COMP 9517 Computer Vision

Deep Learning for Computer Vision

Challenges in CV

Consider object detection as an example:

- Variations in viewpoint
- Differences in illumination
- Hidden parts of images
- Background clutter

Content: link

A bit of History

- Earliest studies about visual mechanics of animals by Hubel and Weisel emphasized the importance of edge detection for solving CV problems
- They found that early processing in cat visual cortex looks like it is performing convolutions that are looking for oriented edges and blobs
- Certain cells are looking for edges with a particular orientation at a particular spatial location in the visual field
- This inspired convolutional neural networks but was limited by the lack of computational power
- Hinton et al. reinvigorated research into deep learning and proposed a greedy layer wise training technique
- In 2012, Alex Krizhevesky et al. won Imagenet challenge and proposed the now well recognized AlexNet Convolutional Neural Network architecture

Deep Learning

- Deep learning is a collection of artificial neural network techniques that are widely used at present
- Predominantly, deep learning techniques rely on large amounts of data and deeper learning architectures
- Some well known paradigms:
  - **Convolutional Neural Networks (CNNs)**
  - Recurrent Neural Networks
  - Auto-encoders
  - Restricted Boltzmann Machines
- Some well known applications of deep learning in CV include Object recognition – ImageNet, Context Analysis, Semantic Segmentation, Classification, Retrieval, Self-driving cars, ….

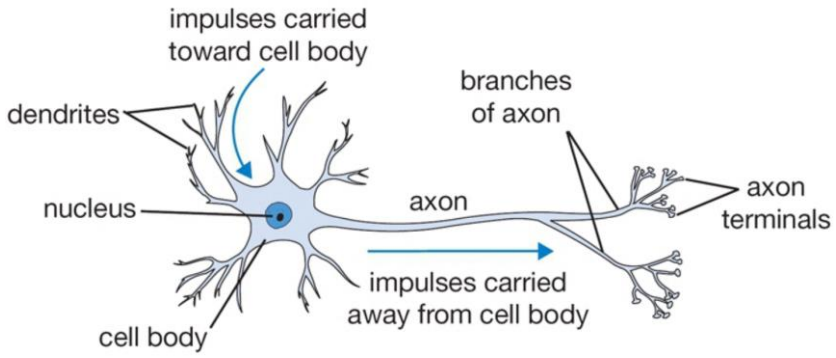
CNNs

- CNNs are very similar to regular neural nets
  - Made up of neurons with learnable weights
- CNN architecture assumes that inputs are images
  - This allows us to encode certain properties in the architecture that makes the forward pass more efficient and significantly reduces the number of parameters needed for the network

link

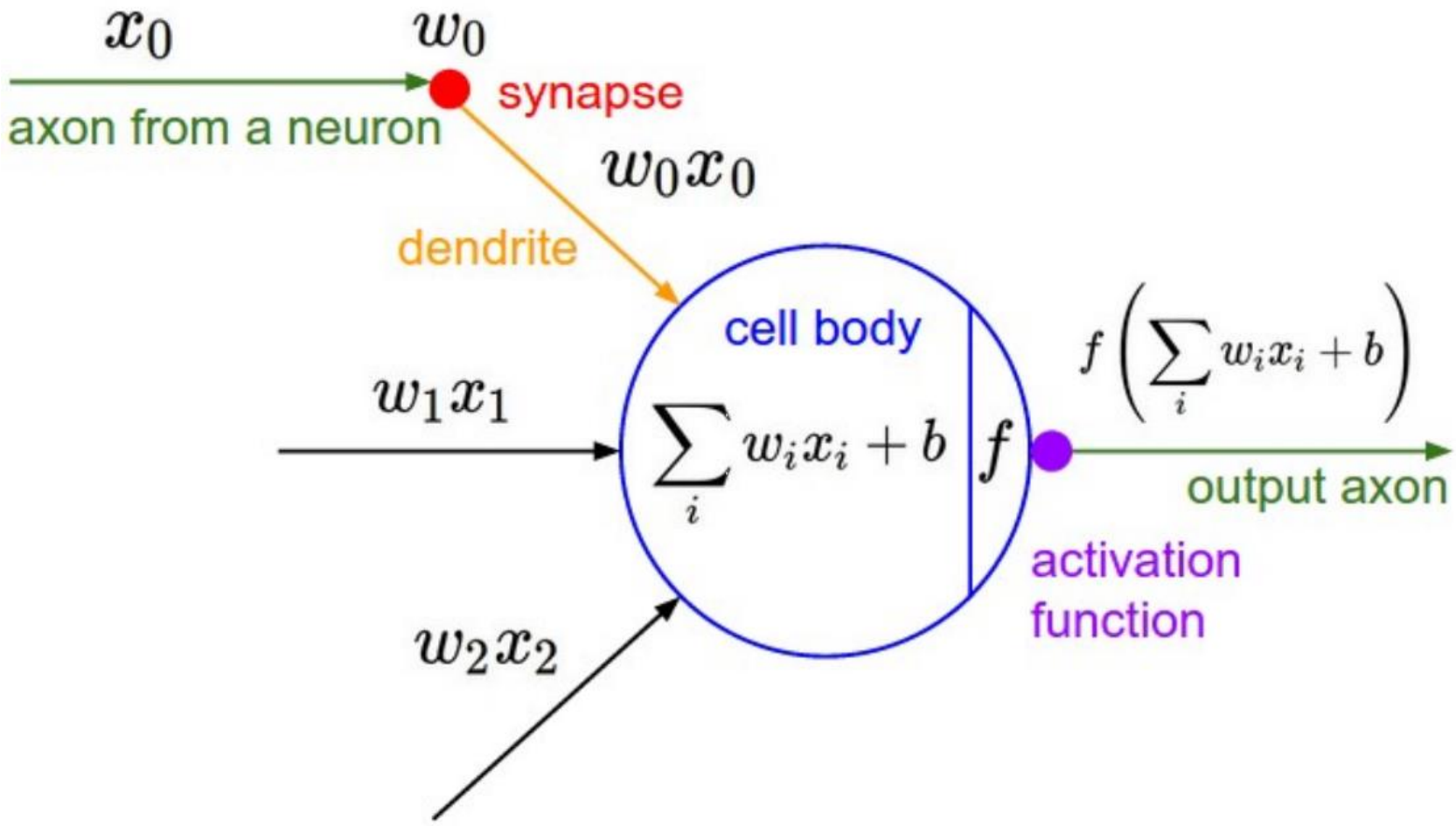
Neural Networks

- Neural Networks are inspired by human nervous system
- The biological interpretation of a perceptron: when it emits a 1 it is equivalent to ‘firing’ an electrical pulse, and when it is 0, it is not firing. The bias indicates how difficult it is for this particular node to send out a signal
- NNs are composed of a large number of interconnected processing elements known as neurons
- They use supervised error correcting rules with back-propagation to learn a specific task



