Chapter VIII. Teaching computers how to see

Supplementary contents at **http://bit.ly/36RxOGX**

We have come a long way since our initial steps towards defining the basic properties of vision in **Chapter I**. We started with characterizing the spatial and temporal statistics of natural images (**Chapter II**). We summarized visual behavior; that is, how observers perceive the images around them (**Chapter III**). Lesion studies helped define specific circuits in cortex that are responsible for processing distinct types of visual information (**Chapter IV**). We explored how neurons in the retina, the thalamus, and the ventral visual cortex respond to a variety of different stimulus conditions (**Chapters II**, **V**, and **VI**).

In this chapter, we will put all of these separate bits and pieces of phenomenological observations into a coherent theoretical framework to understand how neuronal circuits orchestrate processing of visual information. We introduce computational models that instantiate this theoretical framework, endowing machines with the possibility to begin to see and interpret the visual world around us.

VIII.1. Recap and definitions

We start by summarizing key observations from previous chapters to define constraints to solve the problem of vision. A theory of vision, implemented by a computational model, should satisfy the following eight desiderata.

1. *Selectivity*. The visual system shows a remarkable degree of selectivity demonstrated by the ability to differentiate among shapes that appear to be similar at the pixel level (e.g., arbitrary 3D shapes created from paperclips, symbols, letters, and different faces). A model should be able to discriminate among images that are similar in pixel space, yet represent different objects (***Figure I-4***).

2. *Transformation tolerance*. A trivial solution to achieve high selectivity would be to memorize all the pixels in an image, i.e., a pure template matching algorithm (***Figure I-4***). This type of algorithm would not tolerate any changes in the image. An object can cast an infinite number of projections onto the retina (***Figure I-3***). These image transformations arise due to changes in an object's position with respect to fixation, its scale, in-plane or depth rotations, variations in contrast, illumination, color, or occlusion, among others (***Figure III-6***). The importance of combining selectivity and tolerance constitutes a critical feature of vision systems and a substantial challenge for computational models.

3. *Speed*. Vision is very fast, as emphasized by many psychophysical experiments, as well as neurophysiological recordings in humans and monkeys (**Section III-5**). In approximately 150 milliseconds, we can extract the presence of objects and get a good first impression of what is happening in an image. This speed imposes a constraint to the number of computational steps that the visual system can use for visual recognition tasks.

4. *Generic*. We can recognize a large variety of objects. Estimates about the exact number of objects categories that primates can recognize vary widely depending on several assumptions and extrapolations. Certain types of objects that an individual is particularly familiar with may be especially interesting; those objects may have more cortical real estate associated with them, they could be processed faster and could be independently impaired. However, independently of precise figures about the number of shapes that primates can identify, and independently of a non-uniform distribution over object classes, there exists a generic system capable of distinguishing multiple arbitrary shapes. In fact, we can even discriminate shapes seen for the first time (referred to as one-shot learning).

5. *Implementable in an image-computable algorithm*. A successful theory of vision needs to be described in sufficient detail to be implemented through image computable algorithms. An image computable algorithm takes an image as an input – or a sequence of images – and produces an output. The requirement for such a quantitative algorithm is essential because the computational implementation allows us to run simulations and to quantitatively compare the performance of the model against behavioral metrics. The simulations also lend themselves to a direct comparison between the model's computational steps and neurophysiological responses at different stages of the visual processing circuitry. The computational model can be tested with the same images used in behavioral or neurophysiological experiments.

In contraposition to image-computable models, there are various fascinating ideas and theoretical constructs about vision that have not been implemented through computational algorithms. We can refer to these ideas as language-based conceptualizations or verbal models. As a brief example to be concrete, a verbal model can state that the visual system has filters that extract color information, image edges, textures, and the presence of faces. Verbal models can be useful for the field and can inspire the development of computational models. However, verbal models are insufficiently specified and are therefore prone to misinterpretation (What exactly is a texture or a face? How is color information extracted?). Verbal models do not provide quantitative predictions (How fast will a subject distinguish between images with different edge orientations? What will be the firing rates of a population of neurons distinguish one face from another one?). Because verbal models are not well specified, they are not falsifiable. Furthermore, we cannot easily compare

different verbal models, or verbal models versus quantitative models, or verbal models and behavioral or neural responses.

An algorithmic implementation forces us to rigorously state assumptions and formalize the computational steps. In this way, computational models can also be readily compared to other models, in addition to behavior and neural responses. The implementation can also help us debug the theory by discovering erroneous hidden assumptions, processing bottlenecks, and specific challenges that the algorithms cannot solve, or where performance diverges from behavioral or neural metrics.

6. *Restricted to primates*. For simplicity, here we follow the lead of previous chapters, and we restrict the discussion of computational models to primate vision. There are strong similarities in vision at the behavioral and neurophysiological levels between macaque monkeys (one of the prime species for neurophysiological studies) and humans. Some of these models may well apply to other species (e.g., cats and rodents). Some aspects of the model may require refinement and modification for other species. When thinking about invertebrate vision (e.g., flies), there may need to be more drastic changes to the overall models.

7. *Biophysically plausible*. We aim to directly link the theoretical framework for vision to actual brain circuits, thus bridging across the three levels of analyses proposed by David Marr proposed (**Chapter I**). Therefore, the computational implementation should be based on neural networks, meaning that the model must be able to explain how computations take place in terms of the basic elements of computation in neural circuits, that is, neurons. We restrict ourselves to models that are biophysically plausible, in doing so, skipping a vast literature in computer vision where investigators try to solve similar problems without direct reference to cortical circuitry.

Pure engineering approaches to vision are useful from a practical viewpoint irrespective of whether they have any connection to brain circuits. Ultimately, in the same way that computers can be successful at chess without any direct connection to how humans play the game or airplanes can fly with only a tangential relationship to how birds fly, computer vision approaches can achieve high performance in visual tasks without mimicking neuronal circuits. Even though such algorithms can be useful in everyday tasks, they do not constitute a biophysically plausible model of primate vision.

An advantage of paying attention to neural circuits and behavior is that we can take inspiration from Biology to solve tasks that may be easy for humans and hard for machines. The insistence on biological plausibility should not be taken to imply that the operations or architectures in current networks are necessary, let alone sufficient, to understand visual computations. It is likely that we will have to make substantial changes to current neural networks, but the ultimate flavor of

the implementation needs to be mapped onto biological hardware. Finding the correct level of detail when defining biophysical plausibility remains an interesting challenge (**Section VII.10**).

8. *Restricted to the visual system*. The visual system is not isolated from the rest of the brain. There are plenty of connections between the visual cortex and other sensory cortices, memory systems in the medial temporal lobe, and frontal cortex, among other brain regions. Even though we often operationalize multiple tasks to try to separate vision from other processes as much as possible, in the real world, the lines between vision and other computations are often blurred. Connections outside the visual system also play an important role in visual processing, predominantly through feedback signals that incorporate expectations (e.g., the probability that there is a lion in an office setting is minuscule), through prior knowledge (e.g., the object looks similar to another object that we are familiar with), and through cross-modal integration (e.g., the object is likely to be a musical instrument because of the sound). As an initial simplification, and as a strategy to tackle a difficult problem, we restrict the discussion to the visual system.

VIII.2. Common themes in modeling the ventral visual stream

Several investigators have proposed computational models that aim to capture some of the essential principles behind the transformations along the primate ventral visual stream. Before discussing some of these models in more detail, we start by extracting common themes that are shared by many models.

The input to models of visual cortex is typically an image, defined by a matrix that encodes the color of each pixel. Typically, this is a 3D matrix with the Red, Green, and Blue (RGB) intensities for each pixel. Models can also work with unidimensional grayscale inputs, and it is also possible to extend the models to more than three input dimensions, for example, to consider species that can see beyond the visible portion of the spectrum for humans. Dynamic inputs can be incorporated as a sequence of frames. Because of the type of images and video available, and because of the computational resources required, most models deal with a cropped version of the entire visual field; for example, a famous computer vision study by Geoffrey Hinton's group used 224 x 224-pixel images as input.

Because the focus is often on the computational properties of ventral visual cortex, many investigators ignore the complexities of modeling the computations in the retina and LGN. The pixels are meant to coarsely represent the output of retinal ganglion cells or LGN cells. The map between pixels and degrees of visual angle is not always made explicit in the models. Most models use images with a uniform resolution as input, without considering the eccentricity-dependent sampling that is evident in the retina and throughout visual cortex (e.g., ***Figures II.7-8***). These assumptions, of course, are among the

many oversimplifications in typical computation models; images go through several transformations before retinal ganglion cells convey information to the LGN and on to cortex (**Chapter II**). Incorporating a better account of the retina and LGN circuitry will likely improve the performance and robustness of current vision models.

Most models have a hierarchical and deep structure that aims to mimic the approximately hierarchical architecture of ventral visual cortex (**Figure I-5**, **Chapter V**). The properties of deep neural networks have received considerable attention in the computational world, even though the mathematics of learning in deep neural networks with non-linear responses are far less understood than their shallow counterparts. Neocortex and computer modelers have adopted a *Divide and Conquer* strategy whereby a complex problem is divided into many simpler tasks (**Section V-10**). Ascending through the hierarchical structure of the model, units in higher levels typically have larger receptive fields, respond to more complex visual features, and show an increased degree of tolerance to transformations of their preferred features.

Most computational models assume, explicitly or implicitly, that "cortex is cortex"; that is, that there exist canonical microcircuits and computations that are repeated over and over throughout the visual circuitry. Thus, visual processing can be approximated by a hierarchy of sequential computational steps, each one of which is quite simple and encompasses basic biophysically-plausible operations such as computing dot products, applying a non-linear transformation to the integrated activity in the neuronal soma, and normalizing the outputs (**Chapter VII**).

VIII.3. A panoply of models

The oldest idea for visual object recognition is template matching, whereby the model stores a certain number of templates, and any new image is compared at the pixel-by-pixel level with those templates. Straightforward template matching at the pixel level does not work well for pattern recognition. Even shifting a pattern by one pixel would pose significant challenges for an algorithm that merely compares the input with a stored pattern on a pixel-by-pixel fashion.

As noted at the beginning of this chapter, a key challenge in visual recognition is that an object can lead to an infinite number of retinal images. If all objects were always presented in a standardized position, scale, rotation, and illumination, recognition would be considerably easier. Based on this notion, several approaches are based on trying to transform an input image into a prototypical canonical format by shifting, scaling, and rotating objects. The type of transformations required is usually rather complex. While ingenious computational strategies can overcome some of these problems, it is not entirely clear how the brain would implement such complex rotations and inferences, nor

is there any apparent link from this family of models to the neurophysiological responses observed along ventral visual cortex.

Multiple models are based on describing an object based on its parts and interactions among those parts. The idea behind this approach is that there could be a small dictionary of object parts and a small set of possible interactions that act as building blocks of all objects. This intuition can be traced back to the prominent work of David Marr (1945–1980), who proposed that the constituent parts are based on generalized cone shapes.

The artificial intelligence community has also embraced the notion of structural descriptions. In the same way that a well-behaved mathematical function can be decomposed into a sum over a certain basis set (e.g., polynomials or sine and cosine functions), the idea of thinking about objects as a sum over parts is attractive because it may be easier to detect these parts in a transformation-invariant manner. In the simplest instantiations, these models are based on merely detecting a conjunction of object parts, an approach that suffers from the fact that part rearrangements would not impair recognition by the model but, in reality, they should (e.g., a house with a garage on the roof and the chimney on the floor). More elaborate versions include interactions between object parts and relative positions of different object parts. This approach converts the problem of object recognition into the problem of object part recognition plus the problem of recognizing characteristic relations between such parts. It is not entirely evident that object part recognition should be easier than object recognition, nor is it obvious that *any* object can be uniquely and succinctly described by a universal and small dictionary of simpler parts. The distinction between objects and parts is not well defined either. There have been few computational implementations of these part-based structural descriptions. More importantly, it is not entirely apparent how these structural descriptions relate to the neurophysiology of the ventral visual cortex. Despite these caveats, the idea of decomposing an object into parts, and the computational advantages of a compositional representation are appealing and worth studying further.

A series of computational algorithms, typically rooted in the neural network literature, attempt to build deep structures whose purpose is to reconstruct the inputs. One version of this type of model is called an *autoencoder*. In an extreme version of this type of network, there is no information loss along the deep hierarchy, and backward signals carry information capable of re-creating arbitrary inputs in lower visual areas. There are interesting applications for such autoencoder deep networks, in particular, the possibility of performing dimensionality reduction. However, the purpose of cortex is precisely the opposite of perfect input reconstruction, namely, to lose information in biologically helpful ways (**Chapter VI**). The data processing inequality stipulates that the information contained in a signal cannot be increased via any kind of processing, without adding external information. Consider an input image that is processed via a sequence of steps from A to B to C. The representation at the C level can

contain less information about the original image than at the A level (as a trivial example, we can multiply the signal in B times 0). The representation at the C level can contain the same amount of information as at the A level (as a trivial example, we can copy the signal from A to B to C). However, the representation at the C level cannot contain more information about the original image than the original image itself; the processing steps cannot create new information. Ascending through the visual system, and without adding external information, information content has to either decrease or stay the same. It is not clear why one would build an entire network to copy the input (even if the copy requires fewer units). In other words, a key goal of ventral visual cortex is to extract relevant information such as object identity despite changes in the input at the pixel level.

Not all information is lost due to processing along ventral cortex, not even if that information is orthogonal to a given task. For example, it is possible to read out object location from neurons in inferior temporal cortex despite their relative tolerance to position changes (**Chapter VI**). However, whereas the retina has neurons with receptive field sizes spanning a few minutes of an arc (1 degree of visual angle = 60 minutes of arc), and which can follow temporal changes at a rate of 100 Hz, neurons in inferior temporal cortex are coarser in space and time. Even though we do not have a complete understanding of what information is lost in the transformations along the ventral visual cortex and what is preserved, it is clear that some information is lost; the goal of cortical processing is *not* reconstruction with perfect fidelity.

