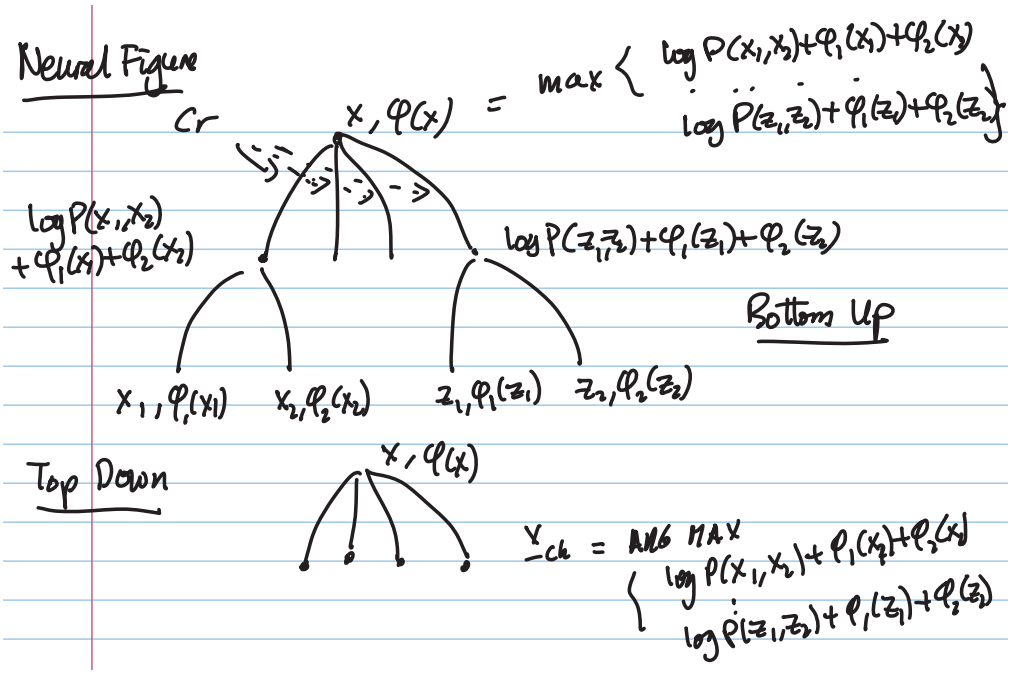

Fig. 9. Parallel implementation of Dynamic Programming. The upper figure shows the bottom-up pass of dynamic programming. The local evidence for the parent node is obtained by taking the maximum of the scores of the C_r possible states of the child nodes. This can be computed by a two-layer network where the first level computes the scores for all C_r child node states, which can be done in parallel, and the second level compute the maximum score. This is like an AND operation followed by an OR. The top-down pass requires the parent node to select which of the C_r child configurations gave the maximum score, and suppressing the other configurations.

V. Complexity Analysis

We now analyze the complexity of the inference algorithms for performing the tasks. Firstly, we analyze complexity for a single object (without part-sharing). Secondly, we study the complexity for multiple objects with shared parts. Thirdly, we consider the complexity of the parallel implementation.

Recall that $|\mathcal{D}_0|$ is the image size. There are $\mathcal{H}$ levels and the lattice sizes decrease by a factor q ($q < 1$) at each level, so $|\mathcal{D}_h| = |\mathcal{D}_0|q^h$. Each part has r subparts which are constrained to take C_r values.

A. Single Object

The basic computation of the bottom-up pass of dynamic programming requires computing:

$$\phi(x_\nu, \tau_\nu) = \max_{\vec{x}_{Ch(\nu)}} \{\sum_{i=1}^{r} \phi_i(x_{\nu_i}) + \log P(\vec{x}_{Ch(\nu)}|x_\nu, \tau_\nu)\}. \tag{13}$$

This requires a total of C_r computations for each position x_ν for each level-h node. There are $r^{\mathcal{H}-h}$ nodes at level h and each can take $|\mathcal{D}_0|q^h$ positions. This gives a total of $|\mathcal{D}_0|C_r q^h r^{\mathcal{H}-h}$ computations.

We sum over the levels to obtain the total number of computations for the bottom-up pass:

$$N_{single} = \sum_{h=1}^{\mathcal{H}} |\mathcal{D}_0|C_r r^{\mathcal{H}}(q/r)^h = |\mathcal{D}_0|C_r r^{\mathcal{H}} \sum_{h=1}^{\mathcal{H}} (q/r)^h = |\mathcal{D}_0|C_r \frac{qr^{\mathcal{H}-1}}{1-q/r}\{1-(q/r)^{\mathcal{H}}\}. \tag{14}$$

This uses $\sum_{h=1}^{\mathcal{H}} x^h = \frac{x(1-x^{\mathcal{H}})}{1-x}$. For large $\mathcal{H}$ we can approximate N_{single} by $|\mathcal{D}_0|C_r \frac{qr^{\mathcal{H}-1}}{1-q/r}$ (because $(q/r)^{\mathcal{H}}$ will be small).

