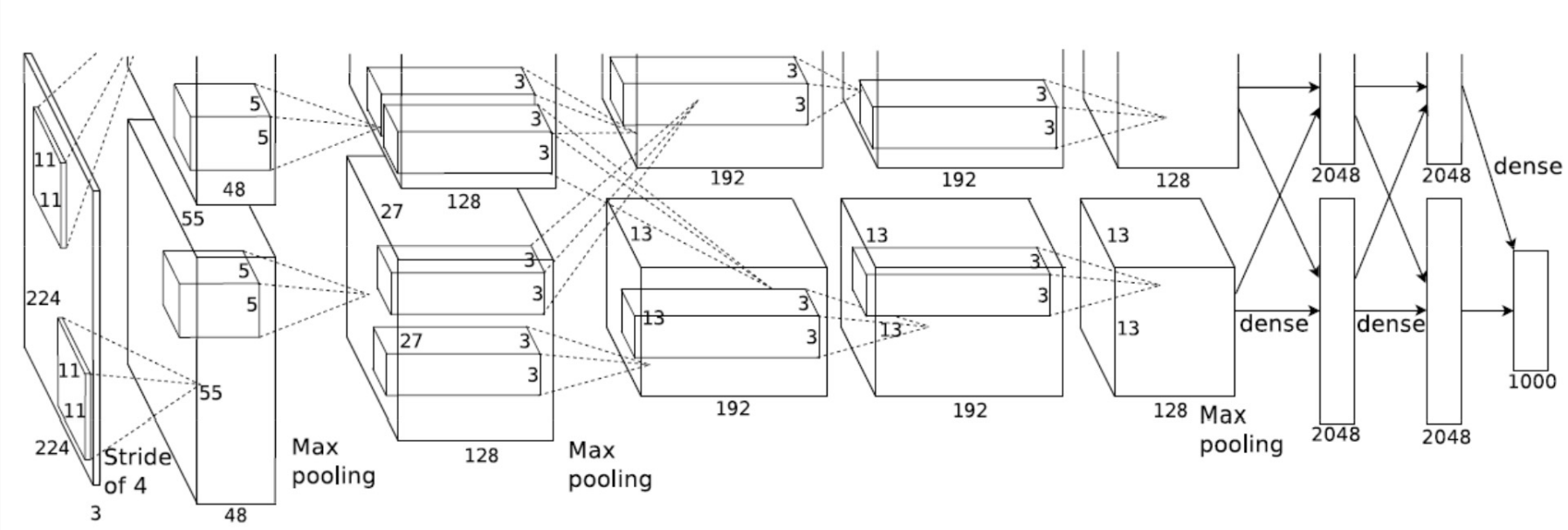
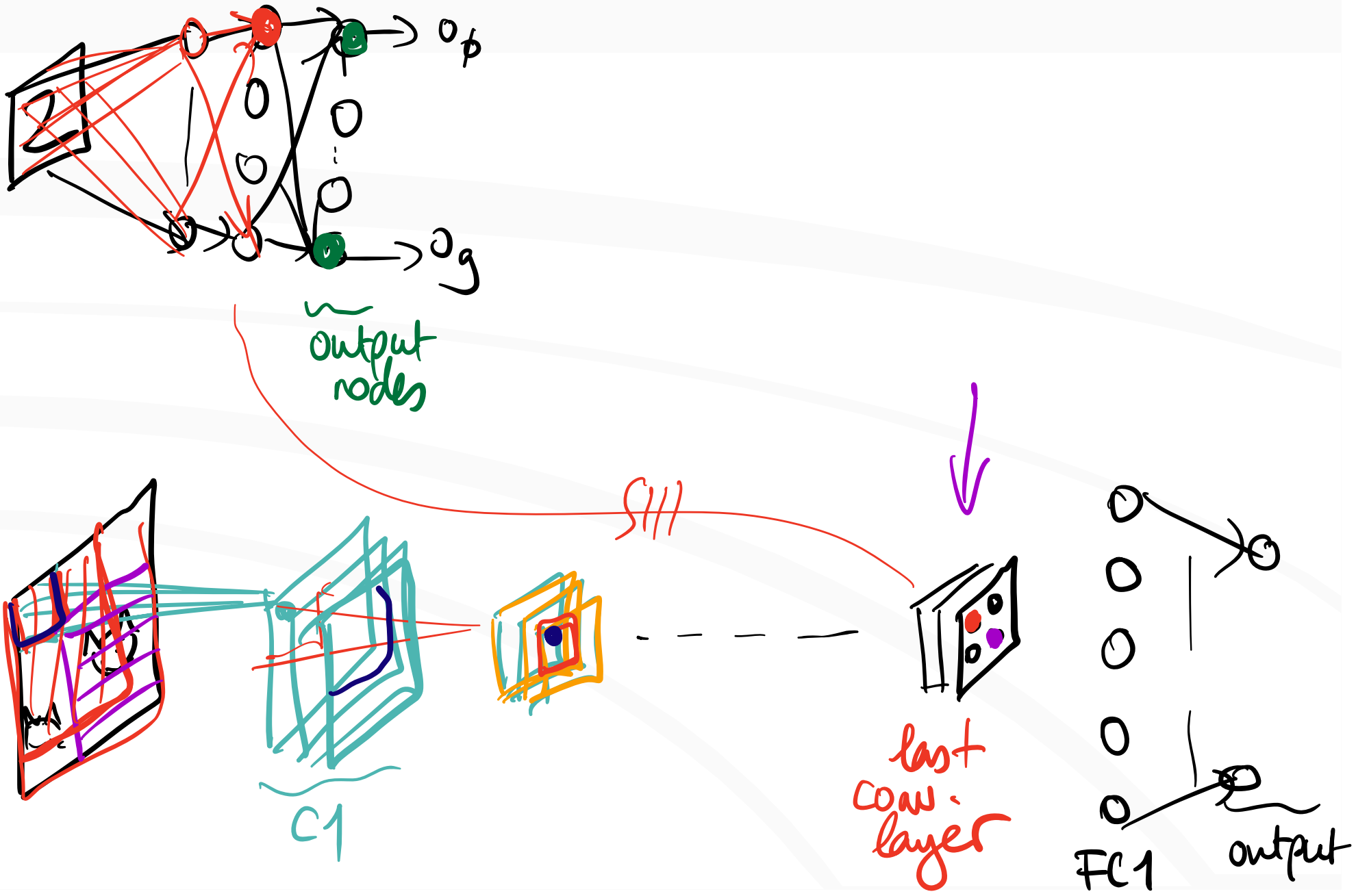
Convolutional Neural Networks



