ARTIFICIAL NEURAL NETWORKS III

Prof. Alexander Huth

4.2.2020

RECAP

* Artificial neural networks with **hidden layers**

  * General function approximators

  * Weights are optimized using gradient **backpropagation**

RECAP

* Backprop is a dynamic programming trick for efficiently computing derivatives of a loss function w/r/t some weight inside the network
  * It involves one *forward pass* (to compute the **activation** a_j at each unit in the network)
  * And one *backward pass* (to compute the **derivative** δ_j at each unit)

IS THIS STILL SYSTEM IDENTIFICATION?

* Our discussion of model fitting techniques has centered on **system identification**:

  * Given input-output pairs {(X,y)} from a system we don’t understand, find a function f(X)≈y

* System identification enables us to learn *something* about the computation performed

IS THIS STILL SYSTEM IDENTIFICATION?

- Some non-linear models (Volterra series, kernel regression) are used for this purpose, but are less **interpretable** than linearized models
  - i.e. given a non-linear model, it's harder to make **specific** & **generalizable** statements about the system

IS THIS STILL SYSTEM IDENTIFICATION?

* Artificial neural networks are especially notorious for being **uninterpretable**
* And like all non-linear methods, they are **data-hungry**
* So there is **not** a lot of work (yet) that directly uses artificial neural networks for system identification
  * (At least in the *encoding* framework that we have discussed)

SYSTEM CONSTRUCTION

* Thus far we have done system identification where **the output (y) was the activity** within some biological neural network

* Instead, let’s consider output y to be the **computational goal** of the biological neural system

* Then construct a new (artificial) neural network to achieve the same goal

SYSTEM CONSTRUCTION

* **Example:** visual object identification (aka *image classification*)

* We know the human brain does this

* Can we build a network that does this?

  * And would it work like the brain?

→ cat

→ dog

→ owl

IMAGE CLASSIFICATION

* Formalization of the classification problem:

  * **Input:** an image (256 x 256 RGB pixels = 196,608 features)

  * **Output:** a 1-hot vector (e.g. [0, 1, 0, 0, 0, …, 0]) indicating the “class” or “label” of the image

$\downarrow$

[1,0,…,0]

IMAGE CLASSIFICATION

* The methods we will use are data-hungry. Where will we get the data?

* **ImageNet:** a database of ~15 million images from thousands of categories

  * Developed by Fei-Fei Li & her group

Fei-Fei Li

IMAGE CLASSIFICATION

* How should a neural network be designed to solve this problem?
  * Suppose we used a simple network with 1 hidden layer containing 1000 units, and 1000 image classes
  * **How many weights** would this network contain?