Observe that the main contributions to N_{single} come from the first few levels of the hierarchy because the terms $(q/r)^h$ decrease rapidly with h.

The other computations involve taking the $\max, \arg\max$ at the top level and performing the top down pass. For a single object, the $\max$ and $\arg\max$ take $|\mathcal{M}_\mathcal{H}|$ computations and the top-down pass requires only $C_r\mathcal{H}$ calculations. Hence these computations are far smaller than those for the bottom-up pass. So we will neglect them.

The computations will scale linearly with the number of objects. Hence, the number of bottom-up computations for $|\mathcal{M}_\mathcal{H}|$ object categories is:

$$N_{single} \times M_\mathcal{H} = |\mathcal{D}_0|C_r M_\mathcal{H} \frac{qr^{\mathcal{H}-1}}{1-q/r}\{1-(q/r)^{\mathcal{H}}\}. \tag{15}$$

B. Computation with Shared Parts

See figure (10)

Now suppose we exploit the shared parts. This requires doing computations over the dictionaries only. At level h there are $|\mathcal{M}_h|$ dictionary elements. Each can take $|\mathcal{D}_h| = q^h|\mathcal{D}|$ possible states. For each part-subpart composition we require C_r computations.

The number of computations with shared parts is:

$$N_{shared} = \sum_{h=1}^{\mathcal{H}} |\mathcal{M}_h|C_r|\mathcal{D}_0|q^h = |\mathcal{D}_0|C_r \sum_{h=1}^{\mathcal{H}} |\mathcal{M}_h|q^h. \tag{16}$$

We now consider several different regimes which differ by how the size $|\mathcal{M}_h|$ scales with h.

One important regime is when $|\mathcal{M}_h|$ scales as $1/q^h$ where the size of the dictionaries grow exponentially with the number of layers. We can obtain the following result.

Result 1: we can perform dynamic programming with shared parts in time which is linear in $\mathcal{H}$ even if the number of objects increases by $|\mathcal{M}_h| \propto \frac{1}{q^h}$. Hence the number of objects can be exponential in $\mathcal{H}$ but the computations can be linear in $\mathcal{H}$, i.e. $a|\dashv|$. See figure (10)(left panel). By contrast, the computations grow exponentially with $\mathcal{H}$ if we do not use shared parts, i.e. $a\frac{1}{q^\mathcal{H}} \times N_{single}$. Note N_{single} also grows exponentially with $\mathcal{H}$, see equation (15). For parallel implementations, inference is linear in $\mathcal{H}$ while requiring only a number of nodes ("neurons") which is linear in $\mathcal{H}$.

The number of compositions can in principle grow very fast (i.e. exponentially). If there are $|\mathcal{M}_0|$ types at level 0, then we can have $|\mathcal{M}_0|^r$ types at level-1 if we assume only one spatial configuration for all possible combinations of types. This number increases if we allow some type combinations to have several possible spatial configurations (e.g., the T and L have the same type combinations but different spatial configurations). In principle, the size of $|\mathcal{M}_h|$ can grow exponentially with h. (Of course, these may not correspond to objects in the real world).

Now we consider other regimes. Compare the formulae $N_{shared} = KC_r \sum_{h=1}^{\mathcal{H}} q^h |\mathcal{M}_h|$ with $N_{single} \times |\mathcal{M}|_{\mathcal{H}} = KC_r \sum_{h=1}^{\mathcal{H}} |\mathcal{M}_{\mathcal{H}}| q^h r^{\mathcal{H}-h}$. If $|\mathcal{M}_h|$ increases more slowly than $1/q^h$, then the dominant terms occur at small values of h (the computations are the same at $h = \mathcal{H}$). So part sharing helps considerably provided $|\mathcal{M}_h| << |\mathcal{M}_{\mathcal{H}}| r^{\mathcal{H}-h}$. This gives us:

Result 2: If $|\mathcal{M}_h$ grows slower than $1/q^h$ and if $|\mathcal{M}_h| < r^{\mathcal{H}-h}$ then there are gains due to part sharing. This is illustrated in figure (10)(right panel) based on the dictionaries, see figure (**??**), found by unsupervised computational learning [**?**]. In parallel implementations, computer is linear in $\mathcal{H}$ while requiring a limited number of nodes (”neurons”).

There is another important regime the number of parts decreases rapidly as h increases. In this regime part sharing is problematic if performed serially, but may be practical if performed in parallel (see next section). Suppose the size $|\mathcal{M}_h$ increases rapidly as h becomes small and decreases for large h. For example, suppose $|\mathcal{M}_h| = |\mathcal{M}_{\mathcal{H}}| r^{\mathcal{H}-h}$. This situation could occur in real images if we use dictionaries to model the appearance of image patches. There are an extremely large number of different image patches due to the enormous variability of images (many more than the numbers of types of edges). Higher level dictionaries would model only the ”executive summary” of image appearance – e.g., cat fur, grass texture – and so could decrease rapidly. In this case, the shared parts will not help much if we have a serial implementation but, as we will see, a parallel implementation can do a lot better.