Berrin Yanikoglu
Sabancı University

Many slides thanks to: http://cs231n.github.io/convolutional-networks/

Motivation for Convolutional Layers



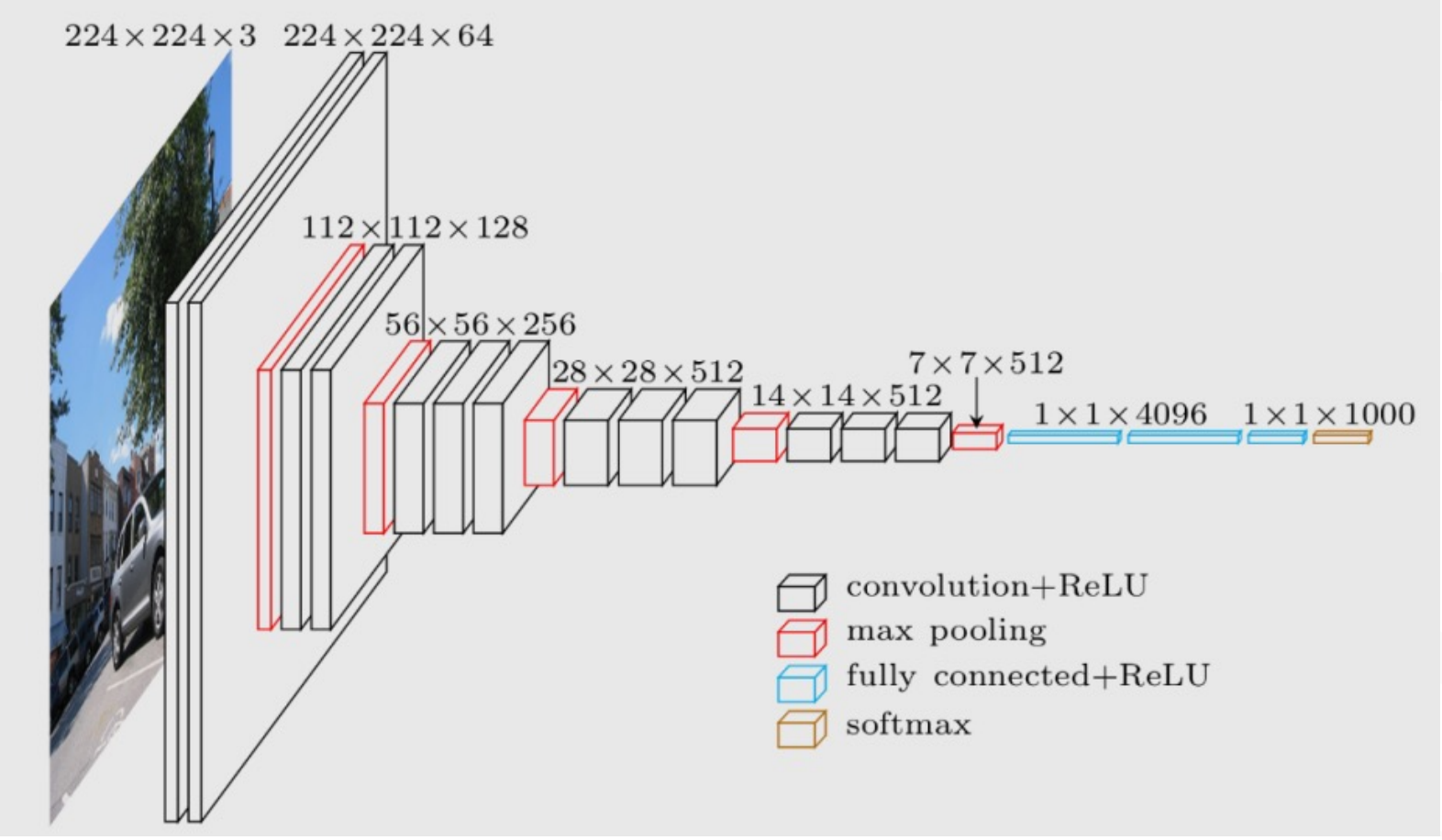
CONVOLUTIONAL NEURAL NETWORKS

Receptive Fields, Convolution Operation

Layers and Feature Maps

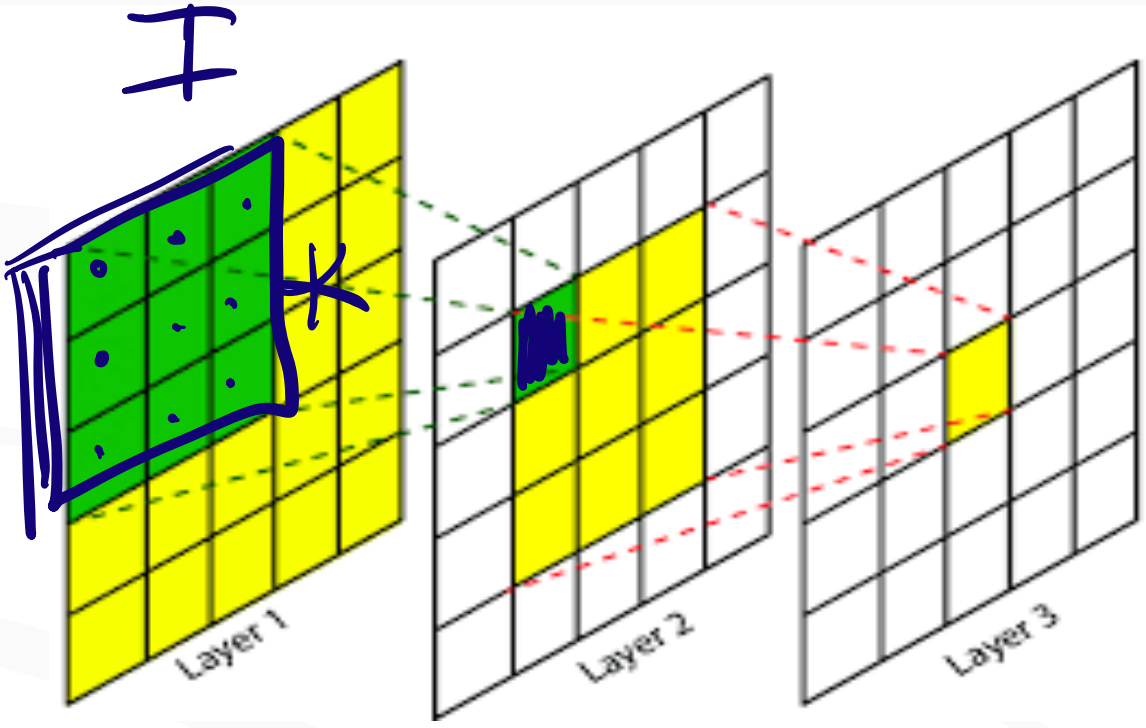
Convolutional Neural Networks (CNNs)

- Cells in a layer take input from small sub-regions of the visual field (receptive field).
- Nearby neurons are connected to nearby regions and respond to spatially local input patterns.
- Receptive fields are tiled to cover the entire visual field.



Receptive Fields

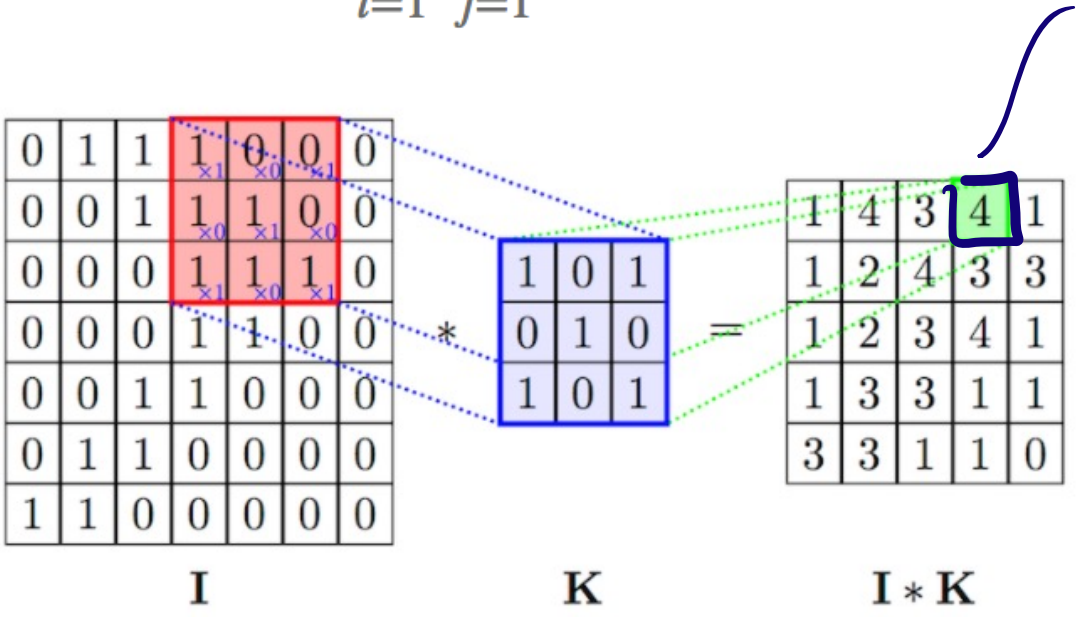
- Receptive fields;
  - have small sizes
  - are spatially contiguous
- Each neuron computes its output by computing the dot product of its weight matrix and its receptive field.
- Stacking many such layers leads to “filters” that become increasingly “global” (i.e. responsive to a larger region of pixel space).



