UNSW AUSTRALIA

Never Stand Still

Neural Learning (2)

COMP9417 Machine Learning & Data Mining

Term 3, 2019

Adapted from slides in COMP9517 & COMP9444

Aims

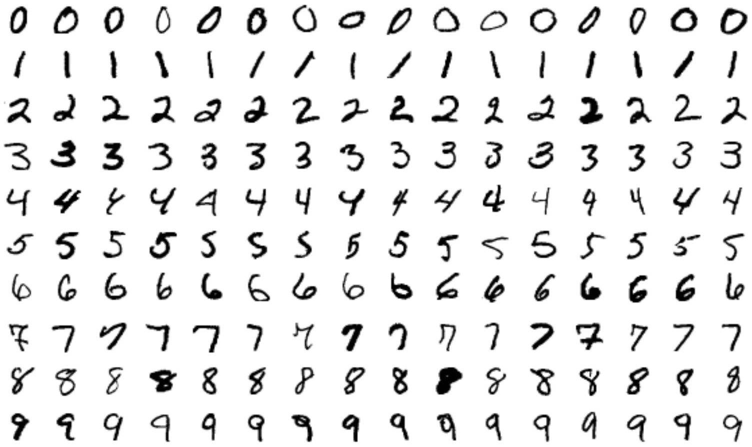
- Deep Learning
  - understand Convolutional Neural Networks
  - understand the main difference between CNN and regular NN
  - know the basics of training CNN
  - get to know several applications of CNN in computer vision

A Bit of History

- Earliest studies about visual mechanics of animals by Hubel and Weisel emphasised the importance of edge detection for solving CV problems
  - 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 well recognised AlexNet Convolutional Neural Network

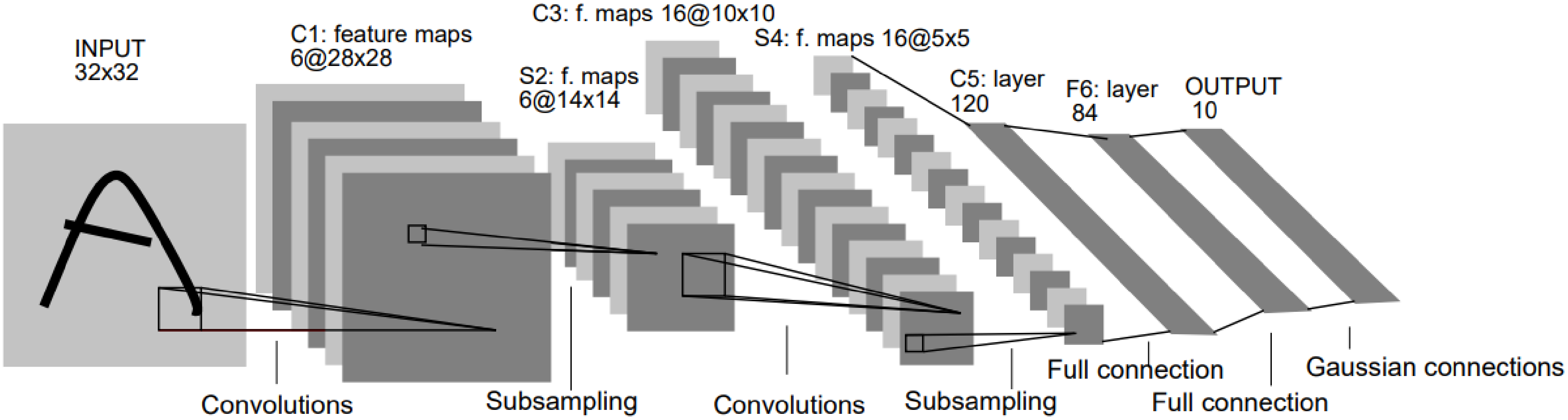
A Bit of History

- LeNet-5
    - The very first CNN
    - Handwritten digit recognition



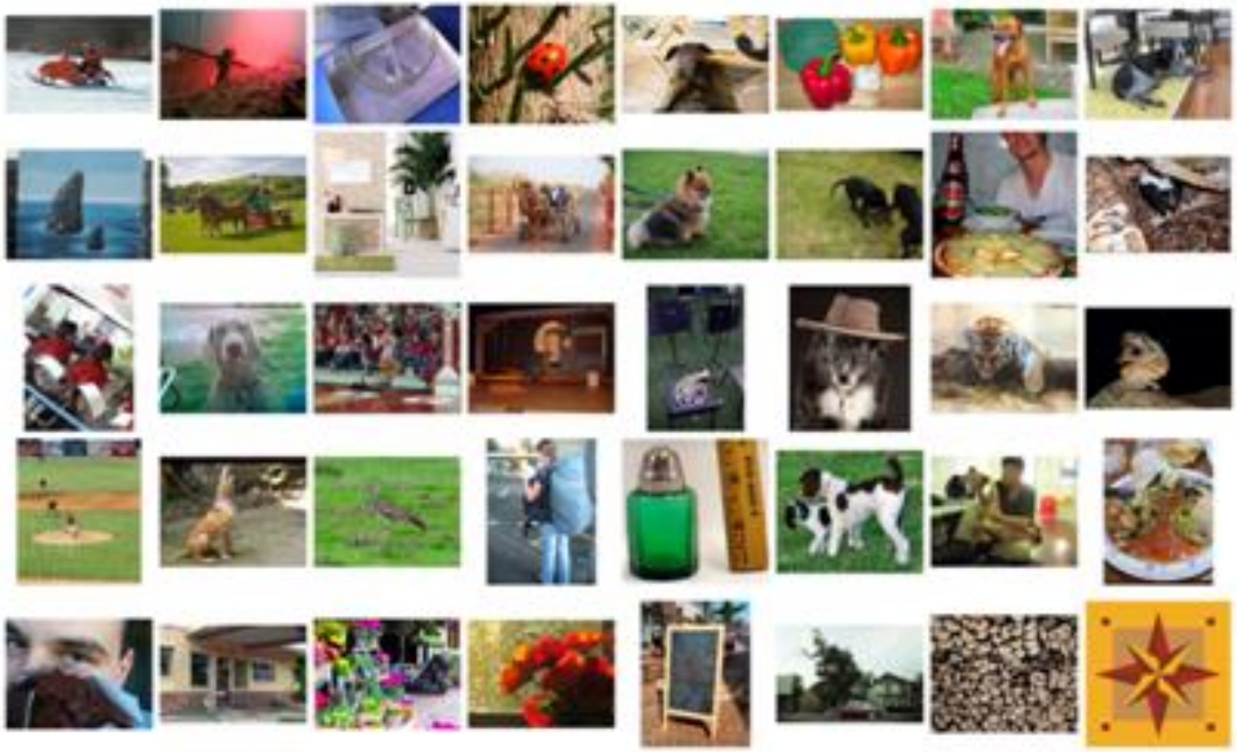
A Bit of History

- LeNet-5
  - The very first CNN
  - Handwritten digit recognition



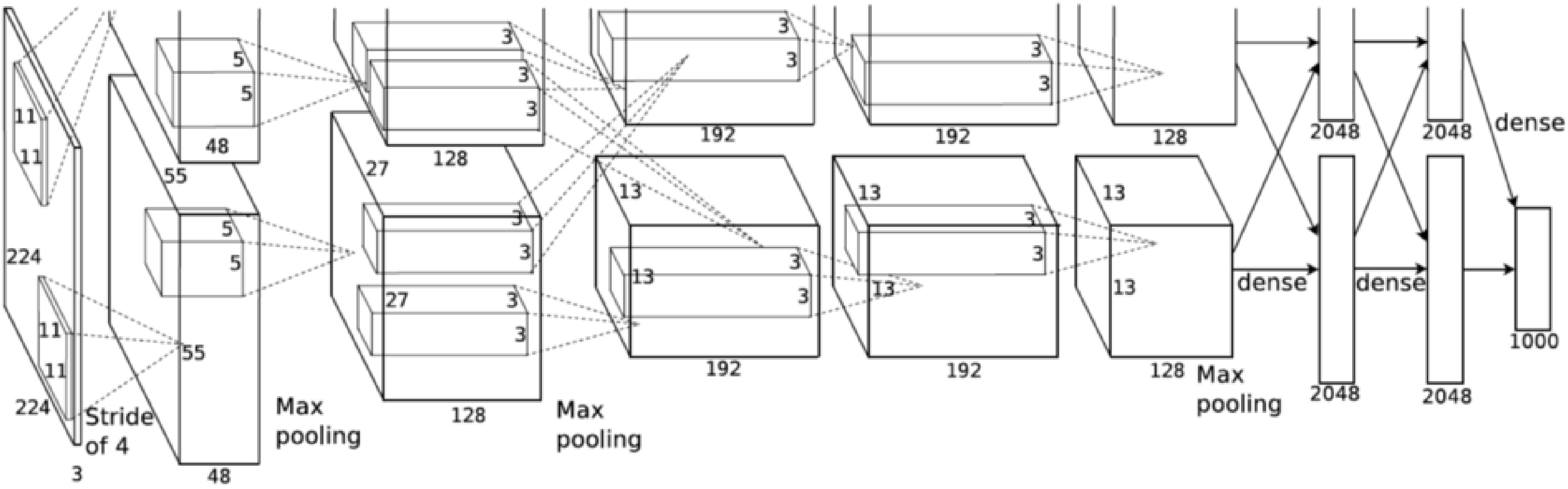
A Bit of History

- AlexNet (2012)
  - ImageNet classification
  - Images showing 1000 object categories



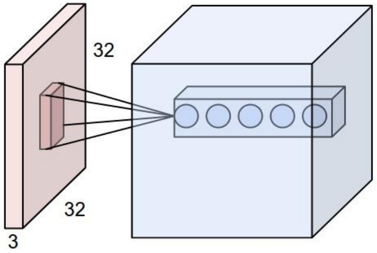
A Bit of History

- AlexNet (2012)
  - ImageNet classification
  - Images showing 1000 object categories



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



Deep Learning

- This lecture focuses on CNNs
- For more in-depth understanding about different techniques of deep learning – ***Check COMP9444***

CNNs

- CNNs are very similar to regular Neural Networks
  - Made up of neurons with learnable weights
- CNN architecture assumes that inputs are images
  - So that we have local features
  - Which 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

Why CNNs?

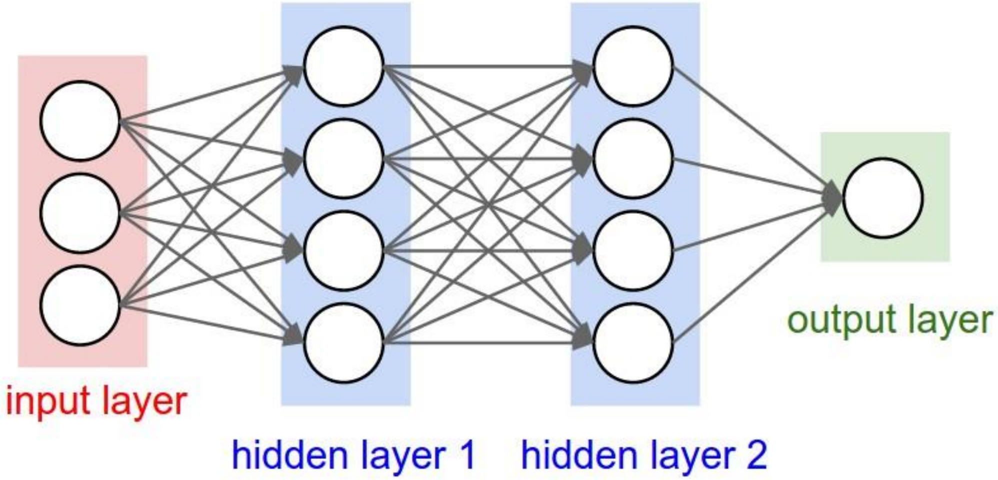
- The problem with regular NNs is that they do not scale well with dimensions (i.e. larger images)
  - Eg: 32x32 image with 3 channels (RGB) – a neuron in first hidden layer would have 32x32x3 = 3,072 weights : manageable.
  - Eg: 200x200 image with 3 channels – a neuron in first hidden layer would have 200x200x3 = 120,000 weights and we need at least several of these neurons which makes the weights explode.

What is different?

