Deep Learning

Convolutional neural networks for image classification, feature extraction and reinforcement learning

Jan Zikeš

About me

- Masters degree in AI from CTU
- Machine Learning researcher at Spaceknow Inc.
- github/twitter: @ziky90

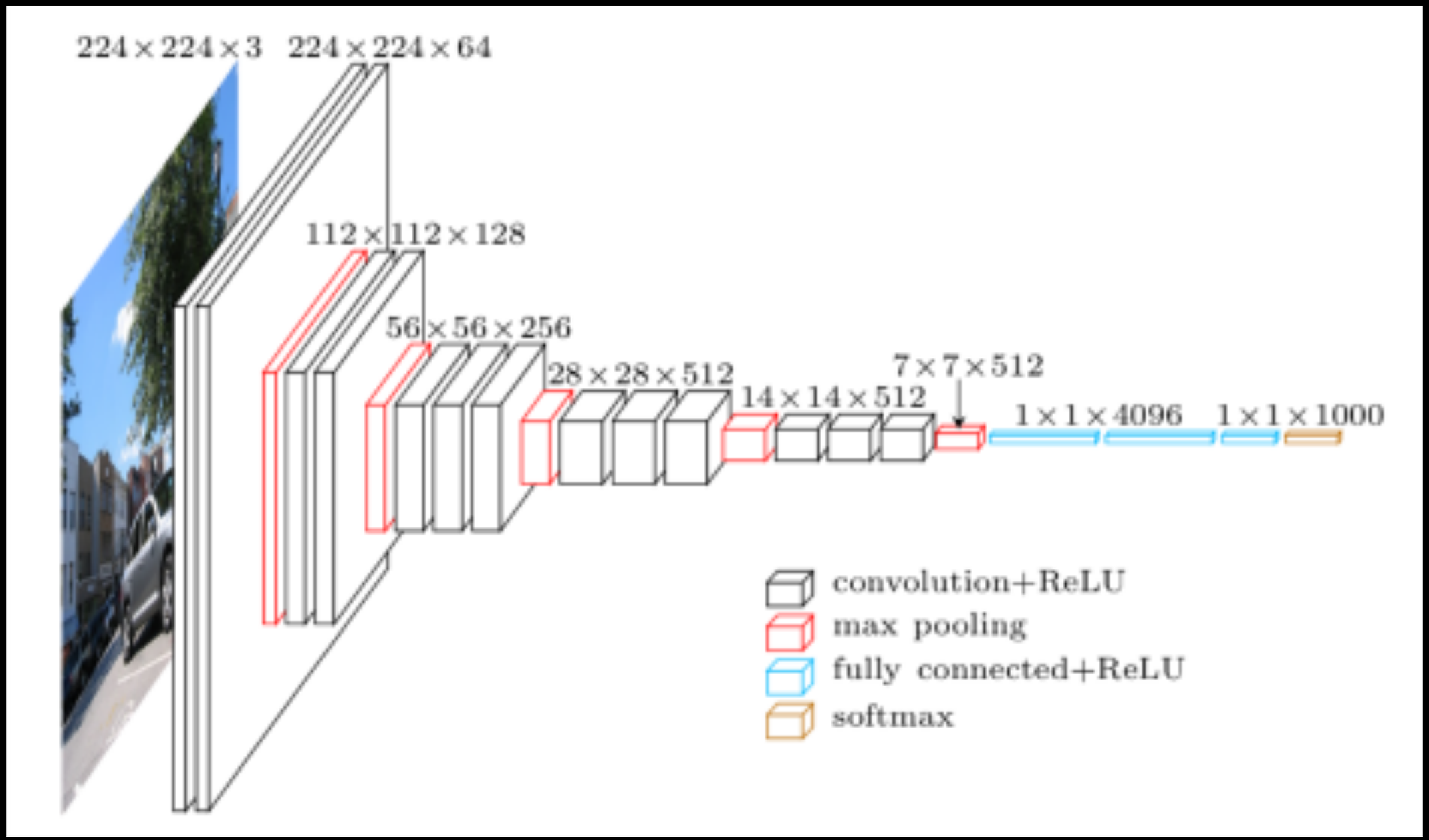
Knowledge check

- Supervised / unsupervised learning
- Perceptron / MLP
- CNNs AlexNet, VGG, ResNet

