Visual AI

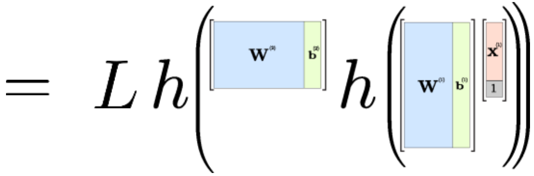
CPSC 532R/533R – 2019/2020 Term 2

Lecture 4. Advanced architectures and sparse 2D keypoints

Helge Rhodin

Recap: Automatic differentiation and backpropagation

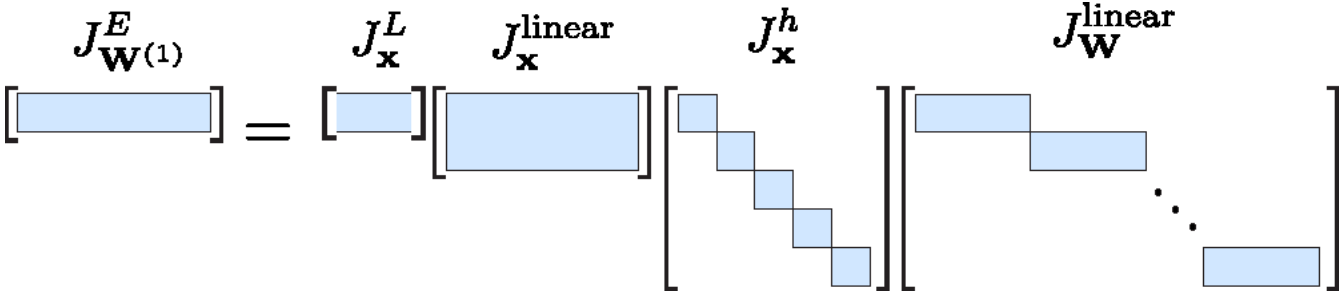
Forward pass $$L(h(\text{linear}(h(\text{linear}(x, W^{(1)})), W^{(2)})))$$



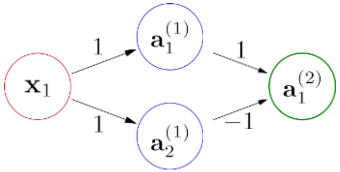
$$\mathbf{J}_x^f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Jocobian matrix

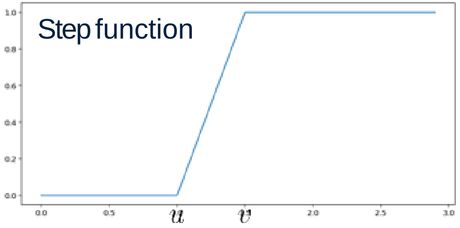
Backwards pass to $W^{(1)}$

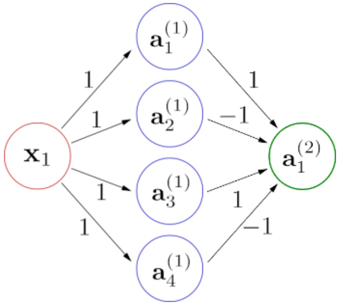


NN theory and universal approximation

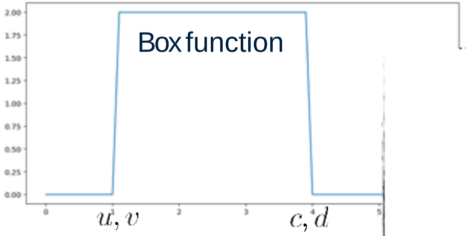


$$\mathbf{a}_1^{(1)} = \texttt{relu}(x - u)$$
$$\mathbf{a}_2^{(1)} = \texttt{relu}(x - v)$$
$$\mathbf{a}_1^{(2)} = \mathbf{a}_1^{(1)} - \mathbf{a}_2^{(1)}$$

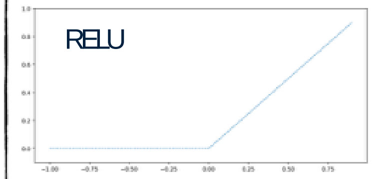


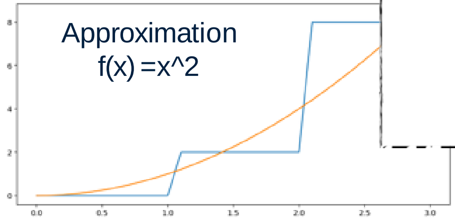


$$\mathbf{a}_1^{(1)} = \texttt{relu}(x - u)$$
$$\mathbf{a}_2^{(1)} = \texttt{relu}(x - v)$$
$$\mathbf{a}_3^{(1)} = \texttt{relu}(x - c)$$
$$\mathbf{a}_4^{(1)} = \texttt{relu}(x - d)$$
$$\mathbf{a}_1^{(2)} = \mathbf{a}_1^{(1)} - \mathbf{a}_2^{(1)} - (\mathbf{a}_3^{(1)} - \mathbf{a}_4^{(1)})$$

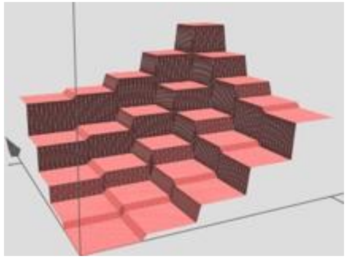


⋮





Approximation in 2D



[neuralnetworksanddeeplearning.com]

Mathematical prove in [Hornik et al., 1989; Cybenko, 1989]

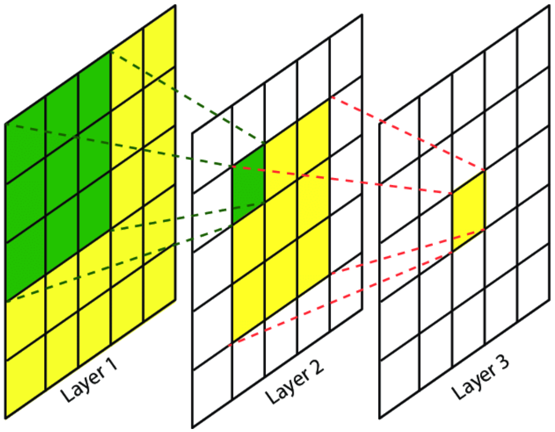
What is the benefit of **deep** neural networks?

In principle a fully-connected network is sufficient

- universal approximator
- but hard to train!

Empirical observation (for image processing)

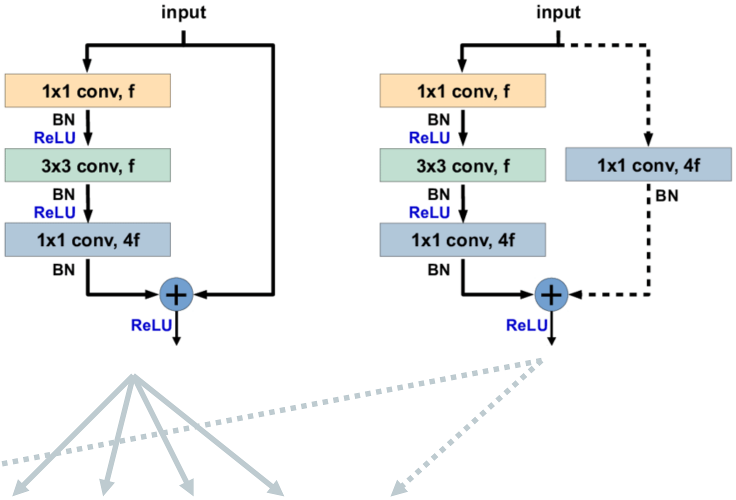
- convolution and pooling operations act as a strong prior
  - locality
  - translational invariance
- a deep network increases the receptive field
  - such large context helps
- many simple operations work better than a monolithic one
  - separable conv., group conv., 3x3 instead of 5x5, …
    (this lecture)



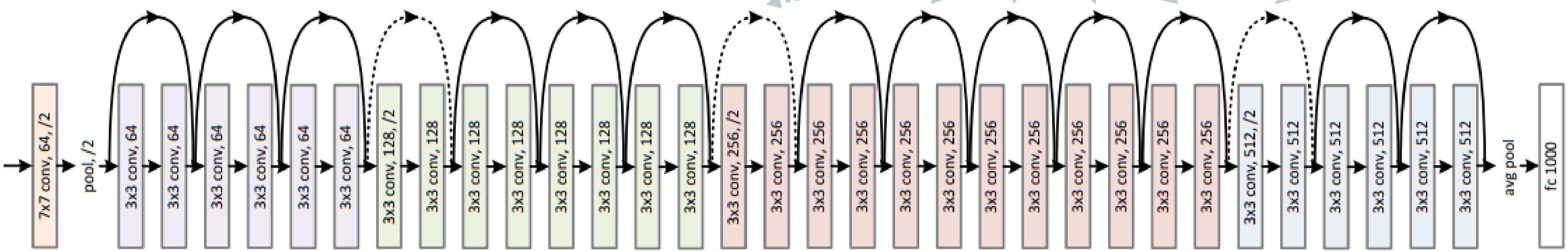
ResNet details

What if number of channels changes?