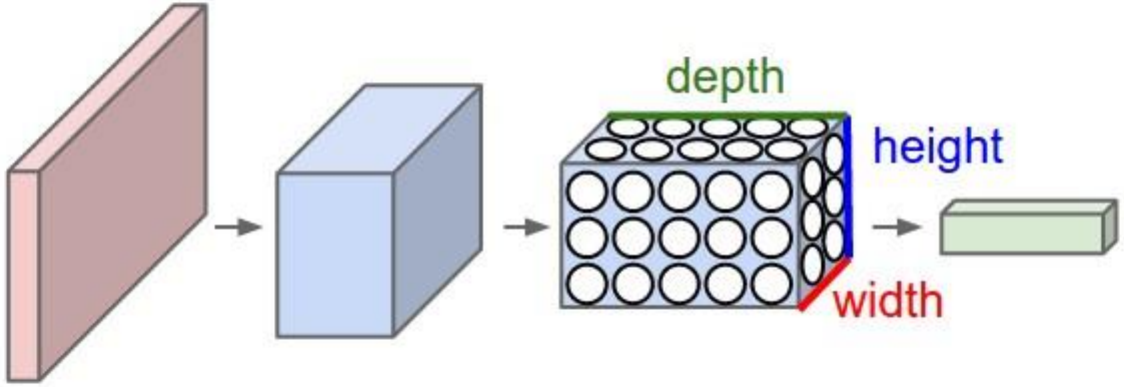
- In contrast, CNNs consider 3-D volumes of neurons and propose a parameter sharing scheme that minimises the number of parameters required by the network.
- CNN neurons are arranged in 3 dimensions: Width, Height and Depth.
- Neurons in a layer are only connected to a small region of the layer before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

What is different?

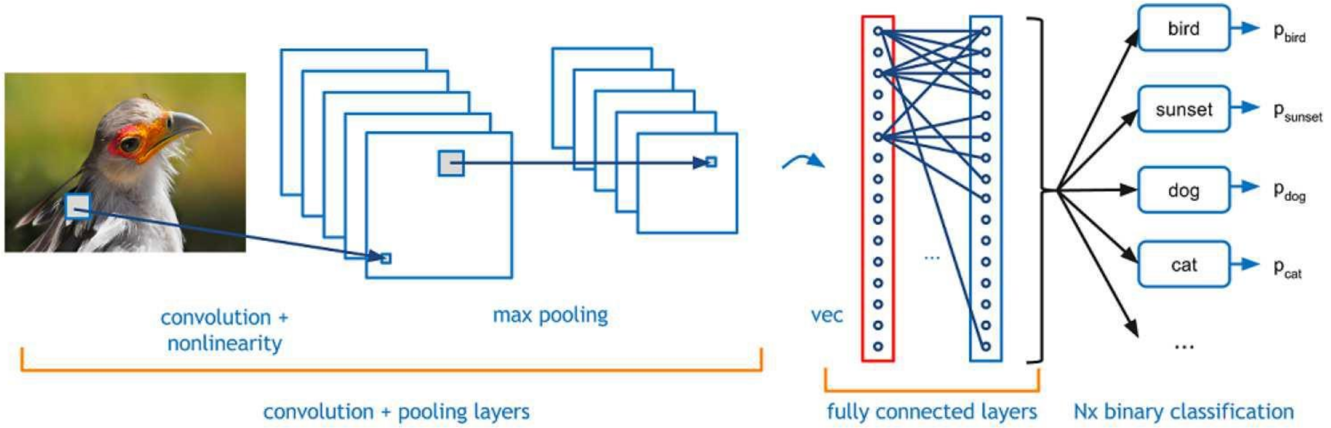
NN



CNN



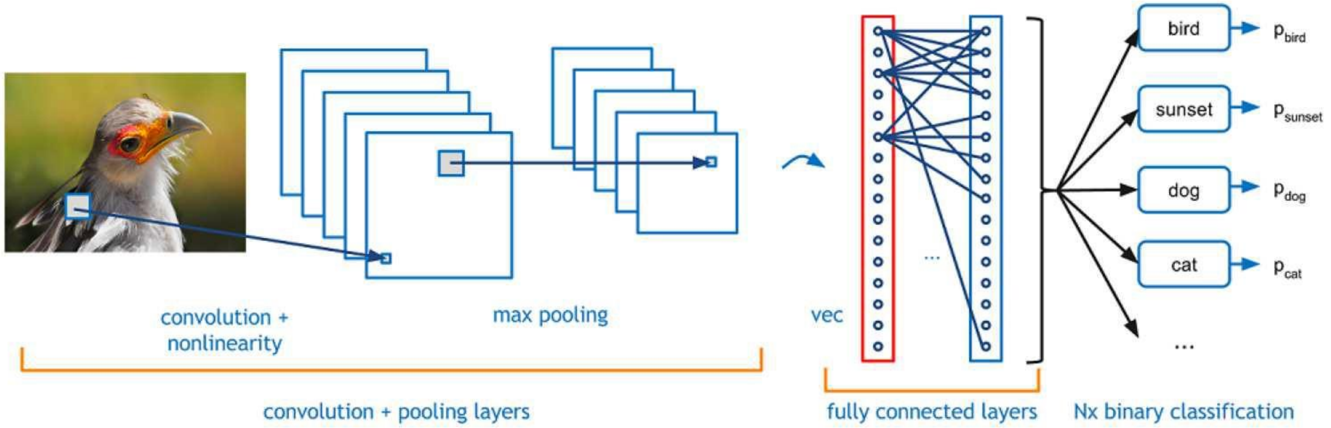
CNN architecture



Main layers:

Convolutional, Pulling, ReLU, Fully-connected, Drop-out, Output layers

Convolutional Layer

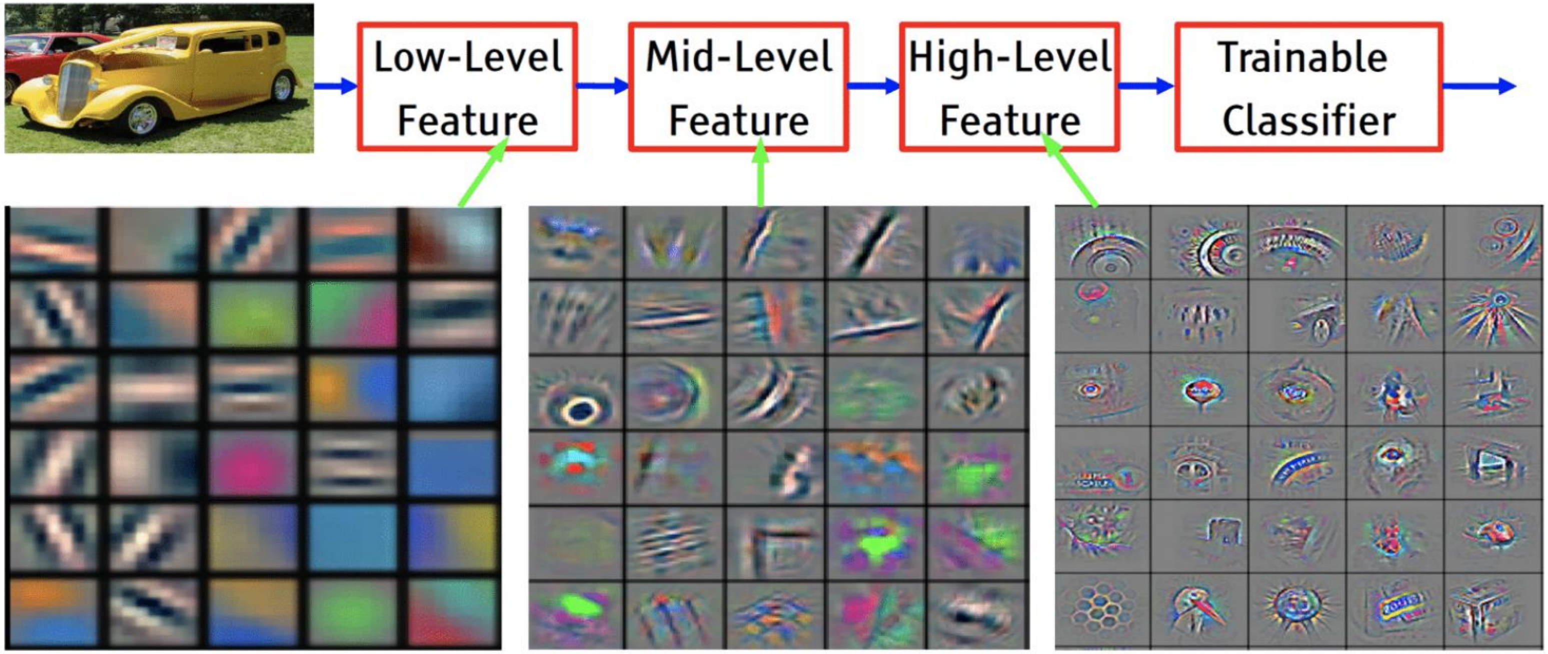


Suppose we want to classify an image as a bird, sunset, dog, cat, etc.

If we can identify features such as feather, eye, or beak which provide useful information in one part of the image, then those features are likely to also be relevant in another part of the image.

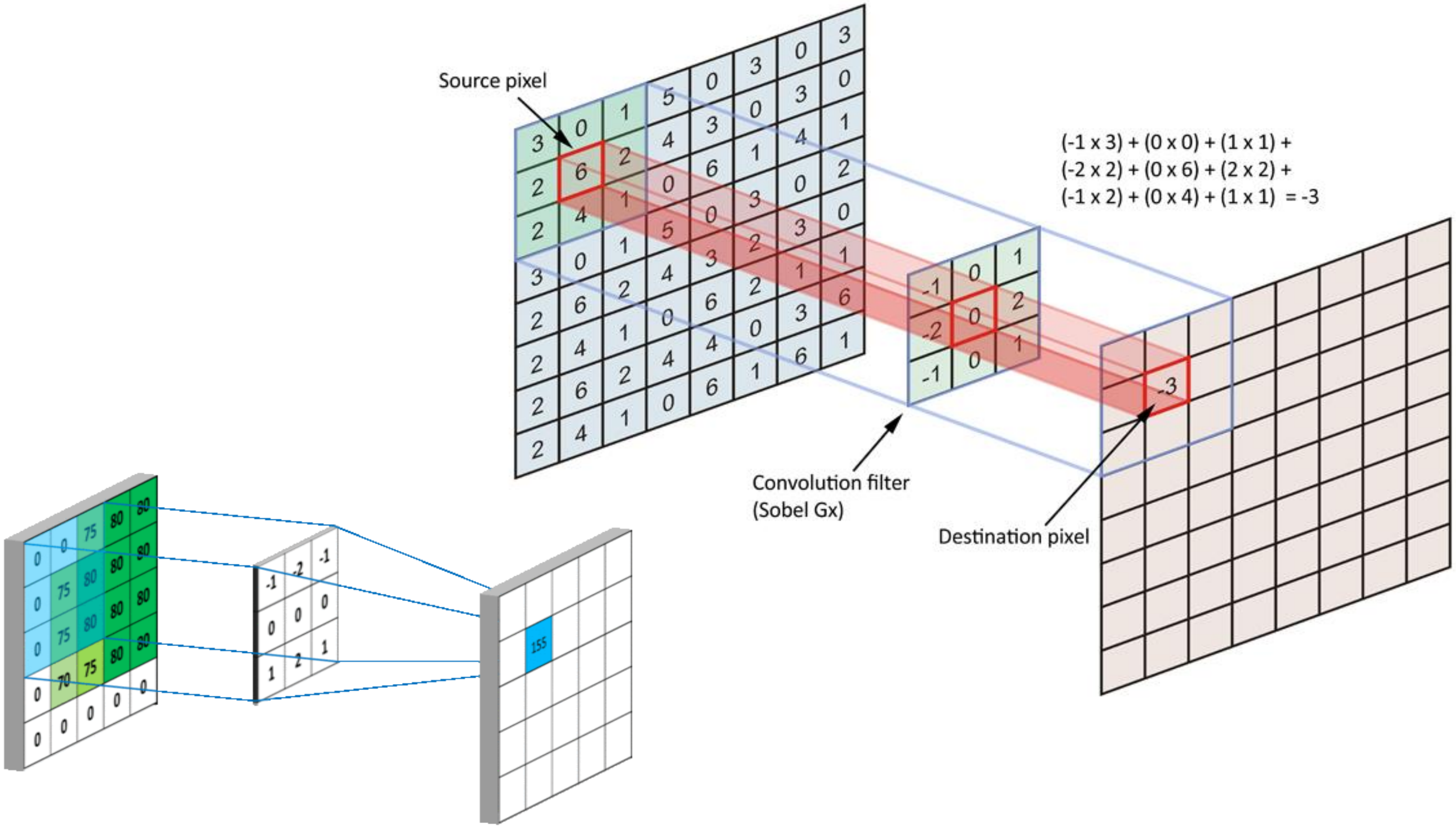
We can exploit this regularity by using a convolution layer which applies the same weights to different parts of the image.

