

COMS E6998 012/15099

# Practical Deep Learning Systems Performance

Lecture 6 10/10/19

# Logistics

- Homework 2 due Oct 20 by 11:59 PM
- Project proposals due 10/17/2019
- Seminar reading list due 10/17/2019
- Start working on your 10% for technical community participation

## Recall from last lecture

- Downpour SGD and its scaling
- Ring all-reduce algorithm
- Distributed training scaling work
  - Goyal et al (Facebook)
  - Cho et al (IBM )
- Horovod and its benchmarking numbers
- Tensorflow and distribution support (links only)
- Tensor Processing Units
- Convolutional Neural Networks: convolution, padding, pooling, strides

# LeNet (1998)

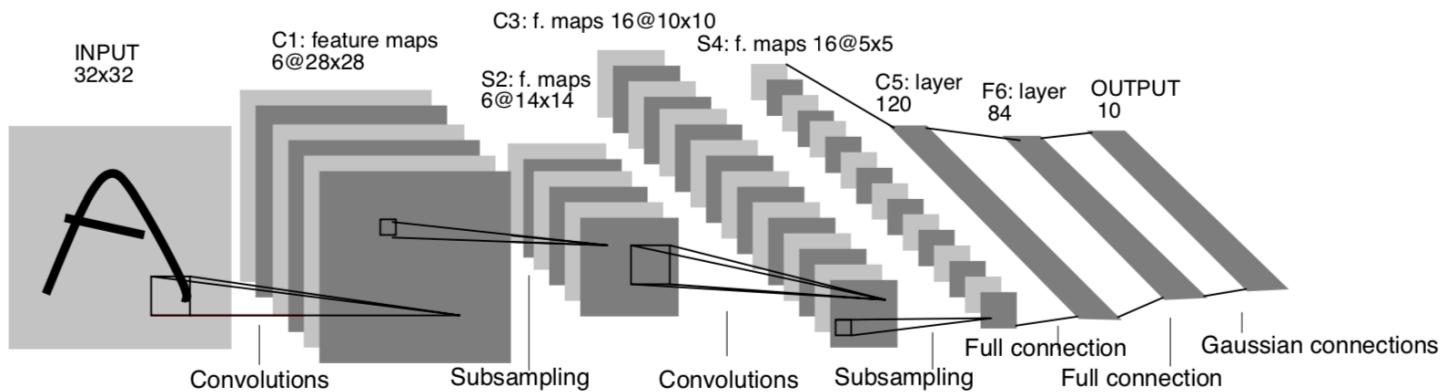


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- First convolutional layer: 6 filters, 5x5, stride 1; # parameters = 156
- First pooling layer: 6 filters, 2x2, stride 2; # parameters = 12
- Second convolutional layer: 16 filters, 5x5, stride 1; # parameters = 1516 (not a regular convolution layer, each unit in a feature map connected to a subset of input feature maps at identical locations, no weight sharing in same map)
- Second pooling layer: 16 filters, 2x2 stride, 2; # parameters = 32
- Third convolutional layer: 120 filters, 5x5, stride 1; # parameters =  $25 \times 16 \times 120 + 120 = 48,120$ . It is a fully connected layer
- Fully connected layer: #parameters =  $84 \times 120 + 84 = 10,164$

# Alexnet (2012)

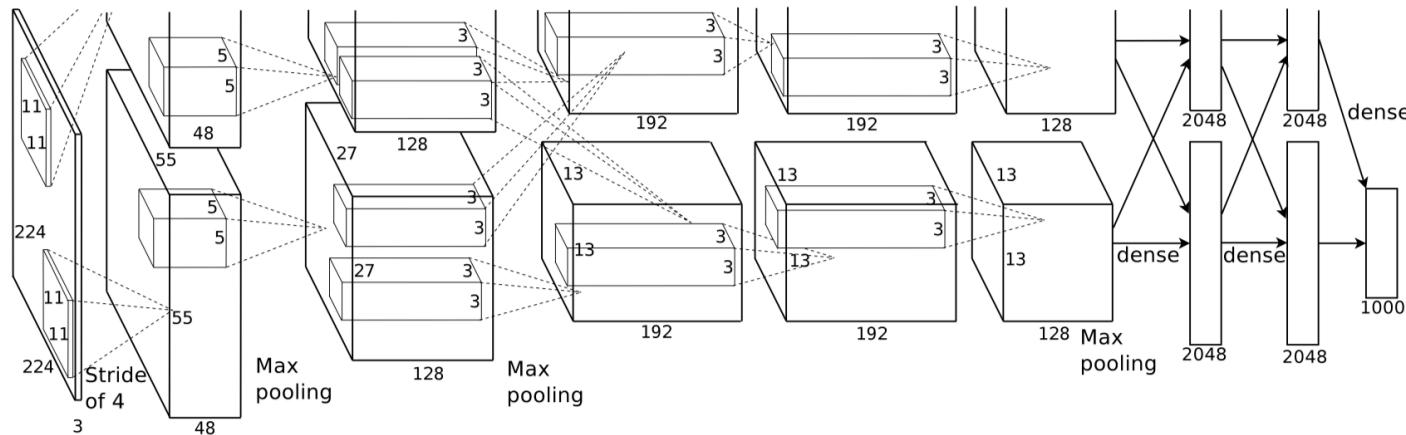


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

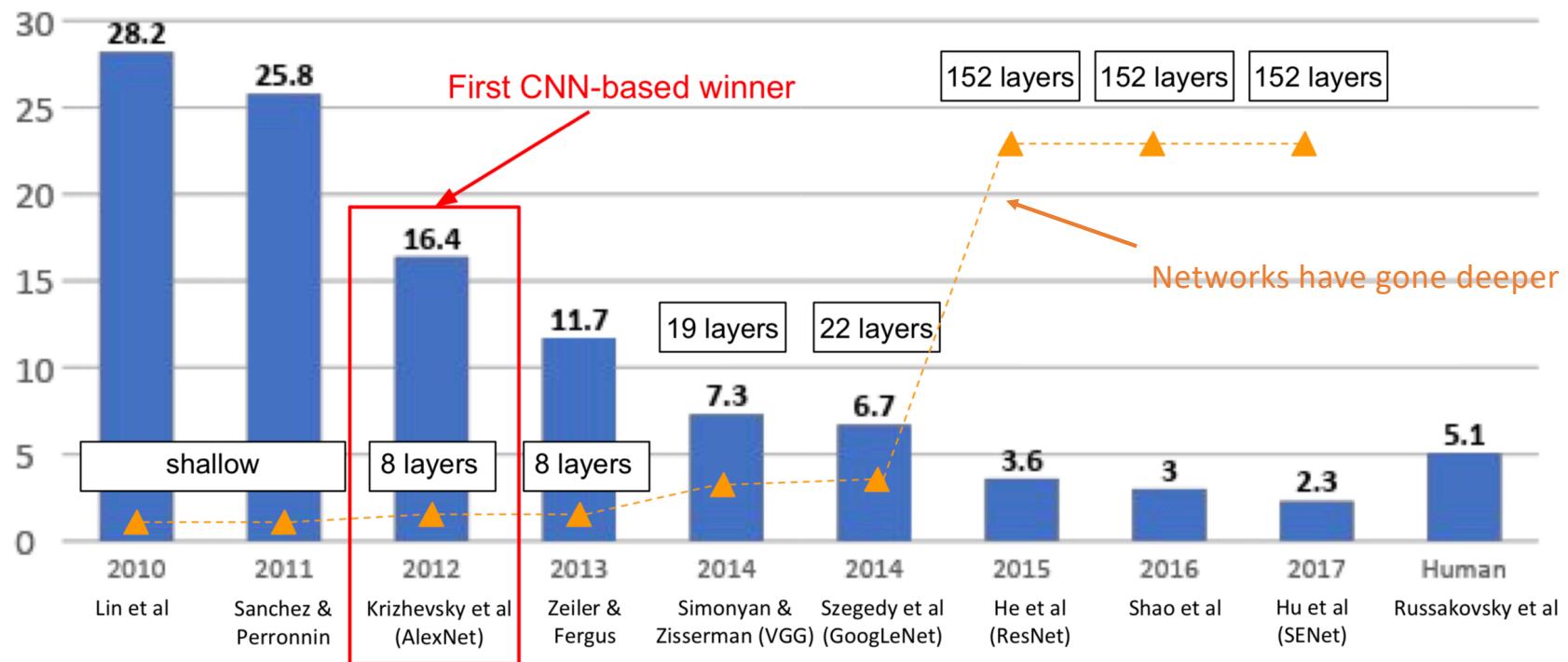
# Alexnet architecture details

- ReLU activation functions (first to use in CNNs)
- Training on multiple GPUs
  - Single GTX 580 GPU has only 3GB
  - Net spread across two GPUs; Puts half of the kernels (or neurons) on each GPU
  - Inter-GPU communication only in certain layers
- Local response normalization (LRN)
  - Normalize ReLU output (not used in later networks)
- Overlapping pooling
- 8 layers with weights
  - First five are convolutional
  - Remaining three are fully-connected
  - Output of the last fully-connected layer is fed to a 1000-way softmax