CNNs Image classification

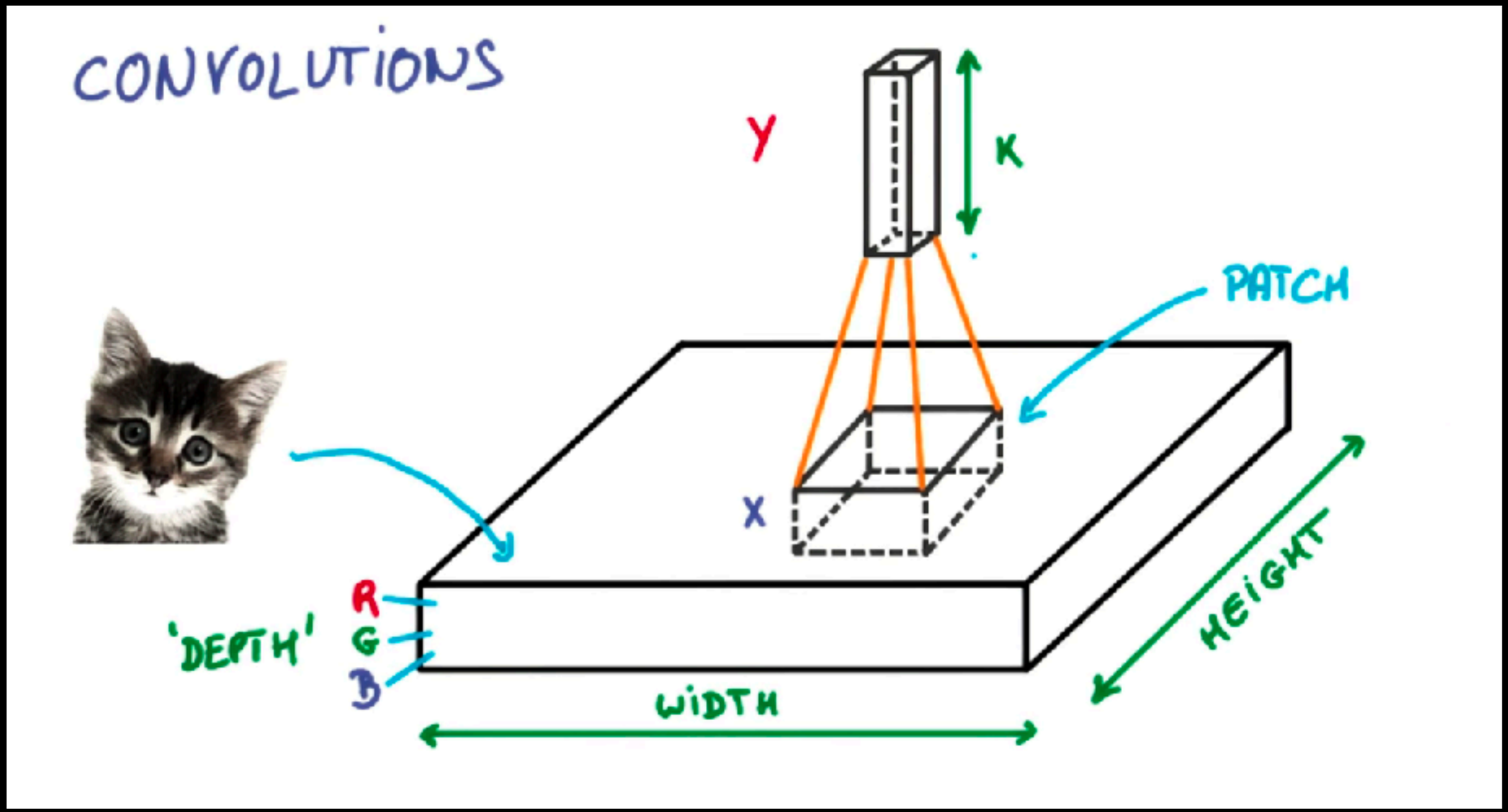
- Usually first task for transfer learning
- Enough for base feature extractor net
- Lot of available datasets
  - imagenet, pascal VOC, MS COCO