Convolutional Layer



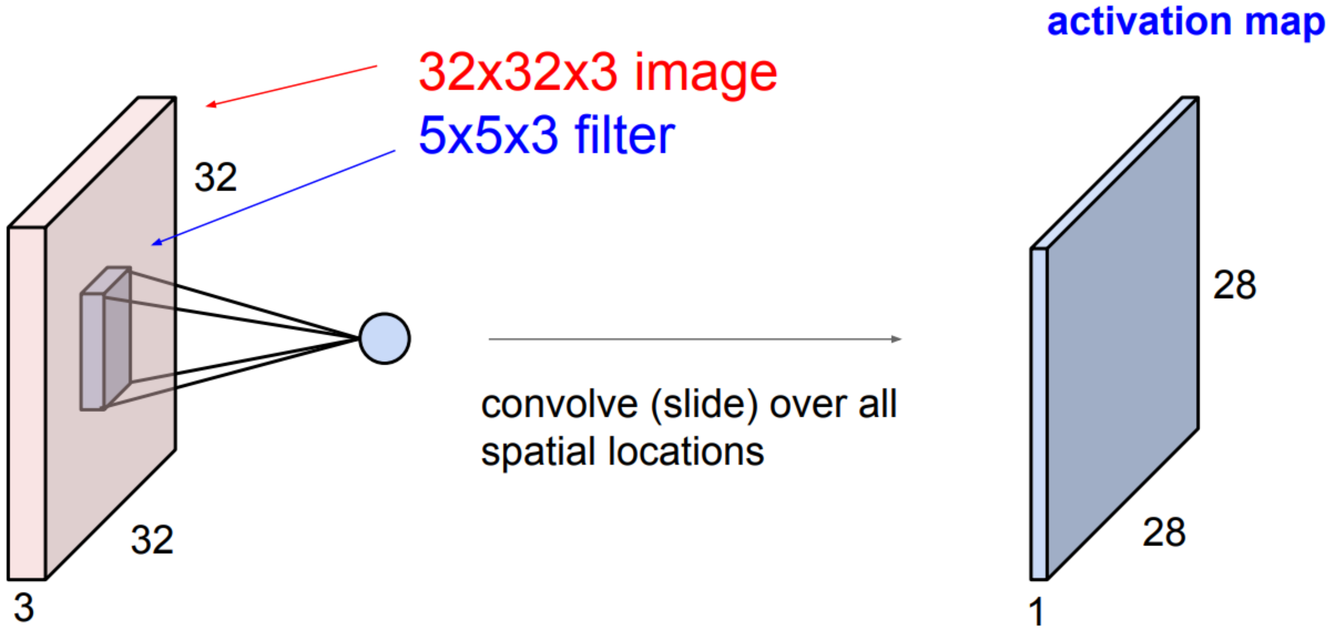
Convolutional Layer

Convolution

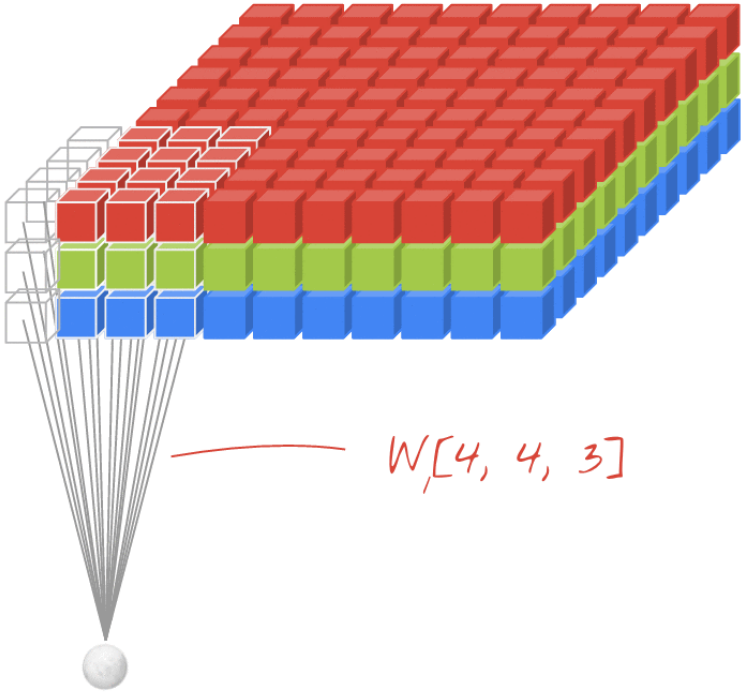


(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3

Convolutional Layer



Convolutional Layer



Original LINK

Convolutional Layer

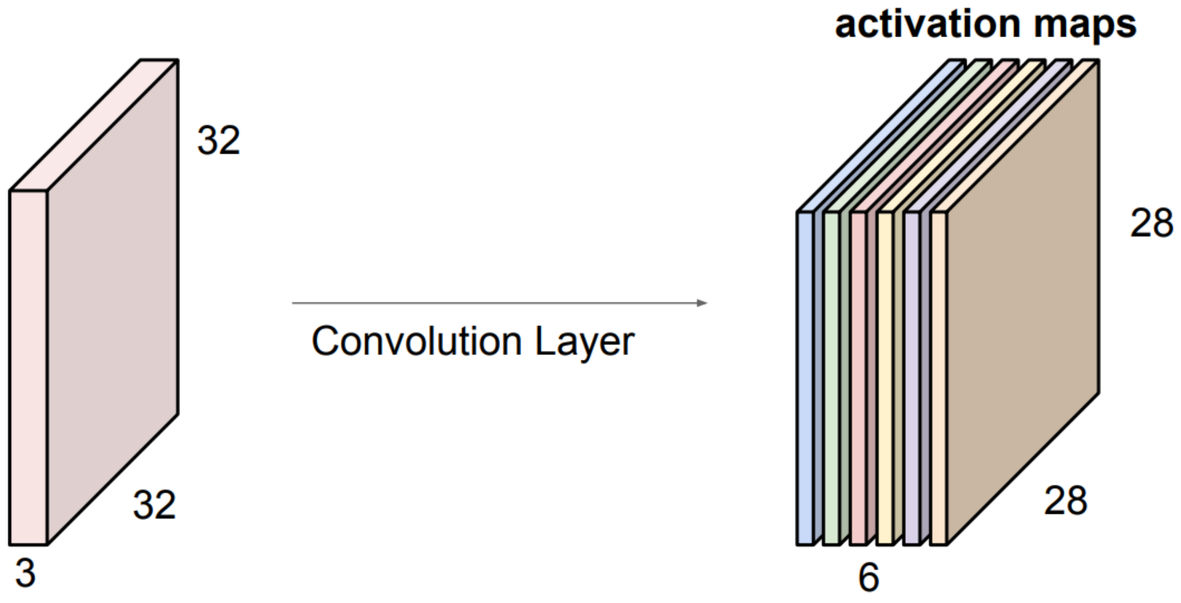
- The output of the Conv layer can be interpreted as holding neurons arranged in a 3D volume.
- The Conv layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume.
- During the forward pass,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

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)

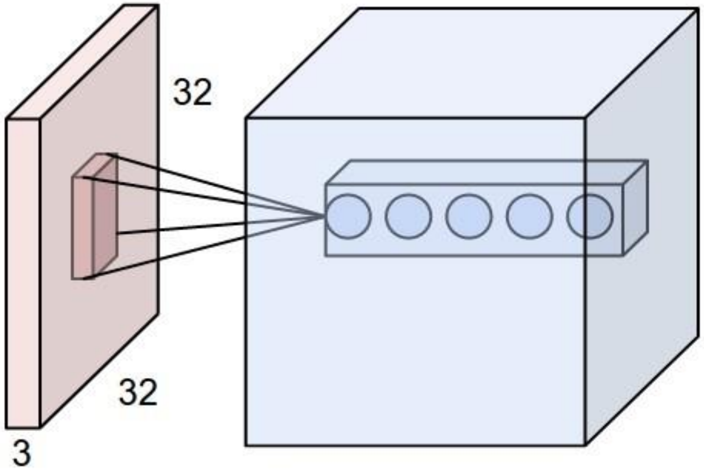
Convolutional Layer

With 6 filters, we get 6 activation maps



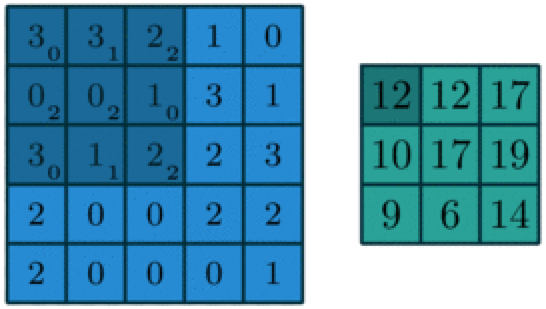
Convolutional Layer

- 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



Convolutional Layer

- Three hyperparameters control the size of the output volume: the ***depth, stride*** and ***zero-padding***
  - ***Stride*** is the distance that the filter is moved by in spatial dimensions



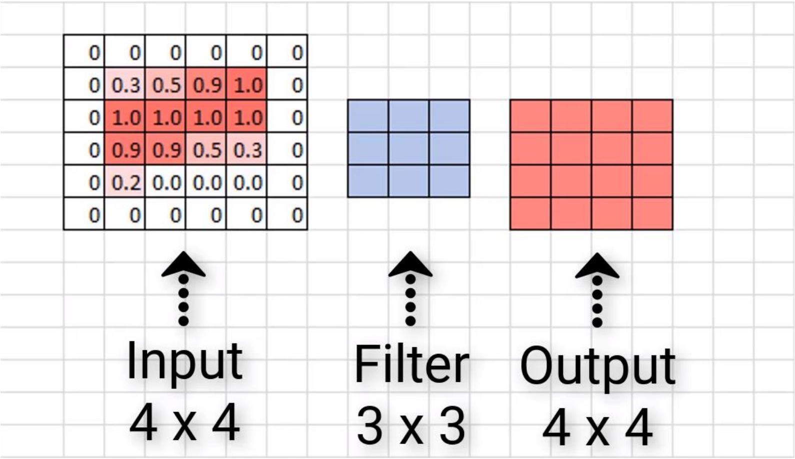
3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12	12	17
10	17	19
9	6	14

http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

Convolutional Layer

- Three hyperparameters control the size of the output volume: the ***depth, stride*** and ***zero-padding***
  - ***Zero-padding*** is padding of the input with zeros spatially on the border of the input volume



0	0	0	0	0	0
0	0.3	0.5	0.9	1.0	0
0	1.0	1.0	1.0	1.0	0
0	0.9	0.9	0.5	0.3	0
0	0.2	0.0	0.0	0.0	0
0	0	0	0	0	0

https://deeplizard.com/learn/video/qSTv_m-KFk0

Convolutional Layer

- 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:

$$(W-F+2P)/S+1$$

- 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.

Example: AlexNet

For example, in the first convolutional layer of AlexNet,
$W = 227, F = 11, P = 0, S = 4.$

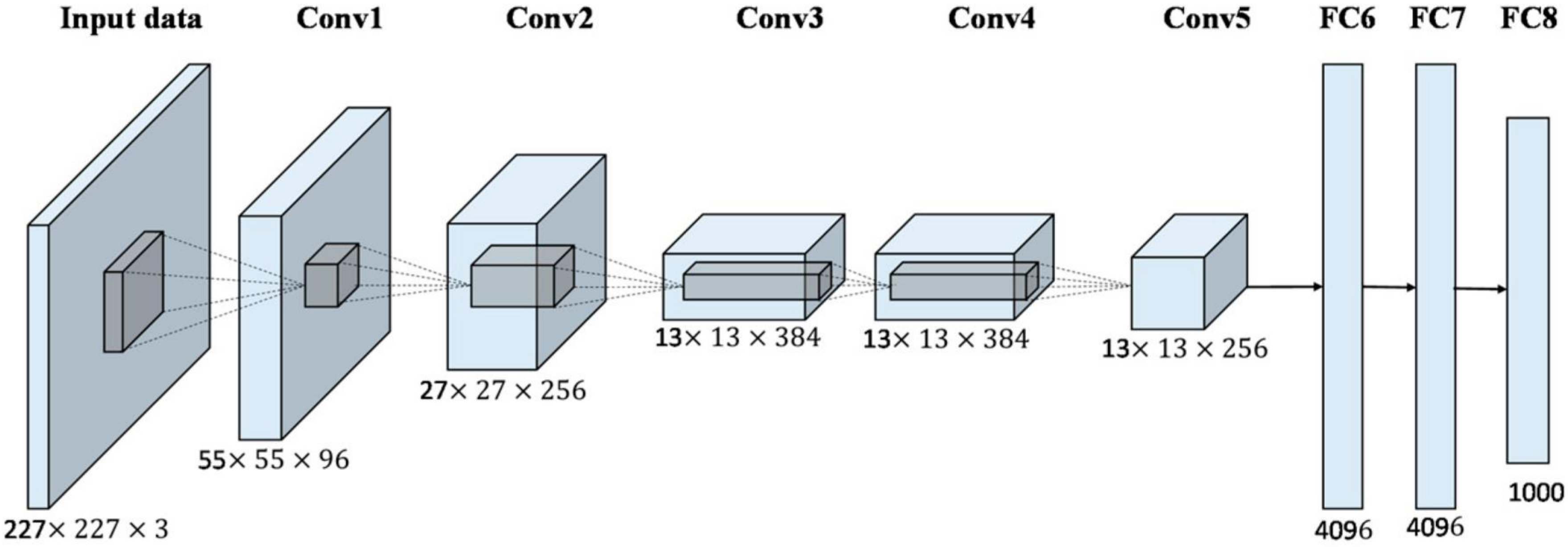
The width of the output is

$$(W-F+2P)/S+1 = (227 - 11 + 0)/4 + 1 = 55$$

There are 96 filters in this layer, the output volume of this layer is thus

$$55 \times 55 \times 96$$

Example: AlexNet



Convolutional Layer

- Main property – ***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.