Particularly within the neurophysiology community, there exist several "metric" modeling approaches where investigators attempt to parametrically define a space of shapes and then record the activity of neurons along the ventral visual stream in response to the pre-defined parameters. For example, in some cases, investigators start by presenting different shapes in search of a stimulus that elicits strong responses. Subsequently, they manipulate the "preferred" stimulus by removing different parts and evaluating how these transformations modify the neuronal responses. While interesting, these approaches suffer from the difficulties inherent in considering arbitrary shapes that may or may not constitute truly "preferred" stimuli. Additionally, in some cases, the transformations examined only reveal anthropomorphic biases about what features could be relevant. Another approach is to define shapes parametrically. For example, several exciting studies considered a family of simple geometric shapes parametrized by different types of curvatures, and modeled neuronal responses in a six-dimensional space defined by a sum of Gaussians with parameters given by the curvature, orientation, relative position, and absolute position of the contour elements in the display. This approach is appealing because it has the attractive property of allowing investigators to plot "tuning curves" similar to the ones used to represent the activity of units in earlier visual areas. However, this approach also makes strong assumptions about the type of shapes preferred by the units. These parametrized stimulus shape

descriptors have not readily lent themselves to image computable models applicable to any possible natural image.

The visual system does much more than recognition, and the development of the visual system involves partly unsupervised learning from the environment during navigation, social interactions, and playing. Nevertheless, many of the computational models of vision have been tested in visual object recognition tasks. An appealing feature of recognition tasks is that they can be readily evaluated at the behavioral, neurophysiological, and computational levels. Additionally, many recognition tasks can be directly used in non-human animals. Before we delve into state-of-the-art image computable models, let us define the approaches to solve object recognition tasks.

VIII.4. A general scheme for object recognition tasks

INSERT **Figure VIII-1** AROUND HERE

***Figure VIII-1**. **General scheme for visual classification tasks.** Features are extracted from an image (or video). Those features are used to train a classifier via supervised learning. The resulting boundary is used to assign labels to novel images (i.e., images different from the ones used during training).*

Figure VIII-1 illustrates a typical approach in computer vision approaches to solve visual recognition tasks. Consider a series of M labeled images $(\boldsymbol{x}_i, y_i)$ where $i=1,...,M$, $\boldsymbol{x}$ is a matrix representing the image, and y is a label (e.g., whether a face is present or not, or a categorical label applied to the image). A set of features $\boldsymbol{f}$ is extracted from the images: $\boldsymbol{f}_i=g(\boldsymbol{x}_i)$ where g is the computational model. Those features may include properties such as edges, principal components, and colors, among many others. How those features are chosen is one of the key aspects that separates different computer vision algorithms. The function g that extracts features could be hard-coded, or it could have multiple parameters, we shall call them $\boldsymbol{w}$, that require tuning for a specific recognition challenge.

We will consider how those parameters are learned in **Section VIII-6**. For now, let us assume that the parameters $\boldsymbol{w}$ are known and fixed. After extracting an adequate set of features $\boldsymbol{f}$, a supervised learning scheme is used to learn the map between those features and the labels $\boldsymbol{y}$. For example, a support vector machine (SVM) classifier with a linear kernel may be used to learn the correspondence between the features and the labels. This process is analogous to the readout from a population of neurons described in **Section VI.7**, except that here we use computationally extracted features as opposed to the biologically extracted features encoded in the firing rates.

A cross-validation procedure is followed (**Section VIII.8**), separating the data into a training set and a test set to ensure that the algorithm is evaluated on new data; that is, to ensure that merely memorizing every training data point does not lead to good performance. Some investigators like to separate data into

a training set, a validation set, and a test set. In this case, investigators fine-tune hyperparameters of the model by considering the training and the validation set, and then use the test set for the final performance evaluation. After training, the algorithm is evaluated with the images in the test set. By using different algorithms applied to the same data, the merits of alternative computational models can be quantitatively compared.

VIII.5. Bottom-up hierarchical models of the ventral visual stream

INSERT **Figure VIII-2** AROUND HERE

***Figure VIII-2**. **A deep convolutional neural network.** Schematic architecture of the AlexNet deep convolutional neural network, consisting of the input layer, five convolutional layers (conv1 through conv5), two fully connected layers (fc6 and fc7) and the output layer. The numbers below each layer denote the size in pixels. The small elements inside each layer denote the size of the convolution filters. Modified from Krizhevsky 2012.*

Let us revisit the type of hierarchical neural network models introduced in **Chapter VII**. A hierarchical network model can be described by a series of layers $l=0, 1, \ldots, N$ (***Figure VII-4***). Each layer contains n_l x n_l units arranged in a matrix (the matrix does not need to be square), each unit having a circumscribed receptive field, and therefore being activated by a specific location in the image. Additionally, there may be multiple different filters K_l at each location (sometimes referred to as kernels), creating a collection of such matrices; for example, the input image ($l=0$) may have $K_0=3$ colors. Such a collection of matrices in each layer is called a tensor. The activity of all units in each layer can be represented by the value $\mathbf{x}_l$ ($\mathbf{x}_l \in \mathbb{R}^{n_l \times n_l \times K_l}$). Each entry in $\mathbf{x}$ is typically represented by a scalar value, usually referred to as its activation, and which can be coarsely interpreted as the firing rate of the unit. The image is the input to the initial layer; $\mathbf{x}_0$ represents the pixel values in the image. A different image would lead to a different set of activations $\mathbf{x}_l$ (in the previous section, we used the subindex i to denote the activations, or features, for each individual image, which is dropped here for simplicity and should not be confounded with the subindex l used here to denote a given layer).

From one layer to the next, the matrices are transformed through various convolution operations (**Section VII.7**), non-linearities like the ReLU operation (**Figure VII.2**), and through pooling operations like max-pooling (**Figure VII.6**). In the most typical scenario, these operations take place between layer l and layer $l+1$. All of these operations together are referred to as a *convolution block*, including the convolution operation itself, ReLU, and the pooling operation. In fact, a single biological neuron may implement all of these computations as schematically illustrated in ***Figure VII-1***.

This formulation based on sequential processing assumes that the activity in a given layer depends exclusively on the activity pattern in the previous layer. This simplification implies that at least three types of connections are ignored

(***Figure VII-4***): (i) connections that "skip" a layer in the hierarchy (e.g., synapses from V1 directly onto area V4, skipping area V2); (ii) top-down connections (e.g., synapses from V2 to V1), and (iii) connections within a layer (e.g., horizontal connections between neurons with similar preferences in V1). Some variations, like the so-called ResNet architecture, also include connections that bypass some of the layers (introduced in **Section VII.4**).

It is tacitly assumed in most models that there exist general rules, often summarized in the epithet "cortex is cortex," such that only a few types of transformations are allowed in the computations from one layer to the next. One of the early models that aimed to describe object recognition, inspired by the neurophysiological findings of Hubel and Wiesel (**Chapter V**), was the *Neocognitron*, developed by Japanese computer scientist Kunihiko Fukushima. This model had two possible operations: a linear tuning function (performed by "simple" cells) and a non-linear OR operation (performed by "complex" cells, **Section V.5**). These two operations were alternated and repeated through the multiple layers in the computational hierarchy. This model demonstrated that such linear/non-linear cascades can achieve scale and position tolerance in a letter recognition task. The Neocognitron architecture inspired several subsequent efforts.

One such effort to expand on the computational abilities of the Neocognitron is the HMAX model developed by Max Riesenhuber, Thomas Serre, and Tomaso Poggio at MIT. This model is characterized by a purely feed-forward and hierarchical architecture. An image, represented by grayscale values, is convolved with Gabor filters (**Section V.7**) at multiple scales and positions to mimic the responses of simple cells in V1. Like other computational efforts, the model consists of a cascade of linear and non-linear operations. These operations come in only two flavors in the model: a tuning operation and max-pooling operation. The HMAX and similar architectures have been submitted to several tests, including comparison with psychophysical measurements and neurophysiological responses, which we will discuss later in this chapter.

This family of models is referred to as *deep* neural networks, as opposed to *shallow* networks where the goal is to perform all computations in a single layer. Because these models are image computable, they can also be directly used to solve computer vision tasks. Indeed, in the last decade, these deep hierarchical models have gained appreciation and momentum in computer vision. One such model, often referred to as *AlexNet*, introduced by Alex Krizhevsky and Geoffrey Hinton in 2012 (***Figure VIII-2***), caused an uproar in the computer vision world because it led to a considerable improvement in the ability to label objects in computer vision competitions. AlexNet subsequently inspired progress in many other pattern recognition problems (**Chapter IX**). A critical feature in AlexNet and many other neural networks is the ability to tune the parameters $\boldsymbol{w}$ to improve performance.

Historically, many computer vision efforts consisted of trying to develop better and better features to extract from the image. These features were then submitted to a suitable classifier. All the task-dependent learning happened at the level of the classifier. Various types of features were found to be generally useful for object classification tasks, including extraction of edges, colors, principal components, shift-invariant feature transformations (SIFT), corners, spatial frequency decomposition, and many others. A classifier, such as an SVM, was then in charge of learning the map between those features and the corresponding image labels, as in **Figure VIII-1**. In the Neocognitron, and in the initial implementation of the HMAX architecture, the weights between layers were hand-crafted and fixed.

In stark contrast to these approaches, the bulk of the work nowadays consists of building end-to-end trainable systems, typically with a fixed architecture with randomly initialized weights, and where all the weights are plastic and can be modified to achieve the best possible classification accuracy.

VIII.6. Learning the weights

We have not described yet how the feature extraction parameters $\boldsymbol{w}$ are set. In general, we will consider neural network models (**Section VII.5**), where the main parameters are weights that dictate how activity in a given unit impacts activity in the post-synaptic target units, typically in the next layer. Let us now consider an example of a way of learning those weights that illustrates the interesting computations that can be performed by neural networks. The weights can be learned in a supervised manner (where we have labels for each image), or in an unsupervised manner (e.g., automatically extracting statistical regularities in the images in the environment). Here we will focus on supervised learning strategies, and we will come back to unsupervised tuning of weights in **Section VIII.17**.

One of the earliest instantiations of a biologically-inspired computational algorithm, a two-layer neural network called the *perceptron*, can be trained to perform interesting classification tasks. Imagine that we have some data that we want to classify into two possible groups. For example, there may be a collection of images of dogs or cats (each image contains only one animal), and we want to teach the algorithm to distinguish whether an image contains a dog or a cat. Each image, indexed by i, is a matrix of grayscale values that can be vectorized and represented by $\boldsymbol{x}_i$. With each image, we have an associated label y_i=+1 (dog) or -1 (cat). We have a training set consisting of multiple such example images. In this type of exercise, it is always important to separate the data into a training set (used to fit parameters) and a test set (used to evaluate performance), a process referred to as *cross-validation* (**Section VIII.8;** see further discussion in **Section IX.12**). In the two-layer perceptron network, we will consider the input to the output unit to be $\mathbf{w}\bullet\mathbf{x}$, where $\bullet$ represents a dot product. The output y will take the value $+1$ if $\mathbf{w}\bullet\mathbf{x} - \gamma > 0$, and -1 otherwise, where γ is a threshold value. The

perceptron learning rule tells us how to choose the weights $\mathbf{w}$ to minimize the error in this classification task.

INSERT **Figure VIII-3** AROUND HERE

***Figure VIII-3**. **The weights in a deep neural network can be learned by using backpropagation.** Deep convolutional neural networks take advantage of backpropagation, an efficient procedure to train the weights in a supervised learning fashion. **A**. Example 3-layer neural network. **B**. To change the weight who(2,1) we calculate its effect on the total error by using the chain rule (see text for details). **C**. Similarly, to change the weight wih(2,2), we propagate the error throughout the network (see text for details, adapted from Matt Mazur).*

Instead of a binary classification task, we may be interested in approximating a given output function $h(s)$ (for example, $h(s)$ could represent the firing rate of a neuron in cortex in response to a stimulus s). For a given stimulus s, $h(s)$ is the target output for the neural network, and we define $\hat{h}(s)$ to be the actual output of the network. The error is the squared difference between the two: $E = \left(h(s) - \hat{h}(s)\right)^2$; this Euclidian distance is a typical way of evaluating the error, which has the nice property of being differentiable, which will become useful soon. Gradient descent refers to changing $\mathbf{w}$ to minimize the error in this task by making adjustments to $\mathbf{w}$ along the direction of greatest change in the error, $\mathbf{w} \rightarrow \mathbf{w} + \epsilon \nabla_{\mathbf{w}} E$, where ε is a learning rate, and $\nabla_w E$ is the gradient of the error in the direction of $\mathbf{w}$.

Classification problems need not be restricted to two classes, like cats versus dogs. In general, the goal is to take an image and assign a label to it. For example, the goal may be to detect whether the image contains a handwritten 0, 1, 2, … , or 9, as in the MNIST dataset (**Section VIII.10**), to distinguish cats versus dogs, or to identify a face.

The situation becomes more complex when we have multiple layers. Now, changing the weights in one layer will impact the next layer, which will, in turn, impact the next layer, and those changes are propagated all the way to the output. We need to take all of these interdependencies into account when tuning the weights in a deep neural network. One of the most successful ways of adjusting the weights via supervised learning in a deep neural network is *backpropagation*, where the difference between the target outputs and current outputs (that is, the error) is propagated back via gradient descent throughout the entire network. Backpropagation is an elegant example application of the chain rule from calculus.

