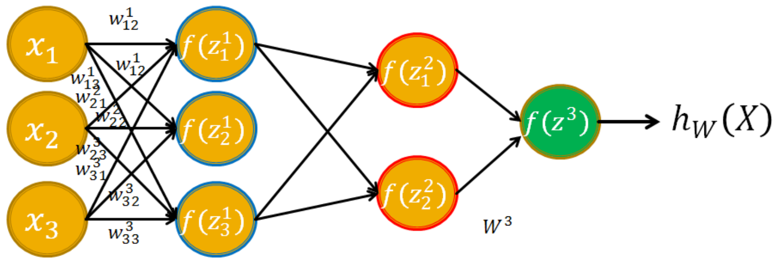
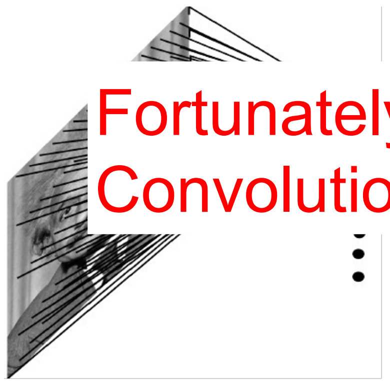
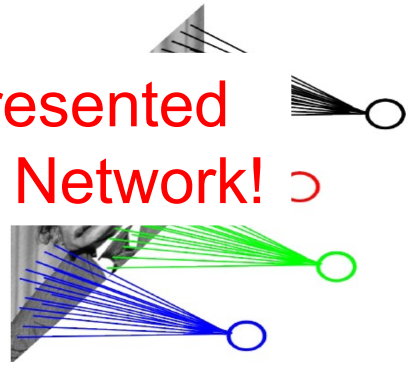
Deep Learning for Image Understanding

Challenge for Deep Learning



- Example: 200x200 image
  - Fully connected NN, 40,000 hidden units=1600000000 parameters to link the input and layer 1.
  - Locally connected, 40,000 hidden units 10x10 fields=4000000 parameters
  - Local connections capture local dependencies.



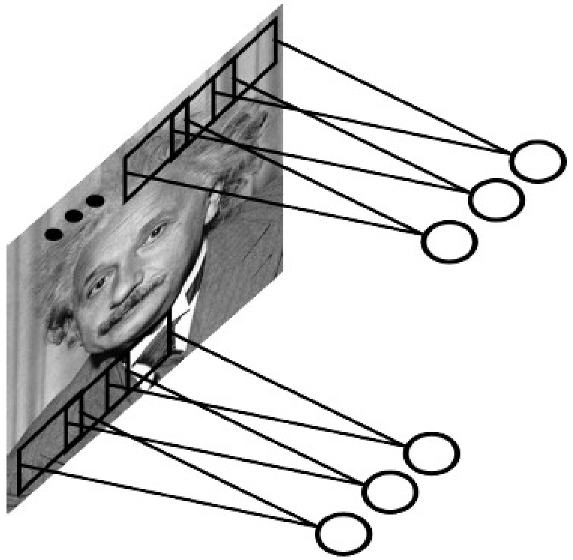


Fortunately, Lecun presented Convolutional Neural Network!

Shared weights & Convolution

- Features that are useful on one part of the image are probably useful elsewhere.
  - Sparse representation
  - “Image is sparse”- A. Leven
- Shift equivalent processing (spatial property)
  - When the input shifts, the outputs also shifts.
- Convolution
  - With a learned kernel (or filter): $A_{ij} = \sum_{kl} W_{kl} X_{i+j,k+l}$
  - The filtered “image” is called a feature map.

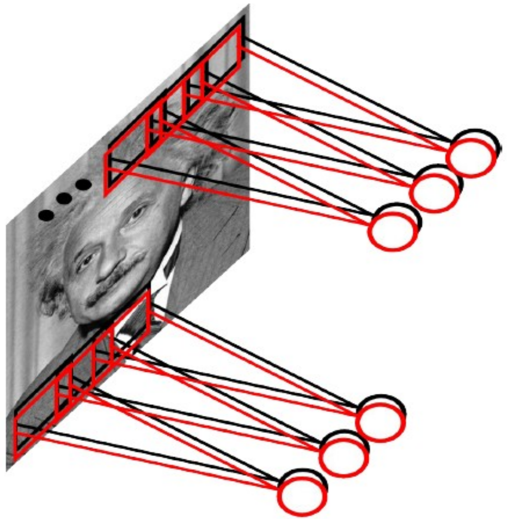
- Example: 200x200 image
  - 10 filters of size 10x10
  - 10 feature maps of size 200x200
  - 400,000 hidden units with 10x10 fields=1000 parameters

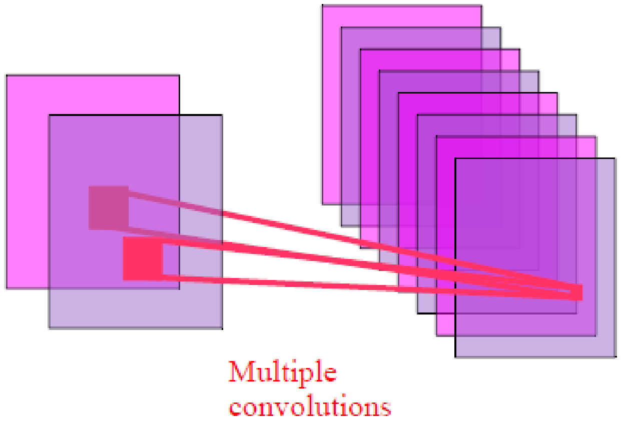


Computation workload is reduced!

Why 10 filters?

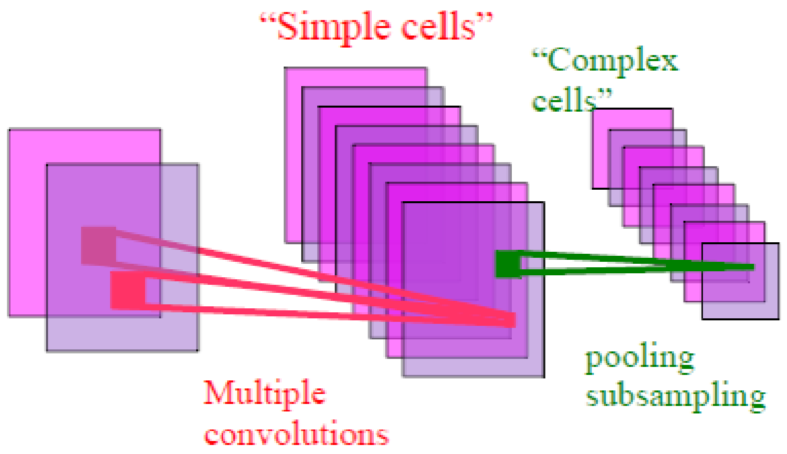
- Detect multiple motifs at each location.
- The collection of units looking at the same patch is akin to a feature vector for that patch.
- The result is a 3D array, where each slice is a feature map.





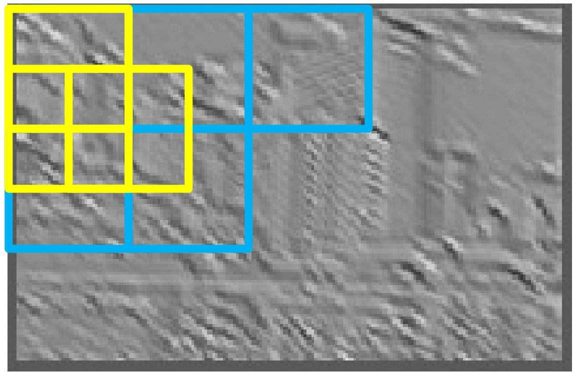
Early hierarchical feature model for vision

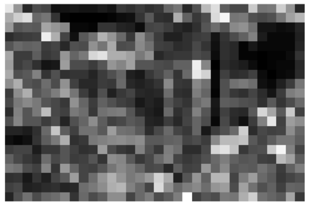
- [Hubel & Wiesel 1962]
  - Simple cells detect local features.
  - Complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.



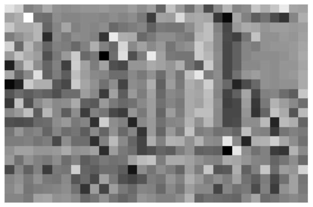
Pooling

- Pooling
  - Spatial pooling
    - Non-overlapping/overlapping regions
    - Average or max
    - Usually combine with subsampling
  - Role of spatial pooling
    - Invariance to small transformation
    - Larger receptive field
    - Smoothness
    - Reduce variants





Max

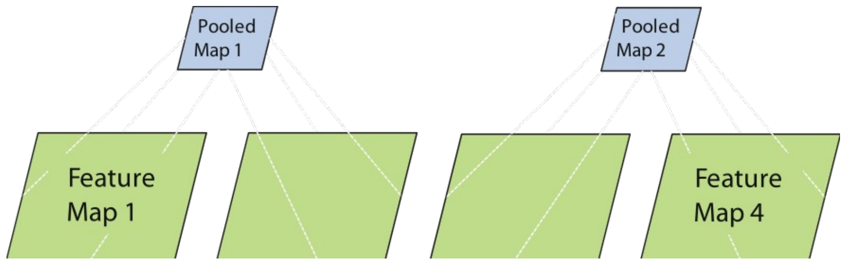


Averag

e

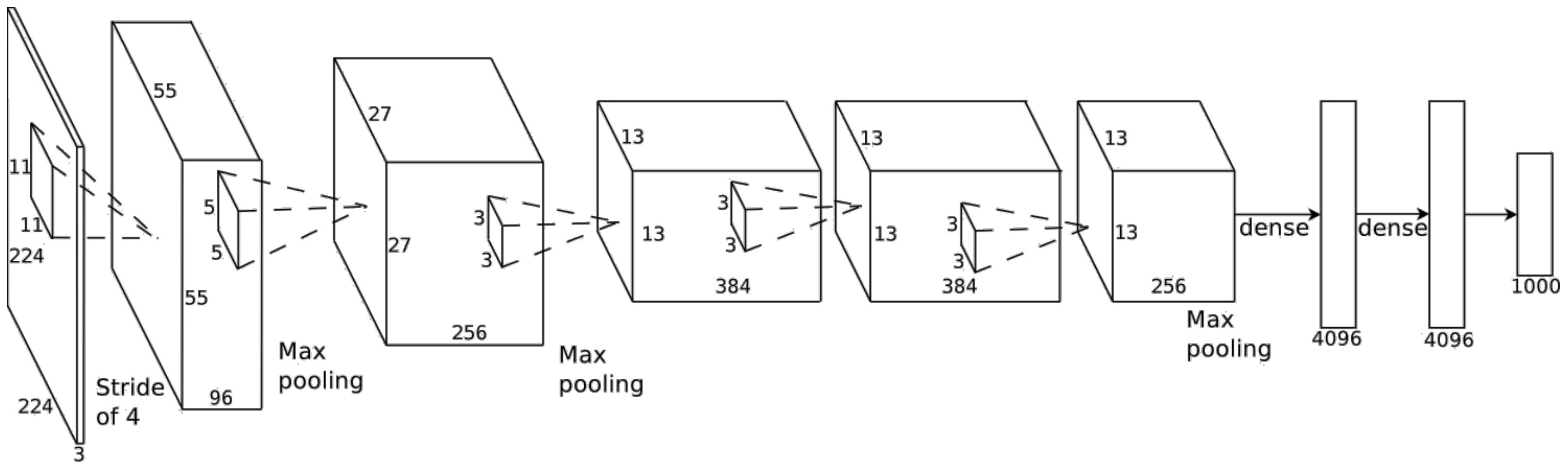
Pooling

- Pooling
  - Pooling across feature maps
    - Additional form of inter-feature competition
    - Max-pooling
  - Role of pooling across maps
    - Find distinct features
    - Reduce variants



A development: cuda-convnet

- Though convolution and pooling/subsampling reduce the variants greatly, the training of CNN is still time consuming.
- Hence, cuda-convnet is presented by extending LeNet [Lecun, 98, 06]
  - A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet classification with deep convolutional neural networks, NIPS 2012.
  - Code: https://code.google.com/p/cuda-convnet/