$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1}$$

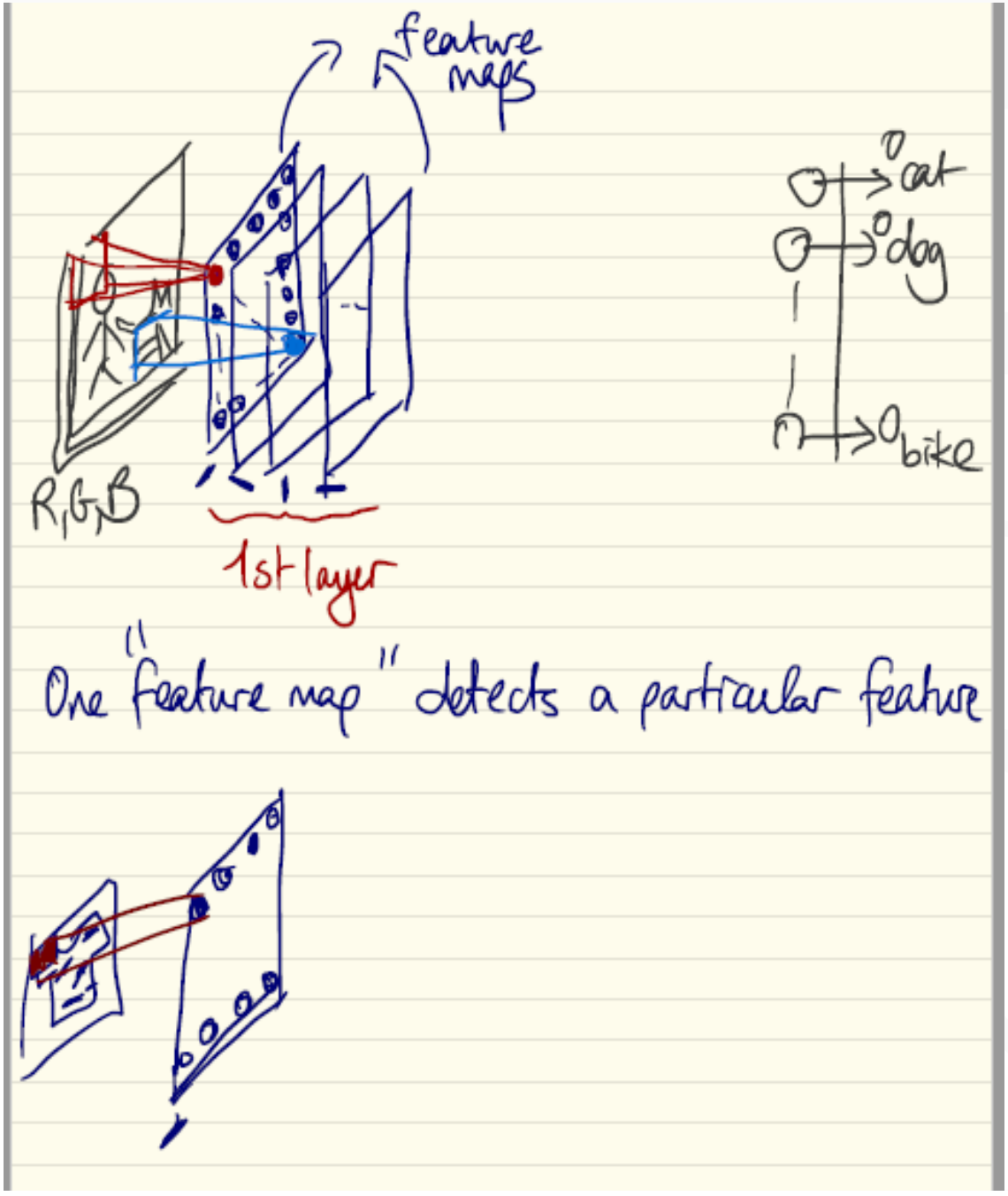
Convolution

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1}$$



Output of this neuron is 4

- Dot product between pixels in the receptive field (subregion of I) and the kernel (K).



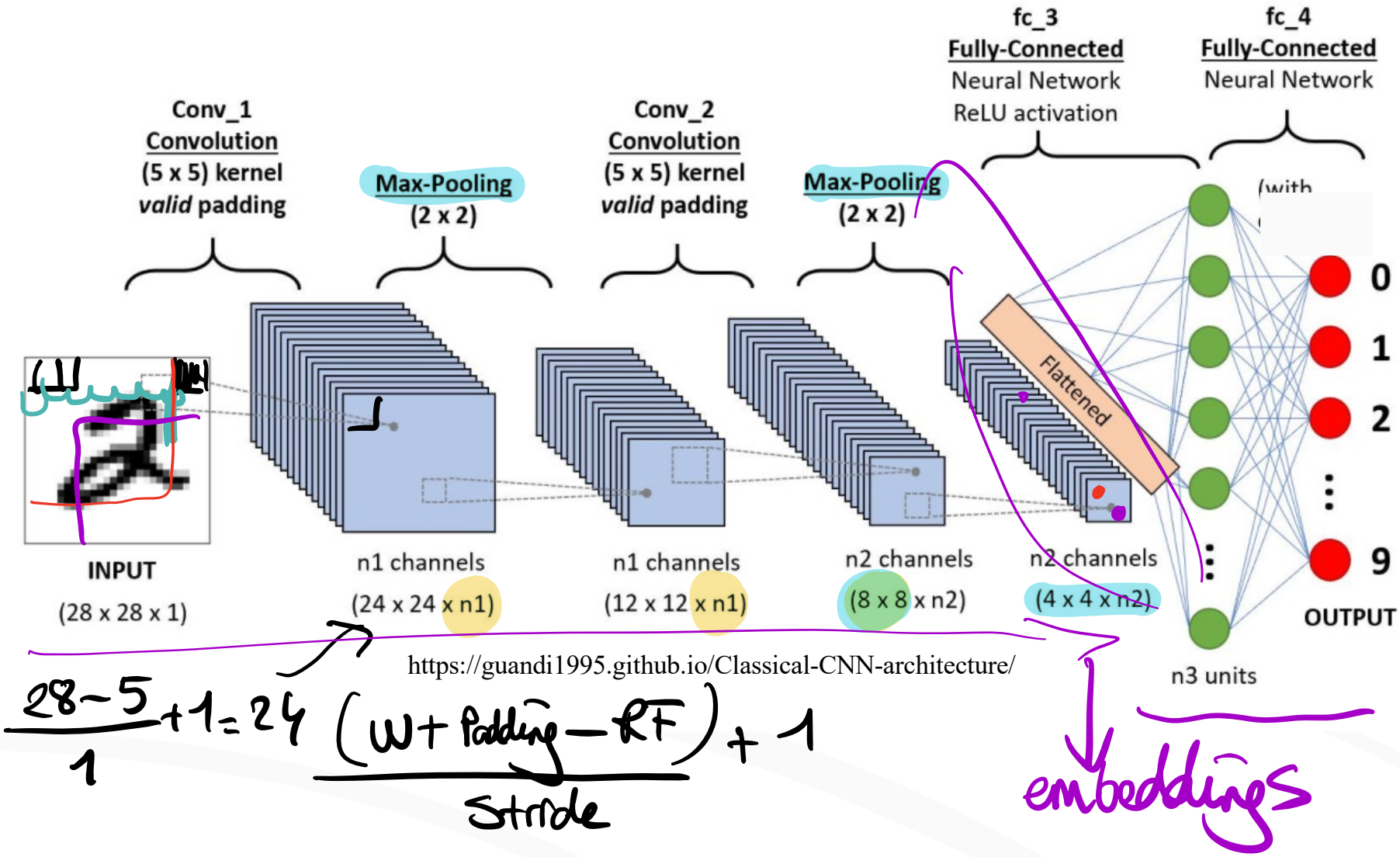
• Each filter is replicated across the entire visual field (image).

• These replicated units share the same weights and form a feature map.

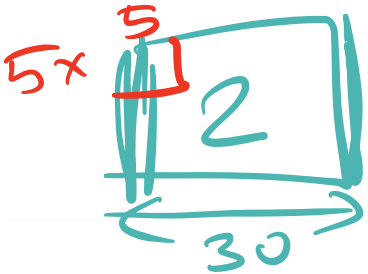
• Replicating units in this way allows for features to be detected regardless of their position in the visual field.

• Additionally, weight sharing greatly reduces the number of free parameters being learnt

• Each neuron that share the same position (in different feature maps) take input from the same cells in the previous layer.