Let us follow one simple example step-by-step to describe the concept of backpropagation. Consider the three-layer network shown in ***Figure VIII-3*****A**. The network consists of an input layer with two units whose activation values are represented by i_1, i_2, a hidden layer with two hidden units whose activation values are represented by h_1, h_2, and an output layer with two output units with

activations o_1, o_2. The term "hidden unit" in a neural network is a somewhat strange nomenclature that refers to all the units that are neither the input or the output. Perhaps the term "intermediate unit" would be more reasonable, but the jargon of hidden units has stuck in the field. The weight from the input unit i to hidden unit h is $w_{ih}(i,h)$, and the weight from hidden unit h to output unit o is $w_{ho}(h,o)$. The bias for the hidden layer is b_{ih}, and the bias for the output layer is b_{ho}. The net input to each hidden unit is:

$$net_{h_1} = i_1 * w_{ih}(1,1) + i_2 * w_{ih}(2,1) + b_{ih} \quad \textbf{Equation VIII.1}$$

$$net_{h_2} = i_1 * w_{ih}(1,2) + i_2 * w_{ih}(2,2) + b_{ih}$$

Instead of the ReLU operation (***Figure VII.2***), here we calculate the output of each hidden unit by passing the net inputs through the non-linear logistic function:

$$h_1 = \frac{1}{1+e^{-net_{h1}}} \quad \textbf{Equation VIII.2}$$

$$h_2 = \frac{1}{1+e^{-net_{h2}}}$$

The outputs of the hidden units are the inputs to the output units. The net input to each output unit is:

$$net_{o_1} = h_1 * w_{ho}(1,1) + h_2 * w_{ho}(2,1) + b_{ho} \quad \textbf{Equation VII.3}$$

$$net_{o_2} = h_1 * w_{ho}(1,2) + h_2 * w_{ho}(2,2) + b_{ho}$$

and those are passed through a logistic function as well:

$$o_1 = \frac{1}{1+e^{-net_{o1}}} \quad \textbf{Equation VIII.4}$$

$$o_2 = \frac{1}{1+e^{-net_{o2}}}$$

Now, given the inputs i_1, i_2 (we can think of these as the image that we are trying to classify), we obtain the outputs o_1, o_2. Our target values are the outputs $target_{o1}$, $target_{o2}$; we can think of these target values as the desired probabilities for class 1 and class 2 labels. For example, if we are classifying an image as a cat or a dog, the desired probabilities maybe 0 and 1 for cats, and 1 and 0 for dogs. The total error is:

$$E_{total} = E_{o1} + E_{o2} = 0.5\,[\,(target_{o1} - o_1)^2 + (target_{o2} - o_2)^2\,] \quad \textbf{Equation VIII.5}$$

Now imagine that we change one of the weights, say $w_{ho}(2,1)$, what would we expect to happen to the total error (***Figure VIII-3*B**)? If we change $w_{ho}(2,1)$, that will cause a change in net_{o1} (but not in net_{o2}, see **Equation VIII.3**). Changing net_{o1} will, in turn, cause a change in o_1 (but not o_2). The change in o_1 will impact E_{total}. We can calculate how much we expect the total error to change by using the chain rule, and decomposing the gradient of E_{total} with respect to $w_{ho}(2,1)$ into each of these parts:

$$\frac{\partial E_{total}}{\partial w_{ho}(2,1)} = \frac{\partial E_{total}}{\partial o_1}\frac{\partial o_1}{\partial net_{o1}}\frac{\partial net_{o1}}{\partial w_{ho}(2,1)}$$ **Equation VIII.6**

Here is where the definition of the square error in **Equation VIII.5** comes in handy because it is easy to calculate derivatives. Each of these three factors can be readily computed by calculating the derivatives in **Equations VIII.5**, **VIII.4**, and **VIII.3**, respectively:

$$\frac{\partial E_{total}}{\partial o_1} = (-1) * 2 * 0.5 * (target_{o1} - o_1)$$ **Equation VIII.7**

$$\frac{\partial o_1}{\partial net_{o1}} = o_1 * (1 - o_1)$$ **Equation VIII.8**

$$\frac{\partial net_{o1}}{\partial w_{ho}(2,1)} = h_2$$ **Equation VIII.9**

In order to make the total error smaller, we will change the weights according to:

$$w_{ho}(2,1) \rightarrow w_{ho}(2,1) - \varepsilon\frac{\partial E_{total}}{\partial w_{ho}(2,1)}$$ **Equation VIII.10**

where ε is a learning rate that controls how big the changes in the weights are in each step.

We can follow the same procedure to change $w_{ho}(2,2)$, $w_{ho}(1,2)$, and $w_{ho}(1,2)$. In general, in a neural network, we want to change *all* the weights to make the output as close as possible to the target. How do we change the weights from the input units to the hidden units, such as $w_{ih}(2,2)$? We follow the same procedure (***Figure VIII-3*C**), back-propagating the error all the way from E_{total} down to the weight that we want to change. Going through multiple layers requires a few more maneuvers, but it follows the same ideas as above. The dependence of the total error on $w_{ih}(2,2)$ goes through hidden unit h_2 (and not h_1). Therefore, we can write:

$$\frac{\partial E_{total}}{\partial w_{ih}(2,2)} = \frac{\partial E_{total}}{\partial h_2}\frac{\partial h_2}{\partial net_{h2}}\frac{\partial net_{h2}}{\partial w_{ih}(2,2)}$$ **Equation VIII.11**

The last two factors are straightforward (in analogy to **Equation VIII.8-9**):

$$\frac{\partial h_2}{\partial net_{h2}} = h_2(1 - h_2)$$

$$\frac{\partial net_{h2}}{\partial w_{ih}(2,2)} = i_2$$

The dependence of the total error on h_2 goes through both output units. Therefore, the first factor in **Equation VIII.11** becomes:

$$\frac{\partial E_{total}}{\partial h_2} = \frac{\partial E_{o_1}}{\partial h_2} + \frac{\partial E_{o_2}}{\partial h_2} = \frac{\partial E_{o_1}}{\partial o_1}\frac{\partial o_1}{\partial net_{o1}}\frac{\partial net_{o1}}{\partial h_2} + \frac{\partial E_{o_2}}{\partial o_2}\frac{\partial o_2}{\partial net_{o_2}}\frac{\partial net_{o_2}}{\partial h_2}$$

$$\frac{\partial net_{o_1}}{h_2} = w_{ho}(2,1)$$

$$\frac{\partial net_{o_2}}{h_2} = w_{ho}(2,2)$$

According to **Equation VIII.8**, we have $\frac{\partial o_1}{\partial net_{o1}} = o_1 * (1 - o_1)$ and, similarly, $\frac{\partial o_2}{\partial net_{o2}} = o_2 * (1 - o_2)$. According to **Equation VIII.7**, we have $\frac{\partial E_{o1}}{\partial o_1} = (-1) *$

$(target_{o1} - o_1)$ and, similarly, $\frac{\partial E_{o2}}{\partial o_2} = (-1) * (target_{o2} - o_2)$. We want to change the weight $w_{ih}(2,2)$ the following amount:

$$w_{ih}(2,2) = w_{ih}(2,2) - \varepsilon \frac{\partial E_{total}}{\partial w_{ih}(2,2)}$$

The beauty of these steps is that we can go on applying the chain rule no matter how deep the network is. In fact, scientists and engineers routinely build neural networks that have more than a hundred layers and train them using essentially the same backpropagation procedure outlined here. In addition, iterating backward from the last layer avoids redundant calculations of intermediate terms in the chain rule such that all the previous terms from late layers can be reused for early layers.

Given an example with input values i_1, i_2, and target output values $target_{o1}$ and $target_{o2}$, we perform the steps outlined above to change all the weights in the network. In general, we have many examples consisting of input pairs and target output values (**Section VIII.4**). In *stochastic gradient descent*, we go through those examples one by one, changing the weights after each iteration. A batch can also be introduced where the calculations are made for a few examples before actually changing the weights. The word stochastic refers to choosing the examples and order randomly.

Once we go through all the examples in the training set in a given iteration, we start a new iteration, re-adjusting the weights. This procedure goes on until convergence. The learning rate ε plays an important role. If the learning rate is too large, the procedure may diverge. If it is too small, convergence can be very slow, and the algorithm may also get stuck in local optima. Several heuristic approaches have been developed to adjust the learning rate, sometimes even changing it throughout learning (with faster learning at the beginning and slower learning towards the end). It is also possible to use distinct learning rates for different layers. There are multiple other variations that go beyond the scope of this chapter and can be found in computer vision and machine learning textbooks.

In general, the initial weight values are assigned randomly, but there has also been considerable empirical investigation of the virtues of different starting conditions. Biological brains probably do not start with entirely random connectivity weights. There is an inherent structure that subsequent plasticity rules act upon during learning. This initial structure could be the product of evolution and also activity-dependent developmental processes.

A particularly interesting starting condition arises when a network is “pre-trained” with a somewhat different problem than the one we are trying to solve. The application of weights trained in one problem to another problem is referred to as “transfer learning.” For example, imagine that we want to build a network that recognizes hand-written letters. One might first train a network to recognize hand-written digits and use the weights from this pre-trained network as a

starting condition for the letter recognition problem. Many of the computations needed to build a handwritten recognizer may be shared with those required to build a handwritten letter recognizer. Thus, starting with such a pre-trained network could accelerate training, and may also lead to the same accuracy using fewer training examples. It is not immediately obvious what kind of problems would be suitable for this type of transfer learning approach. Intuitively, if the two tasks are very similar, then this approach may be advantageous, but if the two tasks are too different, then such pre-training might not provide any advantages.

As noted above, the chain rule enables the propagation of error through deep networks with many layers, and more layers typically mean more weights that need to be tuned. To adjust large numbers of weights, it is useful to have many example pairs of inputs and target outputs. In the case of image classification algorithms, these examples come in the form of images and labels.

VIII.7. Labeled databases

INSERT **Figure VIII-4** AROUND HERE

Figure VIII-4. ***Example images from the ImageNet dataset.*** *The availability of datasets consisting of millions of labeled images provided a significant boost to supervised learning algorithms for object categorization.*

There has been significant progress in a large number of image categorization tasks in the computer vision community. This progress has been fueled by a combination of increased computational resources, access to a large number of digital images (as well as videos), and exciting competitions in academic conferences.

The last decade has seen an explosion in the number of digital images available on the web. In 2019, users uploaded on the order of a few *billion* digital images every day (for instance, Facebook: ~300 million pictures per day; Instagram: ~100 million pictures per day). In addition, many users are inadvertently extremely helpful to the computer vision community by providing more and more content in the form of "tags," brief captions, "likes," and other commentaries. Every minute in 2019, humans took more digital photos than the total number of photographs available in the entire world a century ago. There has also been a rapid increase in the amount of video material being uploaded (for instance, YouTube: 0.5 million hours of video per day). In parallel to the availability of imagery, there are also now accessible platforms such as Mechanical Turk, where users can answer queries on images for a small fee. Investigators upload their images, pay subjects in Mechanical Turk to label them, leading to fascinating datasets with image content annotation and labels. Images, content, and the concomitant exponential growth in computational power have opened the doors to use networks with millions of tunable parameters for recognition tasks.

A typical example is the "ImageNet" large scale visual recognition challenge. This dataset consists of color images downloaded from the web, each one associated with a label. In a typical instantiation of this competition, those labels include categories like "volcano," "hippopotamus," "dome," or "African elephant" (***Figure VIII-4***). The 2014 version of ImageNet has been used extensively to compare different computer vision algorithms for object classification, and included 1,000 object classes, 1,281,413 images for training (732-1,300 images per class), and 100,000 images for testing (100 images per class).

The fact that the images are downloaded from the web is a blessing and a curse: a blessing because the images encompass a wide diversity of properties where the object responsible for the image label can appear in multiple positions, at multiple scales, rotations, colors, illumination, degrees of occlusion, and other variations. To some coarse approximation, this may reflect the natural distribution of objects in the world. This approximation is not exactly accurate because those images are filtered through the lenses and biases of human photographers. For example, there are probably very few images of a hippopotamus in the middle of a rainy night. Images taken from the web are also a curse because of their uncontrolled nature and a large number of other somewhat miscellaneous contextual factors that contribute to classification. For example, in the three pictures of "Domes" in ***Figure VIII-4*** (top row, third column), the pixels in the upper left are mostly blue. It seems likely that when people take pictures of Domes, the pictures are set against the sky, and there is a higher propensity of blue at the top. In contrast, none of the "Baboon" examples (bottom row, third column) contain blue at the top. Blue at the top is not a unique identifying feature of Domes, though. Many other pictures also typically contain blue at the top (e.g., volcanos, elephants, zebras, and castles). There are also probably pictures of Domes without blue at the top, and pictures of baboons with blue at the top. The point is that there are many complex correlations in the images that are only minimally related to the object labels themselves. Depending on the particular task and objective, these contextual correlations can represent a confounding factor or a useful property.

Another curious property of the ImageNet dataset is that several of the categories are quite intriguing. In fact, there are many category labels that I would have to look up in the dictionary (e.g., tench, junco). Additionally, many of those 1,000 classes correspond to specialized and refined groups of animals (how many humans can distinguish between the whiptail lizard, the alligator lizard, the green lizard, the komodo lizard, and the frilled lizard?). Nevertheless, computers are trained to recognize these categories from scratch, and the distinction between whiptail lizards and frilled lizards may be as crystal clear for a computer as the differences between sunglasses and domes are for humans. The point of these competitions was to quantitatively evaluate and compare computer vision algorithms, which can be readily done with whiptail and frilled

lizards. We will return to discuss further aspects of training datasets in **Section IX.12**.

Computer vision algorithms that capitalize on supervised learning approaches with randomly initialized weights are typically data-hungry. ImageNet contains on the order of 1,000 categories x 1,000 examples per category = 10^6 images. A database of this size was not available before circa 2012, and therefore these images provided a nice playground to develop, refine, and build more complex deep convolutional neural networks. Given the huge amount of digital content, we should expect enormous growth in the size of vision datasets.

Another noteworthy aspect of ImageNet and similar datasets is that it empowered direct comparison and benchmarking of different algorithms. Comparing how algorithm X processes a dataset I_x to how an algorithm Y processes a different dataset I_y is challenging, a bit of an apples versus bananas comparison. Although this is a simple concept, benchmarks based on standard datasets are not common in other domains. For example, in Neuroscience, almost every lab creates its own tasks, using their custom-made images, rendering results challenging to integrate and compare, and data sharing is still in its infancy.

VIII.8. Cross-validation is essential

Armed with such a large dataset of labeled images, we prepare to train a computational algorithm to learn the map between the image features and their labels. Critically, we do not want to merely memorize every single image/label pair. Instead, we want to be able to infer correct labels for *novel* images that the algorithm has never seen before. To avoid sheer rote memorization camouflaged as high performance, it is critical to use *cross-validation* by separating the images within each label into a *training set* and a *test set*. All the model parameters can be modified *ad libitum* only while examining the training set, but we are not allowed to change any more parameters when evaluating the model on the test set.

In many cases, we use multiple random splits of the same dataset into non-overlapping training and test sets. The proportion of trials that go into the training set may not matter too much; the data can be split with 50% of the examples going into the training set, or 70%, or even leave-one out where a single example is used for testing in each iteration. This procedure is repeated multiple times, and results are reported as the average performance over all random splits plus a measure of variation within the splits. As a control, it is useful to randomly shuffle the image labels, repeat the same procedure, and also report average and variation in performance for the case of shuffled labels.

In deep convolutional network models, the training step typically amounts to modifying the weights in a supervised fashion via back-propagation (**Section**

VIII.6). However, it may also be possible to explore other aspects of the model, including its architecture, number of layers, size of each layer, and computational motifs, as long as we limit ourselves to the training set. After training, the algorithm is tested with new images, and the fraction of images that are correctly labeled is reported.

In general, splitting the data into a training set and a test set is done randomly. To the extent that there are no duplicates in the dataset, no crops of the same images, and no other potential confounds, a random split should suffice to avoid deceiving ourselves via rote memorization, though we will have more to say about cross-validation in **Section IX.12.** Importantly, datasets should be carefully curated to avoid problems, such as duplicates or slightly modified versions of the same image, in order to properly assess performance. For example, consider an algorithm to recognize the faces of celebrities. People like to crop the same images of celebrities over and over again, and upload them to their favorite social media. If a random split causes the training set to contain the same or essentially the same picture as in the test set, then we are not doing cross-validation properly, and we may be deluding ourselves into thinking that the algorithm is more impressive than it actually is.