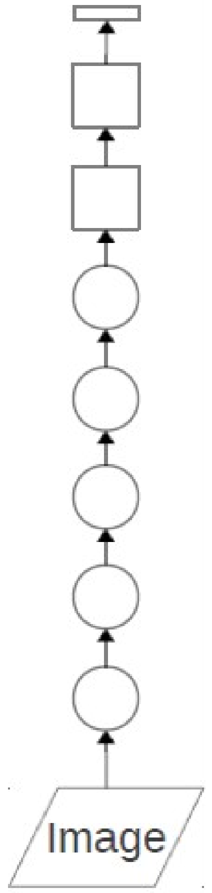


Cuda-convnet

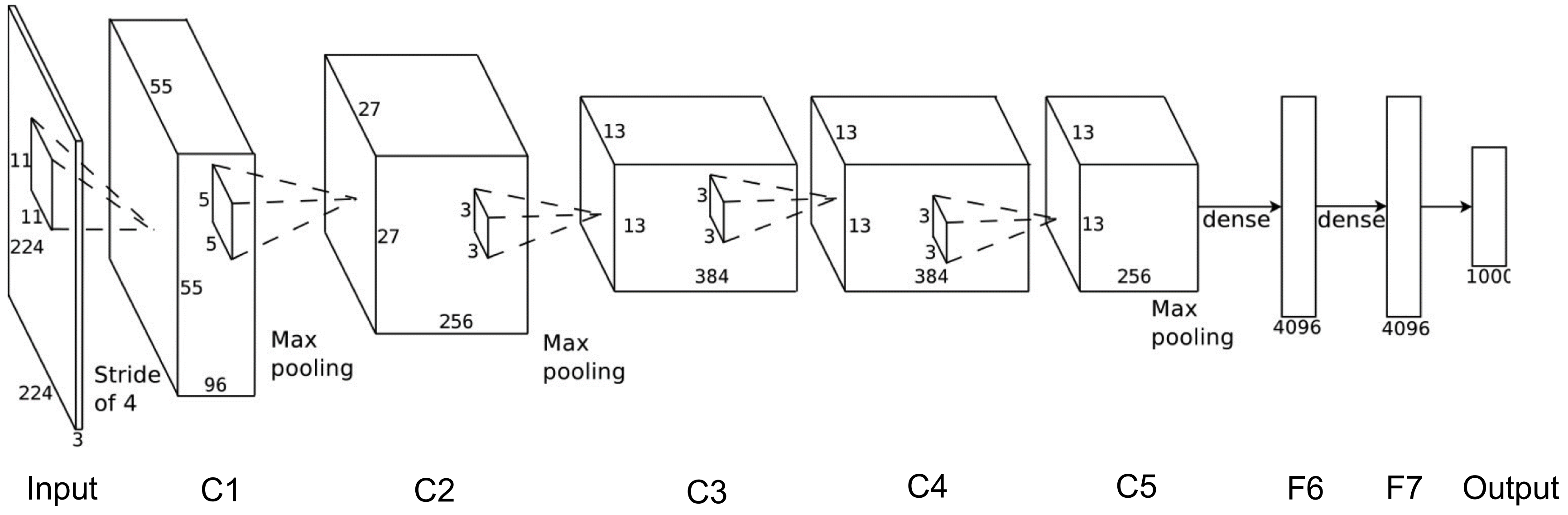
- Features
  - Trained with gradient decent on two nVidia GPUs for about a week @ ImageNet
  - 650,000 neurons
  - 60,000,000 parameters
  - 7 hidden "weight" layers
  - Final feature layer: 4096-dimensional

○ Convolutional layer: convolves its input with a bank of 3D filters, then applies "sigmoid"-like operation.

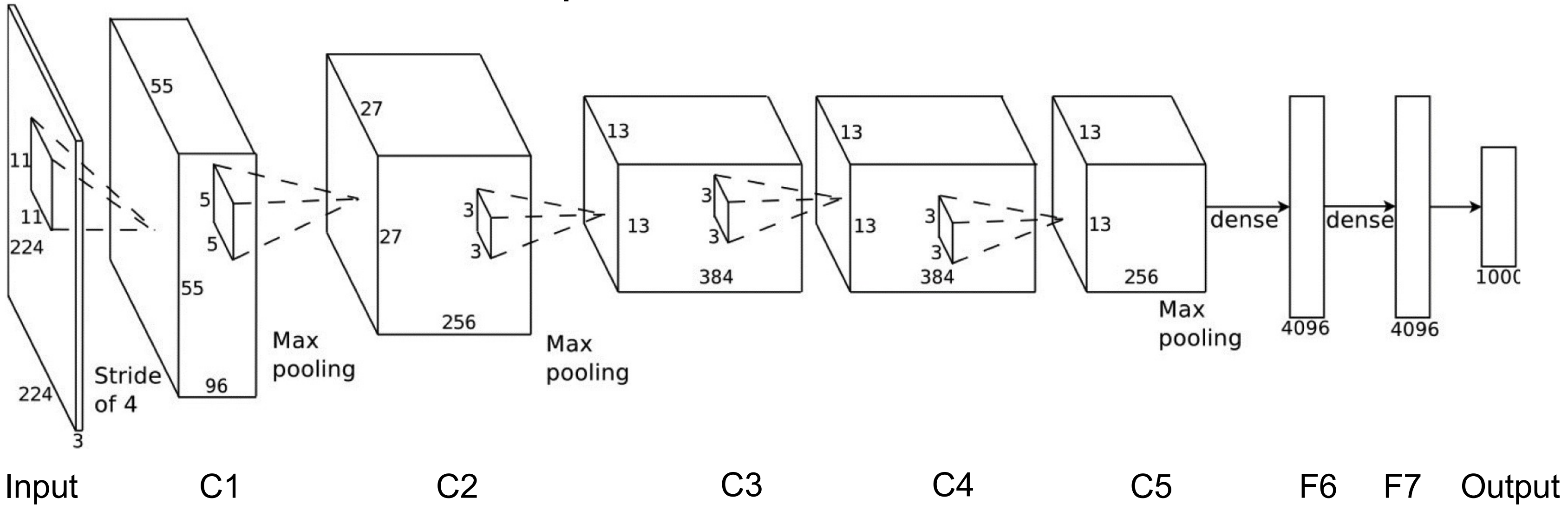
□ Fully-connected layer: applies linear filters to its input, then applies "sigmoid"-like operation.



- Input: 3@224x224 → C1: 96@55x55
  - 96 11x11x3 convolutional kernels (3D filters) are applied on 224x224x3 input image.
  - Sliding with the stride of 4.
  - 96 55x55 feature volumes are obtained.

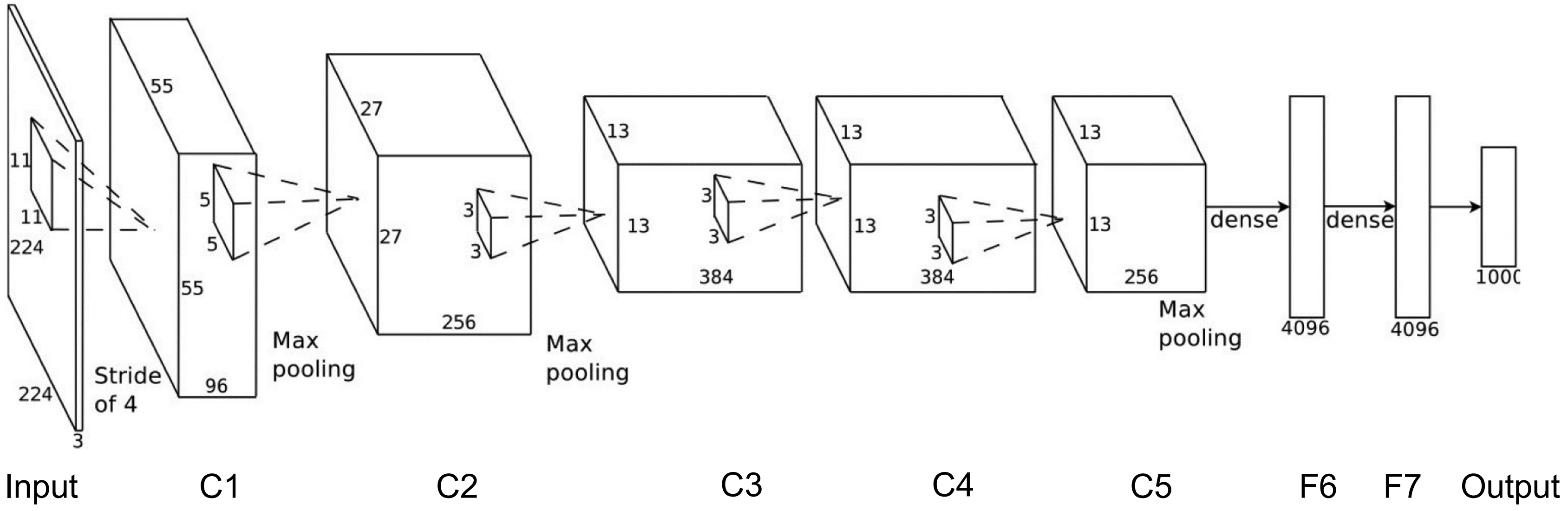


- C1: 96@55x55 → C2: 256@27x27
  - Max pooling: choose the maximal value in each 2x2 neighborhood, change C1 to be 96@27x27.
  - 256 5x5x96 3D kernels are applied on C1: 96@27x27.
  - 256 27x27 feature maps are obtained.