Convolutional Layer

- Examples:
  - 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.
  - Eg2: Suppose an input volume had size [16x16x20], i.e. . 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).

Convolutional Layer

- Main property – ***parameter sharing***:
  - Parameter sharing scheme used in Convolutional Layers to control the number of parameters
    - 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!
  - Motivation of parameter sharing
    - 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).

Convolutional Layer

- Example:
  - In AlexNet, without parameter sharing, 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.
  - 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).
  - Alternatively, it can be viewed as all 55*55 neurons in each depth slice will now be using the same parameters.

Example: AlexNet

For example, in the first convolutional layer of AlexNet,

$W = 227$, $F = 11$, $P = 0$, $S = 4$.

The width of the output is

$$(W-F+2P)/S+1 = (227 - 11 + 0)/4 + 1 = 55$$

There are 96 filters in this layer. Compute the number of:

weights per neuron?
neurons?
connections?
independent parameters?

Example: AlexNet

For example, in the first convolutional layer of AlexNet,

$W = 227, F = 11, P = 0, S = 4.$

The width of the output is

$$(W-F+2P)/S+1 = (227-11+0)/4+1 = 55$$

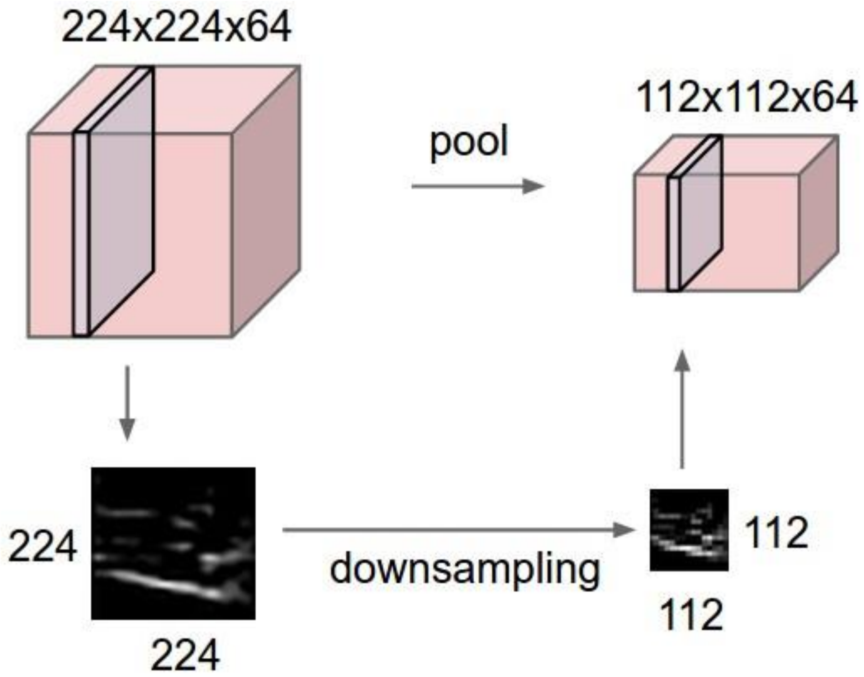
There are 96 filters in this layer. Compute the number of:

$$\begin{array}{llcr}
\text{weights per neuron?} & 1+11\times 11\times 3 & = & 364 \\
\text{neurons?} & 55\times 55\times 96 & = & 290{,}400 \\
\text{connections?} & 55\times 55\times 96\times 364 & = & 105{,}705{,}600 \\
\text{independent parameters?} & 96\times 364 & = & 34{,}944
\end{array}$$

Pooling Layer

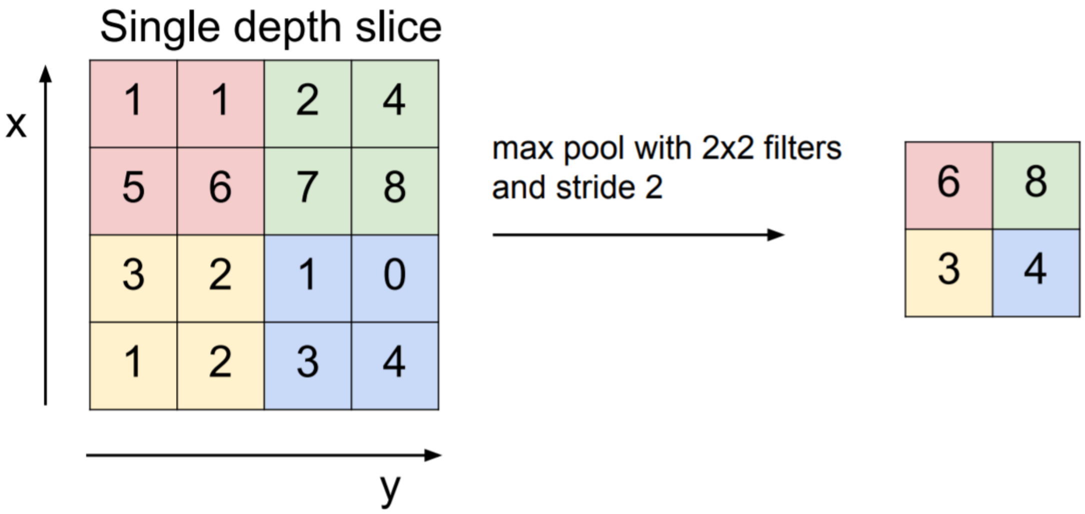
- The function of pooling layer
  - to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and
  - hence to also control overfitting
- The Pooling Layer operates
  - independently on every depth slice of the input and resizes it spatially, typically 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

Pooling Layer



Pooling Layer

Max pooling



Single depth slice

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2

6	8
3	4

Pooling Layer

不规则 长宽不相等

- If the previous layer is $J \times K$, and max pooling is applied with width F and stride S, the size of the output will be

$$(1+(J-F)/s)\times(1+(K-F)/s)$$

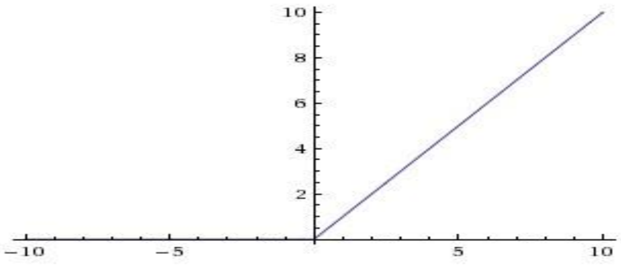
- If max pooling with width 3 and stride 2 is applied to the feature map of size 55 ×55 in the first convolutional layer of AlexNet, what is the output size after pooling?
  - Answer: $1+(55-3)/2=27$.
- How many independent parameters does this add to the model?
  - Answer: None! (no weights to be learned, just computing max)

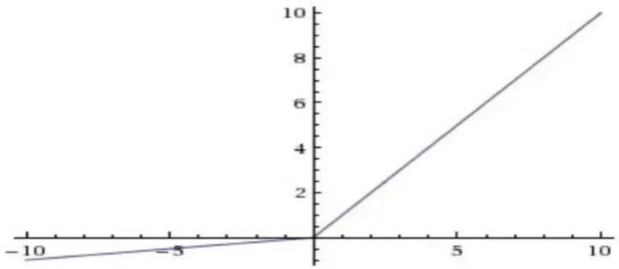
ReLU Layer

- Although ReLU (**Re**ctified **L**inear **U**nit) is considered as a layer, it is really an activation function:

$$f(x) = max(0, x)$$

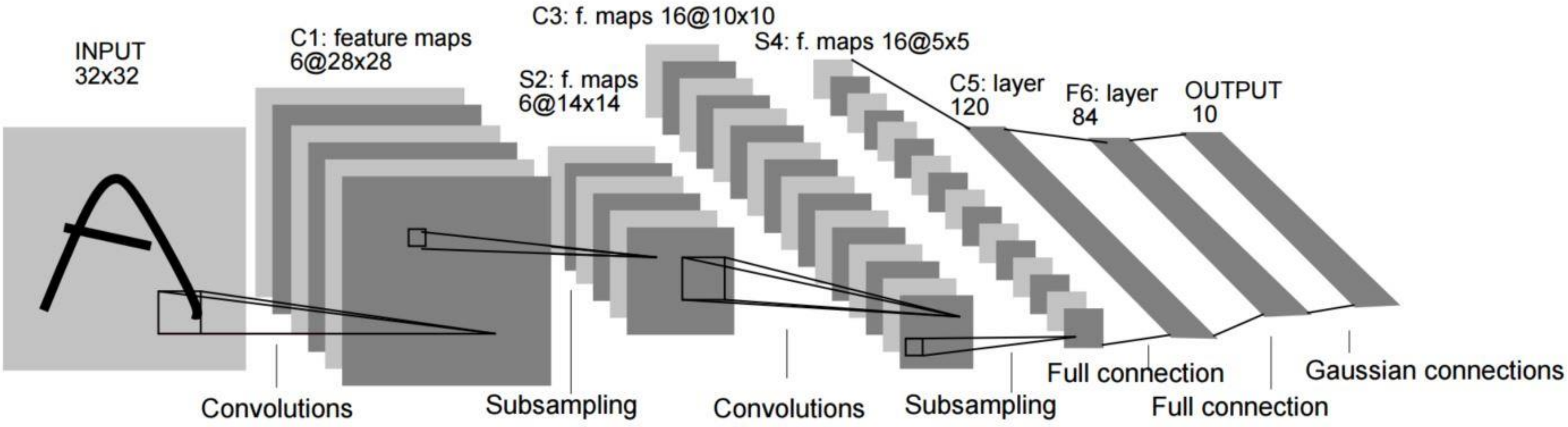
- This is favoured in deep learning as opposed to the traditional activation functions like Sigmoid or Tanh
  - To accelerate the convergence of stochastic gradient descent
  - Be computationally inexpensive compared to traditional ones
- However, ReLu units can be fragile during training and 'die'. Leaky ReLUs were proposed to handle this problem.





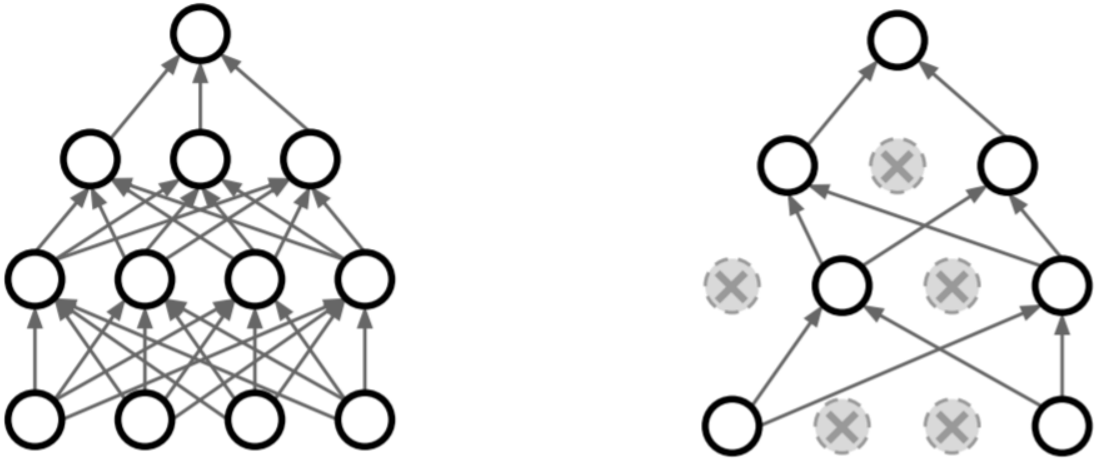
Fully-connected Layer

- 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.