# Alexnet Training

- 60 Million parameters to learn
- Techniques employed to reduce overfitting
  - Data augmentation (done on CPU)
    - Artificially enlarge the dataset using label-preserving transformations
    - Random cropping (224x224 from 256x256) and horizontal reflections
    - Altering intensities of RGB channels in training images
  - Dropout
    - In first two fully connected layers with dropout probability 0.5
- Batch size: 128
- SGD with 0.9 momentum
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 0.0005
- 7 CNN ensemble: 18.2% -> 15.4%

# Imagenet Large Scale Visual Recognition (ILSVRC) Challenge Winners



# VGG (2014)

ConvNet Configuration						VGG16	VGG19
A	A-LRN	B	C	D	E		
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers		
input (224 × 224 RGB image)							
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64		
maxpool							
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128		
maxpool							
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>		
maxpool							
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>		
maxpool							
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>		
maxpool							
FC-4096							
FC-4096							
FC-1000							
soft-max							

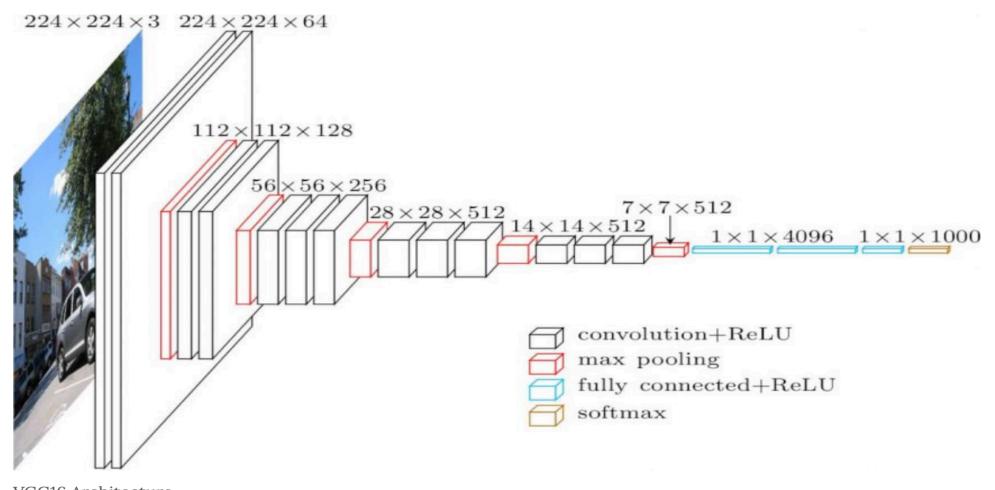
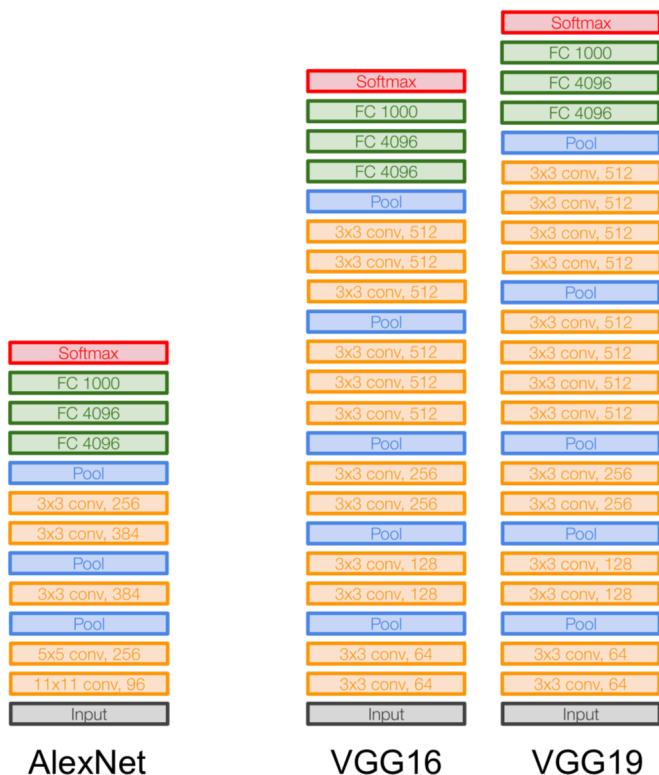


Table 2: **Number of parameters** (in millions).

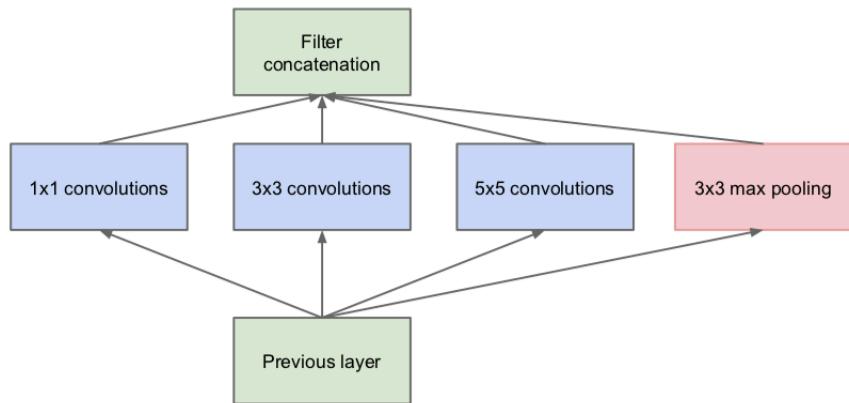
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

# VGG Architecture

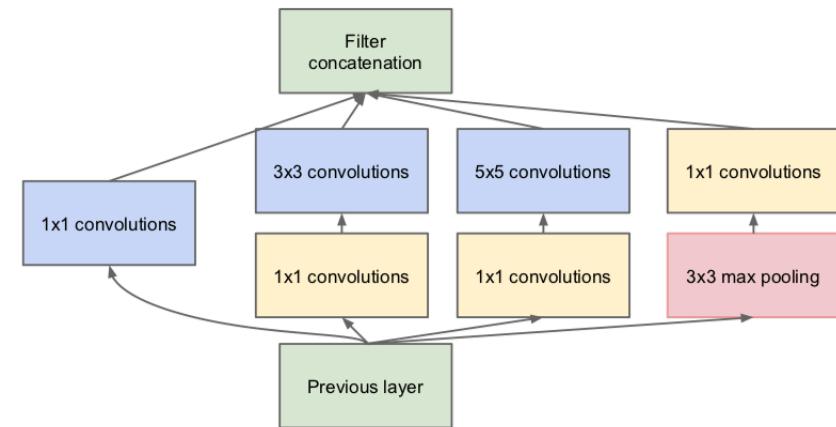


- Smaller filters (3x3, stride 1) compared to 11x11 and 5x5 in Alexnet
- Deeper nets: more layers compared to Alexnet (8 vs 16 or 19)
- Why smaller filters ?
  - Receptive field of 3 3x3 stacked conv. layers is same as a single 7x7 conv layer
  - More non-linearities with stack of smaller conv layers => makes decision function more discriminative
  - Lesser number of parameters:  $3 * (3^2 C^2)$  vs.  $7^2 C^2$  for C channels per layer (55% less)

