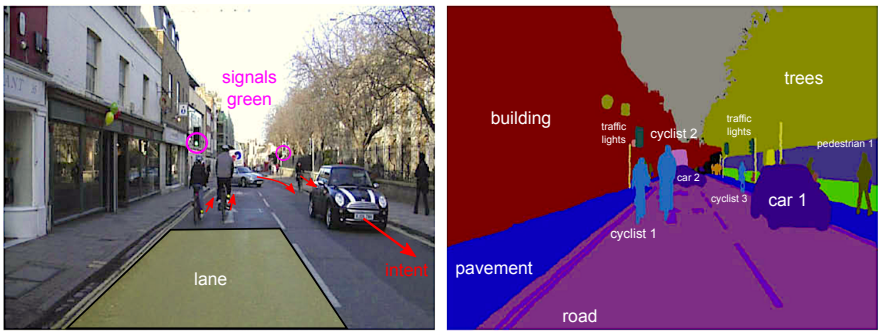
Deep Learning
Part 2: Convolutional Neural Networks

Dr. Richard E. Turner (ret26@cam.ac.uk)

Engineering Tripos Part IB
Paper 8: Information Engineering

Big picture

- **Goal**: produce automatic systems that solve image recognition problems
  - detect and classify objects into categories, independently of pose, scale, lighting and weather conditions, occlusion and clutter
- **initial computer vision approach**: hand-crafted features (not robust)
- **deep learning approach**: learn suitable systems directly from labelled images



Approach

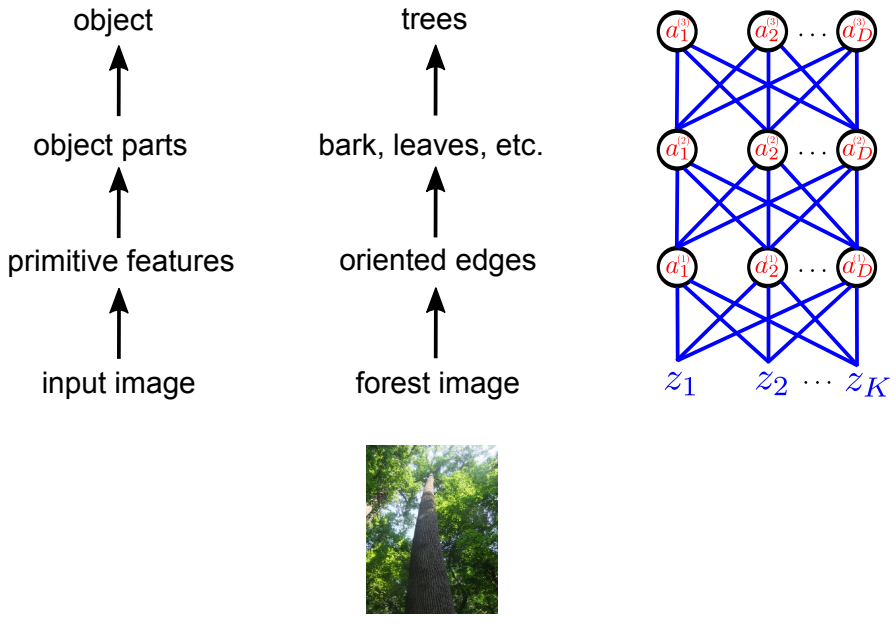
1. use **deep neural networks** that learn representations of images with multiple layers of abstraction
   - the structure of visual scenes and hierarchical organisation of biological vision are motivations for this approach

2. **design bespoke neural networks for images** so that they can handle input images with millions of pixels
   - multi-layer perceptrons (MLPs) cannot handle such high dimensional inputs
   - leverage properties of images to design **convolutional neural networks**

3. **improve the training algorithm** so that it can scale to many images
   - optimisation via gradient descent is slow for large numbers of data points
   - develop **stochastic gradient ascent** to improve scalability