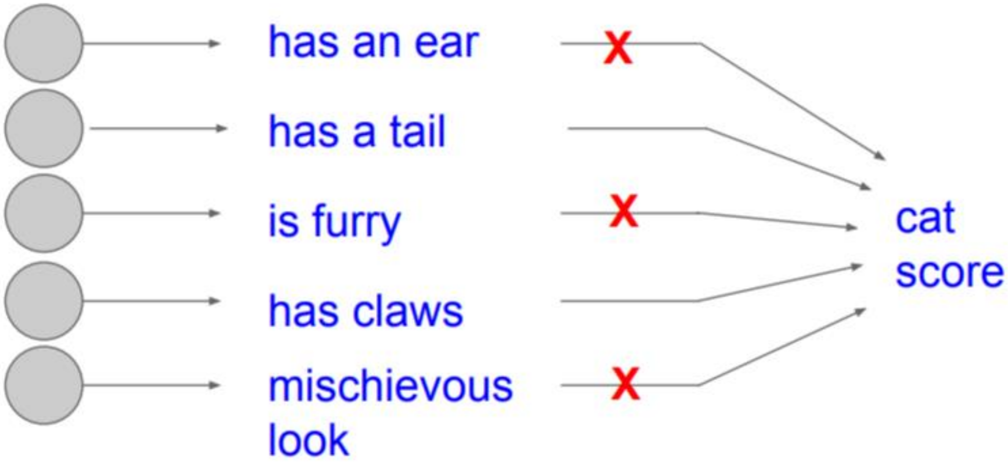
Dropout Layer

- Problem with overfitting – model performs well on training data but generalises poorly to testing data
- Dropout is a simple and effective method to reduce overfitting
- In each forward pass, randomly set some neurons to zero
- Probability of dropping is a hyperparameter, such as 0.5



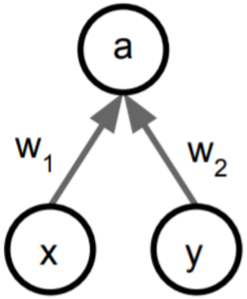
Dropout Layer

- Makes the training process noisy
- Forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs
- Prevents co-adaptation of features and simulates a sparse activation
- Analogous to training a large ensemble of models but with much higher efficiency



Dropout Layer

- During test time, direct application would make the output random
- A simple approach: multiply the activation by dropout probability (e.g. 0.5)



Consider a single neuron.

At test time we have: $E[a] = w_1x + w_2y$

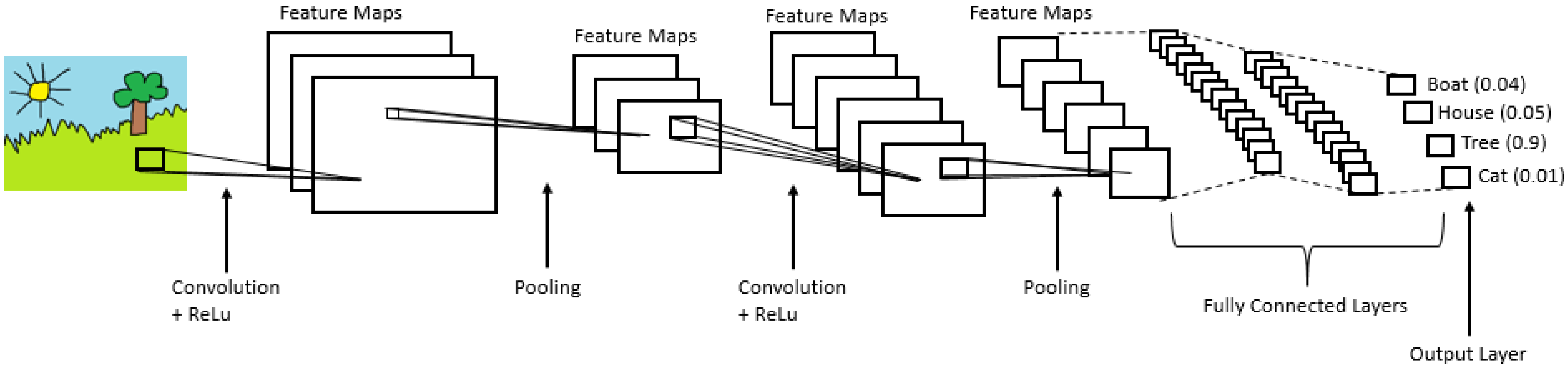
During training we have:

$$\begin{aligned} E[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ &= \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

Output Layer

- The output layer produces the probability of each class given the input image
- This is the last layer containing the same number of neurons as the number of classes in the dataset
- The output of this layer passes through a Softmax activation function to normalize the outputs to a sum of one:

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$



Loss Function

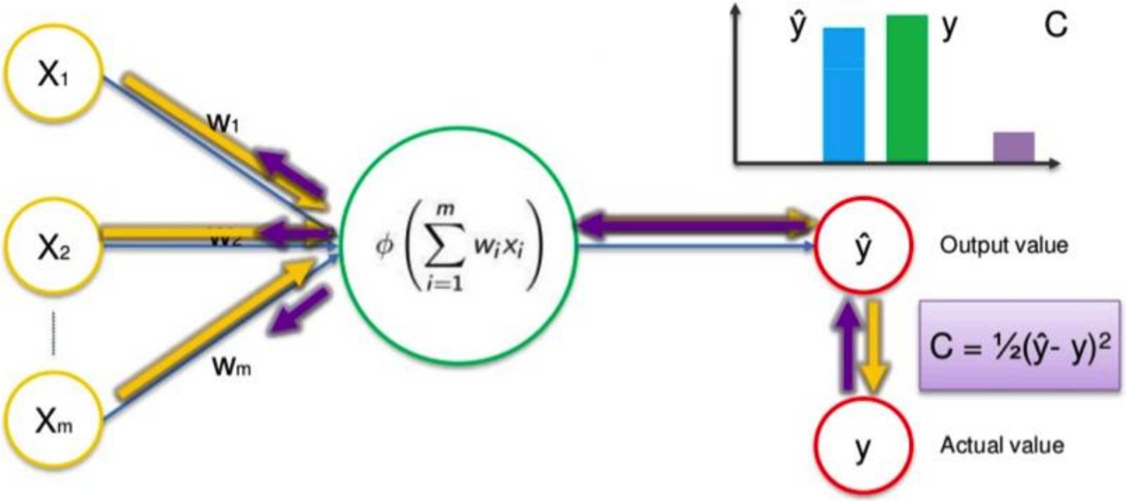
- A loss function is used to compute the model's prediction accuracy from the outputs
  - Most commonly used: categorical cross-entropy loss function

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i$$

- The training objective is to minimise this loss
- The loss guides the backpropagation process to train the CNN model
- Stochastic gradient descent and the Adam optimiser are commonly used algorithms for optimisation

Training

- Backpropagation in general:



https://www.superdatascience.com/blogs/artificial-neural-networks-backpropagation

1. Initialise the network.

2. Input the first observation.

3. Forward-propagation. From left to right the neurons are activated and the output value is produced.

4. Calculate the error in the outputs (loss function).

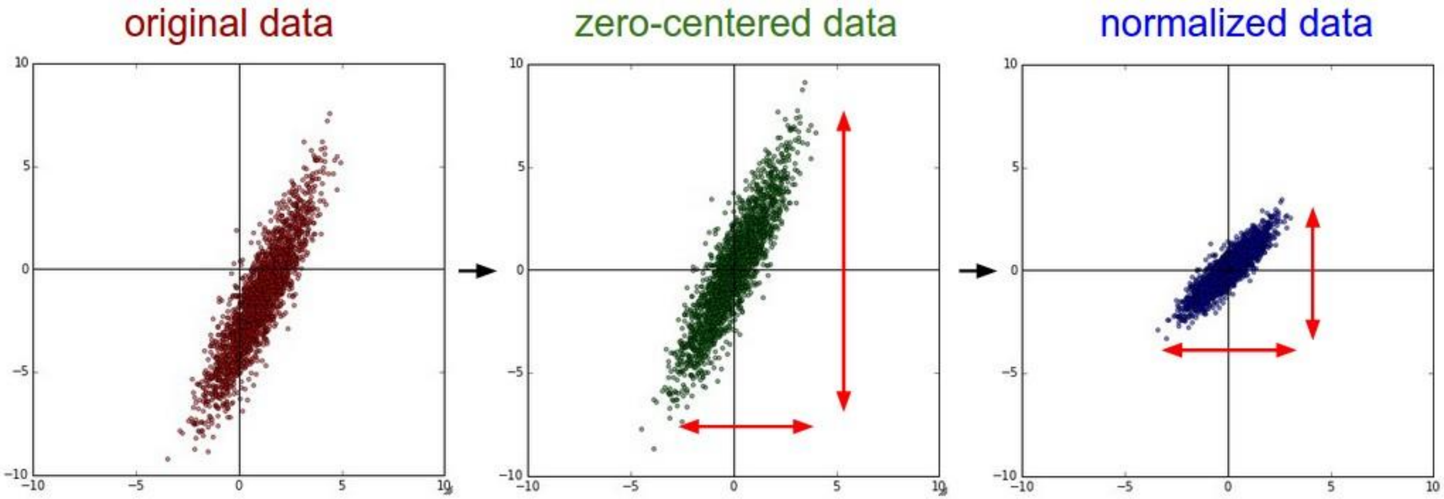
5. From right to left the generated error is back-propagated and accumulate the weight updates (partial derivatives).

6. Repeat steps 2-5 and adjust the weights after a batch of observations.

7. When the whole training set passes through the network, that makes an epoch. Redo more epochs.

Pre-processing

- Pre-processing: image scaling, zero mean, and normalisation

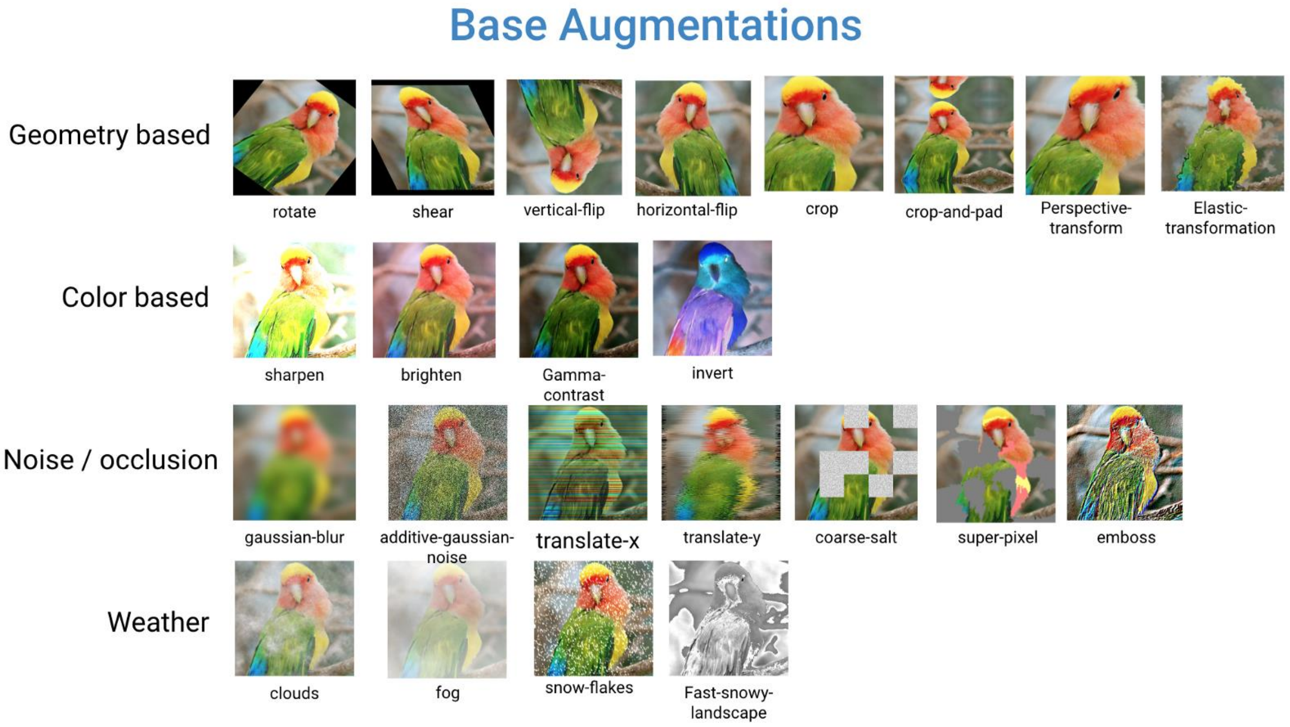


Data Augmentation

- Essential for increasing the dataset size and avoiding over-fitting
- More data augmentation often leads to better performance but also longer training time
- Commonly used techniques include:
  - Horizontal / vertical flipping
  - Random cropping and scaling
  - Rotation
  - Gaussian filtering
- During testing, average the results from multiple augmented input images