http://statsmaths.github.io/stat665/lectures/lec12/lecture12.pdf

NN: Perceptron



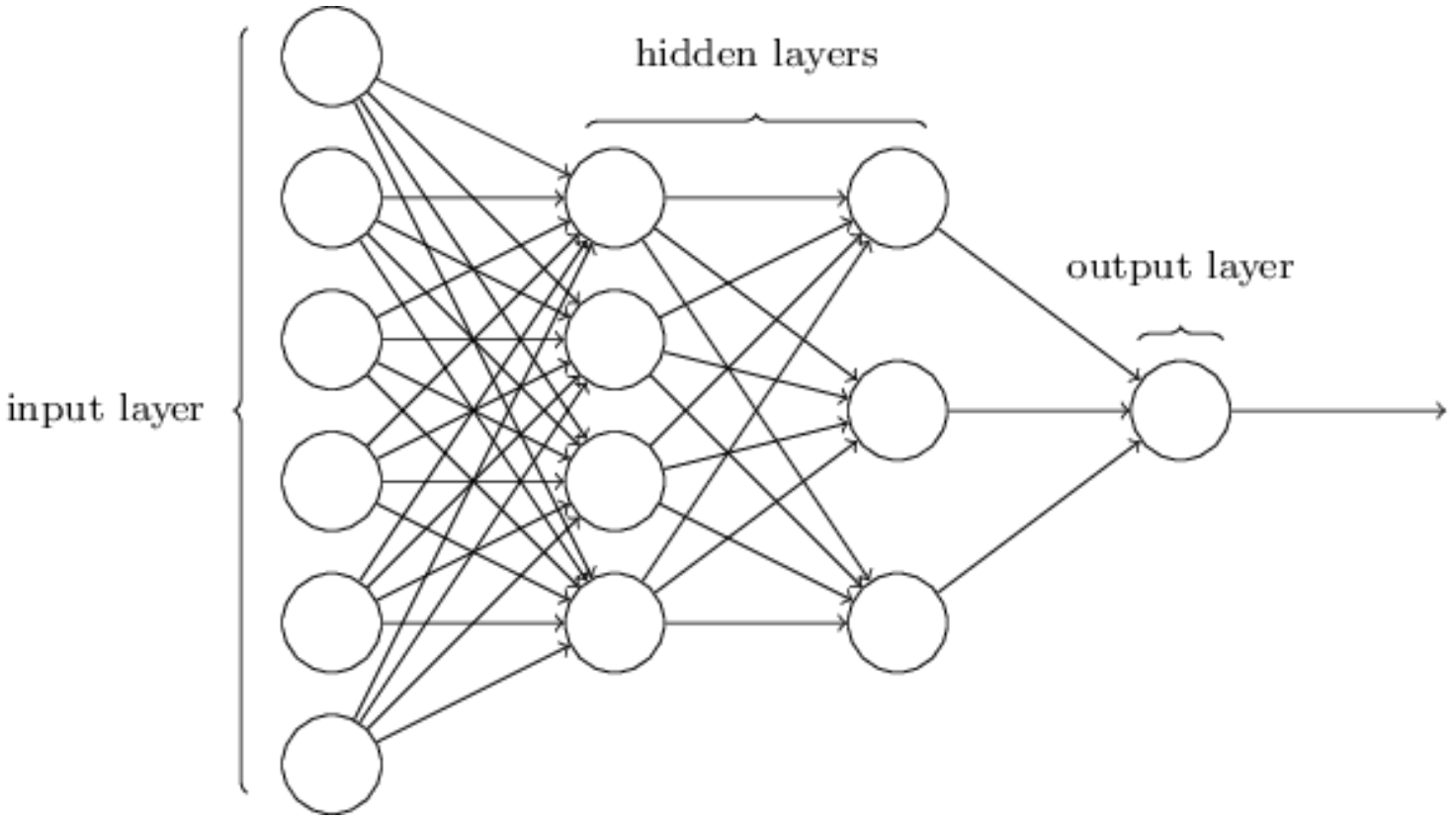
NN: Sigmoid Neuron

- An important shortcoming of a perceptron is that a small change in the input values can cause a large change in the output because each node (or neuron) only has two possible states: 0 or 1.
- A better solution would be to output a continuum of values, say any number between 0 and 1.
- As one option, we could simply have the neuron emit the value:

$$\sigma(x \cdot w + b) = \frac{1}{1 + e^{-(x \cdot w + b)}}$$

- This is known as a sigmoid neuron and uses sigmoid (logit) activation function

NN: Example Architecture



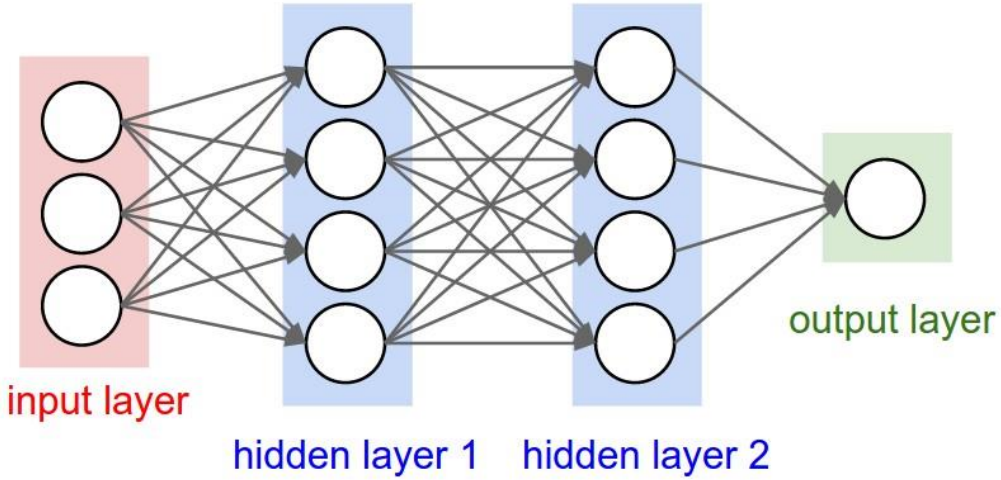
Why CNNs?

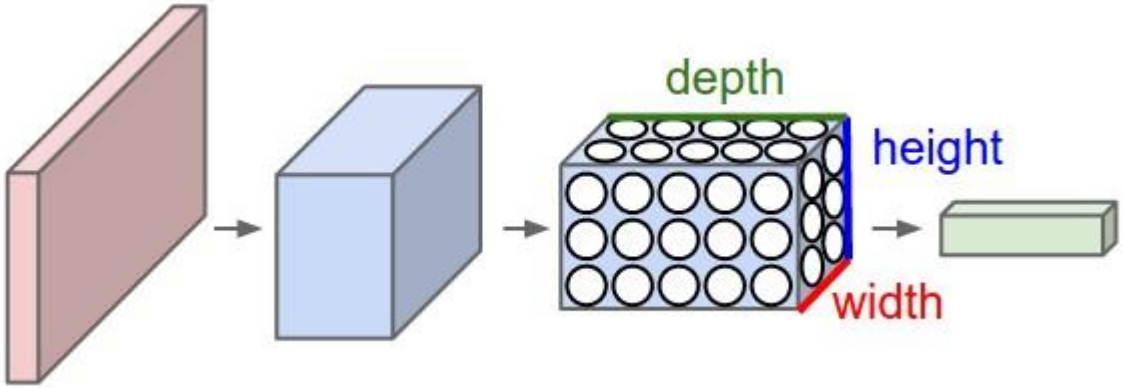
- The problem with regular NNs is that they do not scale well with dimensions (ie: larger images)
  - Eg: 32x32 image with 3 channels (RGB) – a neuron in first hidden layer would have 32x32x3 = 3072 weights : manageable.
  - Eg: 200x200 image with 3 channels – a neuron in first hidden layer would have 200x200x3 = 120000 weights and we need at least several of these neurons which makes the weights explode.

So what is different?

- In contrast, CNNs consider 3-D volumes of neurons and propose a parameter sharing scheme that minimizes the number of parameters required by the network.
- CNN neurons are arranged in 3 dimensions: Width, Height and Depth.
- Neurons in a layer are only connected to a small region of the layer before it (hence not fully connected)

So what is different?