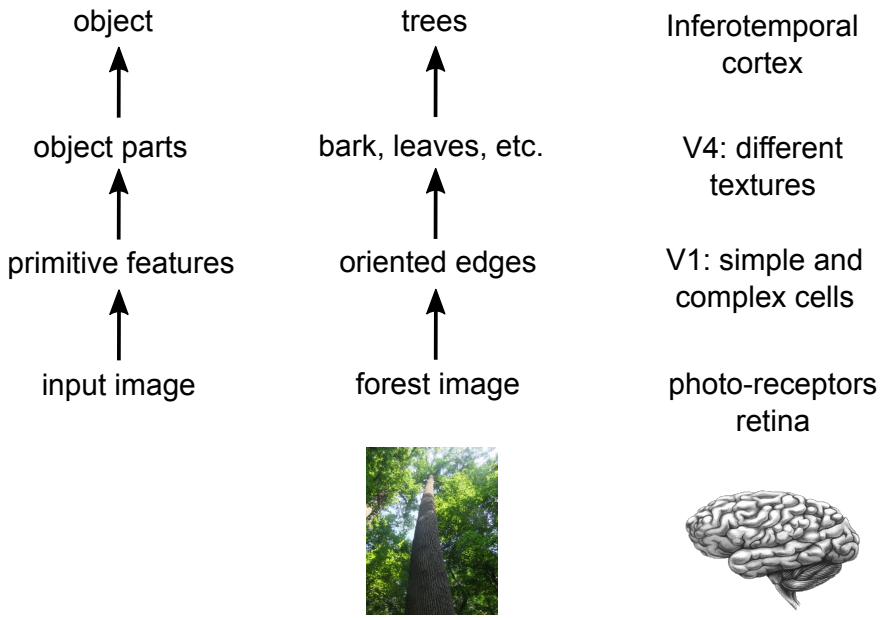
Why use hierarchical multi-layered models i.e. deep learning?

Argument 1: visual scenes are hierachically organised



Why use hierarchical multi-layered models i.e. deep learning?

Argument 2: biological vision is hierachically organised



Couldn't we use standard neural networks like MLPs?

How many parameters does this neural network have?

number of parameters $|\theta| = 3D^2 + D$

For a small 32 by 32 image:

$$|\theta| = 3 \times 32^4 + 32^2 \approx 3 \times 10^6$$

Hard to train

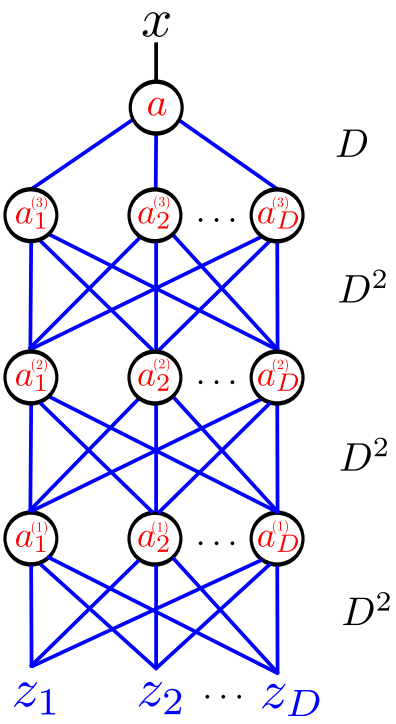
over-fitting and local optima

slow to converge

Huge memory footprint

limits complexity of trained networks

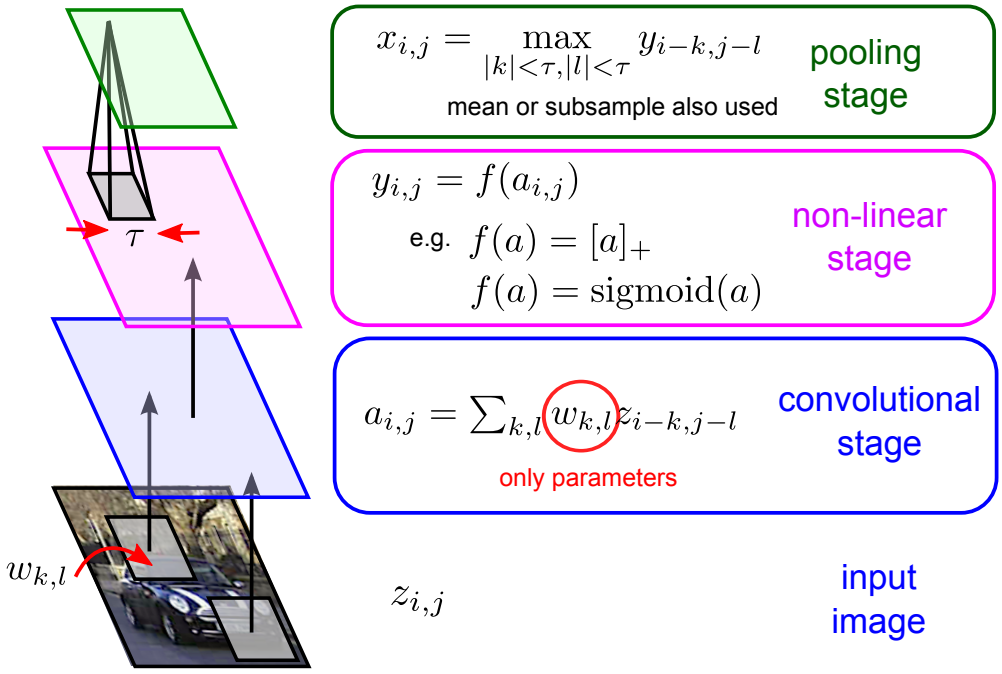
Convolutional nets reduce the number of parameters without compromising network complexity



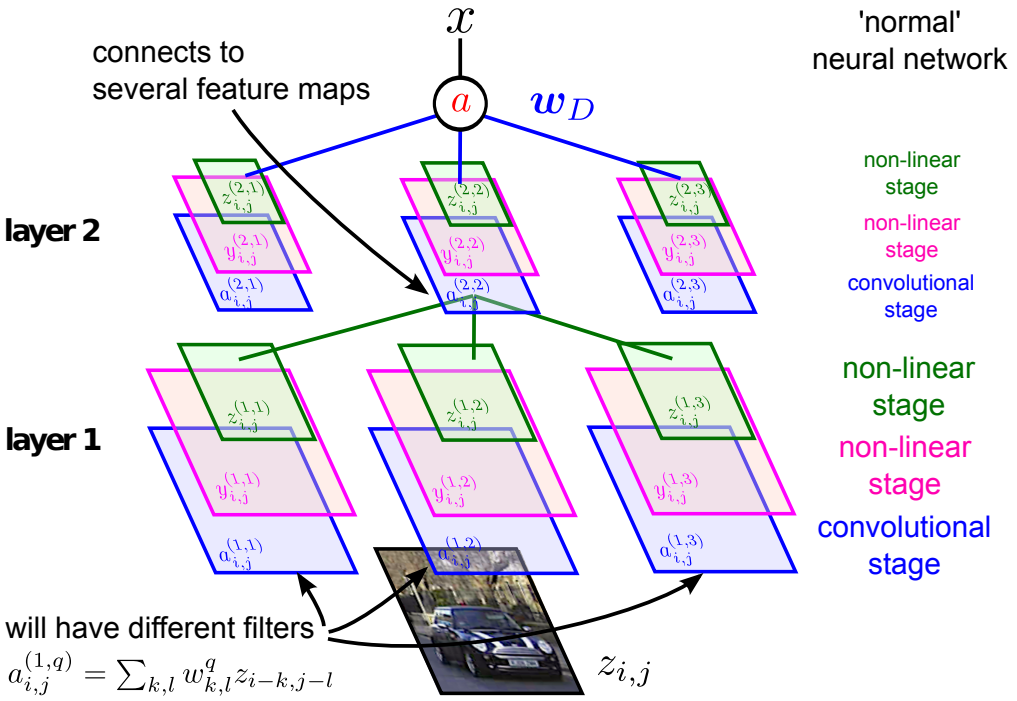
The key ideas behind convolutional neural networks

- **image statistics are translation invariant** (objects and viewpoint translates)
  - build this translation invariance into the model (rather than learning it)
  - tie lots of the weights together in the network
  - reduces number of parameters
- **expect learned low-level features to be local** (e.g. edge detector)
  - build this into the model by allowing only local connectivity
  - reduces the numbers of parameters further
- **expect high-level features learned to be coarser** (c.f. biology)
  - build this into the model by subsampling more and more up the hierarchy
  - reduces the number of parameters again

Building block of a convolutional neural network



Full convolutional neural network



How many parameters does a convolutional network have?