- apply a linear transformation to match the size
  - a projection on a linear subspace,
    related to principal component analysis (PCA)



ResNet 32 (32 layers)



GoogleLeNet (Inception Net V1)

[Szegedy et al., 2014]

even deeper network with **computational efficiency**

— 22 layers

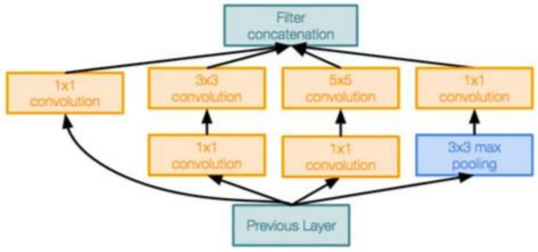
— Efficient "Inception" module

— No FC layers

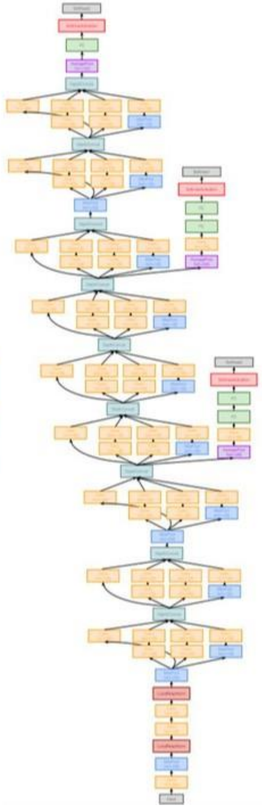
—Only 5 million parameters!

(12x less than AlexNet!)

— Better performance (@6.7 top 5 error)

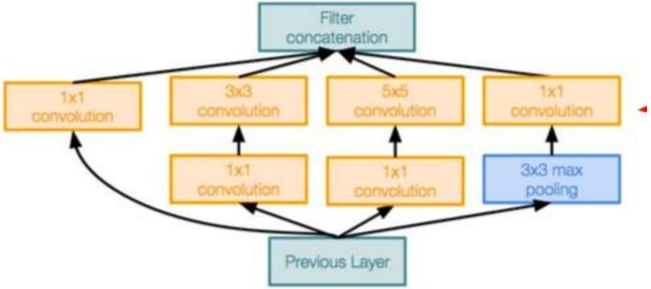


Inception module



GoogleLeNet: Inception Module

Idea: design good local topology ("network within network") and then stack these modules



Inception module

GoogleLeNet: Inception Module

[Szegedy et al., 2014]

Idea: design good local topology ("network within network") and then stack these modules

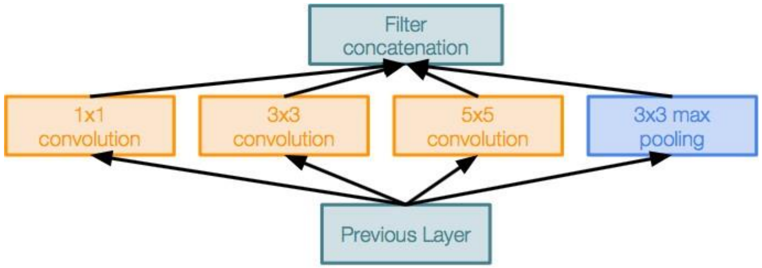
What's the problem?

Apply **parallel filter operations** on the input from previous layer

—Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)

— Pooling operation (3x3)

Concatenate all filter outputs together at output depth-wise



Naive Inception module

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

GoogleLeNet: Inception Module

[Szegedy et al., 2014]

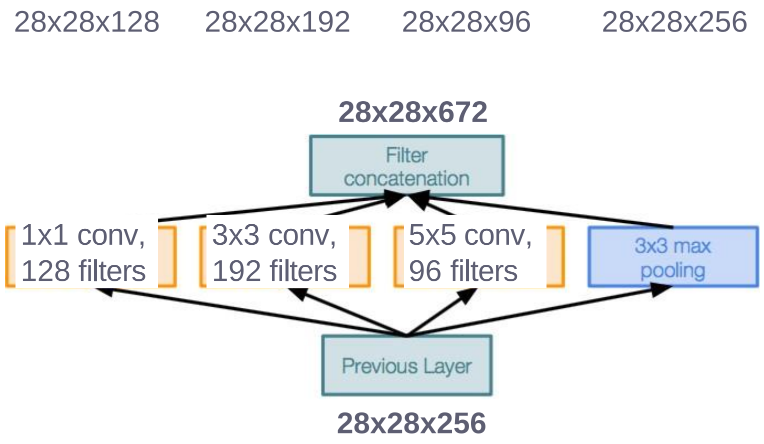
Idea: design good local topology ("network within network") and then stack these modules

Apply **parallel filter operations** on the input from previous layer

—Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)

— Pooling operation (3x3)

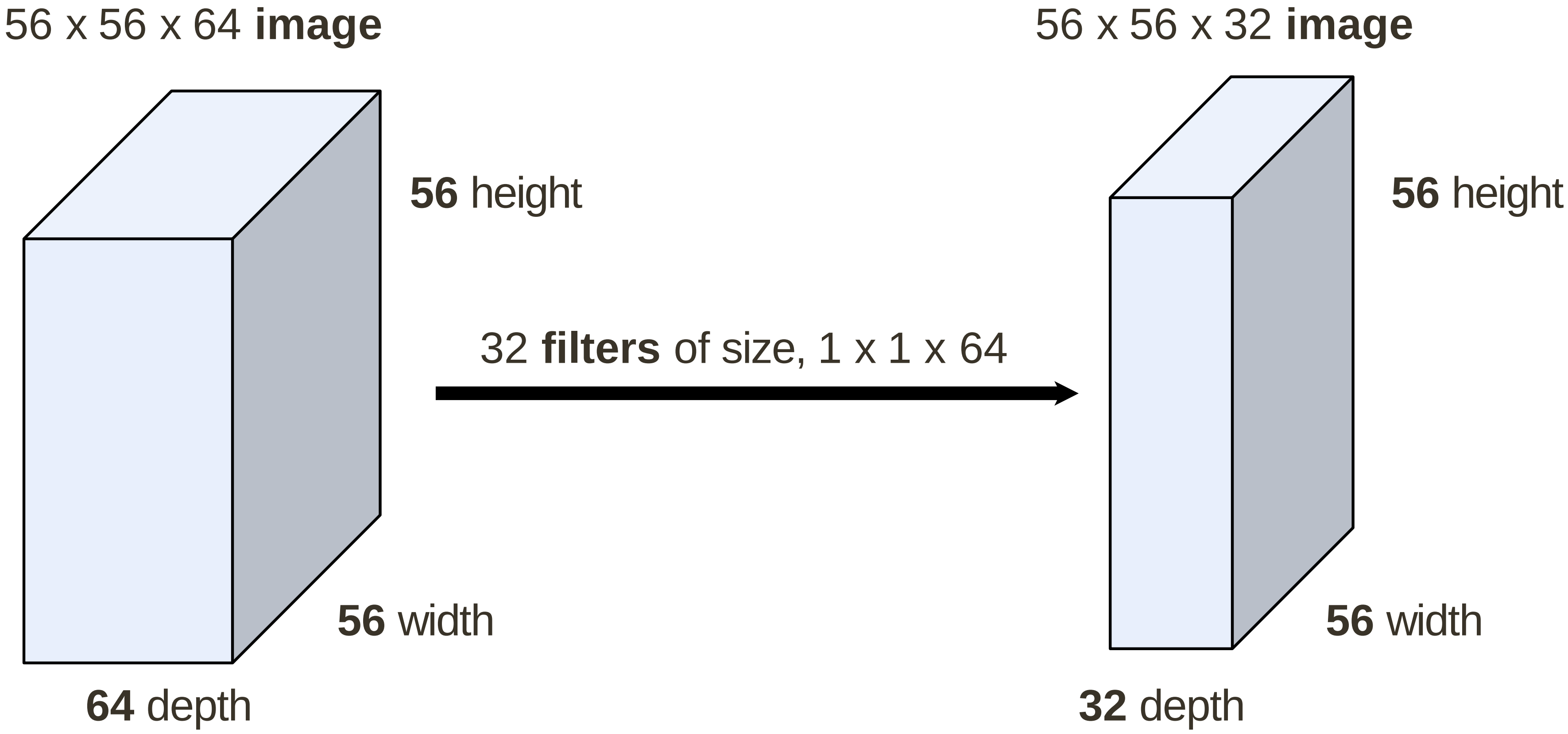
Concatenate all filter outputs together at output depth-wise