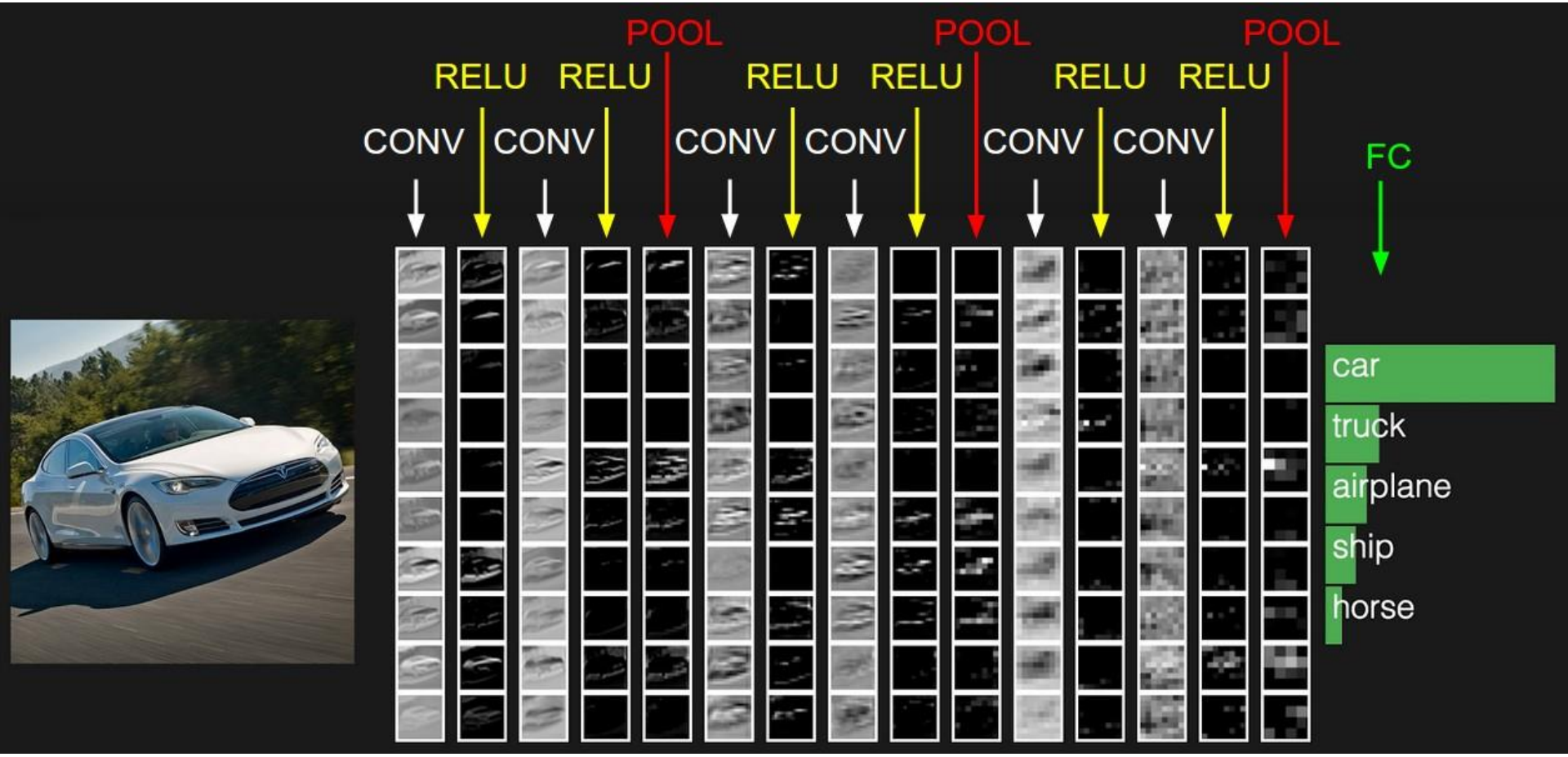




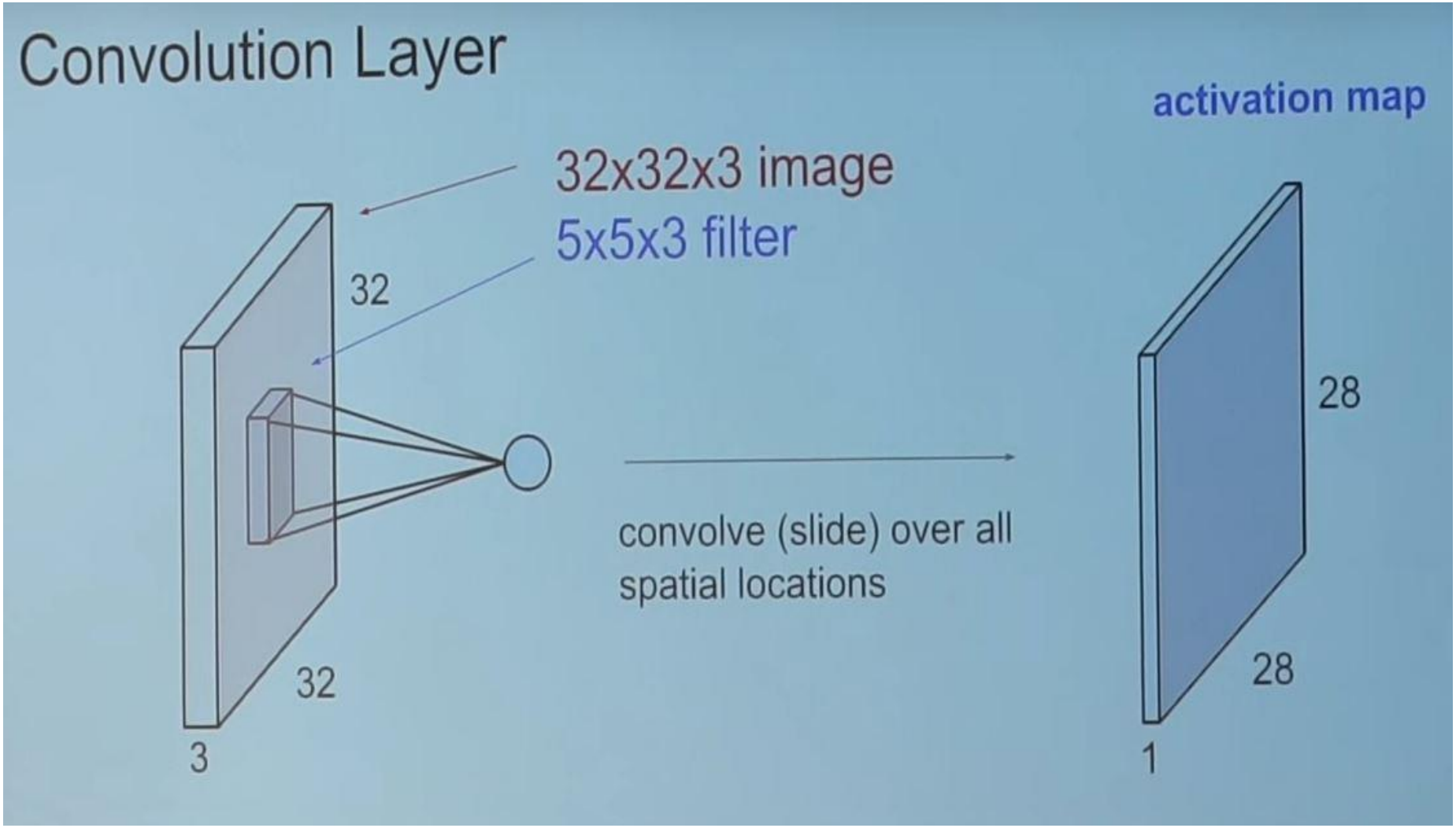
CNN Architecture

- CNNs are usually made up of three major types of layers
  - Convolution Layer
  - Pooling Layer
  - ReLu Layer, and
  - Fully-Connected Layer
- These layers are then stacked to form a full CNN architecture