CNNs


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

VGG16

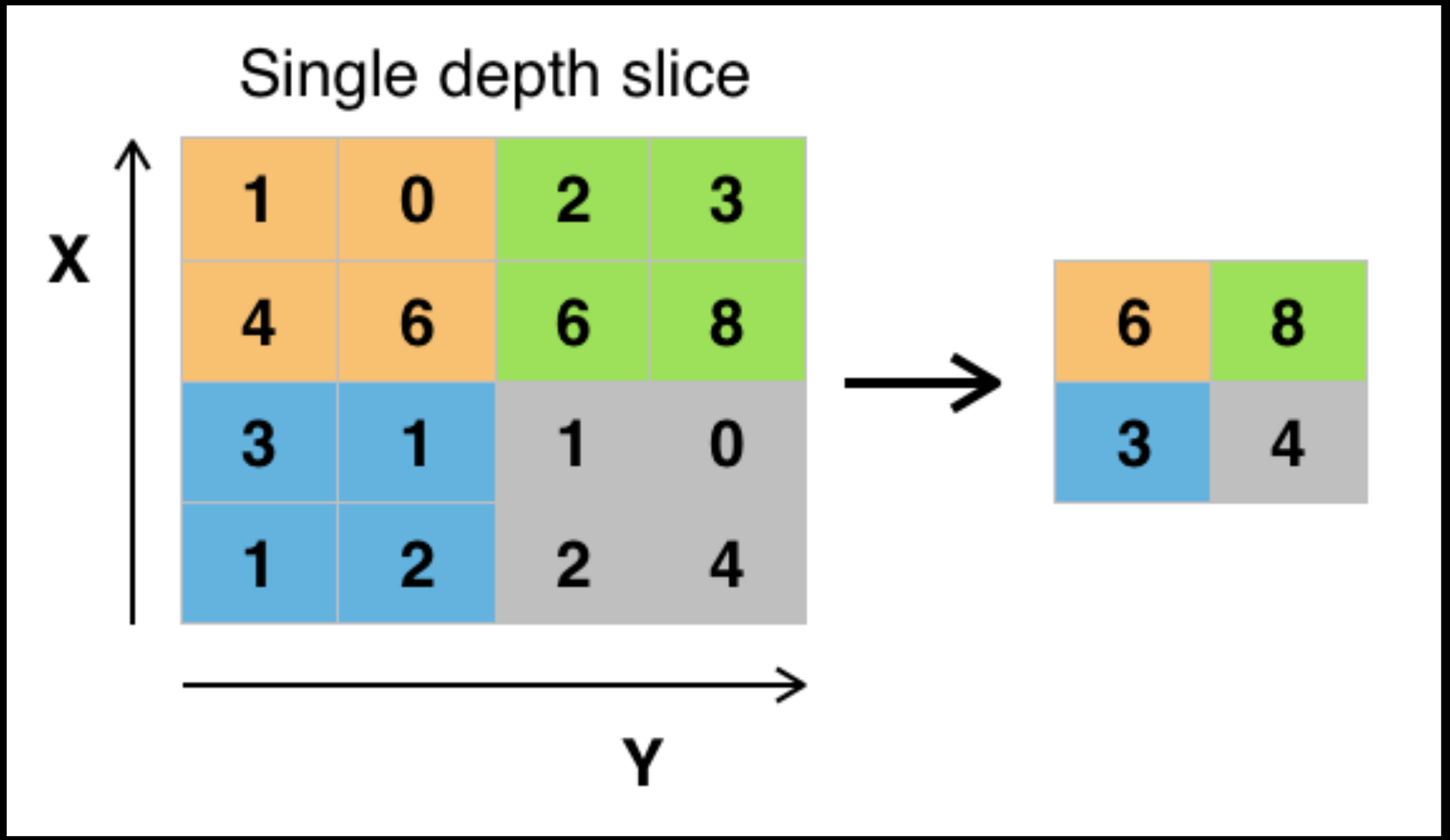
Convolution layers



Important params

- Filter size + output layer depth
- Strides
- Padding

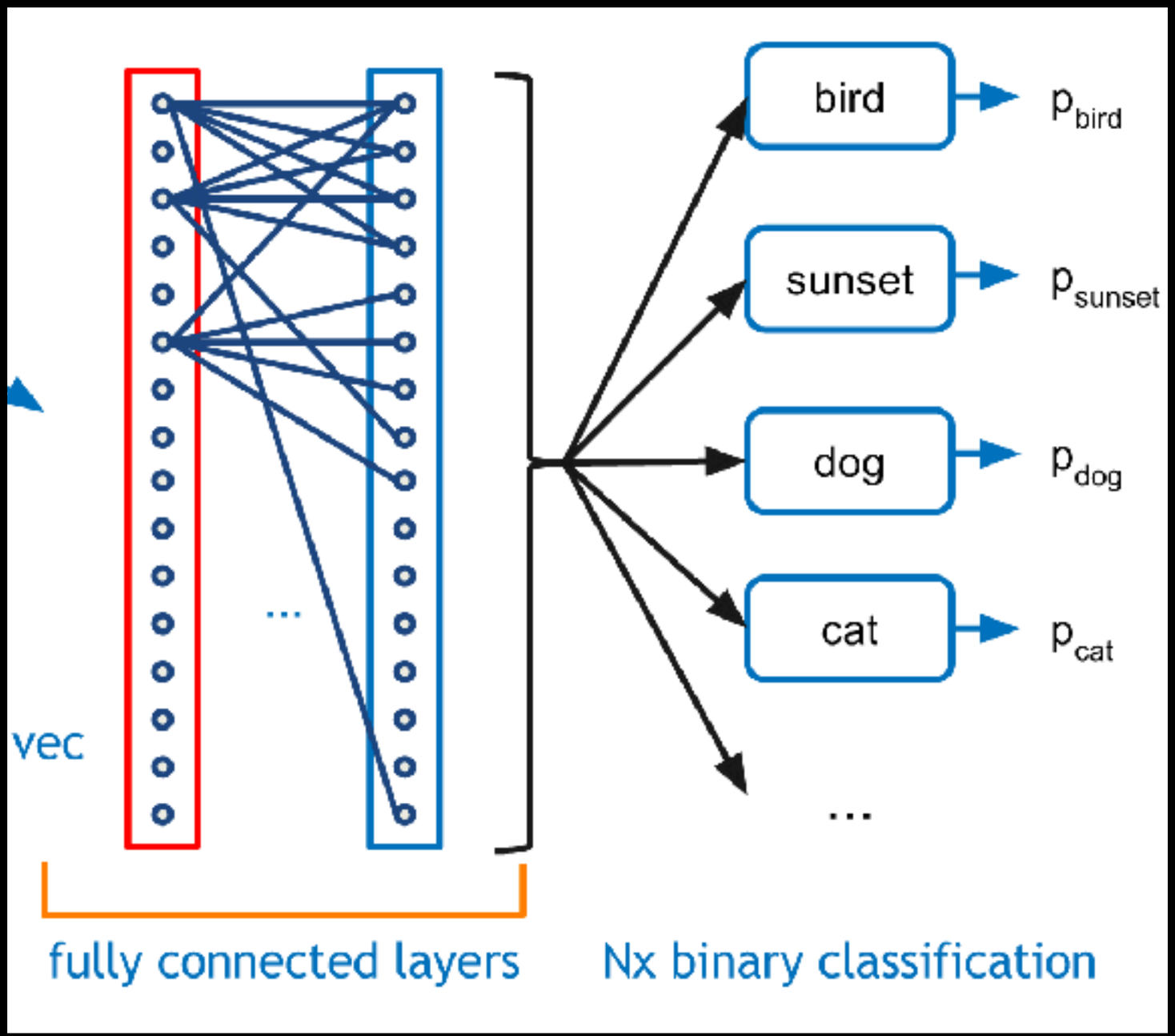
Pooling layers



Important params

- Pooling type (max, avg, fractional max/avg)
- Kernel size
- strides
- padding

Classifier at the end



We can improvise here

- Since we have features extracted we can improvise
- Common are fully connected layers followed by SoftMax