Naive Inception module

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer: **1x1 convolutions**

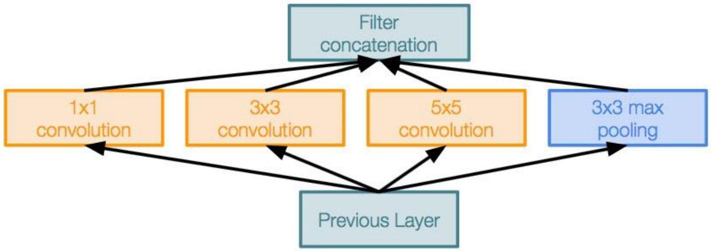


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

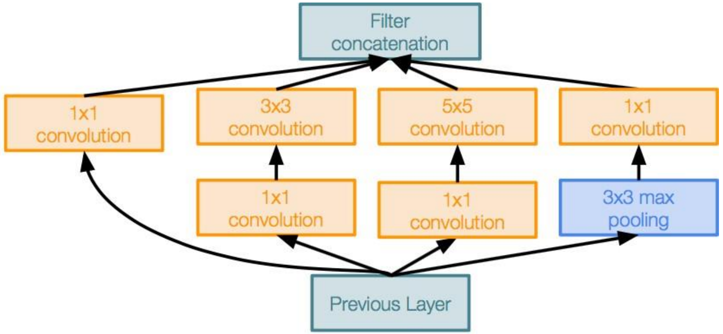
GoogleLeNet: Inception Module

Idea: design good local topology (“network within network”) and then stack these modules

1x1 “bottleneck” layers



Naive Inception module



Inception module with dimension reduction

saves approximately 60% of computations

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

GoogleLeNet

[Szegedy et al., 2014
]

even deeper network with **computational efficiency**

— 22 layers

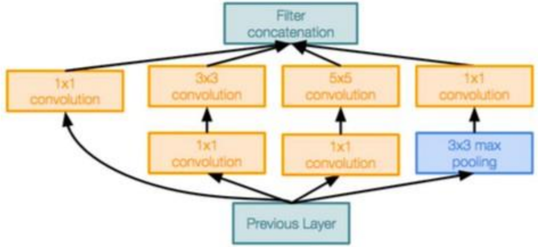
— Efficient “Inception” module

— No FC layers

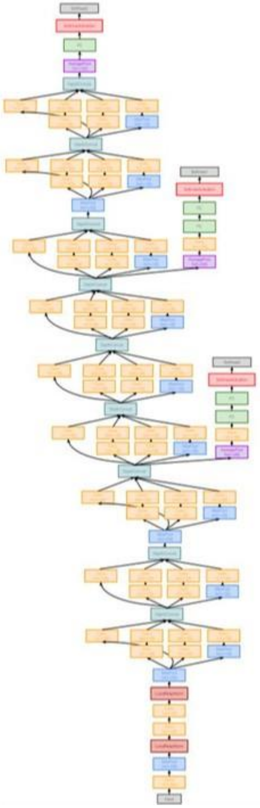
—Only 5 million parameters!

(12x less than AlexNet!)

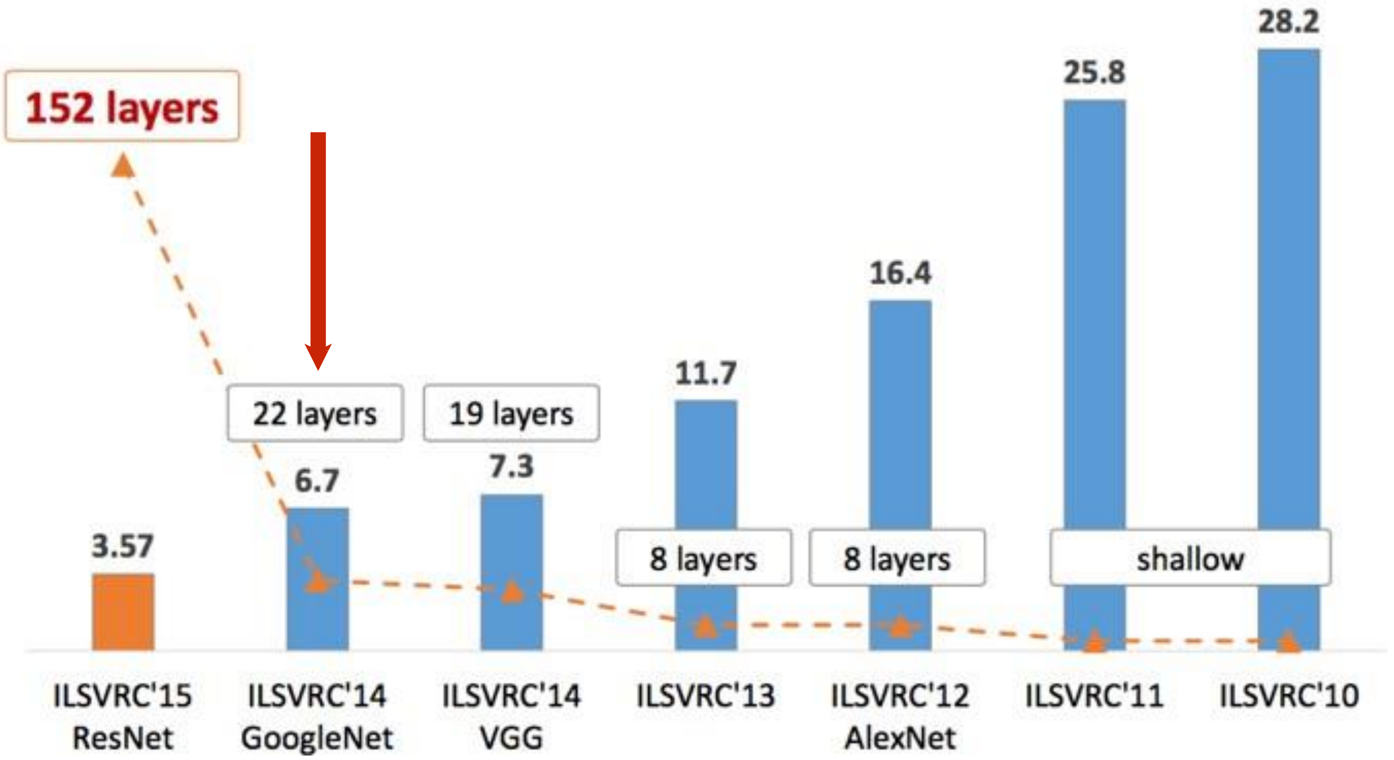
— Better performance (@6.7 top 5 error)



Inception module



ILSVRC winner 2012

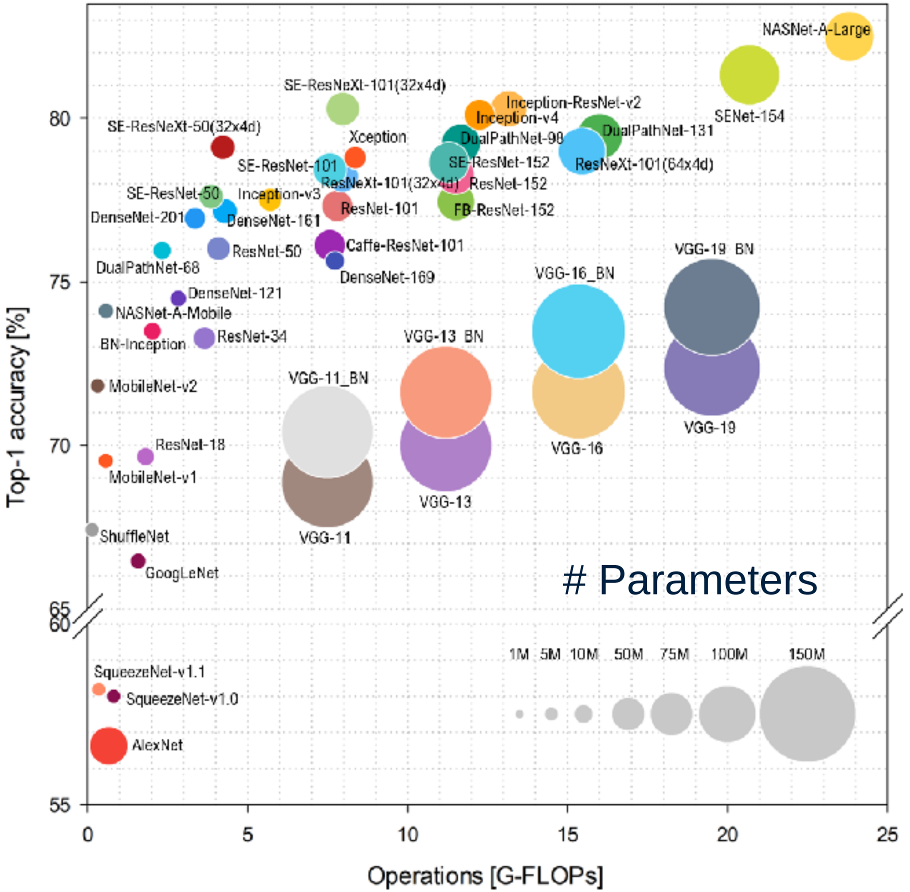


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Network comparison

The goal is to balance

- accuracy (maximize)
- number of parameters (minimize)
- number of operations (minimize)
- efficiency of operations (maximize)
    - cache efficiency
    - parallel execution
    - precision, e.g. float, float16, binary,…