Data Augmentation

- Need evaluation => not all techniques are useful



https://blog.insightdatascience.com/automl-for-data-augmentation-e87cf692c366

Initialisation

- Weight initialisation
  - Cannot be all 0’s => Need to ensure diversity in the filter weights
  - Use small random numbers => might aggravate the diminishing gradients problem
    - With calibration
    - Sparse initialisation
    - More advanced techniques
  - Use ImageNet pretrained models => not always possible

Balancing Data

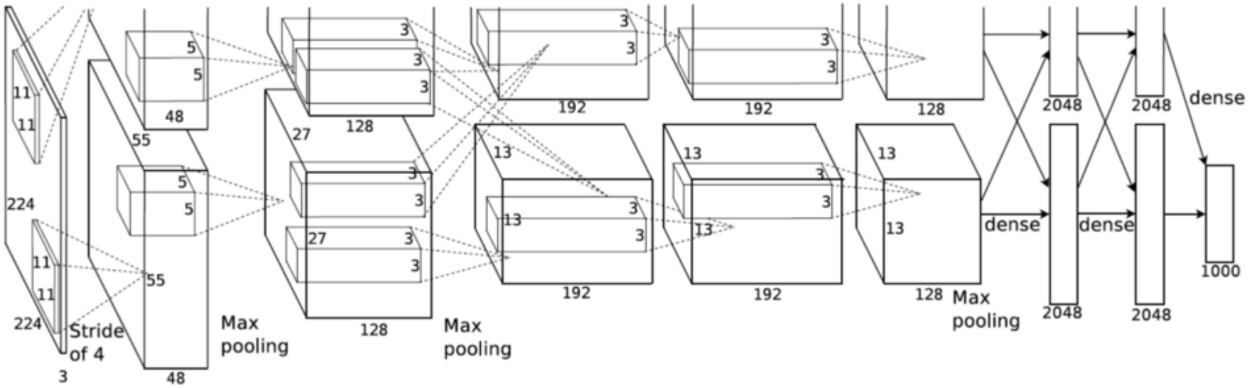
- Balanced training data
  - Important to have similar numbers of training images for different classes, so the optimisation would not be biased by one class
  - Use random sampling to achieve this effect during each epoch of training
  - Assign different weights in the loss function

$$\alpha_c = median\_freq/freq(c)$$

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i$$

Testing

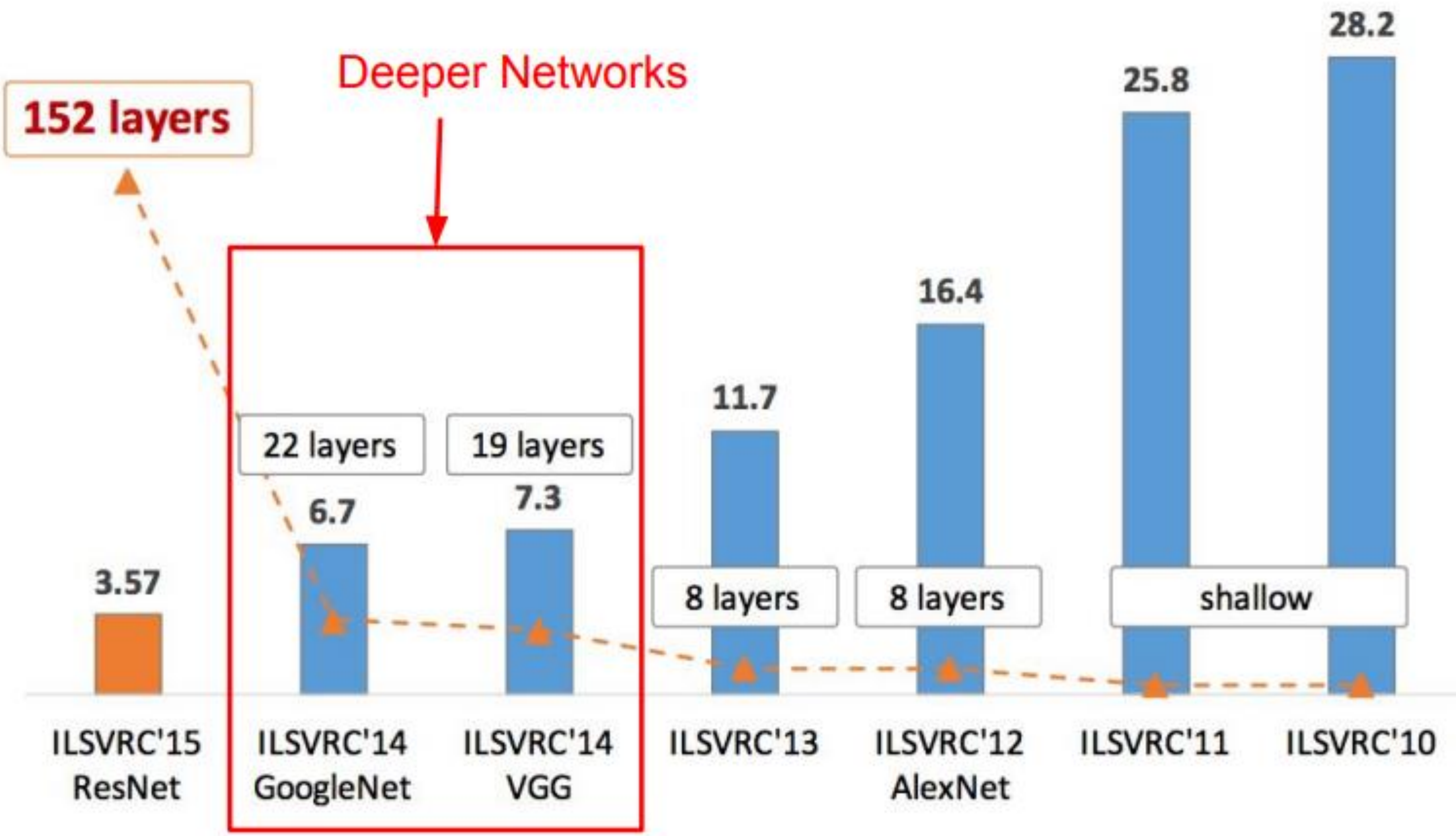
- Forward passes of the network throughout the layers give the prediction output of the input data



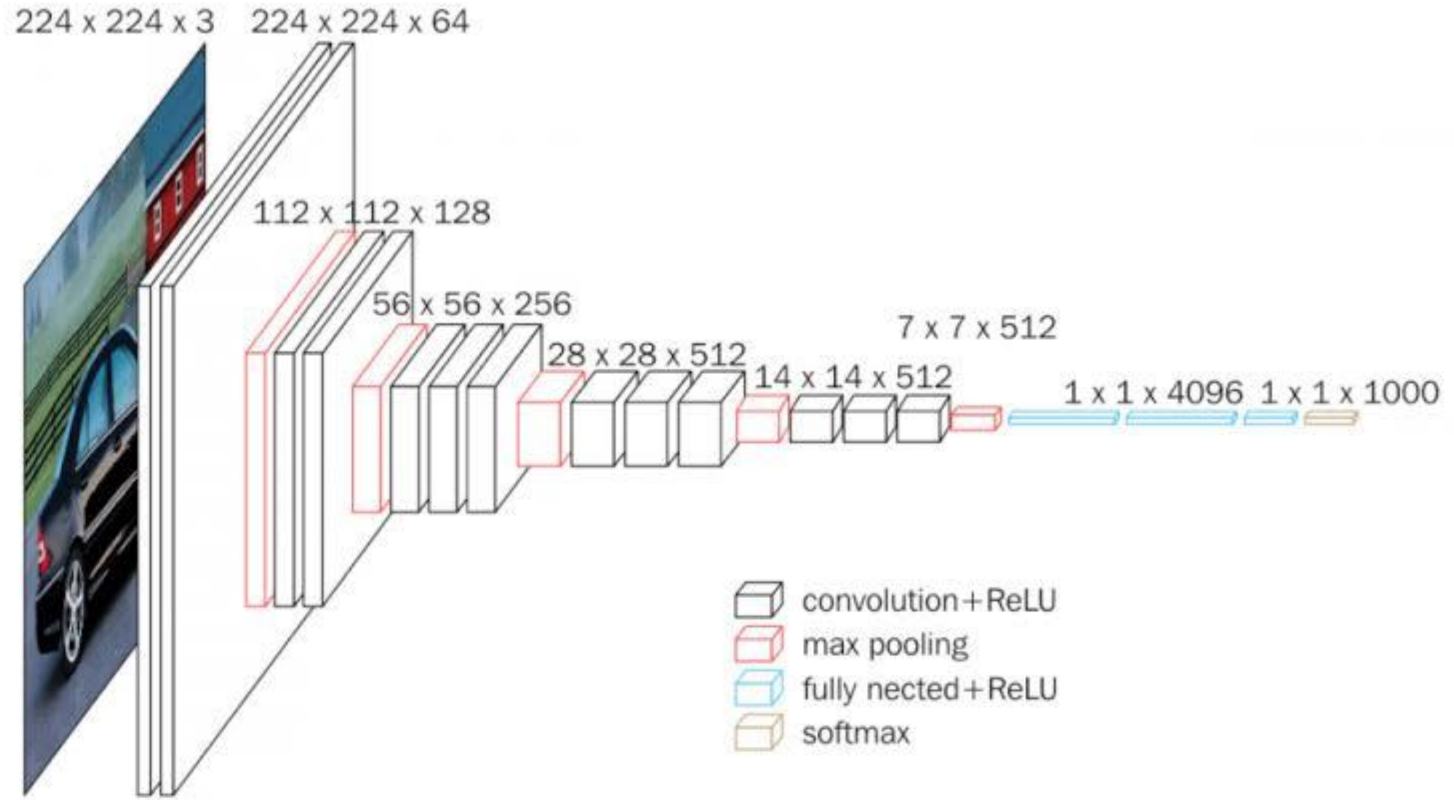
Transfer Learning

- CNN models trained on ImageNet can be applied to other types of images
- It is possible to finetune only the last FC layers to better fit the model to the specific set of images
- Especially useful for small datasets