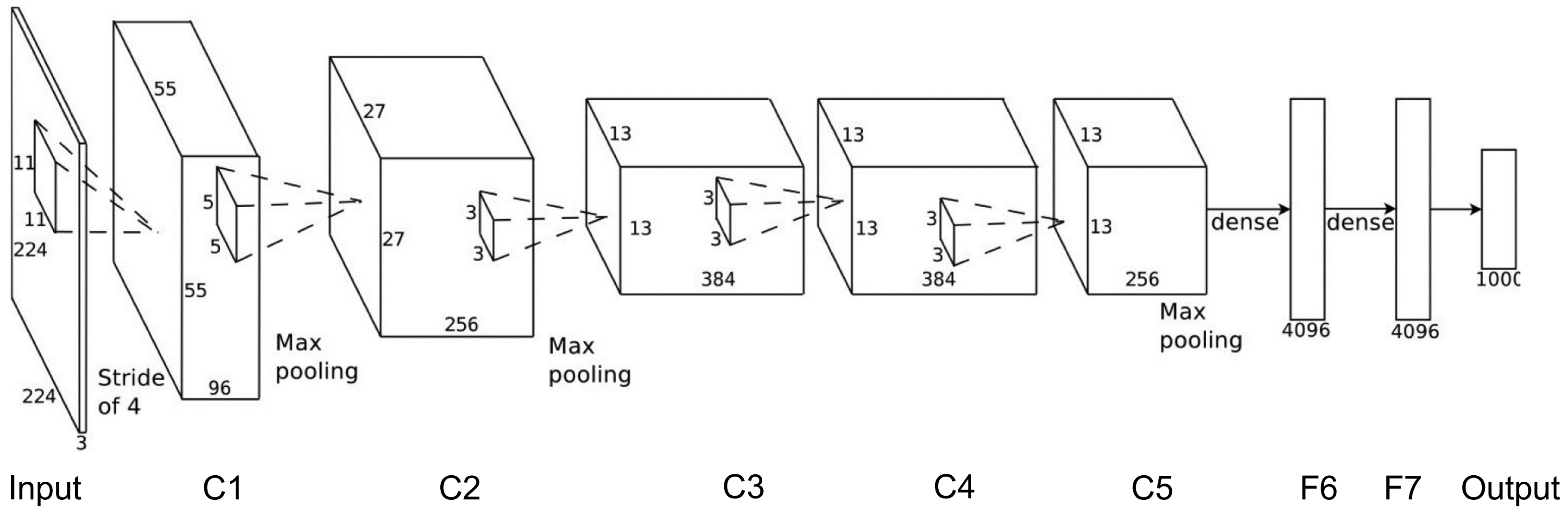


C2: 256@27x27 → C3: 384@13x13

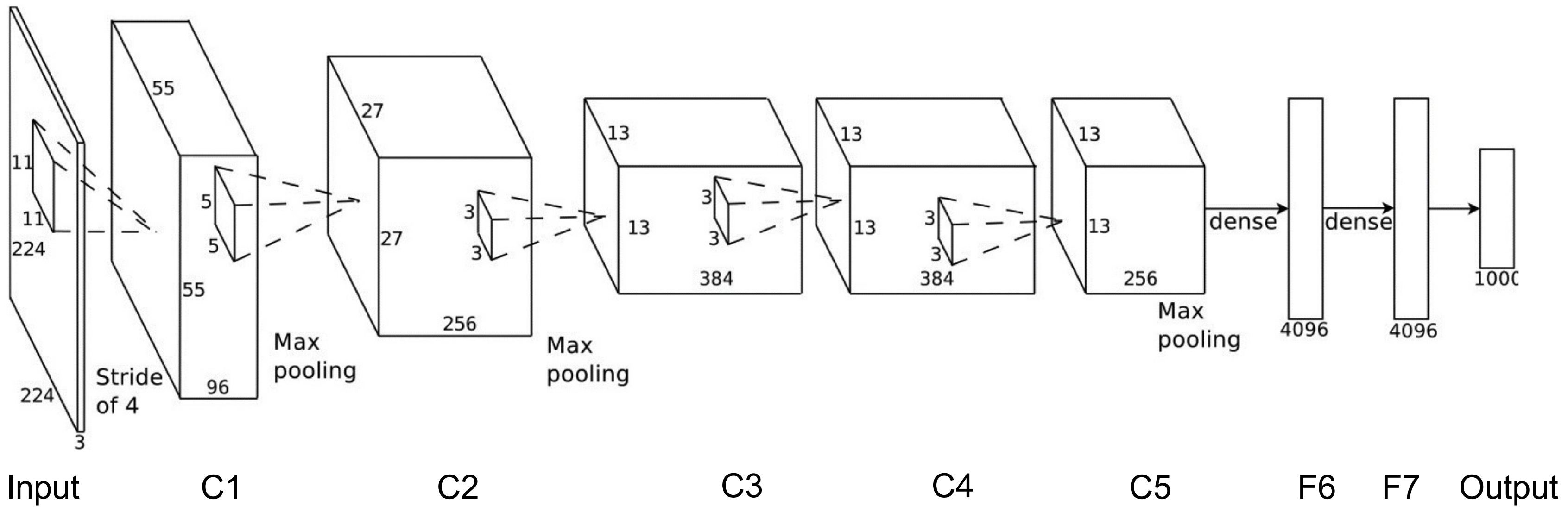
- Max-pooling: choose the maximal value in each 2x2 neighborhood, change C2 to: 256@13x13
- 384 3x3x256 3D kernels are applied on C2: 256@13x13.
- 384 13x13 feature maps are obtained.



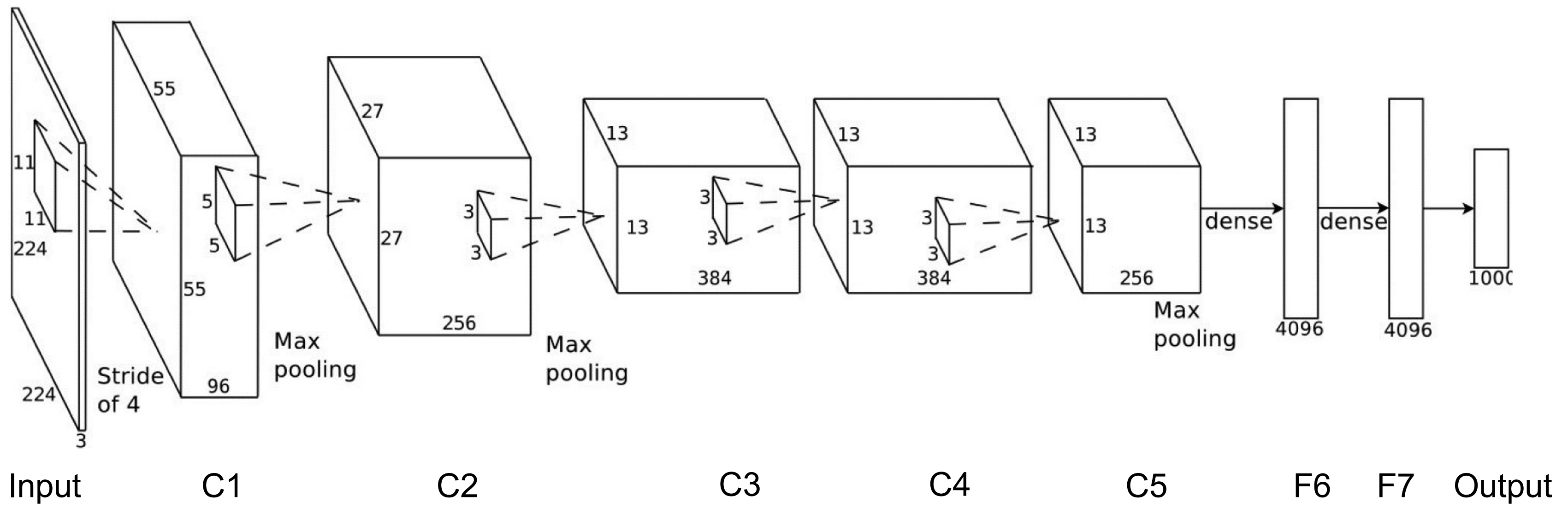
- C3: 384@13x13 → C4: 384@13x13
  - 384 3x3x384 3D kernels are applied on C3.
  - No intervening pooling or normalization.
- C4: 384@13x13 → C5: 256@13x13
  - 256 3x3x384 kernels are applied on C4.
  - No intervening pooling or normalization.



- C5: 256@13x13 → F6: 4096@1x1
  - Max pooling across 256 different feature maps.
  - Fully connected neural network between the pooled C5 and F6.



- F6: 4096@1x1 → F7: 4096@1x1
  - Fully connected neural network between F6 and F7.
- F7: 4096@1x1 → Output: 1000@1x1
  - Fully connected neural network between F7 and output layer.



Tasks for which CNNs are the best

- Handwriting recognition: MNIST (many), Arabic HWX (IDSIA)
- OCR in the wild [2011]: StreetView House Numbers (NYU)
- Traffic sign recognition [2011]: GTSRB competition (IDSIA, NYU)
- Pedestrian detection [2013]: INRIA datasets and others (NYU)
- Volumetric brain image segmentation [2009]: Connectomics (IDSIA, MIT)
- Human action recognition [2011]: Hollywood II dataset (Stanford)
- Object recognition [2012]: ImageNet competetion.
- Scene Parsing [2012]: Stanford bgd, SiftFlow, Barcelona (NYU)
- Scene parsing from depth images [2013]: NYU RGB-D dataset (NYU)
- Speech recognition [2012]: Acoustic modeling (IBM and Google)
- Breast cancer cell mitosis detection [2011]: MITOS (IDSIA)

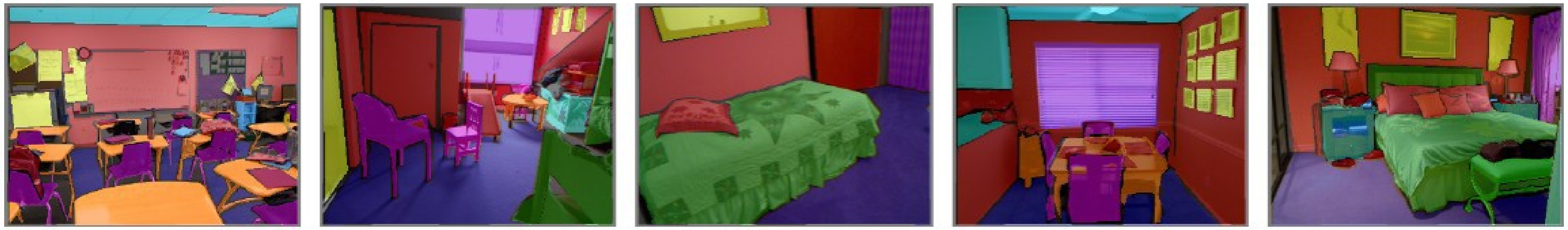
Some state-of-the-art performance

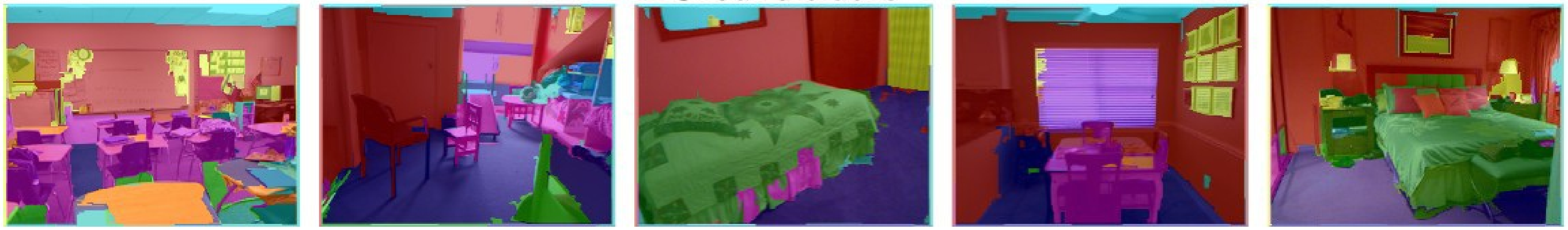
- Traffic sign recognition
  - German Traffic Sign Reco Bench
  - 99.2% accuracy

- House Number Recognition
  - Street View House Numbers
  - 94.3% accuracy

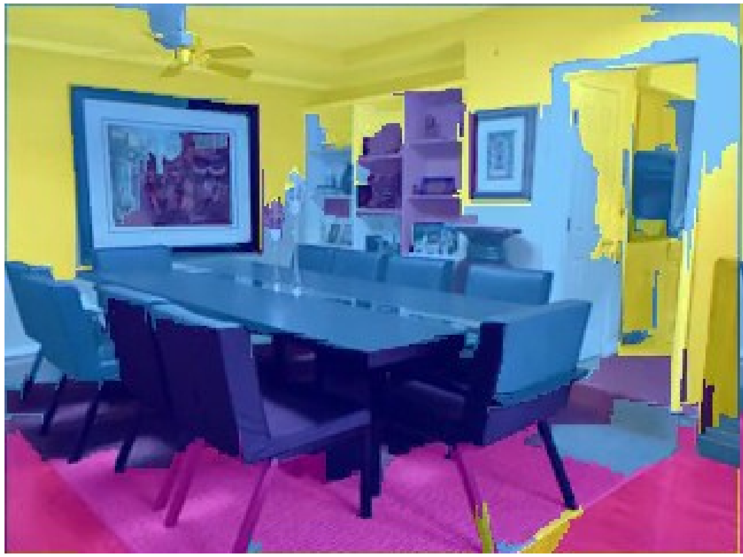
Scene parsing/labeling on RGB-D images

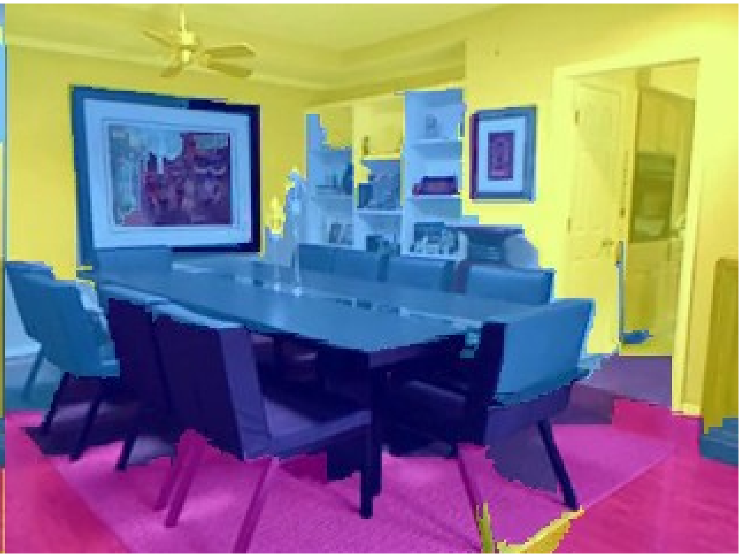
🟥 wall	🟦 books	🟪 chair	🟦 furniture	🟩 sofa	🟥 object	🟫 TV
🟩 bed	🟦 ceiling	🟦 floor	🟨 pict./deco	🟧 table	🟥 window	⬛ uknw