History of Inception Networks

Inception V1 GoogleNet

- Network in network approach

Inception V2

- Use of batch normalization

  [Batch normalization: Accelerating deep network training by …]

Inception V3

- Factorizations

  [Rethinking the inception architecture for computer vision]

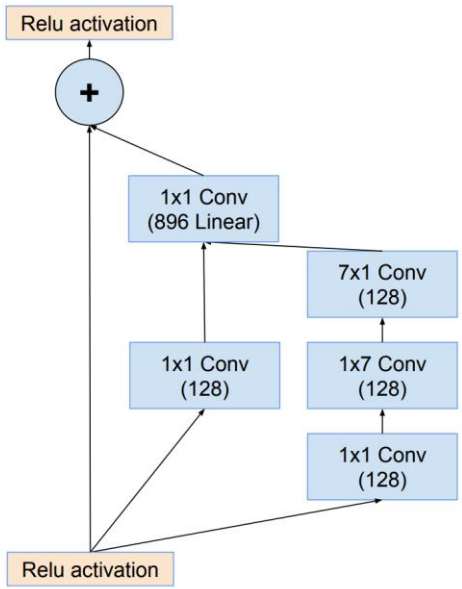
Inception V4

- Tuning of filters

  [Inception-v4, Inception-ResNet and the Impact of …]

Inception-ResNet

- Skip connections instead of concatenations



Inception ResNet block example

Separable Convolutions

Idea:

Separate a single convolution operation into a sequence of simpler operations

- e.g., 7x7 convolution into
  - 1x7 and 7x1
- reduction of parameters
  - e.g., 14 vs. 49 for 7x7 conv.

$$\underbrace{\begin{bmatrix} 6 & 3 & 3 & 6 \\ 4 & 2 & 1 & 8 \\ 2 & 5 & 6 & 4 \\ 9 & 3 & 2 & 1 \end{bmatrix}}_{f} \otimes \underbrace{\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}}_{k} \Rightarrow \underbrace{\begin{bmatrix} 8 & -9 & 12 & -7 \\ 12 & -5 & 14 & -11 \\ 10 & -2 & 2 & -15 \\ 11 & -10 & -5 & -10 \end{bmatrix}}_{f \otimes k}$$

Drawback:

It models simpler functions

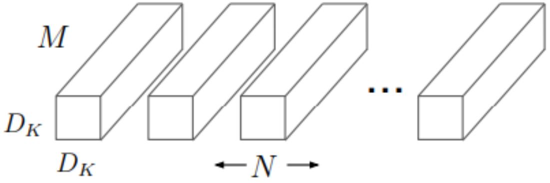
- no ‘diagonal’ entries possible
- successive layers can’t be run in parallel

$$\underbrace{\begin{bmatrix} 6 & 3 & 3 & 6 \\ 4 & 2 & 1 & 8 \\ 2 & 5 & 6 & 4 \\ 9 & 3 & 2 & 1 \end{bmatrix}}_{f} \otimes \underbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}_{c} \otimes \underbrace{\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}}_{r} \Rightarrow \underbrace{\begin{bmatrix} 8 & -9 & 12 & -7 \\ 12 & -5 & 14 & -11 \\ 10 & -2 & 2 & -15 \\ 11 & -10 & -5 & -10 \end{bmatrix}}_{f \otimes c \otimes r}$$

Mobile Net V1

Depthwise separable convolution

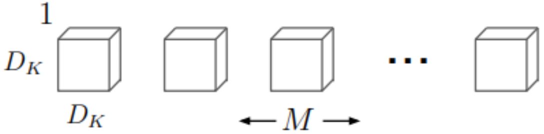
- separate a 3x3 convolution with M input and N output channels
  - M 3x3 convolutions, each applied on a single channel
  - N 1x1 convolutions, combining the M intermediate features
  - add ReLU and Batch Normalization after each layer



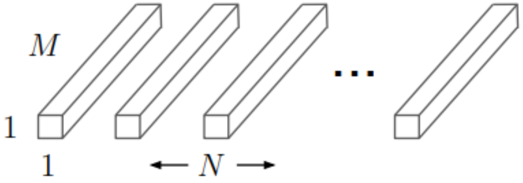
(a) Standard Convolution Filters

Advantages

- fewer add-mul operations
  (8-9 times less than conventional convolution)
- highly efficient operations
  - 95% of time spend on 1x1 convolutions
  - 1x1 convolutions are highly optimized
    - an instance of general matix multiplication (GMM)



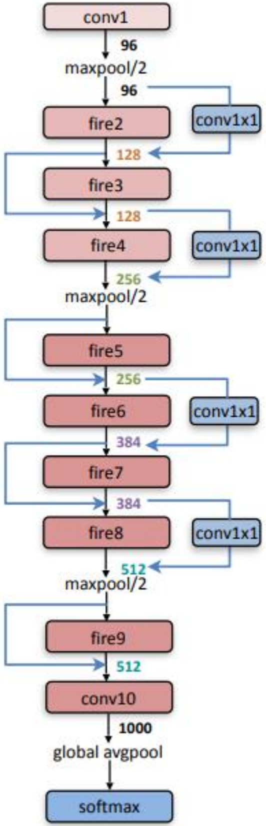
(b) Depthwise Convolutional Filters



SqueezeNet

Goal: Very small model size and efficient execution

- Strategy 1. Replace 3x3 filters with 1x1 filters
  - a 1x1 filter has 9X fewer parameters than a 3x3 filter
- Strategy 2. Decrease the number of input channels to 3x3 filters
  - (number of input channels) * (number of output channels) * (3*3)
- Strategy 3. Downsample late
  - known to yield higher accuracy
- No fully connected layers
  - use global average pooling instead



SqueezeNet architecture

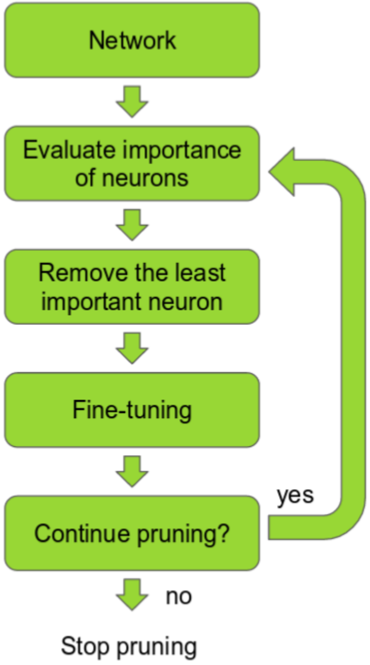
Reducing the model footprint

Pruning

- rank the neurons in the network according to how much they contribute, e.g.:
  - L1/L2 mean of neuron weights
  - their mean activations
  - the number of times a neuron wasn't zero on some validation set
- remove the low-ranking neurons

Deep Compression [Han et al., 2015]

- quantize CNN parameters (e.g. 8-bits of precision)
- uses a codebook



[Pruning Convolutional Neural Networks for Resource Efficient Inference]

ResNeXt: Aggregated Residual Transformations

Idea: “vertical residual blocks”

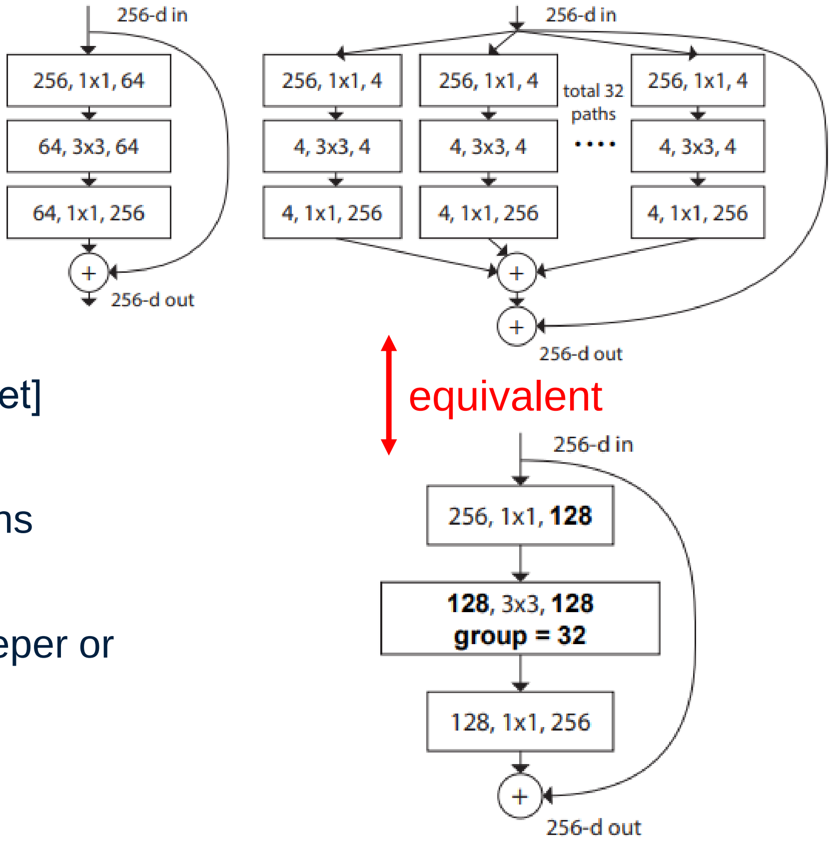
- create blocks with identical topology
    - and independent weights
    - replicate them c times (“cardinality”)
- add these blocks together
- add a skip connection as in ResNet