VIII.9. A cautionary note: lots of parameters!

For modern neural networks with many layers, an intriguing aspect of the backpropagation procedure outlined in **Section VIII.7** to learn from examples is that adjusting the weights involves an enormous number of free parameters. Consider an image of 256 x 256 pixels with three colors: this amounts to 196,608 inputs. If there are 1,000 possible output category labels, and in the simplest scenario of mapping the inputs directly onto the outputs, we would have about 200×10^6 parameters. The ImageNet dataset contains on the order of 10^6 images. In other words, the number of weights in a neural network (free parameters) can be orders of magnitude larger than the number of training examples.

More parameters than constraints can be problematic. As a simple example, let us go back to basic linear algebra and consider a system of linear equations with 4 unknowns. In general, if we have 4 “independent” equations, we are guaranteed to have a unique solution. However, if we only have 2 equations, the system is underdetermined; without any additional constraints, there are infinite possible solutions. The same problem arises when fitting a curve. If the curve has 10 free parameters, and we only have five data points, there are infinite solutions, and it is easy to overfit; that is, to fit the data with a fancy curve that may describe the data exactly, with zero error, but which does not extrapolate to new data. For example, consider a plot with the number of women versus the number of men in a given state, showing data from five states. We would probably not want to fit a polynomial of degree 10 to describe the data!

These classic examples of underdetermined systems and overfitting are well-studied in high school math classes. What is surprising is that the most successful computer vision systems available today work precisely in this overfitting regime. As an illustration of the problem of hyperparametrization, we can randomly shuffle the labels in the training set in ImageNet: an elephant is labeled "chair," another elephant is labeled "tree," a is labeled "sunglasses," and so on. A computer vision system can be trained to achieve high performance *on the training set* in this randomly shuffled set. In other words, a network with more parameters than training examples has a vast expressive power and can even memorize the entire data set. Of course, such a network would be at chance in the test set, further emphasizing the importance of cross-validation.

There is currently significant interest in understanding how neural networks can still perform well on the test set, and thus avoid overfitting, despite the enormous number of free parameters.

One way towards alleviating the potential problem of overfitting is to use more constraints for the same number of parameters. In many computer vision problems, one way to increase the number of constraints is to obtain more data. Getting labeled data can be a bottleneck in many practical applications, as well as in more biologically plausible learning mechanisms. Therefore, there is interest in ways to increase the amount of data without additional labeling, an idea generally referred to as "data augmentation." For example, consider a dataset like ImageNet. One could take each image and crop it, horizontally flip it, blur it, rotate it, add noise, and then use it as a separate training example with the same label.

VIII.10. A famous example: digit recognition in a feed-forward network trained by gradient descent

As an example application of these ideas, consider the task of learning to recognize hand-written digits from 0 to 9. In homage to Kernighan and Ritchie's introduction to coding in the C programming language, most programming courses start by learning how to print "hello world" to the screen. In machine learning, the equivalent to hello-world is learning how to write code to identify handwritten versions of the digits zero to nine. The MNIST (Modified National Institute of Standards and Technology) database consists of 60,000 training images and 10,000 test images.

In 1998, Yann LeCun and colleagues developed a feed-forward network, trained by gradient descent, that could perform this task quite well, achieving an error rate of 7.6%, which was quite remarkable at the time (chance performance would be 90% because there are 10 possible classes). A more recent computational model in 2019 achieves an error rate of 0.21% (that is, about one error in 500 images). This computational model includes a combination of multiple deep convolutional neural networks, a strategy that is common in

computer vision systems: creating many expert systems, and combining their predictions. Recognizing hand-written digits is an example task where computers have reached an accuracy that is comparable to, if not better than, human performance.

VIII.11. A deep convolutional neural network in action

INSERT **Figure VIII-5** AROUND HERE

Figure VIII-5. ***Example outputs from a deep convolutional neural network.*** *A neural network where the input image is passed through three convolutional layers and produces six output probability values (see text for details). fc stands for fully connected layer, batchnorm stands for batch normalization, maxpool stands for max pooling, relu corresponds to a ReLU operation. The size of each convolutional filter is shown on the right. The network is trained to classify images from the six groups shown at the bottom. This figure shows the activation of every unit in the network in response to the image of a sports car shown at the top (see scale bar on the left). Of the six output units (bottom), the "sports car" unit shows the maximum activation; therefore, the network correctly infers the label for this example.*

Next, we illustrate step-by-step how all the outputs of a deep convolutional neural network are generated. We want to show the activation of all the units. Because modern networks typically have a huge number of units, we will consider a simplified network for illustration purposes (**Figure VIII-5**). This network takes as input a color image of size 56x56 pixels x 3 colors. The network has three convolutional layers, and it is trained to classify six categories of images from ImageNet (***Figure VIII-4***): images of biological cells, labradors, fire ants, sports cars, roses, and ice. These six categories were chosen randomly for illustration purposes. The output layer contains six values, one for each category. The network is trained by backpropagation (**Section VIII.6**), using a randomly selected training set. After about 1200 iterations, the network achieves an accuracy of 84% on the training set and an accuracy of 76% on the test set (where chance is 1 out of 6 or 16.7%). As noted when discussing cross-validation (**Section VIII.8**), the difference in performance between training accuracy and test accuracy typically reflects overfitting to the specific example images seen during training.

Figure VIII-5 shows the activation of every unit in the network for one particular image of a sports car. To visualize every unit, activation values were normalized to remain between 0 and 1, and they are shown as a grayscale value (see color scale on the left of the figure). The image (top) consists of three channels: R, G, and B (which appear pretty similar in this example because the image is mostly gray). There are eight filters in the first step (conv1), each with a size of 3x3 pixels (like the filters shown in ***Figure VII.6***). The number of filters in each layer is one of the many architecture decisions that we have to make when building a model; here, the number was arbitrarily set to eight for the first layer for illustration purposes. We can still see a semblance of the input image in the first layer activations, with each of the different filters accentuating certain features.

The convolved image is passed through a batch normalization step, followed by the rectifying linear units (***Figure VII.2***). Batch normalization is a technique that improves the speed, performance, and stability of neural networks by normalizing the inputs to a given layer. Finally, the output is passed through a max-pooling step, which reduces the size from 56x56 to 28x28. The conv1 layer consists of 56 x 56 x 8 = 25,088 units, and the max pool 1 layer consists of 28 x 28 x 8 = 6,272 units.

The second and third layers go through the same steps with 16 and 32 filters, respectively, all of size 3x3 pixels. The activations are shown as an "image" in each square in ***Figure VIII-5*** by putting together all the units at a given step and for a given filter. As we go through the calculations from conv1 to relu3, the resulting "images" look less and less like the original one. The entire purpose of the network is *not* to produce an image that looks like the original one, but rather to extract adequate features that can solve the classification task.

The relu3 values are passed onto a fully connected (fc) layer consisting of six outputs, reflecting the probability that the image label corresponds to each of the six possible categories. For the image in this example, unit 4 in this fc layer shows the maximum activation, which corresponds to sports car. However, other units in this layer still show non-zero probabilities. The resulting values, $z_1, \ldots, z_6$, are passed through a *softmax* function, $\sigma(\boldsymbol{z}) = \frac{e^{z_i}}{\sum_{j=1}^{6} e^{z_j}}$, an operation that converts the values into probabilities that add up to 1, and then through a threshold to create the final winner-take-all value. These final activation values indicate the most likely label. In this case, the network correctly infers that the label for the image is a "Sports car."

INSERT **Figure VIII-6** AROUND HERE

Figure VIII-6. ***Examining the activity of the output layer in a deep convolutional neural network. A****. Rendering of the activity of the six fc units from the model shown in* ***Figure VIII-5*** *in response to every image, after reducing the dimensionality to two axes using a technique called tSNE (see text). Each color connotes a different object category.* ***B****. Activation of each of the six fully connected (fc) units in response to every image. Vertical dashed lines separate object categories; the color accentuates the images from the category that the fc unit is meant to classify.* ***C****. Confusion matrix showing the proportion of images whose actual category label is the one shown in each column and where the model predicted the category shown in each row.* ***D-E****. Even though this network was never trained to recognize human faces or houses, we can still find units that respond differentially to faces versus houses, and we can still use the same network to detect faces or houses. Modified from Kreiman 2018.*

The activation of the fc units after the softmax function can thus be interpreted as the probabilities for each of the six labels. It is difficult to visualize the activation patterns in response to each of the ~8,000 images in this 6-dimensional space. To represent the 8,000 x 6 matrix of activations, there are many dimensionality reduction techniques, including principal component analysis, independent component analysis, multi-dimensional scaling, and others.

In ***Figure VIII-6A***, we used a dimensionality reduction technique with the fancy name of *t-distributed stochastic network embedding*, or tSNE for short. It is beyond the scope of this chapter to describe the mathematics of this technique (see the work of van der Maaten and Hinton, and, of course, there is also a Wikipedia page for it). With tSNE, like other dimensionality reduction techniques, similar images (in the sense of their distance in the 6-dimensional fc space) are represented by nearby points, and different images are represented by points that are farther away. Each point represents one image, and the points are colored according to the actual labels (the coloring is not part of tSNE, which is an entirely unsupervised procedure and does not use any labels). Images with the same label (same color) tend to cluster together. Ultimately, classification accuracy depends on the comparison between the output layer (***Figure VIII-5***) and the ground truth labels, but the separation in this two-dimensional tSNE space provides an intuitive hint that it is possible to adequately separate images with different labels.

Notably, this neural network used here for illustration purposes is relatively simple, in order to show the activation of all the units. A state-of-the-art network would achieve higher accuracy, and the clusters would be much better separated.

One could also use the same dimensionality reduction technique to render the activation of units in each of the other layers. What is particularly interesting about the fc layer is that its activation values are directly correlated with the network's output. In other words, in terms of the behavior of this network, we could refer to the fc unit number four as a "sports car" unit. We can also plot the activation of each of the six fc units for every image (***Figure VIII-6B***). As expected, on average, the "sports car" unit (fc unit 4) shows higher activation for the images with a ground truth label of "Sports car" (highlighted in cyan in ***Figure VIII-6B***). However**,** there are some images containing sports cars that elicit low activation in this unit (for example, the gray arrow on the bottom), and there are images that do not contain Sports cars but still elicit high activation in this unit (for example, the gray arrow on the top). In other words, the "Sports car" fc unit may fail to be activated by many images of sports cars, and it may show high activation by other images that do not contain a Sports car. In **Section IX.9**, we will introduce techniques that can be used to describe what types of images elicit high activation for a unit in a network, and also for neurons in the brain.

The network's output depends on the maximum fc value across all six units. If the ground truth label for a given image is "Sports car," but the activation in the sports car fc4 unit is lower than the activation in another fc unit, then the network will make a mistake by selecting a different label. Conversely, if the ground truth for a given image is not "Sports car," but the fc4 unit shows the maximum activation, then the image will be erroneously labeled sports car. ***Figure VIII-6C*** shows how often these mistakes happen in the form of a confusion matrix. Columns indicate the actual category labels, and the rows indicate the

predicted category labels. All of the entries along the diagonal are correct responses. For example, 88% of the time, when the image contained a Sports car, the network correctly labeled it Sports car (row 4, column 4 in the matrix). Sometimes, the image contained a Sports car, and yet the network labeled it as "ice"; this happened for 5% of the Sports car images in the test set (row 6, column 4 in the matrix). Other times, the image depicted ice, but the network labeled it as "sports car"; this happened for 4% of the ice images (row 4, column 6 in the matrix).

The six object categories used in ***Figure VIII-5*** and ***Figure VIII-6*** were chosen randomly. The fc units are also activated by images that the network has *not* been trained on. One of the advantages of working with image computable models is that we can directly quantify the response of the network to *any* image. For example, we can ask whether the same fc units, in the same network, would be activated by images of houses or human faces. Importantly, we are not retraining the network with these new images. The weights are fixed and we merely monitor the activation of the fc units. The network has never seen a house or a face before; however, the units in the network are still activated by those images (in the same way that neurons in our visual cortex would be activated if we were shown a unicorn, even if we have never seen one before).

Again, we use tSNE to render the activation of all six fc units to all the images of houses and faces (***Figure VIII-6D***). Even though the network was never trained with those images, it can still separate them quite well: the network achieved an accuracy of 86% in distinguishing faces from houses (where chance is 50%). The fc unit that showed the most distinct separation between faces and houses was the "sports car" unit (fc unit 4, ***Figure VIII-6E***). This unit showed a stronger activation to faces (0.47 ± 1.72), compared to houses (-1.54 ± 1.18). If an investigator were to conduct a study with this network, and only showed images of faces and houses, the investigator would probably call this unit a "face" unit. Yet, the activation of this fc unit to sports cars was 4.59 ± 2.27. Thus, merely showing a set of random images is not sufficient to interpret the activation of units in a network (see also the discussion in **Section VI.4** and **Section IX.9**).

The exercise of evaluating the activity of the network for images that it was never exposed to before also helps us make another point. Without re-training, the network can solve visual classification problems that it was not trained on. The dictionary of features and computations learned by the network while trying to separate six arbitrary random object classes is sufficiently rich to be able to distinguish other image categories. One could even go on and, starting with the network pre-trained to distinguish among these six categories, re-train the network for a new task. Such re-training is another example of transfer learning, introduced in **Section VIII.6**: training in one task first, and then using the pre-trained network as an initial condition to learn a new task.

VIII.12. To err is human and algorithmic

INSERT **Figure VIII-7** AROUND HERE
Figure VIII-7. ***Confusion matrix for the MNIST dataset.*** *The values highlighted in gray in entry (i,j) indicate the percentage of MNIST test images that belong to category label j and were classified as category label i by a two-layer neural network. The diagonal entries correspond to correct classifications, and off-diagonal entries are mistakes. Examples of mistakes are shown for each possible combination. The boxes highlight the most likely type of mistake for each number.*

In the type of visual classification problems that we have been discussing, the ground truth is set by humans. Let us go back to the MNIST dataset of handwritten digits; those digits have labels assigned by humans. Only by comparison to human behavior, we can define whether a computational model of vision makes mistakes or not.