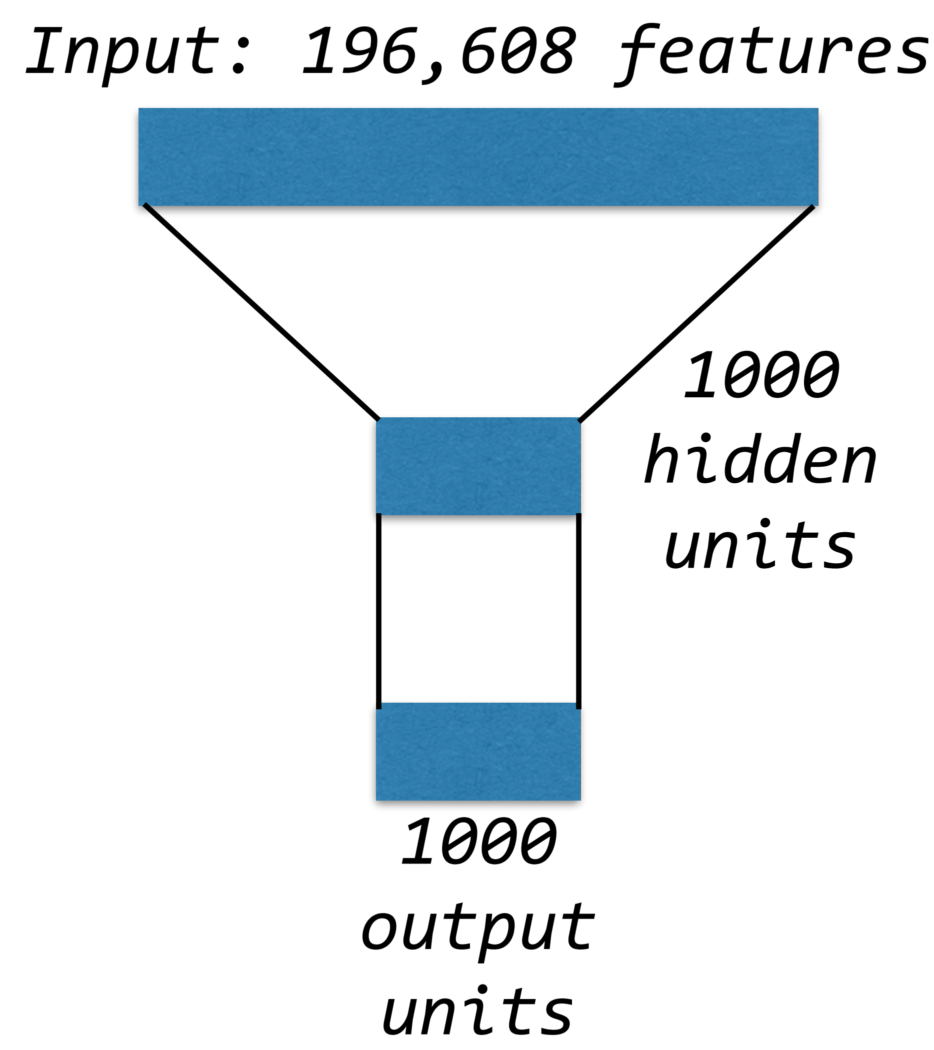


IMAGE CLASSIFICATION

* This simple network just won't work: **too many weights** (increases overfitting), **not trainable enough** (too shallow)

* What **tricks** can we use to make the problem easier?

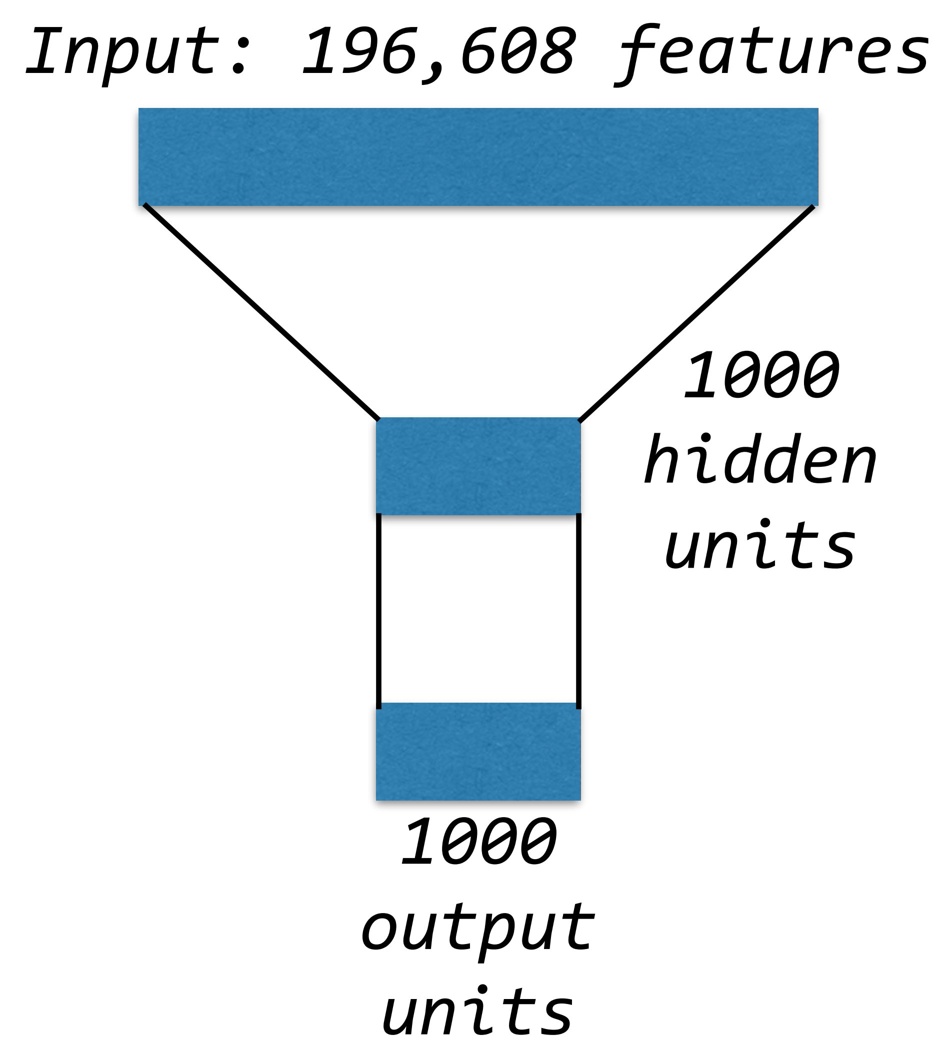


IMAGE CLASSIFICATION

* Idea 1: ***localized receptive fields***

* We know from neuroscience (Hubel & Wiesel experiments, lectures 3-4!) that visual neurons have limited **receptive fields**