# GoogleNet (2014)



(a) Inception module, naïve version

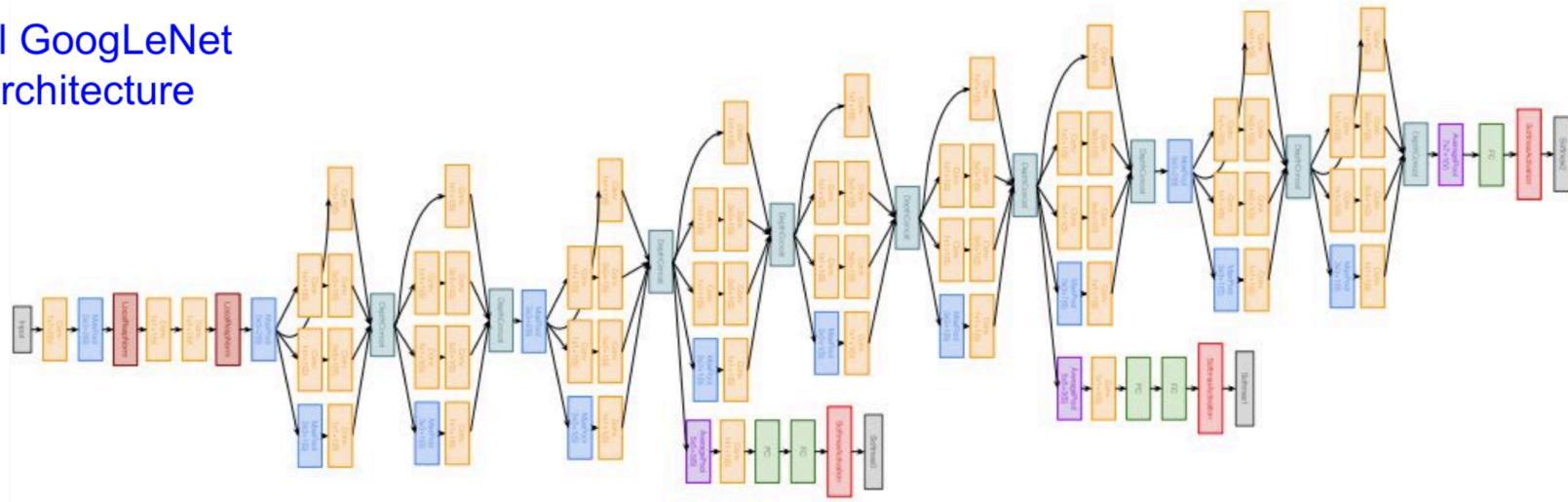


(b) Inception module with dimension reductions

- Inception architecture: Inception modules stacked to create Googlenet
- Dimension reduction module has reduced computational complexity compared to naïve version

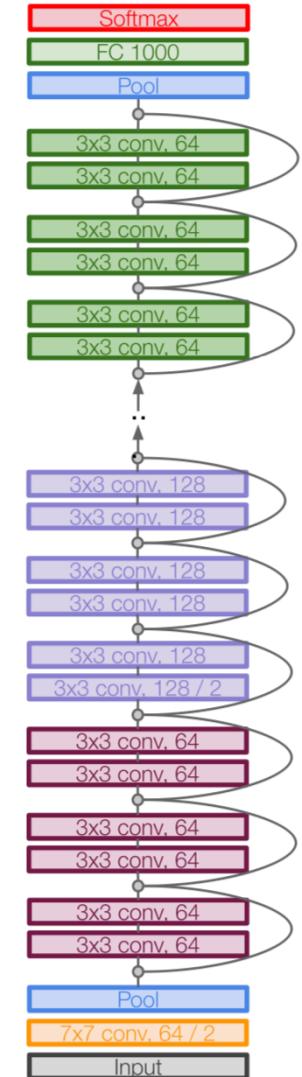
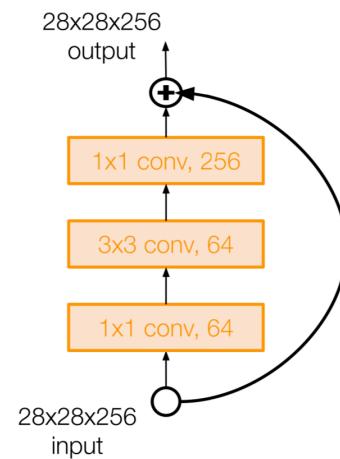
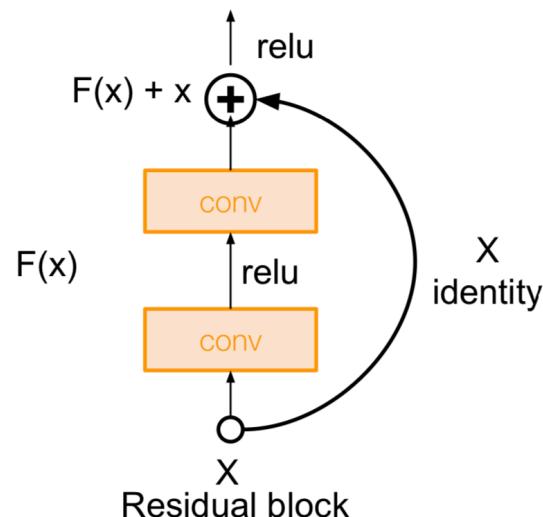
# GoogLeNet(2014)

Full GoogLeNet  
architecture



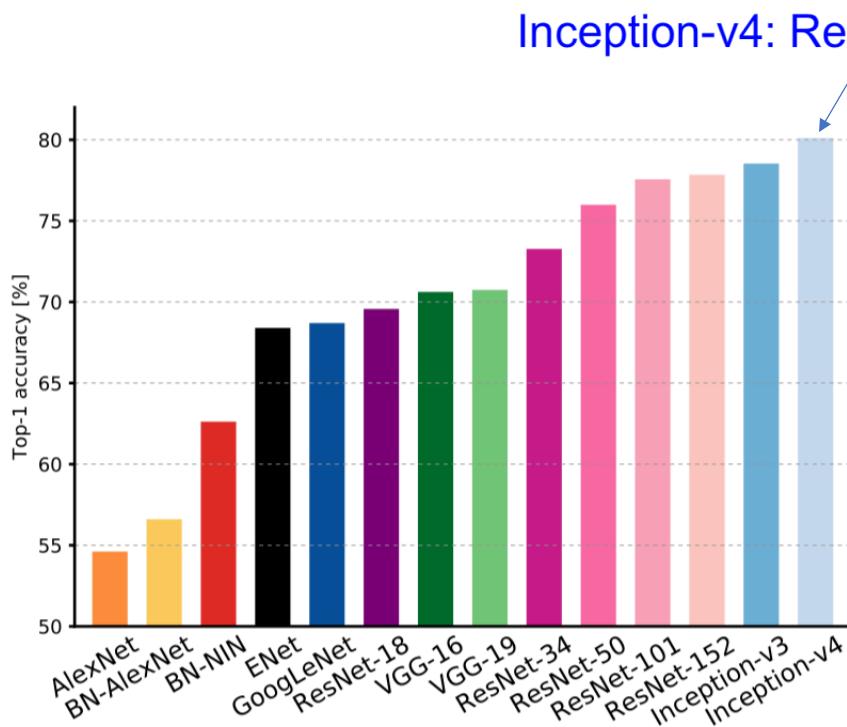
- Deeper network (22 layers)
- Computationally efficient “Inception” module
- Avoids expensive FC layers → uses average pooling
- 12x less params than AlexNet

# ResNet

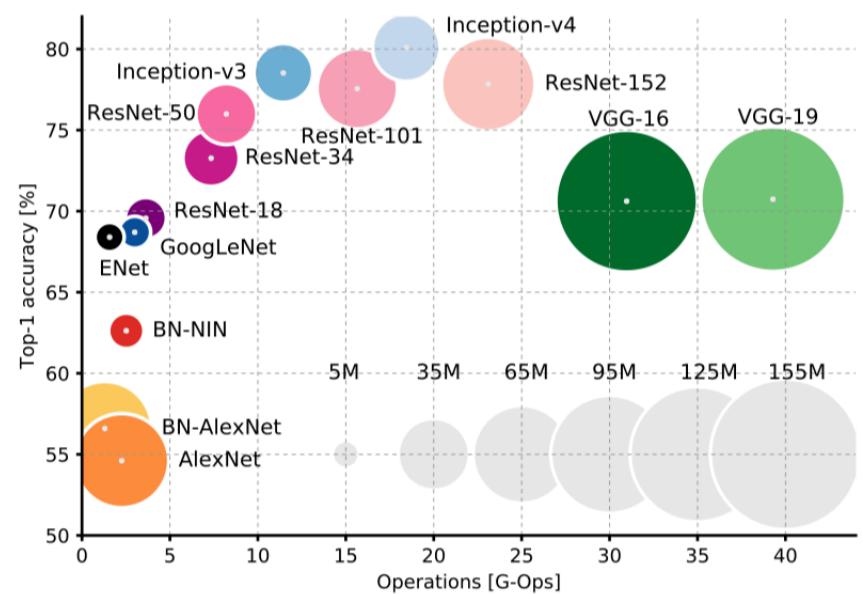


- Stack of residual blocks: each residual block has 2 3x3 conv layers; number of filters is doubled periodically
  - Global average pooling layer after last conv layer
  - Different depths: 34, 50, 101, 152
  - Deeper network (ResNet50+) add 1x1 conv for computational efficiency