In ***Figure VIII-6B-C***, we showed how computational models make mistakes, again by referring to labels provided by humans. Similarly, ***Figure VIII-7*** shows a confusion matrix comparing the outputs of a two-layer neural network and humans for MNIST. The overall performance of this network was 95%. Better algorithms certainly abound, but here we deliberately use this two-layer network to illustrate the mistakes made by an algorithm. All of the percentages along the diagonal correspond to cases where the neural network correctly classified the images. Not all digits were equally well classified. The accuracy for number seven was 89.3%, that is, it was easier to confuse number seven with other numbers, perhaps reflecting the heterogeneity in how people draw sevens. Number “1” had the highest accuracy at 98.9%. In the test set of 10,000 images used in ***Figure VIII-7***, some mistakes never happened; for example, the network never mistook a “1” for a “0” or a “0” for a “1.” The most confusable digits in every column are highlighted in red. The worst case was number “7” being confused by number “9,” which happened 5.1% of the cases when a “7” was presented.

Staring at some of the example mistakes in ***Figure VIII-7***, it is perhaps intuitive to appreciate how the model may have misinterpreted some of those digits. For example, number six in the bottom row does look like a zero. To err is not only algorithmic but human as well. What would it mean in this context for humans to make mistakes? We could consider a behavioral experiment where a set of subjects is asked to classify those images. Of course, this set of subjects should be independent of the original set of subjects who established the ground truth labels in the first place. In this manner, we can assess the degree of between-subject variability in visual recognition. In the same fashion, we can also compare performance between humans and other species. For example, a monkey can be trained to behaviorally discriminate a set of images, and then we can directly compare the human and monkey labels on an image by image basis.

INSERT **Figure VIII-8** AROUND HERE
Figure VIII-8. ***Of humans, monkeys, and computers.*** *Comparison of classification performance between different computational models (blue) and humans, and between*

*monkeys (gray dot) and humans. The degree of consistency (y-axis) shows the correlation in performance at the level of object categories (**A**) or individual images (**B**); see text for details. Modified from Rajmalingham et al. 2018.*

Rajmalingham and colleagues followed this path to compare visual recognition performance between monkeys, humans, and computational models (***Figure VIII-8***). The authors considered 24 objects, including "elephant," "shorts," "wrench," and others. Rotated and scaled photographs of these objects were pasted onto natural images. Human or monkey subjects were shown a test image for 100 ms, followed by a choice screen containing a canonical rendering of the test object and a canonical rendering of an object from one of the other 23 objects. Subjects had to indicate which of the two matched the test image. Based on these behavioral measurements, the authors computed the discriminability of each object versus every other object (object-level comparisons, ***Figure VIII-8A***), averaging across all images (all rotations and backgrounds). They also computed the discriminability of every image against every other image where an image is a particular combination of an object, rotation, and background (image-level comparisons, ***Figure VIII-8B***). Human subjects were quite consistent with each other, as evaluated by excluding some of them and comparing their performance with the rest (black dot in ***Figure VIII-8A***, consistency > 0.9). There is more variability at the level of individual images, as demonstrated by the ~0.8 degree of consistency between human subjects in ***Figure VIII-8B*****).** Monkeys also thrived in this task and showed ~0.8 consistency with humans in both ***Figure VIII-8A-B*** (gray dots).

Next, the authors considered several computational models tested on precisely the same images. They considered six deep convolutional neural networks: AlexNet, NYU, VGG, GoogleNet, ResNet, and Inception-v3. Computer vision scientists have incorporated the fervor of biologists about naming their models. The names in ***Figure VIII-8*** correspond to several popular computer vision models, and we will not get into the details of their different architectures (all of these architectures are expansions and variations of the one shown in ***Figure VIII-2***). Suffice to say that all of these models contain anywhere from 7 layers (AlexNet) to more than 150 layers (ResNet), that they were trained on the ImageNet dataset (***Figure VIII-4***), and that they have been successful in computer vision recognition challenges. The authors also considered a model that used only pixel-level information and a model that aimed to mimic the computations performed in primary visual cortex (V1). Showing performance metrics for pixels is always a simple low-level benchmark to include in these comparisons: after all, if one can solve a given problem using the pixel values, why bother with more complex models?

At the object level (***Figure VIII-8A***), all the deep convolutional neural network models (but not the pixel and V1 models) showed a remarkable degree of consistency with human behavior. These models were slightly more similar to humans than monkeys were, and slightly less similar than accounted for by between-subject variability. In contrast, for the image level comparisons (***Figure***

VIII-8B), even though all of the deep models better accounted for human behavior than pixel-based or V1-based models, they all fell short in accounting for human behavior. Both humans and models can perform quite well, but not perfectly, in this task. The pattern of mistakes of humans and models is still distinct when considering each image separately. When considering all the variability in object positions, sizes, and rotations, there was still more consistency between different human subjects, or between monkeys and humans, than between any of the models and humans.

VIII.13. Predicting eye movements

We can further constrain computational models by going beyond assessing whether we can match the pattern of mistakes in image classification tasks. One prominent aspect of human visual behavior is the rapid sequence of eye movements that take place about three times per second under normal viewing conditions (**Chapter II**). The types of deep neural network models that we have described so far do not have anything akin to eye movements, yet we can modify the models to predict what aspects of an image are salient and may drive eye movements.

One of the most salient aspects of an image is sudden motion changes. If a person in our field of view starts running, our eyes will be drawn to that person. While such temporal changes provide strong saliency cues, spatial changes in the image, including contrast, color changes, and texture changes, also attract shifts in spatial attention via eye movements. For example, if we are looking at an image where everything is gray except for a yellow car, that car will be very salient. These notions of saliency have been extensively studied in the psychophysics literature. In many cases, these principles were rediscovered by people making movies and also in the advertisement industry.

Task goals also influence eye movements. For example, if we are looking for our car in the parking lot, our eyes may be drawn to locations where there are cars as opposed to buildings or the sky, especially to other cars that share the same color and shape. We may even disregard other strong saliency cues like movement (in all likelihood, our car is not moving). The target features take over and have a prominent role in dictating our spatiotemporal sequence of eye movements.

Let us consider an example visual search task (***Figure VIII-9A***). Subjects are presented with a target image containing an object to look for (in this example, a horse). Next, subjects are presented with a search image containing six objects presented around the fixation point. One of these six objects is a horse, and the other five are distractors. In principle, one could solve the search problem by simple template matching, by exhaustively moving a template of the same size as the target object throughout the entire image until a perfect match is found. To avoid this solution, the objects in the search image are shown at a

different scale, and they are also randomly rotated. Furthermore, the target horse is actually a different exemplar from the same object category (that is, a different horse in this case).

We are interested in a visual search algorithm that can find the target object efficiently (that is, without exhaustively scanning the entire image), selectively (to differentiate the target object from the distractors), and in an invariant fashion with respect to changes in the target object's scale, rotation, and even different exemplars. Furthermore, we also want to test whether the algorithm can capture fundamental aspects of how humans move their eyes to solve the search problem.

INSERT **Figure VIII-9** AROUND HERE

Figure VIII-9. ***A neural network that predicts eye movements during visual search.*** ***A****. Visual search task where subjects or the model need to move their eyes to find a target object (left) in an image.* ***B****. Schematic illustration of the Invariant Visual Search Network (IVSN) model (see text for details).* ***C****. Cumulative performance of humans (red) and IVSN (red) in the task (dashed line indicates chance). Adapted from Zhang et al. 2018.*

Figure VIII-9B shows a schematic of such a computational model. At the heart of the model is a deep convolutional neural network that extracts visual features from the target and search images, allegorically referred to here as "ventral visual cortex." In this case, the authors used the "VGG" neural network, which is one of the networks also tested in ***Figure VIII-8***. This convolutional neural network was pre-trained using ImageNet (***Figure VIII-4***) so that it would have an extensive dictionary of visual features from natural images.

The model needs to keep in memory the information about the target object to be able to look for it in the search image. We also need to decide what aspects of the target image the model should keep in memory. Should the model store all the features in every layer of the network? Keeping all the features would correspond to an entire multi-level representation of the target object in terms of the layer 1 responses, the layer 2 responses, and so on. Alternatively, we could keep only layer 1 responses. The problem with keeping exclusively layer 1 features is that those low-level features are too sensitive to the metric properties of the image, and are not ideal for searching for objects that have been scaled and rotated. At the other extreme, we could keep only the top layer responses; this is the approach illustrated in ***Figure VIII-9B***. As a simplification, the model perfectly and indefinitely stores all the features in the top layer of the ventral visual cortex model. In reality, this type of memory, often referred to as working memory, decays rapidly over a few seconds. This part of the model is referred to as "pre-frontal cortex" because investigators have found neurons in pre-frontal cortex that play an important role in storing task-dependent information in working memory tasks.

Information about the target object is used to modulate the activations of the model in response to the search image. This *top-down* modulation takes place in parallel throughout the entire image. The result is an activation map that essentially describes how similar each part of the image is to the target, where similarity is defined by the high-level features stored in pre-frontal cortex, and where the spatial resolution depends on which level of the hierarchy is modulated. This activation pattern is referred to as an *attentional map*. In this example, this map has a resolution of 16x16 regions. A winner-take-all (WTA) mechanism selects the maximum of the attentional map. This location corresponding to the maximum in the attentional map becomes the position of the model's first "fixation."

If the target is found at this location, the search ends. If the target is not found, the model goes back to the attentional map. Because the model is deterministic, if we select the maximum again, the model would always keep fixating on the same location. To avoid this problem, the model uses an infinite inhibition of return (IOR) mechanism, meaning that it never goes back to fixate on a location that it has already selected before. The winter-take-all mechanism selects the next maximum in the attentional map for fixation. Thus, by adding a few computational steps, we can use a deep convolutional neural network to make a sequence of eye movements, and detect the location of target objects in a search image.

Does this work? First, let us examine human behavior in this task (***Figure VIII-9C***). Because there are six objects (one target and five distractors), by chance, there is a probability of 1/6 of finding the target in the first fixation, 2/6 of finding the target by the second fixation, and so on. Humans do much better than chance. They are not able to find the target in just one fixation, but they can do so with a probability of approximately 1/3. These numbers are not too critical; the exact probabilities likely depend on multiple factors such as how large the objects are, how far they are from the fixation point, how similar the distractors are to the target, how many distractors there are, and how different the target in the search image is from the one in the target image. Regardless of the quantitative numbers, humans can efficiently find the target objects. Intriguingly, humans are slightly below 100% performance, which is below chance, at six fixations because humans do not have infinite inhibition of return, as the model assumes. Humans are stubborn creatures, and sometimes they move their eyes back to the same location even when that location does not contain the target.

Next, let us examine the model depicted in ***Figure VIII-9***, tested on the same images and task used in the human psychophysics experiment. The model does surprisingly well: it can localize target objects more efficiently than sequentially or randomly scanning the entire image. Both humans and the model can find the target in a manner that shows invariance to some target features given the experimental design choice of using different exemplars from the same categories and scaling and rotating the objects. Of note, this model had never

seen these specific objects before, and therefore it was able to find objects even without any kind of object-specific training. The strong similarity between human and model behavior is partly coincidental. Experiments with other images, including searching for objects in natural images or the famous example of searching for Waldo, show that the model does not entirely account for human eye movement behavior. It should be noted that the model was not trained to match human behavior; i.e., there is no data fitting in this procedure. A model that was trained for object classification can be adapted for the task of visual search and explain human eye movements, without tuning any parameters.

VIII.14. Predicting neuronal firing rates

The previous two sections demonstrate that deep convolutional neural network models that are trained for object classification tasks can provide a reasonable first-order approximation to human and monkey visual behavior, both in terms of tasks like object categorization, but also in terms of other tasks such as making eye movements during visual search. Nonetheless, current models do not provide a perfect description of human behavior. We pointed out above several cases where models may qualitatively capture certain aspects of visual behavior (e.g., object-level classification performance, eye movements during visual search in object array images). However, in other aspects, there is ample room for improvement (e.g., image-level object classification performance, eye movements during visual search in natural images). We will come back to several astonishing failures of current models in **Chapter IX**.

We now turn our attention to what happens inside the brain, and we ask whether current models can capture the internal mechanisms of visual function. Even if we had a model that explained visual behavior exceptionally well, this would not necessarily imply that the inner workings of brains and the model are the same. Brains and models could be solving the same problem in entirely different ways. Understanding the differences between brain mechanisms and computational mechanisms can inspire the development of better models.

INSERT **Figure VIII-10** AROUND HERE

Figure VIII-10. ***Computational models can approximate neuronal firing rates. A****. Example images shown to a monkey while recording the activity from two different sites (site 150 and site 56) in inferior temporal cortex (ITC).* ***B****. Neuronal responses (black) versus predicted responses from a deep convolutional neural network model (red). Each entry along the x-axis corresponds to one out of 1,600 different images divided into eight object categories (animals, boats, and others). Modified from Yamins et al. 2014.*

The question of whether the inner workings of a brain and the model are similar or not requires further specification. If we go down to the level of individual molecules, the hardware is very different. To assess whether a model captures aspects of neural circuit function, and therefore whether a model can help us better decipher neural mechanisms, we need to define which aspects of neural function we want to explain. A natural question is to try to explain the firing rate

properties of neurons. As discussed in **Chapters II, V**, and **VI**, neuronal spikes constitute the gold standard to study neural function and represent the main mechanism by which neurons can send signals over long distances. We therefore consider whether a model can predict the number of spikes emitted by a neuron in response to a given image.

An example of this type of analysis is shown in ***Figure VIII-10***. Investigators presented an extensive collection of images to a monkey while recording the activity of neurons in inferior temporal cortex (ITC). Similar to the study described in ***Figure VIII-8***, the monkeys were presented with images of animals, boats, faces, and five other object categories. The objects were pasted in front of natural backgrounds (example images are shown in ***Figure VIII-10A***). As described in **Chapter VI**, ITC neurons showed selective responses to different types of images. For example, the recording site in ***Figure VIII-10A1/B1*** (black trace) demonstrates generally higher responses to images containing chairs, and to a lesser degree planes (where "responses" are defined here as the total number of spikes in a fixed window from 70 to 170 ms after stimulus onset). As discussed in **Chapters II** and **V**, neuronal responses can be variable upon repeated presentation of the same stimulus; the neuronal responses shown in ***Figure VIII-10B*** represent averages over tens of repetitions of the same stimulus.