Result 3: If $|\mathcal{M}_h| = r^{\mathcal{H}-h}$ then there is no gain for part sharing if serial computers are used, see figure (10)(center panel). Parallel implementations can do inference in time which is linear in $\mathcal{H}$ but require an exponential number of nodes (”neurons”), see next subsection.

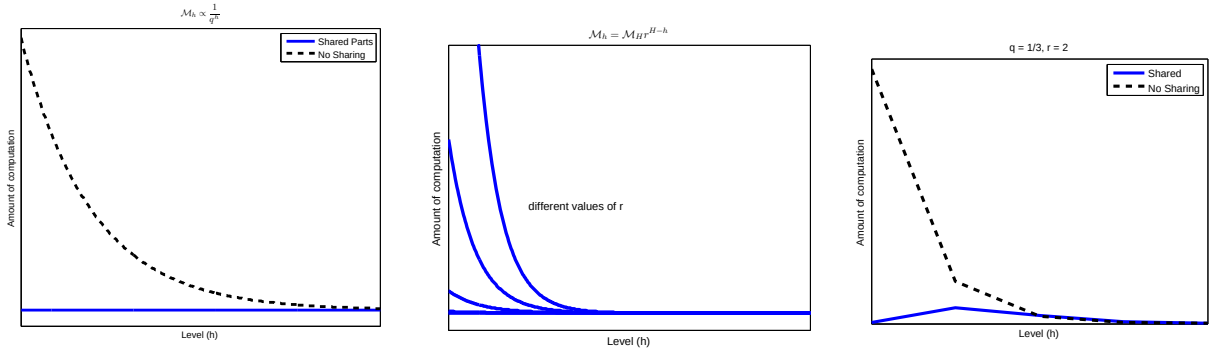

Fig. 10. The curves are plotted as a function of h. Left panel: The first plot is the case where $M_h = a/(q^h)$. So we have a constant cost for the computations, when we have shared parts. Center panel: The second plot is the case where M_h decreases exponentially. The amount of computation is the same for the shared and non-shared cases. A set of plots with different values of r. Right panel: The third plot is the Leo’s experiment.

C. Parallel Version

Now consider our parallel implementation. We can tradeoff between speed and the amount of other resources, in this case the number of ”neurons” which we use.

The number of ”neurons” is the total number of dictionary elements multiplied by the number of spatial copies of them:

$$N_{neurons} = \sum_{h=0}^{\mathcal{H}} |\mathcal{M}_h| q^h |\mathcal{D}_0|. \tag{17}$$

The computation, both the forward and backward passes of dynamic programming, are linear in the number H of levels. We simply have to perform the computations illustrated in figure (9) between all adjacent levels. These can be done in parallel.

Hence the parallel implementation gives speed which is linear in $\mathcal{H}$ at the cost of a possibly large number of "neurons" and connections between them.

There are several regimes to consider. First, the regime where $|\mathcal{M}_h|$ grows exponentially with h, but no faster than $\frac{1}{q^h}$. In this case the number of neurons is linear in $\mathcal{H}$ but the number of objects grows exponentially with h.

Result 4: In the parallel implementation we can detect an exponential number of objects $(1/q^h)$ with time linear in $\mathcal{H}$ and with a number of neurons linear in $\mathcal{H}$.NOTE: this result was stated earlier. Need to make the results consistent. Do not mention results for parallel implementations in previous section.

Another interesting regime if where the low level dictionaries are very big, perhaps representing the intensity in terms of a set of image patches, while the dictionary sizes decrease with h. This is the situation discussed at the end of the previous subsection where we concluded that part sharing does not give major improvements. But in the parallel implementation, we can perform linear inference in this case provided we have a very large number of "neurons" at the lower levels.

This may relate to one of the more surprising facts about the visual cortex in monkeys and humans. The first two visual areas, V1 and V2, are enormous compared to the higher levels such as IT where object detection takes places. As Lennie stated, when discussing V1 and V2 in his review of the visual cortex, perhaps the most troublesome objection to the picture I have delivered is that an enormous amount of cortex is used to achieve remarkably little.

From our perspective, one possibility is that visual areas V1 and V2 are encoding dictionaries of low level patches model local image appearance. According to our analysis, this still enables inference which is linear in H for large numbers of objects. This is highly speculative but, to our knowledge, there are few hypotheses about why V1 and V2 are so large. Of course, if this conjecture is correct, it has slightly worrying implications for implementing computer vision systems in parallel computers.