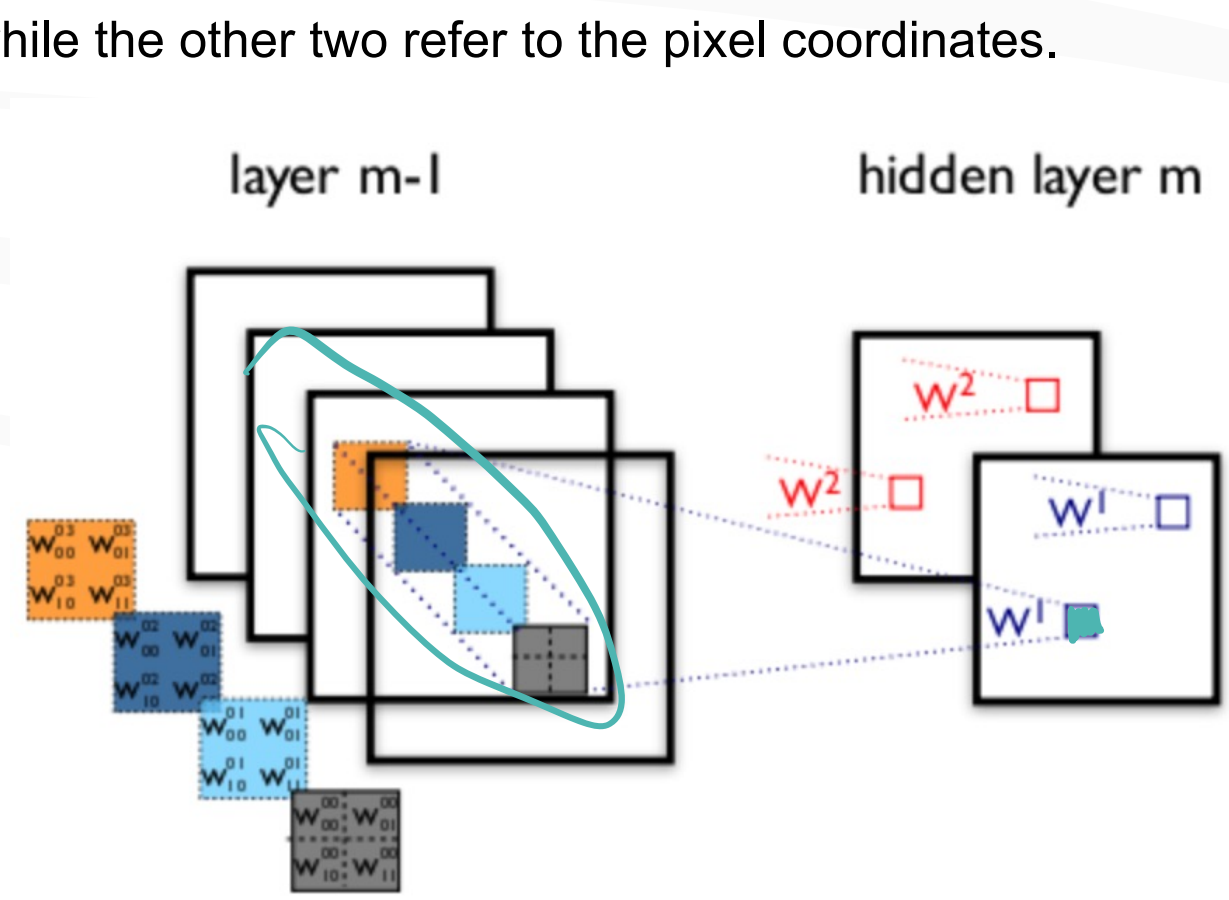


- In one layer, there are multiple feature maps, to detect different patterns:
  - One can detect vertical edges, another horizontal edges, another hollow centers,,…
  - n1 feature maps (channels) on layer1 above



Feature Maps and Receptive Fields

- There are 4 and 2 feature maps in layers m-1 and m, respectively.
- The receptive field of nodes in layer m spans all four input feature maps.
- The weights are 3D weight tensors.
  - The leading dimension indexes the input feature maps, while the other two refer to the pixel coordinates.

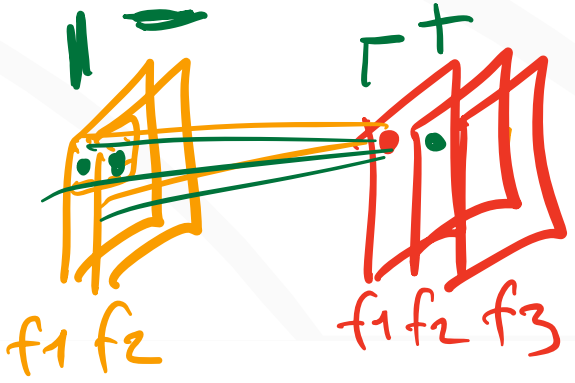


Choosing Hyper Parameters: Receptive Field Size

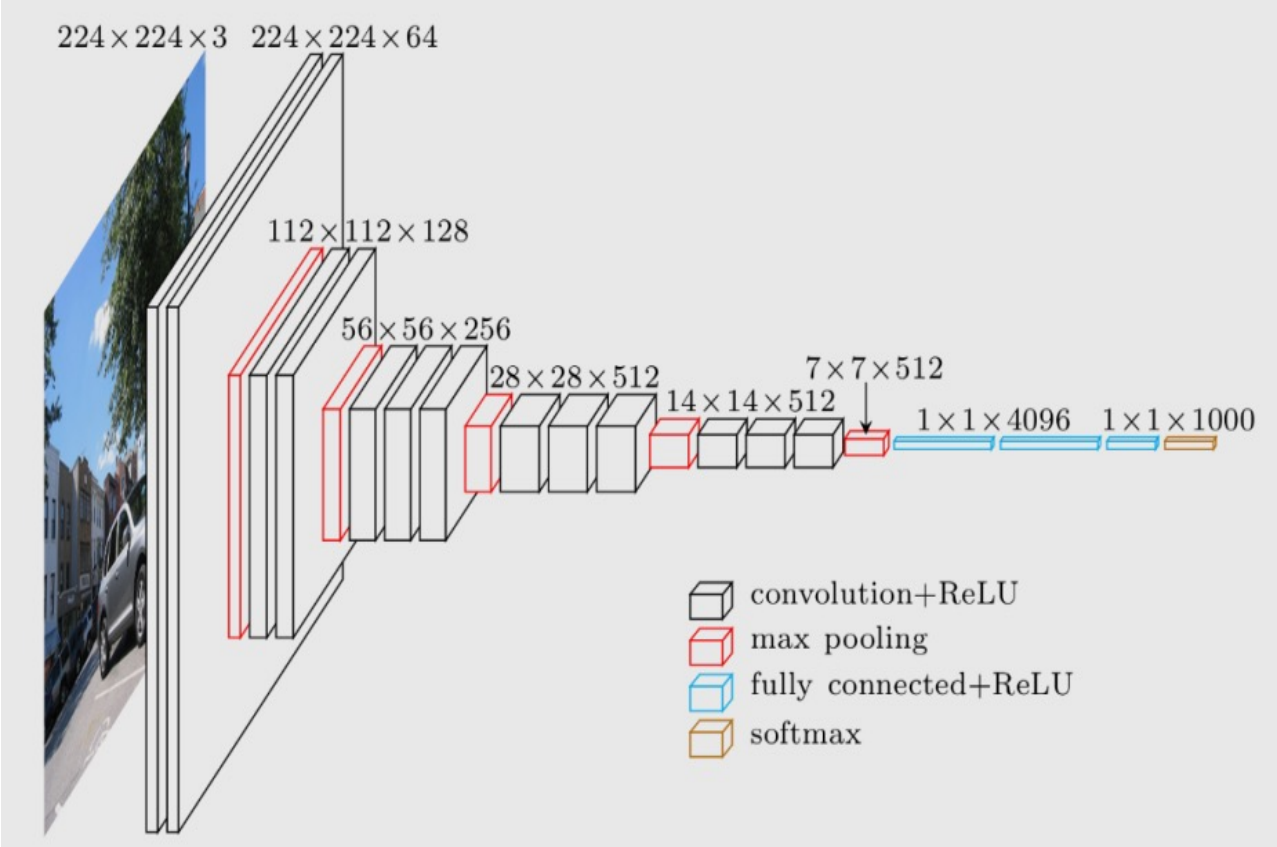
Common filter sizes found in the literature vary greatly, usually based on the dataset.

Best results on MNIST-sized images (28x28) are usually in the 5x5 range on the first layer, while natural image datasets (often with hundreds of pixels in each dimension) tend to use larger first-layer filters of shape 12x12 or 15x15.

The trick is thus to find the right level of "granularity" (i.e. filter shapes) in order to create abstractions at the proper scale, given a particular dataset.



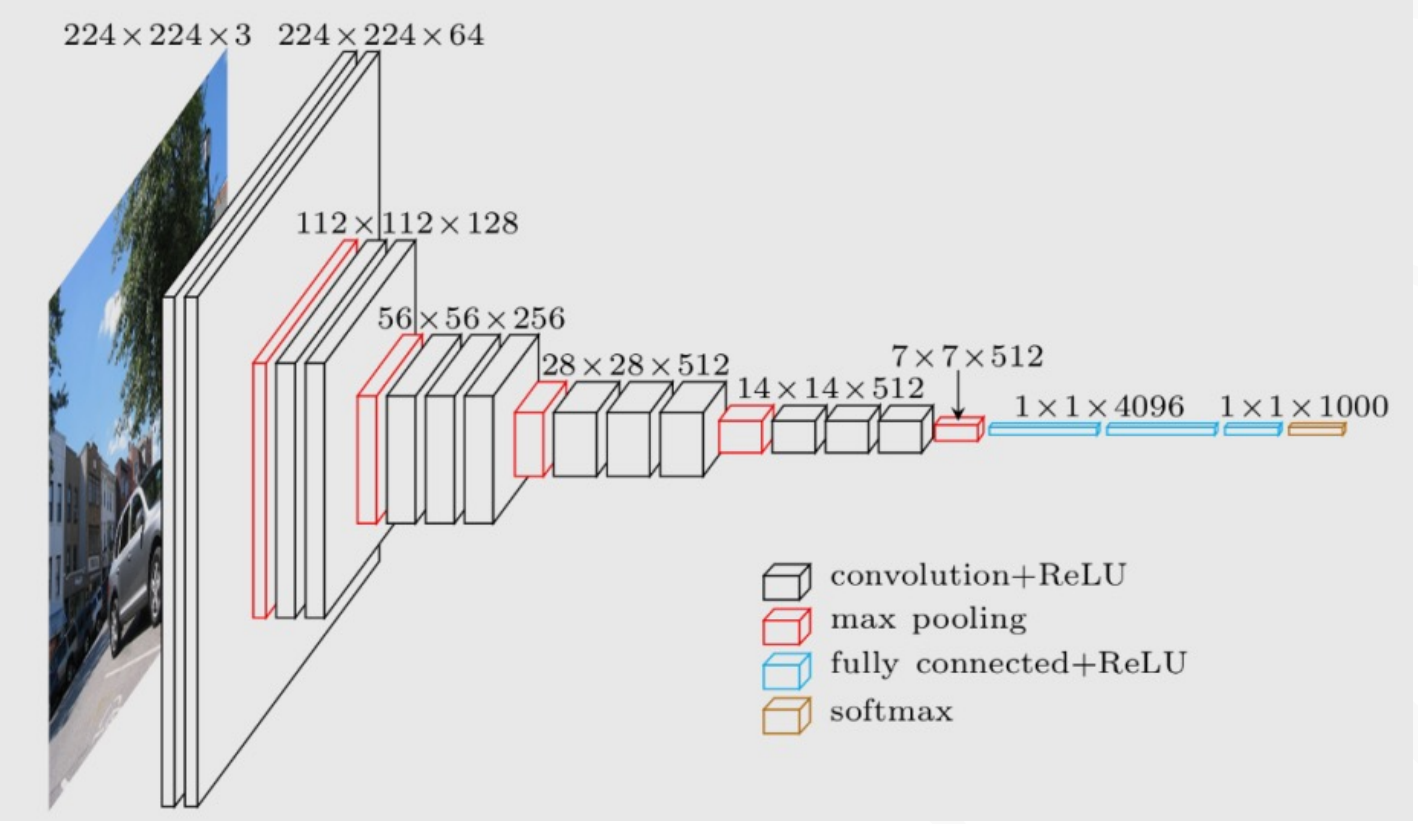
Choosing Hyper Parameters: Number of Feature Maps