# Top1 Accuracy Comparison



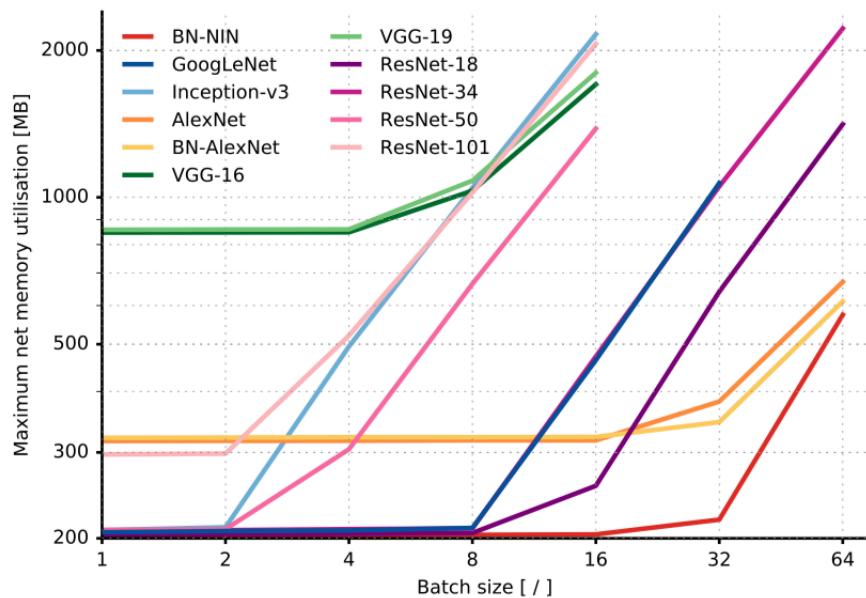
**Top1 vs. network.**



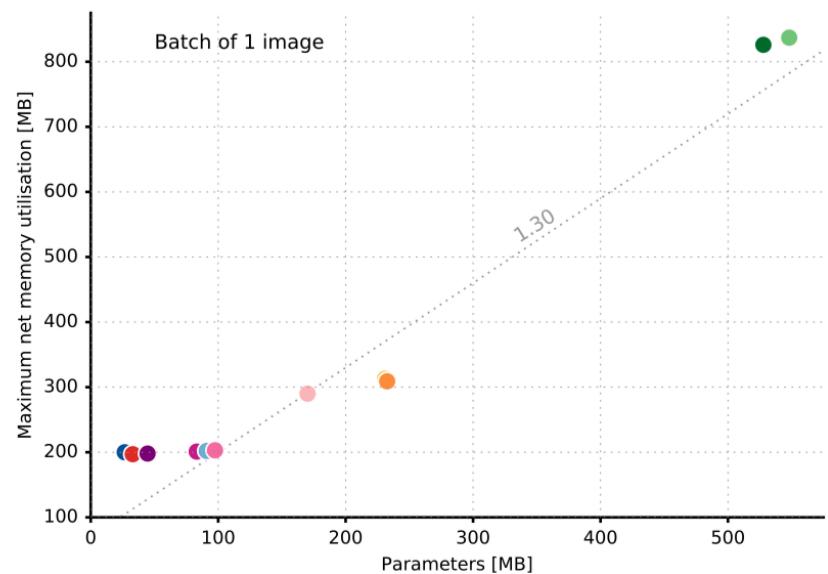
**Top1 vs. operations, size  $\propto$  parameters**

# Memory Consumption

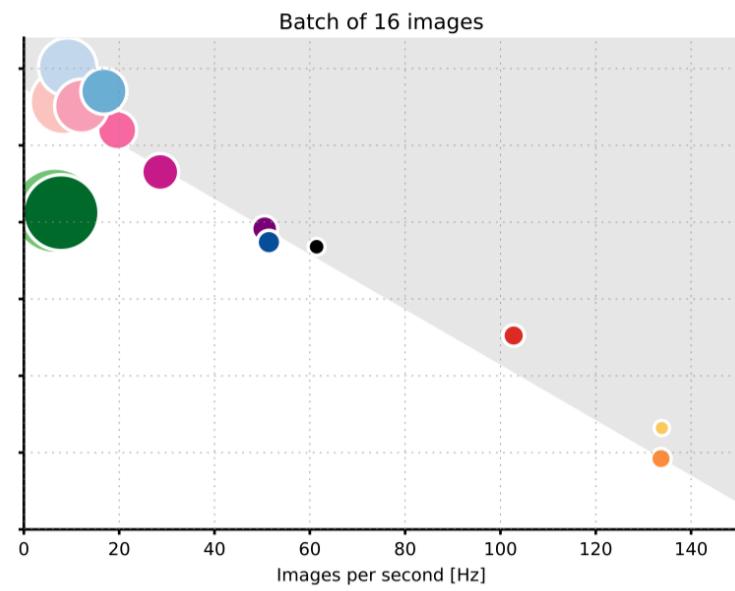
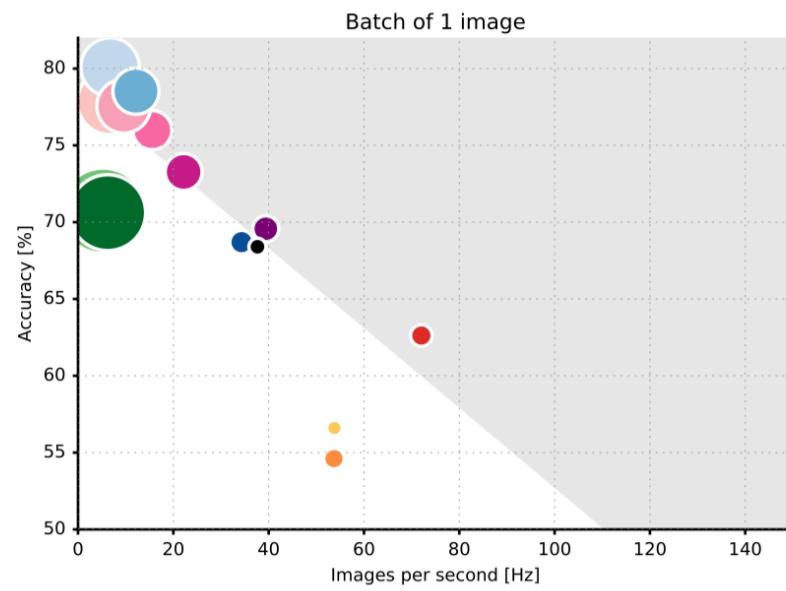
Memory vs. batch size.



