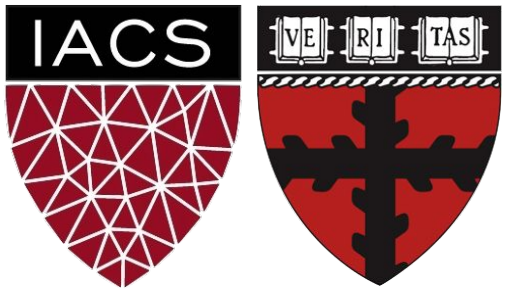
Lecture 6-7-8: Deep Learning - Computer Vision

Advanced Practical Data Science, MLOps

AC295

Pavlos Protopapas
Institute for Applied Computational Science, Harvard



Outline

1. Introduction to Transfer Learning
2. Review CNNs
3. SOTA Deep Models
4. Transfer Learning across Tasks
5. Tutorial: Segmentation
6. Model Compression Techniques
7. Tutorial: Model Compression
8. Tutorial: Mushroom App Models

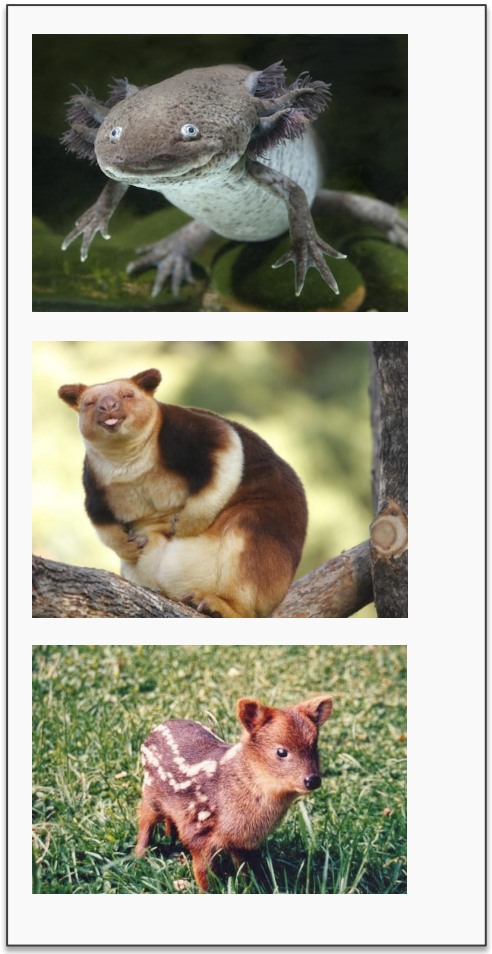
Outline

1. **Introduction to Transfer Learning**
2. Review CNNs
3. SOTA Deep Models
4. Transfer Learning across Tasks
5. Tutorial: Segmentation
6. Model Compression Techniques
7. Tutorial: Model Compression
8. Tutorial: Mushroom App Models

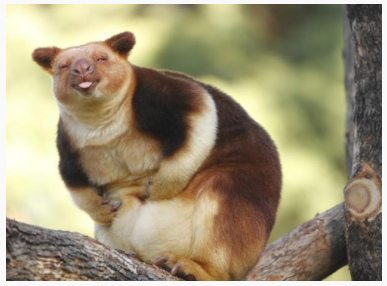
Communication

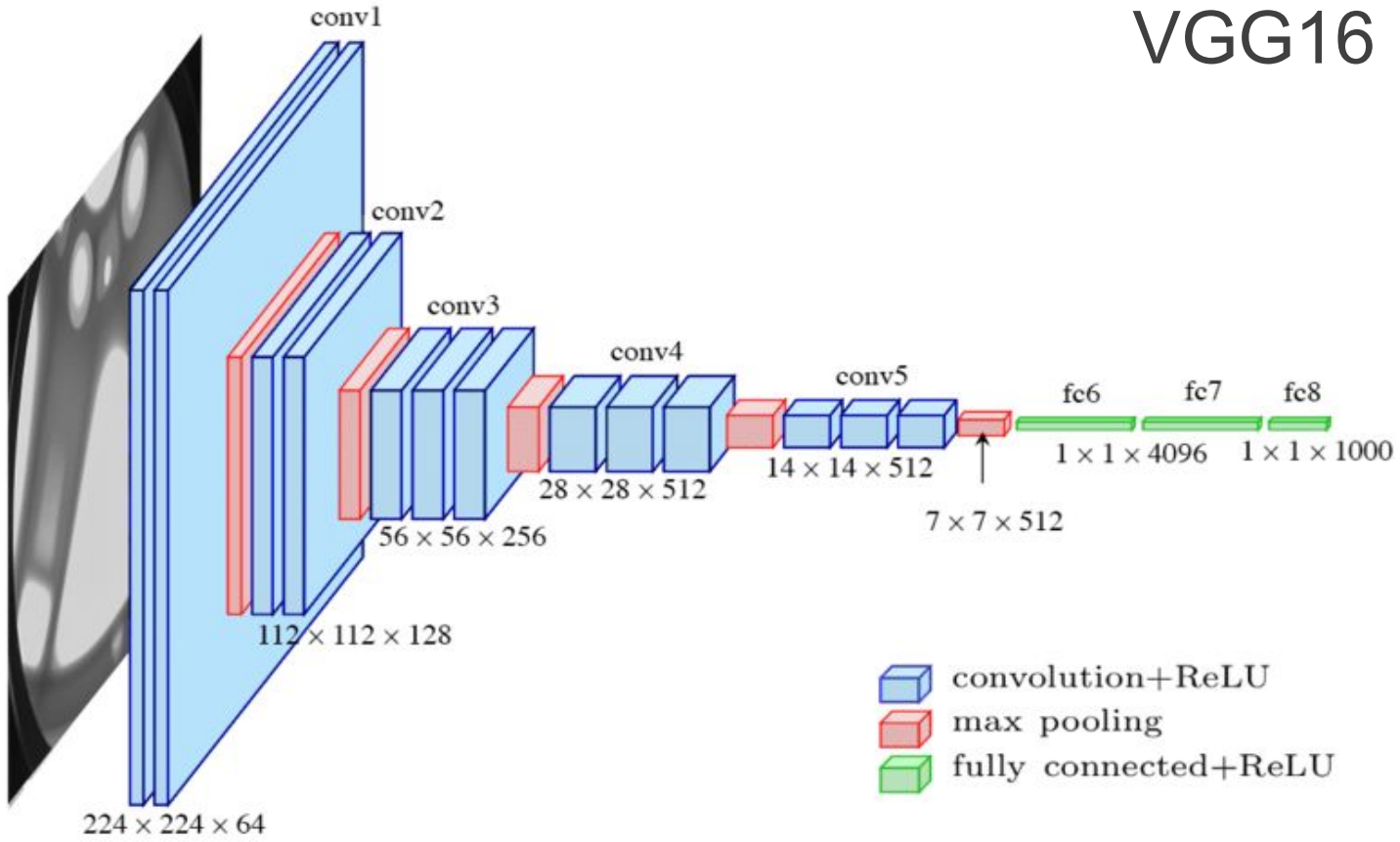
- We received all milestones. Thank you. Exercise 1 and milestone 1 will be graded by Fri/Sat.
- Exercise 2 due 09/23
- Please make sure outputs are visible in exercise notebooks. (There will be penalties otherwise)
- Quiz 4/5 due 10/05

Motivation: Classify Rarest Animals



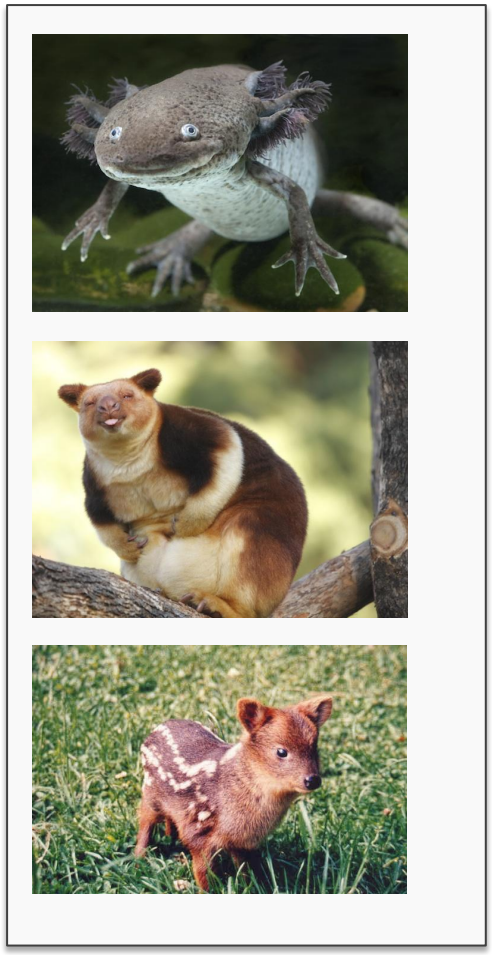
Classify Rarest Animals



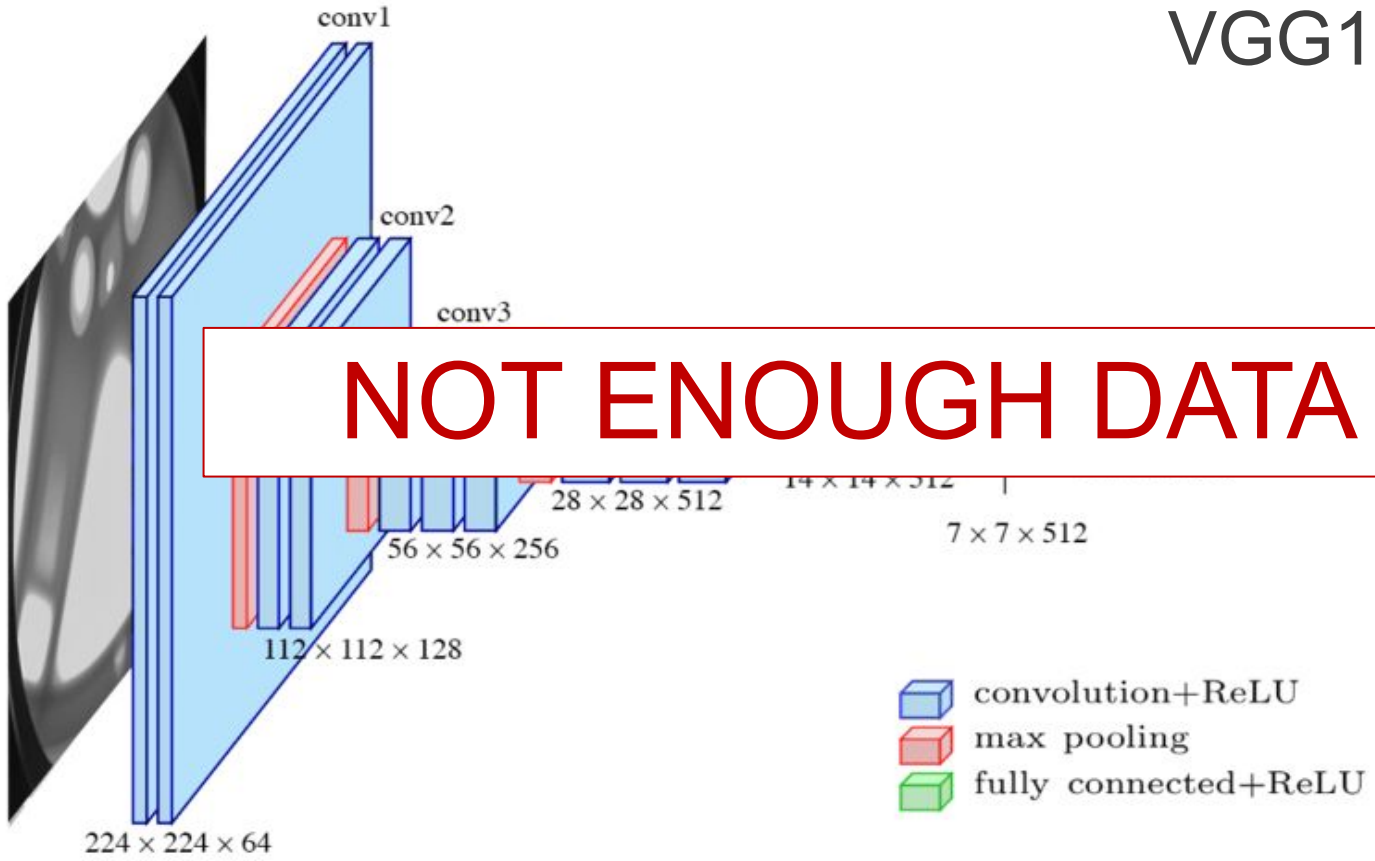


Number of parameters: 134,268,737
Data Set: Few hundred images

Classify Rarest Animals



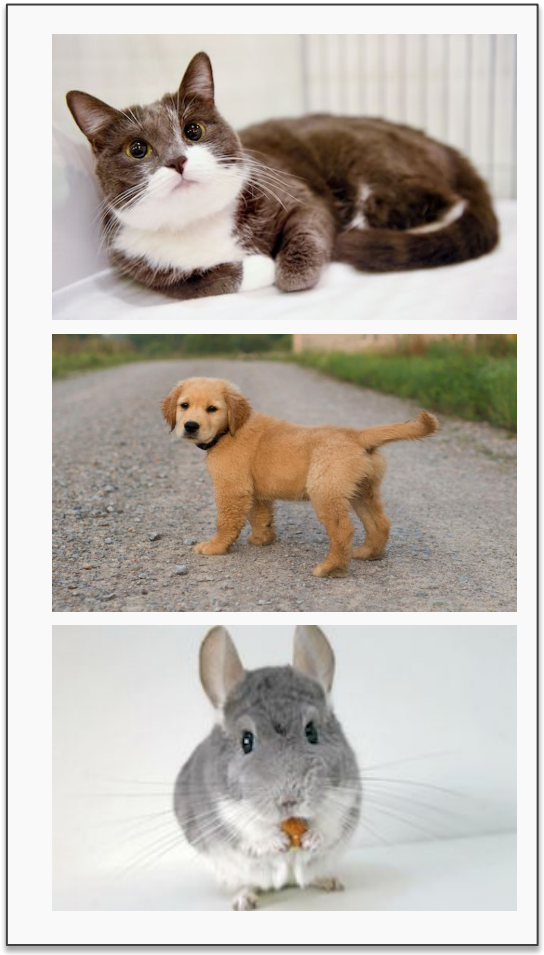
VGG16



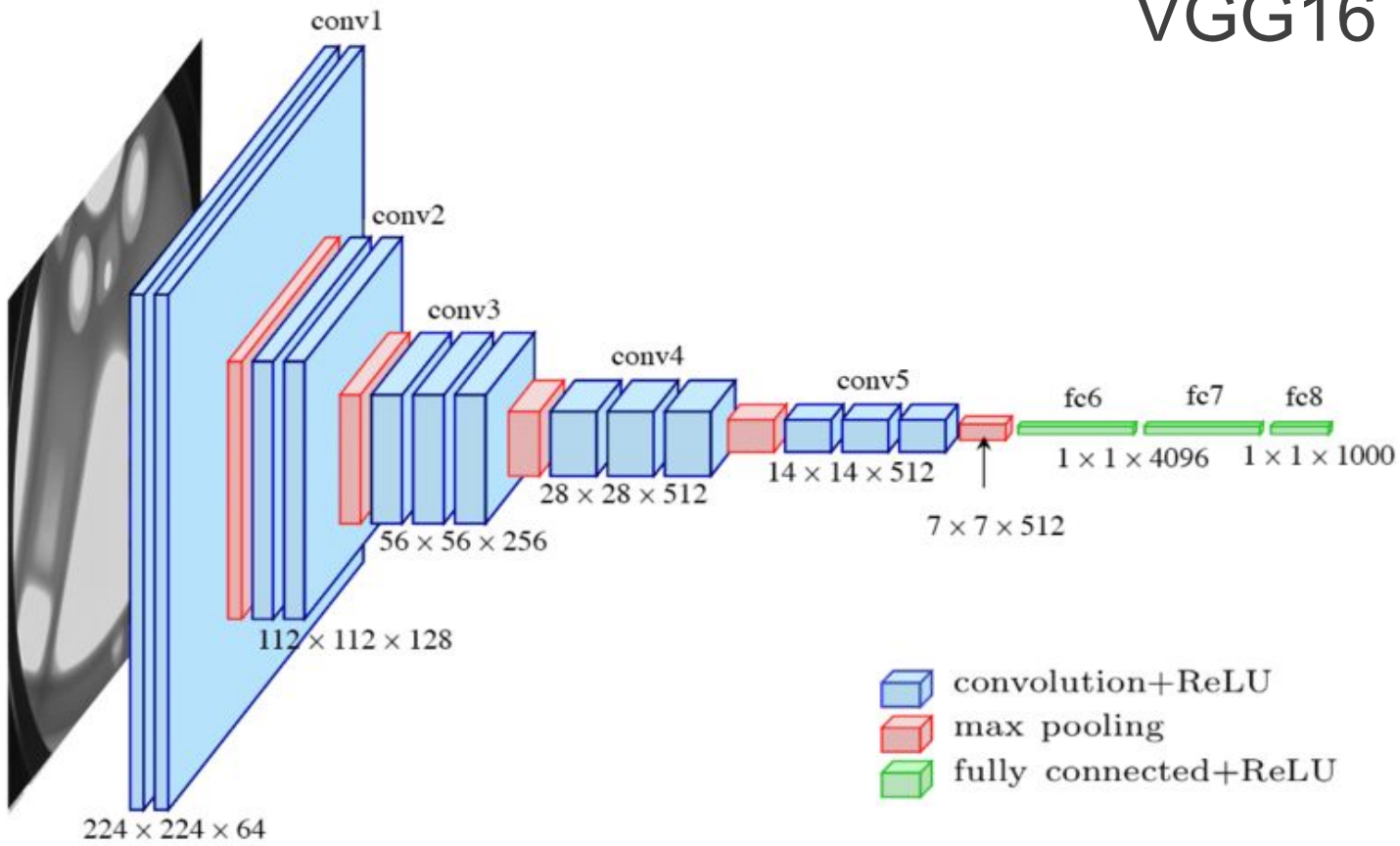
NOT ENOUGH DATA

Number of parameters: 134,268,737
Data Set: Few hundred images

Classify Cats, Dogs, Chinchillas etc

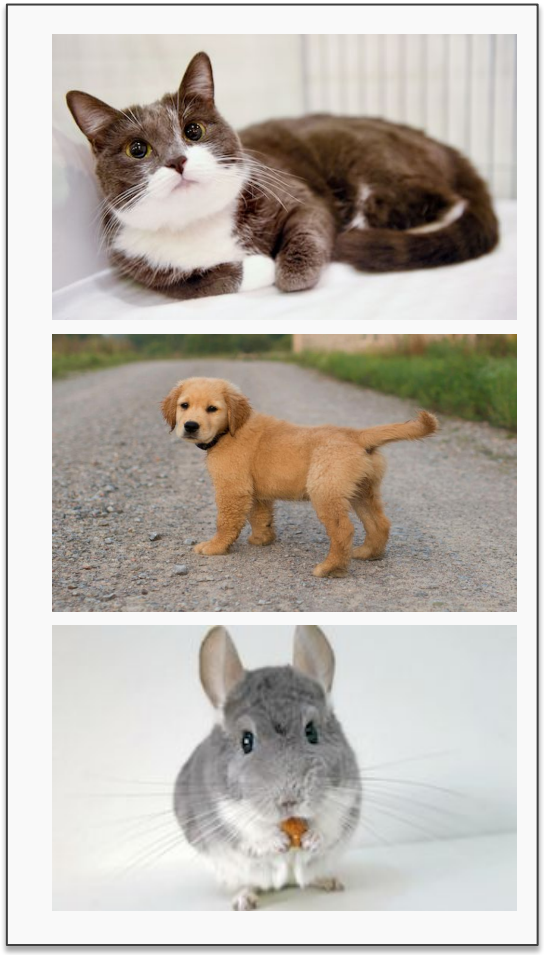


VGG16

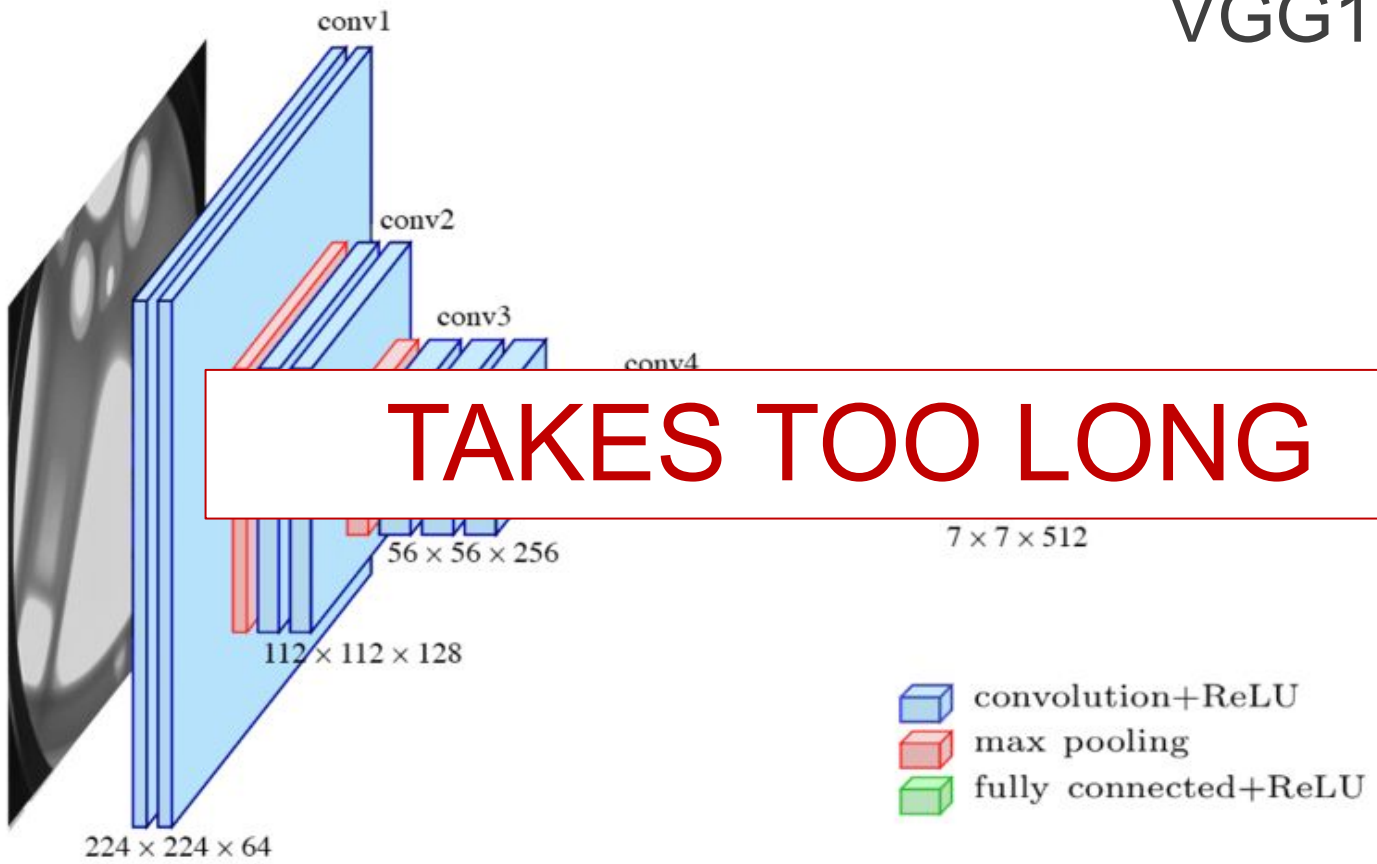


Number of parameters: 134,268,737
Enough training data. ImageNet approximate 1.2M

Classify Cats, Dogs, Chinchillas etc



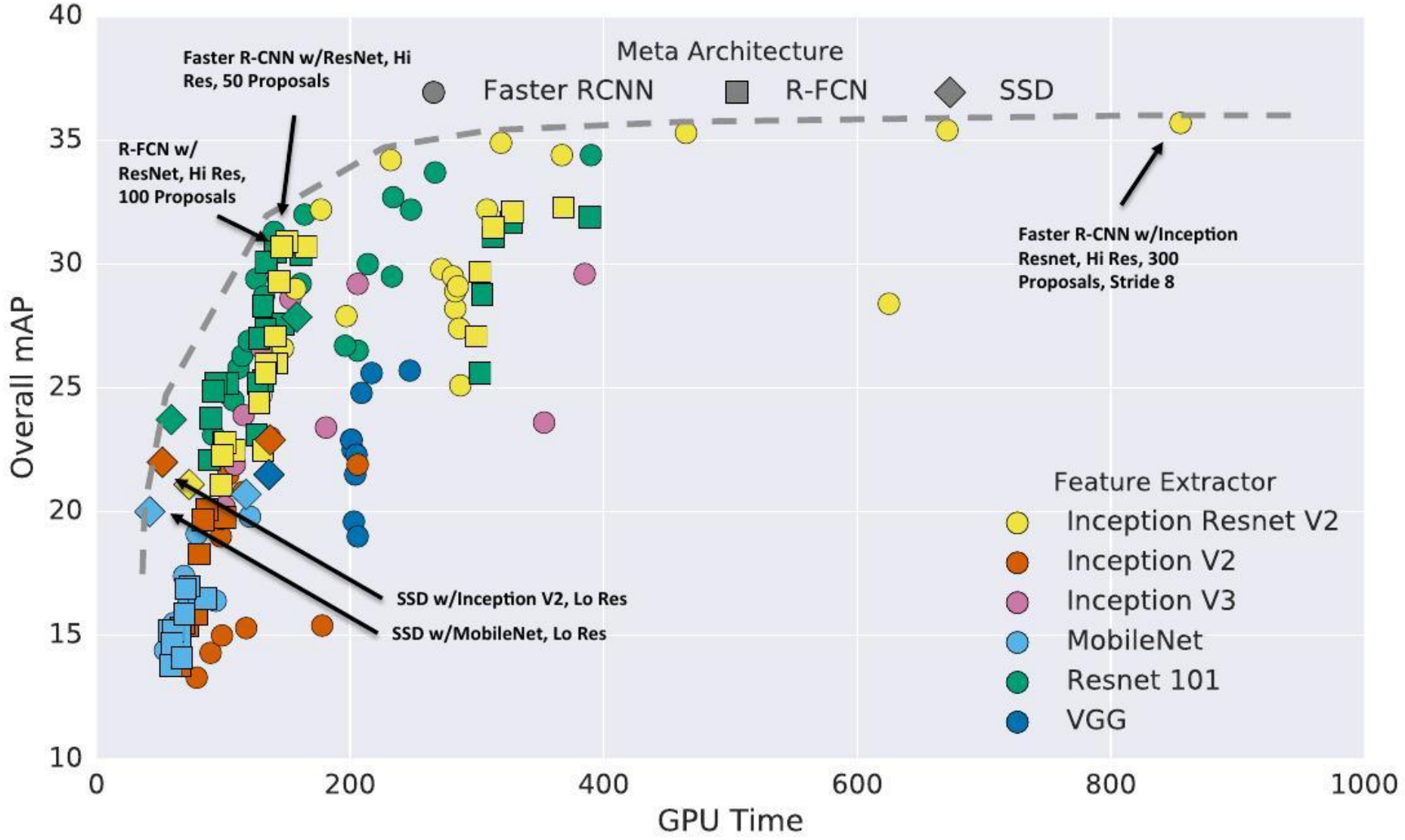
VGG16



TAKES TOO LONG

Number of parameters: 134,268,737
Enough training data. ImageNet approximate 1.2M

Training Time for SOTA Models



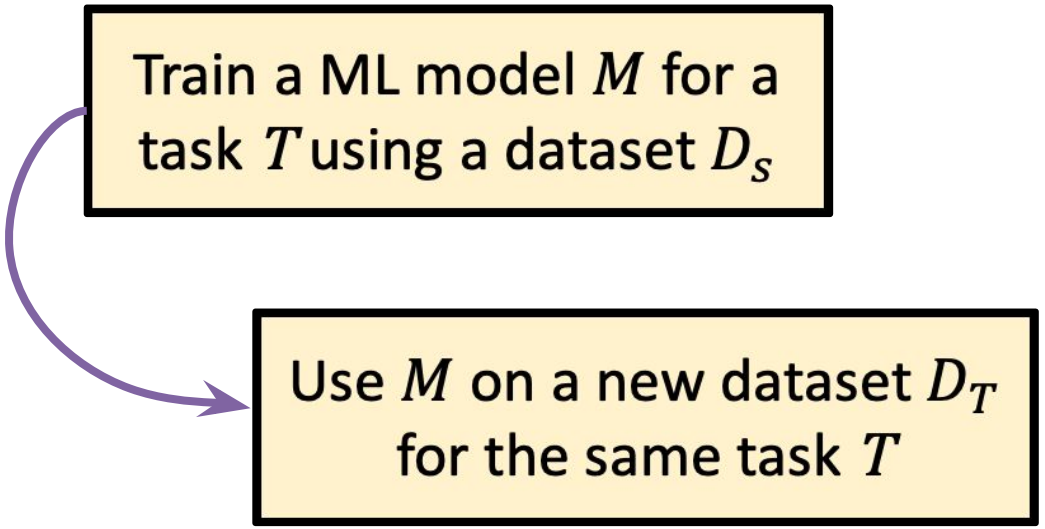
Transfer Learning To The Rescue

How do you build an image classifier that can be trained in a few minutes on a CPU with very little data?