Ground truths

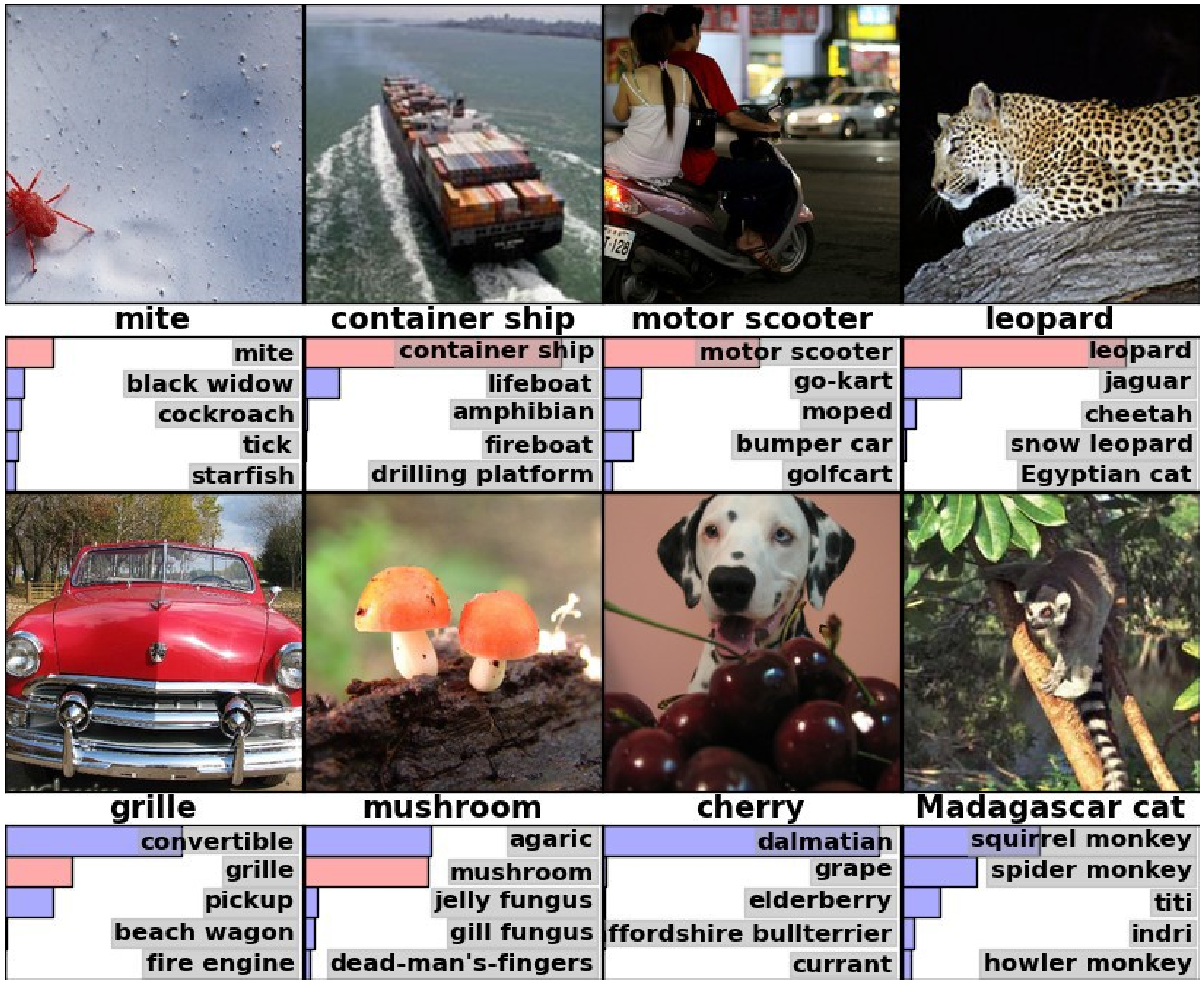

Our results

Semantic segmentation on RGB-D images


Result


Ground truth

Validation classification



Validation localization

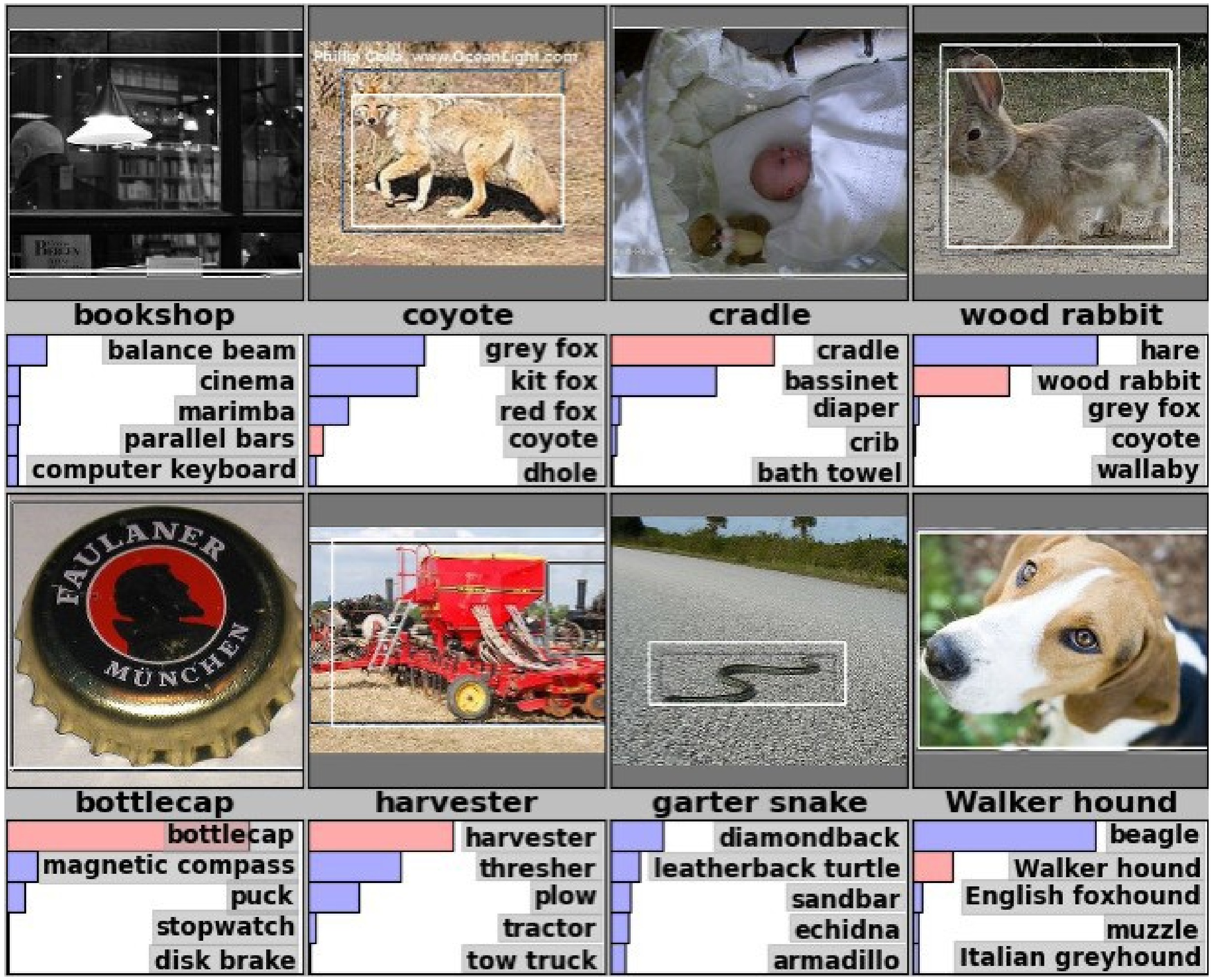
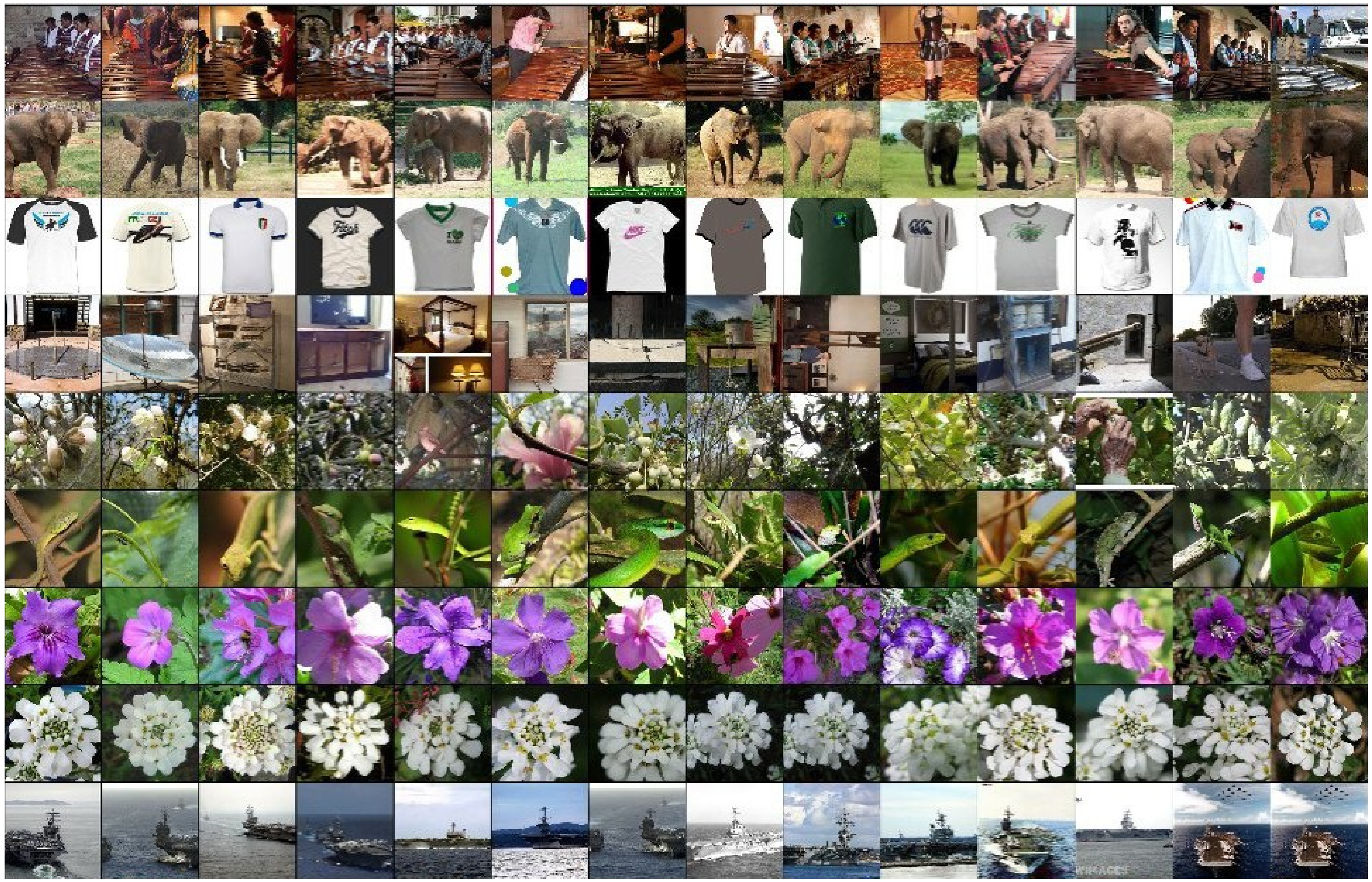
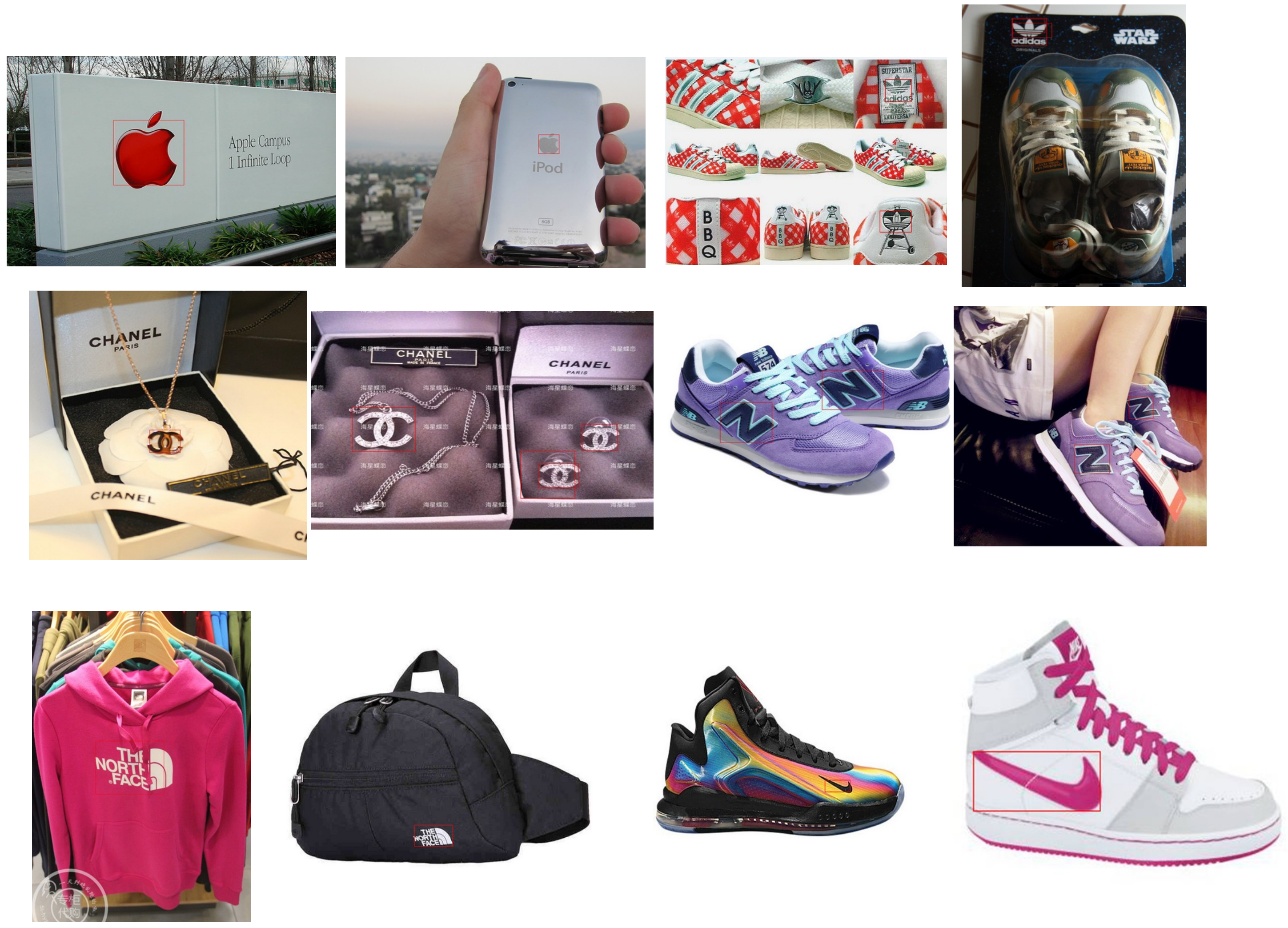