- But we can use features and train for example SVM on top of them

Architectures overview

- AlexNet
- VGG (2014)
- Inception
- ResNet (2015)
- DenseNet (2016)

AlexNet

- by Alex Krizhevsky of University of Toronto

- Classification on ImageNet (16.4% test error, top5)

VGG16

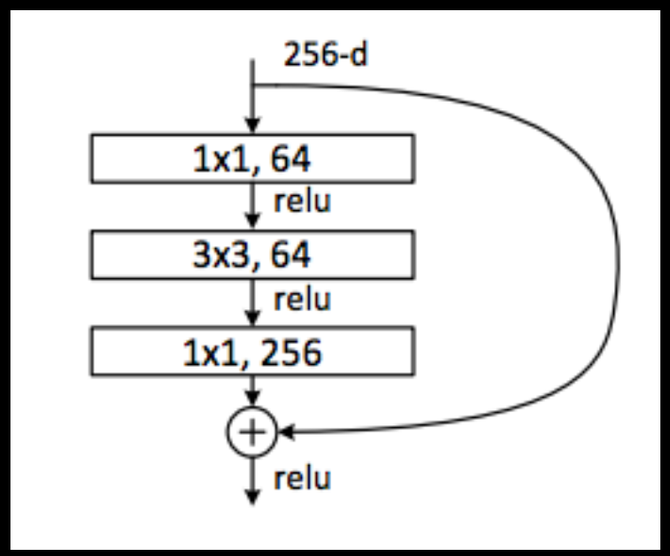
- By Visual Geometry Group at University of Oxford
- 138M parameters
- 7.4% test error on ImageNet (top5)
- Available pre-trained model in Caffe, Tensorflow and others