- Total weights in the first conv. layer using 3x3 kernels is small
  [(3x3x3)+1]x64 = 1,728
- Number of computation in the first conv. layer is expensive
  (224x224)x(3x3x3)x64 = 86,704,128

Choosing Hyper Parameters: Number of Feature Maps and Layers

- Layers near the input layer will tend to have fewer filters while layers higher up can have much more.
- We want deeper networks, but at the same time want to reduce the number of weights in the network, to prevent overfitting.



CONVOLUTIONAL NEURAL NETWORKS

Stride, Output Volume

Stride, Zero-Padding

- Stride: How many pixels the filters are shifted spatially. Typically 1 or 2.
- Zero-padding: How many zero pixels are padded to the image to have a good/matching size of output filters.

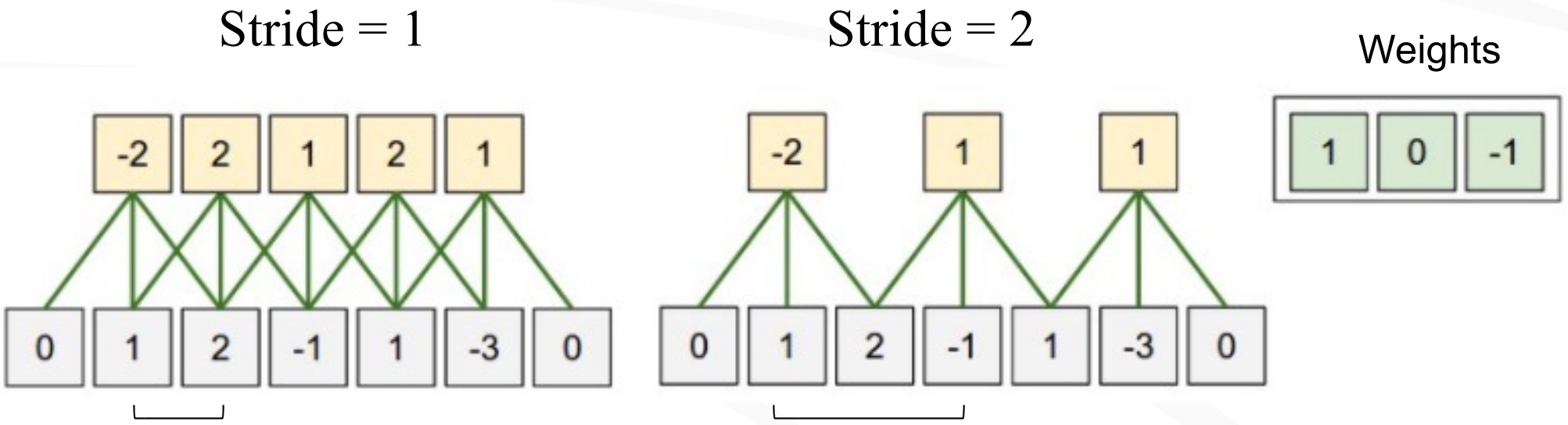
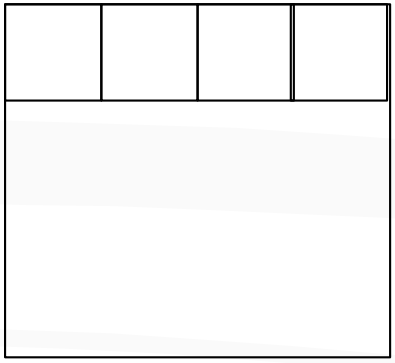


Figure from http://cs231n.github.io/convolutional-networks/

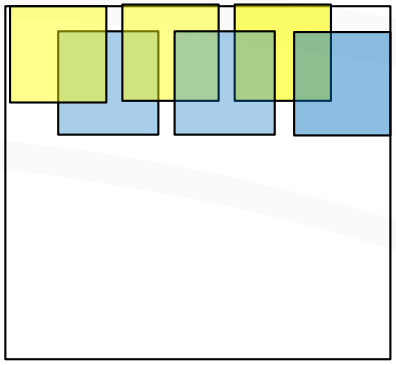
Number of Nodes for a Given Padding and Stride



Assume image width is 20, padding is 0, receptive field size is 5x5 and **stride is 5** (in this case same as the receptive field size F).

W=0, P=0, F=5, **S=5**

(W+2P-F)/S+1=(20-5)/5+1 = 4



Assume image width is 20, padding is 0, receptive field size is 5x5 and **stride is 3** (smaller than the receptive field size > overlapping receptive fields).