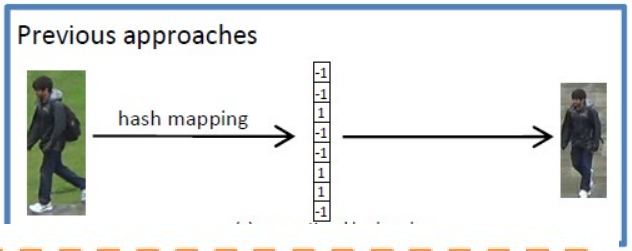
Image retrieval

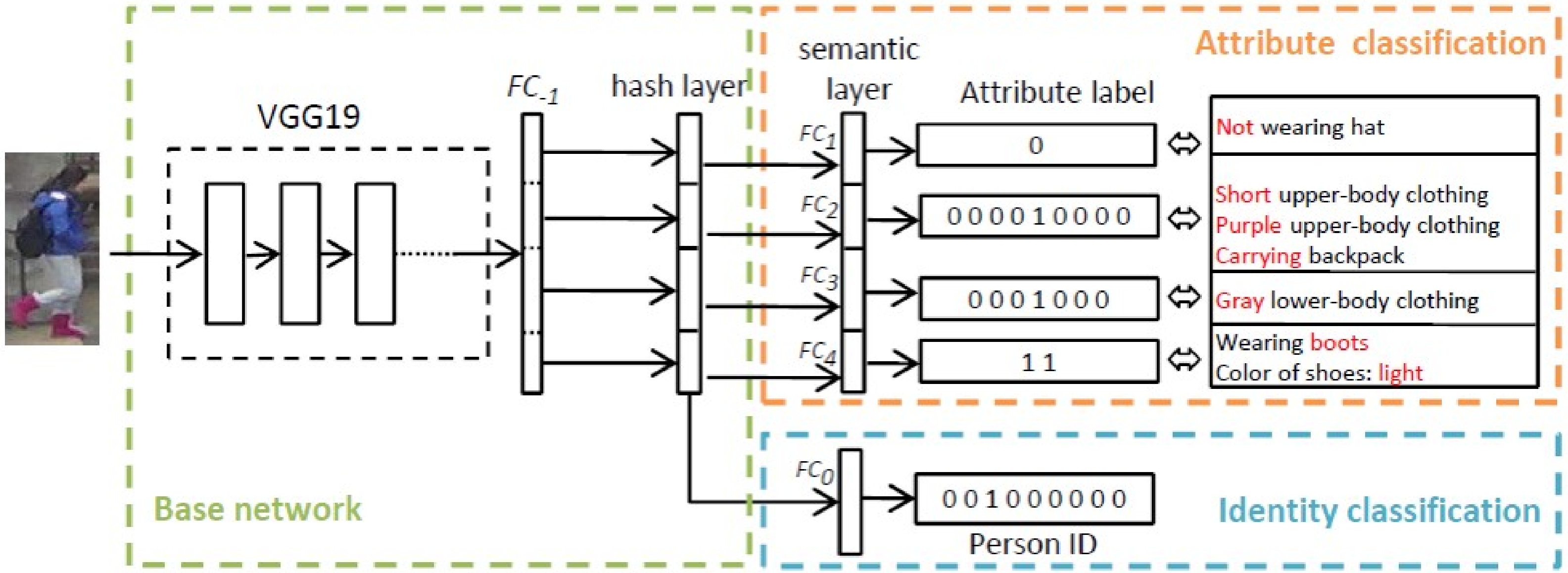


Logo detection

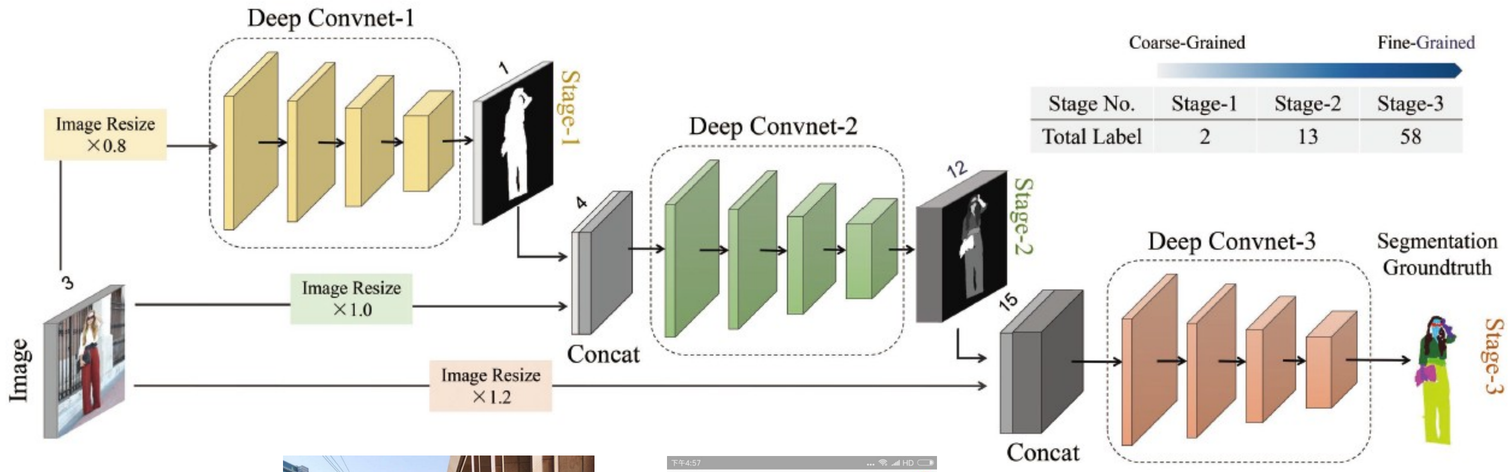


Semantic Structured Hashing for Person Re-identification

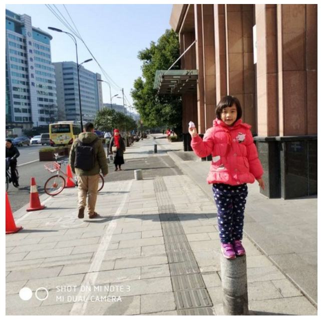


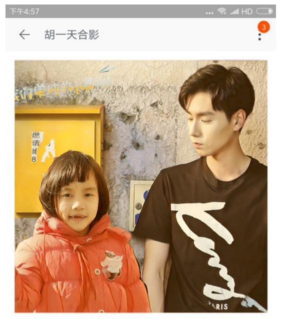


Finer-Net: Cascaded Human Parsing with Hierarchical Granularity

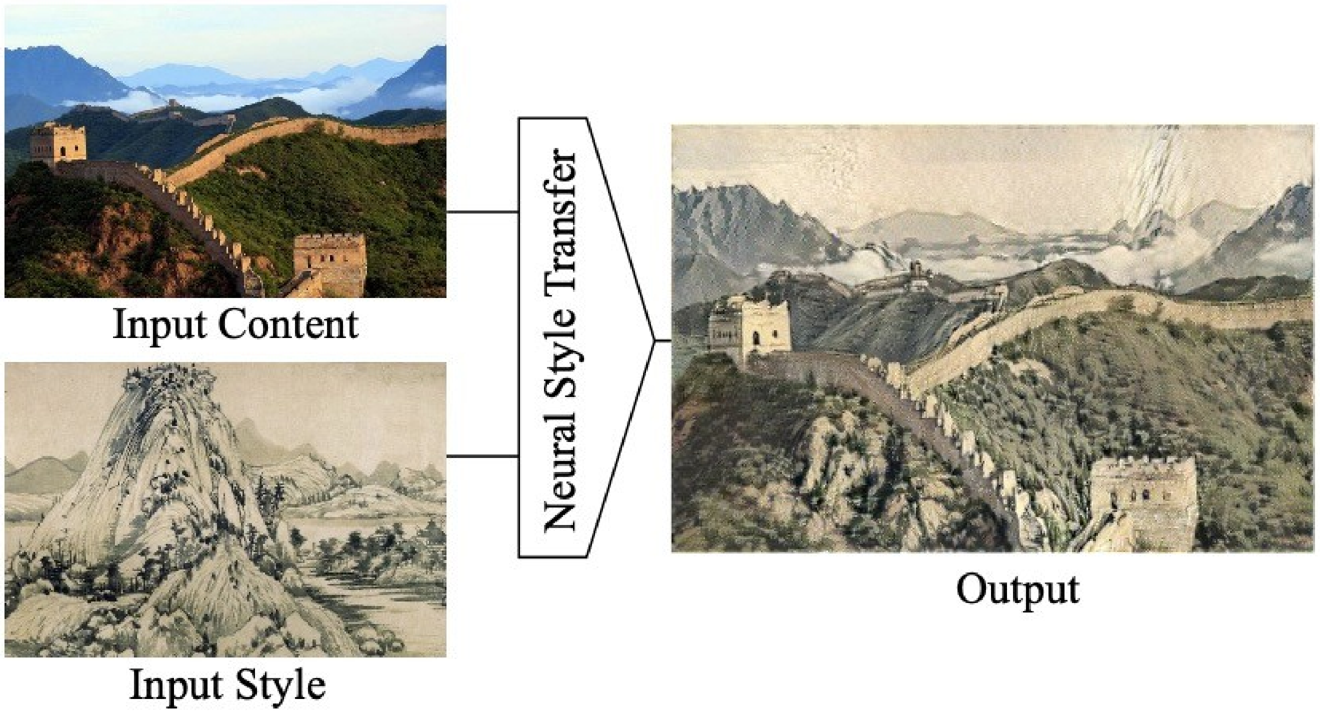


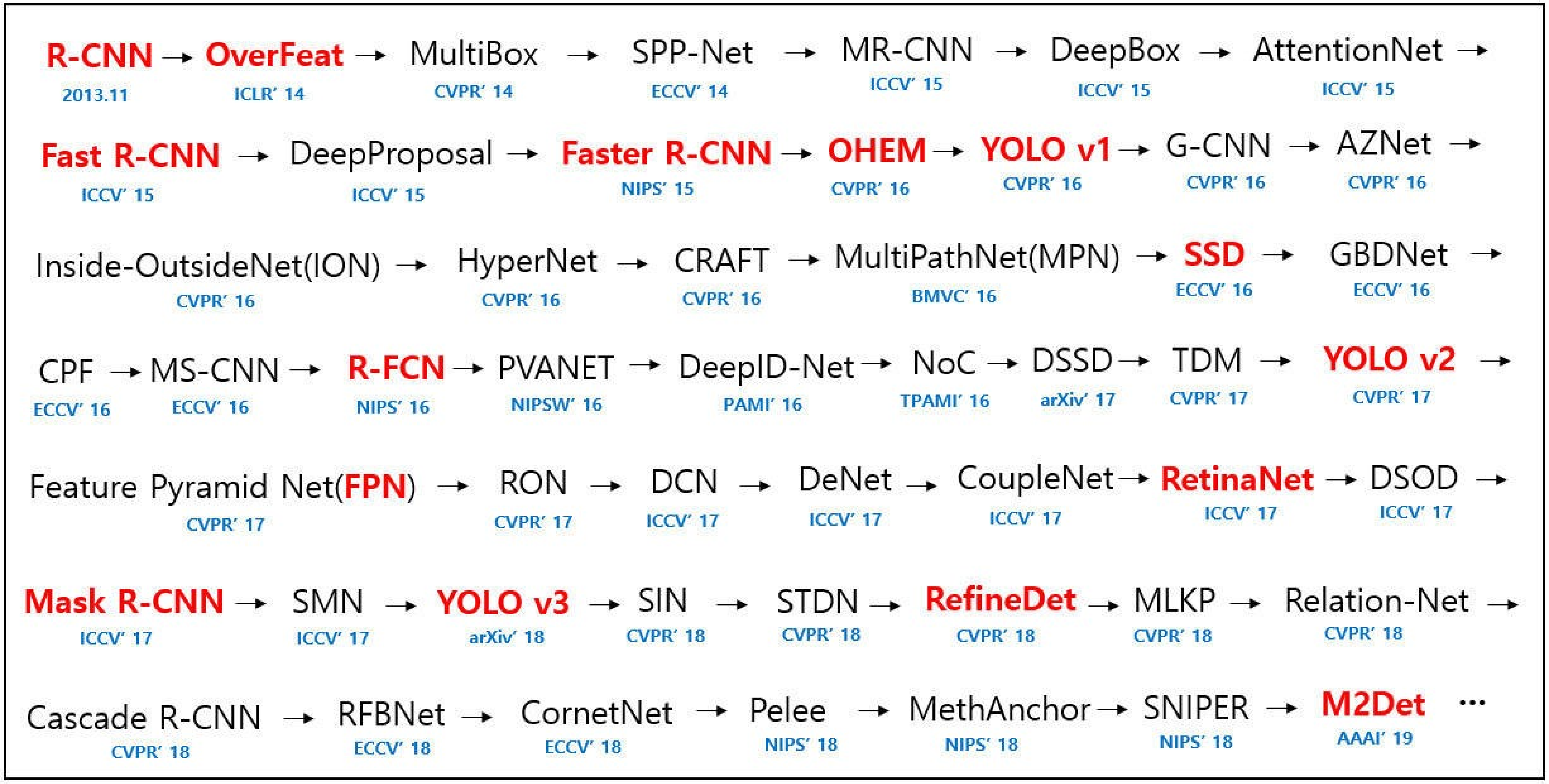
Stage No.	Stage-1	Stage-2	Stage-3
Total Label	2	13	58





Neural Style Transfer





R-CNN (2013.11) → **OverFeat** (ICLR' 14) → MultiBox (CVPR' 14) → SPP-Net (ECCV' 14) → MR-CNN (ICCV' 15) → DeepBox (ICCV' 15) → AttentionNet (ICCV' 15) →

Fast R-CNN (ICCV' 15) → DeepProposal (ICCV' 15) → **Faster R-CNN** (NIPS' 15) → **OHEM** (CVPR' 16) → **YOLO v1** (CVPR' 16) → G-CNN (CVPR' 16) → AZNet (CVPR' 16) →