VI. Compositional Learning

The hierarchical dictionaries shown in figure (5) were learnt in an unsupervised manner by methods described in [**?**],[**?**]. We refer to those papers for details of the implementation. In this section we describe the theory for learning these dictionaries (which was not discussed in the original conference papers) and how they relate to the framework discussed in this paper.

We first give a procedural description of compositional learning and then give a theoretical justification. The procedure is to build the hierarchical dictionaries recursively. We first learn level-1 dictionaries which consist of part-subparts compositions where the subparts are elements of a pre-defined level-0 dictionary (e.g., horizontal and vertical bars). These level-1 dictionaries are learnt by clustering over the types of the level-0 elements and their spatial relations. Hence, if we input data in figure (11)(left panel) then we would obtain T and L as elements of the level-1 dictionary. Next we detect instances of these level-1 models in the image and perform clusters of them to get level-2 models. This process repeats and stops automatically at a level when we fail to find any clusters. The theory justifies this as a hierarchical breadth-first search through the space of generative models for the images. It is important that this search follows the principle of least commitment and keeps several, possibly conflicting, image interpretations at the lower levels of the hierarchy which are only resolved at higher levels. In particular, we do *not* require that the level-h models give a better encoding of the image than the level-(h-1) models.

The input to computational learning are a set of images. We assume a set $\mathcal{M}_0$ of level-0 models which is pre-specified (these could be learnt by a clustering procedure). We also assume that we can estimate the types of each image pixels in $\mathcal{D}_0$, for example by solving $\hat{\tau}(x) = \arg\max P(I(x)|\tau)$ for each position $x \in \mathcal{D}_0$ (e.g., by performing edge detection to find horizontal and vertical bars). We reject all image points which are assigned to the default type τ_0. For each image, we obtain a set of points with their corresponding types $\{(x_i, \tau(x_i))\}$, with $x_i \in \mathcal{D}_0$ and $\tau_i \in \mathcal{M}_0$.

To create the level-1 dictionaries we cluster sets of r points from $\{(x_i, \tau(x_i))\}$ which have fixed $\vec{\tau} = (\tau_1, ..., \tau_r)$ to find frequently occurring spatial relations (e.g., the spatial relations between the horizontal and vertical bars for the T and L). Hence we search for examples $\{(x_1^\mu, \tau_1), ..., (x_r^\mu, \tau_r) : \mu = 1, .., n\}$ and parameters λ such that:

$$\log \frac{P(\vec{x}^\mu, \vec{\hat{x}}^\mu, \vec{\tau}, \lambda))}{\prod_{i=1}^r P_D(x_i^\mu, \tau_i))} > K_1, \ \forall \mu \tag{18}$$

where $\hat{x}^\mu$ is the optimal estimate of the parent node – i.e. $\hat{x}^\mu = \arg\max_x P(\vec{x}^\mu, \vec{\tau} | \hat{x}^\mu, \vec{\tau}, \lambda) - P_D(x_i)$ is a default distribution for the positions of the points (e.g., the uniform distribution), and K_1 is a threshold.

We take the local maxima over the value λ to obtain a level-1 dictionary $\mathcal{M}_1$. Each dictionary element is indexed by its type $\tau^1 = (\vec{\tau}, \lambda^1)$, where $\vec{\tau}$ are the types of the r children, and λ_1 parameterizes the spatial relations. We not impose consistency so a pair (x, τ) can be used in many different clusters, see figure (11). This lack of consistency is desirable because we do not want to make premature decisions. It gives an over-complete representation of the image in terms of level-1 models. Note that equation (18) is similar to thresholding the local evidence for a part, with the main difference being the lack of the data terms $\log \frac{P(x|\tau(x))}{P(I(x)|\tau_0)}$ (which we neglect during learning for simplicity).

We repeat the process to learn the higher order dictionaries. More specifically, we apply the level-1 dictionaries in the image to find instances of each dictionary element where the local evidence is above threshold (e.g., we detect T's and L's). Each instance has a position $x \in \mathcal{D}_1$ and a type $\tau \in \mathcal{M}_1$. This gives a new set of data $\{(x_i, \tau(x_i)) : i = 1, ..., N_1\}$ with $x_i \in \mathcal{D}_1$ and $\tau_i \in \mathcal{M}_1$. We cluster again using equation (18). At this stage we do impose consistency between level-1 instances which are used in the same compositions. For example, in figure (11) we do *not* allow the T and L to form a composition because they share the same vertical bar. This outputs the Level-2 dictionary $\mathcal{M}_2$ and a set of instances of the Level-2 dictionary elements, which we use as input to learn the dictionary at the next level, and so on. The process terminates when we fail to find new clusters. Intuitively, this is because we have reached the largest size of structures that occur frequently in the images.