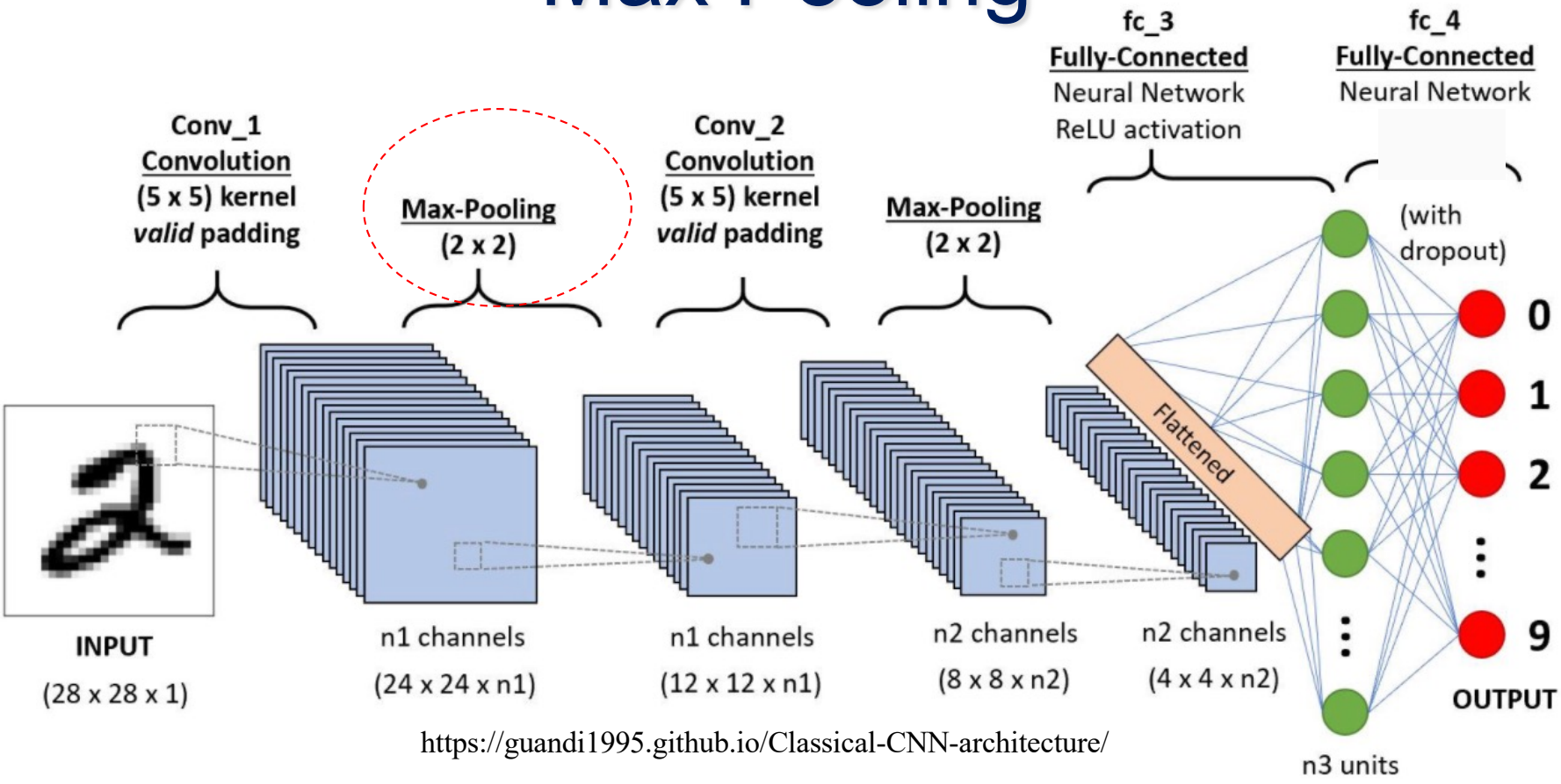
W=0, P=0, F=5, **S=5**

(W+2P-F)/S+1=(20-5)/3+1 = 6

Max pooling & Fully connected layers

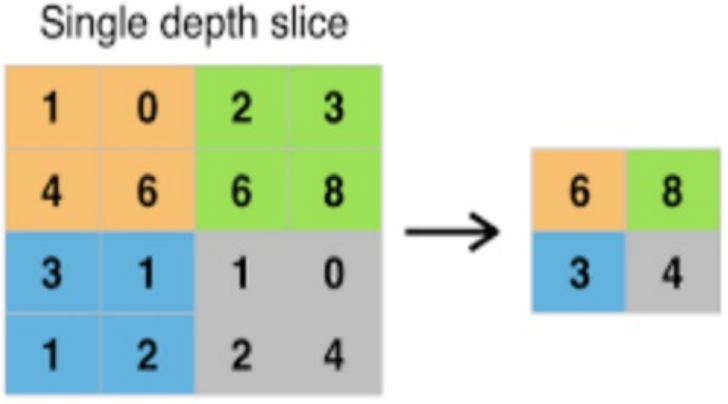
CONVOLUTIONAL NEURAL NETWORKS

Max Pooling



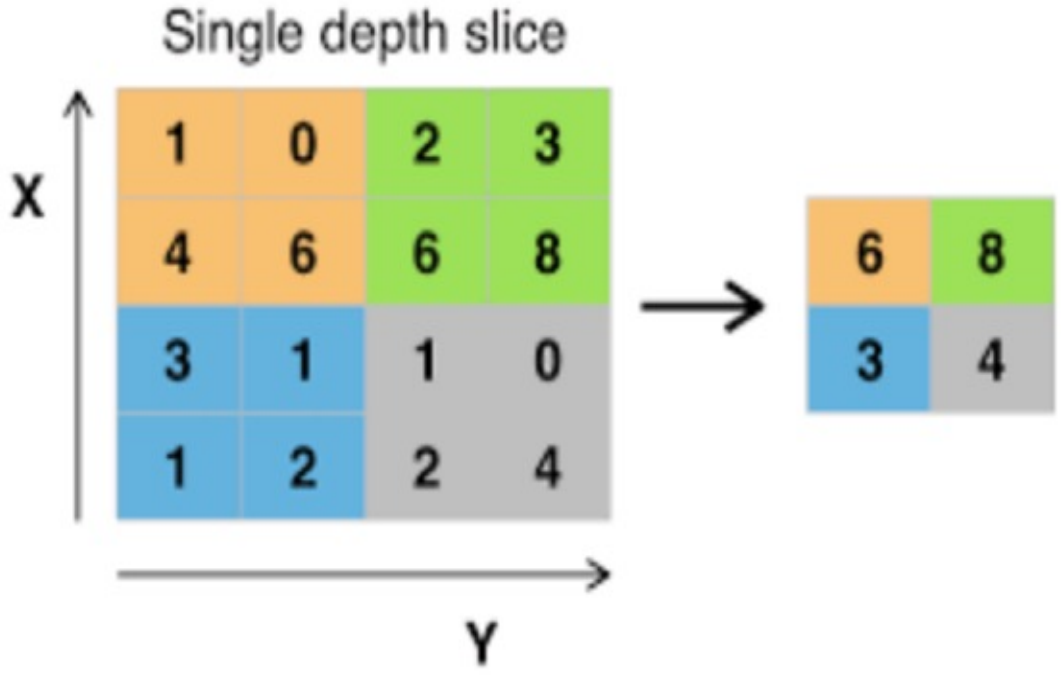
https://guandi1995.github.io/Classical-CNN-architecture/

- A node in a max-pool layer gets as input the maximum of the activation of the nodes in its receptive field.
- Non-linear down-sampling.



Max Pooling

- Non-linear down-sampling: a node in a MaxPooling layer gets as input the maximum of the activation of the nodes in its receptive field.
  - Typically 2x2 receptive fields.
- The depth of the maxpool layer is the same as previous layer since maxpool is applied on one feature map!



Max Pooling

Max-pooling is useful in vision for two reasons:

- By eliminating non-maximal values, it reduces computation for upper layers
- Provides translation invariance

- Common application of MaxPool is done on a 2x2 region with a stride of 2.
  - Very large input images may warrant 4x4 pooling in the lower-layers.
  - But this will reduce the dimension of the signal by a factor of 16, and may result in throwing away too much information!

Transfer Learning vs Fine-Tuning

Transfer learning

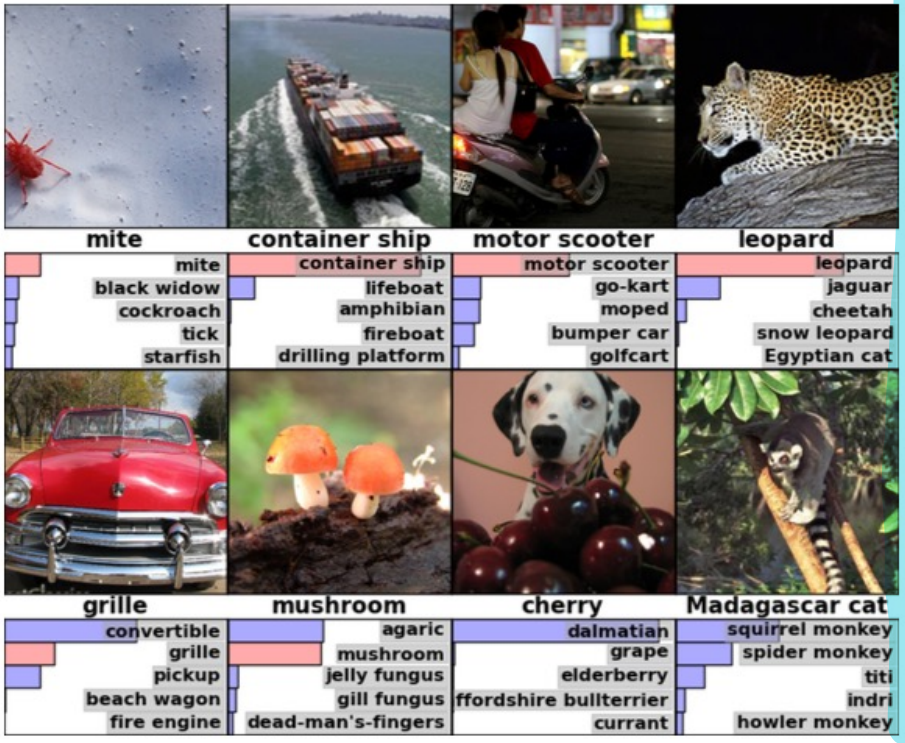
Instead of training a deep network from scratch, you can:

- take a pretrained network that is trained on a different domain (e.g. trained with ImageNet data (1000 classes))
- adapt it for your domain and your target task

ImageNet:
1.2M training images
(~ 256 x 256px)
50k validation images
1000 classes