Memory vs. parameters count



# Accuracy vs Throughput

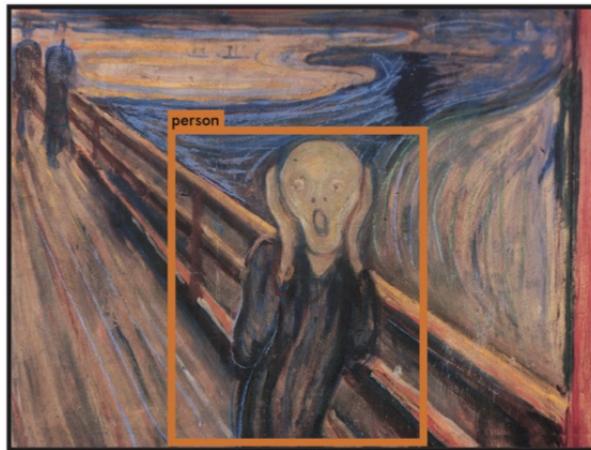


# CNN Applications

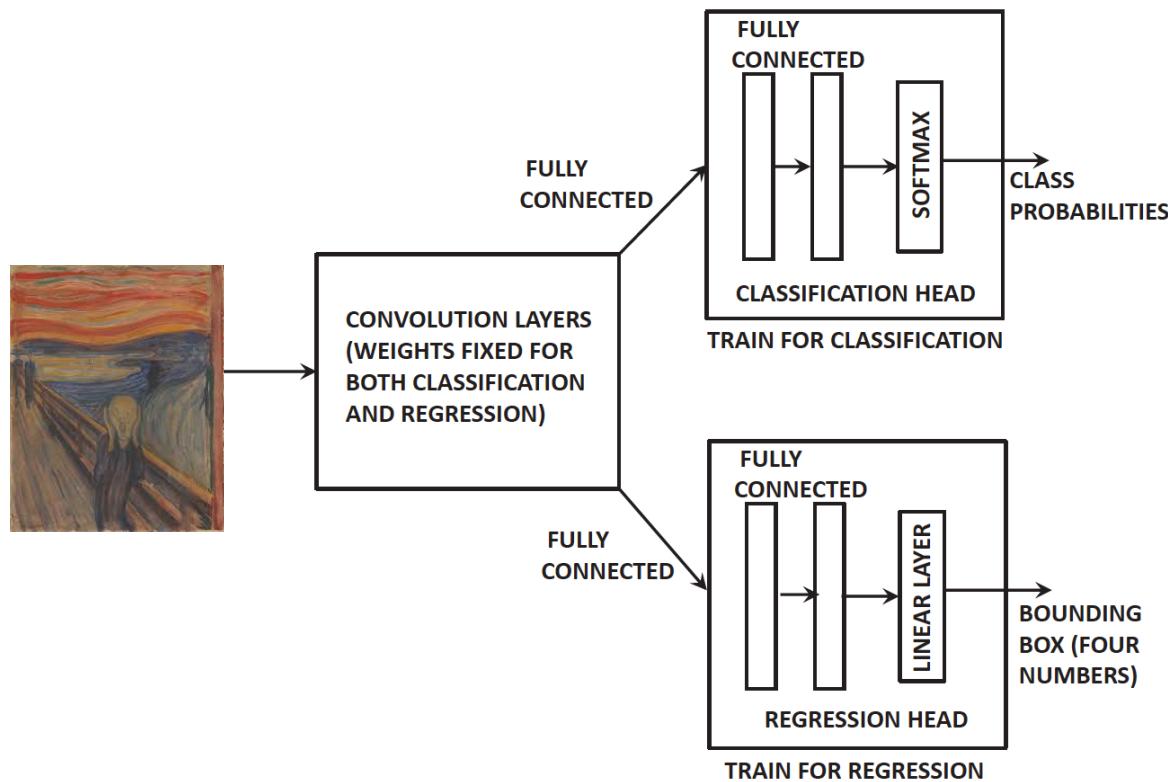
- Image classification
- Object localization
- Object detection

# Object Localization

- Given a fixed set of objects in an image, identify rectangular regions in the image in which the object occurs
- Four numbers to identify a bounding box: coordinates of the top-left corner and two dimensions (length and width)
- Often integrated with classification of objects

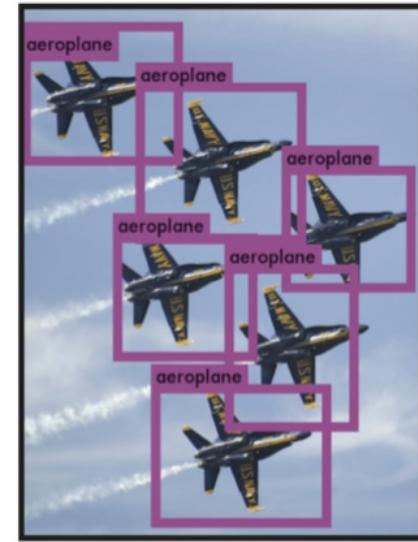
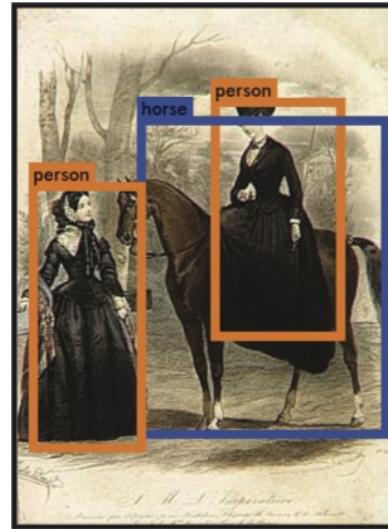
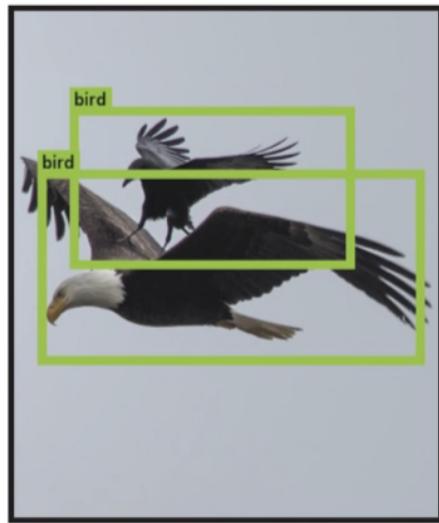


# Object Localization



# Object Detection

- Variable number of objects of different classes in an image
- Identify all the objects and their class



Cannot use object localization architecture as the number of classification and regression heads needed is not known

# Identify region in image containing an object



(a)

(b)

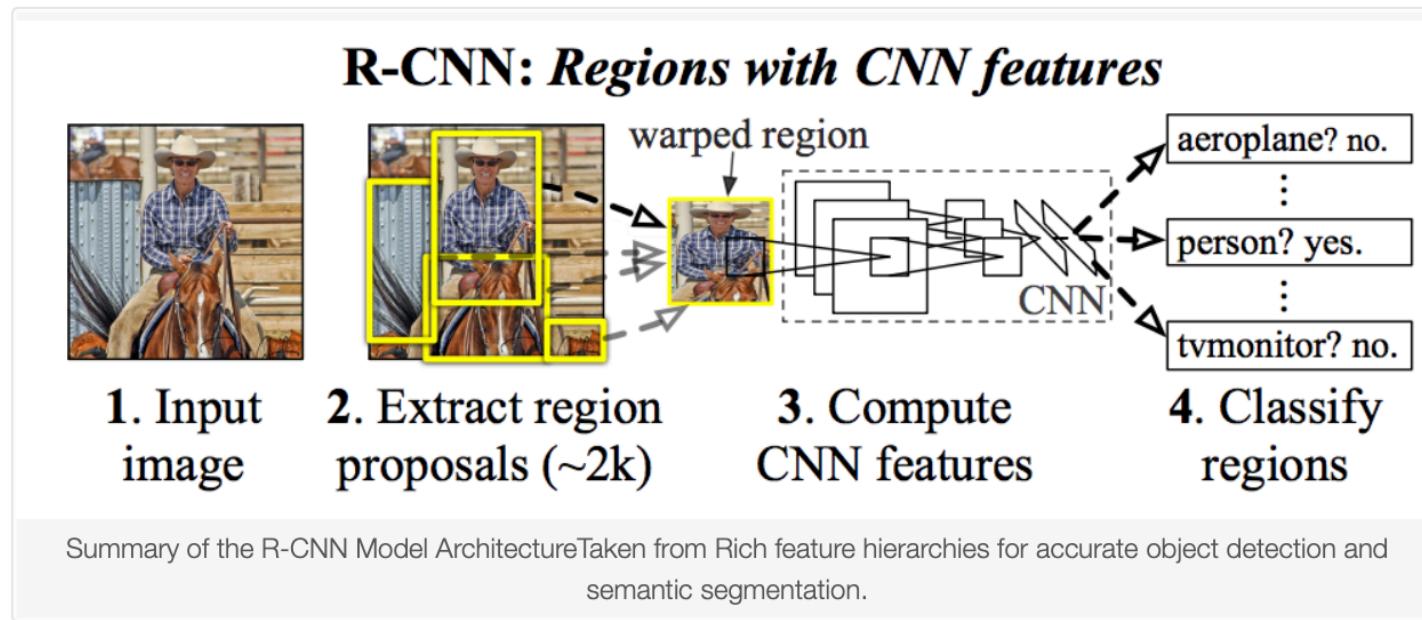


(c)

(d)

Figure 1: There is a high variety of reasons that an image region forms an object. In (b) the cats can be distinguished by colour, not texture. In (c) the chameleon can be distinguished from the surrounding leaves by texture, not colour. In (d) the wheels can be part of the car because they are enclosed, not because they are similar in texture or colour. Therefore, to find objects in a structured way it is necessary to use a variety of diverse strategies. Furthermore, an image is intrinsically hierarchical as there is no single scale for which the complete table, salad bowl, and salad spoon can be found in (a).