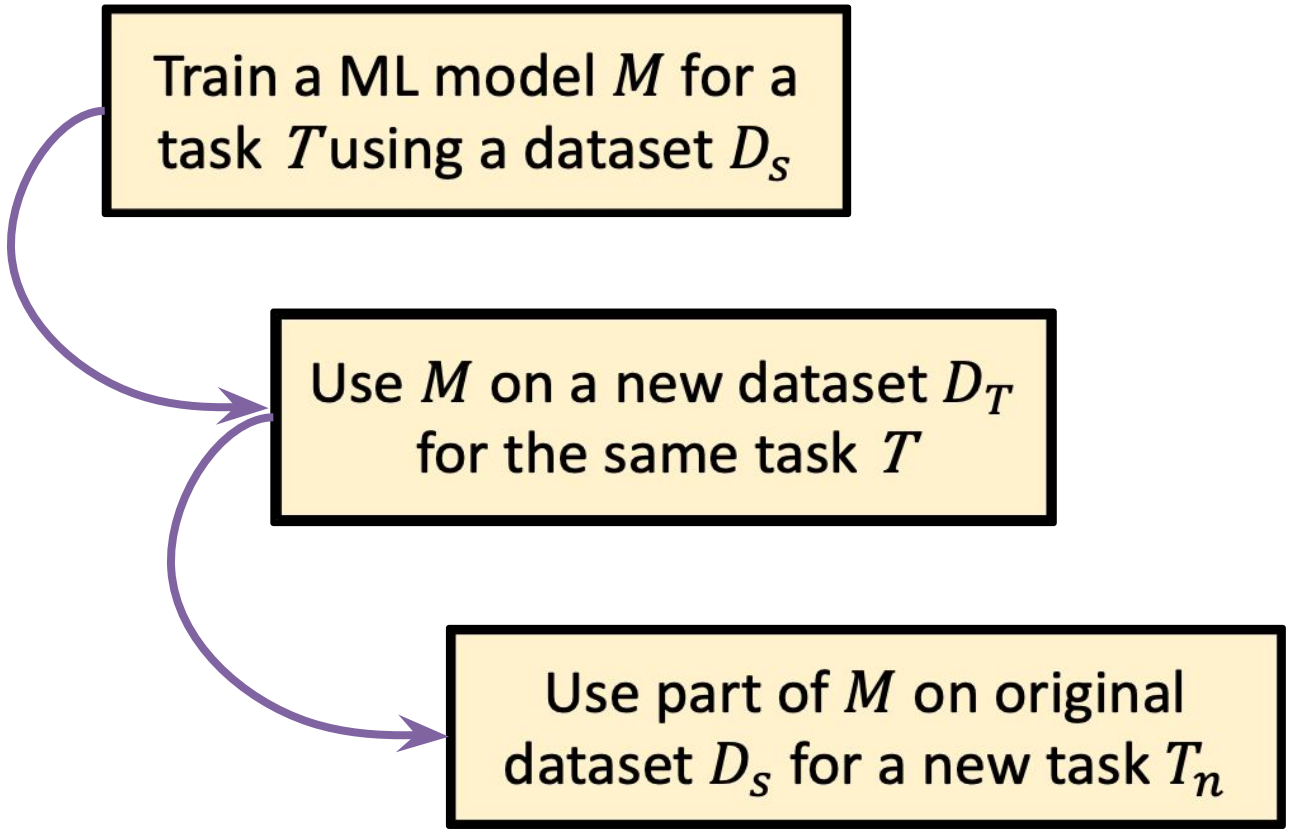
Basic Idea of Transfer Learning

Train a ML model M for a task T using a dataset D_s

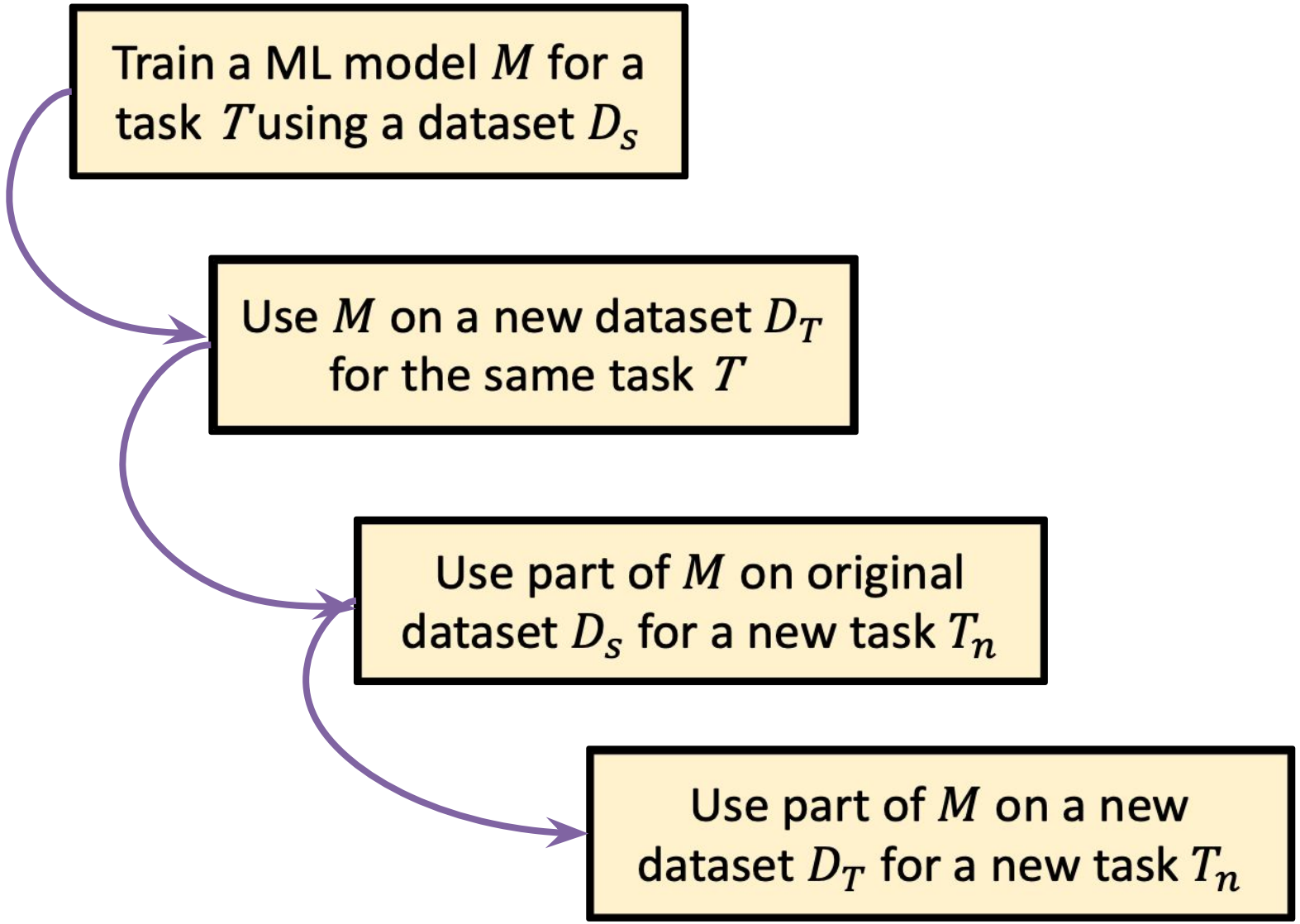
Basic Idea of Transfer Learning



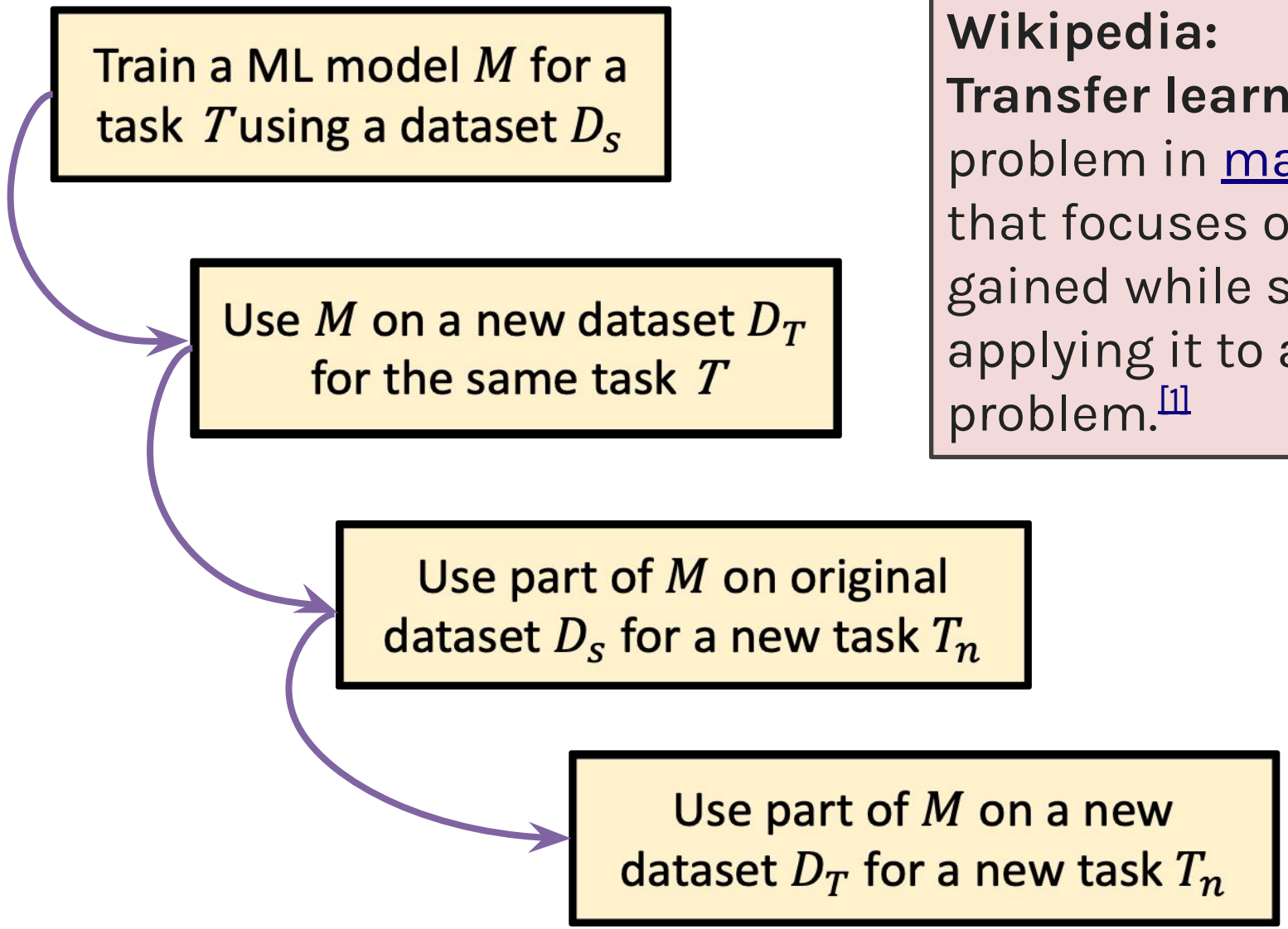
Basic Idea of Transfer Learning



Basic Idea of Transfer Learning



Basic Idea of Transfer Learning



Wikipedia:
Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.[1]

Basic Idea of Transfer Learning

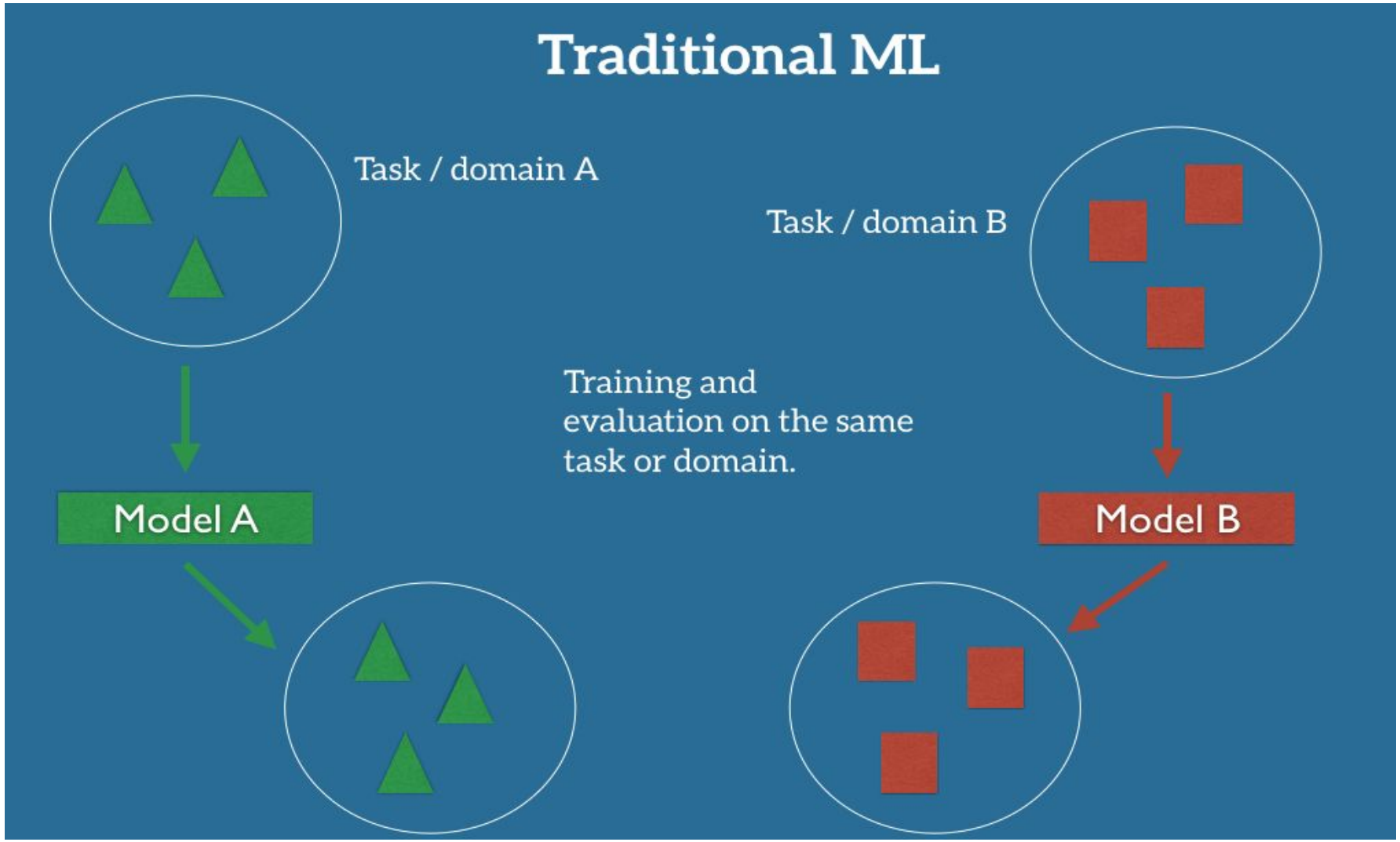
How do you make **an image classifier** that can be **trained in** a few minutes on a CPU with very little data?

Basic Idea of Transfer Learning

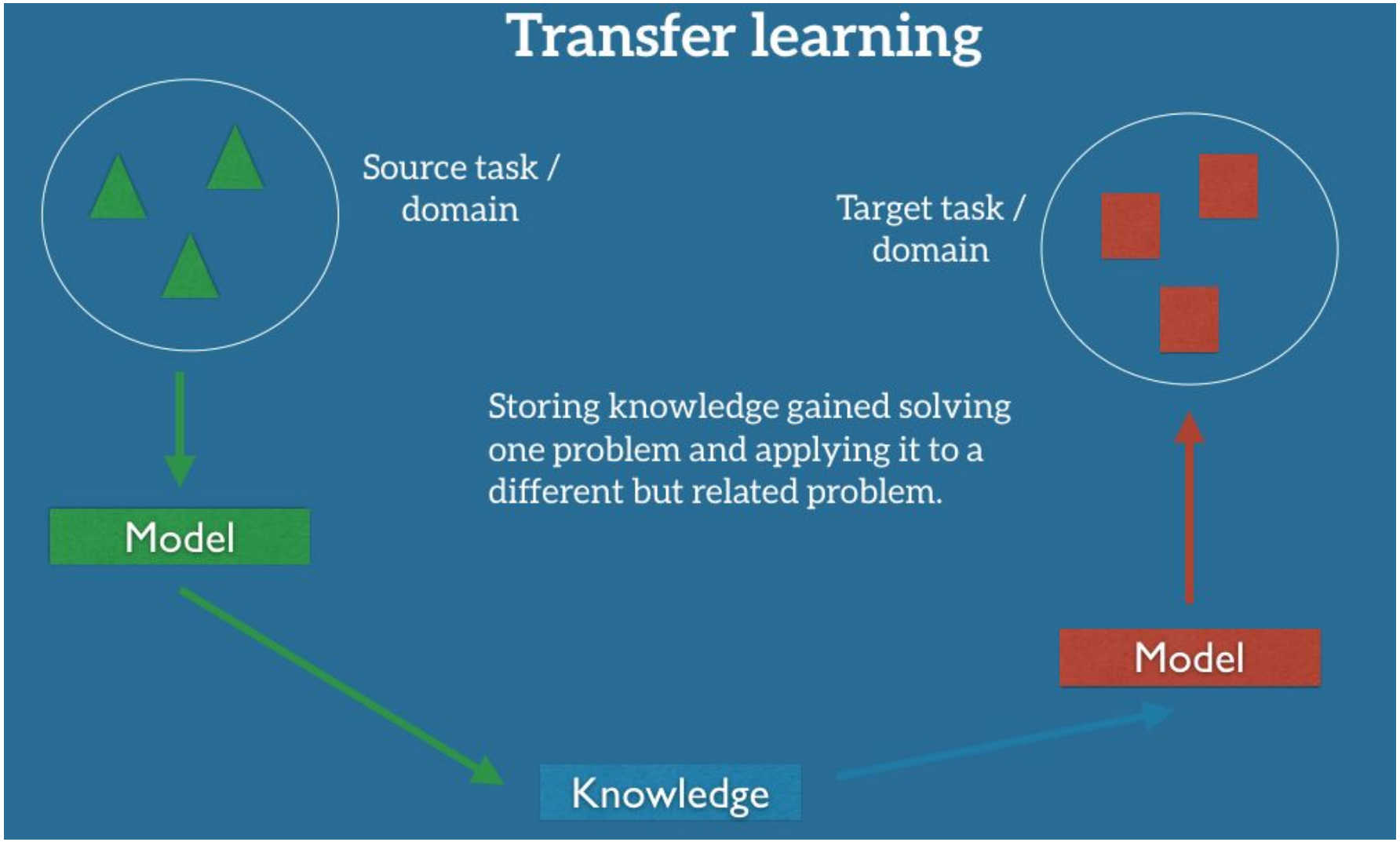
How do you make **an image classifier** that can be **trained in** a few minutes on a CPU with very little data?

Use pre-trained models, i.e., models with known weights.

Basic Idea of Transfer Learning



Basic Idea of Transfer Learning



Basic Idea of Transfer Learning

How do you make **an image classifier** that can be **trained in** a few minutes on a CPU with very little data?

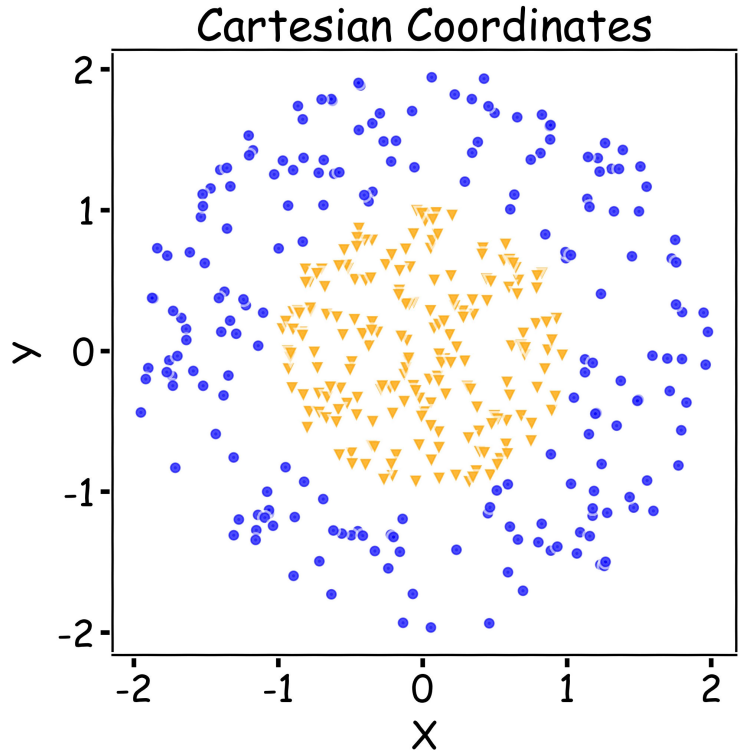
Use pre-trained models, i.e., models with known weights.

Main Idea: Earlier layers of a network learn low level features, which can be adapted to new domains by changing weights at later and fully-connected layers.

Example: Use ImageNet trained with any sophisticated huge network. Then retrain it on a few images.

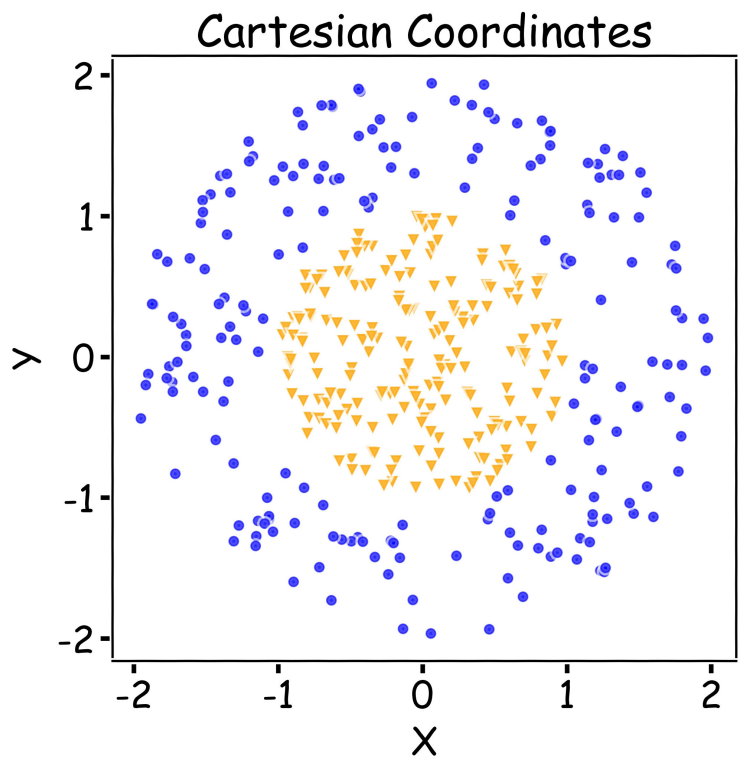
Key Idea: Representation Learning

Relatively difficult task



Key Idea: Representation Learning

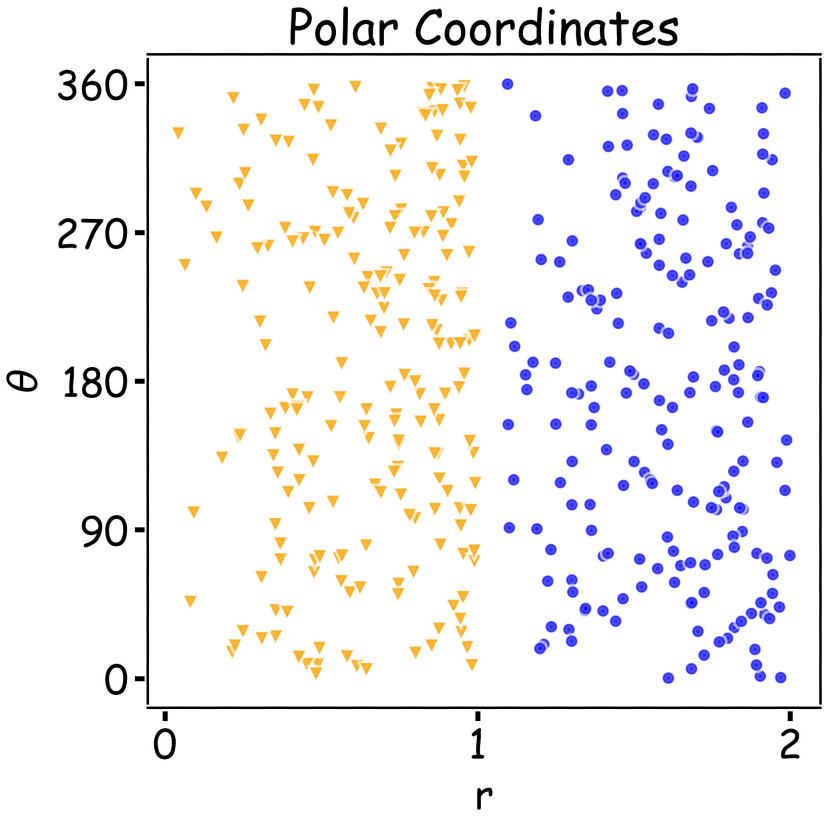
Relatively difficult task



Tranform:

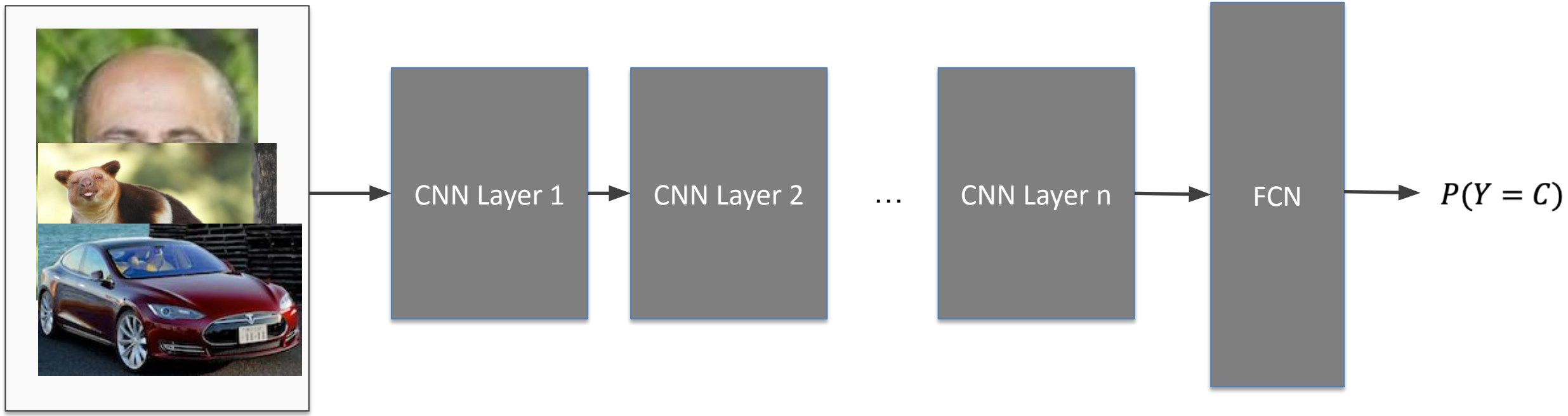
$(X, Y) \rightarrow (r, \theta)$

Easier task



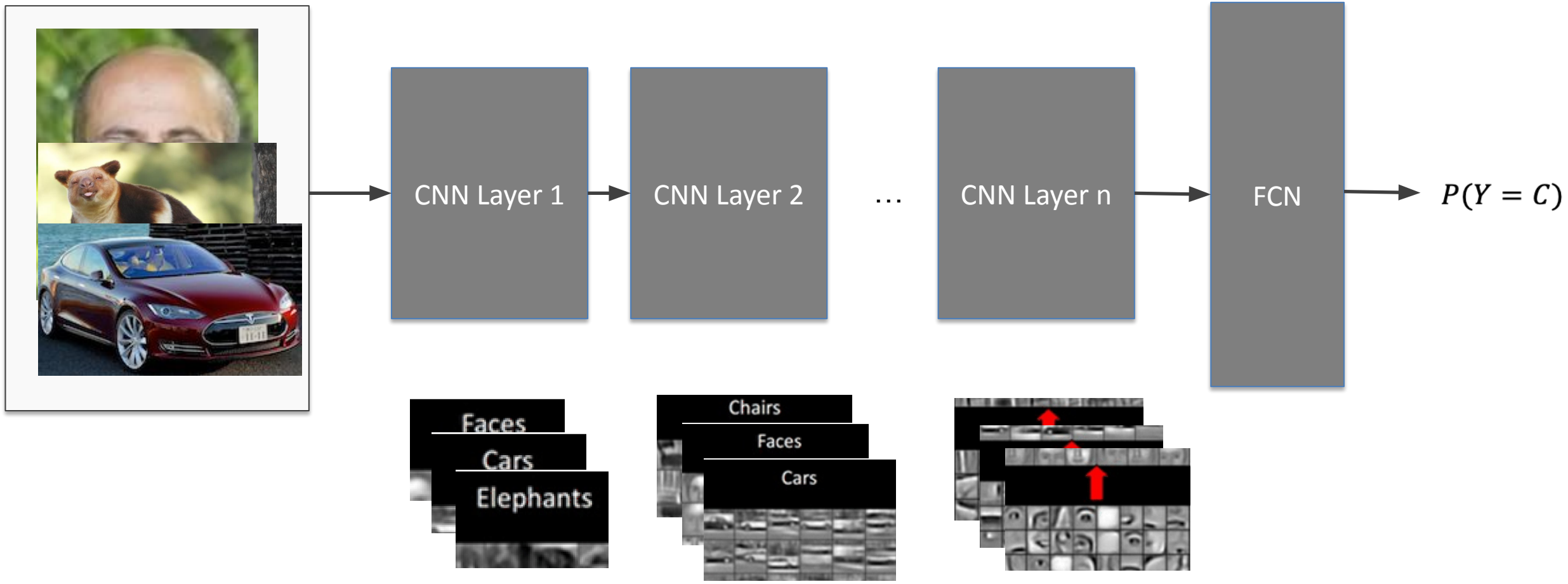
Representation Learning

Task: classify cars, people, animals and objects



Representation Learning

Task: classify cars, people, animals and objects



Basic Idea of Transfer Learning

- Train on a big "**source**" data set, with a big model, on one downstream tasks (say classification). Do it once and save the parameters. This is called a **pre-trained model**.
- Use these parameters for other smaller "**target** " datasets, say, for classification on new images (possibly different **domain**, or training distribution), or for image segmentation on old images (new **task**), or new images (new task and new domain).

Basic Idea of Transfer Learning

- Train on a big "**source**" data set, with a big model, on one downstream tasks (say classification). Do it once and save the parameters. This is called a **pre-trained model**.
- Use these parameters for other smaller "**target** " datasets, say, for classification on new images (possibly different **domain**, or training distribution), or for image segmentation on old images (new **task**), or new images (new task and new domain).
- Less helpful if you have a large target dataset with many labels.
- Will fail if source domain (where you trained big model) has nothing in common with target domain (that you want to train on smaller data set).

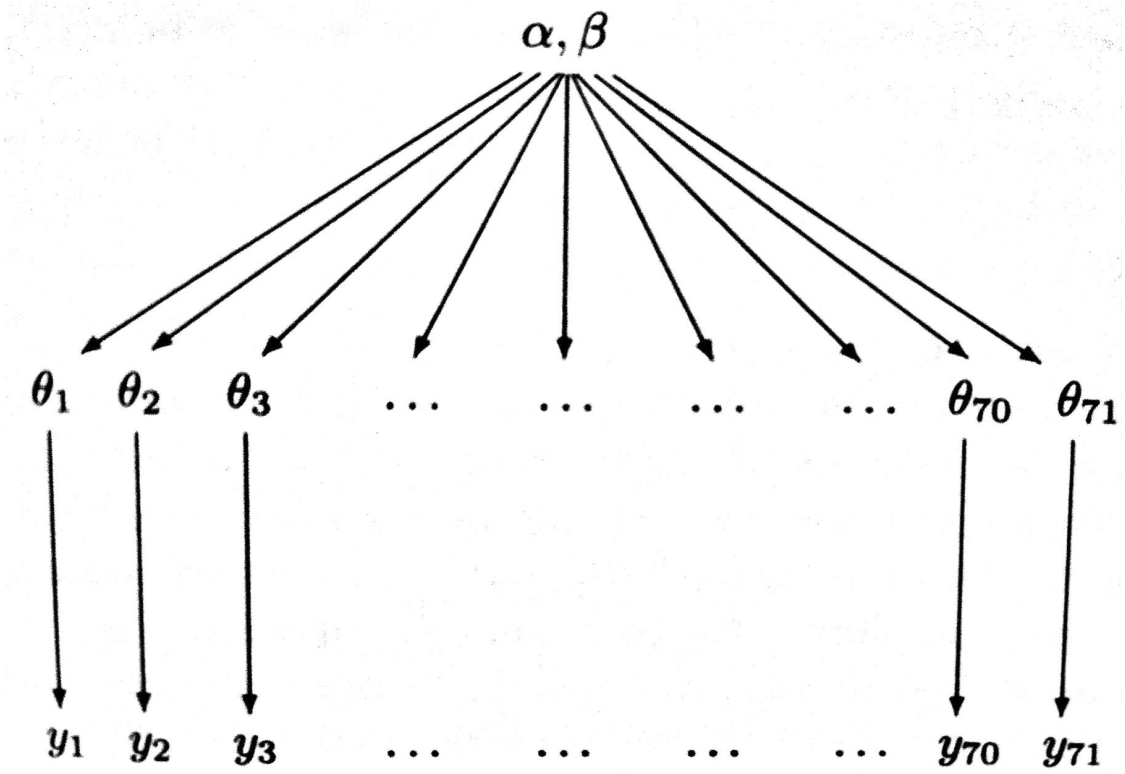
Basic Idea of Transfer Learning

Not a new idea!

It has been there in the ML and stats literature for a while.

- An exemplar is hierarchical **glm models** in stats, where information flows from higher data units to lower data units to the lower data.

- Neural networks **learn hierarchical representations** and thus are particularly suited to this kind of learning. Furthermore, since we learn representations, we can deal with domain adaptation/covariate shift.



Transfer Learning for Deep Learning

What people think

- you can't do deep learning unless you have a million labeled examples.

What people can do, instead

- **You can learn representations from unlabeled data**
- You can train on a nearby objective for which is easy to generate labels (imageNet).
- You can transfer learned representations from a relate task.