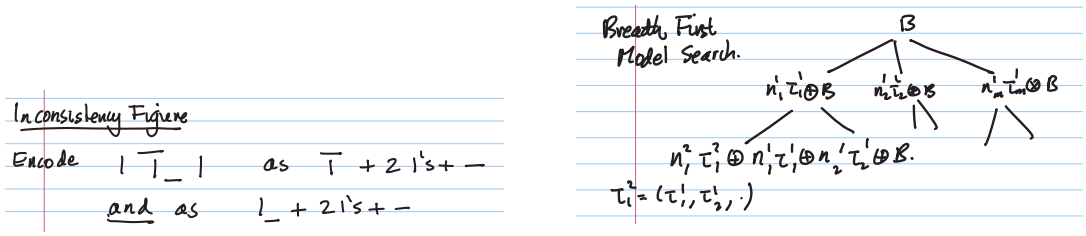

Fig. 11. Left Panel: Inconsistency of the Level-1 models. When learning the level-1 models in $\mathcal{M}_1$ we do not impose consistency between the models. Consistency is imposed as needed at the higher levels. It is premature, and unnecessary, to impose consistency at the low levels. It risk removing small structures, such as the ears of a cat, which occur only infrequently. Right Panel: This gives a breadth first search over the space of models.

There are certain trade-offs in this learning algorithm. If we make the thresholds for detecting new compositions too small (i.e. the K_1 and the threshold on the number of instances) then we will make very big dictionaries which require large memory space and/or computation, see section (V), and which may be due to noise in the data rather to any underlying structure. But if the thresholds are too small, then we risk missing image structures which are infrequent but significant (e.g., the ears of a cat are only a very small part of the cat silhouette but their characteristic shape is very helpful for detecting cats). Note, there are also some technical details which are described in the original papers (e.g., in practice, how spatial locality is used to constrain the search when clustering).

Now we justify the learning procedure theoretically. For each Level-1 model, we can make a generative model for the image which consists of sampling n instances of the Level-1 model – i.e. points and types $\{(x_i^\mu, \tau_i) : \mu = 1, ..., n\}$ – and sampling the rest of the image points from a default background model $P_D(x_i, \tau_i)$. Observe that the clustering condition for Level-1 models requires that equation (18) is satisfied

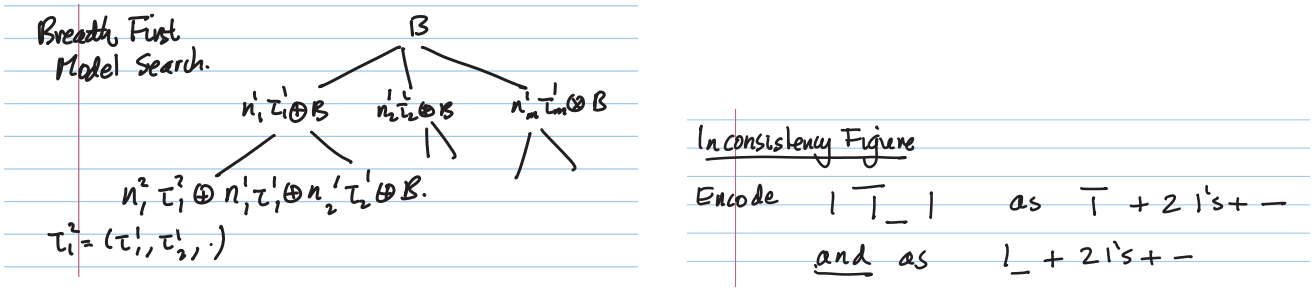

Fig. 12. Left Panel: The compositional learning algorithm searches for models in a parallel, breadth first manner. These models attempt to describe part of the data using a default background model to fill in the rest. Each model is required to give a better encoding of its data than its parent model. But, Right Panel, the models at the same level are not required to be consistent with each other. The stimulus is ambiguous and can be encoded in several different ways. We do not want to impose consistently at this level and, instead, prefer to keep multiple possible encodings. These inconsistencies will be resolved at higher levels – e.g., if we try to build a model that combines T's and L's.

for each model. This is precisely the condition that the generative model (i.e. several samples of the Level-1 model plus background) gives a better encoding of the image than assuming that it is generated only by the default background model:

$$\sum_{\mu=1}^{n} \log P(\vec{x}^{\mu}|\hat{x}^{\mu}, \tau) + \sum_{x \notin \{\vec{x}^{\mu}\}} \log P_D(x, \tau) > nK_1 + \sum_{x} \log P_D(x, \tau), \tag{19}$$

where K_1 is a penalty (e.g., AIC/BIC) because the default model P_D and the Level-1 model have different complexities.

In summary, every Level-1 model corresponds to a generative model for encoding the image. The condition in equation (18) ensures that all the Level-1 models are better at encoding the data than the default model.

Similarly every Level-2 model corresponds to a generative model for encoding the data. This generates the data in terms of a number of instances of the Level-2 model, some instances of each of its constituent Level-1 models (those which do not combine to form an instance of the Level-2 model) and some points generated by the background model. By construction, this model will give a better encoding of the data than the encoding of any of the constituent Level-1 models (i.e. in terms of instances of the Level-1 models plus background). Hence all Level-2 models correspond to better encoding of the data than their parent generative models, see figure (11).