The same images can be passed to a deep convolutional neural network, extracting the activation values in each layer, as shown in ***Figure VIII-5***. Next, we can compare those model activations to the neuronal responses. One way to make this comparison is to take the activation values in a given layer and build a linear map onto the responses of a given neuron. This linear fitting procedure has one free parameter per unit in the neural network, and the number of equations is the number of images. Some of the images are used to fit the linear map, and the rest is used to test how well the model can approximate neuronal responses to novel images (***Figure VIII-10A1/B1*** red trace). The correlation between the predicted responses and the actual neuronal responses is better when using model activations from higher layers than in the early layers of deep convolutional networks, suggesting that more complex features may be required to explain the activity of neurons in ITC. These models can typically account for more than 50% of the variance in the neuronal responses. Thus, despite the fact that neural network models constitute a far cry from the intricate complexities of biological tissue, the unit activations provide a good initial approximation to predict neuronal responses.

VIII.15. All models are wrong, some are useful

State-of-the-art deep convolutional neural networks are appealing because they fulfill many of the desiderata articulated in **Section VIII.1**. Furthermore, we will show in **Chapter IX** that these models have been successful in a wide range of visual processing problems in the real world. As discussed in the previous

three sections, these models also provide an imperfect but reasonable first order-approximation to visual behavior and visual neurophysiology.

It is intriguing that deep convolutional neural networks can capture aspects of behavior and physiology, given that they abstract away so much of the underlying neuronal circuitry. As discussed in **Sections VII.3** and **VII.4**, an educated guess about the right level of abstraction in modeling neural circuits is essential for progress. Biologists examining a deep convolutional neural network are appalled at the lack of a myriad of actual elements present in nervous tissue. To mention only a few, from larger scales down to smaller scales, the visual system is characterized by a mesmerizingly complex array of interconnections (***Figure I-5***), most of which are not present in current models. We also know that there are many different neuronal types, including at least tens, if not hundreds, of different types of interneurons in the brain, whereas current models have essentially only one or a handful of different types of computational units depending on how we count. Neurons are characterized by complex geometries, and the spatiotemporal distribution of inputs to different dendrites can have a significant impact on the biophysics of single-neuron computations. Biochemists may even wonder about the intricate expression patterns of the approximately twenty thousand genes in the human genome in different types of neurons.

While biologists worry about abstracting away these and many other aspects of the neural circuitry, at the other end of the scientific spectrum, psychologists worry that there is too little abstraction in current models. Psychologists are appalled at the lack of a myriad of conceptual structures. To mention only a few in increasingly more ethereal levels, these models do not have a clear sense of semantic knowledge (see discussion in **Section VI.8**), beyond what is imposed by the labels used during training. Additionally, common-sense intuitions about the visual world, including the notion of "objectness" or the notion of agents that interact with each other, are not explicitly incorporated into these models. Psychologists argue based on introspection that these concepts are critical to interpreting the visual world. Some psychologists further argue that we cannot understand visual processing isolated from how we interact with the world, and that vision cannot be dissociated from language.

The discussion of the biological and psychological components that are missing in current models can be approximately mapped onto David Marr's three levels of analysis (**Section I.9**). Psychologists tend to think about the high-level computations that the system may want to implement, and biologists tend to think about the hardware required to perform all of these computations. An essential goal of models is to bridge these levels of analysis by building algorithms that can implement those computations, and by linking those algorithms to the actual biological hardware. Inputs from both biologists and psychologists will be invaluable in further improving current computational models.

VIII.16. Horizontal and top-down signals in visual recognition

One of the several simplifications in deep convolutional neural networks that deserves further scrutiny is the lack of horizontal and top-down signals. We know that there are abundant horizontal and backprojections throughout neocortex. The functions of top-down connections have been less studied at the neurophysiological level, but there is no shortage of computational models illustrating the rich array of computations that could emerge with such connectivity.

Essentially all computational models consider that top-down signals play a critical role during learning. In fact, the procedure of back-propagation described in **Section 6** requires a top-down propagation of errors throughout the network. However, purely bottom-up models do not capitalize on top-down signals after learning and *during* visual processing.

Several models have used top-down connections to guide attention to specific locations or features within the image (**Section V.17**). The allocation of attention to specific parts of an image can significantly enhance recognition performance by alleviating the problems associated with image segmentation and clutter. The model introduced in ***Figure VIII-9*** uses top-down signals to guide eye movements, that is, overt attention, towards specific locations that may contain the sought object.

Horizontal and top-down signals can also play an important role in recognition of occluded objects. When only partial object information is available, the visual system must be able to perform pattern completion and interpret the image based on prior knowledge (**Section III.5**). Attractor-based recurrent networks can retrieve the identity of stored memories from partial information (***Section VII.9***). Similarly, computational models have combined bottom-up architectures with attractor networks at the top of the hierarchy. The attractor-based recurrent dynamics can help make inferences from partial information and thus recognize heavily occluded objects. In addition to horizontal connections, top-down signals could also play an important role during pattern completion by providing prior stored information that influences the bottom-up sensory responses.

The idea that top-down signals can carry task-relevant prior information has been embraced by several proposals formulating visual recognition as a Bayesian inference problem. Considering three layers of the visual cascade (e.g., LGN, V1, and higher areas like V2), and denoting activity in those three layers as $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, and $\boldsymbol{x}_h$, respectively, the probability of obtaining a given response pattern in V1 ($\boldsymbol{x}_1$) depends both on the sensory input as well as feedback from higher areas:

$$P(\mathbf{x}_1|\mathbf{x}_0) = \frac{P(\mathbf{x}_0|\mathbf{x}_1)P(\mathbf{x}_1|\mathbf{x}_h)}{P(\mathbf{x}_0|\mathbf{x}_h)} \qquad \textbf{Equation VII.12}$$

where $P(\mathbf{x}_1 | \mathbf{x}_h)$ represents the feedback biases conveying prior information.

VIII.17. Predictive coding

INSERT **Figure VIII-11** AROUND HERE

Figure VIII-11. ***PredNet, a deep predictive coding architecture. A****. Schematic illustration of two layers of the PredNet architecture. R units send predictive feedback signals to the previous layer. The bottom-up inputs pass a difference between the predicted signals and the signals from the previous step (see text for details).* ***B****. Surround suppression in a monkey V1 neuron (**B1**) and a PredNet layer 1 unit (**B2**). Responses to an optimal bar of increasing lengths in the original conditions (red) or in the absence of feedback (blue).* ***C****. Sequence prediction by monkey IT neurons (**C1**) or PredNet layer 3 unit (**C2**) when the second stimulus is predictable (blue) or unpredictable (red) (**C3**). Modified from Lotter et al. 2018.*

An interesting version of how top-down signals could be used during visual recognition was proposed by Rajesh Rao and Dana Ballard, who argued that feedback connections provide *predictive* signals, whereas bottom-up signals convey the difference between sensory inputs and the top-down predictions. For example, consider the phenomenon of surround suppression (**Section V.5**): if we present an optimally-oriented sinusoidal grating within the receptive field of a V1 neuron, the firing rate increases with the size of the stimulus up to a certain point; when the stimulus size exceeds the receptive field size, the firing starts to diminish with increasing size. According to the predictive coding model, neurons in higher areas with larger receptive field sizes (e.g., V2) send a feedback signal that can predict the responses, and the V1 neuron subtracts those predictions from the sensory inputs, thereby leading to a smaller response for large stimulus sizes. Indeed, silencing activity of V2 neurons leads to a reduced surround suppression effect, that is, to stronger responses in V1 to larger stimuli.

Predictions can take place not only in the spatial domain but also in the temporal domain. A constant visual stimulus (in the absence of any external changes or any internal changes like head or eye movements) can be predicted. According to the predictive coding model, the feedback signals lead to a reduction of the initial transient response. Indeed, transient responses to constant stimuli are the norm throughout the visual system (**Section II.9**).

Such predictive coding ideas can be extended to a multi-layer network. The model architecture shown in ***Figure VIII-11A*** consists of multiple layers (only two of which are shown in the schematic); each layer is composed of four types of units: input units (blue, A_l), recurrent representation units (green, R_l), error representation units (red, E_l), and prediction units (blue, $\hat{A}_l$). If we remove, or silence, the recurrent units, then the pathway from A_l to E_l to A_{l+1} to E_{l+1}, is a standard deep convolutional neural network. The recurrent units provide top-down signals. If we go from the higher layers down to the lower layers, we generate a progressively larger representation, which can be thought of as a generative deconvolutional network, similar to other algorithms to generate

images which are discussed in the next chapter (**Section IX.8** and **IX.9**). The investigators refer to this network as *PredNet*.

In this network, the error units pass the difference between the predictions and the inputs to the next layer. The recurrent units take as input both the error in the current layer and the top-down activity from the next layer. In contrast to standard deep convolutional neural networks like the one in **Figure VIII-2**, here the network shows rich dynamics: the activation of every unit evolves over time.

Let x_t represent the input frame at time t. For the first layer, ${}_tA_l = x_t$. For the next layers, the input units compute a convolution (plus rectification and pooling) over the activation of the error units in the previous layer:

$$A_l^t = \text{MAXPOOL}\left(ReLU\left(CONV(E_{l-1}^t)\right)\right) \qquad \textbf{Equation VIII.13}$$

The recurrent units combine the top-down signals from the recurrent units in the upper layer and the propagated errors in the current layer. The inputs from the upper layer need to be upsampled because of the pooling operation from one layer to the next. The recurrent computations within the layer imply that there are horizontal connections that link the R_l units. We discussed a model with such horizontal connections, the Hopfield network, in **Section VII.9**. Many current models use a different implementation of a recurrent network known as a long short term memory (LSTM) module, a special type of recurrent module that is well suited to learn long-term dependencies in the data. We can schematically describe the activation of recurrent units as:

$$R_l^t = CONVLSTM\left(E_l^{t-1}, R_l^{t-1}, UPSAMPLE(R_{l+1}^t,)\right) \qquad \textbf{Equation VIII.14}$$

The predictions are directly computed from the recurrent unit activations:

$$\hat{A}_l^t = ReLU\left(CONV(R_l^t)\right) \qquad \textbf{Equation VIII.15}$$

The error units signal the difference between inputs and predictions, with both possible signs:

$$E_l^t = \left[RELU\left(A_l^t - \hat{A}_l^t\right), RELU\left(\hat{A}_l^t - A_l^t\right)\right] \qquad \textbf{Equation VIII.16}$$

This network can be trained in an end-to-end fashion. The deep convolutional neural network architecture discussed in ***Figure VIII-5*** was trained to perform object recognition. In contrast, the PredNet model in ***Figure VIII-11*** was trained to predict the next frame in video sequences. The investigators trained the network using videos extracted from a camera mounted on a car and tuned the network to predict the next frame. The loss function was based on minimizing the difference between a predicted frame in the video and the actual frame. The loss function could be based on consecutive frames or with a short interval

between the two frames. Of note, the training procedure is similar to the backpropagation formalism described in **Section VIII.6**, even though the type of loss function is different.

Whereas in the previous examples, the loss function was given by a difference between a target label and a predicted label for an image, here the network is *not* trained using any explicit labels, or even any explicit notion of objects. The type of training procedure where there is a loss function that is directly embedded in the input sequence without any need for external annotation is referred to as *self-supervised learning*. Some people also refer to this scenario as *unsupervised learning*, but it is preferable to use unsupervised to a situation where there is no loss function for every image, video, or trial; such as when clustering data, when applying tSNE, Hebbian learning, spike-timing dependent plasticity, or similar mechanisms.

What can a network like PredNet do? To begin with, the model can predict the next frame in video sequences. After all, this is what the model was trained for. The model can achieve these video predictions even in other videos different from the ones that it was trained on. Furthermore, the unit activations can be used to classify objects. Even though the network is not explicitly trained for object classification, it develops a sufficiently rich set of features that relate to natural images, and a linear classifier can be used to assign labels to objects using this feature set.

We can therefore evaluate the network using all the same tests described earlier for bottom-up neural networks, including object classification performance, but also comparing their output with behavioral and neural data. For example, units in the network show surround suppression (***Figure VIII-2B***, red curve). Like monkey V1 neurons (***Figure VIII-2B1***), units in the first layer show larger activation for longer bars up to a point, and then the activation decreases with longer bars. In large part, surround suppression is due to top-down signals, as demonstrated by silencing the R units in the network (***Figure VIII-2B***, blue curve).

Surround suppression can be thought of as a form of spatial prediction. The model can also make temporal predictions, such as inferring the next frame in a video sequence. One type of paradigm that has been used extensively in Neuroscience is a sequence learning task where animals learn that a given stimulus B typically follows a stimulus A (***Figure VIII-2C3-C4***). Monkeys can be trained to learn this type of temporal contingency, and neurons in ITC show a lower response to a predictable second stimulus compared to a new, surprising, second stimulus (***Figure VIII-2C1***, cf. response to the predicted B (blue) versus the unpredicted B (red)). Error units in layer 3 in PredNet also show this type of novelty detection and display a lower activation when the second stimulus is expected (***Figure VIII-2C2***).

Even though the PredNet model was never trained to label objects, or to show surround suppression, or perform novelty detection, these and other biological properties emerge in this type of network when it is trained to make predictions in video sequences. This emergence of unrelated properties is particularly exciting because it suggests that fundamental aspects of the visual system architecture can develop through experience with the natural statistics of the world, without the need to train the model with millions of labeled examples in a supervised fashion. In sum, it is possible to build biologically plausible neural architectures that learn to extract fundamental structure in the world in a self-supervised manner. Several biological properties emerge naturally in these networks, by training those them with basic principles like predictive coding.

VIII.18. Summary

- Biologically plausible computational models of visual processing should be image computable, based on neural network architectures, and display the fundamental properties of selectivity, invariance, speed, and generalization.
- State-of-the-art vision models are based on a divide-and-conquer hierarchical architecture composed of layers that sequentially process information.
- Ascending through the hierarchy, units show larger receptive field sizes, display preferences for more complex features, and show increasingly more tolerance to metric transformations of those features.
- Deep convolutional neural networks are trained end-to-end so that all weights in the network are modified according to a predefined loss function and without the need for manual tuning of model parameters.
- One of the main ways of learning the weights in deep convolutional neural networks is by using gradient descent implemented by the backpropagation algorithm
- Large datasets such as ImageNet have allowed extensive training of deep convolutional neural networks via supervised learning.
- Modern networks have an enormous number of tunable parameters, raising the question of how they can generalize and avoid overfitting.
- Cross-validation is an essential step to avoid obtaining inflated performance values that do not extrapolate to novel data.
- Once trained, the responses of units in the network to any arbitrary image can be readily computed. These responses can be directly compared to behavioral and neurophysiological measurements.
- Deep convolutional neural networks provide a first-order approximation to primate behavioral performance, and the networks can also approximate the pattern of mistakes in visual recognition tasks and the pattern of eye movements during visual search.
- The activation of units in the network can also be used to approximately predict the responses of biological neurons throughout the ventral visual cortex upon presentation of visual stimuli.