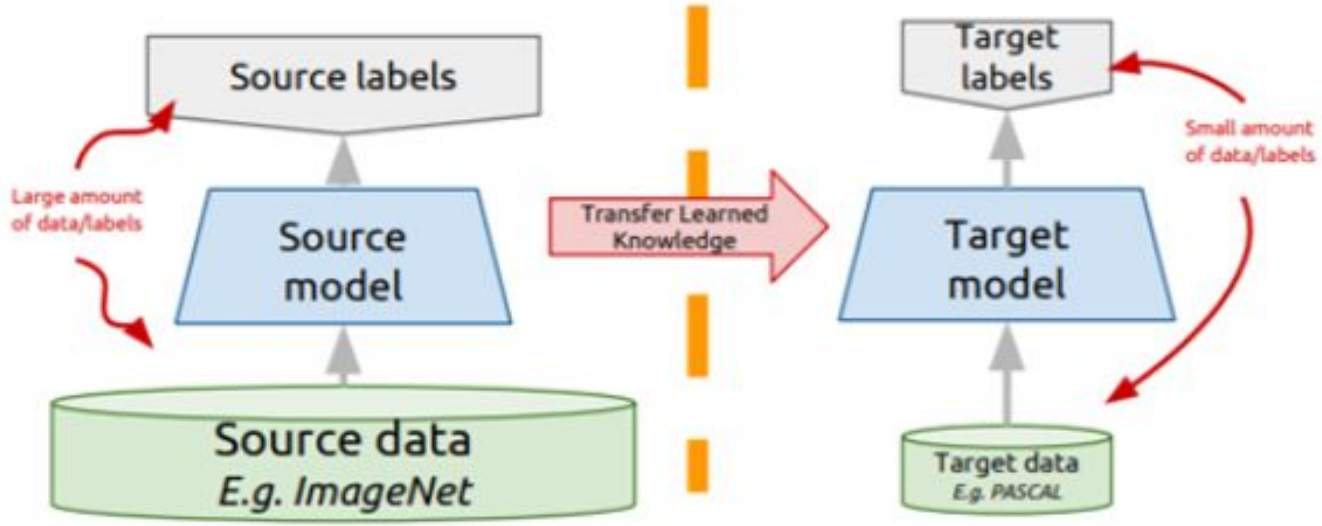
Transfer Learning for Deep Learning

Instead of training a network from scratch:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations

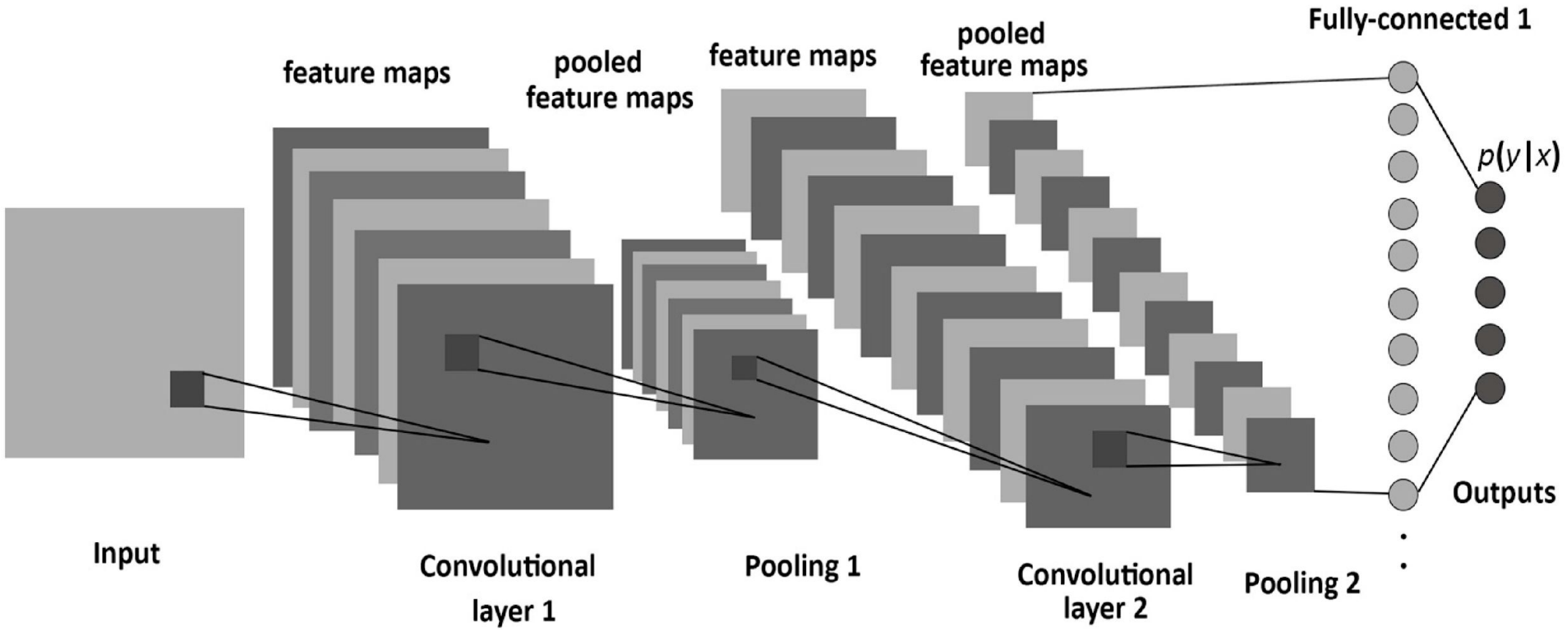
- Same domain, different task.
- Different domain, same task.



Outline

1. Introduction to Transfer Learning
2. **Review CNNs**
3. SOTA Deep Models
4. Transfer Learning across Tasks
5. Tutorial: Segmentation
6. Model Compression Techniques
7. Tutorial: Model Compression
8. Tutorial: Mushroom App Models

Review of CNNs



Review of CNNs

- Let C be a CNN with the following disposition:
  - Input: 32x32x3 images
  - Conv1: 8 3x3 filters, stride 1, padding=same
  - Conv2: 16 5x5 filters, stride 2, padding=same
  - Flatten layer
  - Dense1: 512 nodes
  - Dense2: 4 nodes
- How many parameters does this network have?

$$(8 \times 3 \times 3 \times 3 + 8) + (16 \times 5 \times 5 \times 8 + 16) + (16 \times 16 \times 16 \times 512 + 512) + (512 \times 4 + 4)$$

Conv1 Conv2 Dense1 Dense2

Review of CNNs

Input

(size=32X32,

channels=3)

Filter

8 x (size=3X3x3,

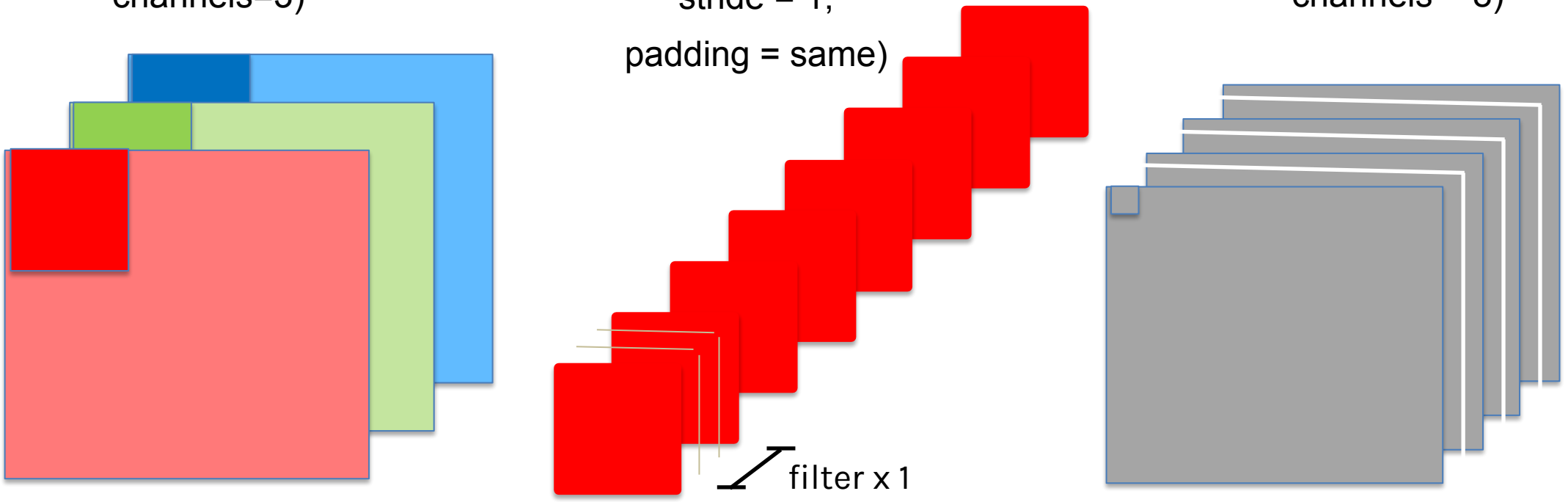
stride = 1,

padding = same)

Output

(size=32X32,

channels = 8)



How many parameters does the layer have if I want to use 8 filters?

n_filters x filter_volume + biases = total number of params

8 x (3 x 3 x 3) + 8 = 224

Review of CNNs

Input

(size=32X32,

channels=8)

Filter

16 x (size=5X5X8,

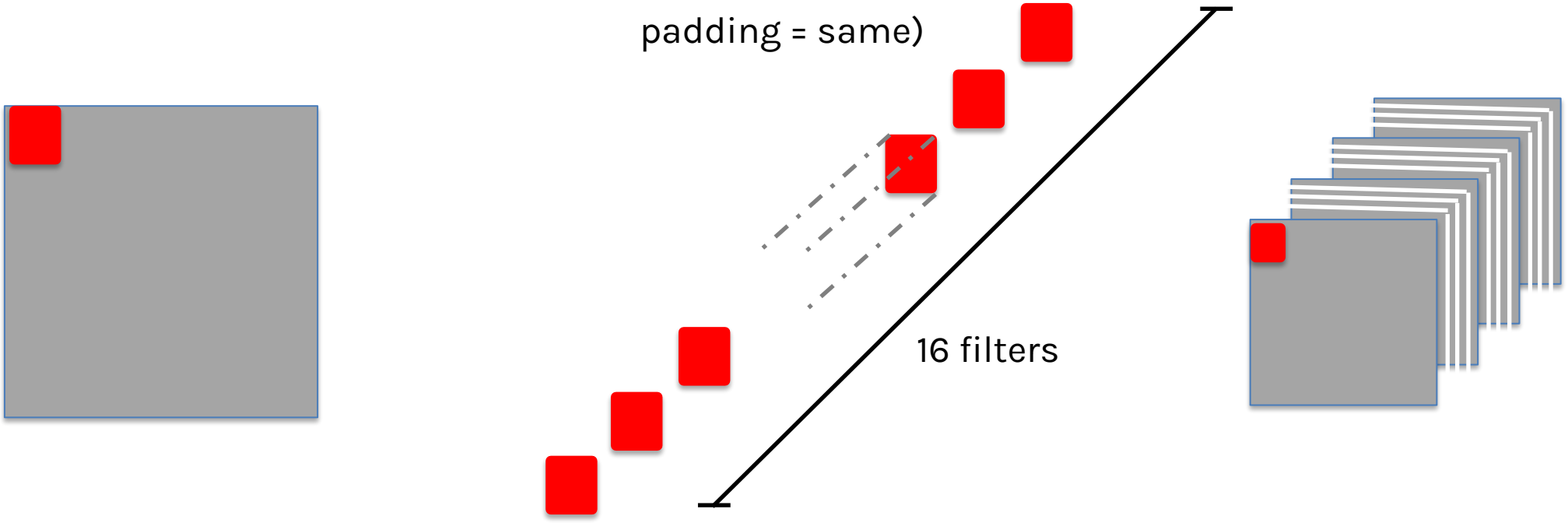
stride = 2,

padding = same)

Output

(size=16X16,

channels=16)



How many parameters does the layer have if I want to use 16 filters?

n_filters x filter_volume + biases = total number of params

16 x (5 x 5 x 8) + 16 = 3216

Review of CNNs

Input

(size=16X16, channels=16)

Flatten

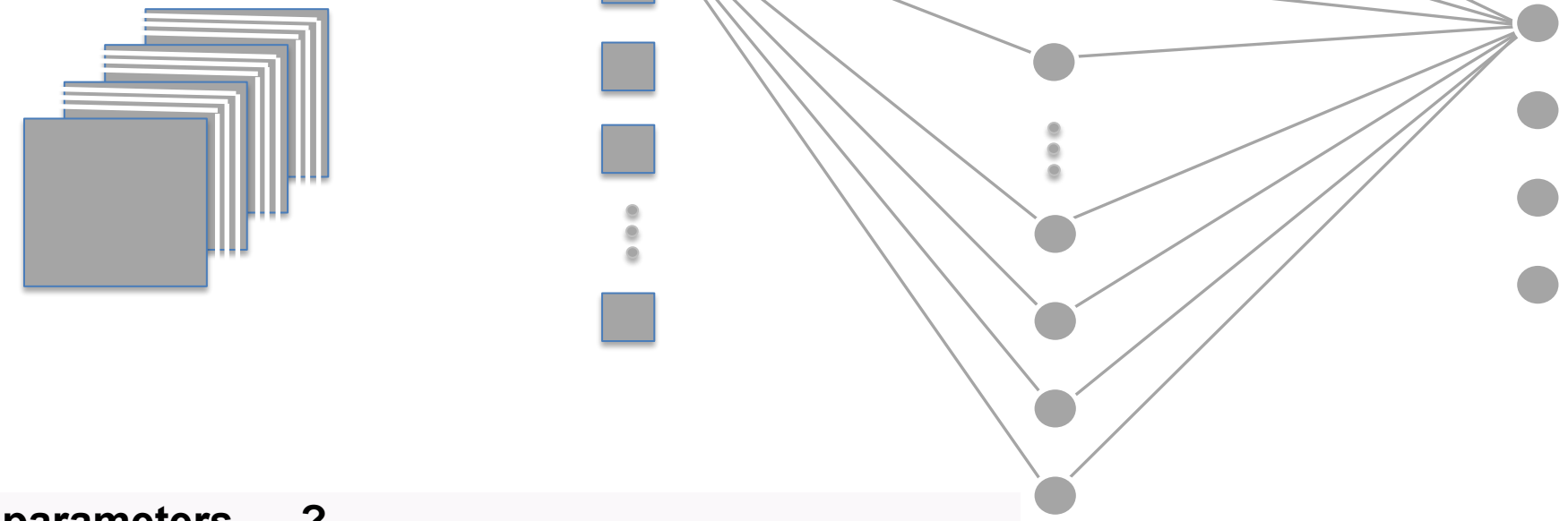
(size= 16X16X16)

Fully Connected

(n_nodes=512)

Fully Connected

(n_nodes=4)



How many parameters … ?

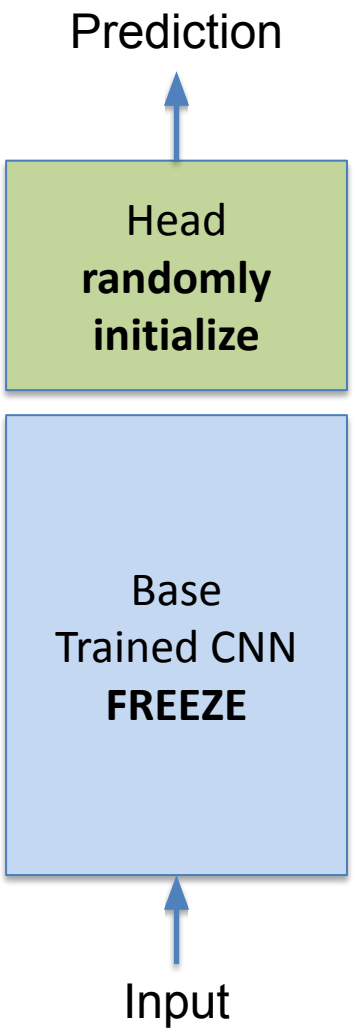
input x FC1_nodes + FC2_nodes = total number of params

(16x16x16) x 512 + 512 + 512 x 4 + 4 = 2,099,716

Representation Extraction

Use representations learned by big net to extract features from new samples, which are then fed to a new classifier:

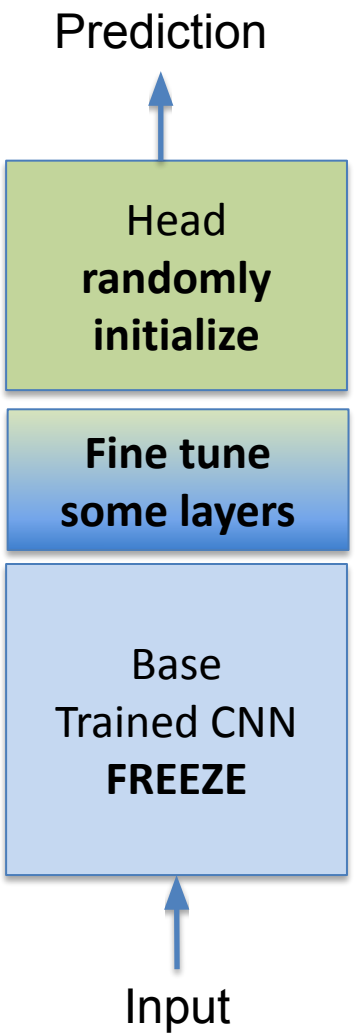
- Keep (frozen) convolutional **base** from big model.
- Generally throw away **head** FC layers since these have no notion of space, and convolutional base is more generic.
- Since there are both dogs and cats in *ImageNet* you could get **away** with using the head FC layers as well. But by throwing it away you can learn more from other dog/cat images.



$$P_s(Y|X) \neq P_t(Y|X)$$

Fine Tuning

- Up to now we have frozen the entire convolutional base.
- Remember that earlier layers learn highly generic feature maps (edges, colors, textures).
- Later layers learn abstract concepts (dog's ear).
- To particularize the model to our task, its often worth tuning the later layers as well.
- But we must be very careful not to have big gradient updates.



$$P_s(x) \neq P_t(X)$$

Procedure for Fine-tuning

1. Freeze the convolutional base.
2. First train the fully connected head you added, keeping the convolutional base fixed.
3. Unfreeze some "later" layers in the base net and now train the base net and FC net together.

Our gradients won’t be terribly high, as we are already in a better area of the loss surface. But, this means need to be careful and generally use a **very low learning rate**.

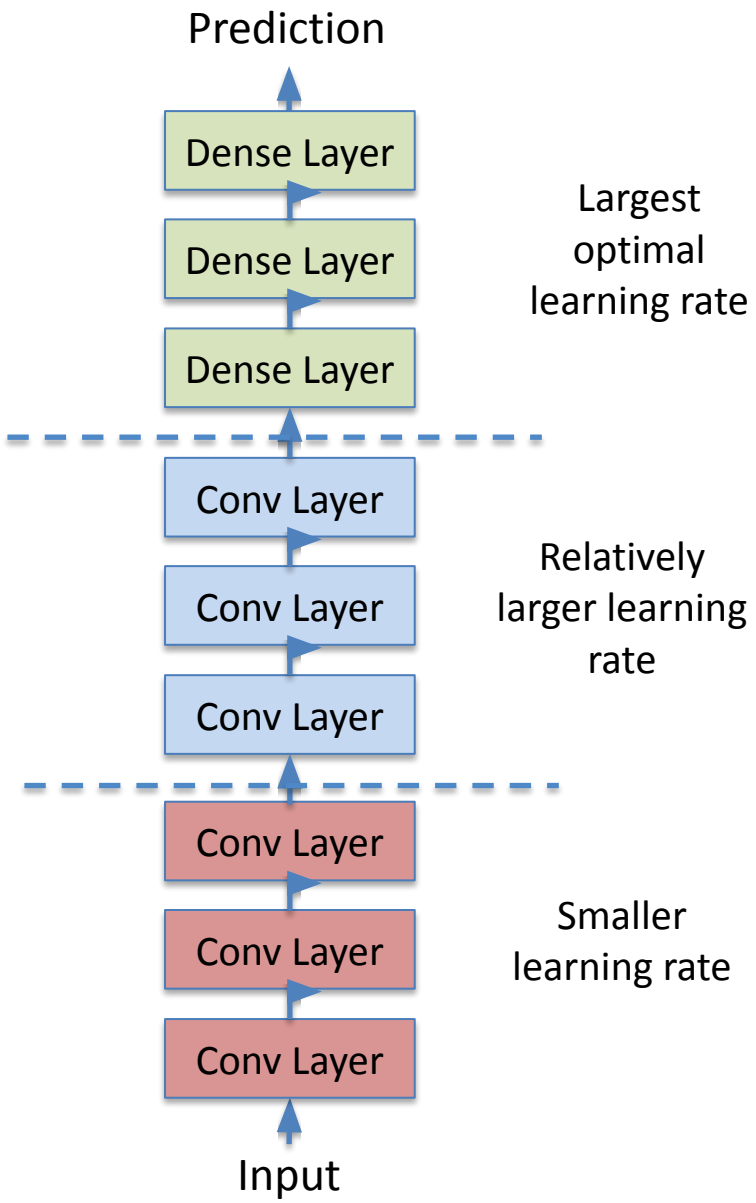
Transfer Learning for Deep Learning: Differential Learning Rates

• A low learning rate can take a lot of time to train on the "later" layers. Since we trained the FC head earlier, we could probably retrain them at a higher learning rate.

• General Idea: **Train different layers at different rates**.

• Each "earlier" layer or layer group (the color-coded layers in the image) can be trained at 3x-10x smaller learning rate than the next "later" one.

• One could even train the entire network again this way until we overfit and then step back some epochs.

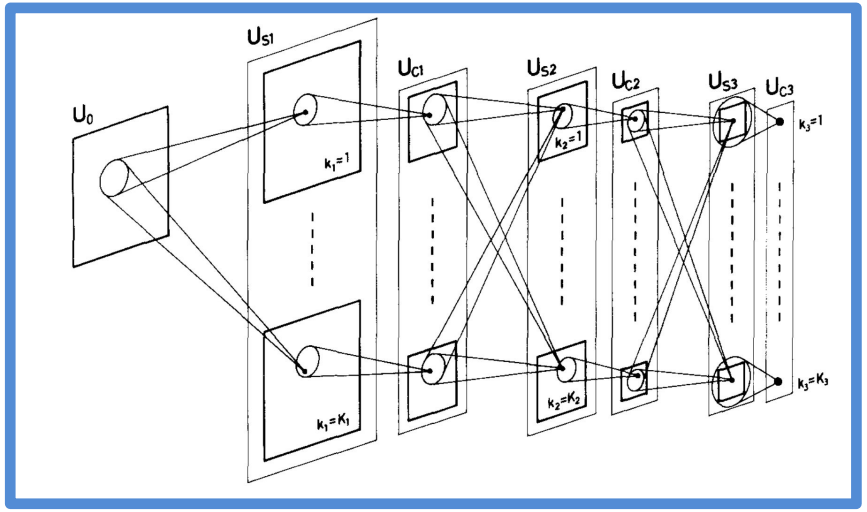


Outline

1. Introduction to Transfer Learning
2. Review CNNs
3. **SOTA Deep Models**
4. Transfer Learning across Tasks
5. Tutorial: Segmentation
6. Model Compression Techniques
7. Tutorial: Model Compression
8. Tutorial: Mushroom App Models

SOTA Deep Models: Initial Ideas

- The first research proposing something similar to a Convolutional Neural Network was authored by Kunihiko Fukushima in **1980** and was called the **NeoCognitron**.
- Inspired by discoveries on visual cortex of mammals.
- Fukushima applied the NeoCognitron to hand-written character recognition.

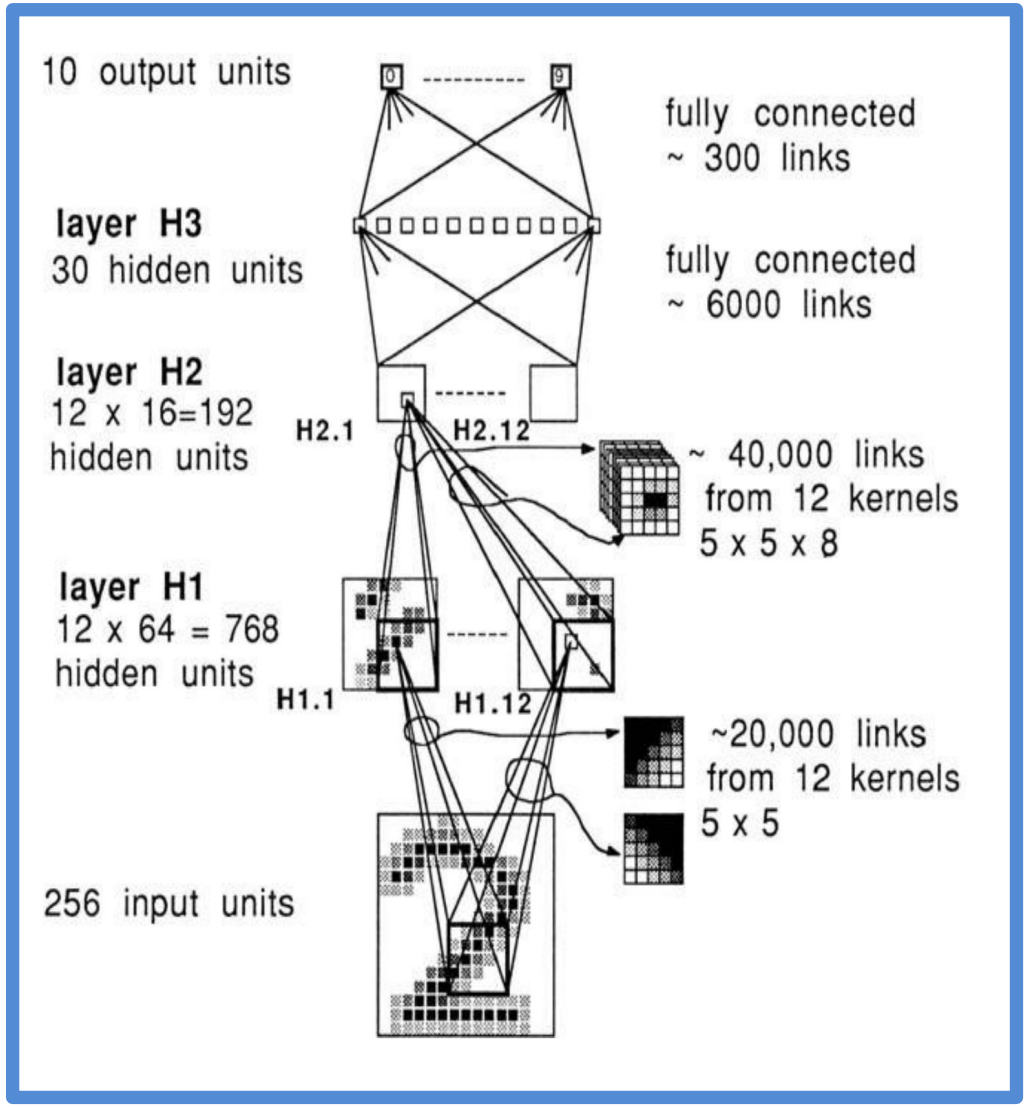


[1] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4): 93-202, 1980.

SOTA Deep Models: Initial Ideas

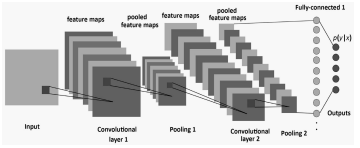
End of the 80's: several papers advanced the field

- **Backpropagation** published in French by Yann LeCun in 1985 (independently discovered by other researchers as well)
- TDNN by Waibel et al., 1989 - **Convolutional-like** network trained with backprop
- Backpropagation applied to handwritten zip code recognition by LeCun et al., 1989

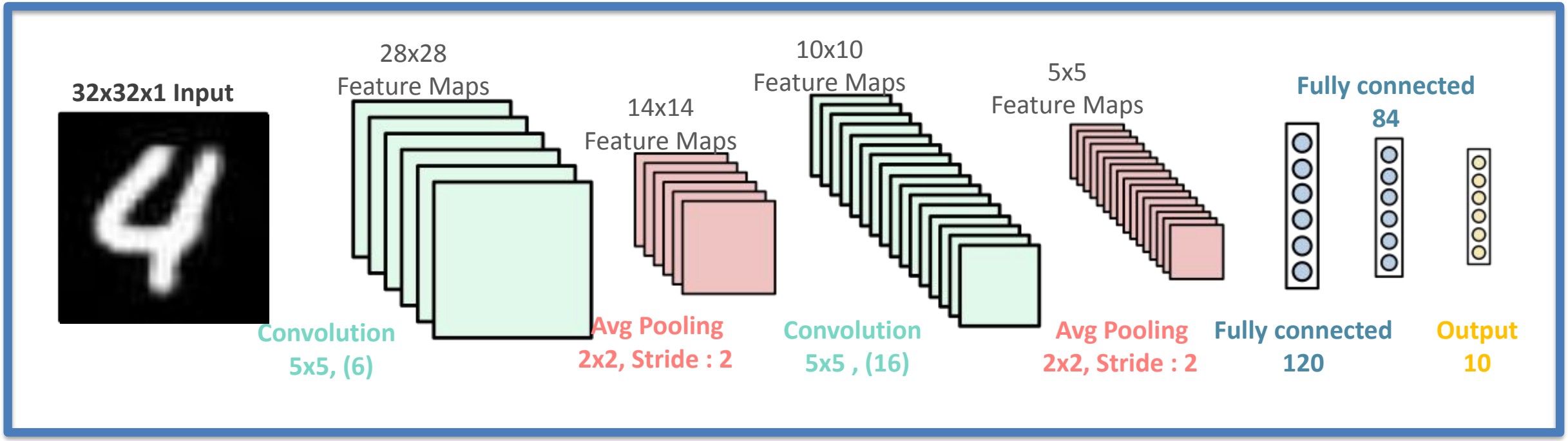


LeCun et al., 1989

LeNet



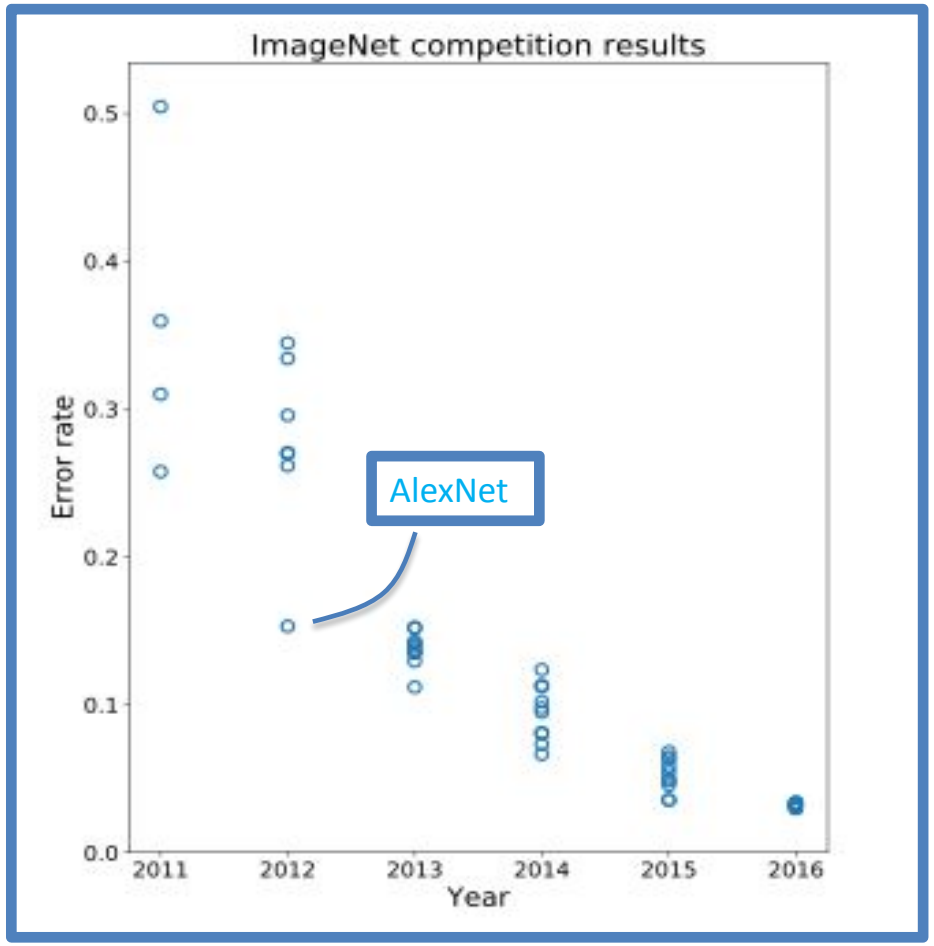
- November 1998: LeCun publishes one of his most recognized papers describing a “modern” CNN architecture for document recognition, called LeNet[1].
- Not his first iteration, this was in fact LeNet-5, but this paper is the commonly cited publication when talking about LeNet.