# R-CNN (Region Based Convolutional Neural Network)

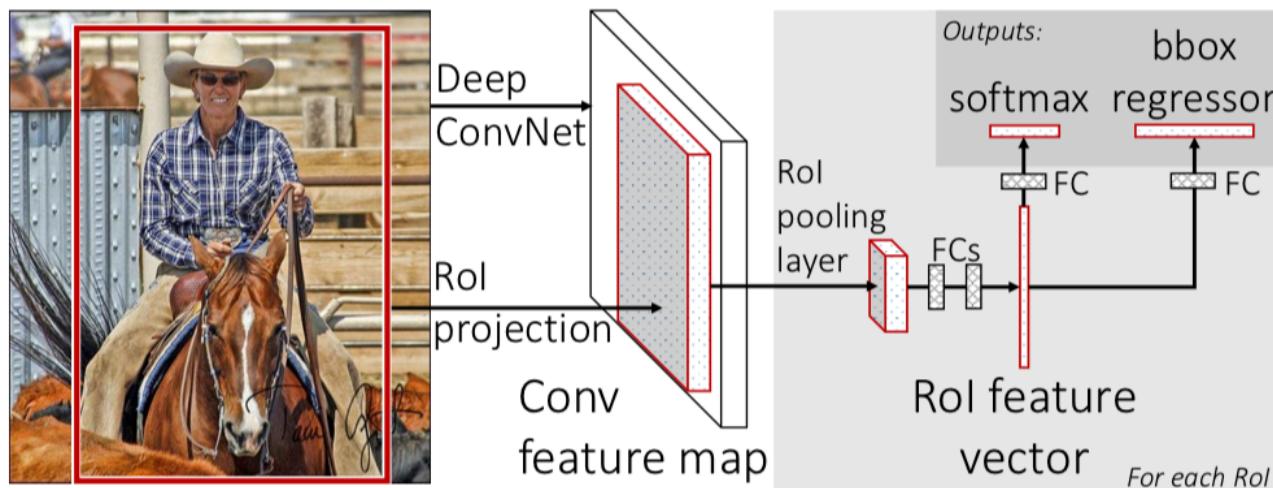


Three main modules:

1. **Region Proposal.** Generate and extract category independent region proposals, e.g. candidate bounding boxes.
2. **Feature Extractor.** Extract feature from each candidate region, e.g. using a deep convolutional neural network.
3. **Classifier.** Classify features as one of the known class, e.g. linear SVM classifier model.

# Fast R-CNN

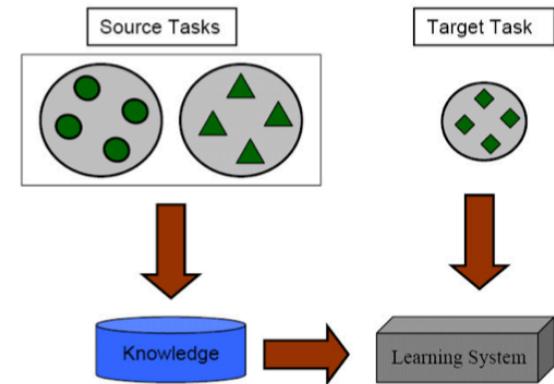
- R-CNN is slow
  - CNN-based feature extraction pass on each of the candidate regions, no sharing of computation
  - ~2,000 proposed regions per image at test-time



# Transfer Learning

- Building good machine learning models require lot of training data
  - To capture robust representation of unknown input distribution
- Small training jobs are common and labeled data is scarce in many domains
  - In commercial VR service (Bhatta et al. 2019), average number of images submitted is 250 and average number of classes are 5; ~50 images per class
- Can we leverage knowledge learnt from related tasks for target task ?
- Transfer learning is a class of techniques to reuse knowledge gathered from “source” tasks (with sufficiently rich set of labeled data) for a “target task (with few labeled data)

Learning Process of Transfer Learning



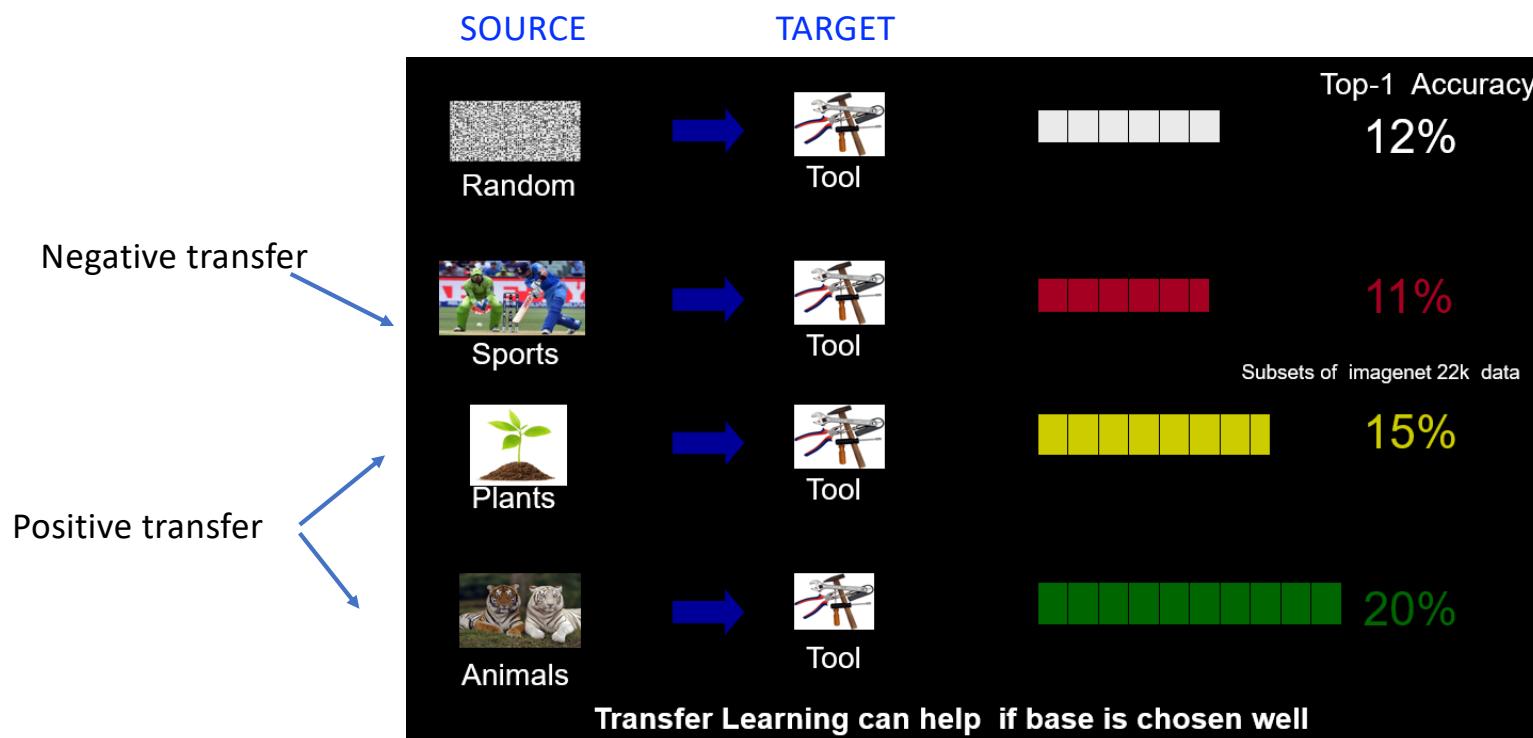
# Transfer Learning Approaches

- *Common intuition*: Networks which have learned compact representations of a "source" task, can reuse these representations to achieve higher performance on a related "target" task.
- *Instance based* approaches attempt to identify appropriate data used in the source task to supplement target task training
- *Feature representation based* approaches attempt to leverage source task weight matrices
  - Trained weights in source network have captured a representation of the input that can be transferred by fine-tuning the weights or retraining the final dense layer of the network on the new task.

# Improving Transferability in Transfer Learning

- **Selection of source model:** What is the best “source” task/model to transfer knowledge for a “target” task ?
  - How to measure “similarity” between source and target task ?
    - Develop similarity measures between source and target datasets in feature space
  - Does size always matter ?
    - Will a source model trained on huge datasets will always outperform source models trained on smaller but more related datasets ?
  - Improper choice of a base dataset/model for a target could result in degraded performance compared to not using transfer learning (**negative transfer**)
- **Degree of finetuning**
  - Which layers to finetune and which to freeze ?
  - What should be the learning rate of layers to be finetuned ?
    - Higher learning rate → loose information from source task