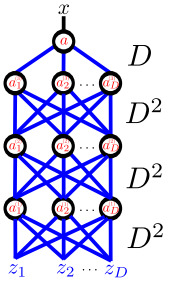
How many parameters does this neural network have?

$$|\theta| = 3K^2 + 9K^2 + 9K^2 + 3(D/S)^2$$
$$= 21K^2 + 3(D/S)^2$$

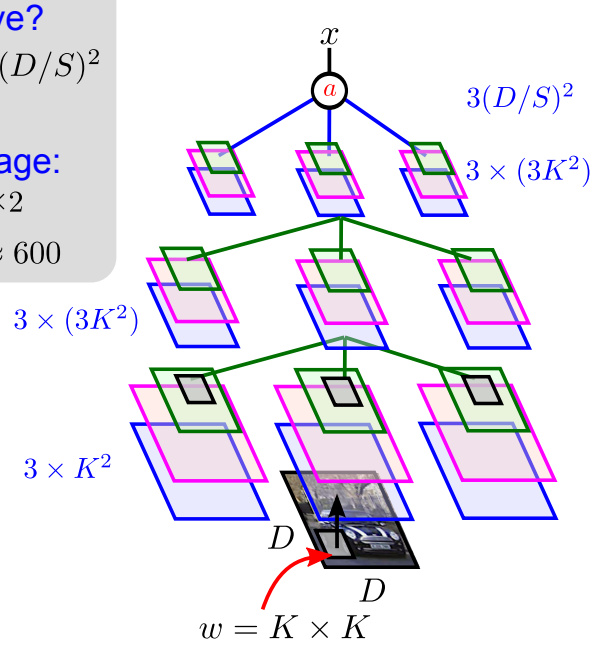
For a small 32 by 32 image:

$K = 5 \qquad S = 2{\times}2{\times}2$

$$|\theta| = 21 \times 5^2 + 3{\times}4^2 \approx 600$$



$$|\theta| = 3D^2 + D \approx 3 \times 10^6$$



Training: gradient descent

Output of convolutional neural network interpreted as a **class label probability**

$$x^{(n)} = x(\mathbf{z}^{(n)}; \mathcal{W}) = p(y^{(n)} = 1 | \mathbf{z}^{(n)}, \mathcal{W})$$

Now $x^{(n)}$ is a more complex function of the inputs $\boldsymbol{z}^{(n)}$ and contains lots of parameters $\mathcal{W}$.

Optimise the **cross entropy** (with regularisation / weight decay)

$$G(\mathcal{W}) = \sum_{n=1}^{N} \left(y^{(n)} \log x^{(n)} + (1 - y^{(n)}) \log(1 - x^{(n)}) \right) = \sum_{n=1}^{N} l(y^{(n)}, x^{(n)})$$

But, will take a **long time** to accumulate objective and derivatives over all N training data points.

$$\frac{\mathrm{d}G(\mathcal{W})}{\mathrm{d}\mathcal{W}} = \sum_{n=1}^{N} \frac{\mathrm{d}\, l(y^{(n)}, x^{(n)})}{\mathrm{d}\mathcal{W}} \qquad \mathcal{W} \leftarrow \mathcal{W} - \eta \left(\frac{\mathrm{d}G(\mathcal{W})}{\mathrm{d}\mathcal{W}} - \alpha \mathcal{W} \right)$$

Training: stochastic gradient descent

Observation: **often only need to see a small number of images before you get a good idea of the gradient direction**

Idea: **at each step of gradient descent, randomly select $M \leq N$ data points from the training dataset $\{t^{(m)}, \mathbf{z}^{(m)}\}_{m=1}^{M}$ and estimate the objective / gradient from these**

$$G(\mathcal{W}) = \sum_{n=1}^{N} l(t^{(n)}, x^{(n)}) \approx \frac{N}{M} \sum_{m=1}^{M} l(t^{(m)}, x^{(m)})$$

$$\frac{\mathrm{d}G(\mathcal{W})}{\mathrm{d}\mathcal{W}} = \sum_{n=1}^{N} \frac{\mathrm{d}\, l(t^{(n)}, x^{(n)})}{\mathrm{d}\mathcal{W}} \approx \frac{N}{M} \sum_{m=1}^{M} \frac{\mathrm{d}\, l(t^{(m)}, x^{(m)})}{\mathrm{d}\mathcal{W}}$$

Turn down learning rate η as learning progresses (e.g. using a validation dataset).