[1] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

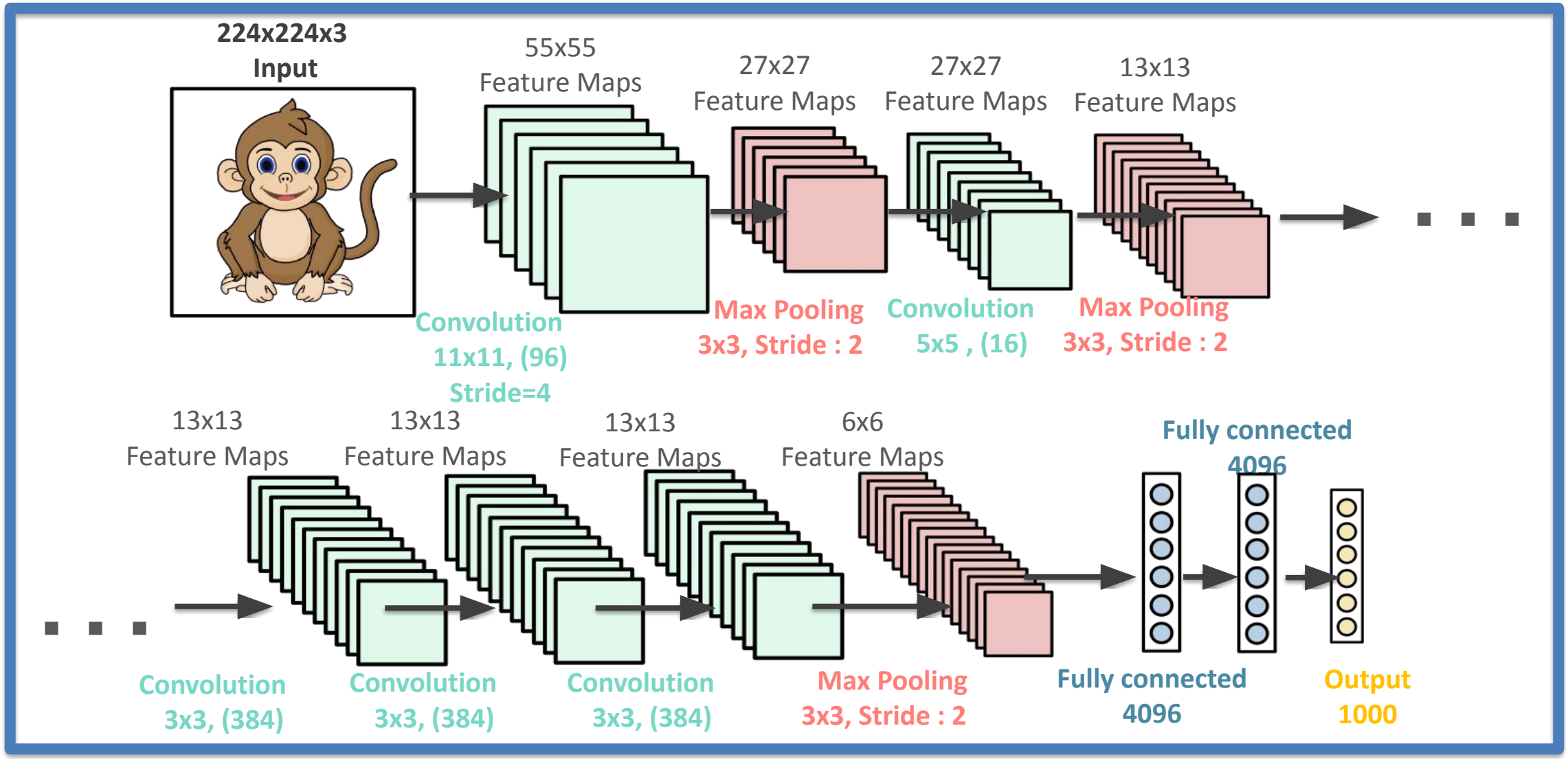
AlexNet

- Developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton at Utoronto in 2012. More than 25000 citations.
- Destroyed the competition in the 2012 ImageNet Large Scale Visual Recognition Challenge. Showed benefits of CNNs and kickstarted AI revolution.
- top-5 error of 15.3%, more than 10.8 percentage points lower than runner-up.



- Main contributions:
  - Trained on ImageNet with data augmentation.
  - Increased depth of model, GPU training (*six days*).
  - Smart optimizer and Dropout layers.
  - ReLU activation!

AlexNet


224x224x3
Input
55x55
Feature Maps
27x27
Feature Maps
27x27
Feature Maps
13x13
Feature Maps
Convolution
11x11, (96)
Stride=4
Max Pooling
3x3, Stride : 2
Convolution
5x5 , (16)
Max Pooling
3x3, Stride : 2
13x13
Feature Maps
13x13
Feature Maps
13x13
Feature Maps
6x6
Feature Maps
Fully connected
4096
Convolution
3x3, (384)
Convolution
3x3, (384)
Convolution
3x3, (384)
Max Pooling
3x3, Stride : 2
Fully connected
4096
Output
1000

AlexNet

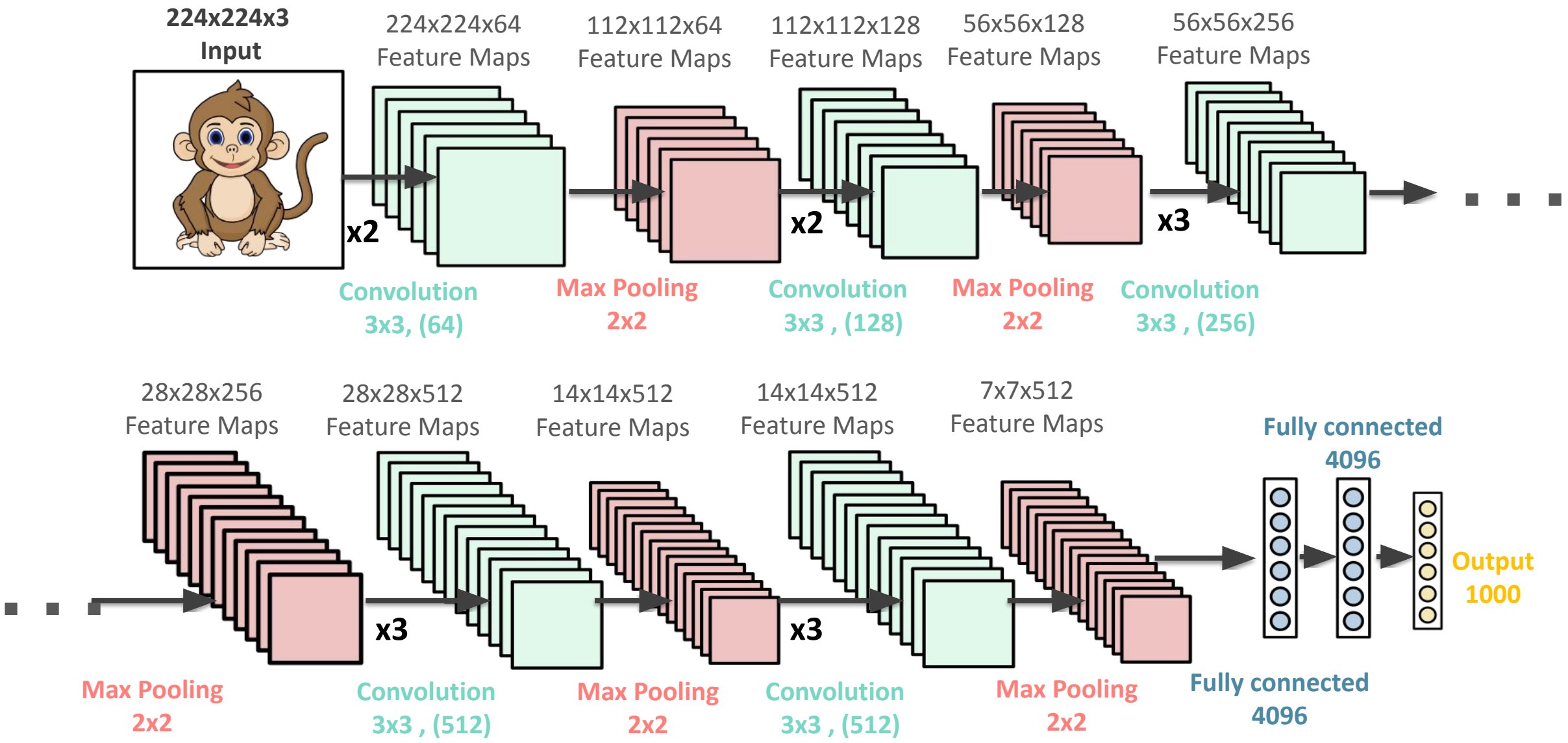
- 1.2 million high-resolution (227x227x3) images in the ImageNet 2010 contest
- 1000 different classes, NN with 60 million parameters to optimize (~ 255 MB)
- Uses ReLu activation functions; GPUs for training, 12 layers

VGG

- Introduced by Simonyan and Zisserman (Oxford) in 2014.
- Simplicity and depth as main points. Used **3x3 filters exclusively** and 2x2 MaxPool layers with stride 2.
- Showed that two 3x3 filters have an effective receptive field of 5x5.
- As spatial size decreases, depth increases.
- Convolutional layers use ‘same’ padding and stride s=1.
- Max-pooling layers use a window size 2 and stride s=2.

- ImageNet Challenge 2014; 16 or 19 layers; 138 million parameters.
- Trained for two to three weeks.
- Still used as of today.

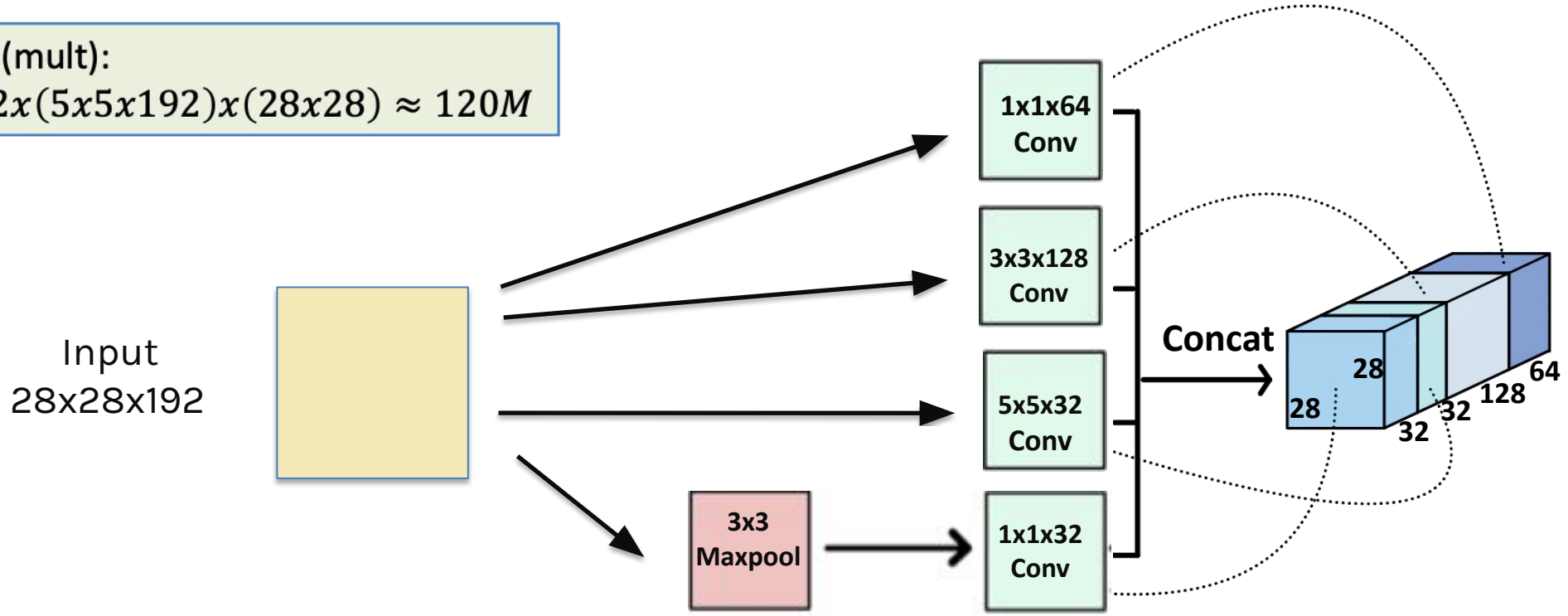
VGG


224x224x3
Input
224x224x64
Feature Maps
112x112x64
Feature Maps
112x112x128
Feature Maps
56x56x128
Feature Maps
56x56x256
Feature Maps
x2
x2
x3
Convolution
3x3, (64)
Max Pooling
2x2
Convolution
3x3 , (128)
Max Pooling
2x2
Convolution
3x3 , (256)
28x28x256
Feature Maps
28x28x512
Feature Maps
14x14x512
Feature Maps
14x14x512
Feature Maps
7x7x512
Feature Maps
Fully connected
4096
Output
1000
x3
x3
Max Pooling
2x2
Convolution
3x3 , (512)
Max Pooling
2x2
Convolution
3x3 , (512)
Max Pooling
2x2
Fully connected
4096

Inception (GoogLeNet)

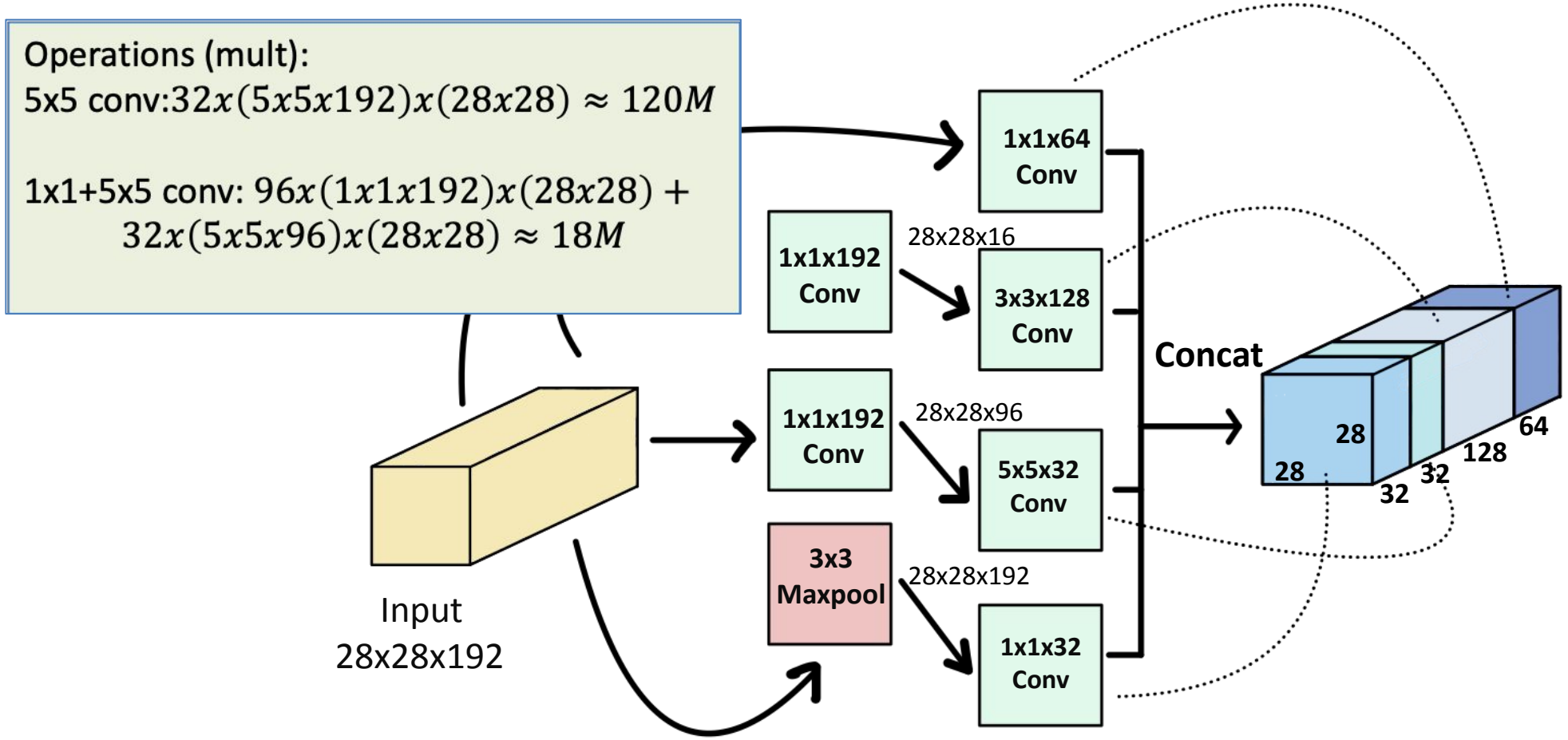
- The motivation behind inception networks is to use more than a single type of convolution layer at each layer.
- Use 1x1,3x3,5x5 convolutional layers, and max-pooling layers in parallel.
- All modules use same convolution.

Operations (mult):
5x5 conv: $32x(5x5x192)x(28x28) \approx 120M$



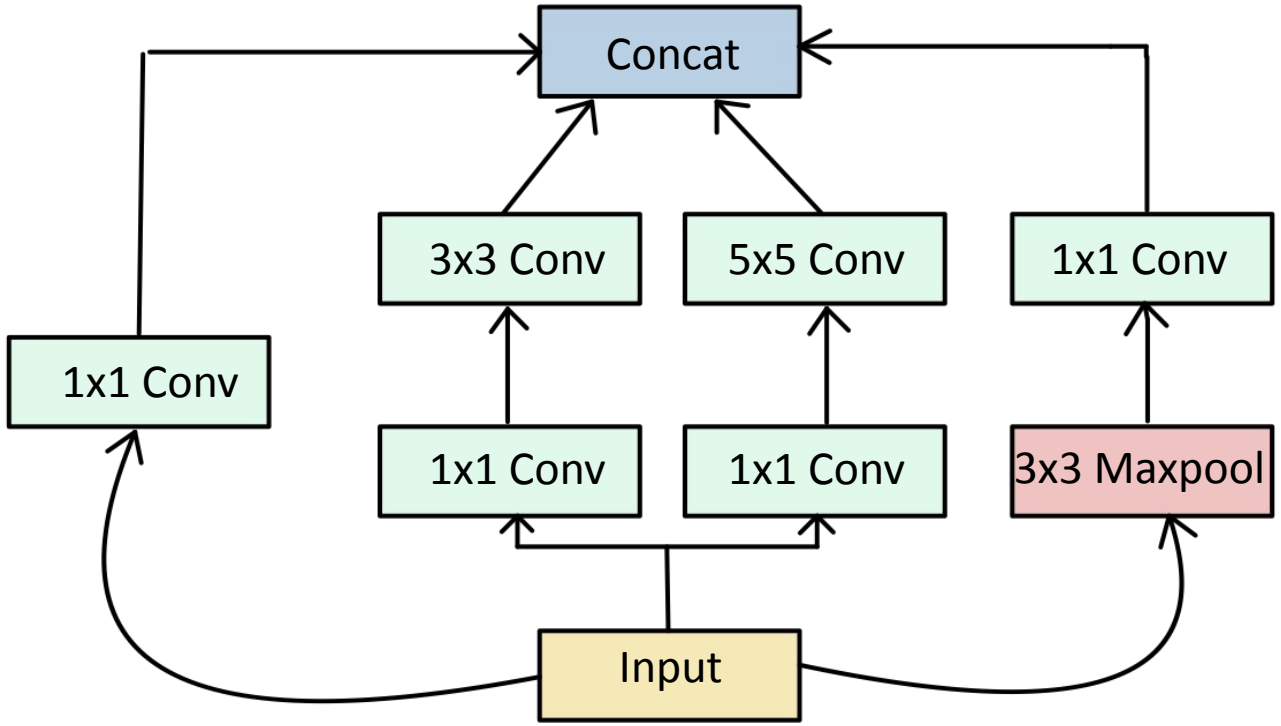
Inception (GoogLeNet)

- Use 1 x 1 convolutions that reduce the size of the channel dimension.
  - The number of channels can vary from the input to the output.

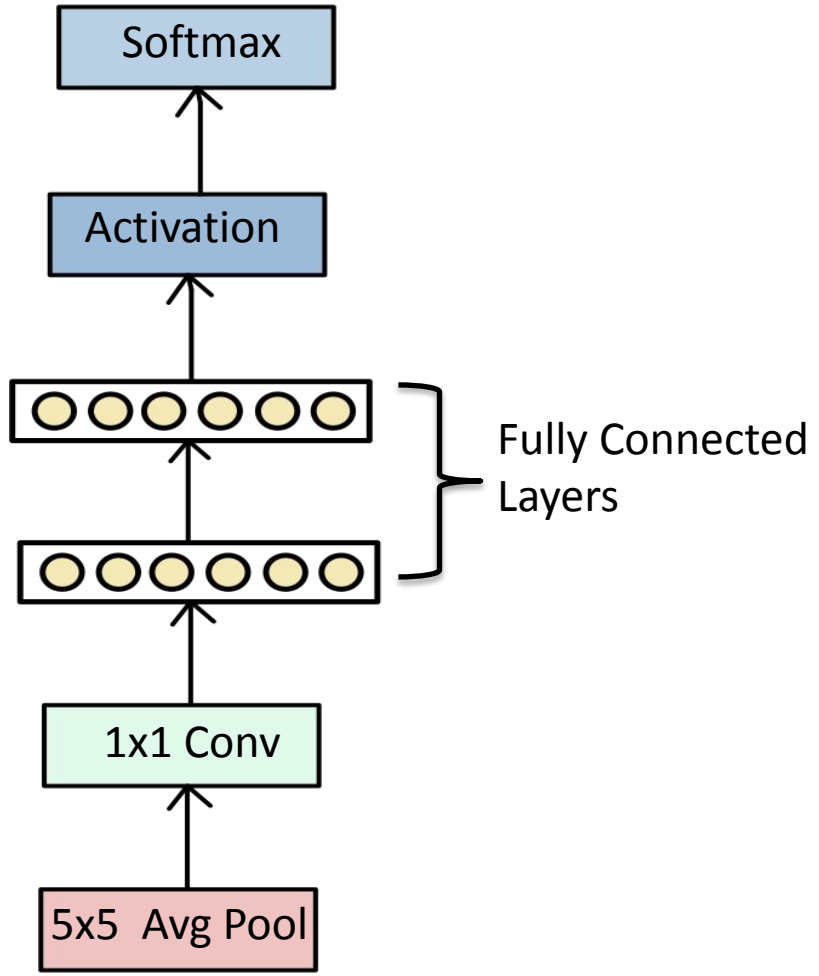


Inception (GoogLeNet)

The Inception Block

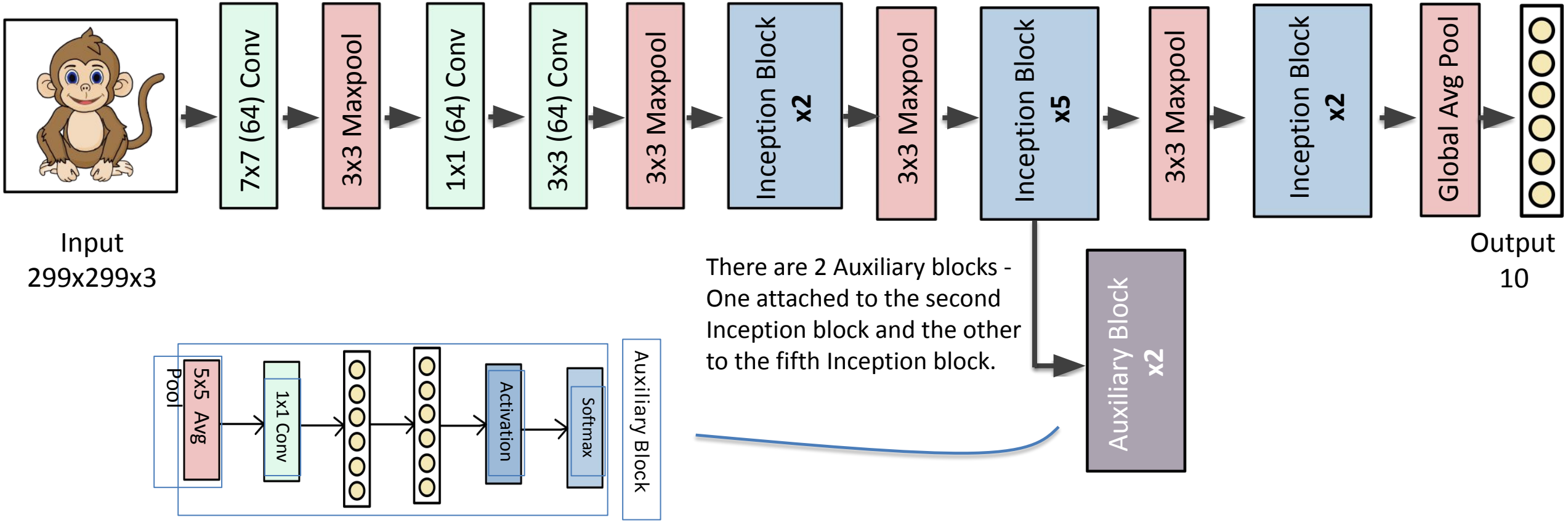


Auxiliary Block



Inception (GoogLeNet)

- The inception network is formed by concatenating other inception modules.
- It includes several softmax output units to enforce regularization.

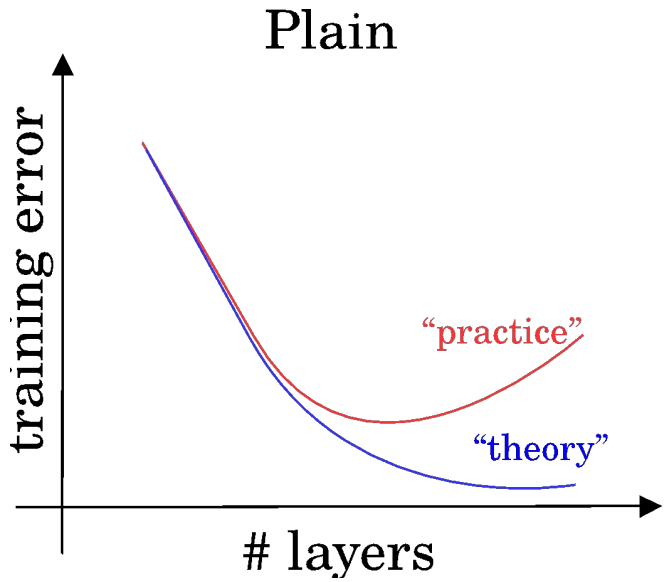


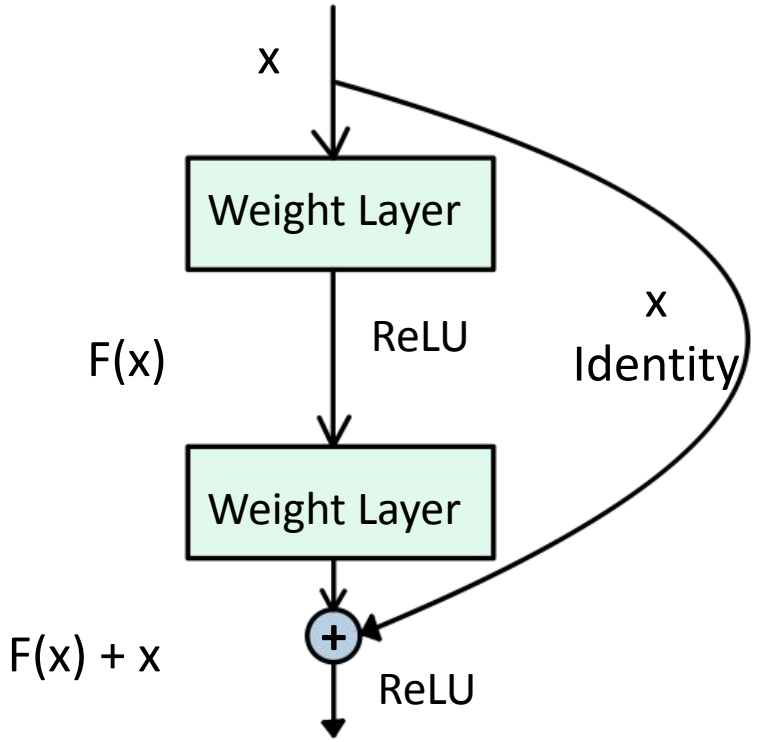
Are we done?

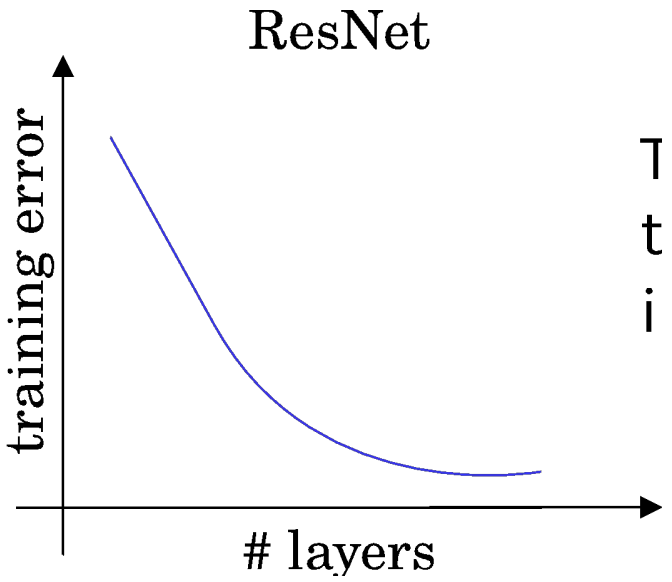
Are we done?

ResNet

- Presented by He et al. (Microsoft), 2015. Won ILSVRC 2015 in multiple categories. Very similar to Highway Networks Srivastava et al. 2015 introduced the same time.
- Main idea: Residual block. Allows for extremely deep networks.
- Authors believe that it is easier to optimize the residual mapping than the original one. Furthermore, residual block can decide to "shut itself down" if needed.