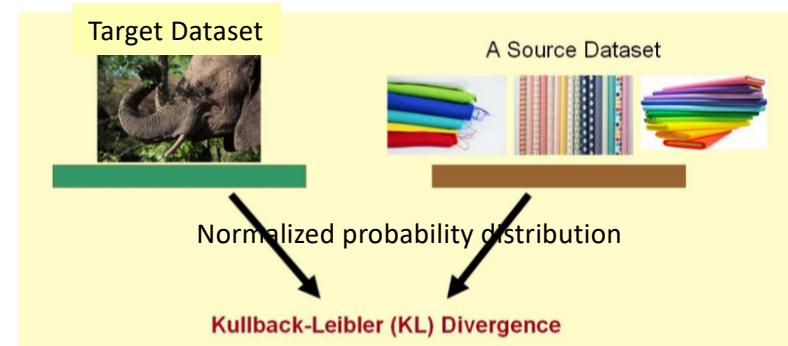
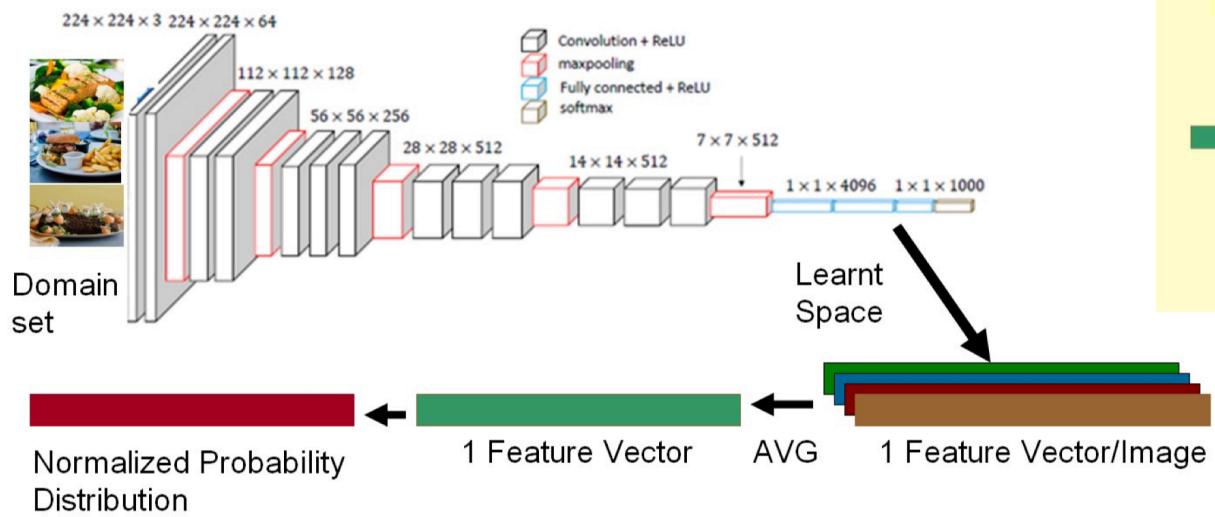
# Good and bad sources



# Efficient Selection of Source Models

- Training consumes valuable time / compute resources, we do not want to do a lengthy optimization process or train many candidate models per task
- Predict to Learn (P2L) approach (Bhatta et al 2019)
  - An efficient method to estimate the appropriateness of a previously trained model for use with a new learning task
  - Precludes the exhaustive approach of fine-tuning all existing source models to search for the best fit.
  - P2L only requires a single forward pass of the target data set through a single reference model to identify the most likely candidate for fine-tuning.
- P2L accounts for two attributes:
  - **Similarity between source and target dataset:** source dataset which is similar to a target dataset in feature space is most likely to have a higher degree of transferability and hence a better candidate for fine-tuning
  - **Size of source dataset**

# P2L Approach: dataset similarity in feature space



# P2L Performance

Target Dataset $t_i$	P2L Picked Best Dataset ?	Random Dataset Selection (OP-RDS)/RDS	No Transfer Learning (OP-A $\phi$ )/A $\phi$
Fruit	Yes	0.59 OP = Accuracy using P2L	1.00
Fabric	Yes	0.59 OP = Accuracy using P2L	0.67
Building	Yes	0.36	0.63
Music	Yes	0.25	0.53
Nature	Yes	0.21	0.42
Food	Yes	0.37	0.31
Tool	Yes	0.12	0.25
Furniture	Yes	0.22	0.22
Person	Yes	0.13	0.25

OP = Accuracy using P2L

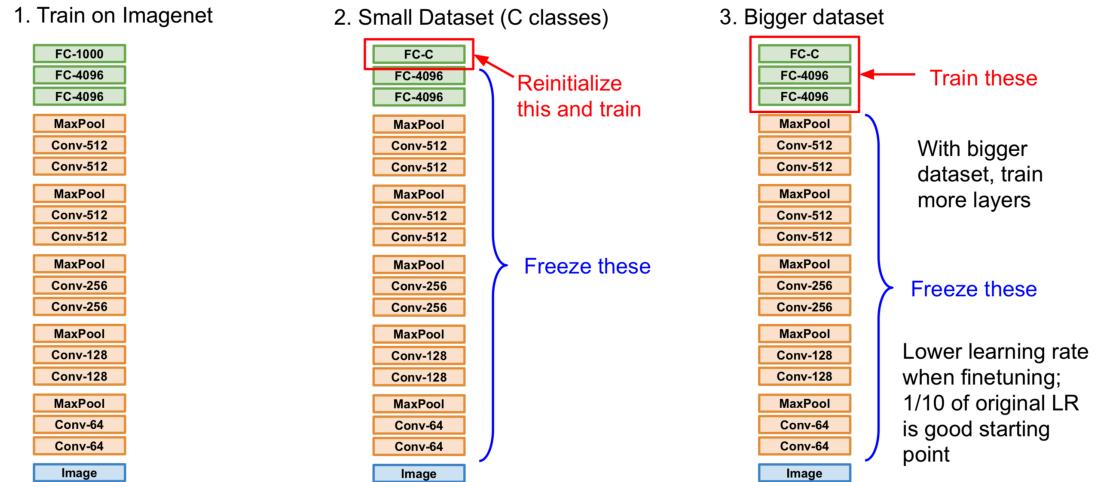
RDS = Avg accuracy of randomly picked source dataset

A $\phi$  = Accuracy of  $\phi$

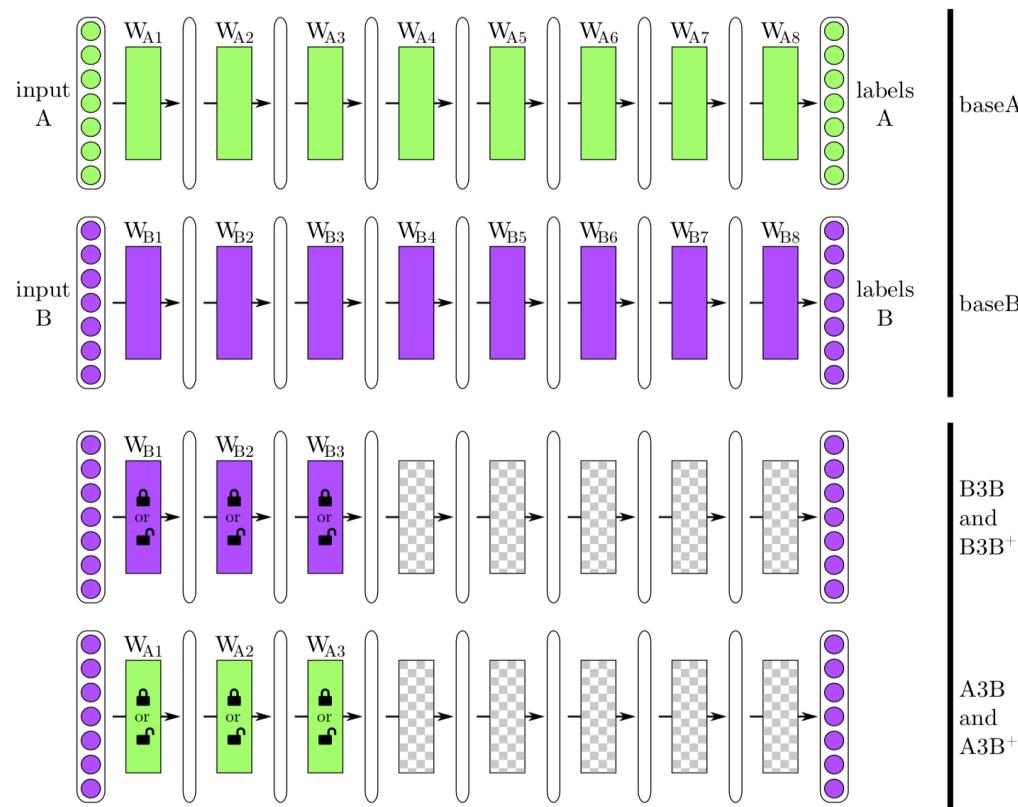
# Transfer Learning: Basic Finetuning

- Take almost any deep network pre-trained on a large dataset of your choice
- Model zoo of pretrained models
  - TensorFlow: <https://github.com/tensorflow/models>
  - PyTorch: <https://github.com/pytorch/vision>
- Replace the last (classification) layer with a randomly initialized one
- Train only the new layer's weights, using the "frozen" embedding

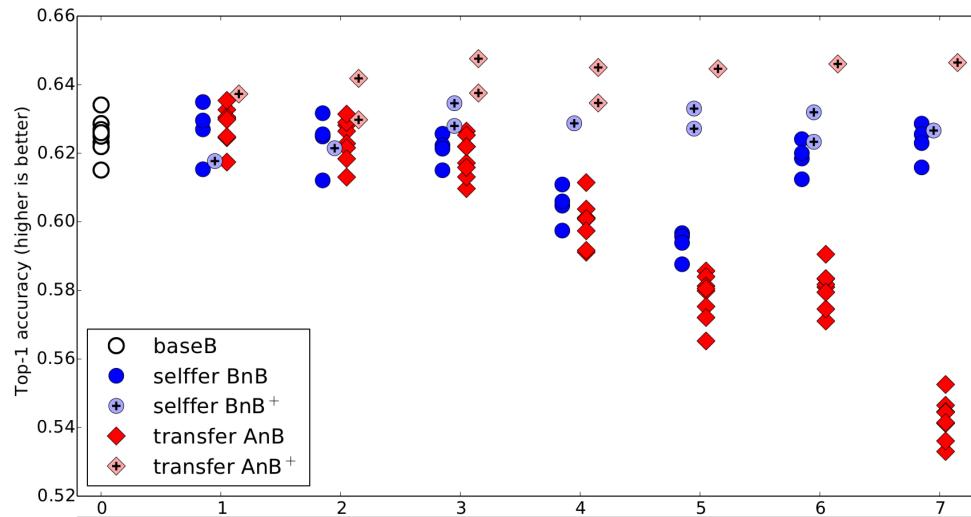
- This baseline method works well in many settings...
- .... Can we improve on it?