Related: Krizhevsky et al.’s grouped convolutions [AlexNet]

Advantage:

- larger number of channels, same number of operations
- improved performance
- increasing cardinality is more effective than going deeper or wider when we increase the capacity

A new (NeXt) dimension: depth, width, and **cardinality**



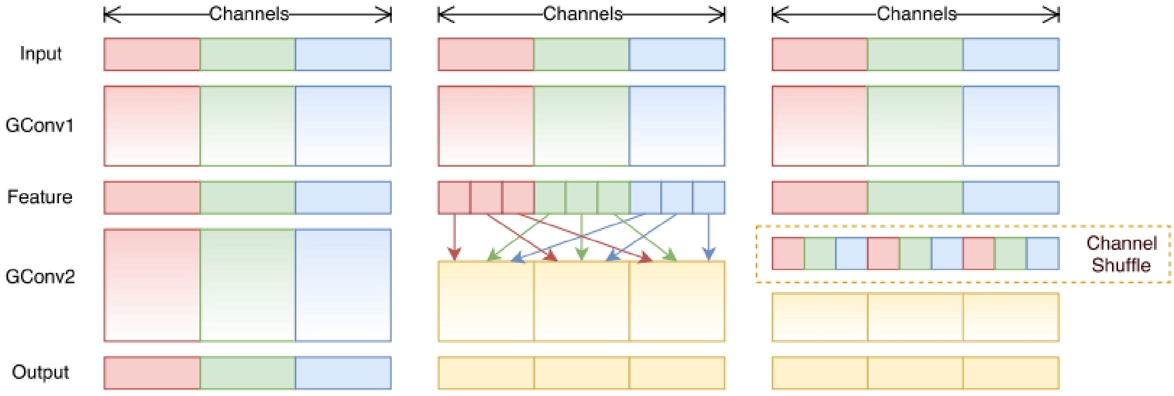
ShuffleNet

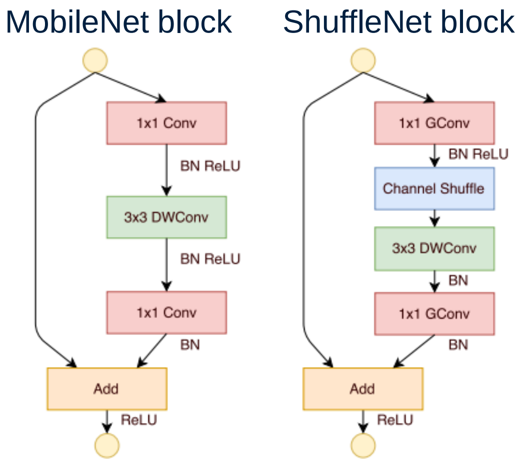
Idea:

- Group-wise convolution
- shuffle features for cross-talk

Advantage:

- less parameters for 1x1 conv
  - 1/ #g parameters, where #g is the cardinality
  - recall that MobileNet spent 95% in 1x1 convolution





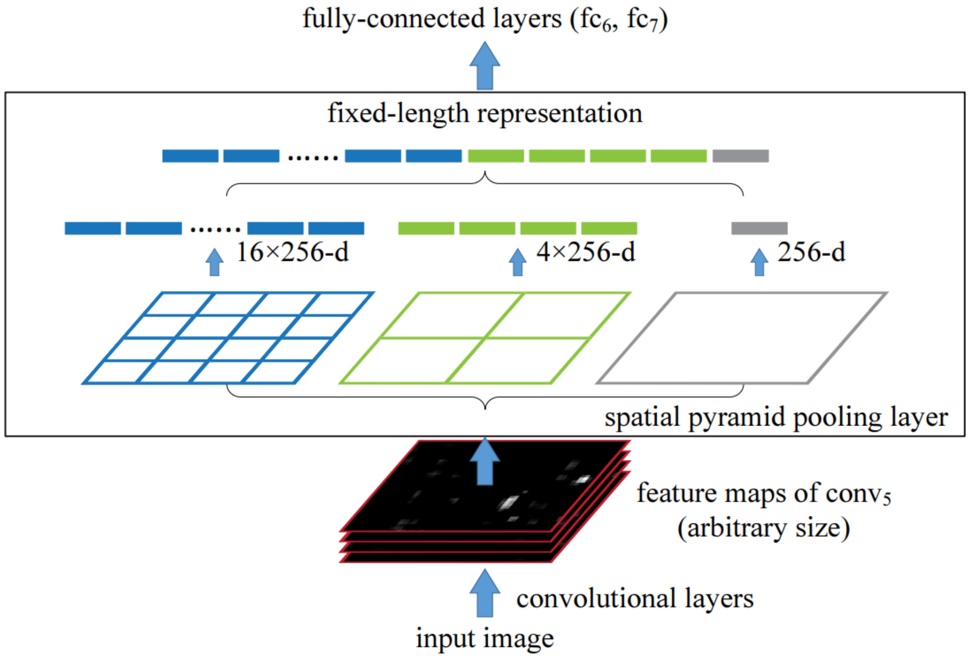
Spatial pyramid pooling

Accumulate features spatially

- at multiple resolutions
- average or max pooling

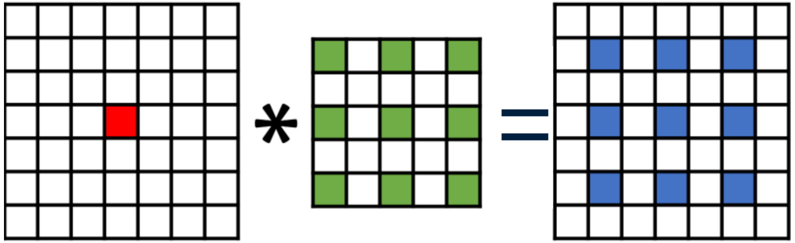
Advantage:

- applies to arbitrary image dimensions



[He et al. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition]

Dilated/Atrous Convolution and ESP Net



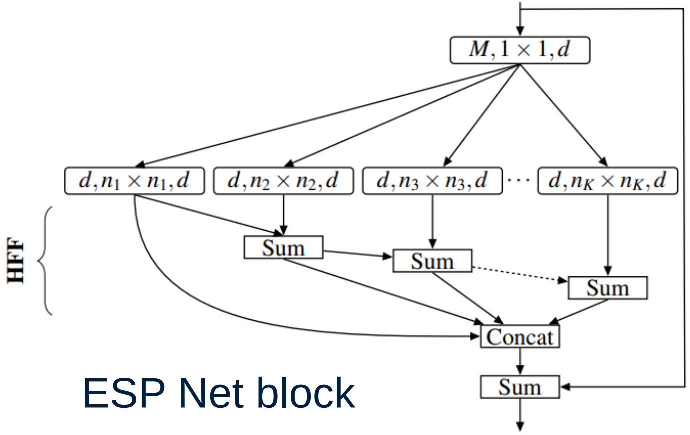
Idea: increase the receptive field

- inserting zeros in the convolutional kernel
    - the effective size of n × n dilated convolutional kernel with dilation rate r, is (n−1)r +1 x (n−1)r +1
    - no increase in parameters
- use a set of dilated filters for multi-scale information

Problem: checkerboard patterns

- Fix: Hierarchical feature fusion (HFF)
    - add output from different dilations before concat

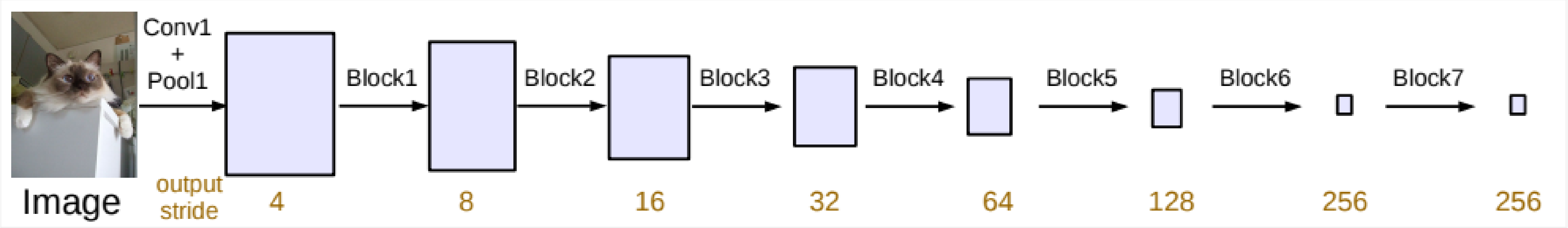
[Mehta et al. ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation]