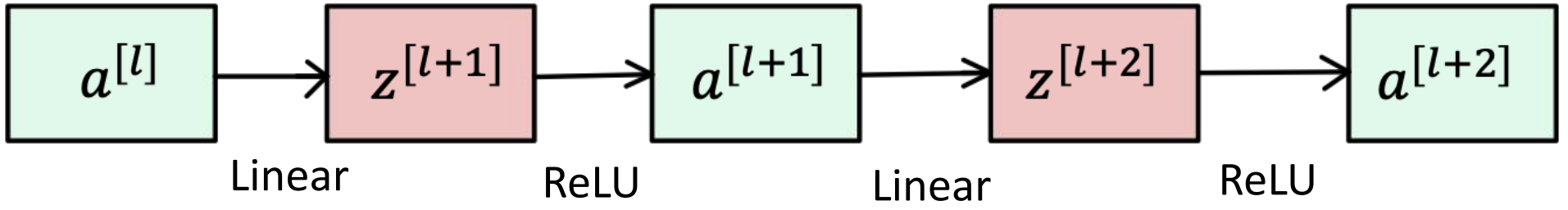




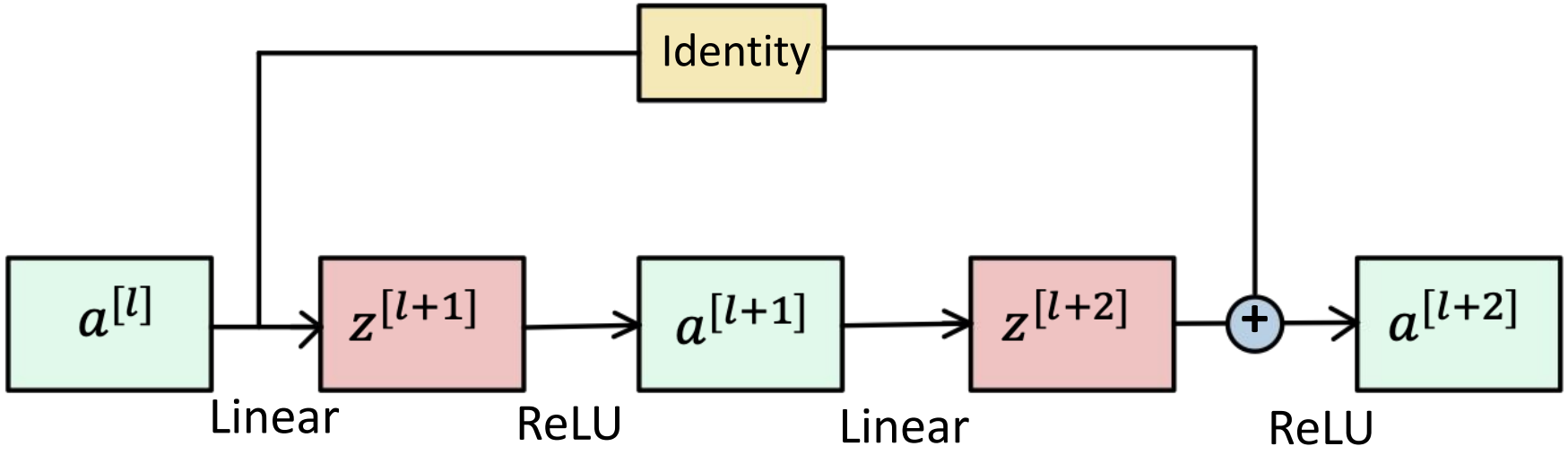


ResNet: Skip Connections

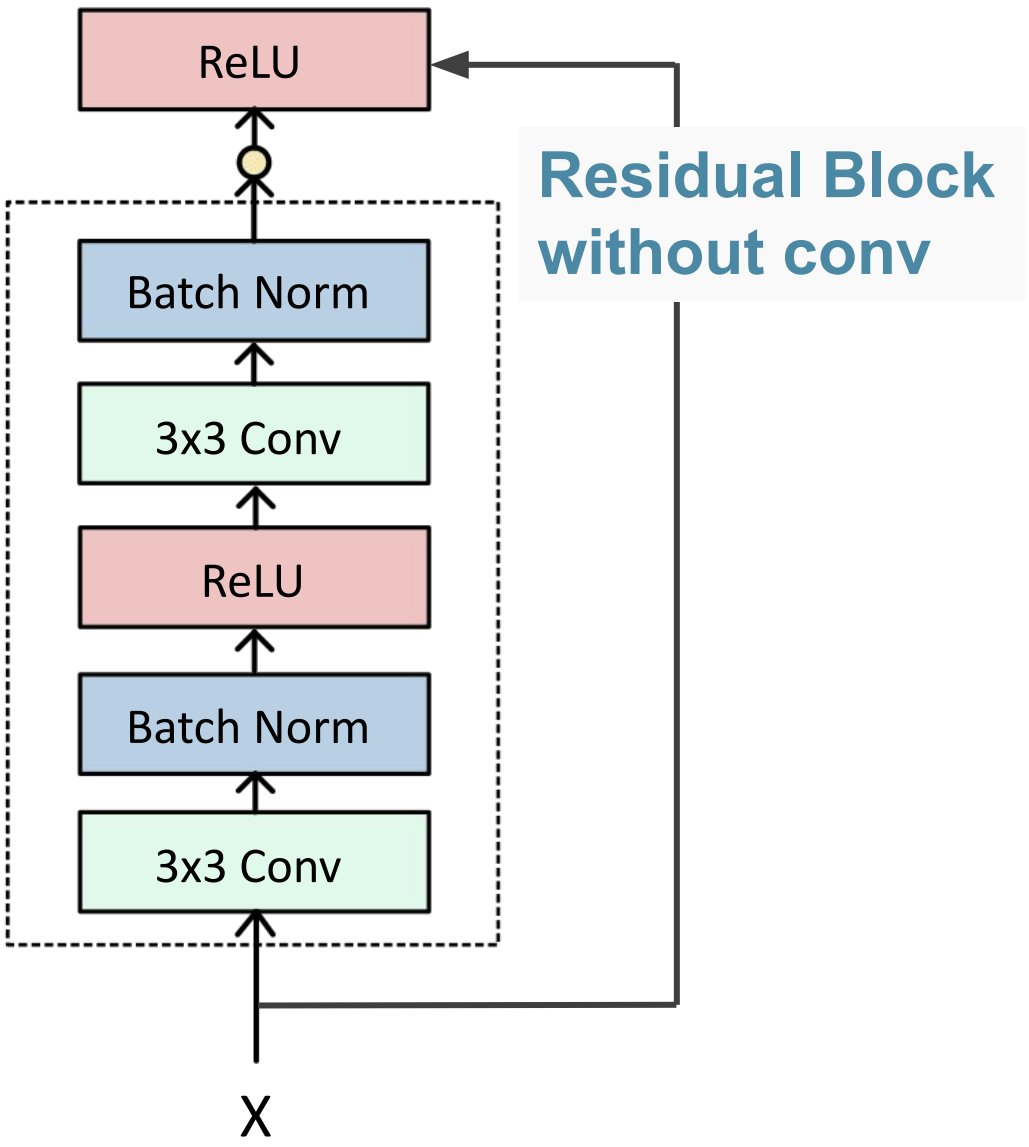
- Residual nets appeared in 2016 to train very deep NN (100 or more layers).
- Their architecture uses ‘residual blocks’.
- Plain network structure:



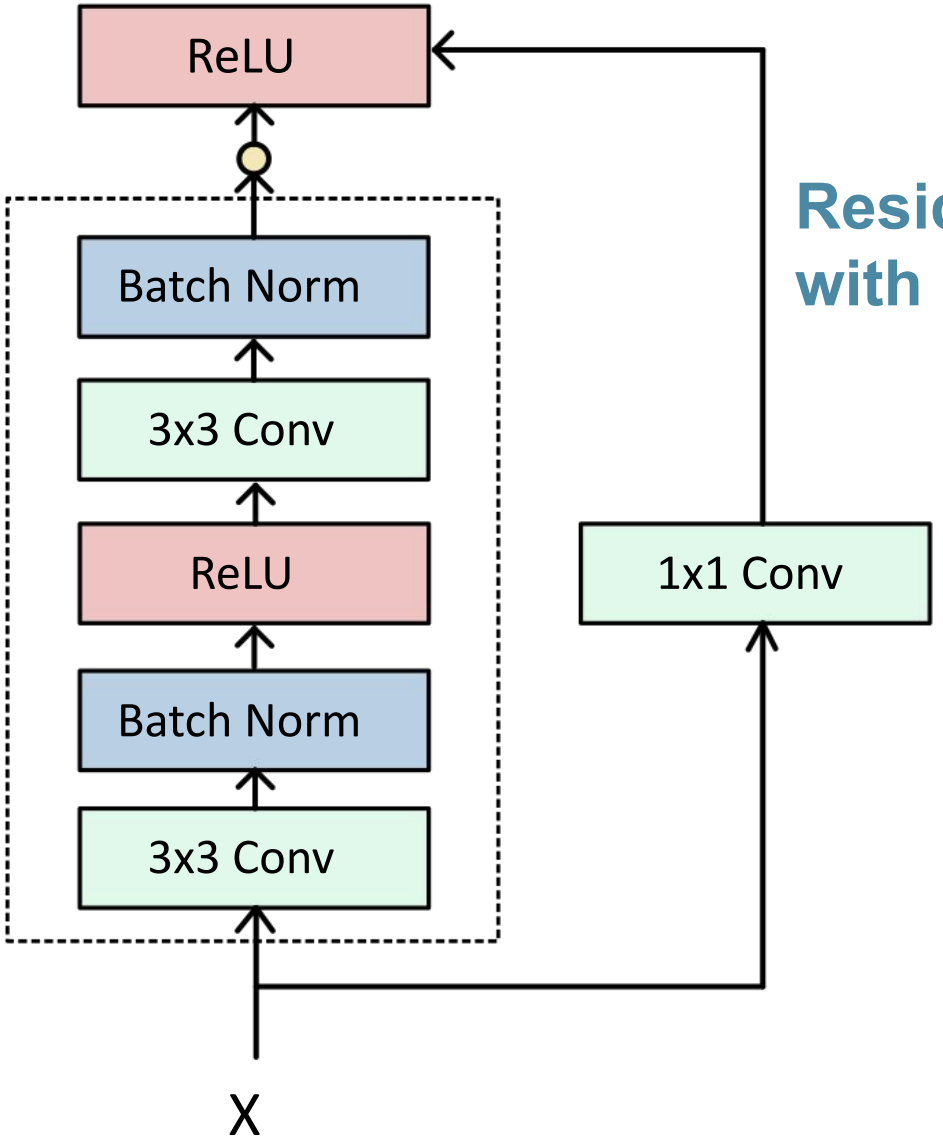
- **Residual network block**



ResNet: Basic Units



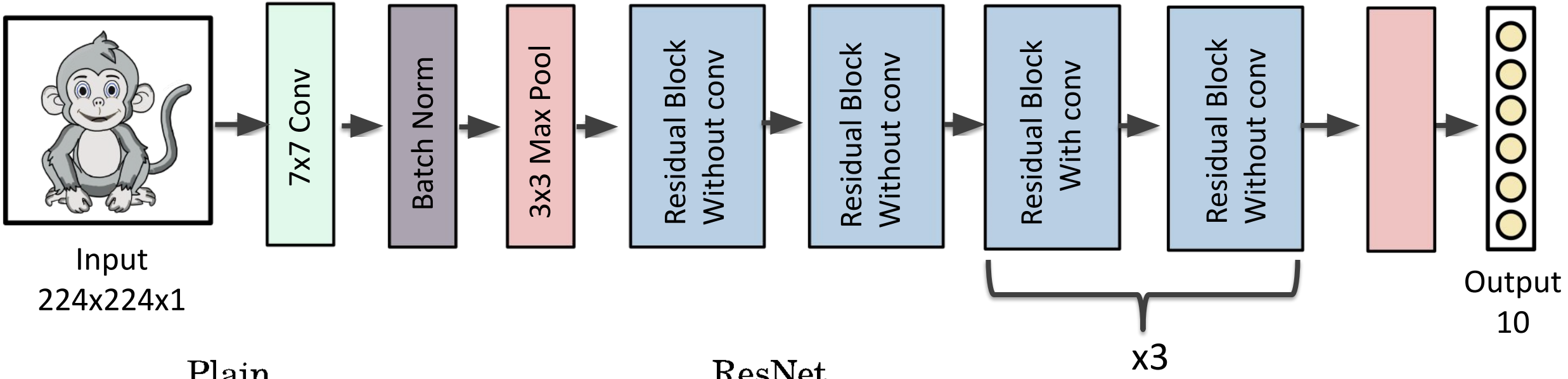
Residual Block without conv

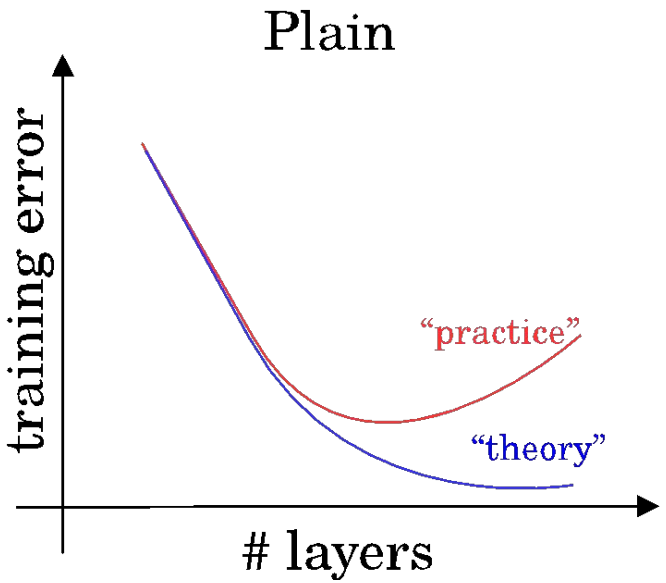


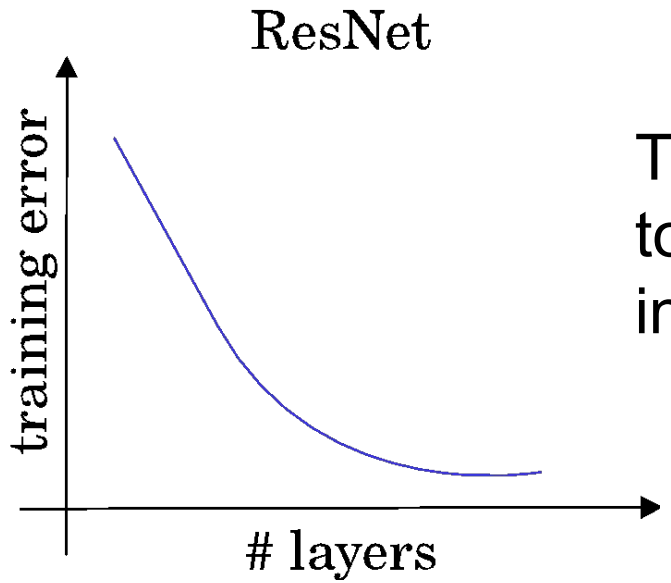
Residual Block with 1x1 conv

ResNet

The residual network stacks blocks sequentially





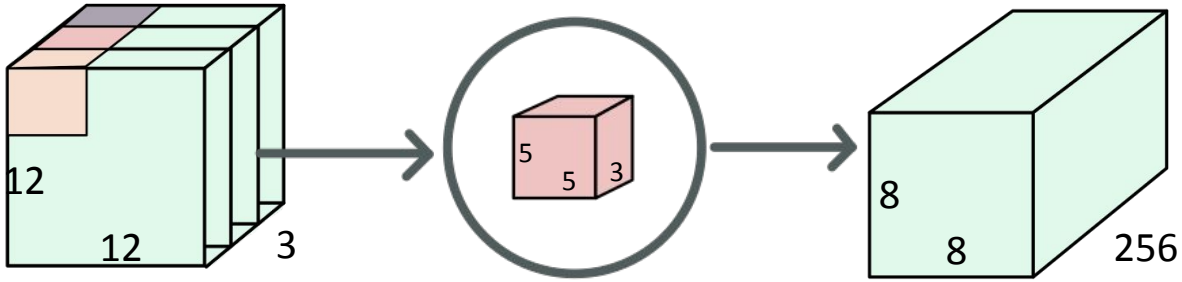


The idea is to allow the network to become deeper without increasing the training time

MobileNet

Standard Convolution

Filters and combines inputs into a new set of outputs in one step



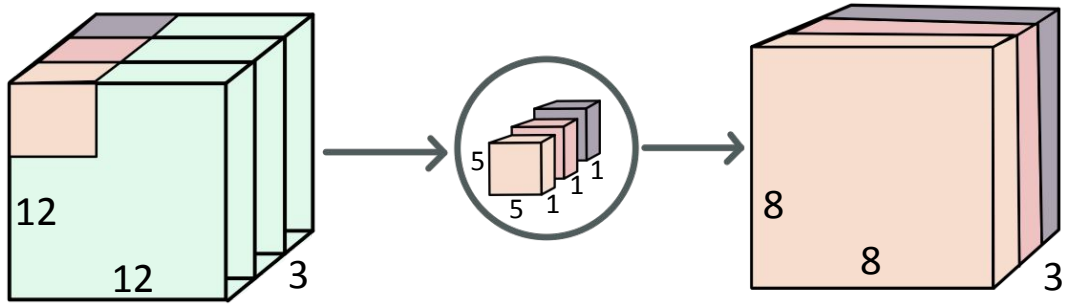
Input: 12x12x3
Filter: 5x5x3x256

Output: 8x8x256
(no padding)

MACs: (5x5)x3x256x(12x12) ~ **2.8M**
Parameters: (5x5x3)x256 + 256 ~ **20K**

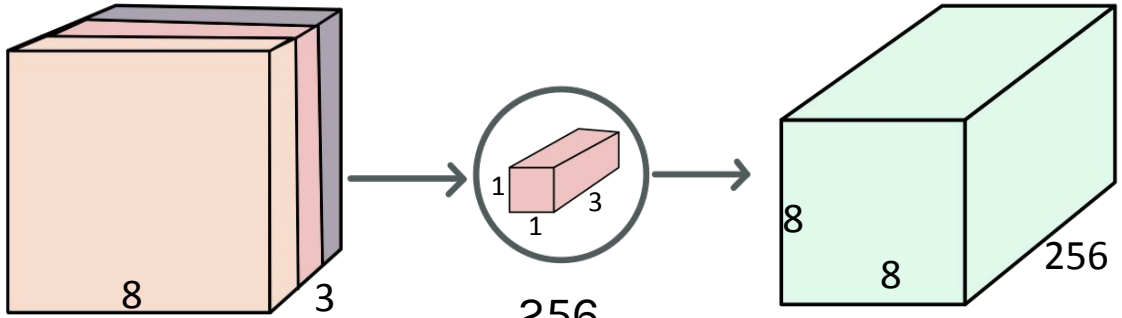
Depth-Wise Separable Convolution (DW)

It combines a depth wise convolution and a pointwise convolution



Input: 12x12x3
Filter: **3x(**5x5)

Output: 8x8x3
(no padding)



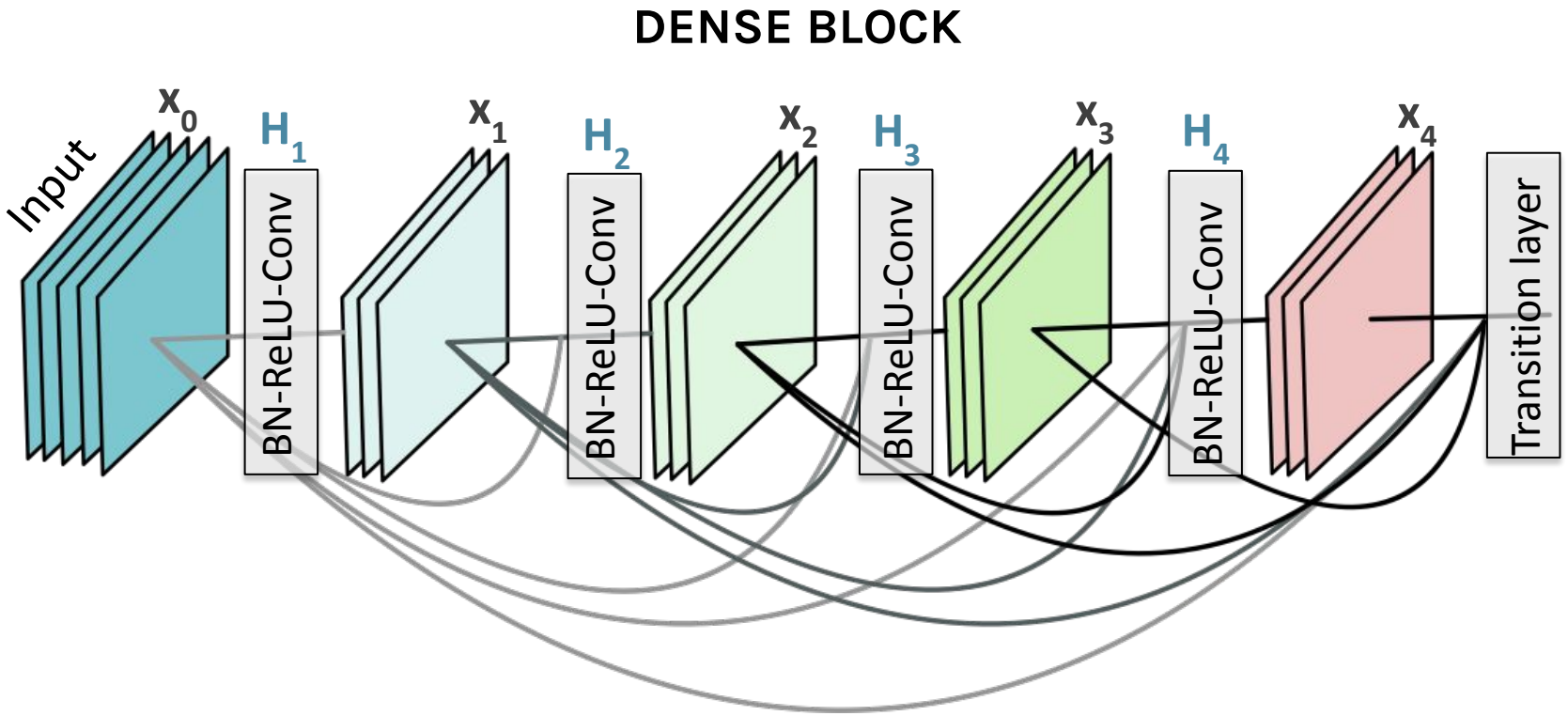
Input: **8**x8x3
Filter: **1**x1x3x256

Output: 8x8x256
(no padding)

MACs: (5x5)x3x(12x12) + 3x256x(8x8) ~ **60K**
Parameters: (5x5x3 + 3) + (1x1x3x256+256) ~ **1K**

DenseNet

- **Goal**: allow maximum information (and gradient) flow → connect every layer directly with each other.
- DenseNets exploit the potential of the network through feature reuse → no need to learn redundant feature maps.
- DenseNets layers are very narrow (e.g. 12 filters), and they just add a small set of new feature-maps.



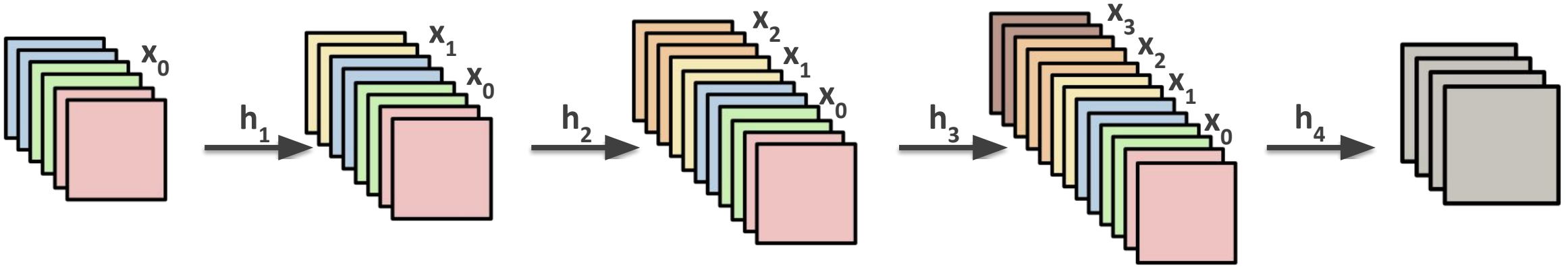
DenseNet

- DenseNets do not sum the output feature maps of the layer with the incoming feature maps but concatenate them:

$$a^{[l]} = g([a^{[0]}, a^{[1]}, \dots, a^{[l-1]}])$$

- Dimensions of the feature maps remains constant within a block, but the number of filters changes between them → **growth rate**:

$$k^{[l]} = k^{[0]} + k(l-1)$$

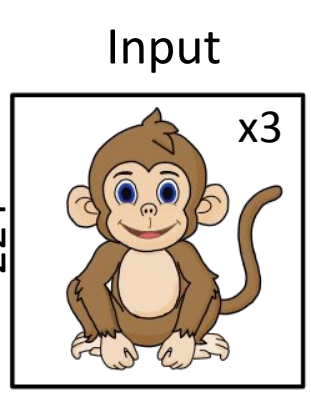


Concatenation during forward propagation

DenseNet

A dense layer contains:
1. Batch Normalization
2. ReLU activation
3. 3x3 Convolution

A transition layer is made of:
1. Batch Normalization
2. 1x1 Convolution
3. Average pooling



Input

64

Conv+Pooling

56x56

64+6*32

Dense Block 1

56x56

128

128 Conv 1x1

Av. Pooling

28x28

128+12*32

Dense Block 2

28x28

256

256 Conv 1x1

Av. Pooling

14x14

256+24*32

Dense Block 3

14x14

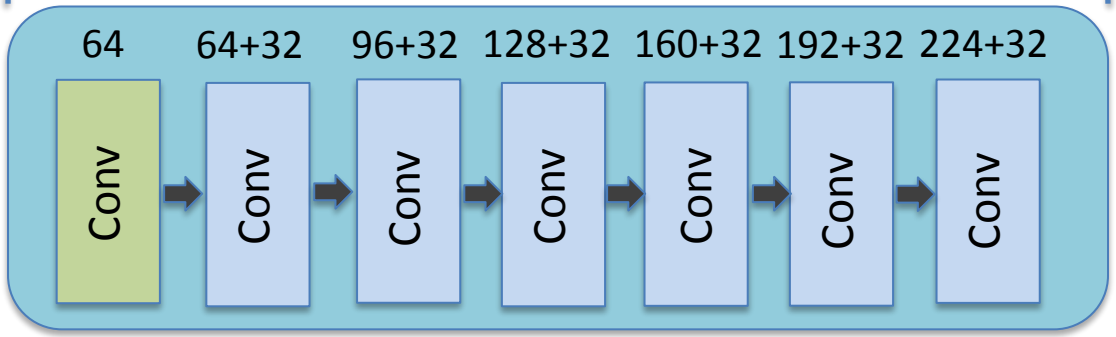
512

512 Conv 1x1

Av. Pooling

7x7

Prediction



Final Layers:
Dense Block 4
Av. Pooling
Softmax

Beyond

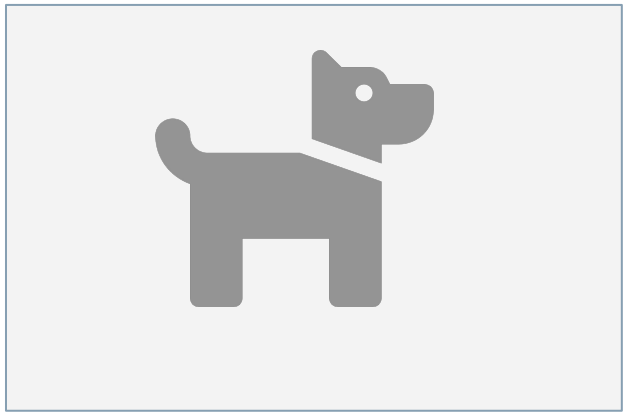
- MobileNetV2 (https://arxiv.org/abs/1801.04381)
- Inception-Resnet, v1 and v2 (https://arxiv.org/abs/1602.07261)
- Wide-Resnet (https://arxiv.org/abs/1605.07146)
- Xception (https://arxiv.org/abs/1610.02357)
- ResNeXt (https://arxiv.org/pdf/1611.05431)
- ShuffleNet, v1 and v2 (https://arxiv.org/abs/1707.01083)
- Squeeze and Excitation Nets (https://arxiv.org/abs/1709.01507)

Outline

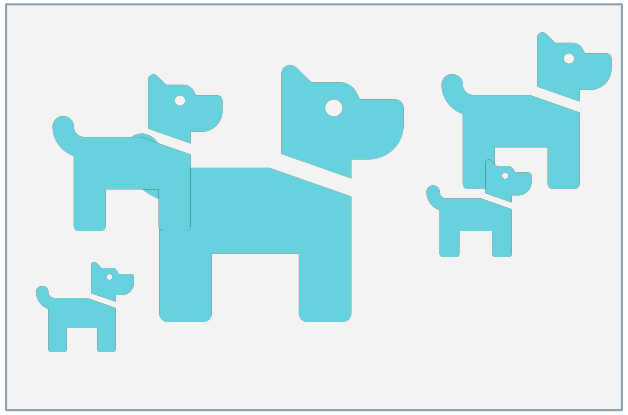
1. Introduction to Transfer Learning
2. Review CNNs
3. SOTA Deep Models
4. **Transfer Learning across Tasks**
5. Tutorial: Segmentation
6. Model Compression Techniques
7. Tutorial: Model Compression
8. Tutorial: Mushroom App Models

Computer Vision Tasks

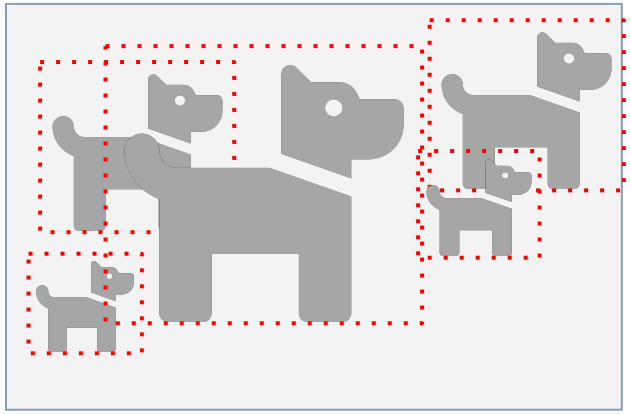
Classification



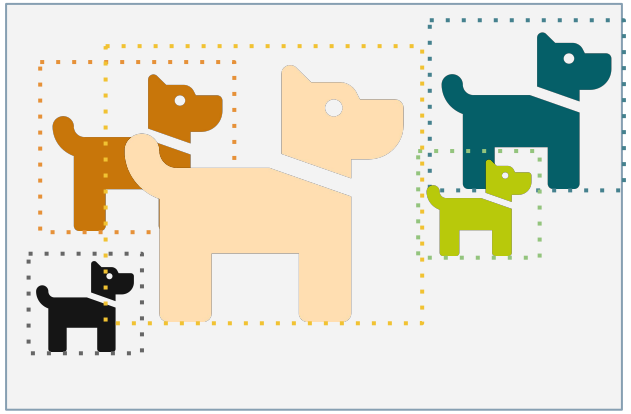
Semantic Segmentation



Object Detection



Instance Segmentation



Object Detection & Semantic Segmentation

Object Detection: let's classify and locate

- Sliding Window versus Region Proposals
- Two stage detectors: the evolution of R-CNN , Fast R-CNN, Faster R-CNN
- Single stage detectors: detection without Region Proposals: YOLO / SSD

Semantic Segmentation: classify every pixel

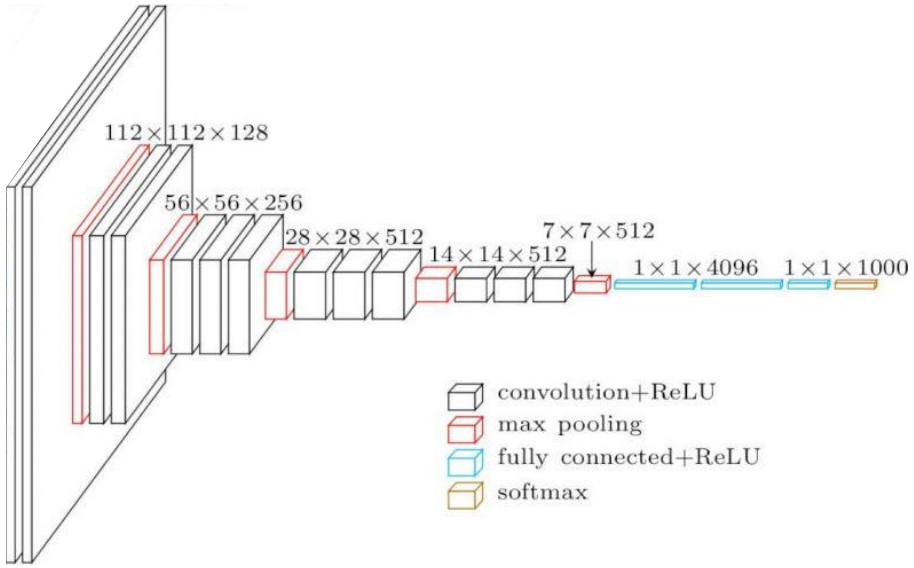
- Fully-Convolutional Networks
- SegNet & U-NET
- Faster R-CNN linked to Semantic Segmentation: Mask R-CNN

Task: Image Classification using Fully-Connected CNN

- Fundamental to computer vision given a set of labels {dog, cat, human, ...};
- Predict the most likely class.

Input

VGG



Output

Classification (C = 1000):

- Dog: 0.95
- Cat: 0.02
- Human: 0.01
- ...

Task: From Classification to Classification + Localization

- Localization demands to compute **where 1 object is present in an image;**
- Limitation: only 1 object (also non-overlapping);
- Typically implemented using a bounding box (x, y, w, h).

Predict

Classification Output:

- Dog: 0.95
- Cat: 0.02
- Human: 0.01
- ...