Inside-OutsideNet(ION) (CVPR' 16) → HyperNet (CVPR' 16) → CRAFT (CVPR' 16) → MultiPathNet(MPN) (BMVC' 16) → **SSD** (ECCV' 16) → GBDNet (ECCV' 16) →

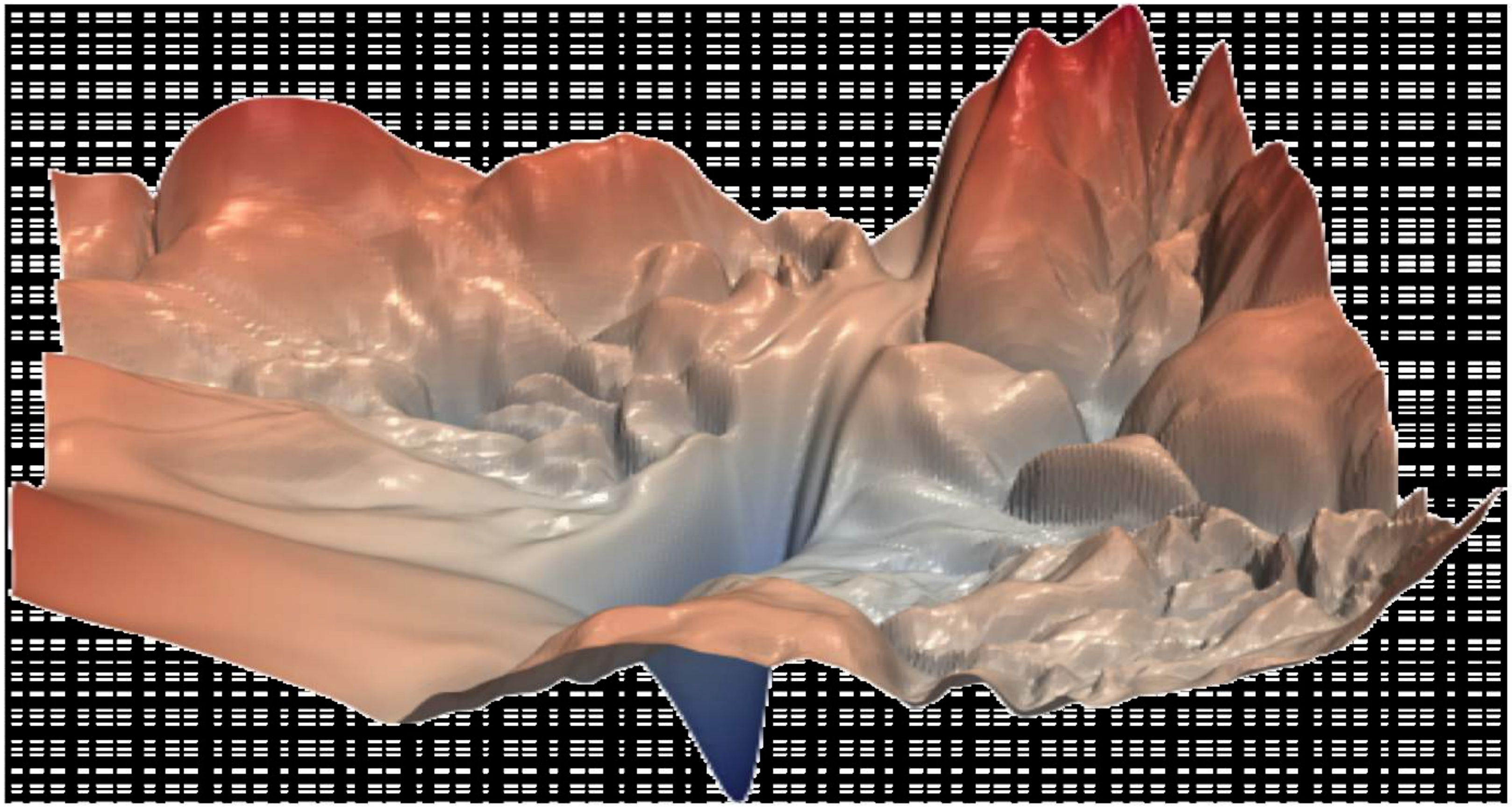
CPF (ECCV' 16) → MS-CNN (ECCV' 16) → **R-FCN** (NIPS' 16) → PVANET (NIPSW' 16) → DeepID-Net (PAMI' 16) → NoC (TPAMI' 16) → DSSD (arXiv' 17) → TDM (CVPR' 17) → **YOLO v2** (CVPR' 17) →

Feature Pyramid Net(**FPN**) (CVPR' 17) → RON (CVPR' 17) → DCN (ICCV' 17) → DeNet (ICCV' 17) → CoupleNet (ICCV' 17) → **RetinaNet** (ICCV' 17) → DSOD (ICCV' 17) →

Mask R-CNN (ICCV' 17) → SMN (ICCV' 17) → **YOLO v3** (arXiv' 18) → SIN (CVPR' 18) → STDN (CVPR' 18) → **RefineDet** (CVPR' 18) → MLKP (CVPR' 18) → Relation-Net (CVPR' 18) →

Cascade R-CNN (CVPR' 18) → RFBNet (ECCV' 18) → CornetNet (ECCV' 18) → Pelee (NIPS' 18) → MethAnchor (NIPS' 18) → SNIPER (NIPS' 18) → **M2Det** (AAAI' 19) …

Training Neural Network is Difficult!