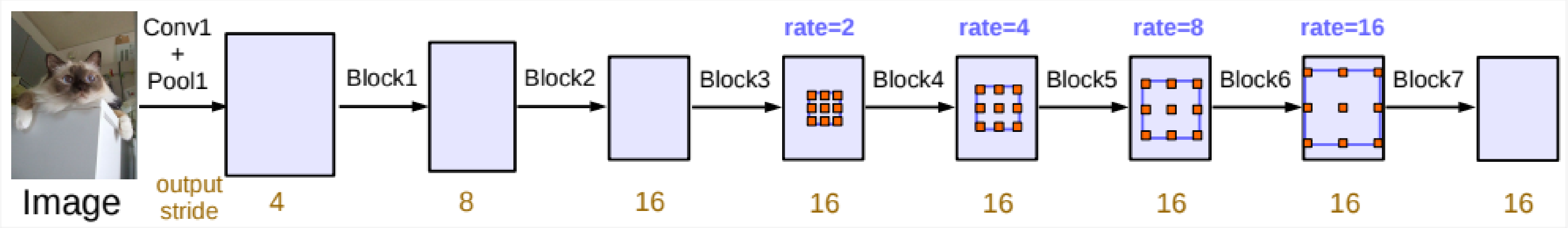
ESP Net block

Sequential application of dilated convolution

- maintains high resolution
- increases receptive field of subsequent layers



(a) Going deeper without atrous convolution.

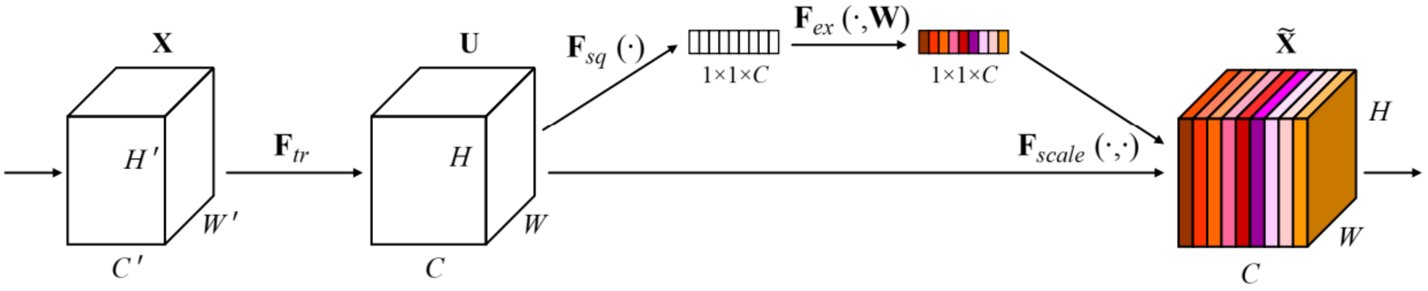


[Chen et al., Rethinking Atrous Convolution for Semantic Image Segmentation]

Squeeze and excitation networks

Idea: feature recalibration:

- use global information to selectively scale informative features



1. Squeeze

- global average pooling yields channel-wise statistics

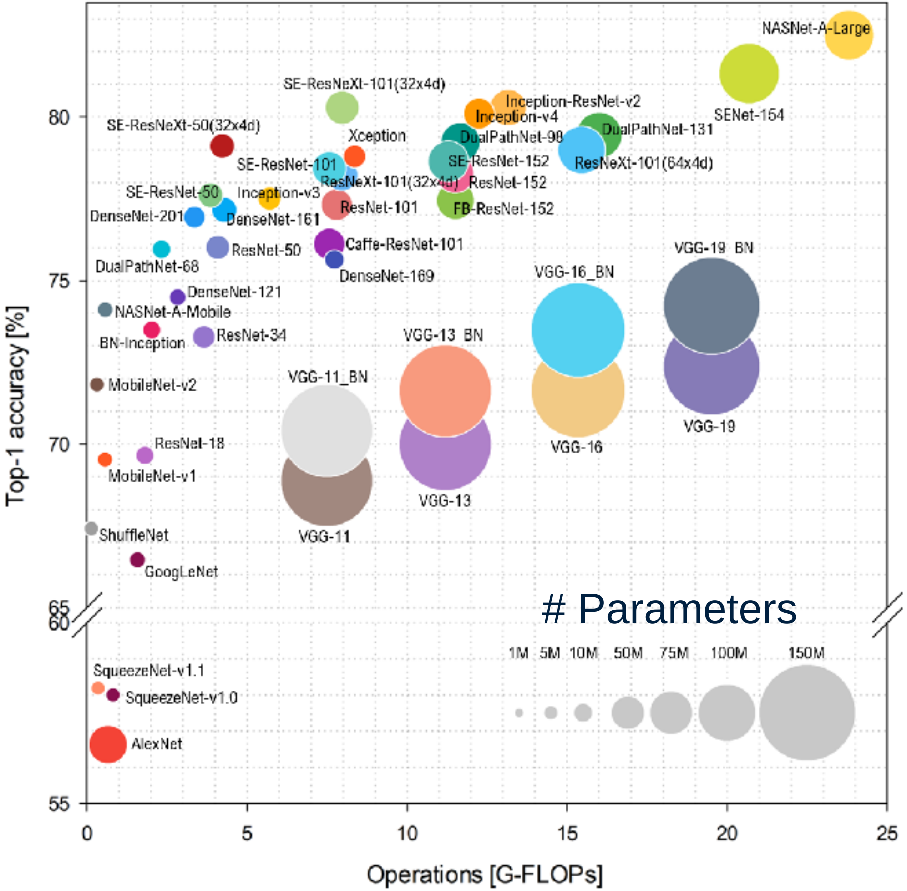
$$\mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i,j)$$

2. Excite

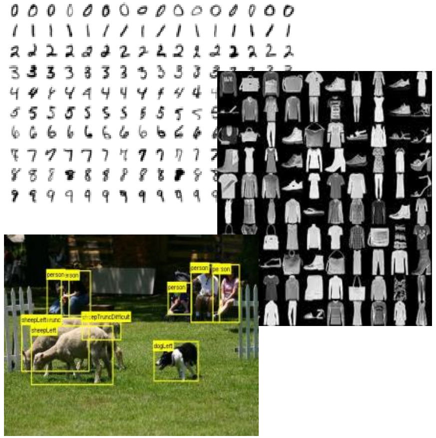
- fully-connected network with a single hidden layer
- scale the original features **U** with the output of *Fex*

$$\mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z}))$$

Network comparison



Assignment I & II



Assignment I

- is due today!
- submit on Canvas
- remember to annotate your solution with # Task X

Assignment II

- the main part will be released tonight
  - to not scoop solutions of Assignment I
- we already released a preparatory notebook
  - to train operations needed for the main task
    - element-wise operations
    - accumulation functions
    - parallelization

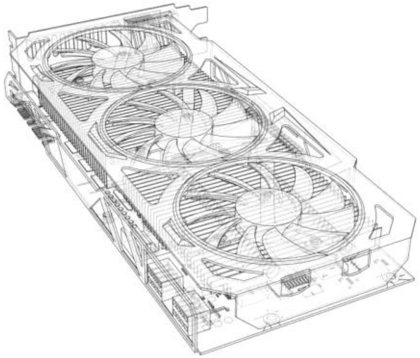
      (tensor operations instead of for loop)

Issues?

Your laptop / desktop

- No GPU?
- Note, parallel dataloaders might not work well on Windows:
  Error: “Can't pickle <function <lambda> …”
- Other issues encountered?