Predict

Classification output:

- Dog: 0.95
- Cat: 0.02
- Human: 0.01

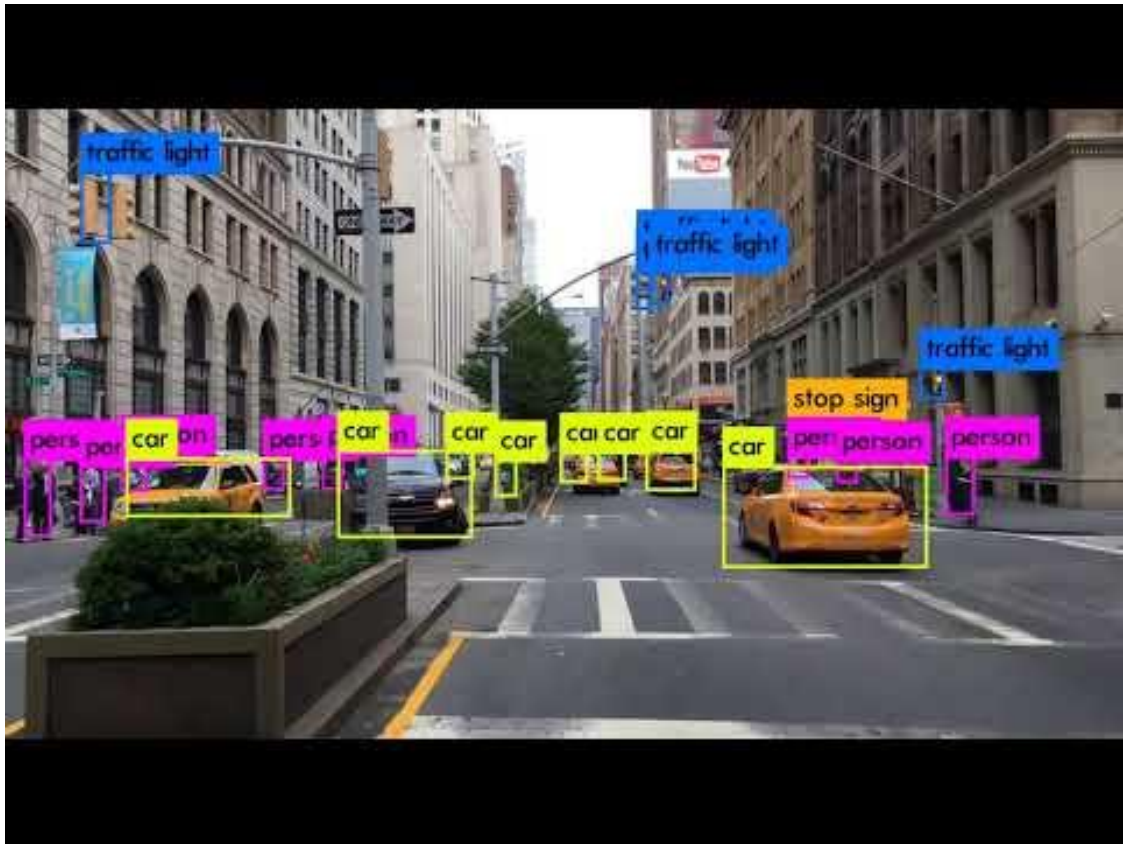
Localization output:

- Bounding-Box: (x, y, w, h)

Output: Regular Image Classification

Task: From Classification + Localization to Object Detection

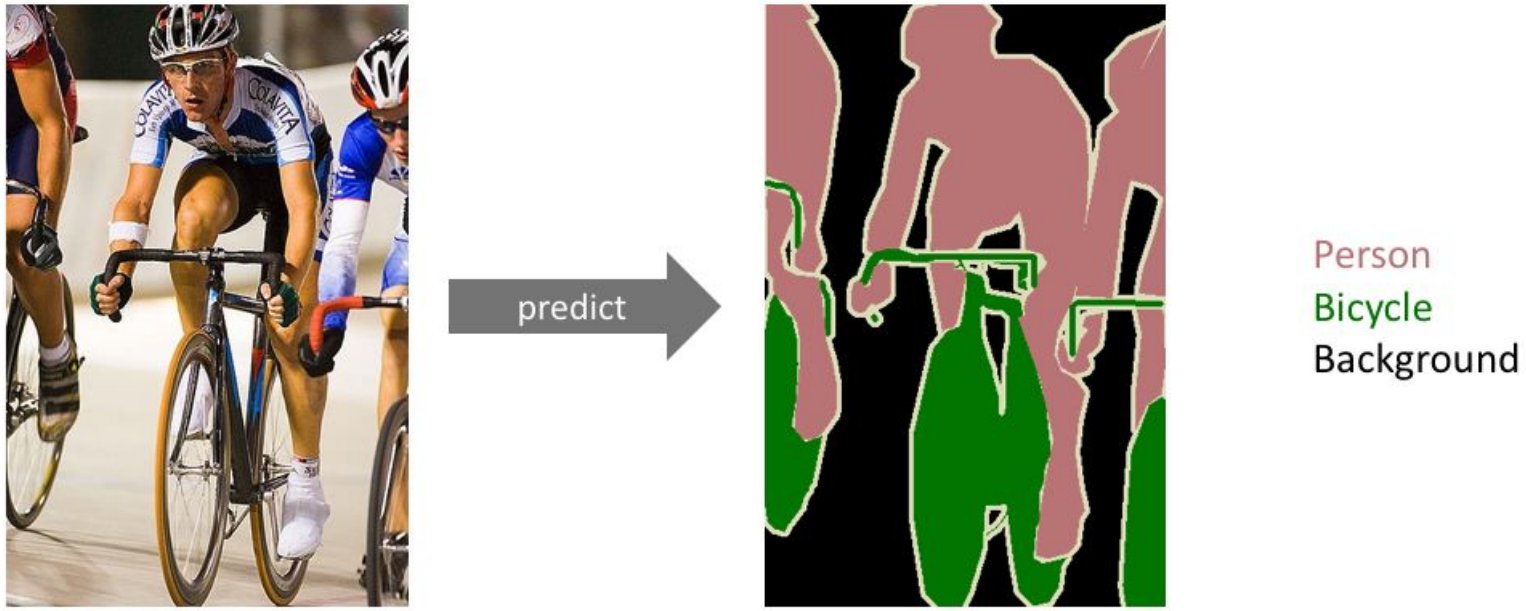
- Classification and Localization extended to multiple objects



Youtube ‘YOLO in New York” by Joseph Redmon (creator of YOLO)

Task: From Classification to Semantic Segmentation

- **Image Classification:** assigning a single label to **the entire picture**
- **Semantic Segmentation:** assigning a semantically meaningful label to **every pixel in the image**



Long, Shelhamer et al. “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015 : Cited by 14480

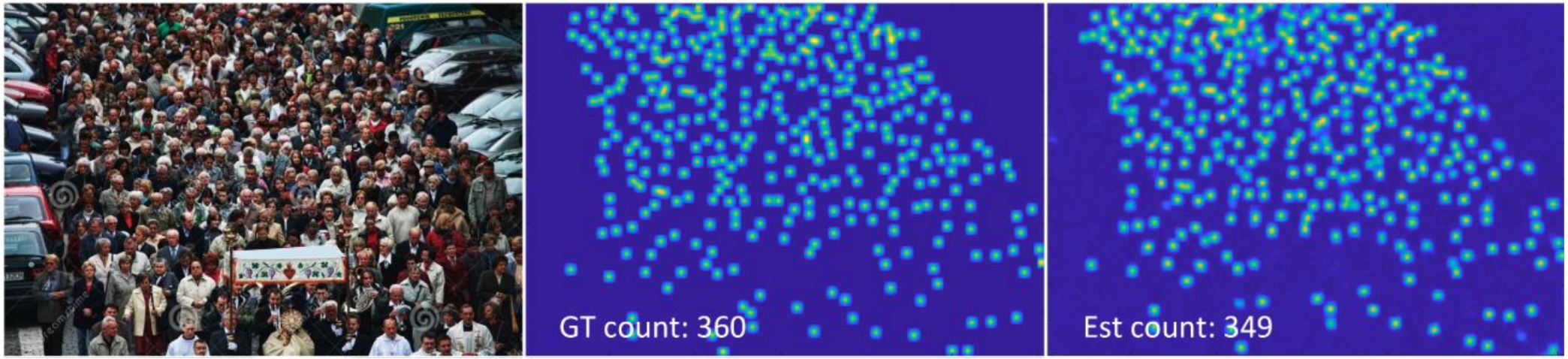
Why Object Detection and Semantic Segmentation

Computer Vision:

- Autonomous vehicles
- Biomedical Imaging detecting cancer, diseases
- Video surveillance:
  - Counting people
  - Tracking people
- Aerial surveillance
- Geo Sensing: tracking wildfire, glaciers, via satellite

Note:

- Efficiency/inference-time is important!
- How many frames/sec. can we predict?
- Must for real-time segmentation & detection.



Why Object Detection and Semantic Segmentation

Youtube: “Tensorflow DeepLab v3 Xception Cityscapes”(link)

How to Measure Quality in Detection and Segmentation?

- **Pixel Accuracy:**
  - Percent of pixels in your image that are classified correctly
  - Our model has 95% accuracy! Great!

Input | **Labels** | **Predict**

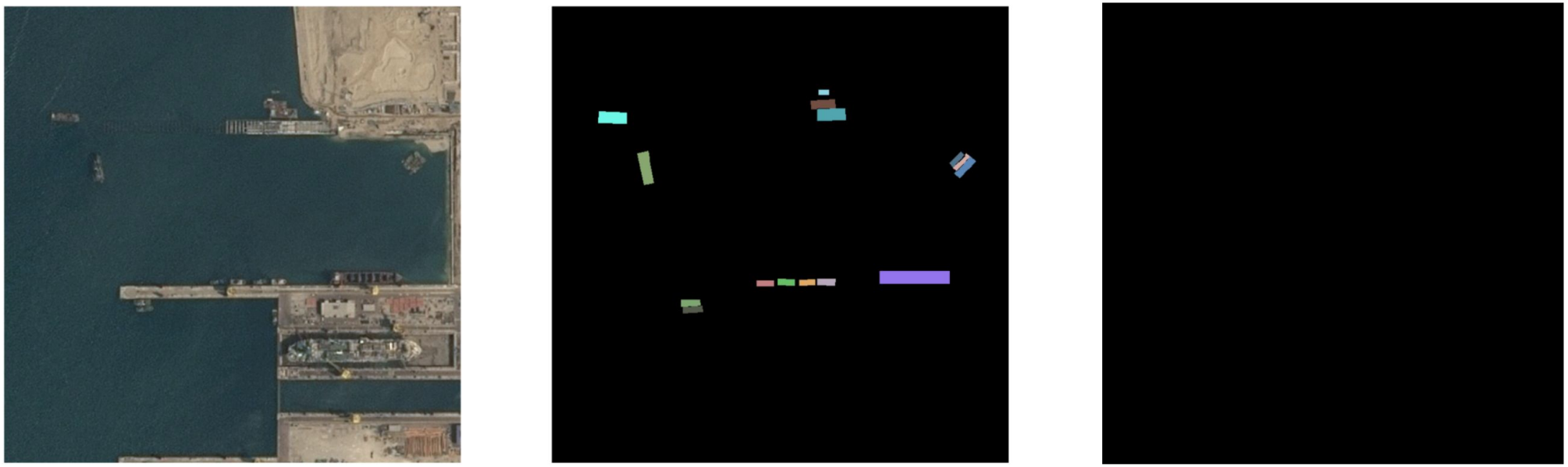
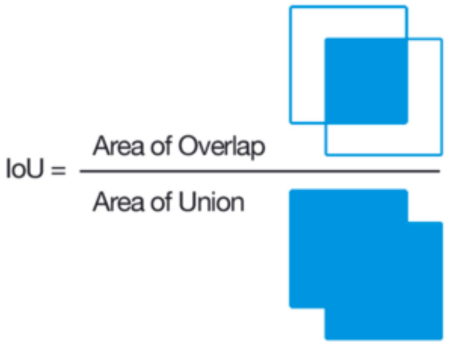


Image from Vlad Shmyhlo in article: Image Segmentation: Kaggle experience in TDS

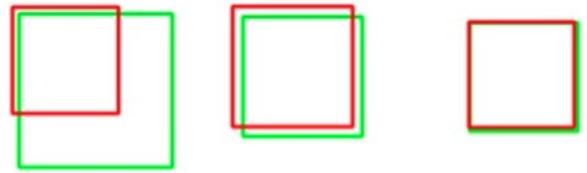
- Problem with accuracy: unbalanced data!

How Do We Measure Accuracy?

- **Pixel Accuracy**: Percent of pixels in your image that are classified correctly
- **IOU:** Intersection-Over-Union (Jaccard Index): Overlap / Union
- **mAP:** Mean Average Precision: AUC of Precision-Recall curve standard (0.5 is high)
- **DICE:** Coefficient (F1 Score): 2 x Overlap / Total number of pixels

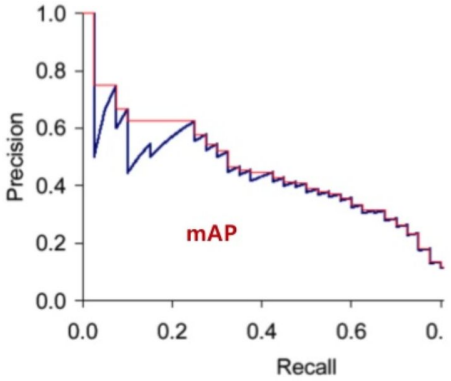


IOU

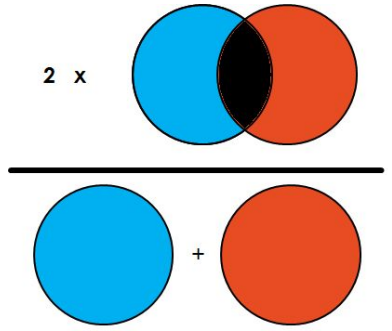


Poor **IoU:** 0.40 **Good** **IoU:** 0.73 **Excellent** **IoU:** 0.92

mAP



DICE



Object Detection

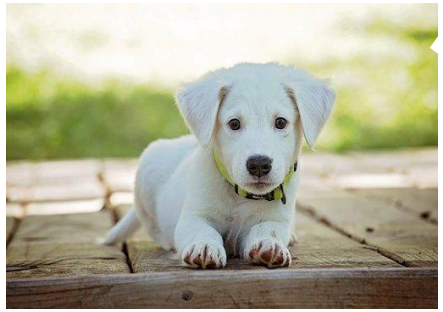
Object Detection: let's classify and locate

- Sliding Window versus Region Proposals
- Two stage detectors: the evolution of R-CNN , Fast R-CNN, Faster R-CNN
- Single stage detectors: detection without Region Proposals: YOLO / SSD

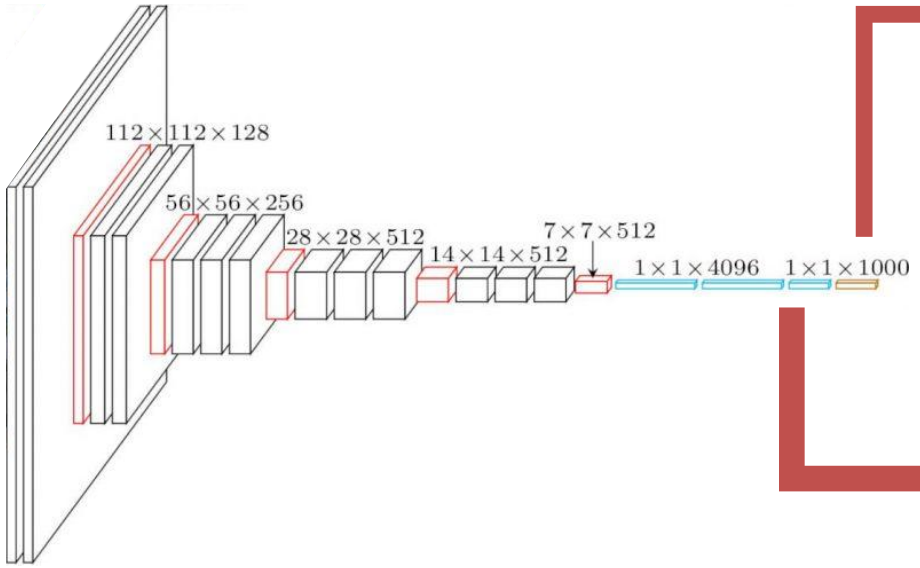
Task: Object Detection - Let's Classify and Locate

- Object detection is just classification and localization combined:
  - Classification using standard CNN;
  - Localization using regression problem for predicting box coordinates
  - Combining loss from Classification (Softmax) and Regression (L2)

Input



VGG



Classification:

- Dog: 0.95
- Cat: 0.02
- Human: 0.01
- ...

Bounding-Box :

- x
- y
- width (w)
- height (h)

Output

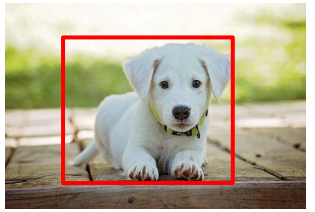
Softmax Loss

Multi-Task Learning

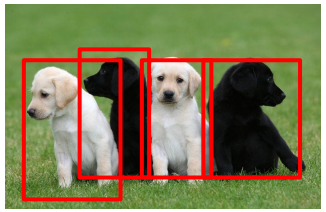
L2-norm Loss

Sliding Windows, from Single to Multiple Objects

- Might work for single object, but not for multiple objects
- Each image containing “x” objects: needs “x” number of classification and localization outputs
- Solution for multiple objects:
  - Crop the image “in a smart way”
  - Apply the CNN to each crop

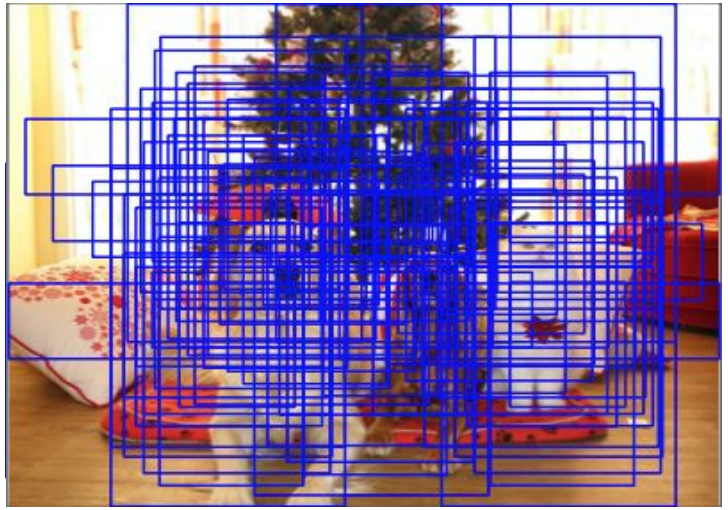


Dog: (x, y, w, h)



Dog: (x, y, w, h)
Dog: (x, y, w, h)
Dog: (x, y, w, h)
Dog: (x, y, w, h)

- Can we just use sliding windows?
  - Problem: Need for applying CNN to huge number of locations, scales, bbox aspect ratios: very computationally expensive;
  - Solution: Region Proposals methods to find object-like regions.



Adapted from Fei-Fei Li & Justin Johnson & Serena Yeung Stanford CS231n 2019
“Convolutional Neural Networks for Visual Recognition” Lecture 12 Slide 37
Uijlings et al, Selective Search for Object Recognition” IJCV 2013 link

Object Detection: Region Proposal Networks!

- **Problem:** Need for applying CNN to huge number of locations, scales, bbox aspect ratios, very computationally expensive!
- **Solution:** Region Proposals methods to find object-like regions:
- **Selective Search Algorithm:** returns boxes that are likely to contain objects:
    - Use hierarchical segmentation;
    - Start with small superpixels;
    - Merge based on similarity.

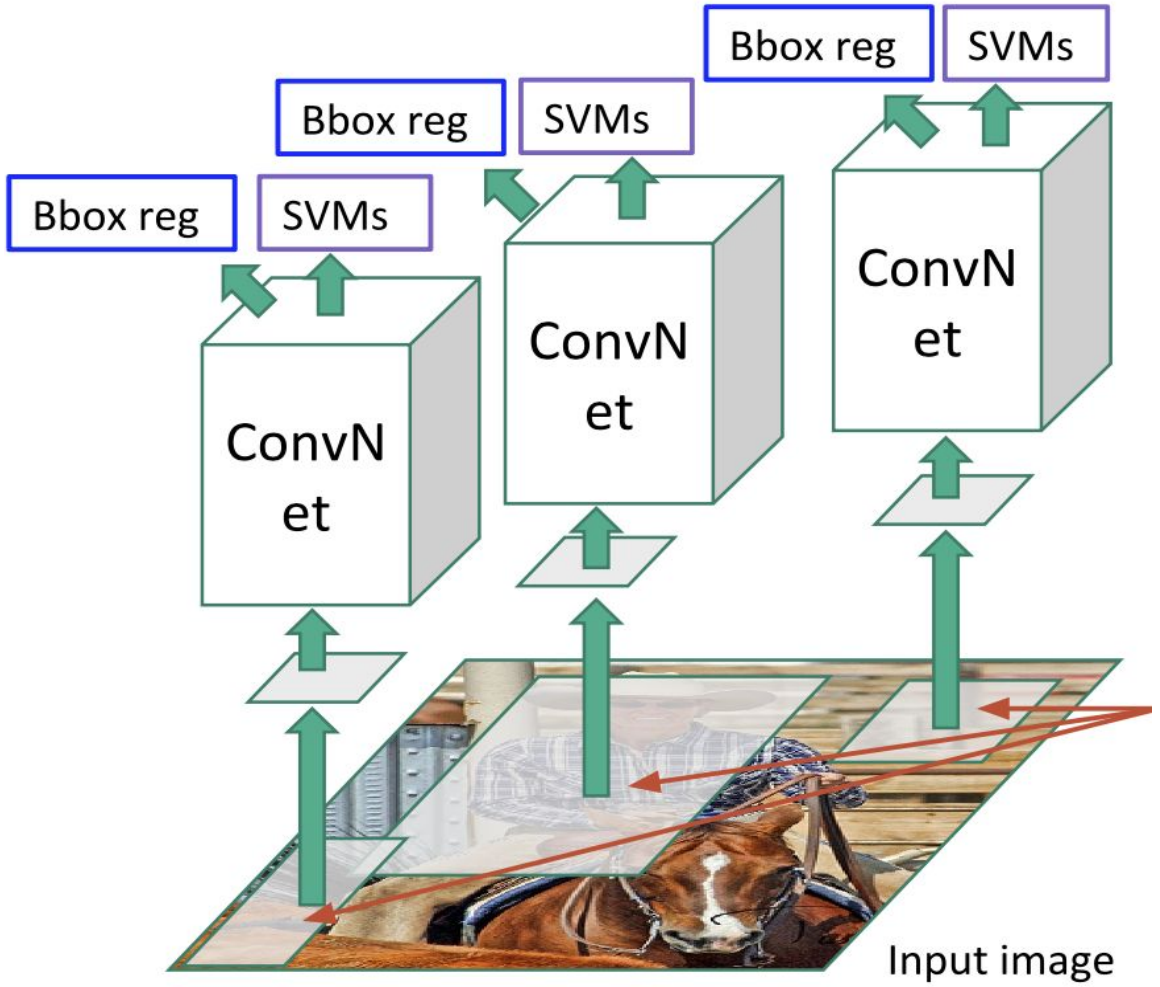
- **Output:** Where are object like regions?
    - No classification yet.

Uijlings et al, Selective Search for Object Recognition" IJCV 2013 link

The Evolution of R-CNN: R-CNN, Fast R-CNN, Faster R-CNN

- **R-CNN = Region-based CNN**
- Correct BBox by Bbox regressor (dx,dy,dw,dh)
- Forward each region through CNN
- Resize proposed RoI (224x224)

- Region of Interest (RoI) from selective search region proposal (approx 2k)
- **Problem:** need to do 2k independent forward passes for each image! **('slow' R-CNN)**



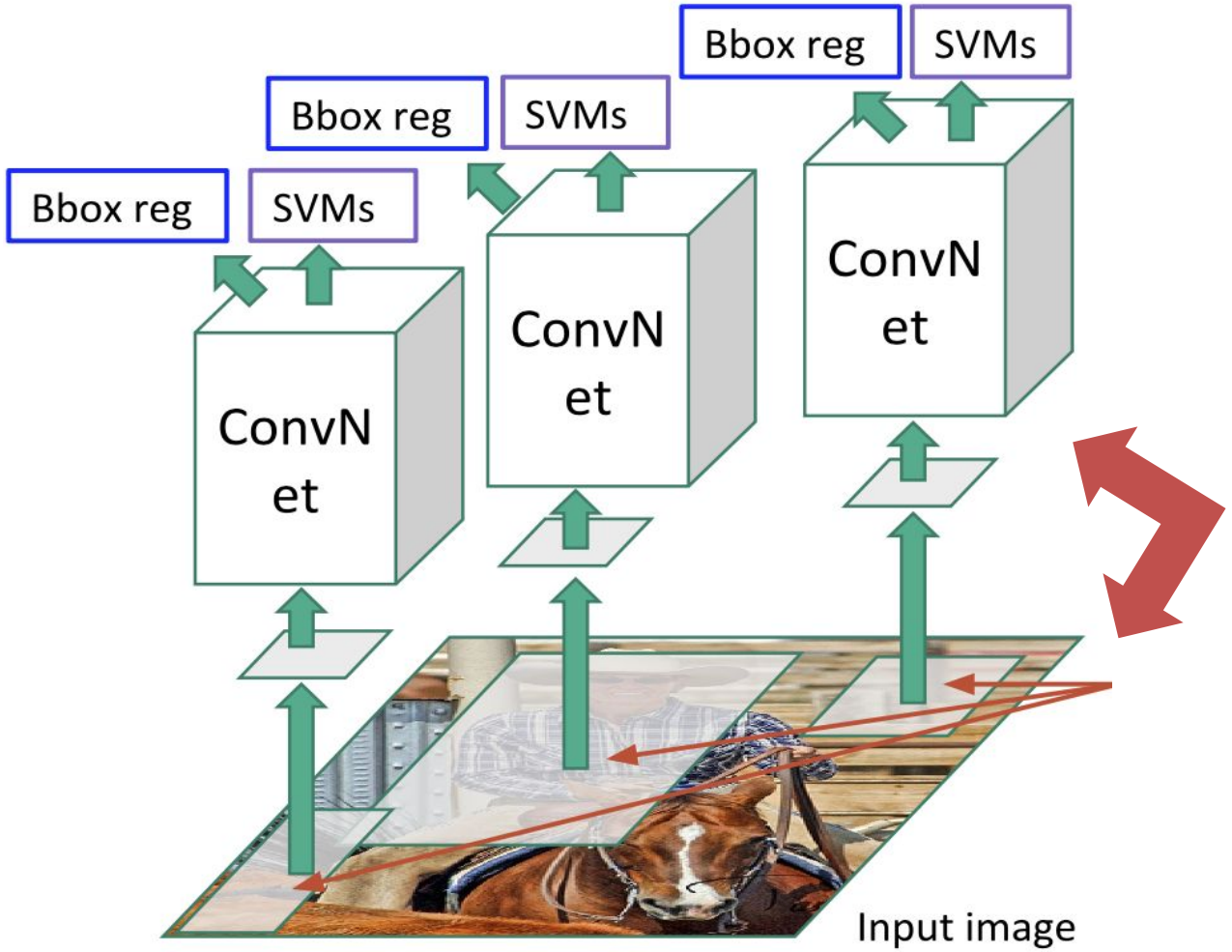
Adapted from Fei-Fei Li & Justin Johnson & Serena Yeung Stanford CS231n 2019 "Convolutional Neural Networks for Visual Recognition"
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation" CVPR2014
Ross Girshick, "Fast R-CNN" Slides 2015

The Evolution of R-CNN: R-CNN, Fast R-CNN, Faster R-CNN

- **R-CNN = Region-based CNN**
- Correct BBox by Bbox regressor (dx,dy,dw,dh)
- Forward each region through CNN
- Resize proposed RoI (224x224)

- Region of Interest (RoI) from selective search region proposal (approx 2k)
- **Problem:** need to do 2k independent forward passes for each image! **('slow' R-CNN)**

- **Solution:** can we process the image before cropping?

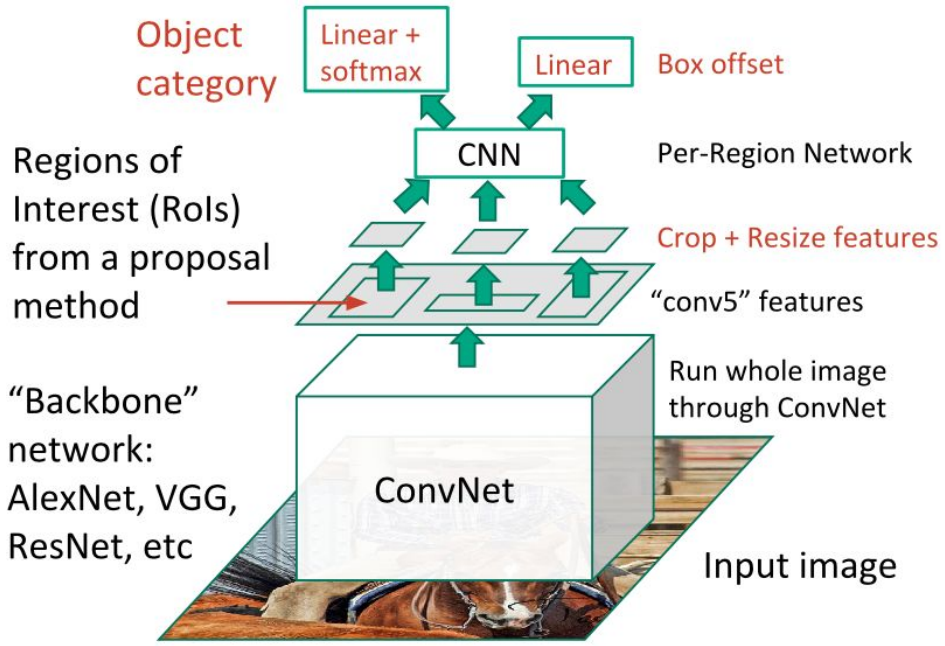


Adapted from Fei-Fei Li & Justin Johnson & Serena Yeung Stanford CS231n 2019 "Convolutional Neural Networks for Visual Recognition"
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation" CVPR2014
Ross Girshick, "Fast R-CNN" Slides 2015

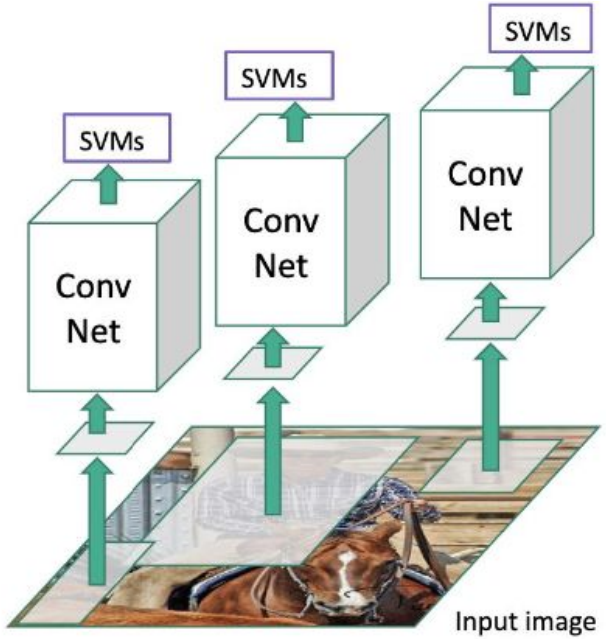
The Evolution of R-CNN: R-CNN, Fast R-CNN, Faster R-CNN

- **Problem**: need to do 2k independent forward passes for each image! **('slow' R-CNN)**
- Even inference is slow: 47s/image with VGG16 [Simonyan & Zisserman, ICLR 15]
- **Solution**: can we process (CNN forward pass) the image before cropping generates 2k regions?

Fast R-CNN



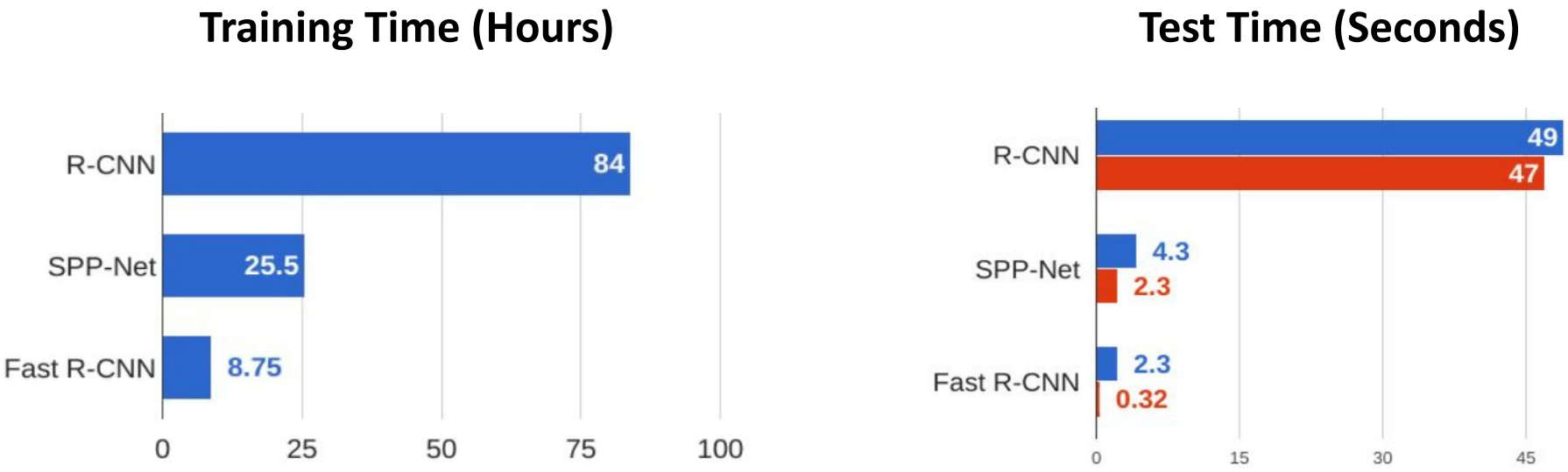
Slow R-CNN



Adapted from Fei-Fei Li & Justin Johnson & Serena Yeung Stanford CS231n 2019 "Convolutional Neural Networks for Visual Recognition"
Ross Girshick, "Fast R-CNN" Slides 2015

The Evolution of R-CNN: R-CNN, Fast R-CNN, Faster R-CNN

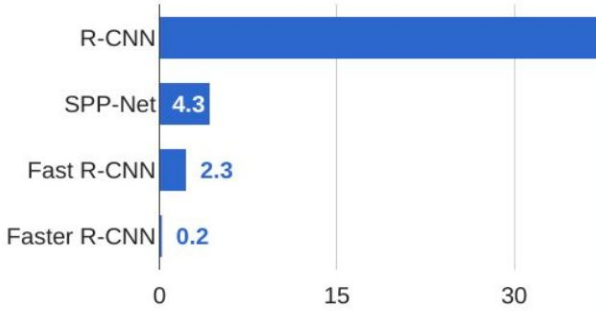
- Fast R-CNN is much faster than R-CNN
- Runtime dominated by region proposals; an iterative method (‘like selective search’);
- **Solution:** Can we make the CNN do proposals?!