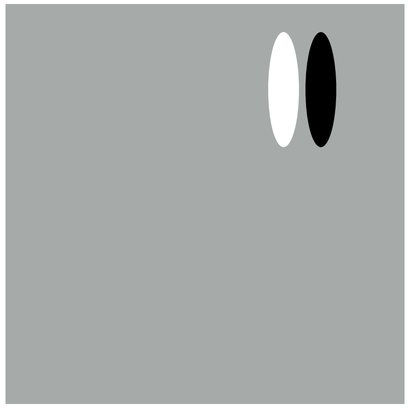


IMAGE CLASSIFICATION

* So: instead of having the entire image feed into each hidden unit, have each unit only look at **a small portion** of the image, e.g. 11x11 pixels (363 weights)

* Let’s offset each 11x11 unit by 4 pixels, giving 4096 “localized” units

* To combine info. across these patches let’s keep our original 1000-unit layer also

* How many total weights now?

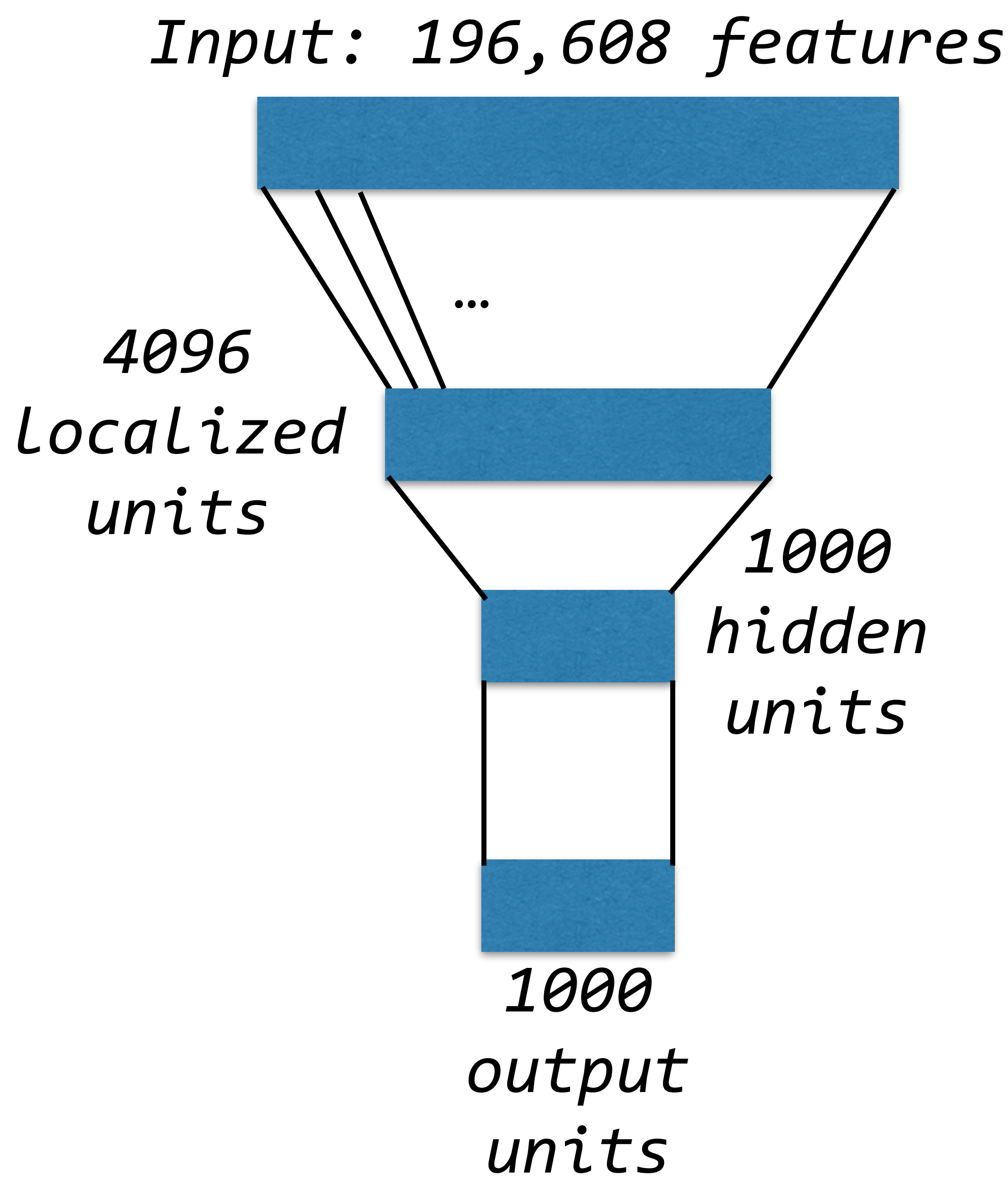


IMAGE CLASSIFICATION