CNN: Architecture



CNN: Convolutional Layer



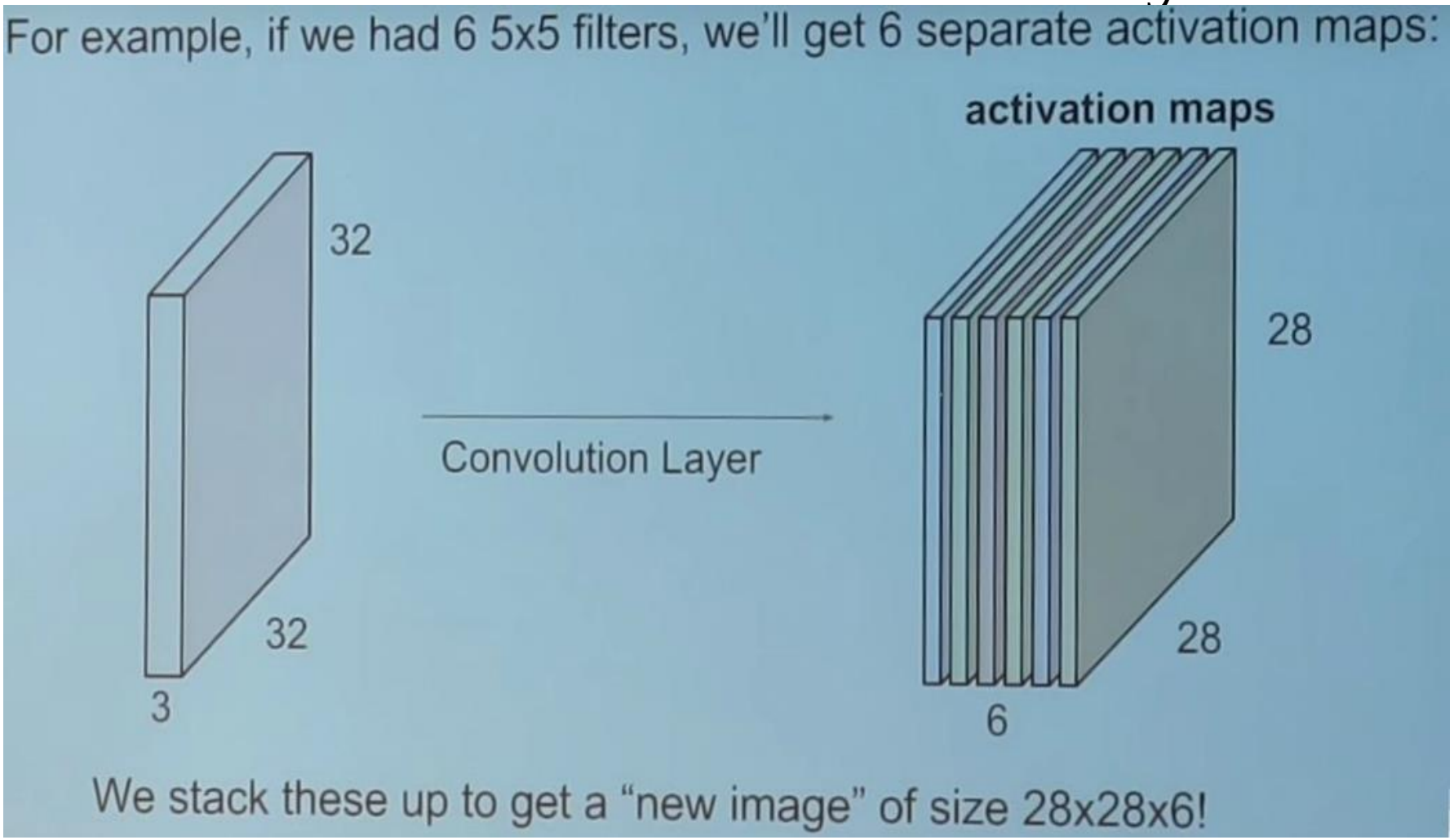
CNN: Convolutional Layer

- The output of the Conv layer can be interpreted as holding neurons arranged in a 3D volume.
- The Conv layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume.
- During the forward pass, each filter is slid (convolved) across the width and height of the input volume, producing a 2-dimensional activation map of that filter.
- Network will learn filters that activate when they see some specific type of feature at some spatial position in the input.

CNN: Convolutional Layer

- Stacking these activation maps for all filters along the depth dimension forms the full output volume
- Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at only a small region in the input and shares parameters with neurons in the same activation map (since these numbers all result from applying the same filter)

CNN: Convolutional Layer



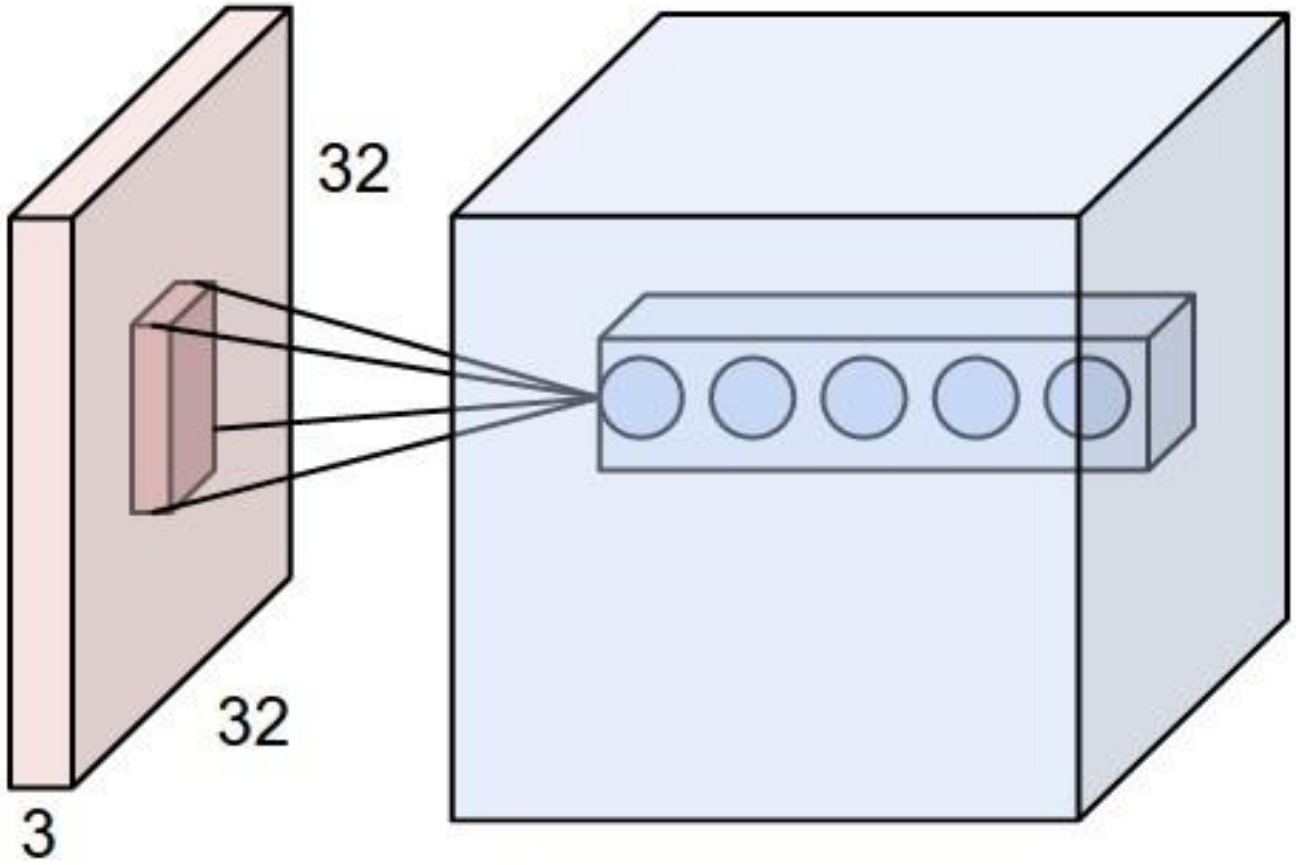
CNN: Convolutional Layer – Local Connectivity

- As we have realized by now, it is impractical to use fully connected networks when dealing with high dimensional images/data
- Hence the concept of local connectivity: each neuron only connects to a local region of the input volume.
- The spatial extent of this connectivity is a hyperparameter called ***receptive field*** of the neuron.
- The extent of the connectivity along the depth axis is always equal to the depth of the input volume.
- The connections are local in space (along width and height), but always full along the entire depth of the input volume.

CNN: Convolutional Layer – Local Connectivity

- Eg1: Suppose that the input volume has size [32x32x3]. If the receptive field is of size 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights. Notice that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.
- Eg 2: Suppose an input volume had size [16x16x20]. Then using an example receptive field size of 3x3, every neuron in the Conv Layer would now have a total of 3*3*20 = 180 connections to the input volume. Notice that, again, the connectivity is local in space (e.g. 3x3), but full along the input depth (20).