- Current models lack many low-level biological mechanisms and also many high-level psychological intuitions.
- Top-down signals are essential to bridge sensory inputs to memories and previous knowledge about the world.
- Top-down signals are essential during learning (an example of which is backpropagation).
- Top-down signals also play an important role during visual processing by merging bottom-up sensory signals with predictive signals based on higher knowledge.

VIII.19. Further reading

- Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. PNAS 79:2554-2558.
- Marr D (1982) Vision. San Francisco: Freeman publishers.
- Rao RP, Ballard DH (1999) Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. Nature Neuroscience 2:79-87.
- Riesenhuber M, Poggio T (1999) Hierarchical models of object recognition in cortex. Nature Neuroscience 2:1019-1025.
- Yamins DL, Hong H, Cadieu CF, Solomon EA, Seibert D, DiCarlo JJ (2014) Performance-optimized hierarchical models predict neural responses in higher visual cortex. Proceedings of the National Academy of Sciences of the United States of America 111:8619-8624.

Figure VIII-1

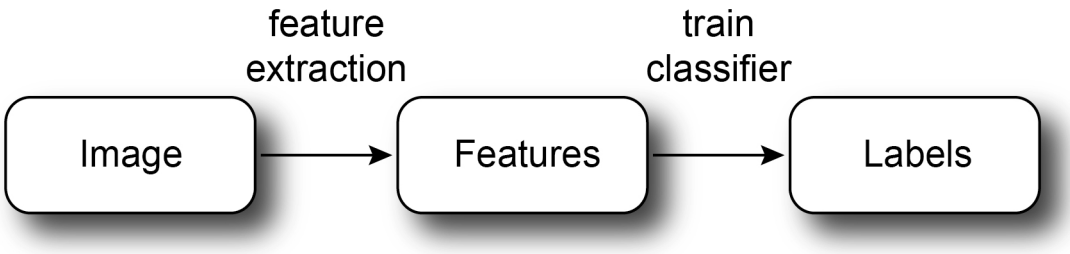


Figure VIII-2

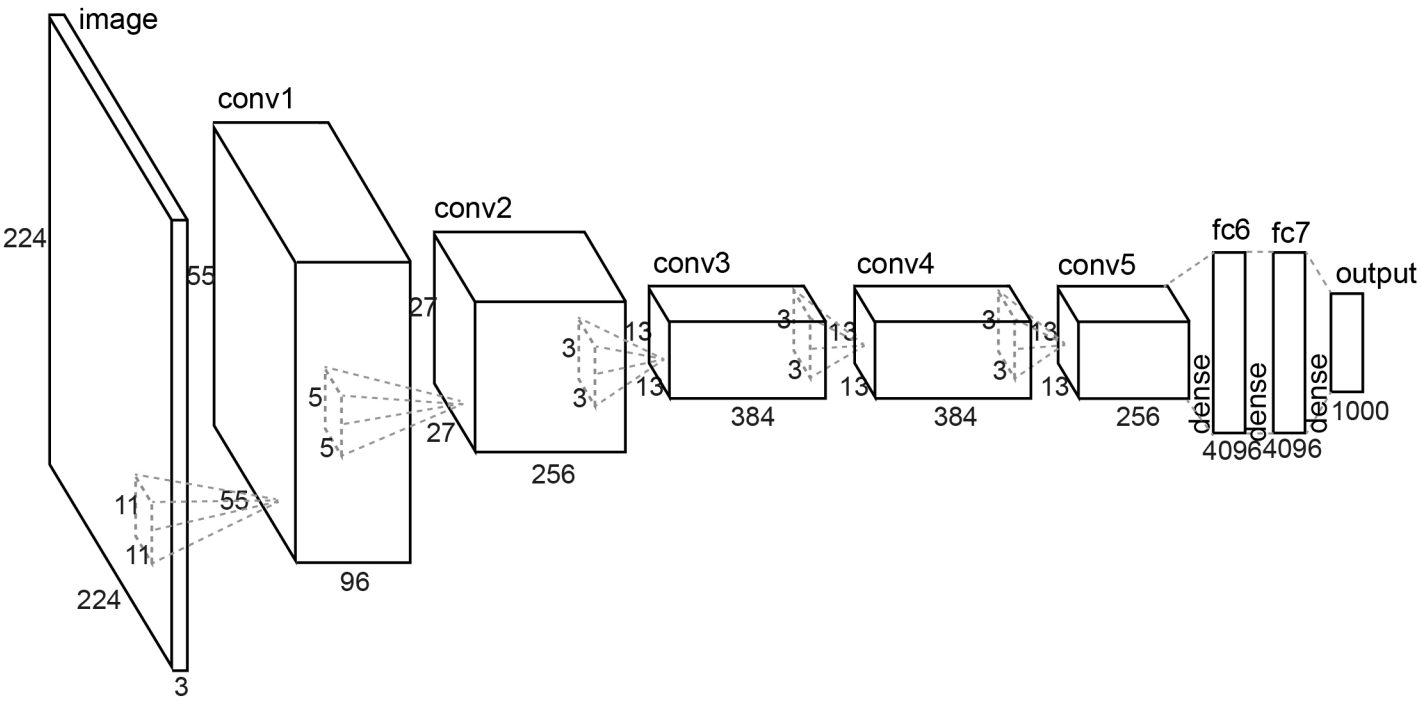


Figure VIII-3

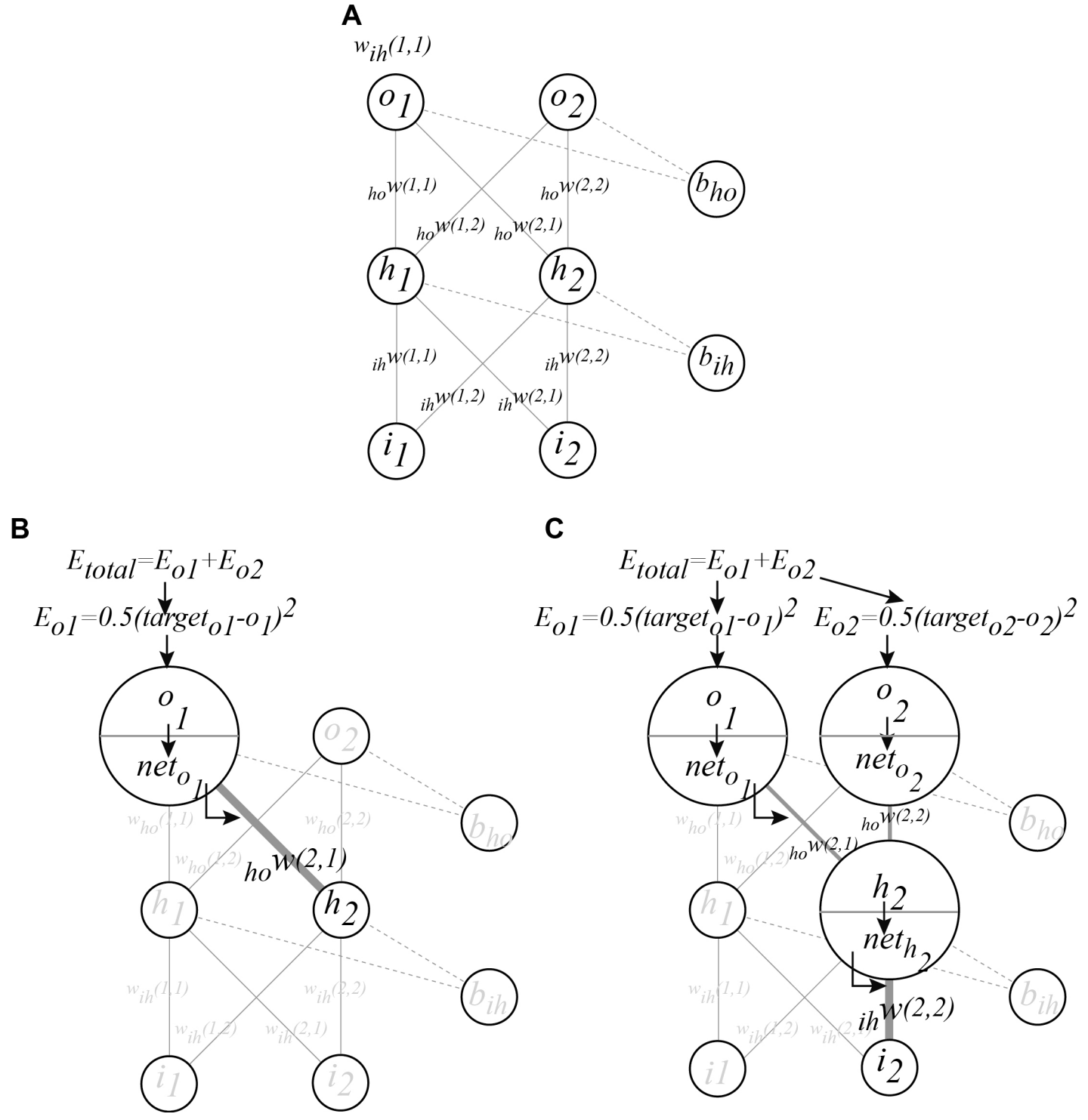
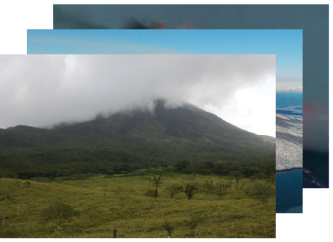