* This localized network is pretty small, but not very **expressive** (it only detects one “feature” at each location!)

* Stacking more “localized” or “fully-connected” layers (or making them wider) could improve things, but at substantial cost

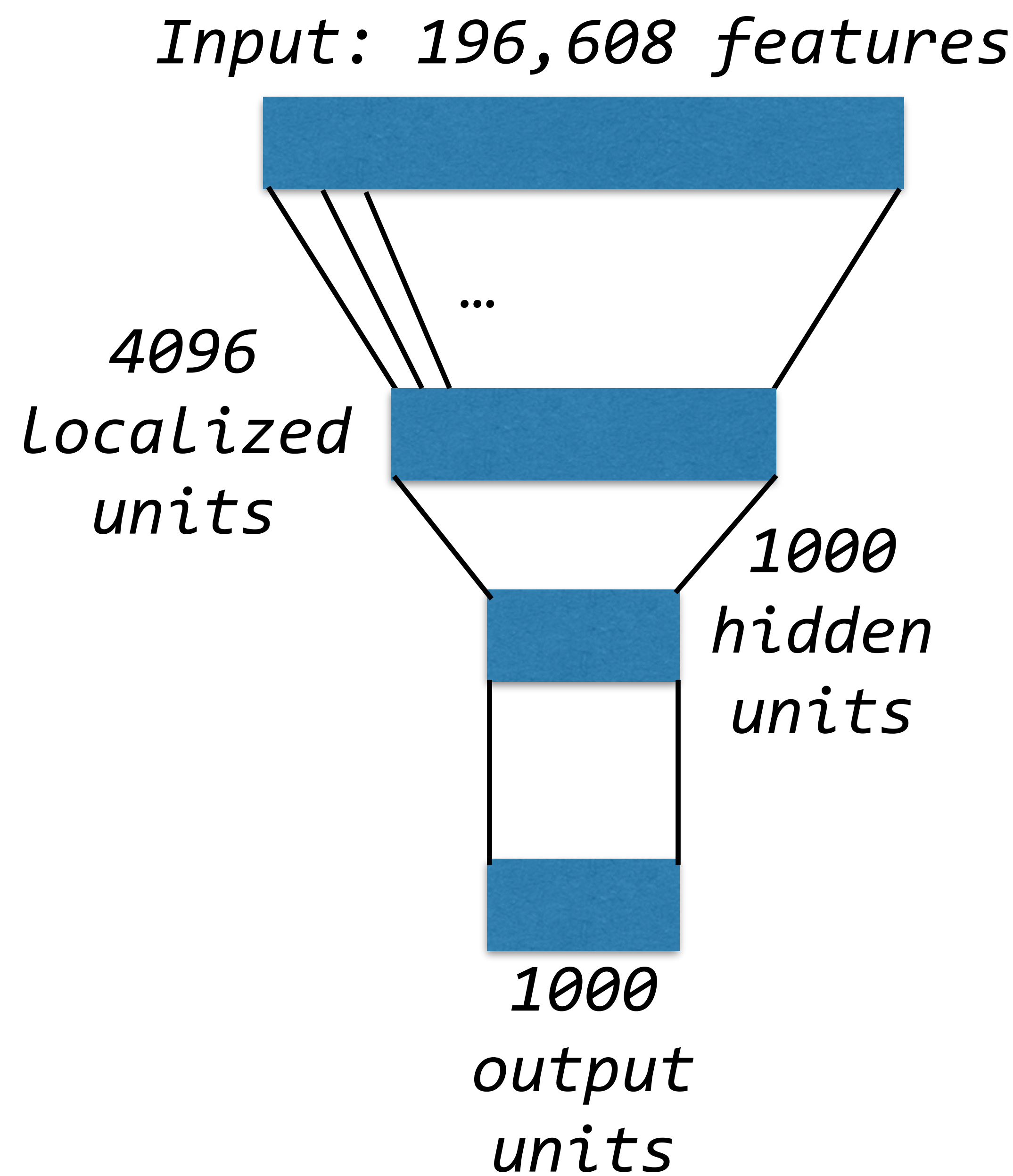


IMAGE CLASSIFICATION

* As Yann LeCun (& others) have noted, object classes are ***translation invariant***
* i.e., object class (mostly) doesn’t depend on where the object is within the image