ILSVRC-2010 images

from AlexNet paper



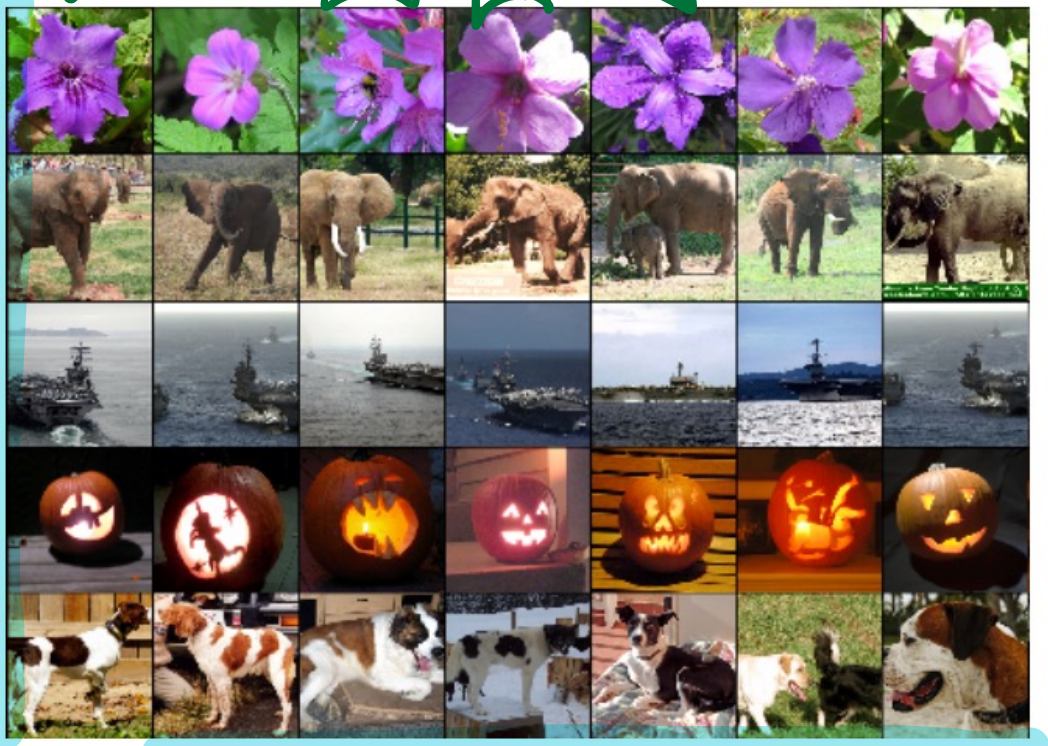
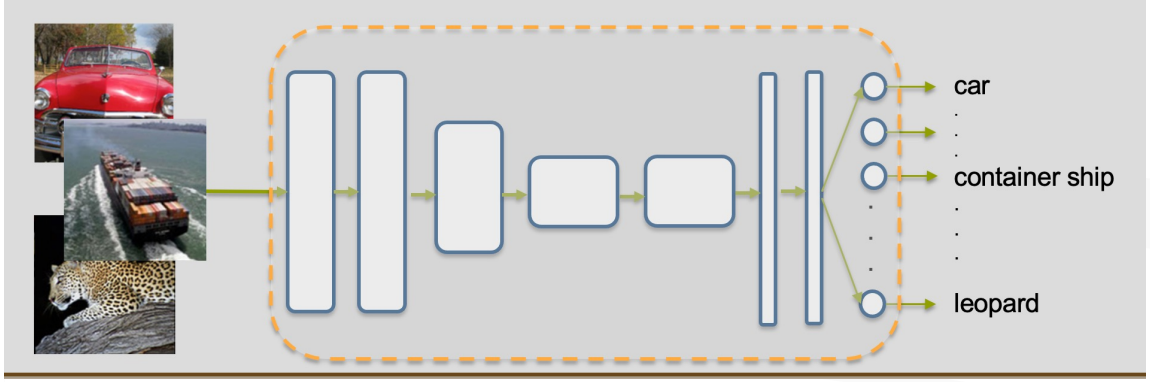
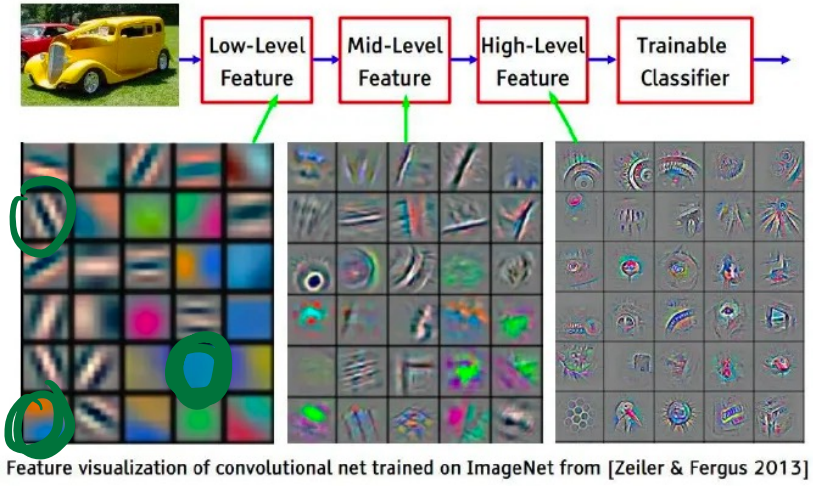


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Transfer Learning





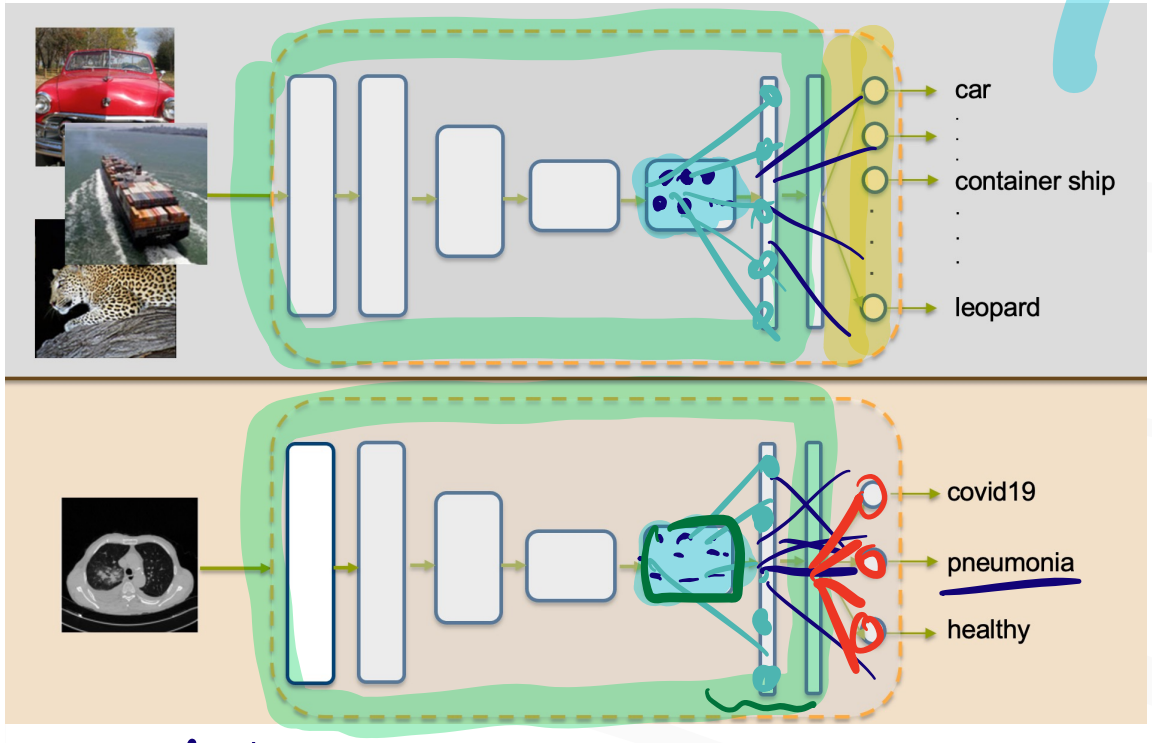
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Training deep neural networks from scratch requires A LOT of data and resources

It is possible to take a deep learning system from a similar task domain and adapt it to a new problem.

- A pre-trained deep network from a similar task domain acts as feature detector
- Change the last layer as needed and train just that layer for your own problem.
- Keep or fine-tune some of the earlier layers' weights.

Transfer Learning



Training deep neural networks from scratch requires A LOT of data and resources

It is possible to take a deep learning system from a similar task domain and adapt it to a new problem.

- A pre-trained deep network from a similar task domain acts as feature detector
- Change the last layer as needed and train just that layer for your own problem.
- Keep or fine-tune some of the earlier layers' weights.