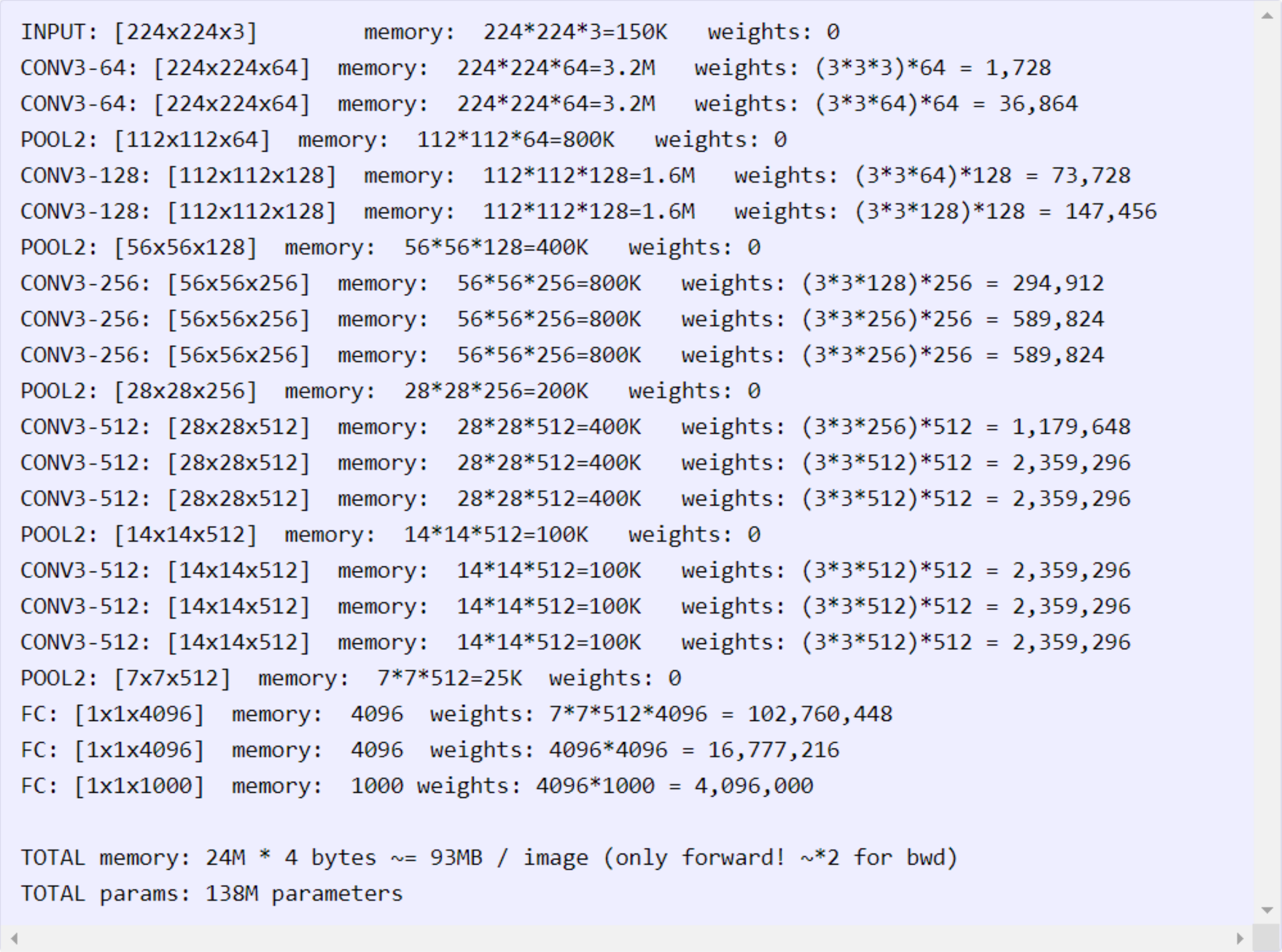
CNN Models


Deeper Networks
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15
ResNet
ILSVRC'14
GoogleNet
ILSVRC'14
VGG
ILSVRC'13
ILSVRC'12
AlexNet
ILSVRC'11
ILSVRC'10

VGGNet


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution+ReLU
max pooling
fully nected+ReLU
softmax