Remember:

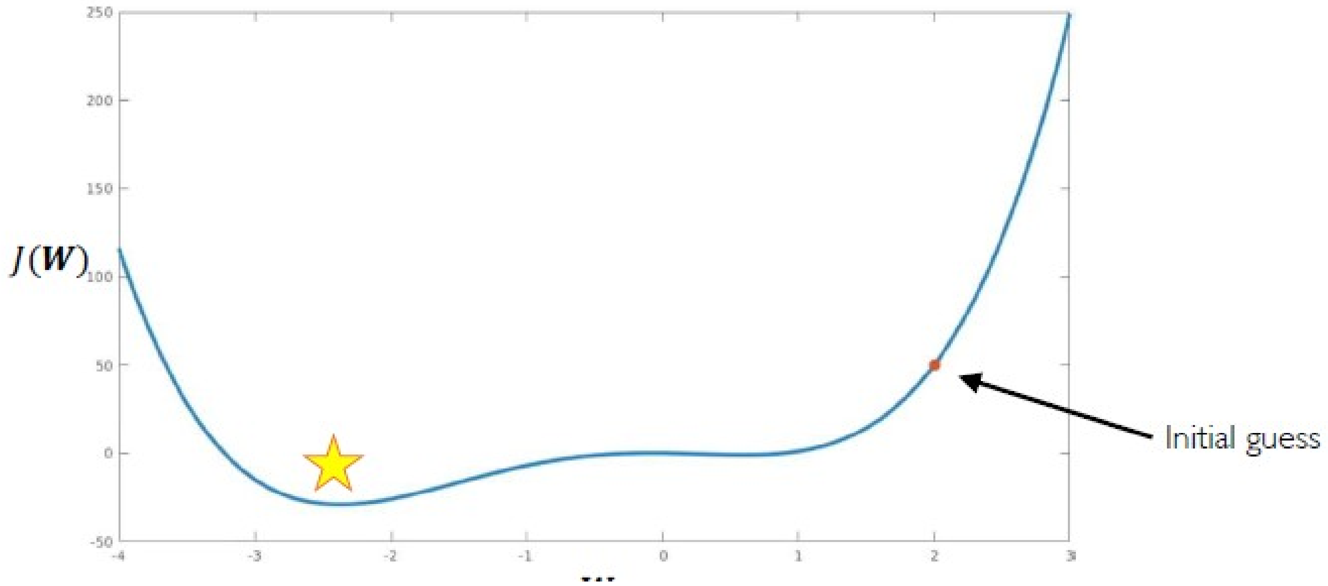
Optimization through gradient descent

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \boxed{\eta} \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

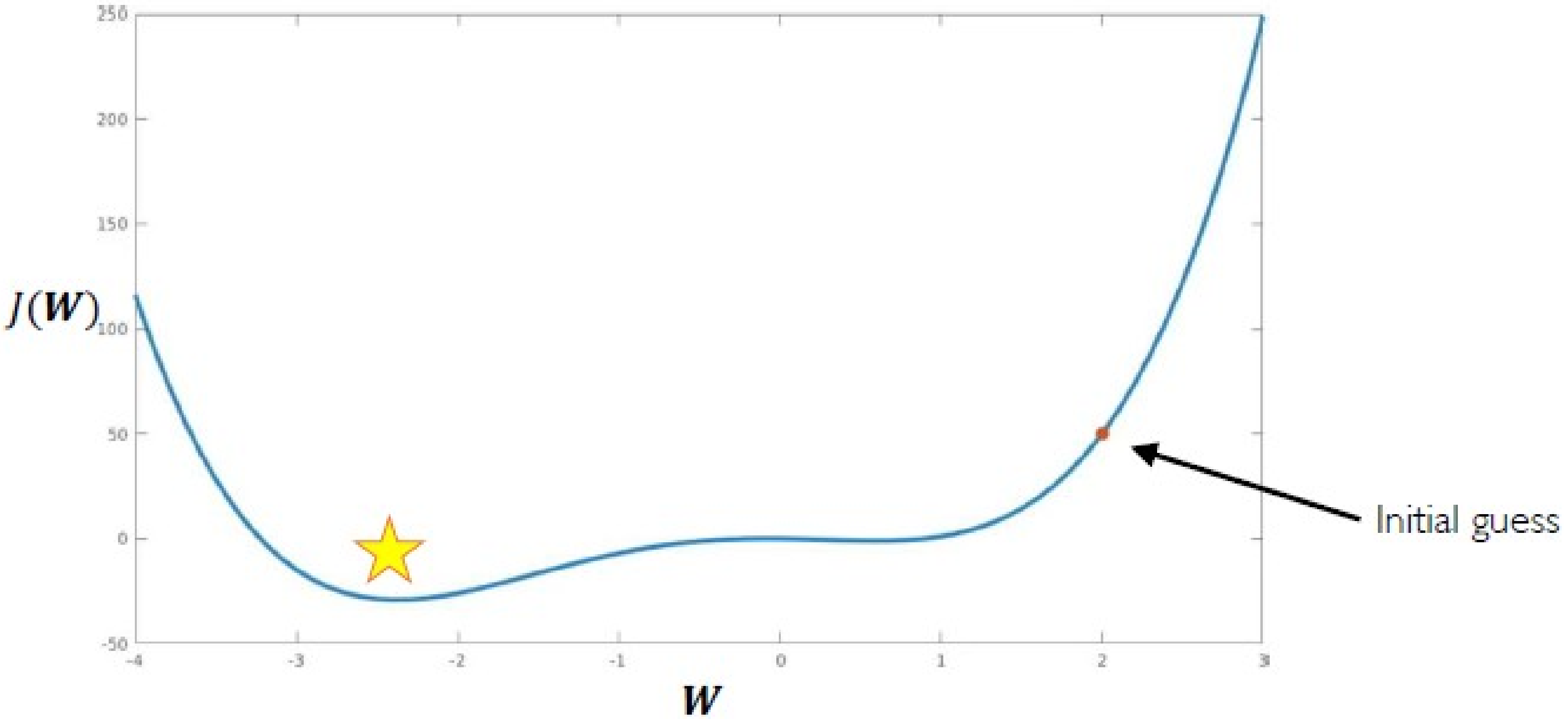
How can we set the learning rate?

Setting the Learning Rate

***Small learning rate** converges slowly and gets stuck in false local minima*



***Large learning rates** overshoot, become unstable and diverge*



How to deal with this?

Idea 1:

Try lots of different learning rates and see what works "just right"

How to deal with this?

Idea 1:

Try lots of different learning rates and see what works "just right"

Idea 2:

Do something smarter!
Design an adaptive learning rate that "adapts" to the landscape

Adaptive Learning Rates

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
    - how large gradient is
    - how fast learning is happening
    - size of particular weights
    - etc...

Adaptive Learning Rate Algorithms

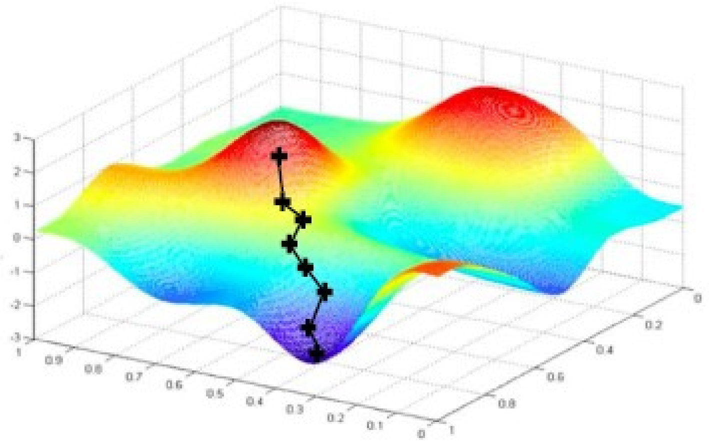
• Momentum	`tf.train.MomentumOptimizer`	Qian et al. "On the momentum term in gradient descent learning algorithms." 1999.
• Adagrad	`tf.train.AdagradOptimizer`	Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.
• Adadelta	`tf.train.AdadeltaOptimizer`	Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.
• Adam	`tf.train.AdamOptimizer`	Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.
• RMSProp	`tf.train.RMSPropOptimizer`	

Additional details: http://ruder.io/optimizing-gradient-descent/

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

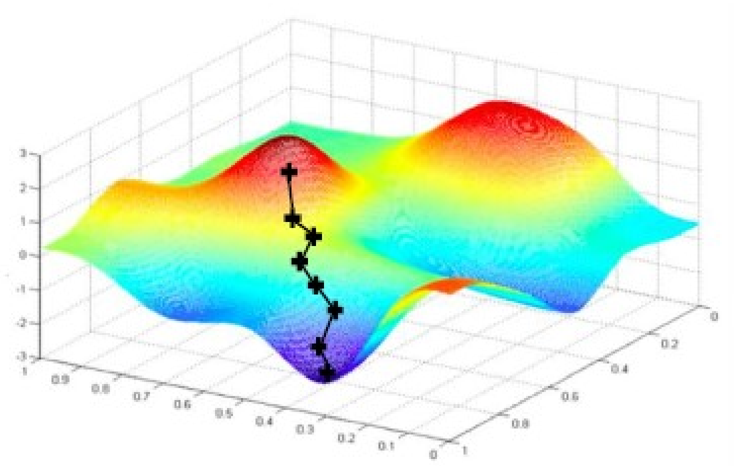


Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

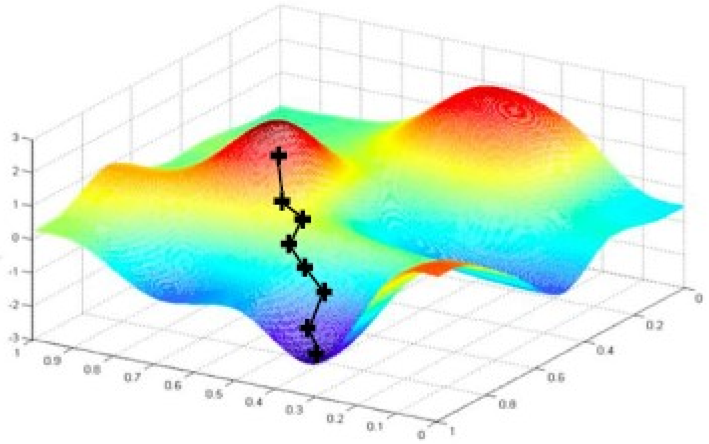
Can be very computational to compute!



Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

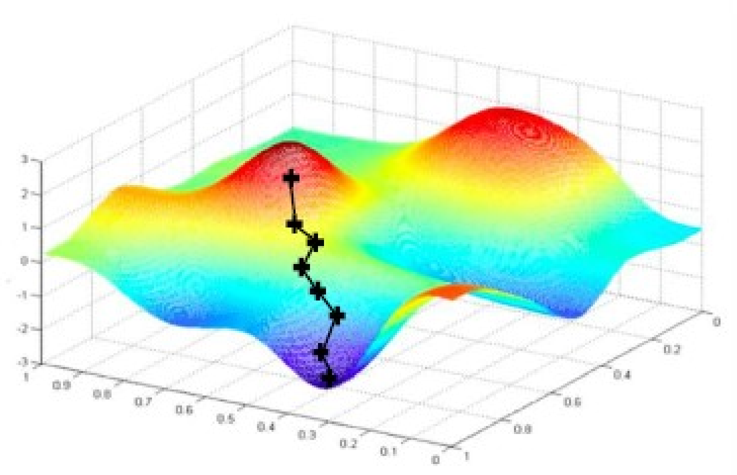


Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\boxed{\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

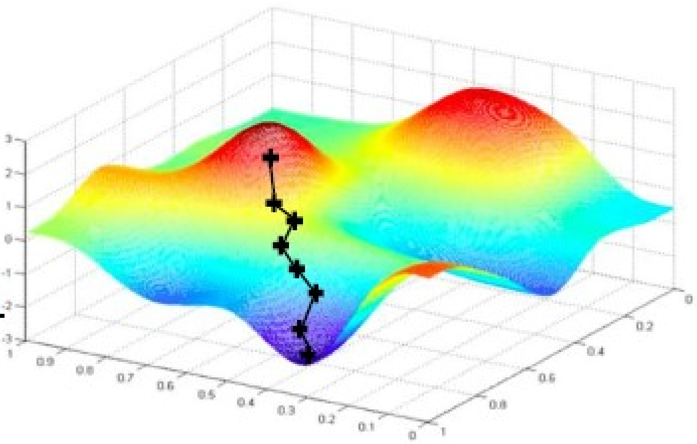
Easy to compute but **very noisy** (stochastic)!