Compute resources at UBC



UBC lin01.students.cs.ubc.ca to lin25.students.cs.ubc.ca

- GTX 1060, 3GB memory, anaconda and pytorch installed
- Create a session (replace colored parts with your name and ports)

  > ssh -N -f -L localhost: 9991:localhost:9992 rhodin@remote.cs.ubc.ca

  > ssh rhodin@remote.cs.ubc.ca

  > ssh -N -f -L localhost:9992:localhost:9991 rhodin@lin01.students.cs.ubc.ca

  Throws error "bind [::1]:9992: Cannot assign requested address" but still works

  > ssh rhodin@lin01.students.cs.ubc.ca

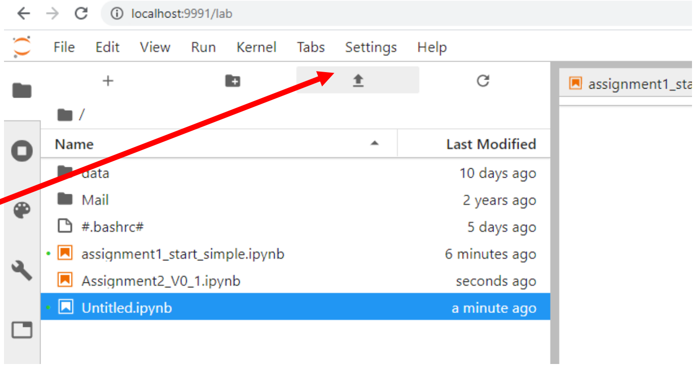
  > /cs/local/lib/pkg/anaconda-2019.07/bin/conda init

  > jupyter-lab --no-browser --port=9991

- Open link provided by jupyter-lab in browser

  http://localhost:9991/?token=6a7ee7feec81f...
- Upload Assignment2.ipynb in Jupiter lab

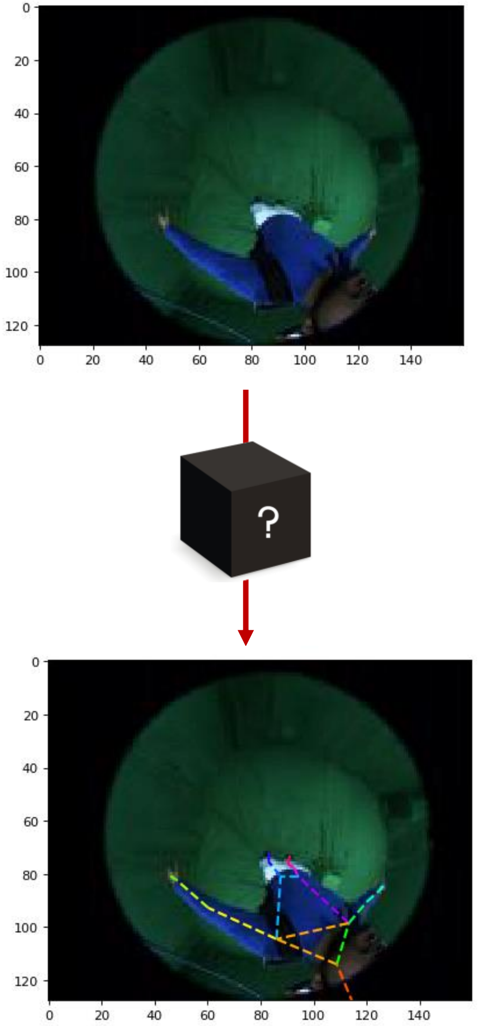
  Note: running a cell with import pytorch might take some seconds



Regression-based 2D pose estimation

A classical regression task

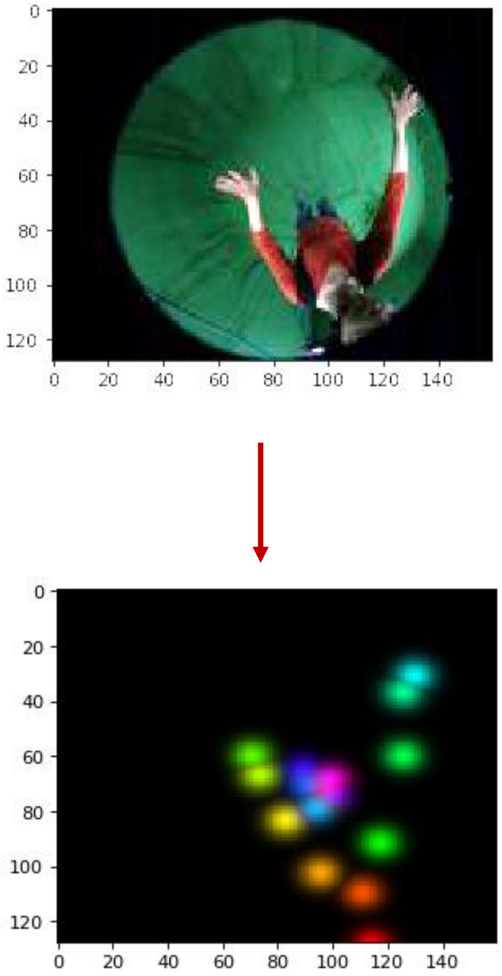
- Input:
  - grid of color values, an image
- Output:
  - pairs of continuous values, the position in the image
  - one pair for each skeleton joint/keypoint
- Neural network architecture:
  - Some convolutional layers to infer an internal representation of the human pose
  - One or more fully-connected layers to aggregate spatial information into the output values



Heatmap-based 2D pose estimation

Phrase the regression task as classification

- separate heatmap H_j for each joint j
- Each pixel of H_j encodes the '*probability*' of containing joint j
  - not a true probability as pixels don't sum to one
- Advantages:
  - Inferred with fully convolutional networks
    - less parameters than fully connected ones (MLPs)
    - applies to arbitrary image resolution and aspect ratio (can be different from training)
    - translation invariance
    - locality
  - Generalizes to multiple and arbitrary number of persons



[Tompson et al., Efficient object localization using convolutional networks.]

Part affinity fields for associating joints of multiple persons

An extension of heatmaps (positions) to vectors (directions)

- Ground truth affinity field L* between joints *c,k*

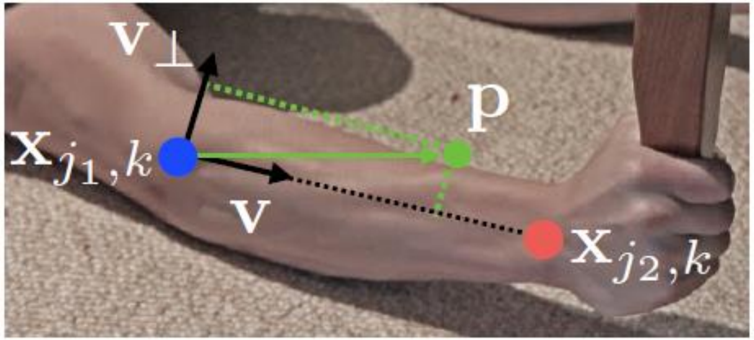
$$\mathbf{L}^*_{c,k}(\mathbf{p}) = \begin{cases} \mathbf{v} & \text{if } \mathbf{p} \text{ on limb } c, k \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Determine presence by

$$0 \leq \mathbf{v} \cdot (\mathbf{p} - \mathbf{x}_{j_1,k}) \leq l_{c,k} \text{ and } |\mathbf{v}_\perp \cdot (\mathbf{p} - \mathbf{x}_{j_1,k})| \leq \sigma_l,$$

with v defined as

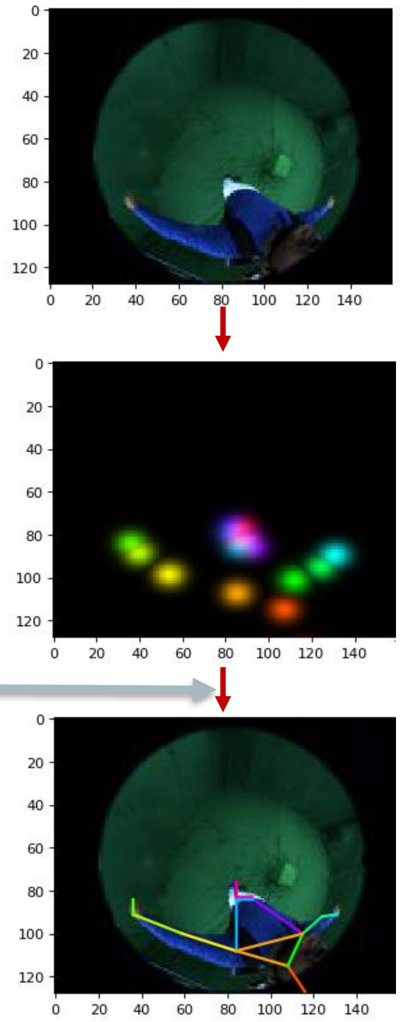
$$\mathbf{v} = (\mathbf{x}_{j_2,k} - \mathbf{x}_{j_1,k}) / ||\mathbf{x}_{j_2,k} - \mathbf{x}_{j_1,k}||_2$$



[Cao et al., Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, CVPR 2017]

Heatmap-based 2D pose estimation II

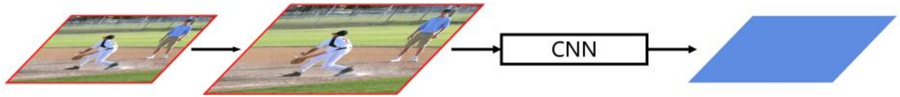
- Disadvantage:
  - Large image scale variations
    - Two-stage pipelines are alleviating this
      1. Detect person bounding box at coarse resolution
      2. Infer skeleton pose within box at high resolution
  - Not end-to-end differentiable
    (pose extraction requires arg-max function)
  - No sub-pixel accuracy
    - multi-scale approaches can overcome this at the cost of execution time
      (average over runs on re-scaled input)