VGGNet



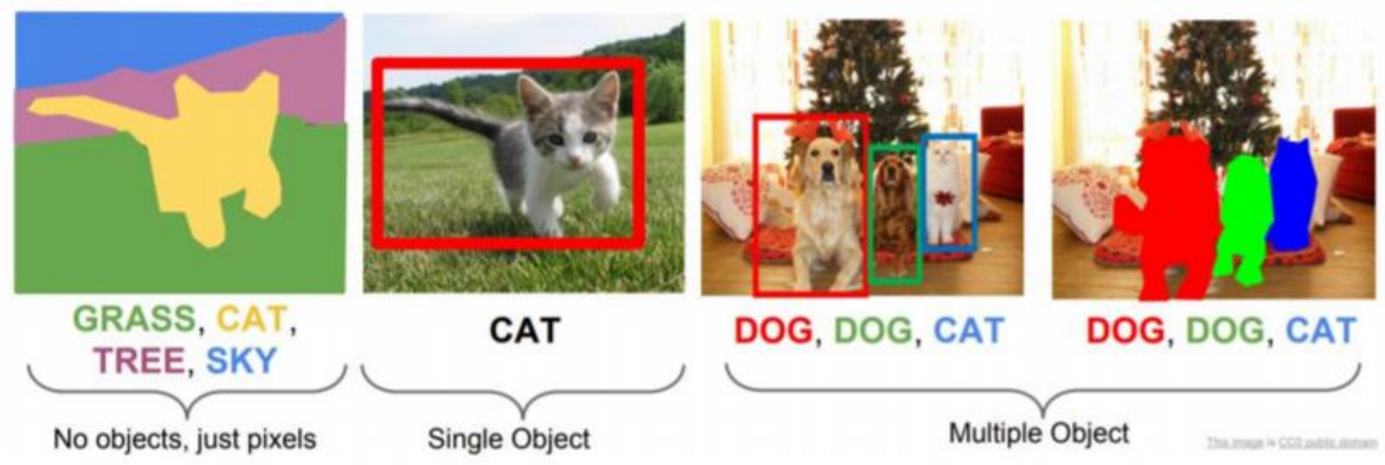
```
INPUT: [224x224x3]        memory:  224*224*3=150K   weights: 0
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory:  112*112*64=800K   weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory:  7*7*512=25K  weights: 0
FC: [1x1x4096]  memory:  4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory:  4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory:  1000 weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

Well-known Models

- Object Recognition
  - AlexNet (2012)
  - GoogLeNet
  - VGGNet
  - ResNet
  - Inception v3/v4
  - DenseNets is the current state-of-the-art
- Semantic Segmentation
  - Multi-scale CNN (2012)
  - FCN
  - U-net / V-net
  - U-net / V-net with skip/dense connections
  - Many other variations
- Adversarial
  - Generative Adversarial Networks (2014)
  - Pixel2pixelGans
  - CycleGans

Advanced Models

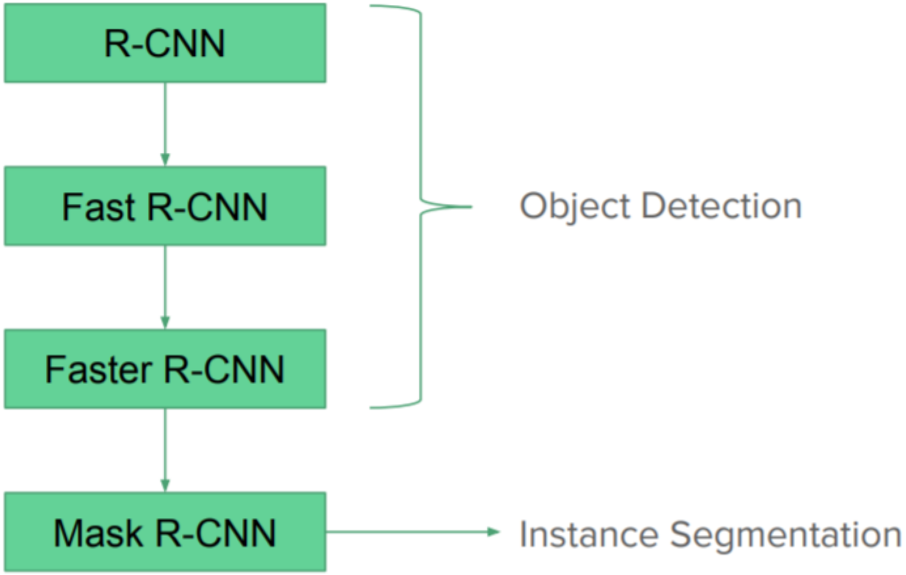
- Background:
  - Semantic segmentation
  - Single object detection
  - Multiple objects detection
  - Instance segmentation



https://cseweb.ucsd.edu/classes/sp18/cse252C-a/CSE252C_20180509.pdf

Advanced Models

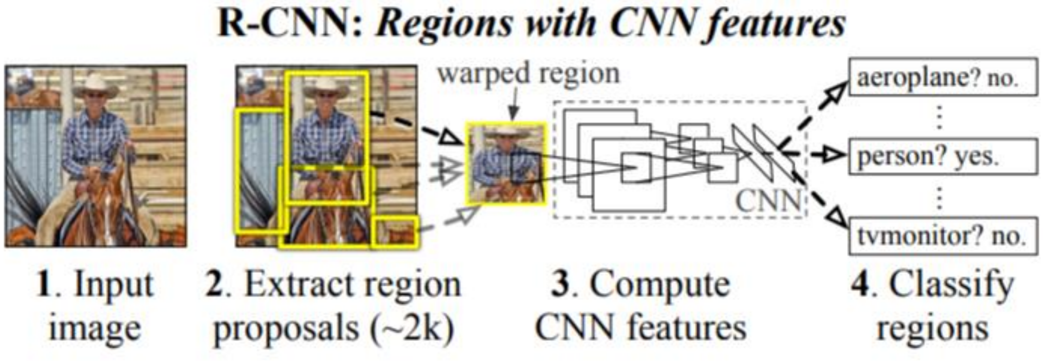
- The R-CNN family



https://cseweb.ucsd.edu/classes/sp18/cse252C-a/CSE252C_20180509.pdf

Advanced Models

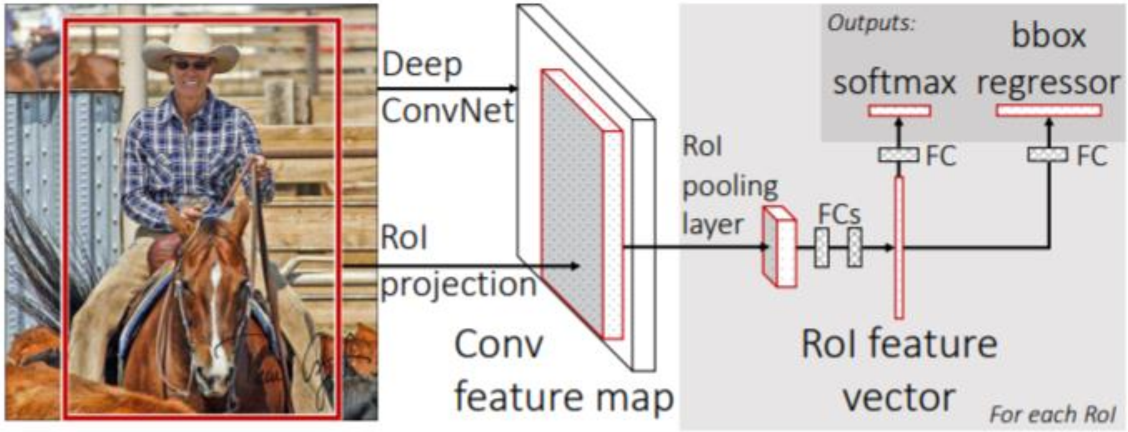
- R-CNN
  - Training is expensive and slow because of selective search and lack of shared computation



Girshick et al., Rich feature hierarchies for accurate object detection and semantic segmentation, CVPR, 2014.

Advanced Models

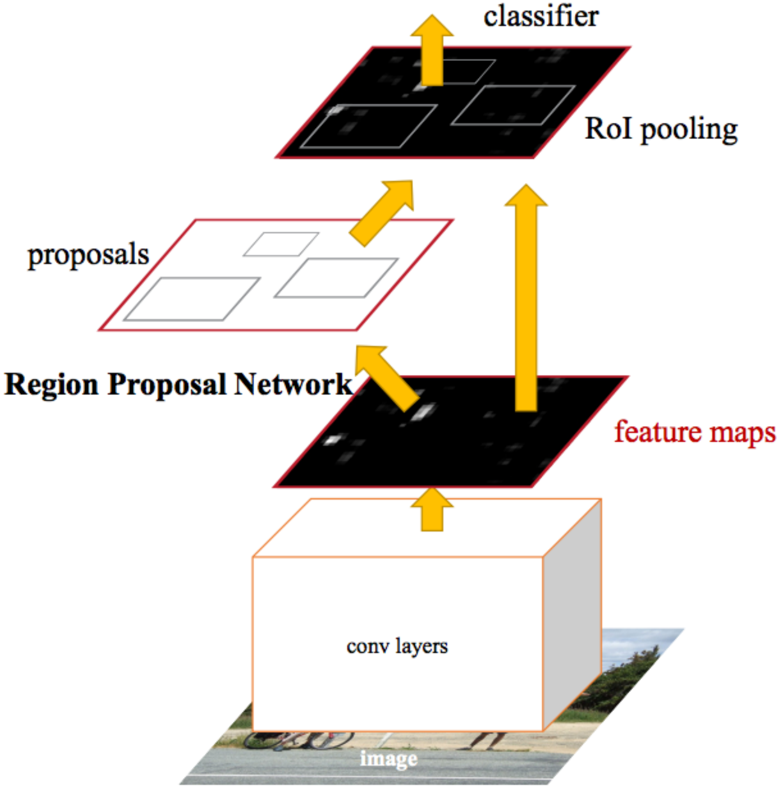
- Fast R-CNN
  - Shared computation of convolutional layers between proposals as a result of ROI pooling
  - Improvement in speed is not large because the region proposals are generated separately by another model



Girshick, Fast R-CNN, ICCV, 2015.

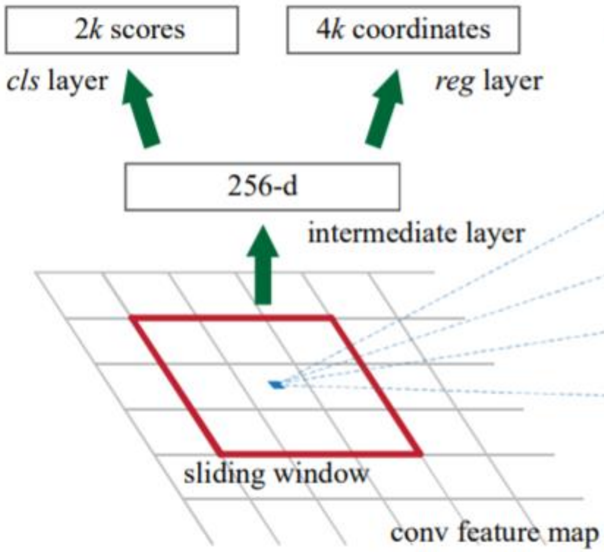
Advanced Models

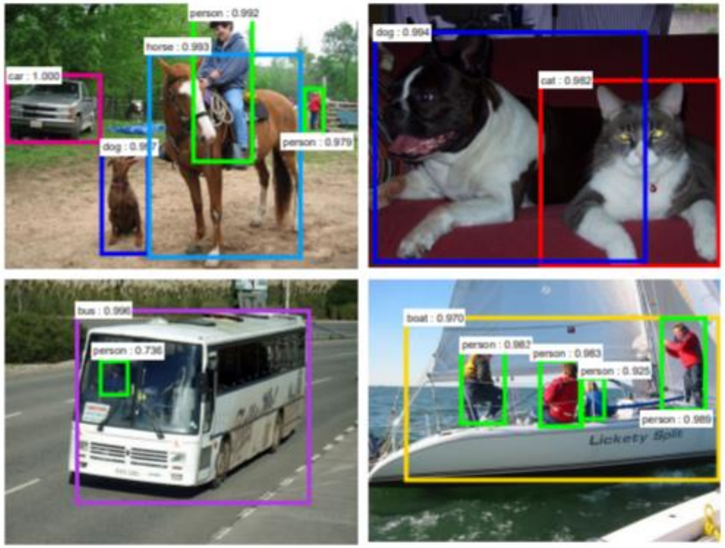
- Faster R-CNN
  - Fast R-CNN + Region Proposal Network (RPN)



Advanced Models

- Faster R-CNN
  - Fast R-CNN + Region Proposal Network (RPN)

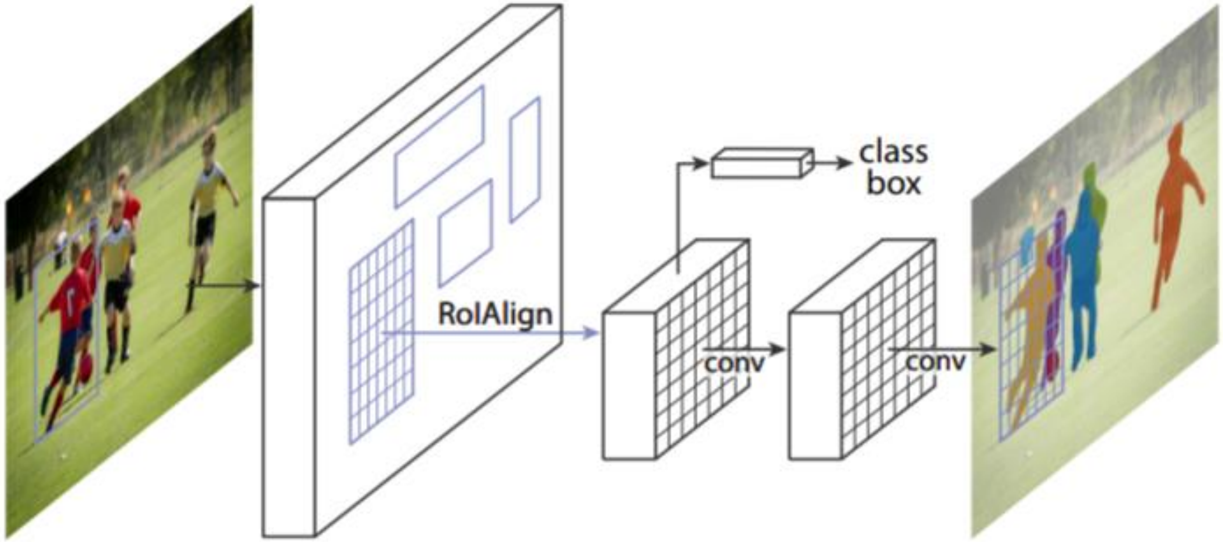




Ren et al., Faster R-CNN: towards real-time object detection with region proposal networks, NeurIPS, 2015.

Advanced Models

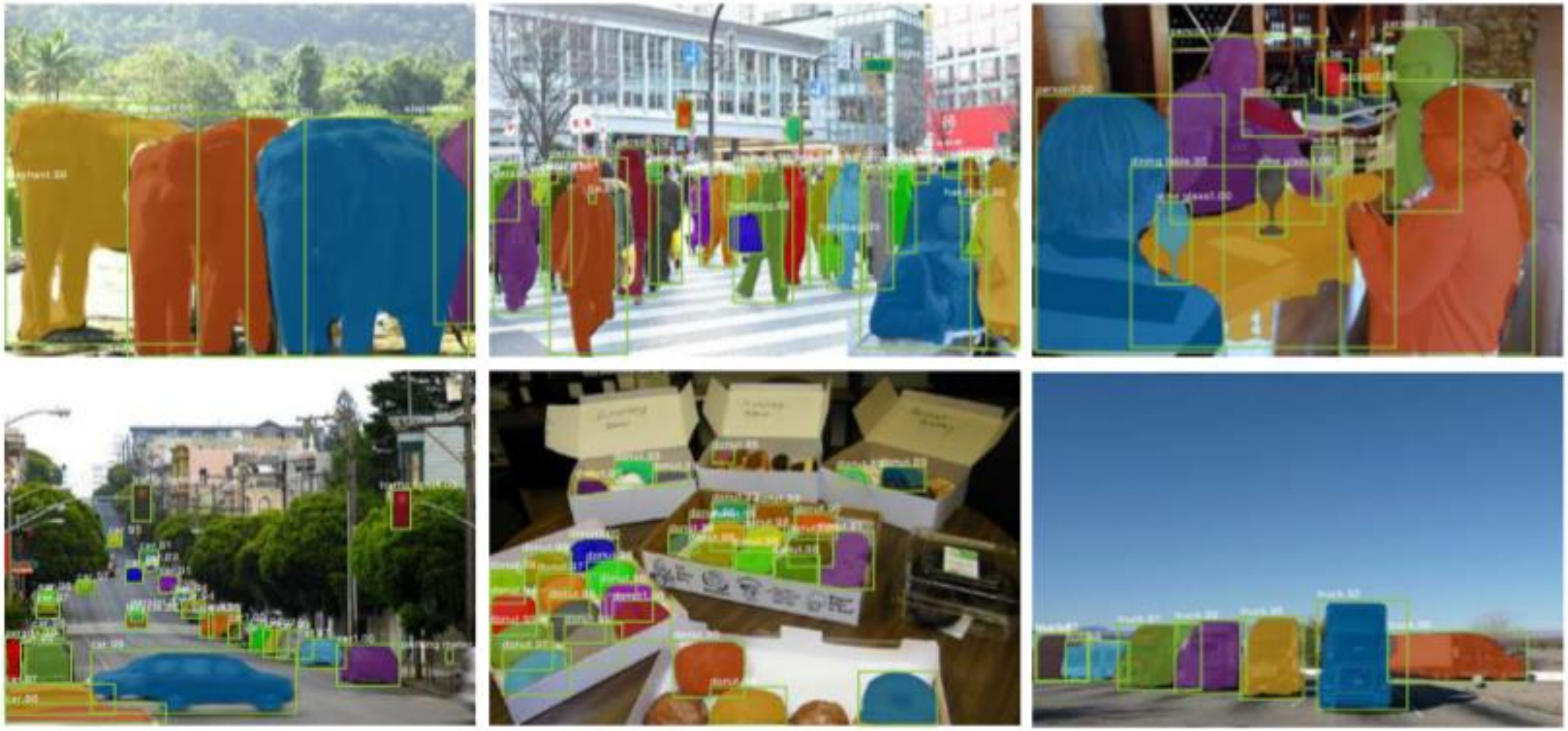
- Mask R-CNN
  - Convolutional backbone + RPN
  - Parallel heads for box regression
  - RoIAlign



He et al., Mask R-CNN, ICCV, 2017.

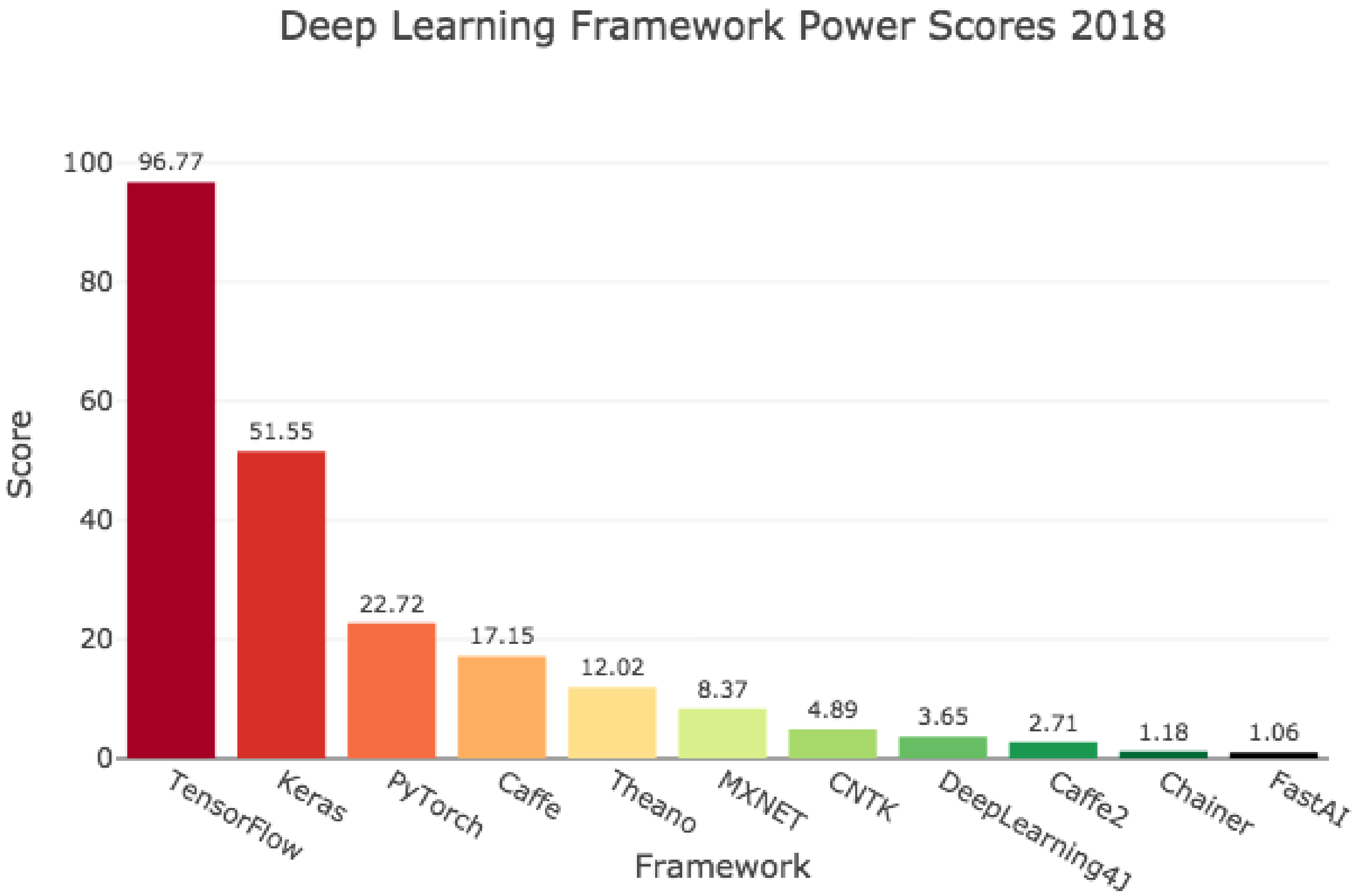
Advanced Models

- Mask R-CNN
  - Instance segmentation



He et al., Mask R-CNN, ICCV, 2017.

DL Frameworks



https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Summary

- A brief introduction of CNN – the most commonly used model of deep learning
- Widely used in computer vision studies
- In-depth knowledge in COMP9444