CONVOLUTIONAL LAYER

* Idea 2: ***translation invariant networks***

* Instead of learning a separate set of weights for each “localized” unit, let’s learn one set of weights that is **shared** across all the localized units

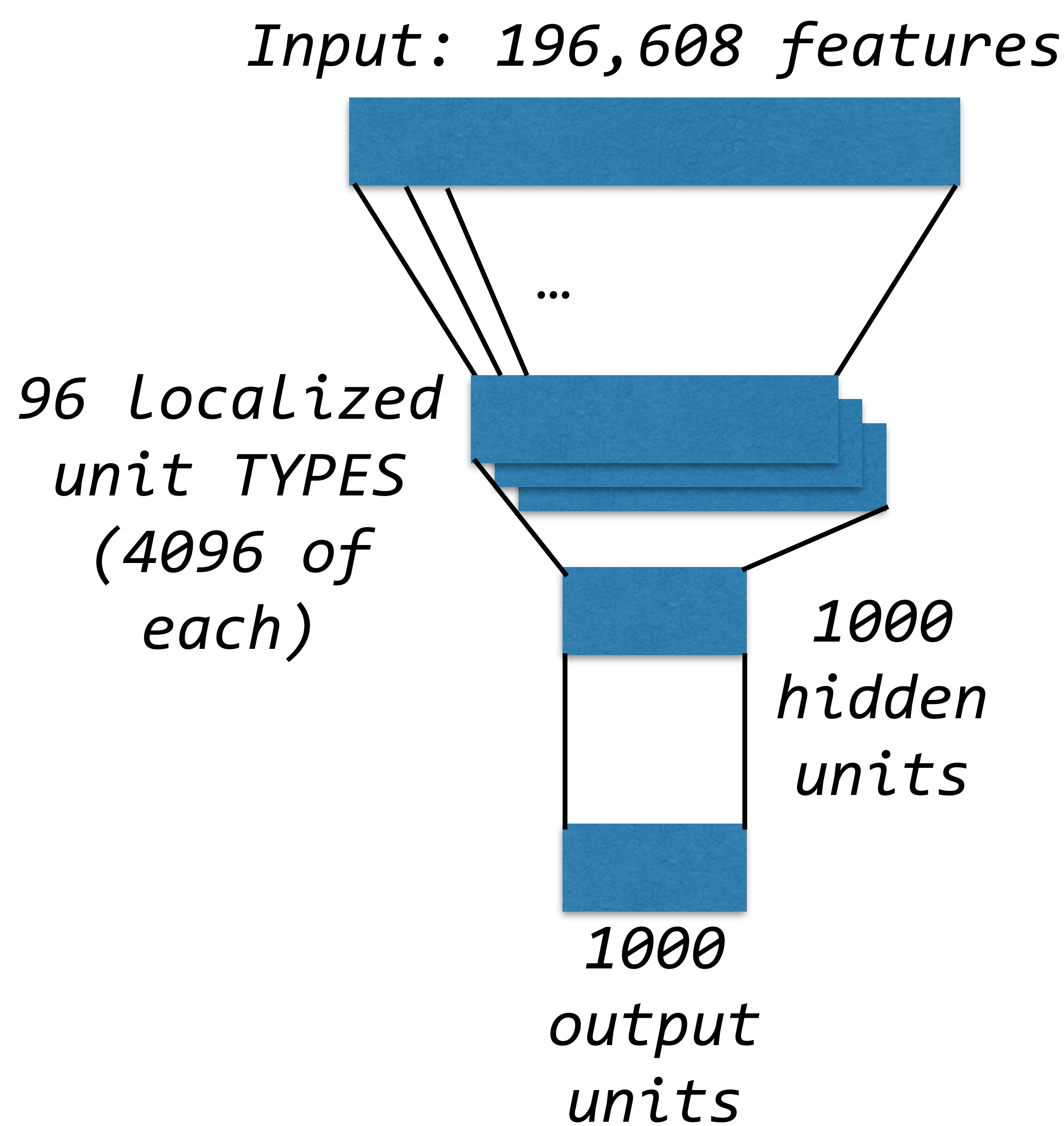
* This operation—applying the same filter to each patch in the image—resembles **convolution** of the image with a filter/kernel