This process repeats as we proceed to higher levels. We have a tree, see figure (11), of better encodings of the data. The encodings at different levels of the tree can be inconsistent and they give alternative ways to encode the data. The enables us to perform a "breadth-first search" through the space of models. There is no guarantee that this approach is optimal. But it does guarantee that we are finding increasingly better encodings of the data.

VII. Neural Plausibility?

If we consider our theory as a model for the visual cortex it is clearly too simplistic in many respects. Nevertheless it has some conceptual advantages compared to some current models. It is a generative model, which could be used to synthesize images or to perform attentional tasks, but it allows very rapid inference to get the top-level description. The relative advantages, and tradeoffs, between conventional feedforward models and top-down (analysis by synthesis) models are discussed in DiCarlo et al. But our "toy theory" presented in this paper has many of the advantages of both. It may also relate to Lennie's metaphor of the visual cortex as a sophisticated content addressable memory where the input on the retina activates a succession of higher level representations to ultimately activate a high level memory (and hence interpret the image). We can certainly think of the hierarchical dictionaries as a type of hierarchical memory.

But our compositional model does differ from current neural models in some respects. In particular, it is hard to see how to implement the inference and learning algorithms using classical "artificial neural network models" such as integrate and fire, and standard learning algorithms such as hebbian rules and back propagation. We may also want "neurons" to have different states which allow for different behavior when doing bottom-up processing, top-down processing, sampling, and learning.

We note that recent models of pyramidal neurons (e.g., Mel and collaborators) suggest that these neurons may be more sophisticated than standard integrate and fire models. These recent models are based on detailed studies, and modeling, of the biophysics of these neurons combined with in vitro experiments. In general, it seems that two layer models of the neurons, where calculations can be performed on parts of the dendridic tree and then combined later. are better able to predict their response than single layer integrate and fire models. As discussed earlier, our inference algorithm can be implemented by a two layer model though ours are different from the two layer models investigated by Mel *et al.* (REF!!). There may also exist mechanisms which can perform the top-down pass required by dynamic programming (Mel, private communication).

In addition, Valiant has made a strong argument for computational models of neurons (called neuroids in his studies) that can perform the types of programs required by computers and which enable complexity issues to be studied. He has developed theoretical models where "neuroids" are similar to integrate and fire models but have a few extra properties which enables them to be programmable. In addition, he has developed algorithms for memorization which involves the represent conjunctions of inputs by having a class of un-assigned neuroids that can be activated to represent conjunctions. Interestingly, his algorithms for memorization might be generalizable to store compositions as required by our compositional learning algorithms (he works mainly with binary valued input and does not treat spatial relations – except possibly in his work with Feldman CHECK!!).

In short, it seems possible that neural models which combine Mel's research on pyramidal cells with Valiant's theoretical ideas, including the requirement than networks of neurons be "programmable", can be used to give implementations of our models which may be consistent with the biology of the brain.

A much more speculative issue relates to neural correlates of consciousness (Koch). A recent theory speculates that consciousness may arise in terms of the complexity of graphical models (Koch + Wisconsin reference). Perhaps the high-level "conscious parts" of the brain only needs access to the executive summaries from the lower-levels? Or, conversely, the need to tame complexity leads to a hierarchical organization which embodies the executive summary principle? But maybe these issues should be best left to professional philosophers (Chalmers).

VIII. Robustness of the Model

This section addresses ways to extend the model and make it more robust. There are three motivations. Firstly, to prevent the model from degrading badly if subparts of the object fail to get detected. Secondly, to describe how the model could be extended to allow for parts to have numbers of subparts or to be partially invariant to certain image transformations. Thirdly, in terms of neural implementations, to be insensitive to malfunction of neurons.

We note that implementations of these models [**?**],[**?**] included several extensions. Firstly, the part-subpart distributions $P(\vec{x}_{Ch(\nu)}|x_\nu\Theta,\tau_\nu)$ were made insensitive to rotations and expansions in the image plane. Secondly, the 2-out-of-3 rule (described below) made the theory robust to failure to detect parts or to malfunctioning neurons. Thirdly, the imaging terms could be extended so that the model generates image features (instead of the image values directly) and could include direct image input to higher levels. All these extensions were. successfully tested on natural images [**?**],[**?**].

The 2-out-of-3 rule is described as follows. In our implementations [**?**],[**?**] a part-subpart composition has three subparts. The 2-out-of-3 rule allowed the parent node to be activated if only two subparts were detected – or if the image response at the third part is inconsistent with the object (e.g., if the object is partially occluded). The intuition is that if two parts are detected then the spatial relations between the

parts, embedded in the term $h(\vec{x}; \lambda_\nu)$ of $P(\vec{x}_{Ch(\nu)}|x_\nu, \tau_\nu)$, is sufficient to predict the position or the third part. This can be used during inference while paying a penalty for not detecting the part. From another perspective, this is equivalent to having a mixture of different part-subparts compositions where the part could correspond to three subparts or two subparts. This can be extended to allow a larger range of possible subpart combinations thereby increasing the flexibility. The amount of computation time would not increase for a parallel model.