CNN: Convolutional Layer – Local Connectivity



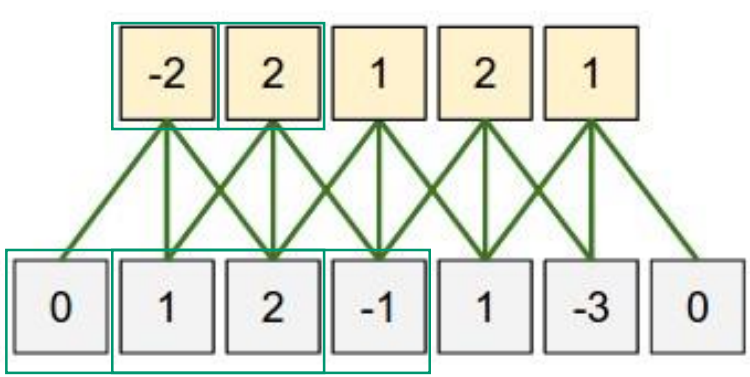
CNN: Convolutional Layer – Spatial Arrangement

- Three hyperparameters control the size of the output volume: the ***depth, stride*** and ***zero-padding***
- ***Depth*** controls the number of neurons in the Conv layer that connect to the same region of the input volume
- ***Stride*** is the distance that the filter is moved by in spatial dimensions
- ***Zero-padding*** is padding of the input with zeros spatially on the border of the input volume

CNN: Convolutional Layer – Spatial Arrangement

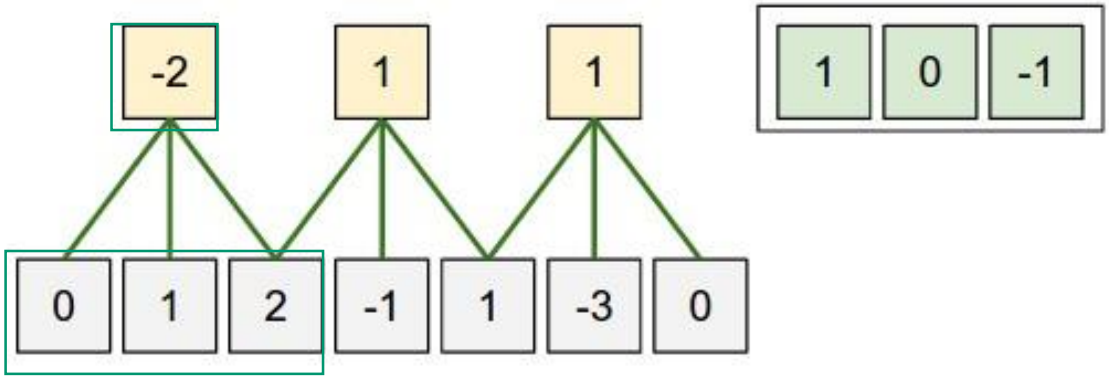
- We can compute the spatial size of the output volume as a function of the input volume size (W), the receptive field size of the Conv Layer neurons (F), the stride with which they are applied (S), and the amount of zero padding used (P) on the border.
- (W−F+2P)/S+1 gives the output volume dimensions. If this number is not an integer, then the strides are set incorrectly and the neurons cannot be tiled so that they "fit" across the input volume neatly, in a symmetric way.

CNN: Convolutional Layer – Spatial Arrangement



W=5 (Input Size)

P=1 (zero-padding level 1)



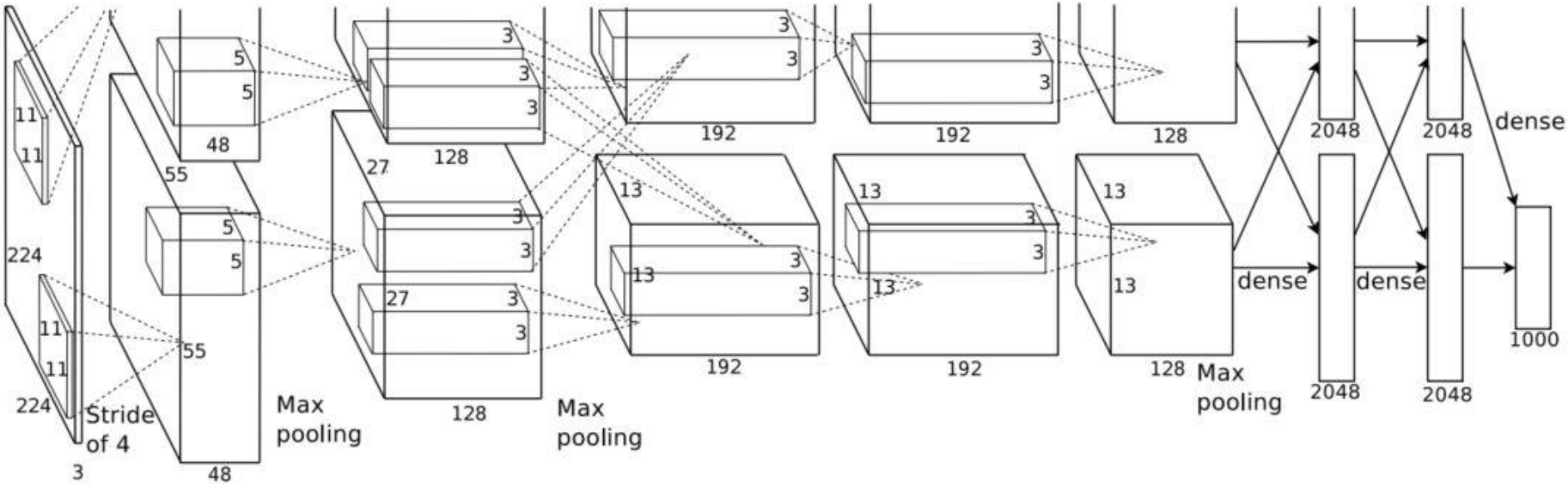
W=5 (Input Size)

P=1 (zero-padding level 1)

CNN: Convolutional Layer – Spatial Arrangement

- Real-world example. The Krizhevsky et al. architecture that won the ImageNet challenge in 2012 accepted images of size 227x227x3
- On the first Convolutional Layer, it used neurons with receptive field size F=11, stride S=4 and no zero padding P=0.
- Since (227 - 11)/4 + 1 = 55, and since the Conv layer had a depth of K=96, the Conv layer output volume had size 55x55x96
- Each of the 55*55*96 neurons in this volume was connected to a region of size 11x11x3 in the input volume
- Moreover, all 96 neurons in each depth column are connected to the same 11x11x3 region of the input, but of course with different weights

CNN: Convolutional Layer – Spatial Arrangement



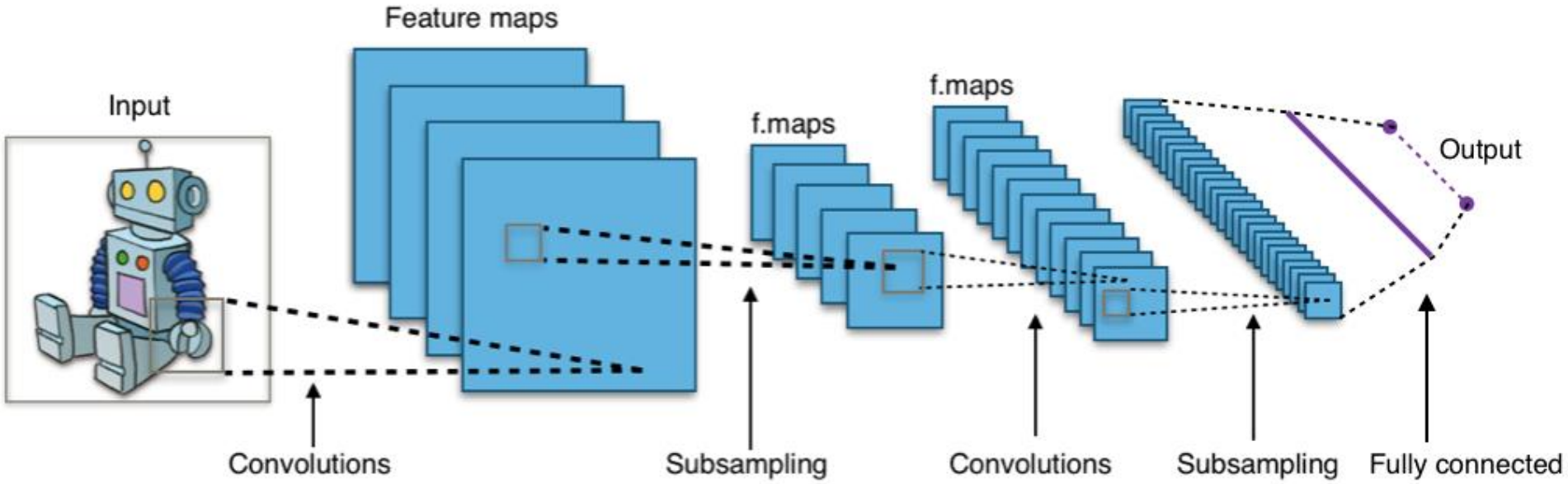
CNN: Convolutional Layer – Parameter Sharing

- Parameter sharing scheme used in Convolutional Layers to control the number of parameters
- In the real-world example above, there are 55*55*96 = 290,400 neurons in the first Conv Layer, and each has 11*11*3 = 363 weights and 1 bias.
- Together, this adds up to 290400 * 364 = 105,705,600 parameters on the first layer of the ConvNet alone. Clearly, this number is very high
- We assume that if one patch feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2).