  * So let’s call it a “convolutional layer”

CONVOLUTIONAL LAYER

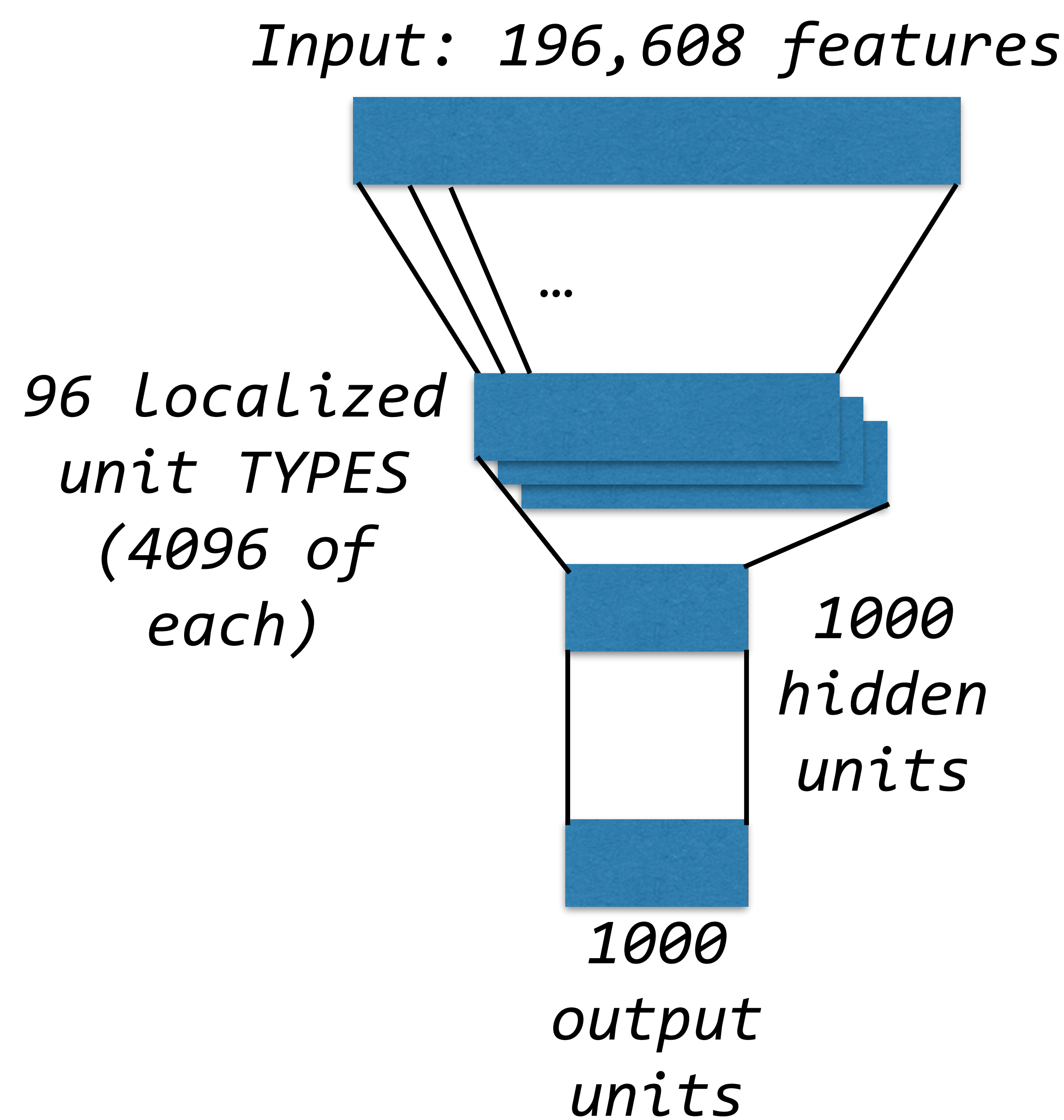
* Using convolution makes it easy to add more “feature detectors” for each location in the image

* Instead of training 4096 detectors (one for each location), we can just train e.g. 96, each of which is shared across the 4096 locations



CONVOLUTIONAL LAYER

* This network is better, and doesn't have too many weights!
* But it's still not very expressive
* This could be improved by making it deeper, but that's expensive



CONVOLUTIONAL LAYER

* Convolutional layers enable us to detect whether the same feature appears at any location in the image