We can analyze the 2-out-of-3 rule to show its robustness to missing parts. Consider two-layer of part-subpart compositions with a parent part, three subparts, and nine sub-subparts. The 2-out-of-3 rule enables us to detect the part even in cases where only four of the nine sub-subparts are detected (provided that one subpart has no sub-subparts detected and the remaining two subparts have two sub-subparts detected each). To study this in more detail, let ρ be the probability of detecting a subpart and $f(\rho)$ be the probability of detecting the part. Using the 2-out-of-3 rule, we obtain the update rule:

$$\rho_{h+1} = f(\rho_h), \quad \text{with } f(\rho) = \rho^3 + 3\rho^2(1-\rho). \tag{20}$$

We can analyze the robustness of the rule by studying the behavior of the iterative map $\rho_{t+1} = f(\rho_t)$. It can be calculated that there are fixed points at $\rho = 0, 0.5, 1$. $f(\rho) > \rho$ for $0.5 < \rho < 1$ and $f(\rho) < \rho$ for $0 < \rho < 0.5$, see figure (13). Hence if the initial value of $\rho < 0.5$ (i.e. the detectors at the leaf nodes miss the object more than half the time) then the iterative map converges to zero and we cannot detect the object. Conversely, if $\rho > 0.5$ (the initial detectors miss parts less than half the time) then the iterative map converges to 1 and we always detect the object if there are a sufficient number of levels.

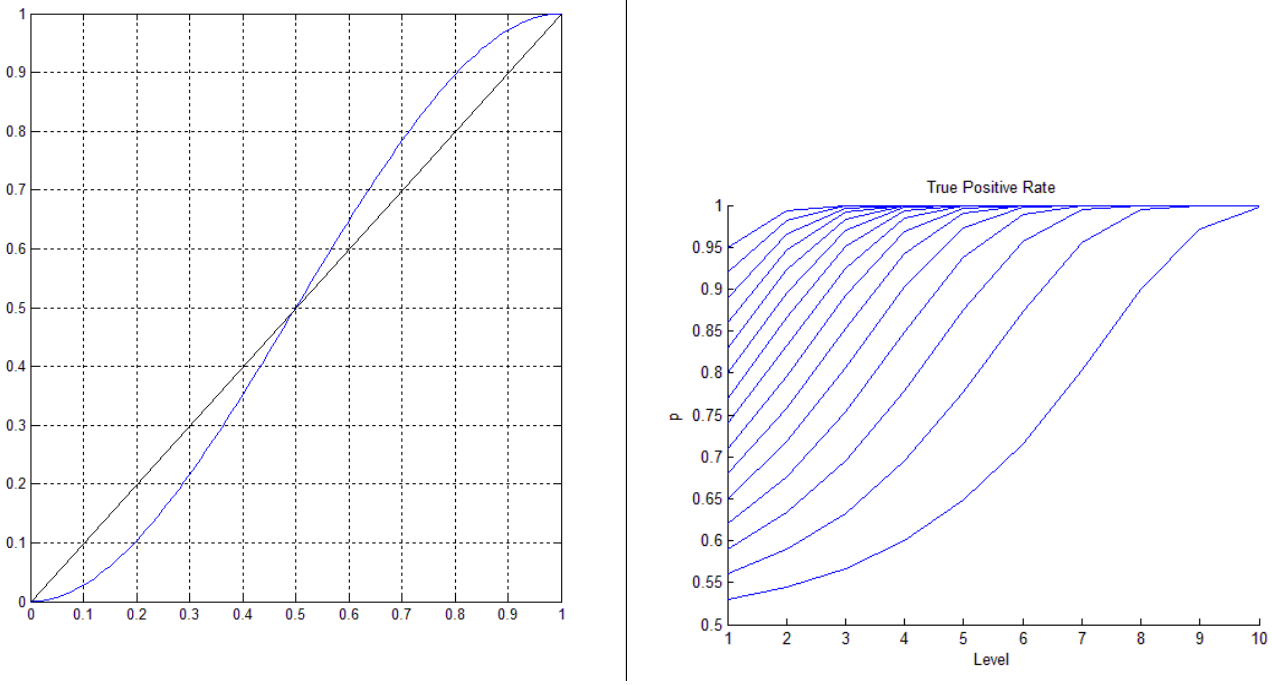

Fig. 13. Left Panel: we plot $f(\rho)$ as a function of ρ, showing fixed points at $\rho = 0, 0.5, 1$, and that $f(\rho) > \rho$ for $0.5 < \rho < 1$, but $f(\rho) < \rho$ for $0 < \rho < 0.5$. Right Panel: we show the true positive rate (the detection rate) as a function of the number of level and in terms of the ρ parameter.