Figure VIII-4

980: volcano



344: hippopotamus

538: dome

380: African elephant

837: sunglasses

340: zebra

372: baboon

483: castle

Figure VIII-5

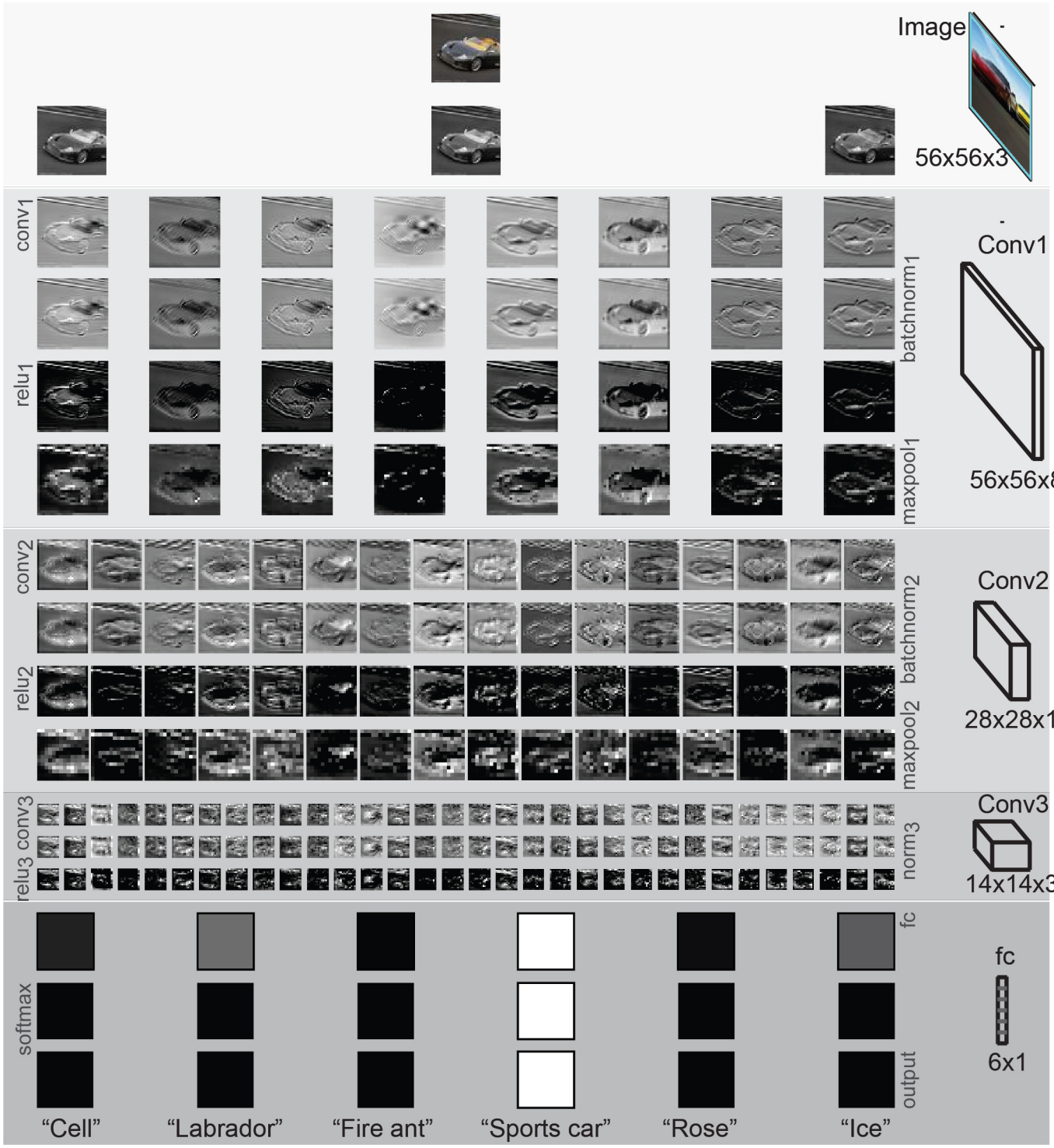


Figure VIII-6

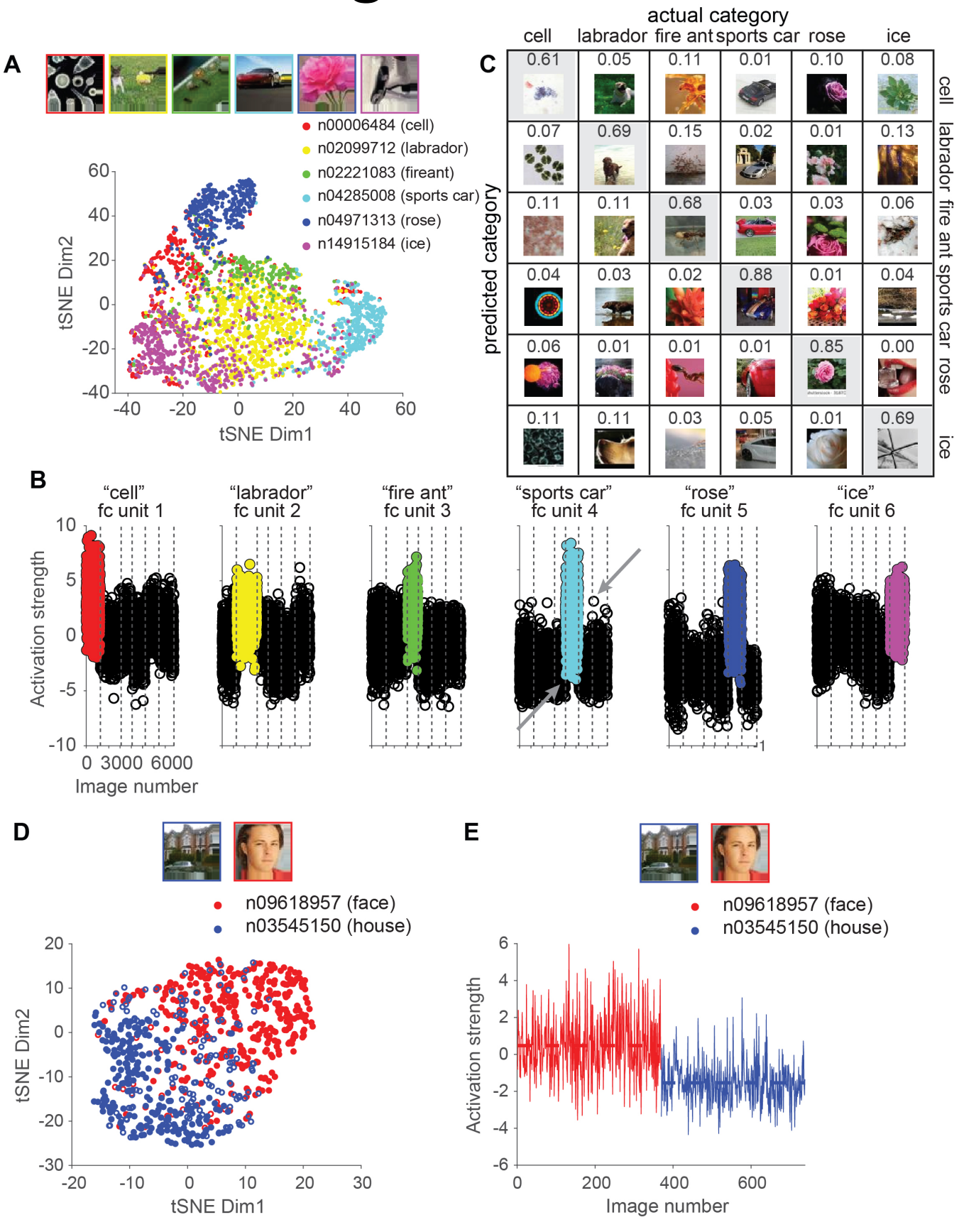


Figure VIII-7

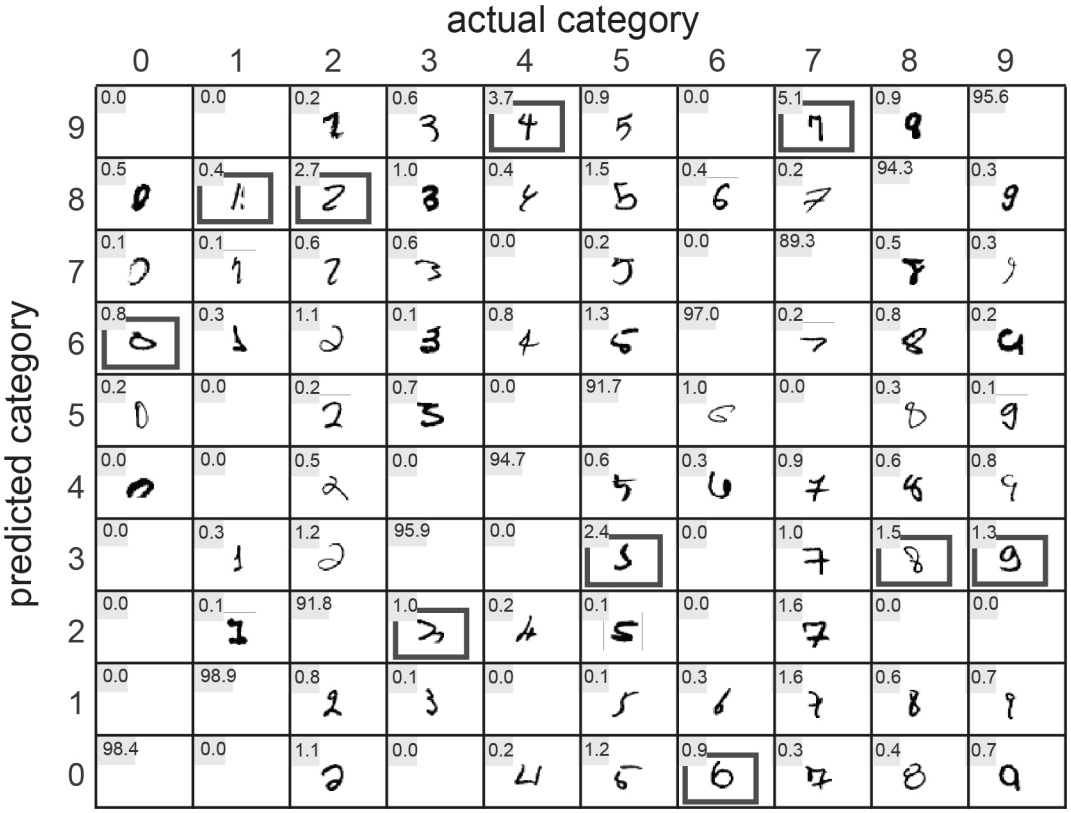


Figure VIII-8

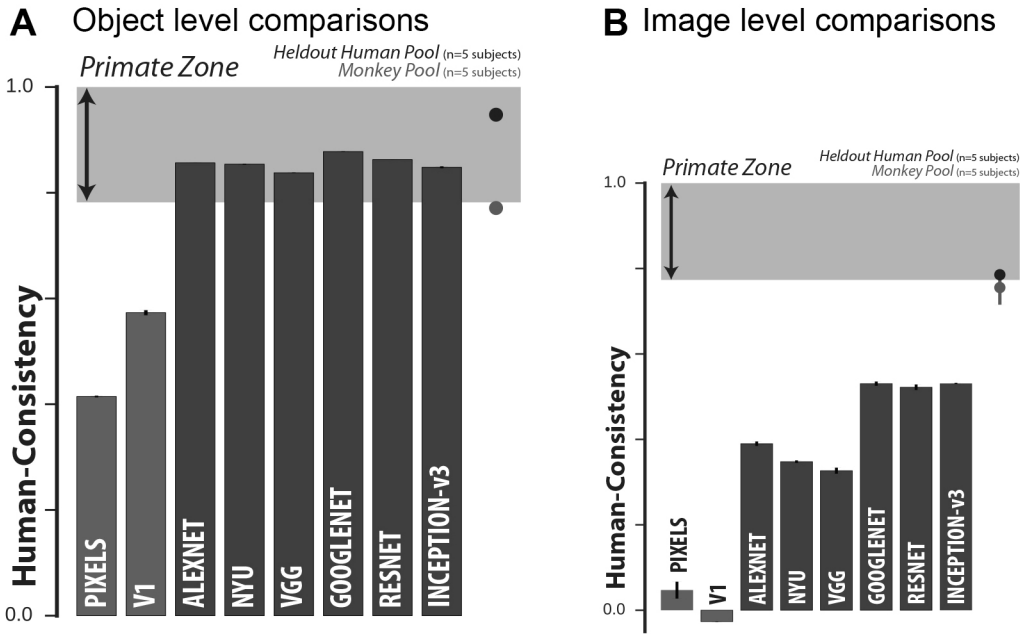


Figure VIII-9

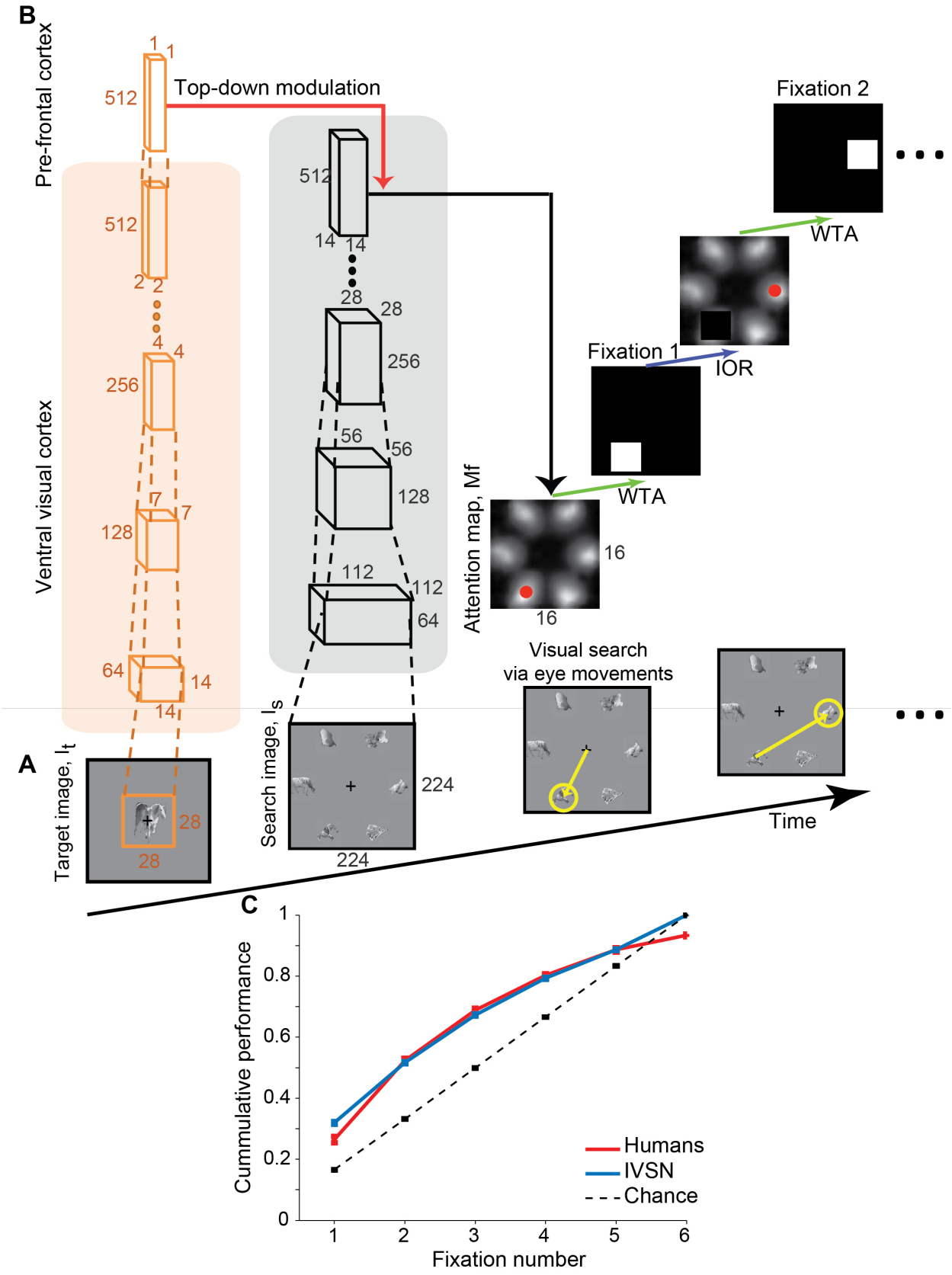


Figure VIII-10

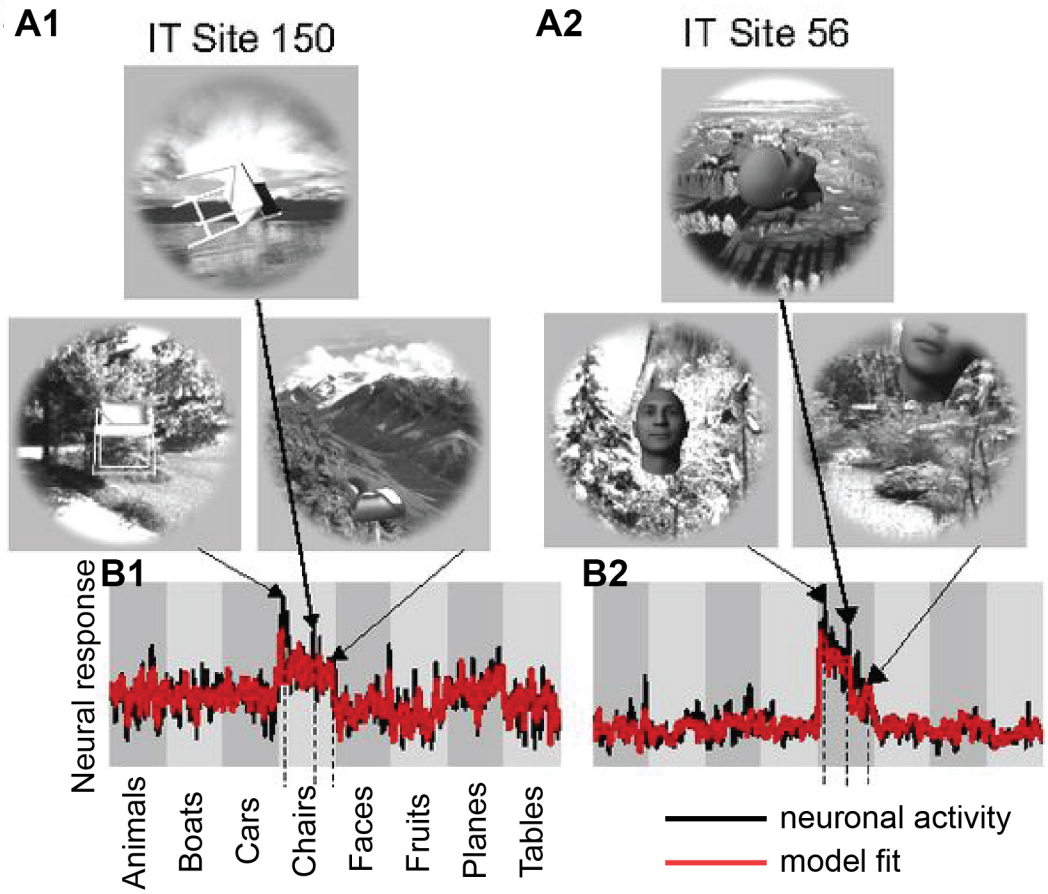


Figure VIII-11