* But what we are ultimately concerned with is whether some feature (e.g. a dog) appears *anywhere* in the image

CONVOLUTIONAL LAYER

* We can try to find things “anywhere in the image” with one more trick: ***max pooling***

* Instead of directly using the output of one convolutional layer as the input for the next, we can reduce the effective “image size” of each convolutional layer by **pooling** the activations across adjacent units

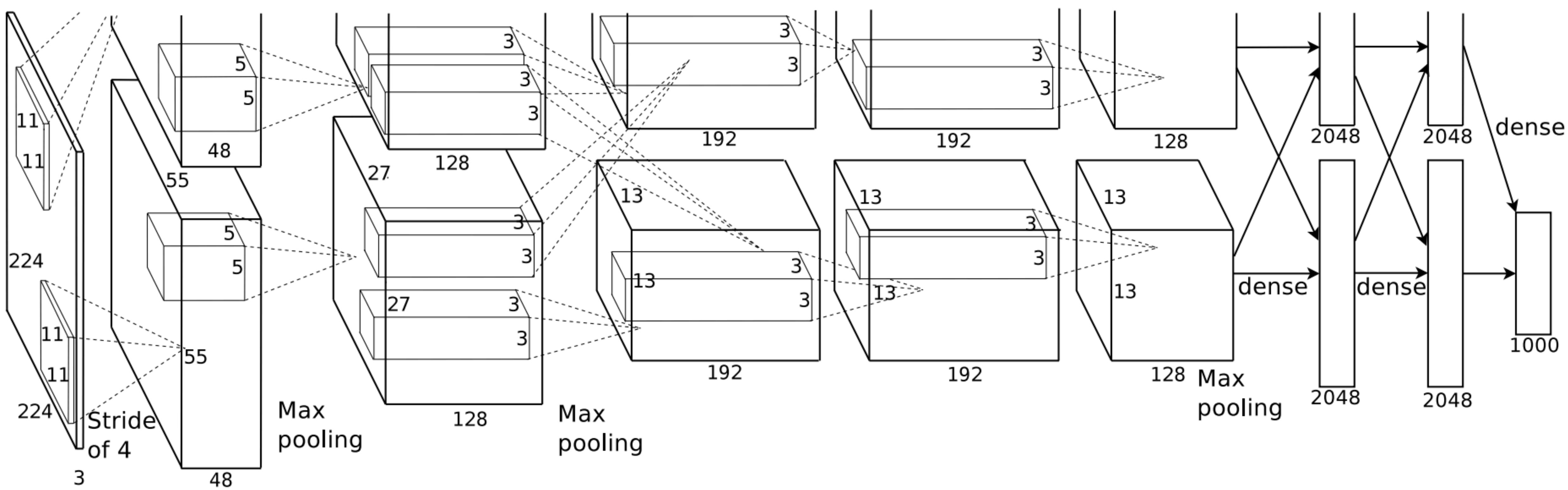
* Taking the **maximum** across a group of units is like a logical OR, captures the “anywhere” property we care about

IMAGE CLASSIFICATION

* Our new bag of tricks:
  * Localized receptive fields
  * Shared weights (i.e. convolution)
  * Downsampling between convolutional layers (i.e. max pooling)
* + lots of data (ImageNet)

ALEXNET

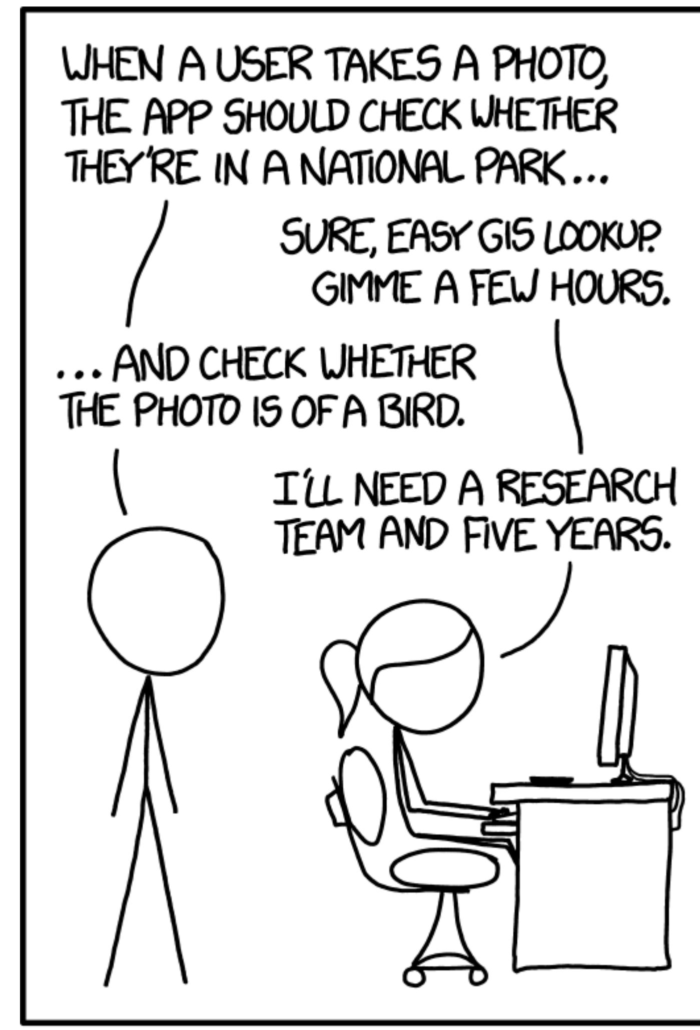
* These tricks all came together in AlexNet (Krizhevsky, Sutskever, & Hinton, 2012)