We performed simulations to verify this result and to estimate how many levels are needed to obtain almost perfect detection as a function of ρ. These are shown in figure (**??**). Observe that if $\rho > 0.6$ then only a small number of levels are needed to get almost perfect performance.

IX. Discussion

The purpose of this paper was to develop a compositional vision theory which has several desirable properties and to analyze its complexity. This theory is certainly too simple and it does not address many aspects of natural images, and natural visual tasks. In its current form it would not perform at state of the art on challenging computer vision problems like the Pascal Object Detection Challenge (refs!!).

Nevertheless this theory allows a complexity analysis of what is arguably the fundamental problem of vision – how, a biological or artificial vision system could rapidly detect and recognize an enormous number of different objects. We describe a novel parallel implementation for compositional models which has several properties of the visual cortex. We quantify the potential gain of compositionality for serial and parallel implementations and demonstrate exponential gains. Perhaps more advanced models of this form, using richer appearance cues and more sophisticated representations, will be able to efficiently describe the enormous classes of visual patterns that arise in the natural world and lead to algorithms for how they can be rapidly detected? To some degree this is an empirical question which requires detailed study of the nature of images, objects, and scenes.

The theory might also serve as a "toy model" for the visual cortex. This theory shares the principle advantage of feedforward models – i.e. the ability to perform inference rapidly (at least about the coarse structure of images) – but it is also a generative model which could enable analysis by synthesis and perhaps attentional mechanisms. The theory also embodies general properties of the visual cortex. The parallel implementation leads naturally to "receptive fields" which are non-linear, are affected by both bottom-up input and top-down context. Also complexity requirements, and the executive summary principle, ensures that the receptive fields become more sensitive to complex image structures, but less sensitive to their location, as we ascend the visual hierarchy. Observe also that the theory moves in a smooth manner from low-level vision (lowest levels), to mid-level vision (middle levels), and then to high-level vision (top levels), see the hierarchical dictionaries in figure (5). It does not require, for example, an initial stage which compute image features followed by a separate later stage that does matching from the feature outputs to the objects. Instead it breaks up the problem of matching the image to an object into a recursive series of simple stages which match subregions of the image to subparts of the object. The computations required by this theory may be possible using the neuronal models developed by Mel (ref) and the conceptual models of Valiant (ref) which take computation and complexity into account.

Acknowledgements

Many of the ideas is this paper were obtained by analyzing models developed by L. Zhu and Y. Chen in collaboration with the first author. In particular, L. Zhu proposed many insights (and could be a co-author if he wanted to be). It is a pleasure to thank Tai Sing Lee and D. Kersten for discussions about how this theory might relate to the brain, and several ideas here will also appear in publications co-authored with them. C. Koch offered many stimulating ideas during early morning runs, B. Mel provided pizza while describing pyramidal cells, and S. Geman discussed compositional models over coffee. Y.Wu and S.C. Zhu have offered contrasting forms of stimulation for developing models of this type. G. Papandreou gave very useful feedback on drafts of this work. The WCU program at Korea University, and the generosity of the host S-W Lee, gave peace and time to develop these ideas.

Appendix: Simple Example of f: for discrete formulation

Now we give a simple example. We consider a one-dimensional model where each parent has two child nodes. $\mathcal{D}_0$ is the set of all integers. The next level $\mathcal{D}_1$ is the set of all even integers. Hence the leaf nodes x_1, x_2 take all integer values while their parent nodes take even integer values. We define $f(x_1, x_2) = 2[(x_1 + x_2 + 1)/4]$, where $[y]$ is the biggest integer less than or equal to y. Intuitively, $f(x_1, x_2)$ computes the (biggest?) even integer which is closest to the average $(x_1 + x_2)/2$.

To understand this in more detail. Suppose x is the state of the parent node (hence is an even integer). Then let $(x_1, x_2) = (x-i, x+i), (x-i, x+i+1), (x-i+1, x+i+1), (x-i+1, x+i+2)$ for integer i.

It follows that $x_1+x_2 = 2x, 2x+1, 2x+2, 2x+3$ respectively, and hence $[(x_1+x_2)/4] = x/2$ (x is even) and $2[(x_1+x_2)/4] = x$. Observe that $x_2 - x_1 = 2i, 2i+1, 2i+2, 2i+3$ for the four cases. Suppose we impose the condition that $|i| \leq m$ (for an integer m). This defines a neighborhood $Nbh(x)$ outside which $P(\vec{x}_{ch(\nu)}|x_\nu;\lambda_\nu) = 0$ – $x_1 \in \{x-m, x+m+1\}, x_2 \in \{x-m, x+m+2\}$ – which gives $C = 2m+3$. Note that this restricts $|x_2 - x_1| \leq 2m+3$.