CNN: Convolutional Layer – Parameter Sharing

- In other words, denoting a single 2-D slice as a depth slice (e.g. a volume of size [55x55x96] has 96 depth slices, each of size [55x55]), we are going to constrain the neurons in each depth slice to use the same weights and bias- this is exactly what we do with spatial filters for signals/images!
- With this parameter sharing scheme, the first Conv Layer in our example would now have only 96 unique sets of weights (one for each depth slice), for a total of 96*11*11*3 = 34,848 unique weights, or 34,944 parameters (+96 biases).

CNN: Convolutional Layer – Parameter Sharing



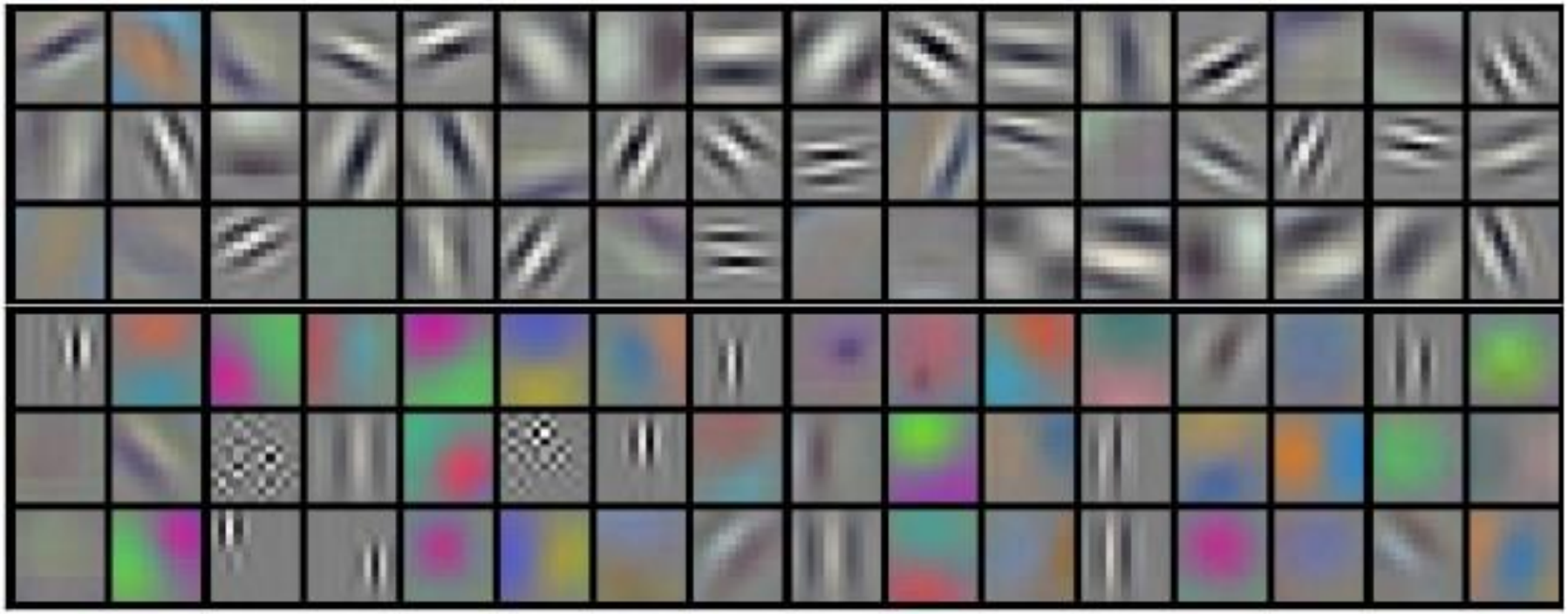
CNN: Convolutional Layer – Parameter Sharing

- Alternatively, all 55*55 neurons in each depth slice will now be using the same parameters.
- In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.
- If all neurons in a single depth slice are using the same weight vector, then the forward pass of the Conv layer can in each depth slice be computed as a **convolution** of the neuron's weights with the input volume (Hence the name: Convolutional Layer).

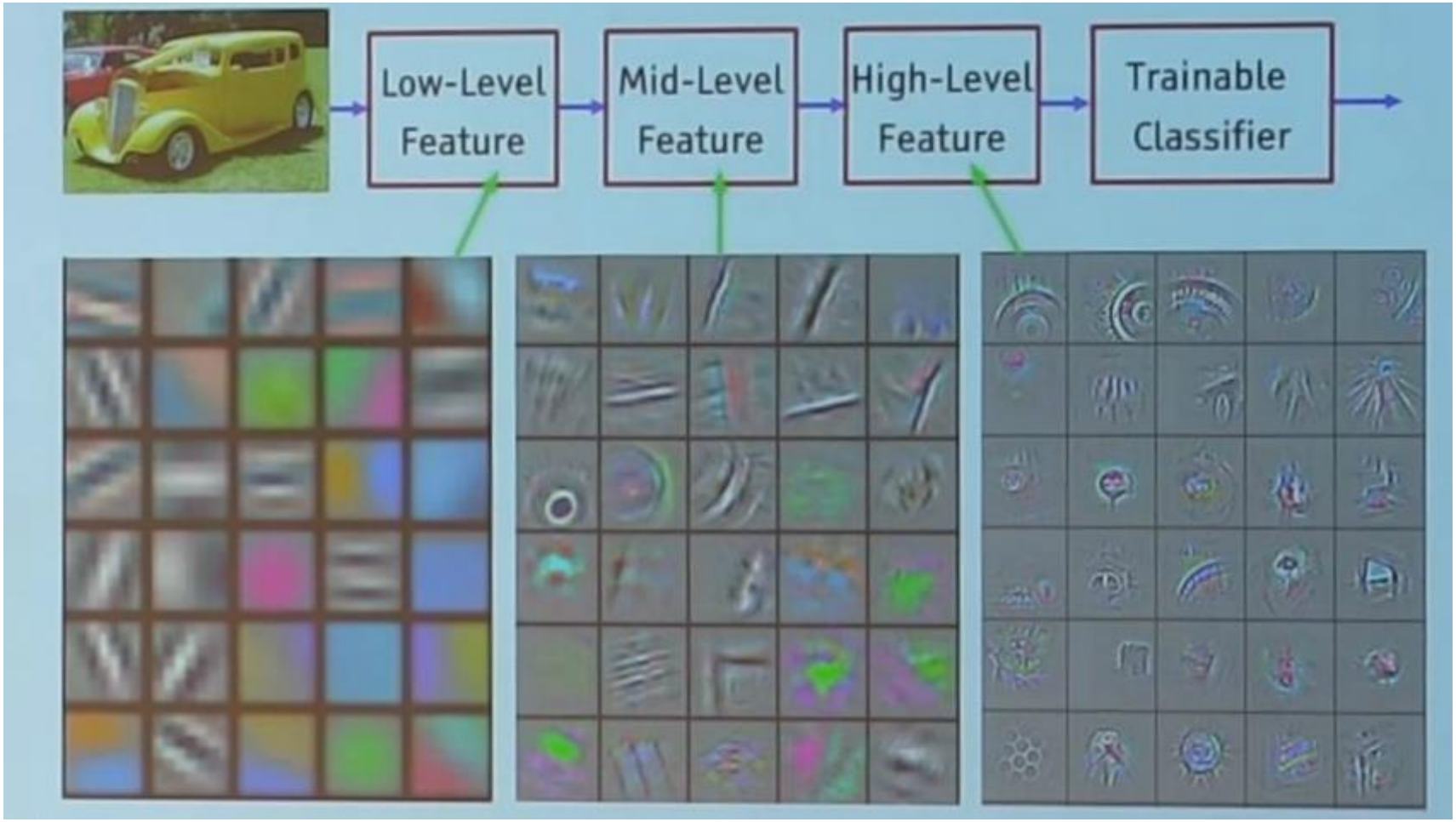
CNN: Convolutional Layer – Parameter Sharing