ALEXNET

* The first really successful deep neural network model

* Showed that artificial neural networks (with the right bag of tricks!) can actually do things (at least somewhat!) that we thought were computationally "hard"



IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

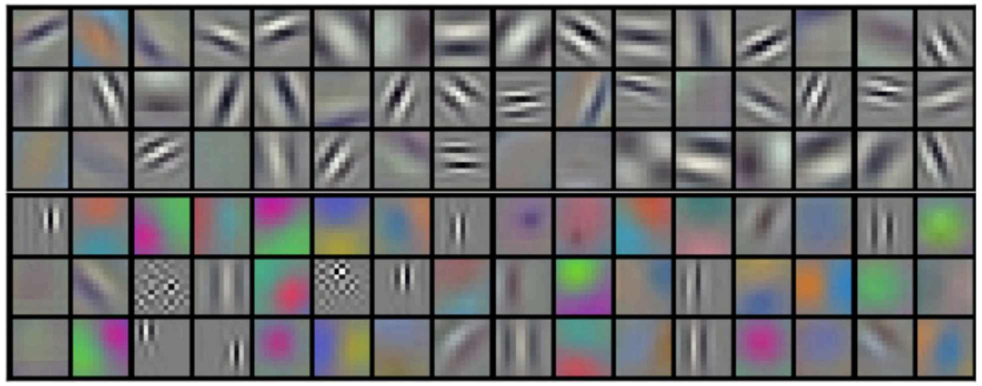
ALEXNET

* Does it solve the problem the same way that our brains do?

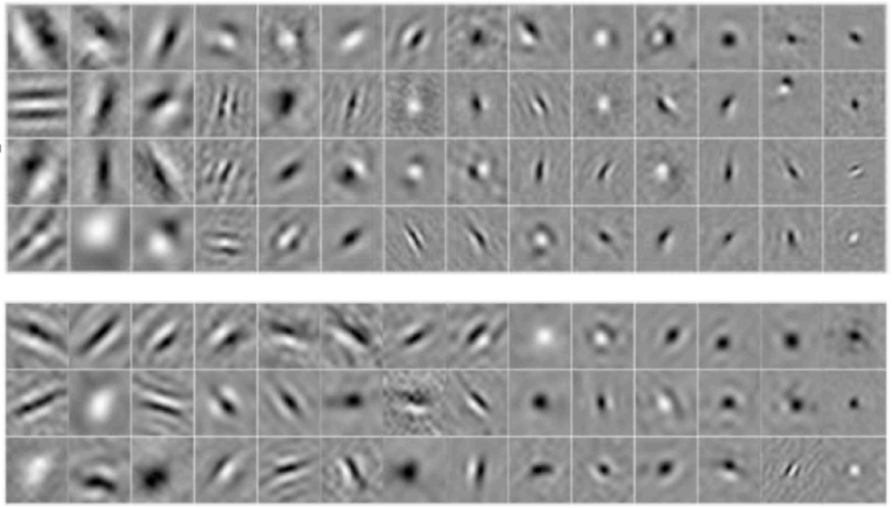
* How would we know?

ALEXNET

* Features learned by 1st layer of AlexNet



* Receptive fields of V1 neurons from a macaque



RECAP

* Switch from “system identification” to “system construction”

* Image classification

  * Localization

  * Convolution (weight sharing)

  * Downsampling (max pooling)

* AlexNet

NEXT TIME

* Comparing artificial neural networks and biological neural networks that perform the same task (in vision)