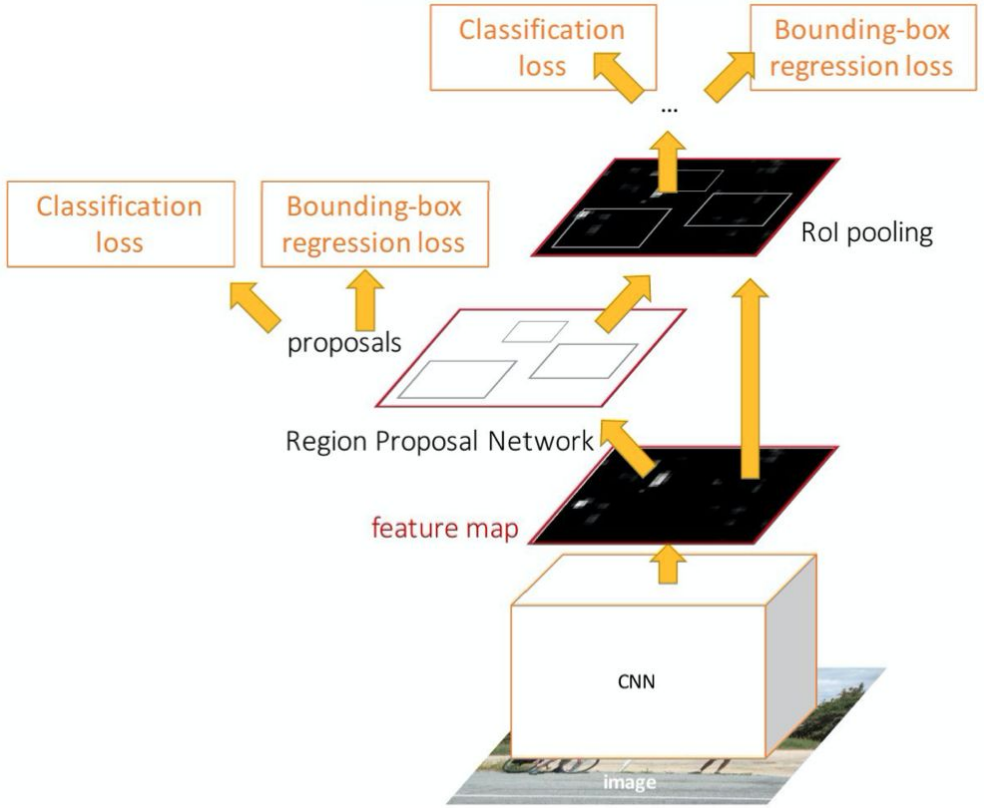


Adapted from Fei-Fei Li & Justin Johnson & Serena Yeung Stanford CS231n 2019 “Convolutional Neural Networks for Visual Recognition”
Ross Girshick, “Fast R-CNN” Slides 2015

The Evolution of R-CNN: R-CNN, Fast R-CNN, Faster R-CNN

- **Faster R-CNN**: Have the CNN make proposals! (single forward, not iterative selective search)
- **CNN Region Proposal Network (RPN)**: Predict region proposals from features
- Otherwise same as Fast R-CNN: crop and classify
- End-to-end quadruple loss:
  - RPN classify object / not object
  - RPN regress box coordinates
  - Final classification score (object classes)
  - Final box coordinates
- Test-time seconds per image:

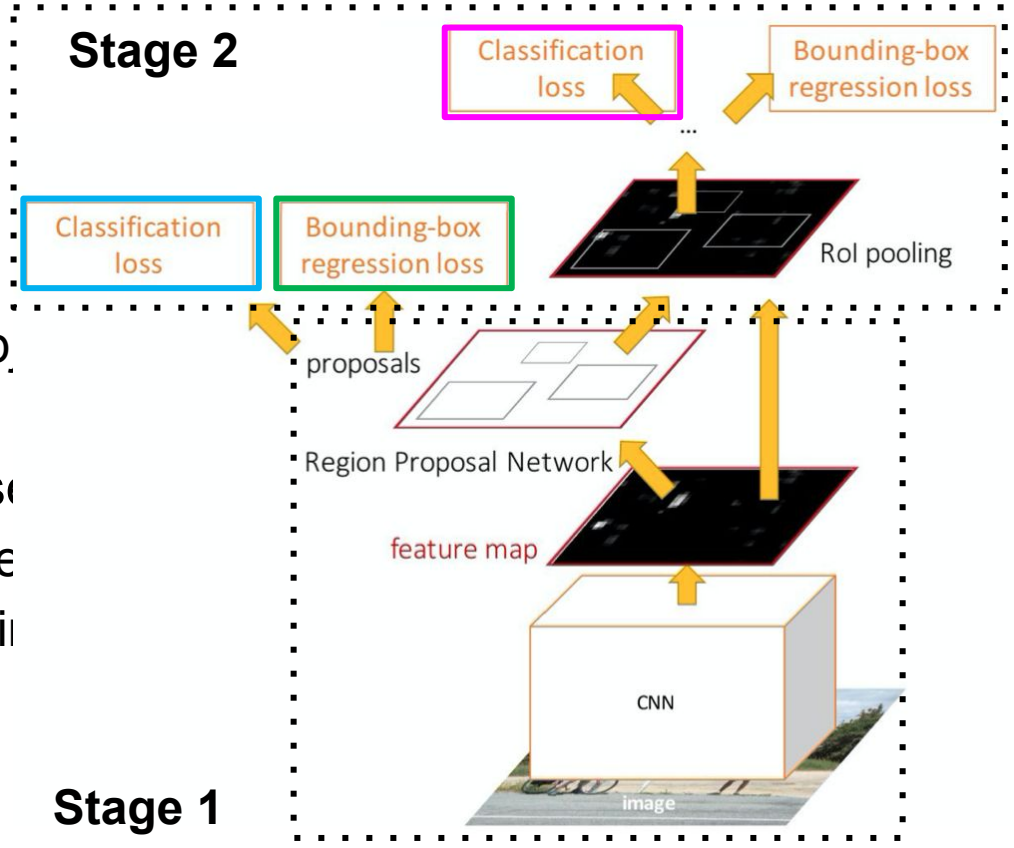




Adapted from Fei-Fei Li & Justin Johnson & Serena Yeung Stanford CS231n 2019 "Convolutional Neural Networks for Visual Recognition"
Ross Girshick, "Fast R-CNN" Slides 2015

The Evolution of R-CNN: R-CNN, Fast R-CNN, Faster R-CNN

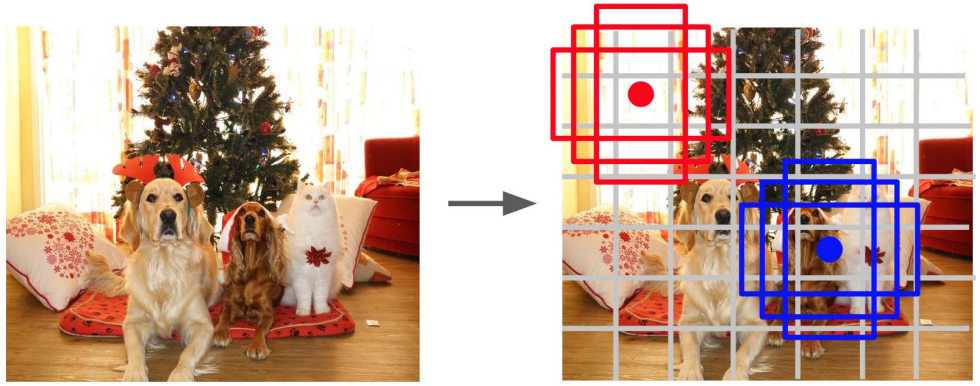
- Previously we said: “Multiple objects? We need Region Proposal Networks!”
- **Faster R-CNN is a two-stage object detector**
  - **Stage 1:** backbone network + RPN (once/image)
  - **Stage 2:** crop - predict object & bbox (once/region)
- What is our RPN again?
- RPN runs prediction on many many anchor boxes:
  - **Loss 1:** Tells is does the anchor bbox contain an ob
  - **Loss 2:** For the top 300 boxes its adjusts the box
- What is the difference between our 2 classification loss
  - one is classifying **object** (i.e. object/not object) – gre
  - one is classifying specific **categories** (e.g. dog) – pi
- Do we really need two stages?



Adapted from Fei-Fei Li & Justin Johnson & Serena Yeung Stanford CS231n 2019 “Convolutional Neural Networks for Visual Recognition”
Ross Girshick, “Fast R-CNN” Slides 2015

Single-Stage Detection Without Region Proposals: YOLO, SSD

- Within each **NxN** grid, regress over each **B** base boxes, predict: (x,y,h,w, confidence = 5)
- **Predict C** category specific class scores
  - Output : N x N x S (5 B + C)
- YOLOv3 (Joseph Redmon):
  - predicts at 3 scales, S = 3
  - predicts 3 boxes at each scale, B=3
  - Darknet-53 as feature extractor (similar to ResNet 152, and 2x faster!)

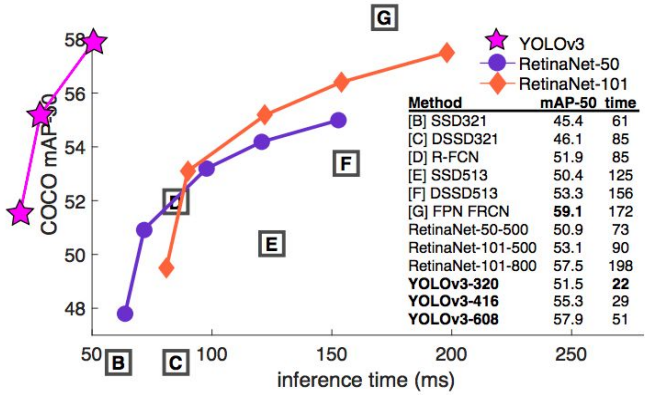


Input image
3 x H x W

Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017



(YOLO) Redmon, "You Only Look Once: Unified, Real-Time Object Detection" CVPR 2015: Cited by 8057 (link)

Semantic Segmentation

Semantic Segmentation: Classify Each Pixel

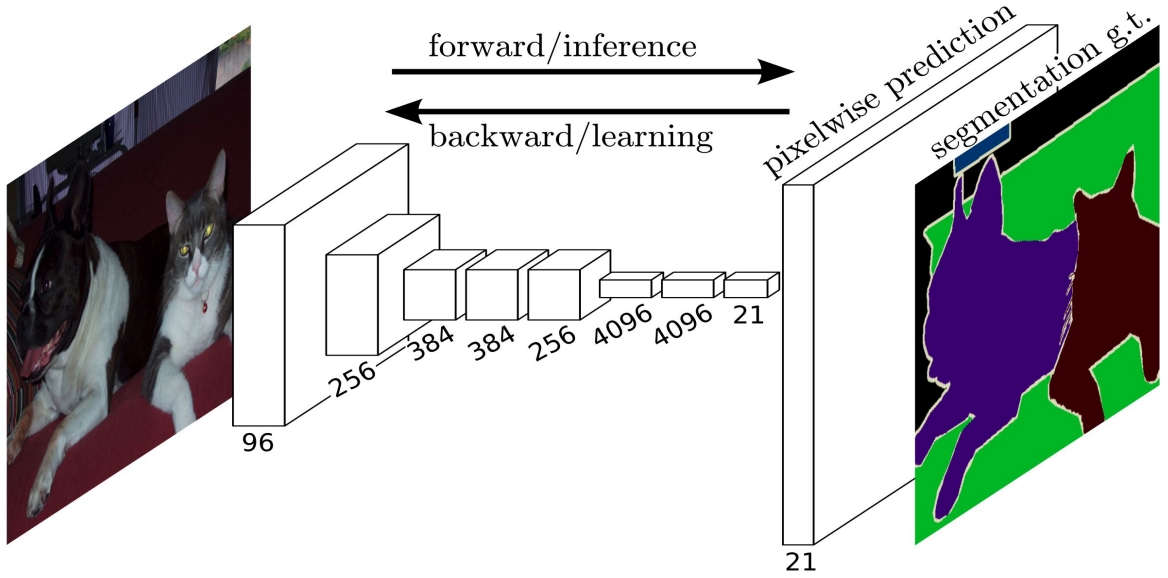
- Fully-Convolutional Networks
- SegNet & U-NET
- Faster R-CNN linked to Semantic Segmentation: Mask R-CNN

Semantic Segmentation: Classify Every Pixel

- **Image Classification:** assigning a single label to **the entire picture**
- **Semantic Segmentation:** assigning a semantically meaningful **label to every pixel in the image**

So our output shouldn't be a class prediction (C numbers) but a picture (C x *w* x *h*)

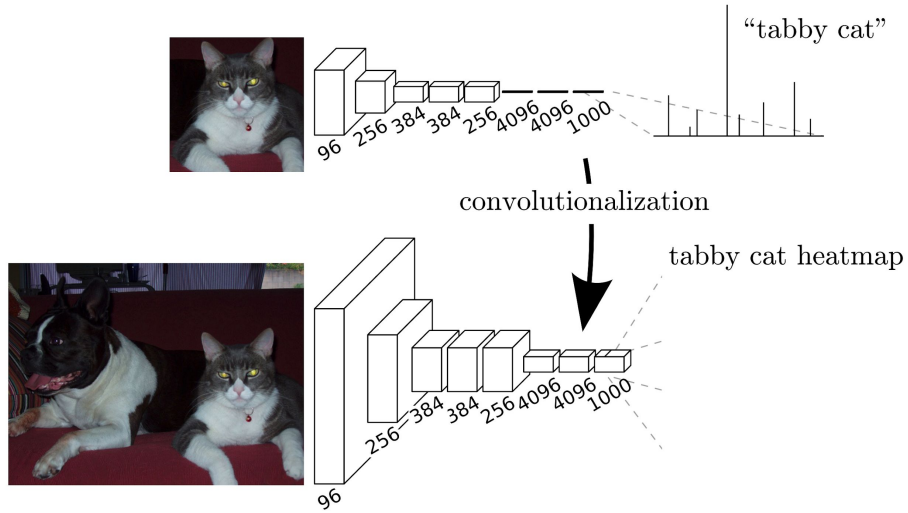
- Can we have a network for each pixel location?
- Sliding window inputs of patches predicting the class of the pixel in the center?
- Many forward passes! Not reusing overlapping patches and features.



(FCN) Long, Shelhamer et al. "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015: Cited by 14480 (link)

Fully-Convolutional Networks

- Semantic segmentation: assigning a semantically meaningful label to every pixel in the image
- So our output shouldn't be a classification prediction (C numbers) but a picture (C x w x h)
  - Maybe we can have a network for each pixel location? Many (w times h) networks!
  - Sliding window inputs of patches predicting the class of the pixel in the center? Many forward passes! Overlapping features not used.
- Solution: FCN = Fully-Convolutional Networks! (not fully-connected)
  - 1 network - 1 prediction would be a lot better
  - Why convolutions? every pixel is very much influenced by its neighborhood



(FCN) Long, Shelhamer et al. "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015: Cited by 14480 (link)

Fig: top, Image Classification (FC), bottom, Image Segmentation (FCN)

89

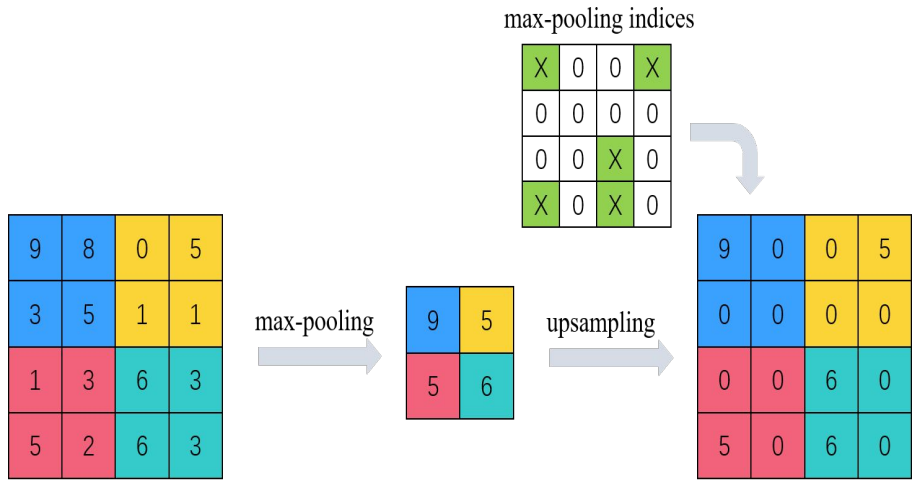
Fully-Convolutional Networks

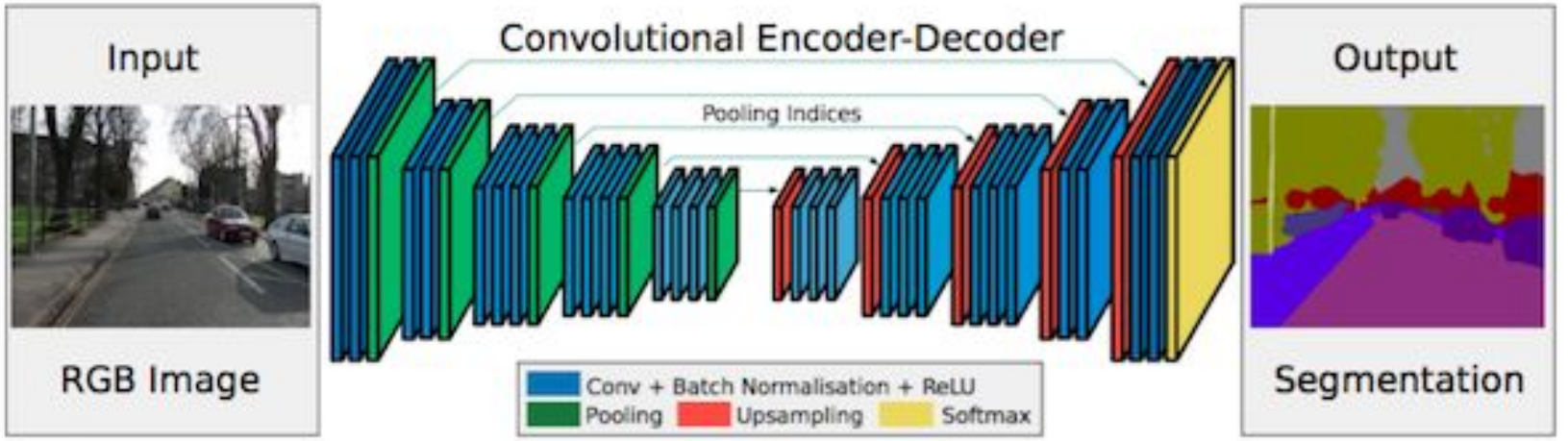
- **FCN:** design a network as a bunch of conv layers to make predictions for all pixels all at once.
  - **Encoder** (= Localization)**:** **downsample** through **convolutions.** Reduces number of params (bottleneck), can make network deeper
  - **Decoder** (= Segmentation): **upsampled** through **transposed convolutions**
  - **Loss:** cross-entropy loss on every pixel.
- **Contribution:**
  - Popularize the use of end-to-end CNNs for semantic segmentation;
  - Re-purpose imagenet pretrained networks for segmentation = Transfer Learning
  - Upsample using transposed layers.
- **Negative:**
  - upsampling = loss of information during pooling;
  - 224x224 image downsampled to 20x20 back upsampled to 224x224.

(FCN) Long, Shelhamer et al. "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015: Cited by 14480 (link)

SegNet

- The indices from max pooling down sampling are transferred to the decoder: **pooling indices**
- Improves fine segmentation resolution, we want “pixel-perfect”;
- More efficient since no transposed convolutions to learn.

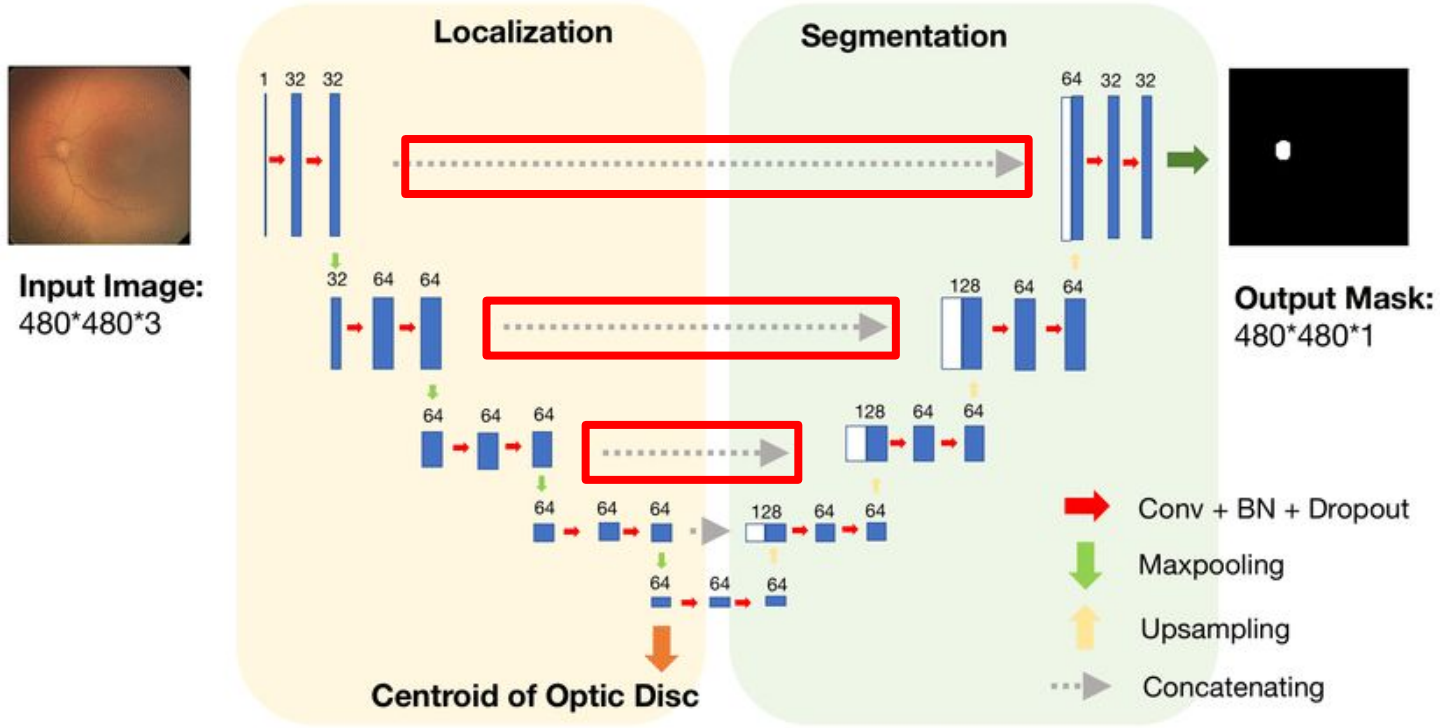




SegNet: A deep Convolutional Encoder-Decoder Architecture for Image Segmentation. (link)

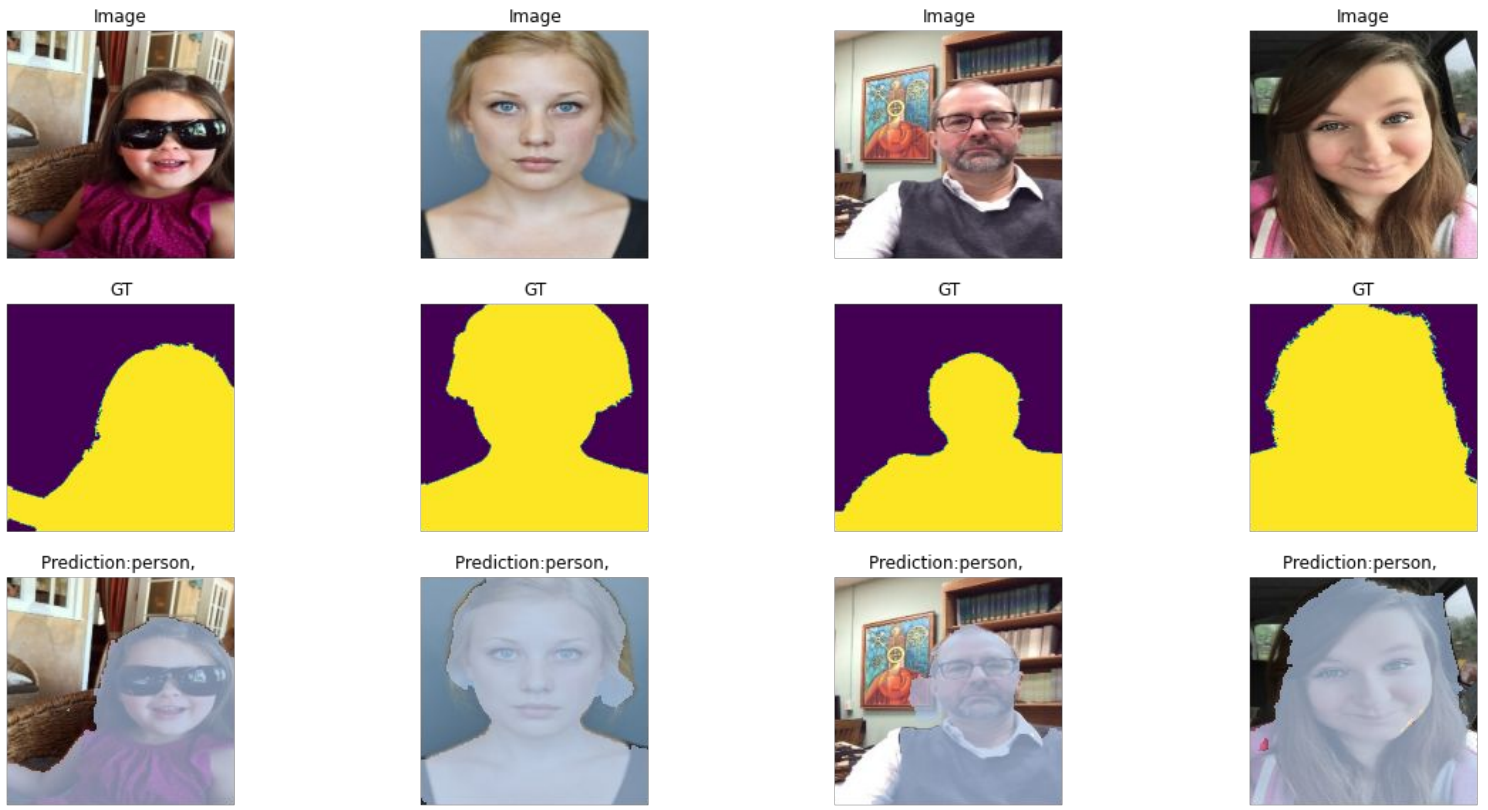
U-NET: Long Skip Connections

- The U-Net is an encoder decoder using:
  - **location information** from the down sampling path of the encoder;
  - **contextual information** in the up sampling path by the "concatenating" long-skip connections.



Tutorial: Using Transfer Learning to train a U-NET

[Colab Notebook]



References

Presentations:

- Fei-Fei Li & Justin Johnson & Serena Yeung Stanford CS231n 2019/2018 “Conv. Neural Networks for Visual Recognition” Lecture 12 !
    - BTW: Great course / youtube series (youtube 2017)
- Ross Girshick, “Fast R-CNN” Slides 2015 (link)

Papers:

- **VGG** Simonyan, Zisserman. “Very Deep CNNs for Large-scale Image Recognition”, ILSVRC 2014: Cited by 34652 (link)
- **Select. Search** Uijlings et al, Selective Search for Object Recognition” IJCV 2013: Cited by 3944 (link)
- **R-CNN** Girshick et al, “Rich feature hierarchies for accurate object detect. & sem. segmentation” CVPR2014: Cited by 12000 (link)
- **Fast-R-CNN** Girshick, ‘Fast R-CNN“ ICCV 2015: Cited by 8791 (link)
- **Faster- R-CNN** Ren et al, “Faster R-CNN: Real-Time Object Det. with Region Proposal Networks” NEURIPS 2015 Cited by 16688 (link)
- **Mask-R-CNN** He et al, “Mask R-CNN” ICCV 2017: Cited by 5297 (link)
- **YOLO** Redmon, “You Only Look Once: Unified, Real-Time Object Detection” CVPR 2015: Cited by 8057 (link)
- **FCN** Long, Shelhamer et al. “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015: Cited by 14480 (link)
- **SegNet** Badrinarayanan et al. “SegNet: A deep Conv Encoder-Decoder Architecture for Image Segmentation”. Cited by 4258 (link)
- **U-Net** Ronneberger et al. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. Cited by 12238 (link)

Outline

1. Introduction to Transfer Learning
2. Review CNNs
3. SOTA Deep Models
4. Transfer Learning across Tasks
5. Tutorial: Segmentation
6. **Model Compression Techniques**
7. Tutorial: Model Compression
8. Tutorial: Mushroom App Models

Motivation

We want to process data (ideally a lot) and we do not have enough computing resources. For example:

1. Your phone can't run GoogleNet to assist you in some tasks
2. You can't compress ginormous number of images coming from space (8Kx8K pixels from 3K satellites)

Using machine learning is resource intensive:

i. Computing power to train millions of parameters or predict for many observations

ii. Limited bandwidth

So what? Model compression techniques

Hannah Peterson and George Williams, *An Overview of Model Compression Techniques for Deep Learning in Space*, August 2020

What is Model Compression?

The main idea is to simplify the model without diminishing accuracy. A simplified model means reduced in size and/or latency from the original. Both types of reduction are desirable.

- Size reduction can be achieved by reducing the model parameters and thus using less RAM.
- Latency reduction can be achieved by decreasing the time it takes for the model to make a prediction, and thus lowering energy consumption at runtime (and carbon footprint).

Karen Hao, *Training a single AI model can emit as much carbon as five cars in their lifetimes*, June 2019

Compression Techniques

- Knowledge distillation
- Pruning
- Quantization
- {Low-rank approximation and sparsity}

Compression Techniques

- **Knowledge distillation**
- Pruning
- Quantization
- {Low-rank approximation and sparsity}

Compression Technique: Distillation

Compression Technique: Distillation

Problem:

- During training, a model does not have to operate in real time and does not necessarily face restrictions on computational resources, as its primary goal is simply to extract as much structure from the given data as possible.
- But latency and resource consumption do become of concern if it is to be deployed for inference.

So what? we must develop ways to compress model for inference.

Compression Technique: Distillation

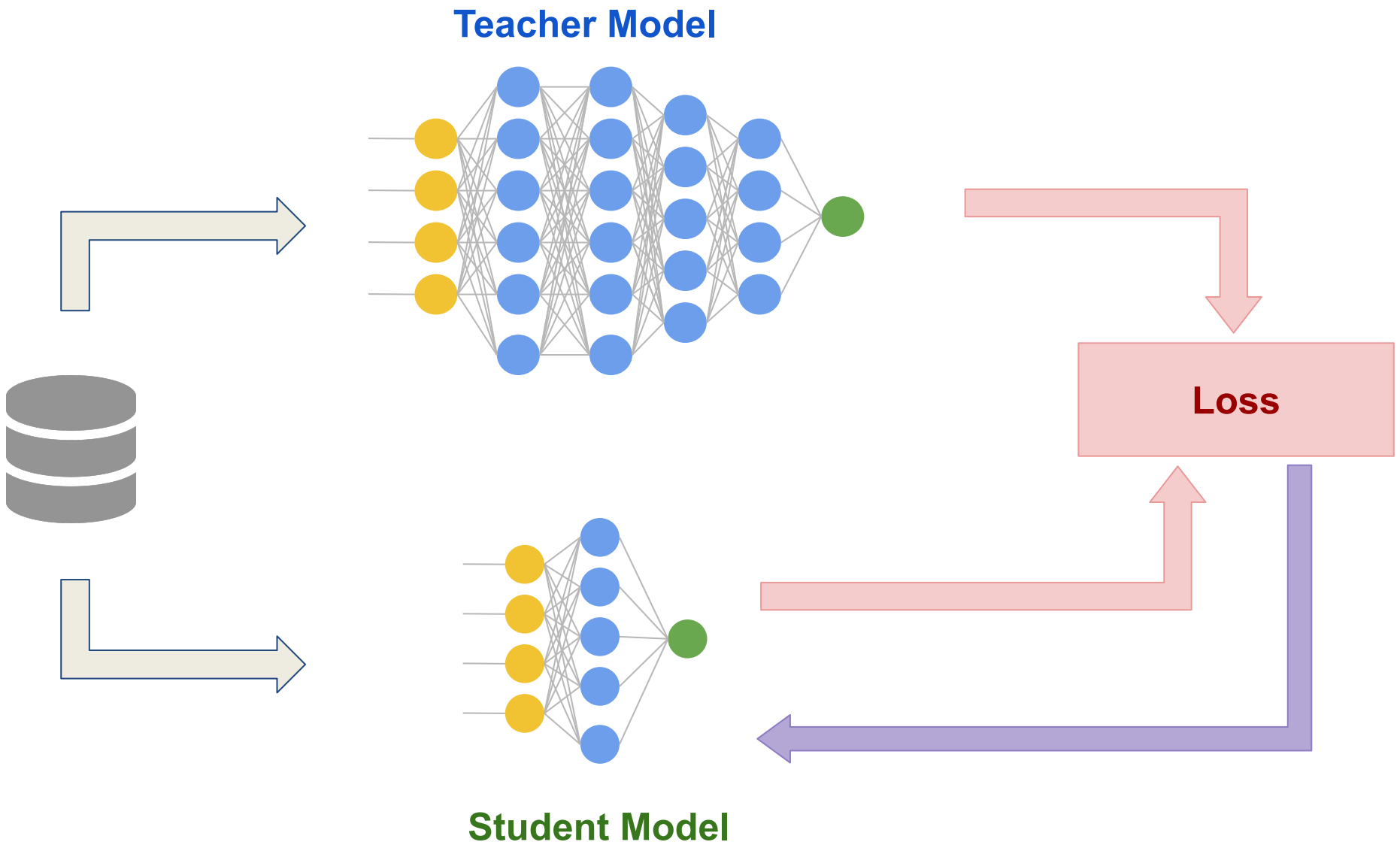
Idea:

- In 2006, Buciluǎ et al. showed that it was possible to transfer knowledge from a large trained model (or ensemble of models) to a smaller model for deployment by **training it to mimic the larger model's output.**
- In 2014 Hinton et al generalized the process and gave the name **Distillation.**

Main idea of distillation is that **training and inference are 2 different tasks;** thus **a different model should be used**.