Stochastic Gradient Descent

Algorithm

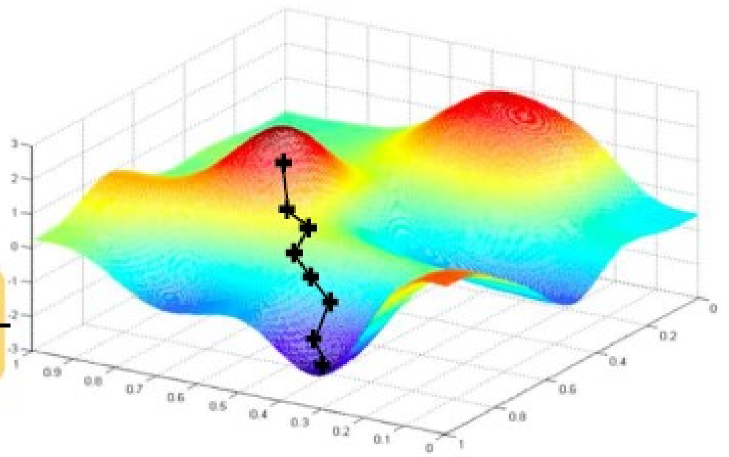
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Stochastic Gradient Descent

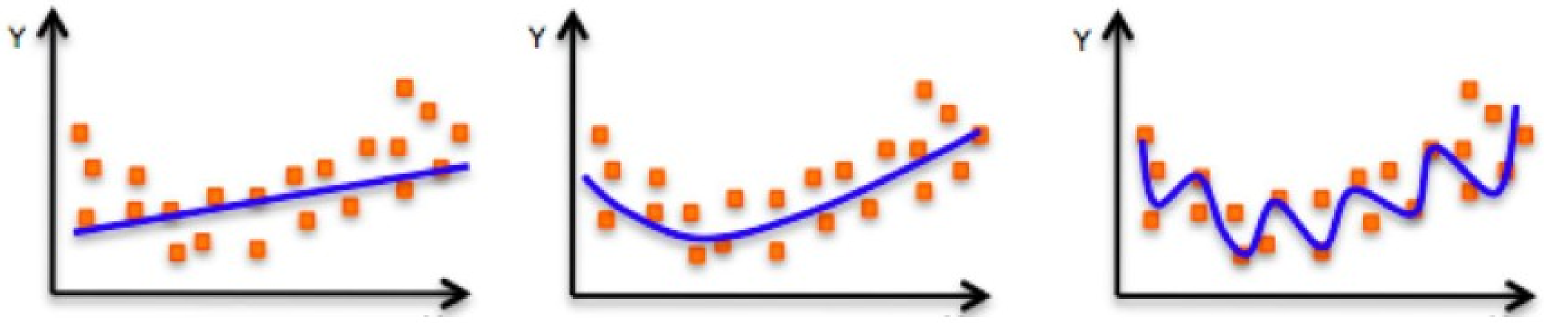
Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Fast to compute and a much better estimate of the true gradient!

The Problem of Overfitting



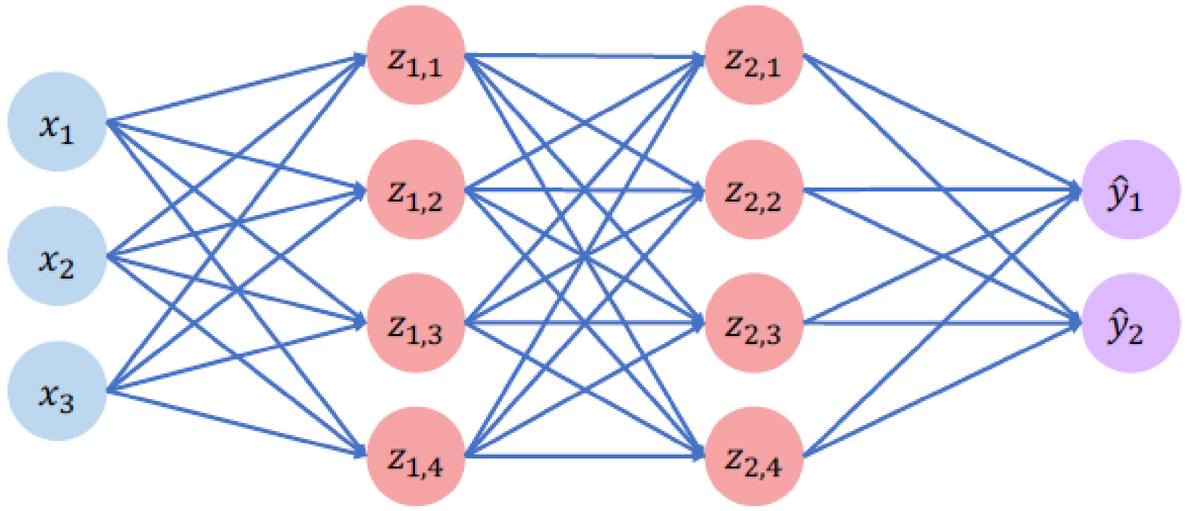
Underfitting
Model does not have capacity to fully learn the data

← **Ideal fit** →

Overfitting
Too complex, extra parameters, does not generalize well

Regularization 1: Dropout

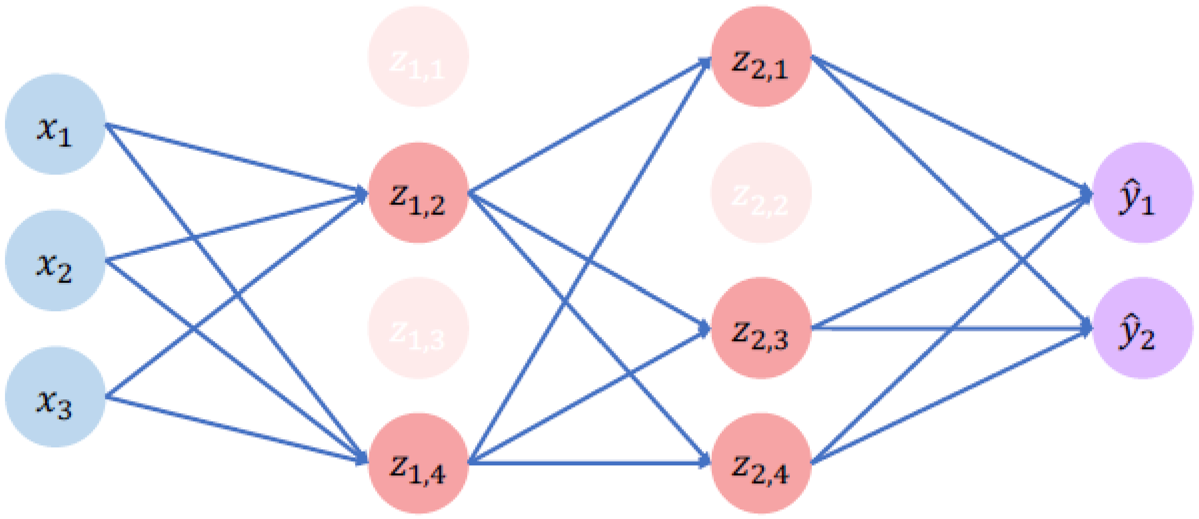
- During training, randomly set some activations to 0



Regularization 1: Dropout

- During training, randomly set some activations to 0
    - Typically 'drop' 50% of activations in layer
    - Forces network to not rely on any 1 node

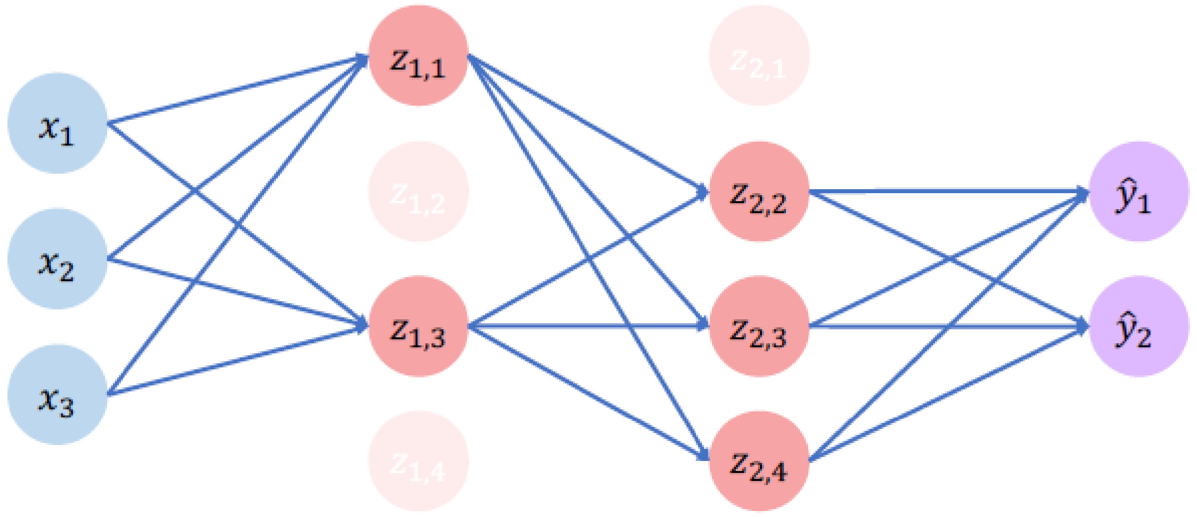
```
tf.keras.layers.Dropout(p=0.5)
```



Regularization 1: Dropout

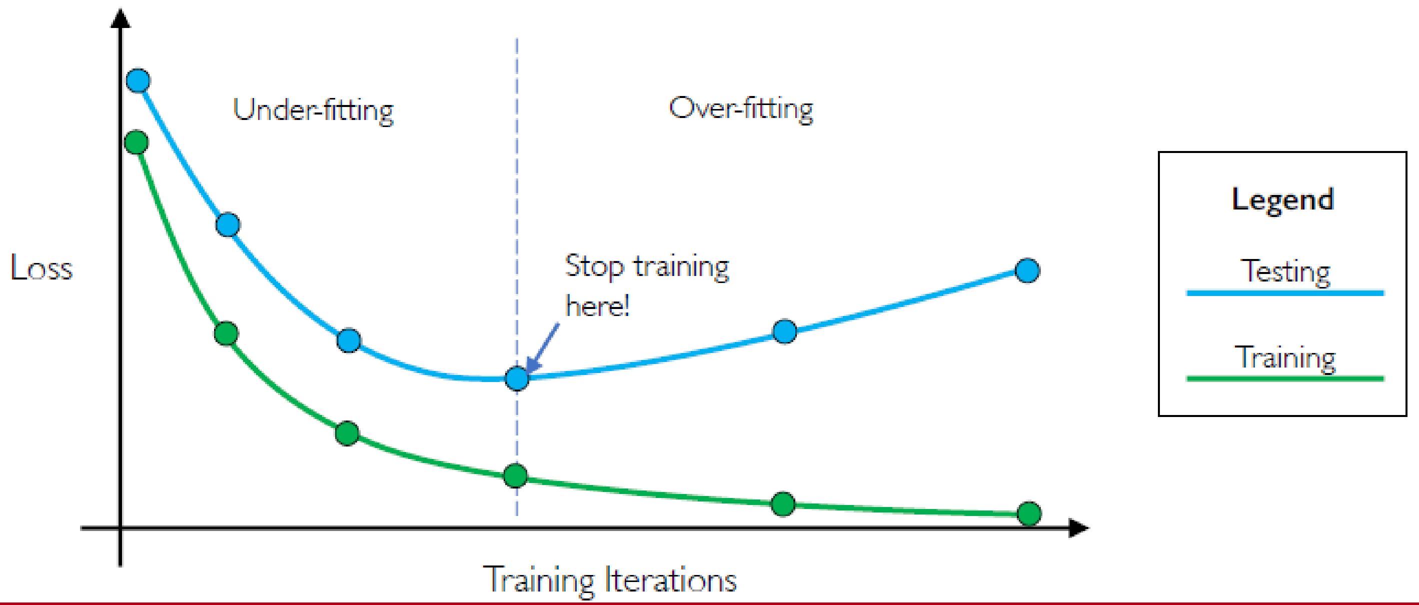
- During training, randomly set some activations to 0
    - Typically 'drop' 50% of activations in layer
    - Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```



Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



Summary

- CNN is the best choice for many computer vision problems, but its limitations are also obvious:
  - Computational cost is very heavy (memory, time);
  - Supervised approach: needs lots of labeled data.
- Other options
  - Model distribution of input data instead of keeping their spatial information explicitly.
  - Unsupervised approach: can use unlabeled data
    - RBM, DBM, Deep Auto-Encoder, etc.