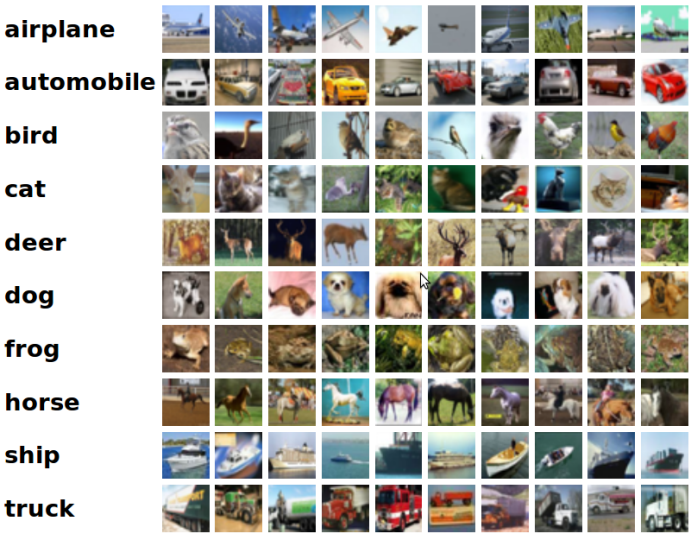
Training: additional tips and tricks

- **back-propagation for training**: stochastic gradient ascent
  - take care to initialise scales of the weights correctly (high weights cause the output to saturate and the gradients fall to zero)
  - using non-saturating hidden units can make the optimisation simpler (e.g. rectified linear hidden units rather than sigmoid units)

- **data-augmentation**: always improves performance substantially (include shifted, rotations, mirroring, cropping, locally distorted versions of the training data, computer graphics)

- **typical numbers**:
  - VGG-16: 16 convolutional layers, $\approx$100 channels, 3$\times$3 filters, 3 layers of 1000 units in top neural network
  - 100 million parameters
  - 1 week to train on the imagenet database (100GB, GPUs)