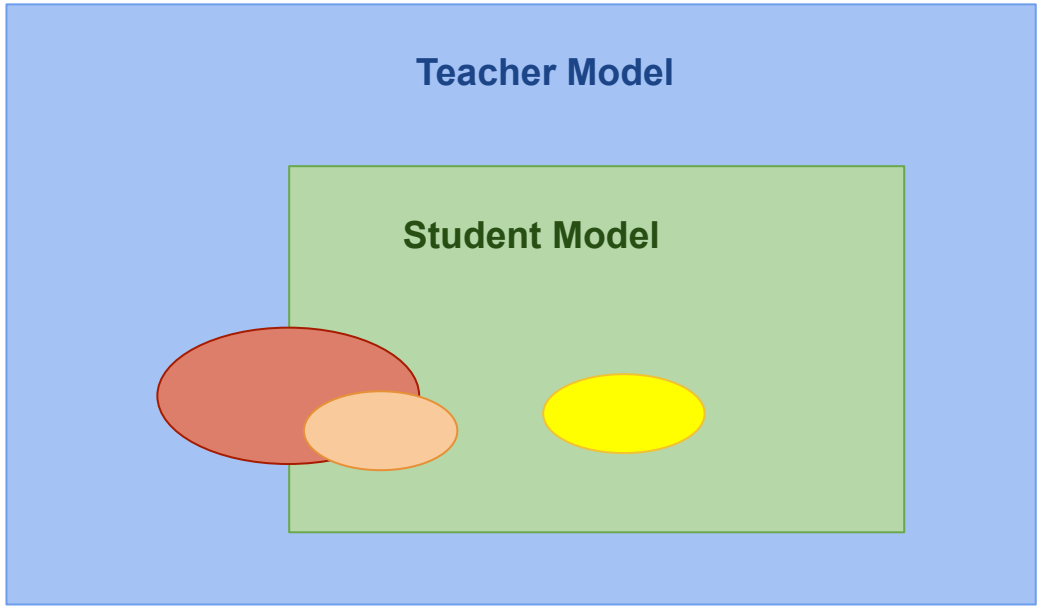
Buciluǎ et al., *Model Compression*, 2006
Hinton et al., *Distilling the Knowledge in a Neural Network*, 2014

Distillation: Teacher Student



Distillation: Teacher Student

Assumption: if we can achieve similar convergence using a smaller network, then the convergence space of the Teacher Network should overlap with the solution space of the Student Network. (design diagram again if needed)



Teacher Convergence Space

Student Convergence Space

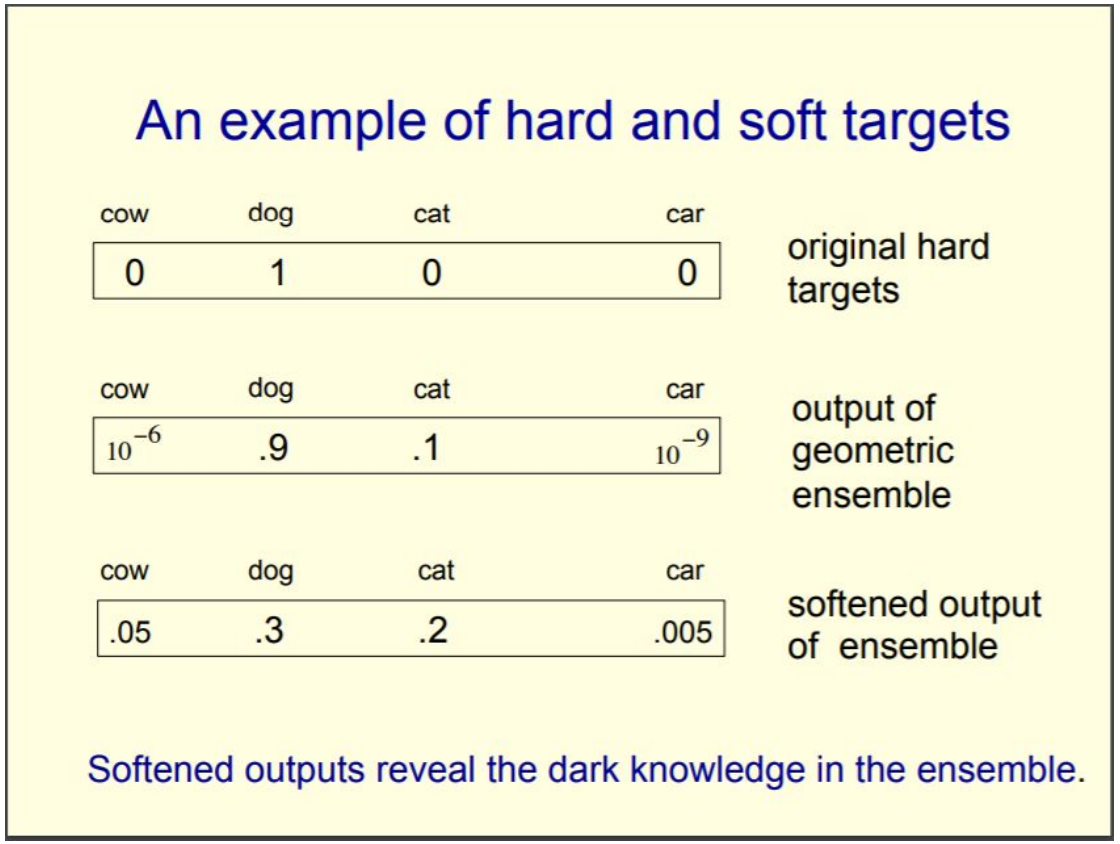
Teacher guided Student Convergence Space

Distillation: Teacher Student Loss

Modified softmax function with Temperature:

$$qi = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

q_i : resulting probability
z_i : logit of a class
z □ : other logits
T: temperature (T=1, “hard output”)

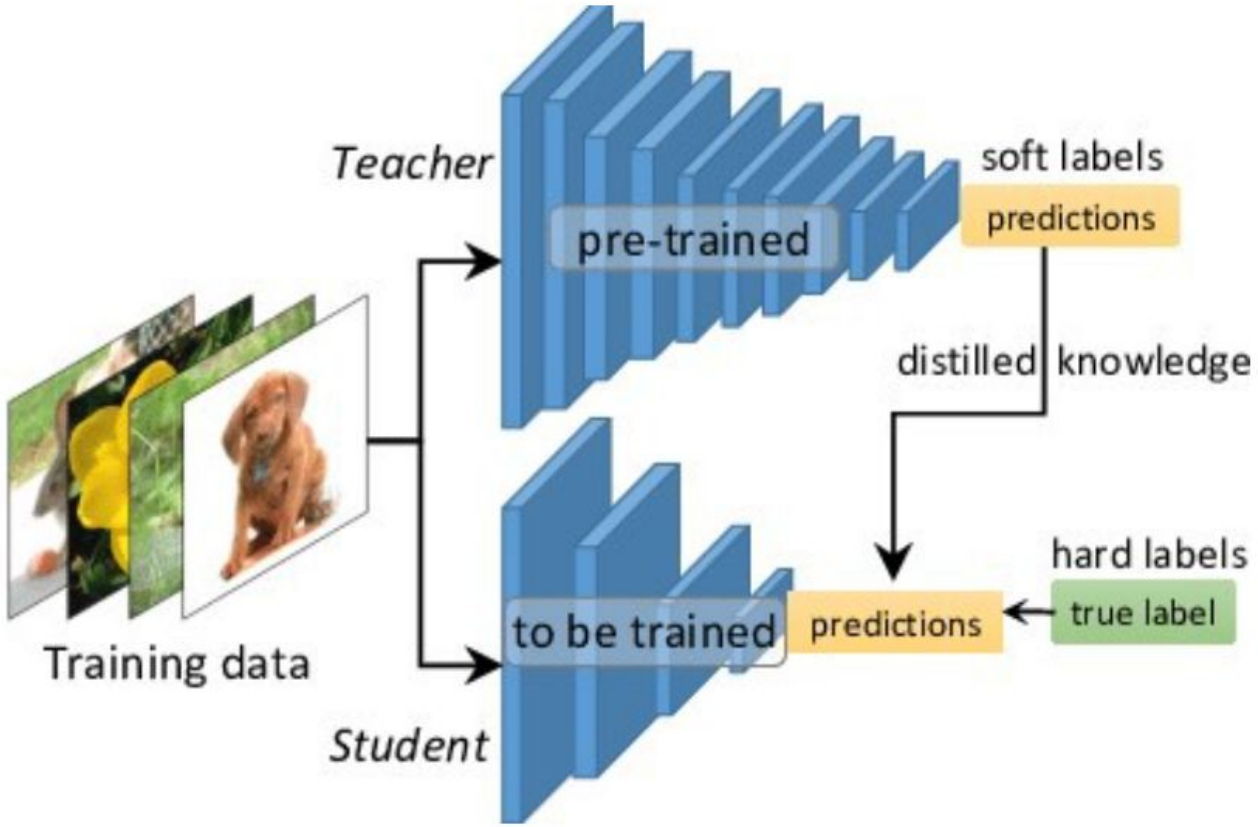


Geoffrey Hinton, Oriol Vinyals & Jeff Dean, *Dark Knowledge*

Distillation: Teacher Student Training

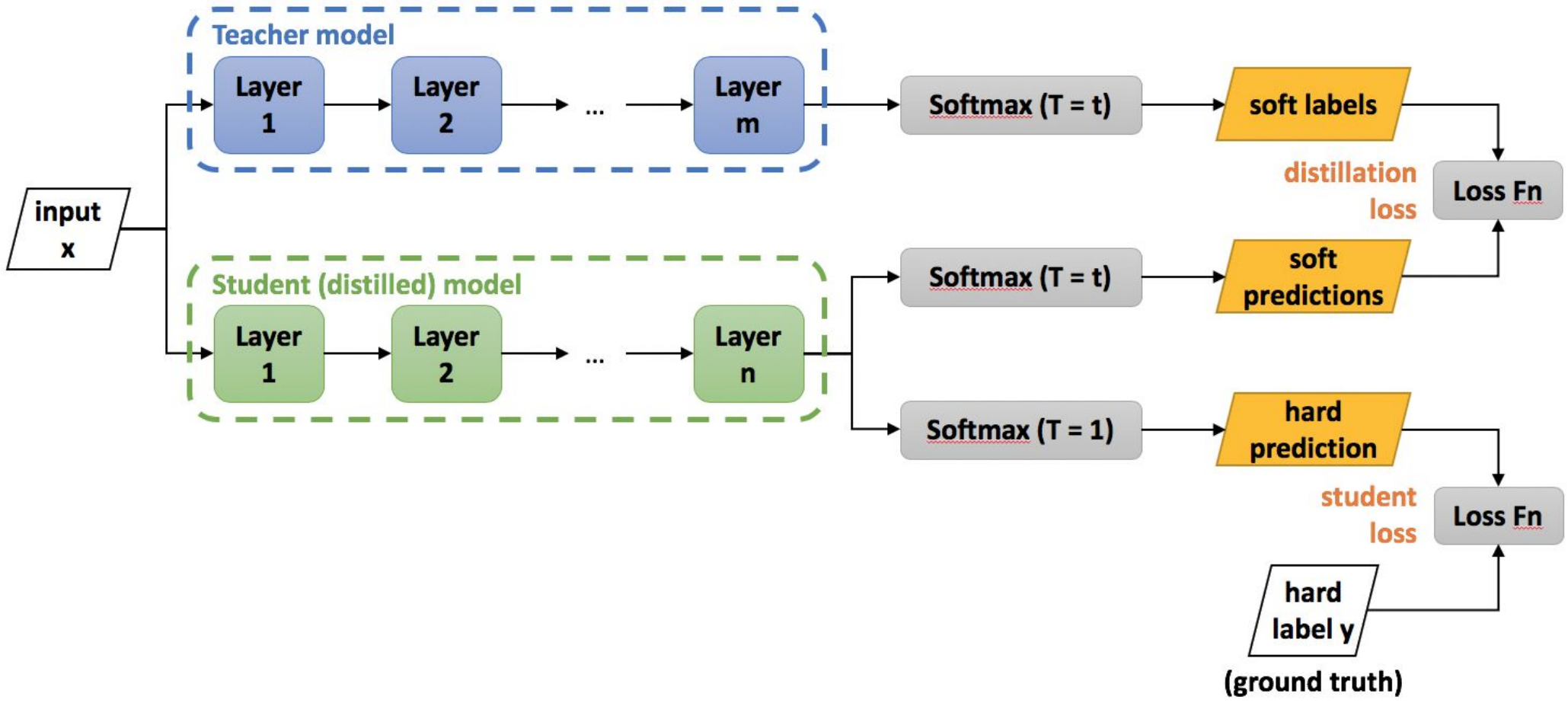
Trained to minimize the sum of two different cross entropy functions:

- one involving the original hard labels obtained using a softmax with T=1
- one involving the softened targets, T>1



Geoffrey Hinton, Oriol Vinyals & Jeff Dean, *Dark Knowledge*

Distillation: Teacher Student Training



Source: https://medium.com/neuralmachine/knowledge-distillation-dc241d7c2322

Tutorial: Model Compression

Colab Notebook

What is next in Distillation?

1: Multiple teachers (i.e. converting an ensemble into a single network).

2: Introducing a teaching assistant (the teacher first teaches the TA, who then in turn teaches the student) etc.

3: Quite young field

A **drawback** of knowledge distillation as a compression technique, therefore, is that there are many decisions that must be made up-front by the user to implement it (student network doesn't even need to have a similar structure to the teacher).

Compression Techniques

- Knowledge distillation
- **Pruning**
- Quantization
- {Low-rank approximation and sparsity}

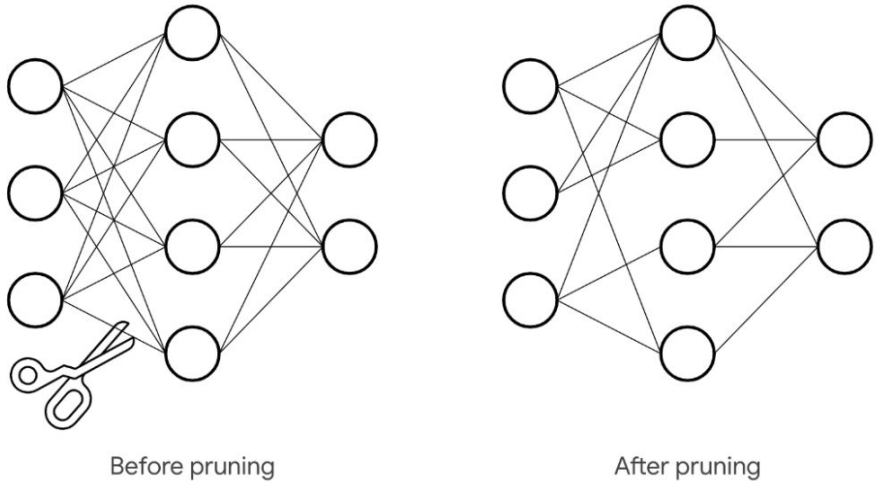
Compression Technique: Pruning

The main idea is to remove features with nearly the same information.

Pruning involves removing connections between neurons, channels, or filters from a trained network. To prune a connection, we set a weight in the matrix to zero.

2 types of pruning:

- Unstructured removes connections or neurons
- Structured removes filters or channels

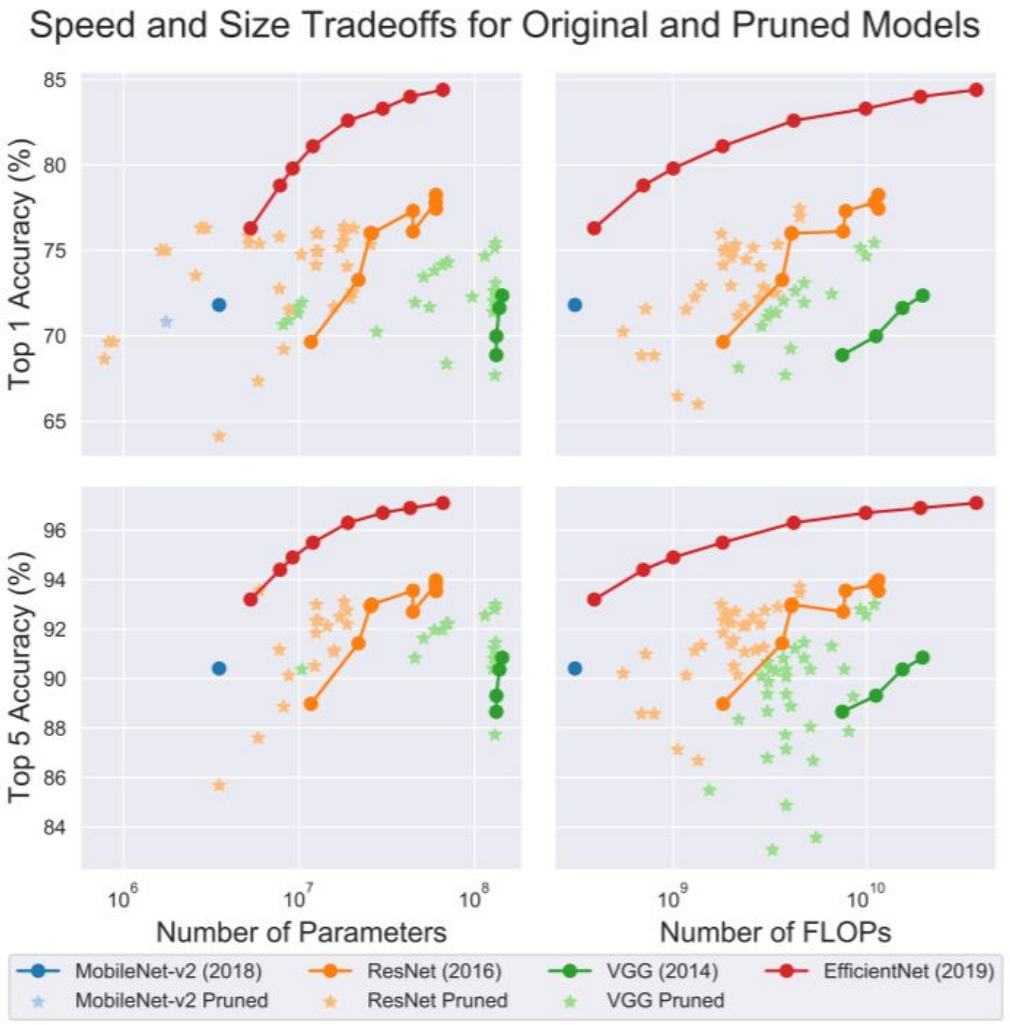


To learn more about pruning: Derrick Mwiti, *Research Guide: Pruning Techniques for Neural Networks*, November 2019

Compression Technique: Pruning

Pruning has a few potential **drawbacks:**

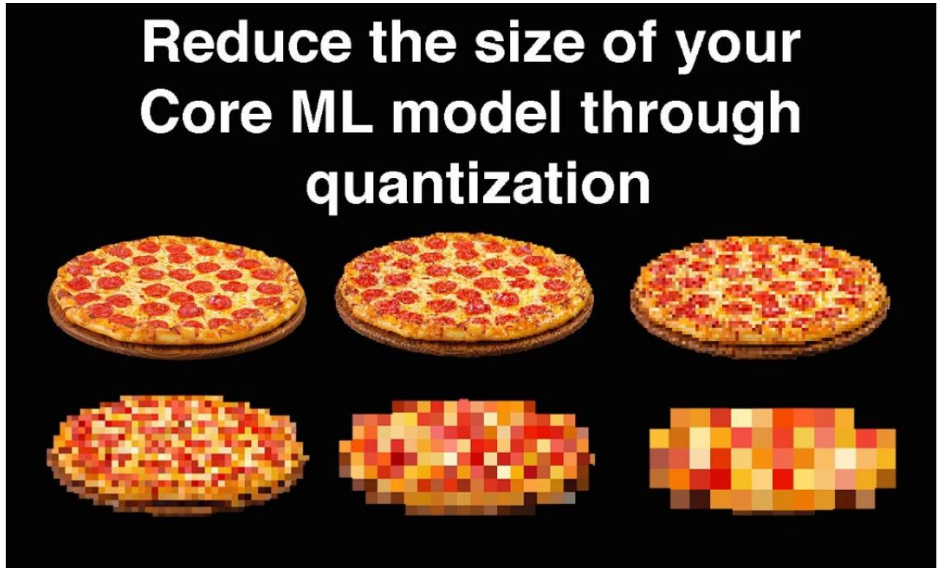
- **Unclear how well given methods generalize** across different architectures.
- **Fine-tuning is cumbersome** and can slow down implementation.
- May be more effective to simply use a **more efficient architecture than to prune a suboptimal one.**



Blalock D. et al, *What is the state of neural network pruning?*, March 2020

Compression Technique: Quantization

Main idea is to map values from a large set to values in a smaller set without losing too much information in the process. So by reducing the number of pixels, the image below should still be clear.

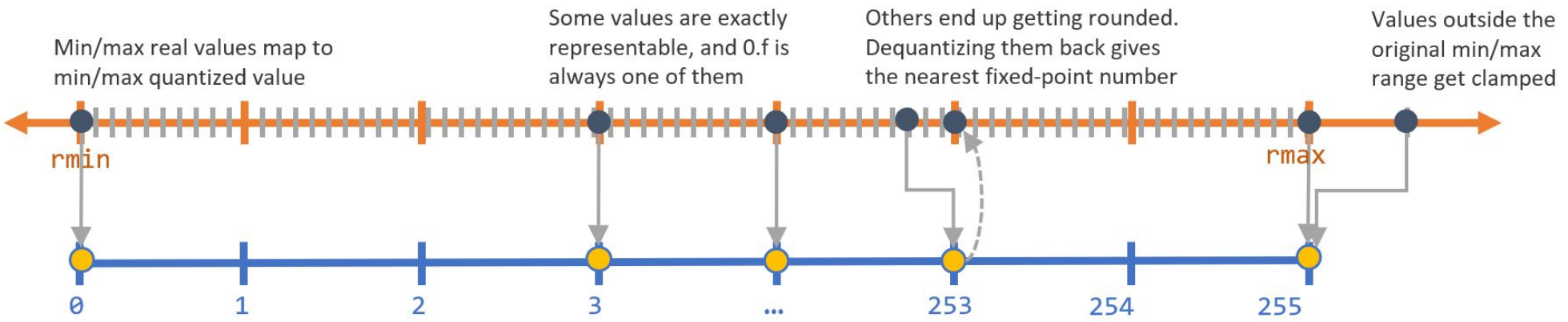


To implement quantization with Tensorflow: MC.AI, *Quantization in Deep Learning using TensorFlow 2.X*, May 2020

Compression Technique: Quantization

Quantization can be achieved by changing the output or NN architecture:

- **Post Training Quantization:** reducing the size of the weights stored (e.g. from 32-bit floating point numbers to 8-bit)



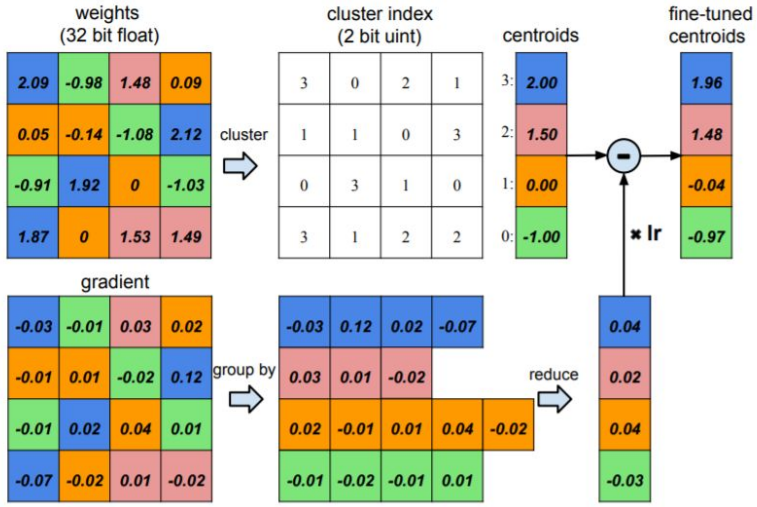
To implement quantization with Tensorflow: MC.AI, *Quantization in Deep Learning using TensorFlow 2.X*, May 2020

Compression Technique: Quantization

Quantization-Aware Training:

There could be an accuracy loss in a post-training model quantization and to avoid this and if you don't want to compromise the model accuracy we do quantization aware training.

This technique ensures that the forward pass matches precision for both training and inference.



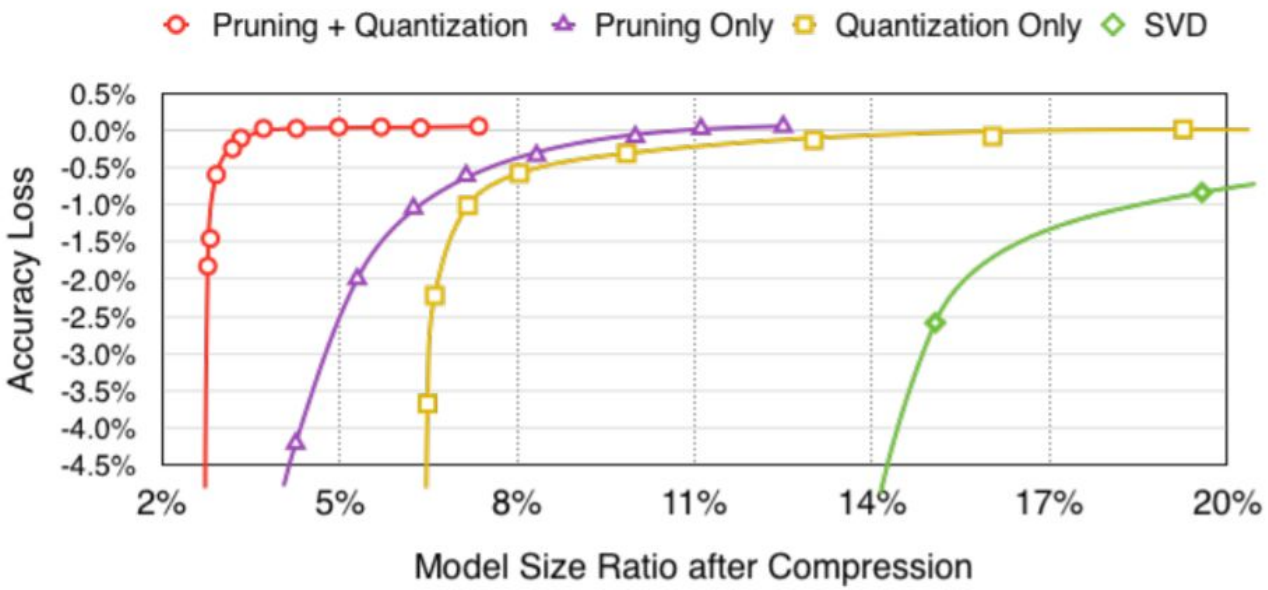
https://www.tensorflow.org/model_optimization/guide/quantization/training

Han S. et al, *Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding*, 2016

Compression Technique: Quantization

Quantization can be **tricky:**

- Requires having a decent **understanding of hardware and bitwise computations**
- **Savings are tied to the features of the hardware** being used



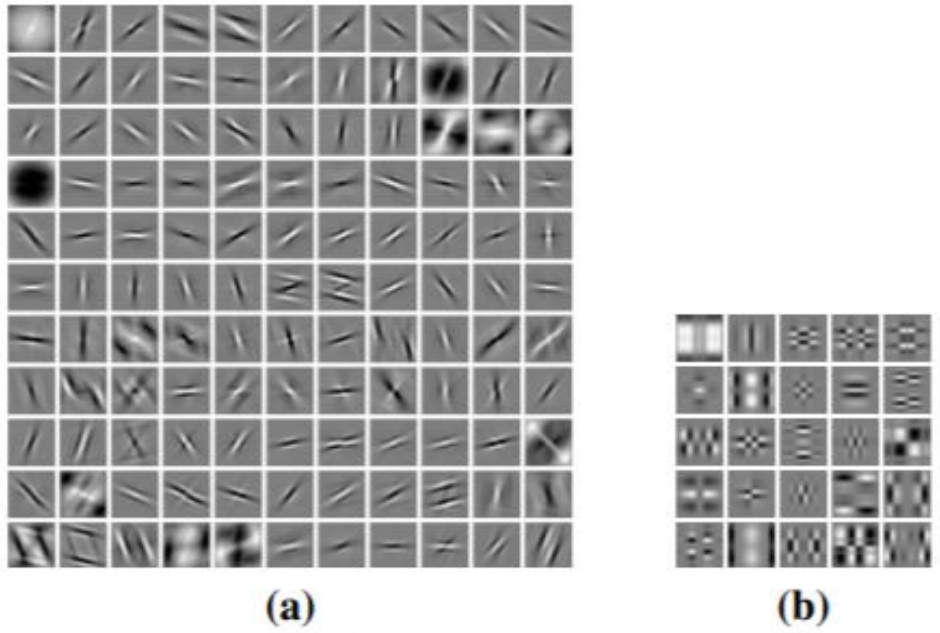
Han S. et al, *Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding*, 2016

Compression Technique: Low Rank Approximation

Main idea is to **approximate the redundant filters of a layer** using a linear combination of fewer filters. Compressing layers in this way reduces the network's memory footprint, the computational complexity of convolutional operations and can yield significant **speedups**.

Examples:

- Singular Value Decomposition
- Tucker decomposition
- Canonical Polyadic decomposition



Rigamonti R. et al., *Learning Separable Filters*, 2013

Compression Technique: Low Rank Approximation

Kim et al. use Tucker decomposition to determine the ranks that the compressed layers should have. They apply the compression to various models for image classification tasks and run them on both a Titan X and Samsung Galaxy S6 phone*:

- Low-rank approximation achieve significant size and latency reductions
- Prove potential deployment on mobile devices
- Reduce parameters simplifying model structure
- Does not require specialized hardware to implement

Model	Top-5	Weights	FLOPs	S6		Titan X
AlexNet	80.03	61M	725M	117ms	245mJ	0.54ms
*AlexNet**	78.33	11M	272M	43ms	72mJ	0.30ms
(imp.)	(-1.70)	(×5.46)	(×2.67)	(×2.72)	(×3.41)	(×1.81)
VGG-S	84.60	103M	2640M	357ms	825mJ	1.86ms
*VGG-S**	84.05	14M	549M	97ms	193mJ	0.92ms
(imp.)	(-0.55)	(×7.40)	(×4.80)	(×3.68)	(×4.26)	(×2.01)
GoogLeNet	88.90	6.9M	1566M	273ms	473mJ	1.83ms
*GoogLeNet**	88.66	4.7M	760M	192ms	296mJ	1.48ms
(imp.)	(-0.24)	(×1.28)	(×2.06)	(×1.42)	(×1.60)	(×1.23)
VGG-16	89.90	138M	15484M	1926ms	4757mJ	10.67ms
*VGG-16**	89.40	127M	3139M	576ms	1346mJ	4.58ms
(imp.)	(-0.50)	(×1.09)	(×4.93)	(×3.34)	(×3.53)	(×2.33)

* S6 has a GPU with 35× lower computing ability and 13× smaller memory bandwidth than Titan

Kim et al, *Compression of deep convolutional neural networks for fast and low power mobile applications*, 2016

Outline

1. Introduction to Transfer Learning
2. Review CNNs
3. SOTA Deep Models
4. Transfer Learning across Tasks
5. Tutorial: Segmentation
6. Model Compression Techniques
7. Tutorial: Model Compression
8. **Tutorial: Mushroom App Models**

Tutorial: Mushroom App Models

Colab Notebook

THANK YOU