Super-resolution heatmaps

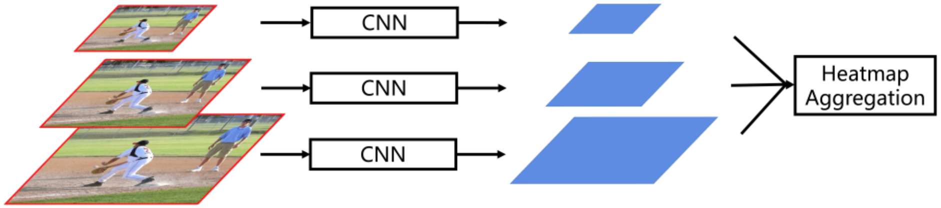
Up sampling the input

- inefficient
- must learn features for different scales (e.g., small and big people)



Multi-scale aggregation

- filters can be scale selective
- inefficient



Picture from [Bottom-up Higher-Resolution Networks for Multi-Person Pose Estimation]

Integral Regression-based 2D pose estimation I

A combination of classification and regression

1. Detection network to produce heatmaps
   - same CNN as for heatmap prediction
2. Soft-max layer to turn heatmap H into probability map P
   - normalizing all pixels in each heatmap H

$$P[u,v] = \text{soft-max}(H,(u,v)) = \frac{e^{H[u,v]}}{\sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} e^{H[x,y]}}$$

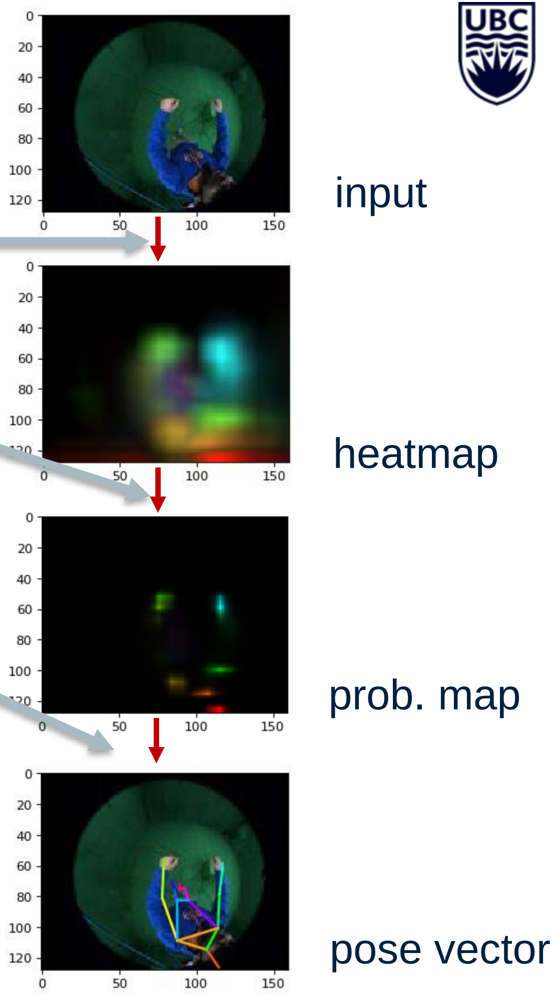
3. Integration layer to regress joint position (expected position)
   - can be interpreted as voting/weighted average

     each pixel votes for its own position, weighted by its probability

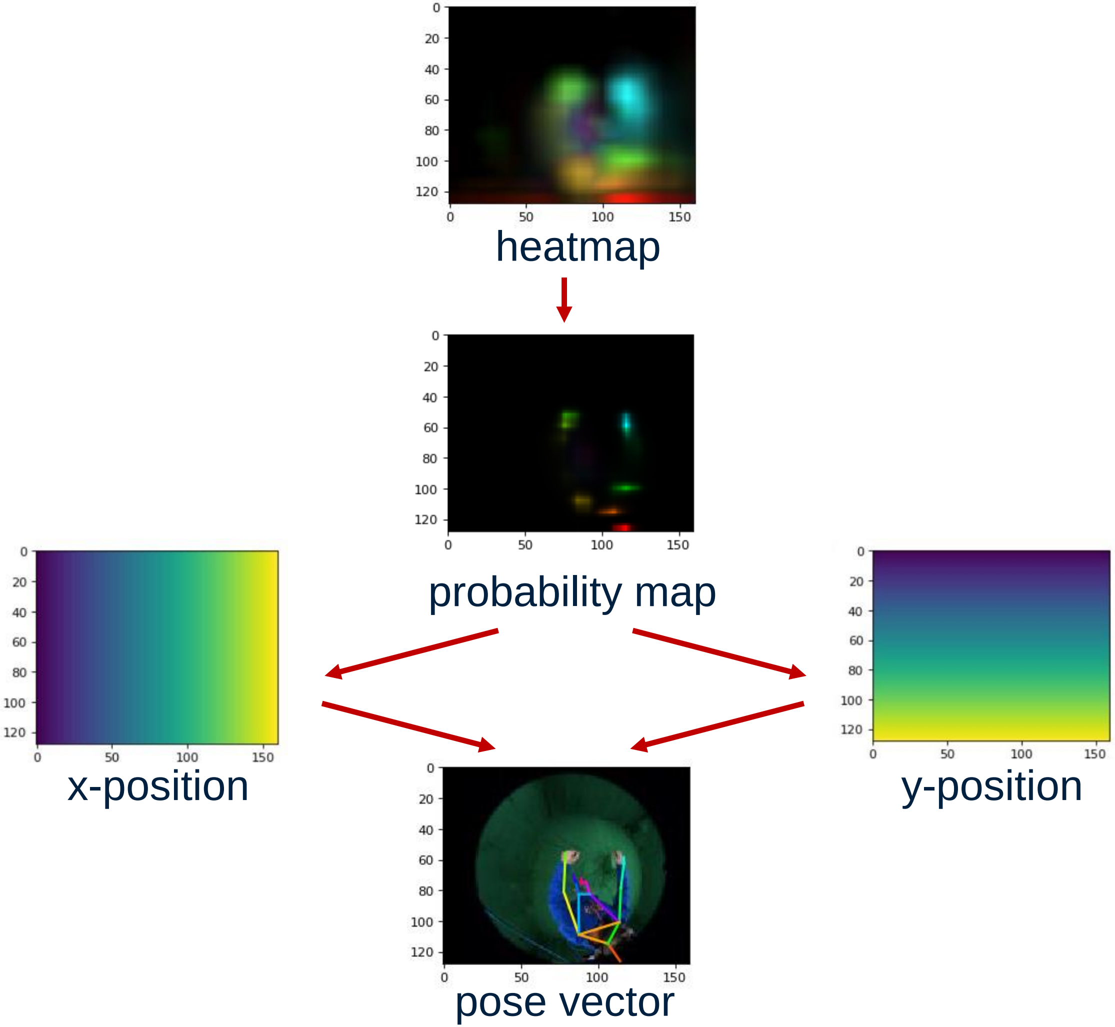
$$\text{pose}_x = \sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} x P[x,y]$$

$$\text{pose}_y = \sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} y P[x,y]$$

[Sun et al., Integral Human Pose Regression.]



Details



Integral Regression-based 2D pose estimation II

Advantages

1. Fully-convolutional CNN (as for heatmap classification)
2. Differentiable 2D pose regression
   - soft-max is differentiable, stable, and efficient to compute

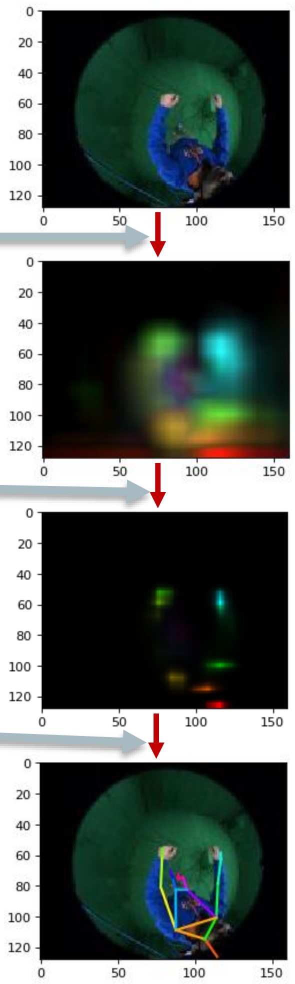
$$P[u,v] = \text{soft-max}(H,(u,v)) = \frac{e^{H[u,v]}}{\sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} e^{H[x,y]}}$$

   - sum over probability map is differentiable

$$\text{pose}_x = \sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} x P[x,y]$$

$$\text{pose}_y = \sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} y P[x,y]$$

3. End-to-end training
   - no difference between training and inference
   - sub-pixel accuracy possible through joint influence of pixels
     - low-resolution heatmaps possible



input

heatmap

prob. map

pose vector

Integral Regression-based 2D pose estimation III

Disadvantages / open questions = possible course projects!

1. Sensitive to outliers
   - if there are two maxima in the heatmap, the predicted position will be in the middle of the two
2. How to support multiple people, at different scales?
   - Some form of hierarchical model?
3. Part affinity fields have been successful, can we develop a differentiable model?
   - An elongated ellipse that has position and orientation?
4. What about occluded joints?
5. What about temporal information?
6. Is it possible to infer neck-centered human pose (not knowing the absolute position, only relative distance of keypoints to the neck)?

Hidden questions