Demo

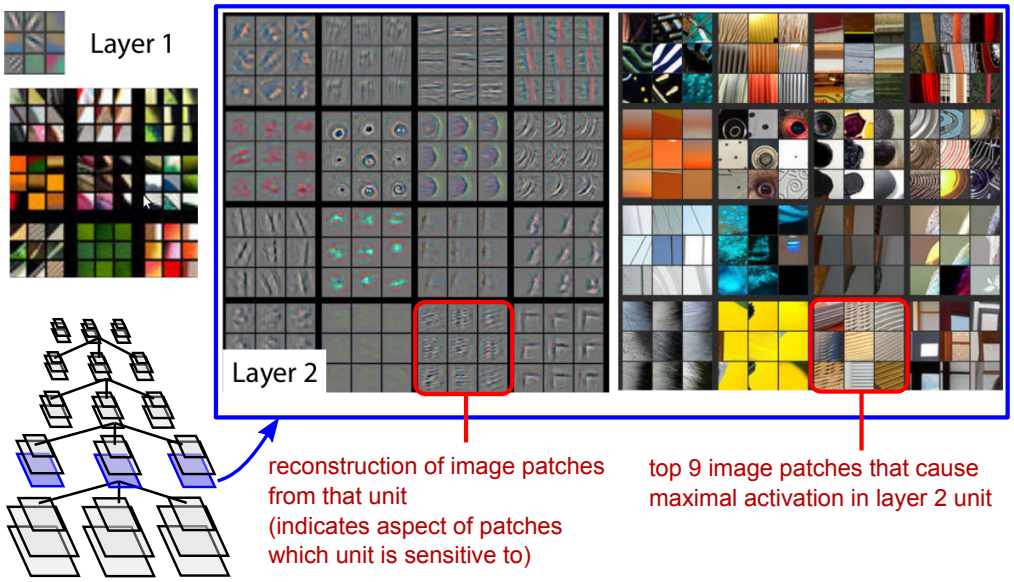


CIFAR 10 dataset: 50,000 training images, 10,000 test images

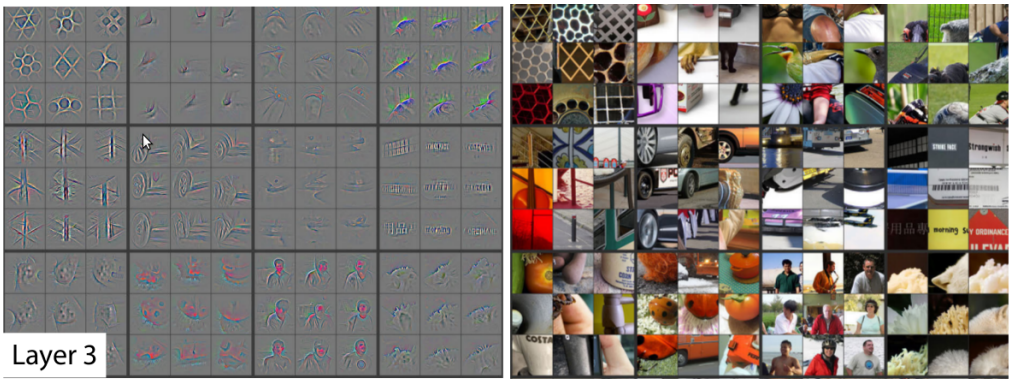
`http://matajoh.github.io/anndemos/`

`http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html`

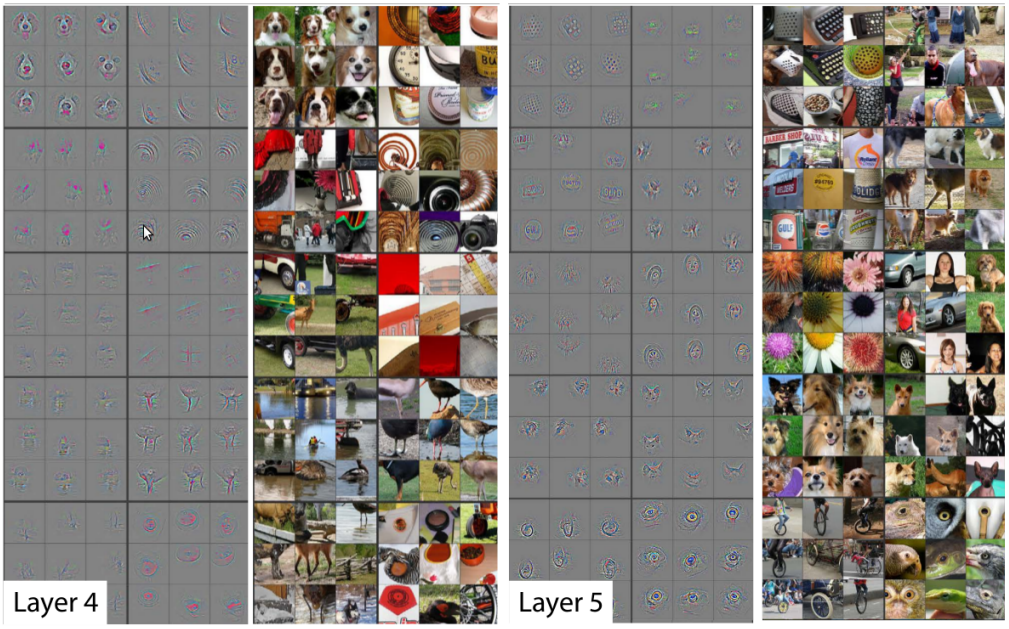
Looking into a convolutional neural network's brain



Looking into a convolutional neural network's brain



Looking into a convolutional neural network's brain



Summary

- convolutional neural networks are the go-to model for computer vision classification problems
  - form of neural network with an architecture suited to vision problems (convolutional structure and pooling)
  - higher level layers encode more **abstract features**
  - higher level layers show more **invariance to instantiation parameters** (translation, rotation, lighting changes, etc.)

- trained on a labelled training data set by optimising the regularised cross-entropy using **stochastic gradient descent**

- network automatically **learns feature detectors**
  - first layer learns edge detectors
  - subsequent layers more complex (corner detectors, blob detectors etc.)
  - integrates training of the classifier with training of the feature representation