Scratch:
- New outp. layer + weights

Fine-tune:

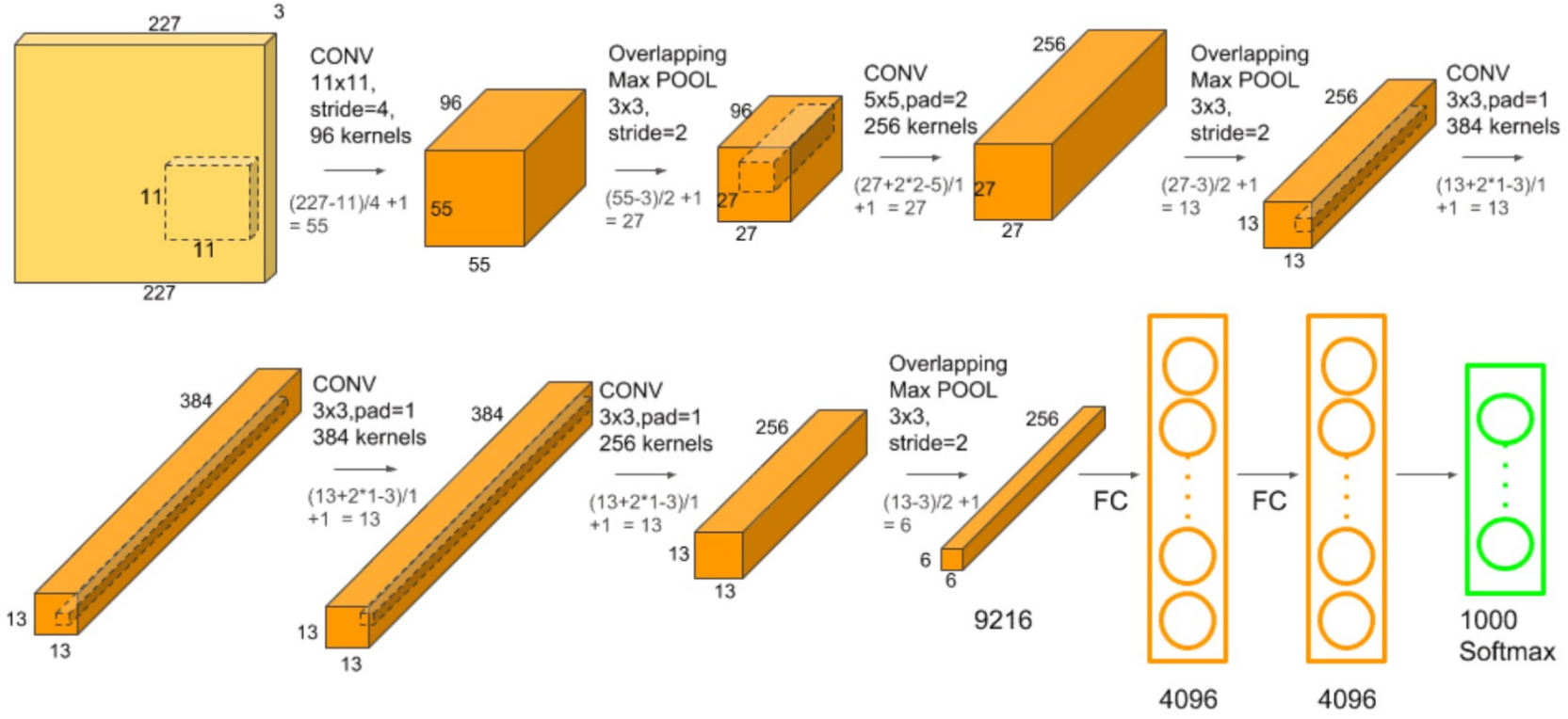
Fine-Tuning

- Take a pretrained network, cut-off and replace the last layer as before, fine-tune the network using backpropagation.
- Bottom n layers can be frozen (kept unchanged during backpropagation).
- How many layers to fine-tune depends on how much data you have for your own problem.
  - Too little data >> Learn the last layer of weights from scratch and fine-tune maybe only 1-2 layers (keeping all other weights frozen)
  - A lot of data >> Learn the last layer of weight s from scratch and fine-tune all the weights

Fully Connected Layer

Usually the last few layers are fully connected (FC).

The FC layer is connected to all nodes in the previous layers (all neurons in each feature map)



AlexNet Architecture

Data Augmentation

Training data is augmented with artificial samples, through cropping, rotation, scale, reflection, elastic deformations, intensity variations..., in order to obtain more robust systems through increased data.

- Data augmentation is done internally within frameworks and can be programmer-supplemented as well.
- Data augmentation is done to increase train data size by 10xfold or more and is very effective in reducing overfitting.

Data Augmentation

Training data is augmented with artificial samples, through cropping, rotation, scale, reflection, elastic deformations, intensity variations..., in order to obtain more robust systems through increased data.

Fig. 6. Augmentation examples (MediaId 378991)

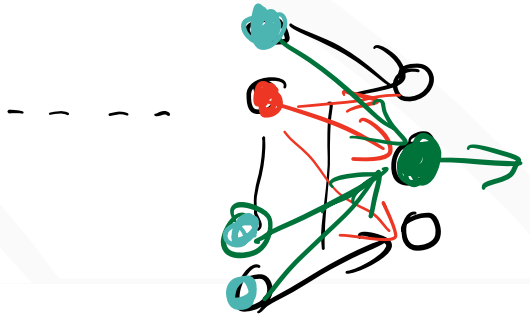
Dropout

Deep convolutional neural networks have a large number of parameters.

- If we don't have sufficiently many training examples to constrain the network, the neurons can learn the noise (idiosyncrasies) in the data.
- Regularization puts constrains on a learning system to reduce overfitting (e.g. penalizing for large weight magnitudes in NNs, ...).

Another novel idea in deep learning is dropout, whereby some nodes are “dropped out” during the network training with a preset probability.

- Network learns to cope with failures.
- Sampling from an ensemble of deep networks, to harvest benefits of ensembles.
- Must scale weights during testing.



Stopped here

TRAINING CONVOLUTIONAL NEURAL NETWORKS

Mini-Batch

Gradient descent normally can be done in:

- batch mode (more accurate gradients) or
- stochastic fashion (much faster learning with large amounts of training data).

In deep learning, we use mini-batches, which is the set of training images from which the gradient is calculated, before a single update.

- Mini-batch size should be as high as possible, as your computer allows (20-256 are typical)

Learning Rate

- Large learning rates may cause divergence
- Reduce in time
- Batch normalization helps decide the learning rate for different layers
- ...

Rest Not included in exams

Batch Normalization

Use this - good a little advanced -

From Ioffe and Szegedy, 2015:

- The distribution of each layer's inputs changes during training, as the parameters of the previous layers change (internal covariate shift)
- This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it difficult to train models with saturating nonlinearities.
- Make normalization a part of the model architecture and perform normalization of node activations for each training mini-batch.
  - Normalization applied to each feature dimension independently
  - Same accuracy with 14x fewer iterations.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

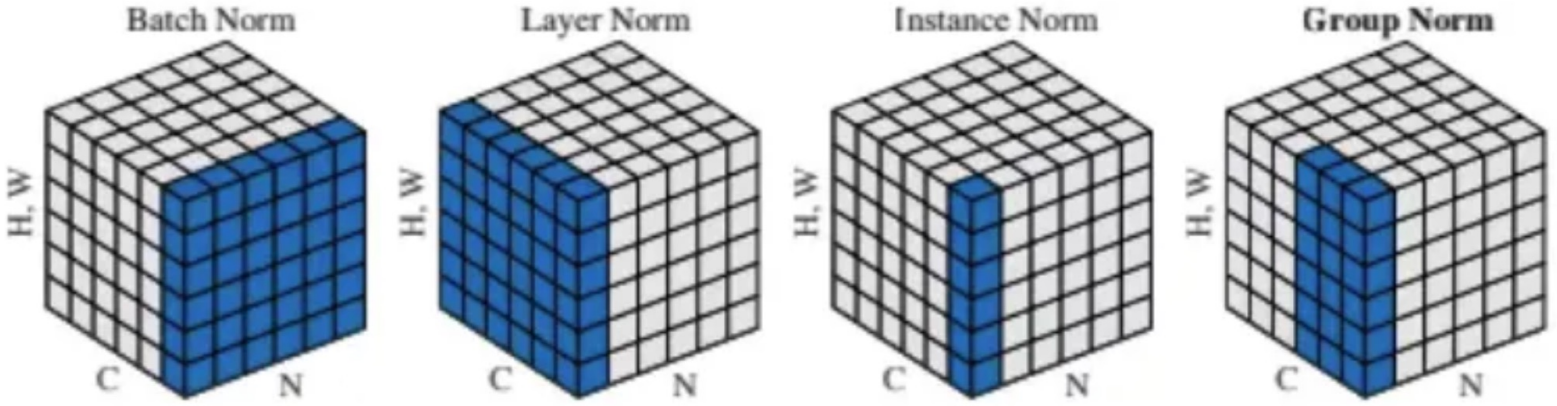
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

https://arxiv.org/pdf/1502.03167.pdf

See also Batch Renormalization, 2017



A visual comparison of various normalization methods

This image is taken from https://arxiv.org/pdf/1803.08494.pdf

Further reading (where some of the content and images are taken from):

- deeplearning.net
- http://cs231n.github.io/convolutional-networks/