Inceptions

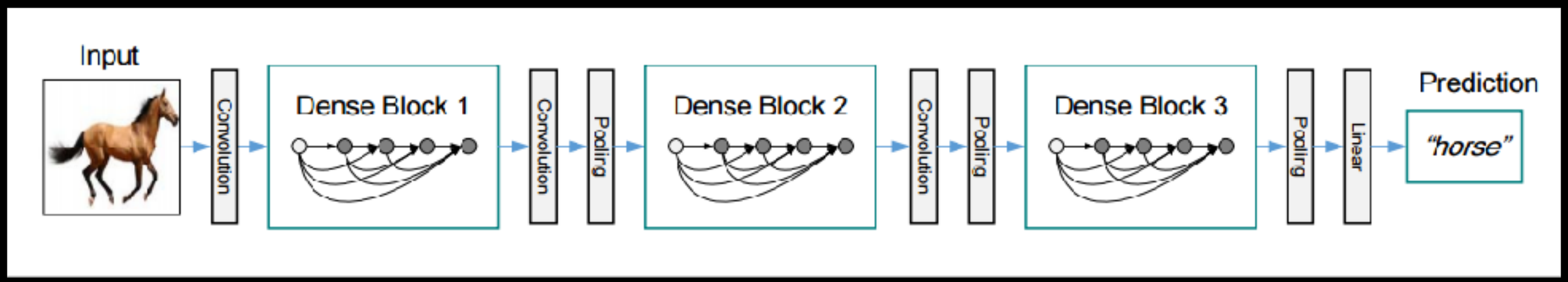
- Set of various architectures by Google
- Available pre-trained model in Tensorflow
- Not Good as a feature extractor
- Tuned a lot for ImageNet

ResNet

- By Microsoft research
- Available pre-trained model in Torch, Keras and recently also in Tensorflow
- 110 layers -> 1.7M params
- 4.6% top5 error (ResNet-101)
- 3.57% ensemble of ResNets



DenseNet



- Comparable with ResNet, but with less parameters

- https://arxiv.org/pdf/1608.06993v1.pdf

HW enablers

- ANNs are known techniques for a while
- GPUs are enabling training in reasonable time (usually days - weeks)
- benchmarks: https://github.com/jcjohnson/cnn-benchmarks

ResNet time performance

ResNet-50

(input 16 x 3 x 224 x 224)

This is the 50-layer model described in [4] and implemented in fb.resnet.torch.

GPU	cuDNN	Forward (ms)	Backward (ms)	Total (ms)
Pascal Titan X	5.1.05	35.03	68.54	103.58
Pascal Titan X	5.0.05	35.03	70.76	105.78
GTX 1080	5.1.05	50.64	99.18	149.82
GTX 1080	5.0.05	50.76	103.35	154.11
Maxwell Titan X	5.1.05	55.75	103.87	159.62
Maxwell Titan X	5.0.05	56.30	109.75	166.05
Maxwell Titan X	4.0.07	62.03	116.81	178.84
Pascal Titan X	None	87.62	158.96	246.58
GTX 1080	None	109.79	201.40	311.18
Maxwell Titan X	None	137.14	247.65	384.79
CPU: Dual Xeon E5-2630 v3	None	2477.61	4149.64	6627.25

https://github.com/jcjohnson/cnn-benchmarks

Practical tips for quick prototype

- Use some existing architecture with existing implementation
- Use transfer learning
- Use GPUs and give training some time (1-20 epochs at least)

Practical tips for production stuff

- Always look on the Bias/Variance tradeoff
- Maintain experiment replicability and results!!!
- Feel free to experiment with anything (parameters, nonlinearities, architecture, data)
- Unless you work for Google / FB or Stanford you'll not have enough resources for grid search, use your reinforced intuition :)

Questions?