- Therefore, it is common to refer to the sets of weights as a filter (or a kernel), which is convolved with the input
- The result of this convolution is an *activation map* (eg. of size 55x55), and the activation maps for each different filter are stacked together along the depth dimension to produce the output volume (e.g. 55x55x96)

CNN: Convolutional Layer – Parameter Sharing



CNN: Convolutional Layer – Parameter Sharing



CNN: Convolutional Layer – Summary

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size W2×H2×D2 where:
  - $W_2=(W_1-F+2P)/S+1$
  - $H_2=(H_1-F+2P)/S+1$
  - $D_2=K$

CNN: Convolutional Layer – Summary

- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

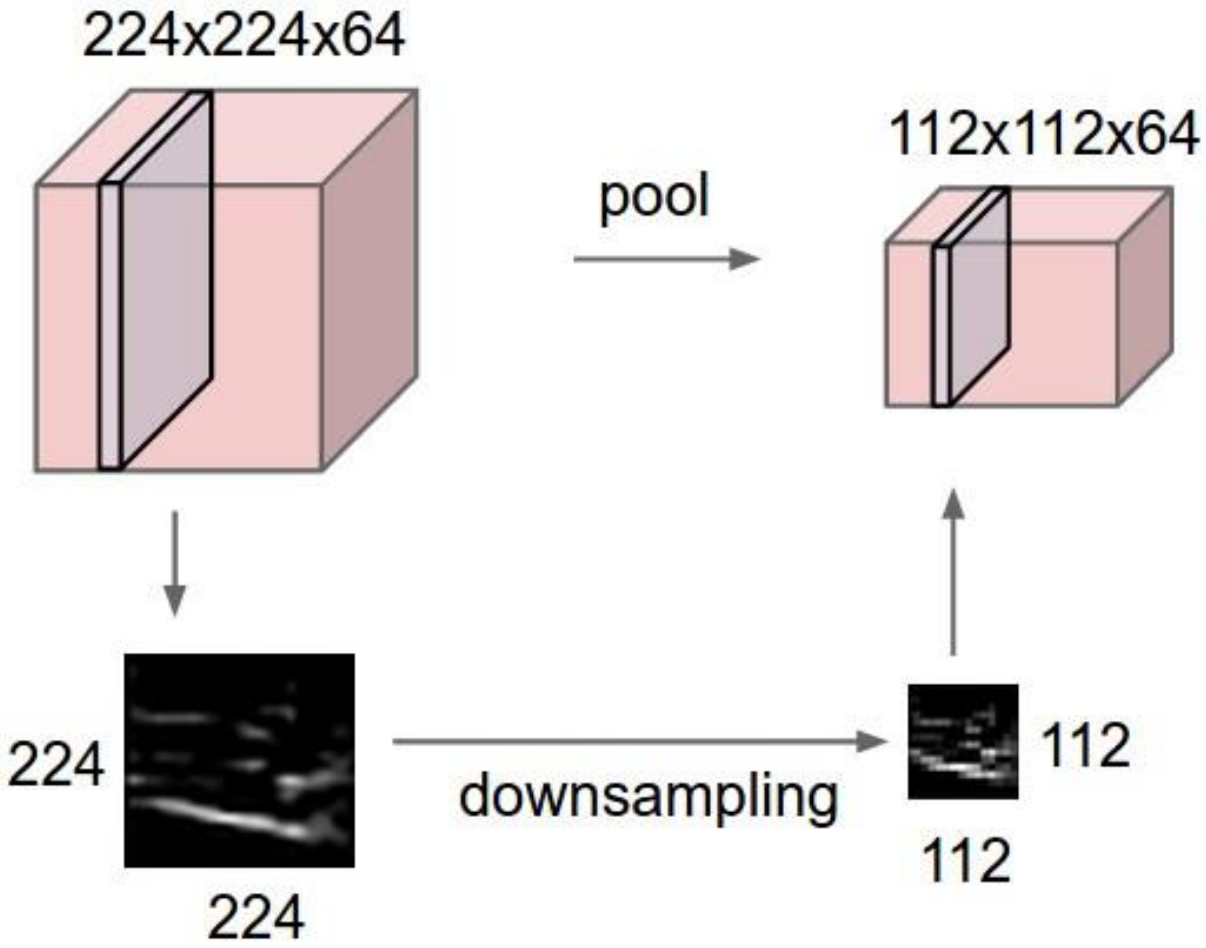
CNN: Convolutional Layer – Demo

- http://cs231n.github.io/convolutional-networks/#conv
- http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

CNN: Pooling Layer (or Spatial Downsampling)

- The function of pooling layer is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and hence to also control overfitting
- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation (ie: max pooling)
- The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2, which downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations

CNN: Pooling Layer

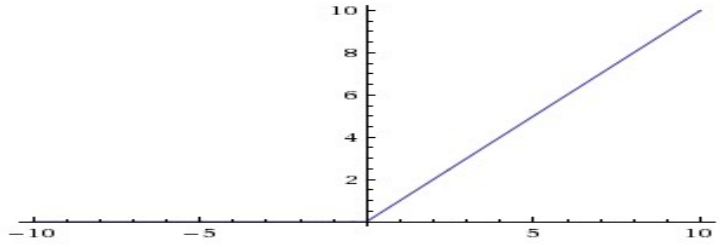


CNN: Pooling Layer - Summary

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$where:
  - $W_2=(W_1-F)/S+1$
  - $H_2=(H_1-F)/S+1$
  - $D_2=D_1$
- Introduces zero parameters since it computes a fixed function of the input

CNN: ReLu Layer

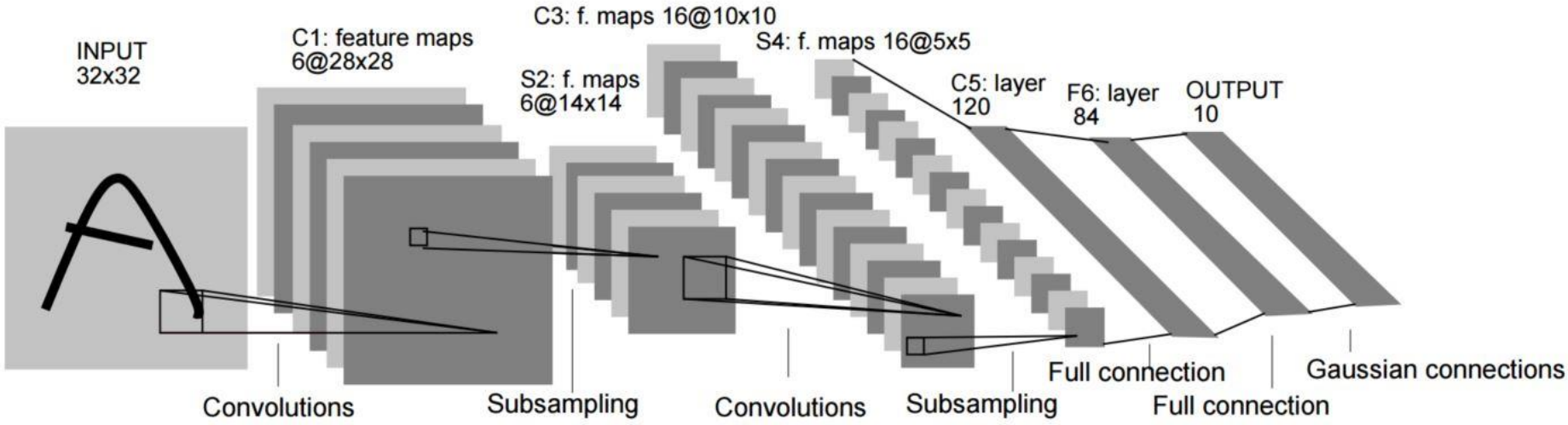
- Although ReLu (**Re**ctified **L**inear **U**nit) is considered as a layer, it is really an activation function
- It computes the function *f(x) = max(0, x)*.
- This is favoured in deep learning as opposed to the traditional activation functions like Sigmoid or Tanh, because it was found to accelerate the convergence of stochastic gradient descent. They are also computationally inexpensive compared to traditional ones.
- However, ReLu units can be fragile during training and ‘die’. Leaky ReLus were proposed to handle this problem.