# Transferability

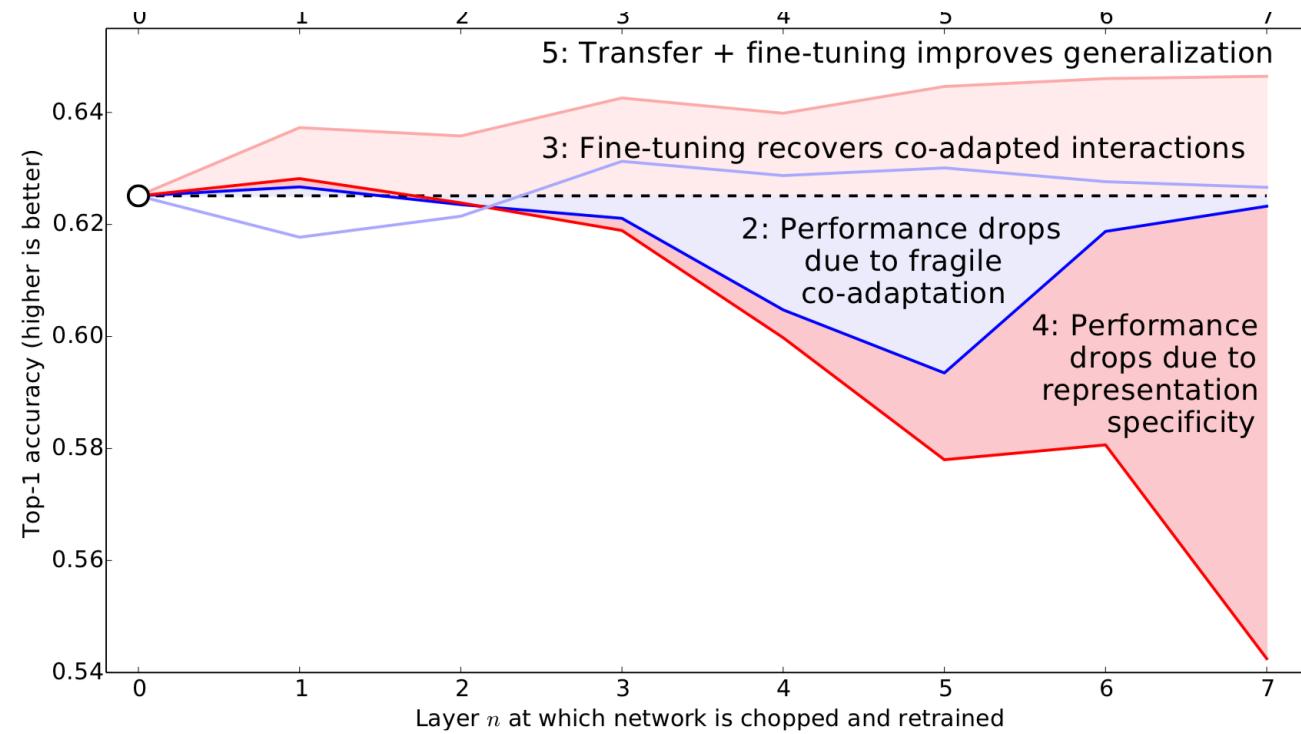


# Transferability



Each marker in the figure represents the average accuracy over the validation set for a trained network. The white circles above  $n = 0$  represent the accuracy of baseB. There are eight points, because we tested on four separate random A/B splits. Each dark blue dot represents a BnB network. Light blue points represent BnB<sup>+</sup> networks, or fine-tuned versions of BnB. Dark red diamonds are AnB networks, and light red diamonds are the fine-tuned AnB<sup>+</sup> versions.

# Transferability



Lines connecting the means of each treatment.

# Degree of Finetuning

Table 1: Transfer Learning Accuracy with varying  $L1$

Target	Source	LL-0.01	LL-0.1	% Gain
fabric	garment	<b>13.09%</b>	11.33%	15.47%
tool	weapon	14.54%	<b>14.78%</b>	1.63%
oxford	plants	<b>91.06%</b>	73.17%	24.44%
food	fruit	<b>5.71%</b>	5.07%	12.52%
fungus	plant	<b>13.12%</b>	5.80%	127.79%
person	food	<b>4.49%</b>	2.81%	59.75%
fruit	garment	9.30%	<b>10.50%</b>	12.92%
music	plant	<b>15.37%</b>	9.47%	62.22%

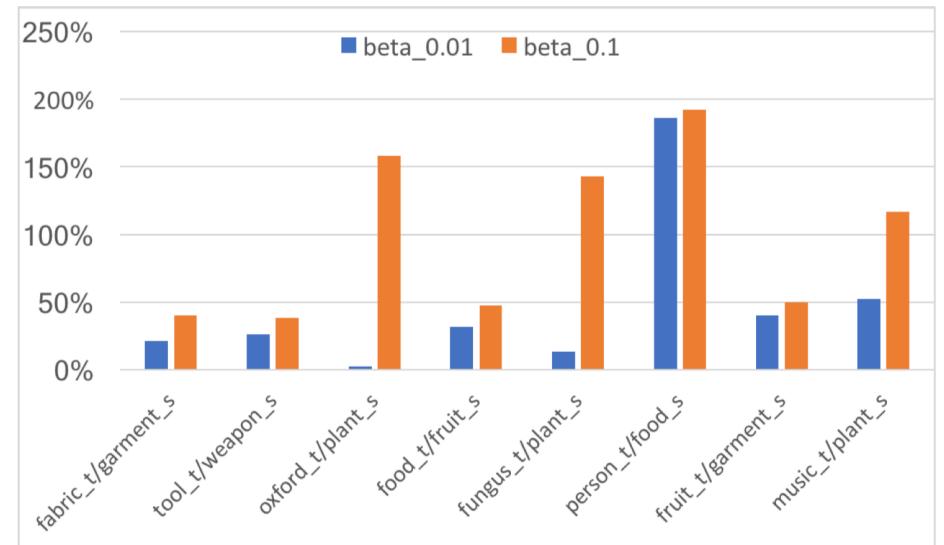


Fig. 8: Range of variation in accuracy with varying  $IL$

# IBM Visual Recognition Service

- [IBM Watson Visual Recognition Service](#)
- Cloud based service
- Supports both image classification and object detection
  - Use of pre-trained models or developing custom models
- Video: [Visual Recognition in Watson Studio](#)
- [Custom object detection](#)
- Visual recognition with CoreML
  - Source code and instructions for yourself to try:  
<https://github.com/bourdakos1/visual-recognition-with-coreml>
  - Tutorial <http://ibm.biz/coremlworkshop>

---

# Seminar Reading List

- **Convolutional Neural Networks**

- Yann LeCun et al. [Gradient-Based Learning Applied to Document Recognition](#) 1998
- Alex Krizhevsky et al. [ImageNet Classification with Deep Convolutional Neural Networks](#) 2012
- Karen Simonyan et al. [VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION](#) 2015
- Christian Szegedy et al. [Going Deeper with Convolutions](#) 2014

- **Object Detection**

- Ross Girshick et al. [Rich feature hierarchies for accurate object detection and semantic segmentation](#) 2015
- Ross Girshick. [Fast R-CNN](#) 2015

- **Transfer Learning in Vision**

- Bhatta et al. [P2L: Predicting Transfer Learning for Images and Semantic Relations](#) 2019

# Suggested Reading

- Adrian Rosebrock. [ImageNet: VGGNet, ResNet, Inception, and Xception with Keras](#) 2017