CNN: Fully-Connected Layer

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

CNN: Example

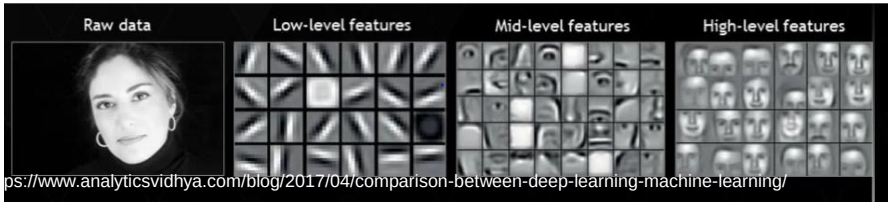


LeCun et al, 1998

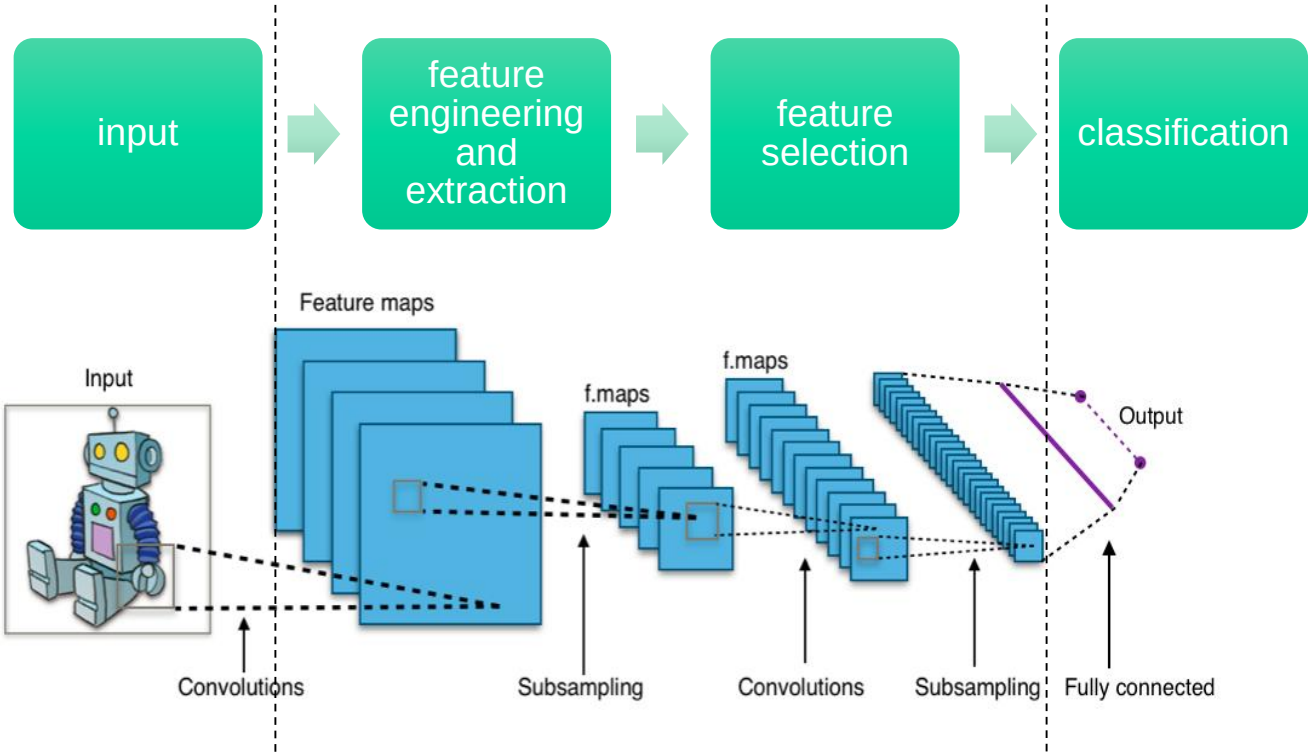
Traditional Approach vs DL

CNNs can be interpreted as gradually transforming the images into a representation in which the classes are separable by a linear classifier.

CNNs will try to learn low-level features such as edges and lines in early layers, then parts of faces of people and then high-level representation of a face in subsequent layers.



Traditional Approach vs DL

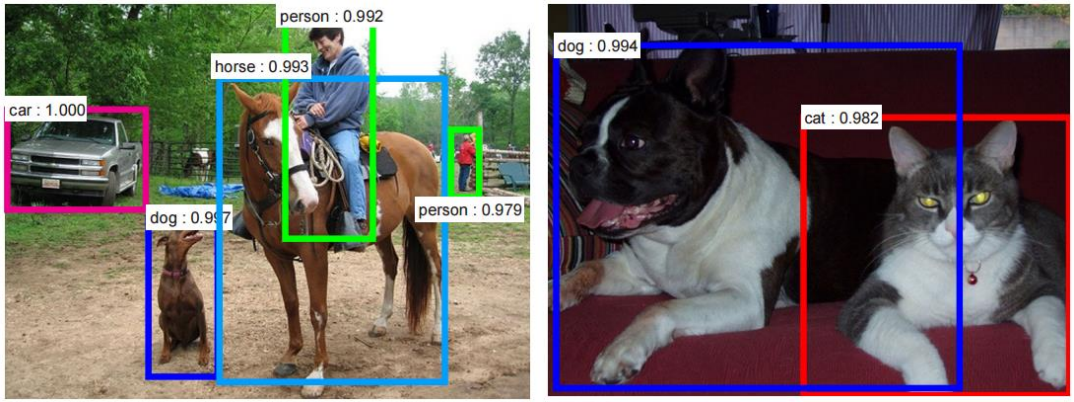


State-of-the-art

- Object Recognition
  - AlexNet (2012)
  - GoogLeNet
  - VGGNet
  - ResNet
  - Inception v3/v4
  - DenseNets is the current state-of-the-art
- Semantic Segmentation
  - Multi-scale CNN (2012)
  - U-net / V-net
  - U-net / V-net with skip/dense connections
- Adversarial
  - Generative Adversarial Networks (2014)

Object Recognition

- R-CNN
- YOLO
- R-FCN
- SSD



References and further reading: https://github.com/kjw0612/awesome-deep-vision#object-detection

Semantic Segmentation

- Deep Parsing Networks
- BoxSup
- Fully Convolutional Networks for Semantic Segmentation
- SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation
- Fusing the Boundaries of Boundary Detection Using deep Learning

References and further reading: https://github.com/kjw0612/awesome-deep-vision#object-detection

Object Tracking

- DeepTrack
- Visual Tracking with fully Convolutional Networks
- Learning Multi-Domain Convolutional Neural Networks for Visual Tracking

References and further reading: https://github.com/kjw0612/awesome-deep-vision#object-detection

Scene Understanding

- Explain Images with Multimodal Recurrent Neural Networks
- Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models
- Deep Visual-Semantic Alignments for Generating Image Description
- Learning Query and Image Similarities with Ranking Canonical Correlation Analysis

References and further reading: https://github.com/kjw0612/awesome-deep-vision#object-detection

References

- Gradient-based learning applied to document recognition, Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner
- A fast learning algorithm for deep belief nets, Hinton, G. E., Osindero, S. and Teh, Y.
- ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky and Sutskever, Ilya and Hinton, Geoffrey E
- Visualizing and Understanding Convolutional Networks, Zeiler, Matthew D. and Fergus, Rob
- Very Deep Convolutional Networks for Large-Scale Image Recognition, Karen Simonyan, Andrew Zisserman
- U-Net: Convolutional Networks for Biomedical Image Segmentation, Olaf Ronneberger, Philipp Fischer, Thomas Brox
- Going deeper with convolutions, C. Szegedy et al.
- Generative Adversarial Nets, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio
- Deep Residual Learning for Image Recognition, K. He, X. Zhang, S. Ren and J. Sun
- Densely Connected Convolutional Networks, Gao Huang, Zhuang Liu, Kilian Q. Weinberger