EE7207: Lecture 12

Deep Neural Network in Application

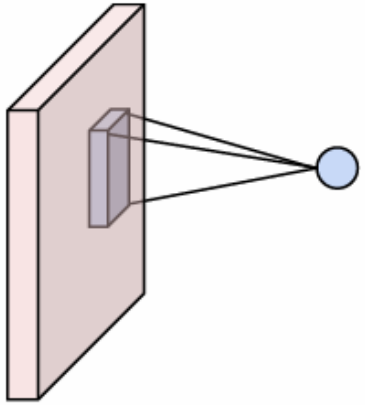
Lecturer: Zhang Handuo
https://handuo.top/
handuo.zhang@shanda.com

Outline

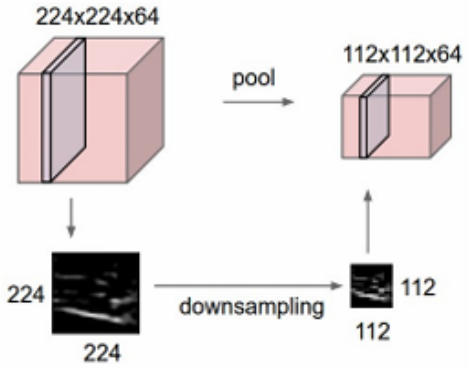
- 1 Tricks for Better Performance in CNN
- 2 Classic Network Modules
- 3 Computer Vision Tasks
- 4 Advanced Networks

CNN Common Components

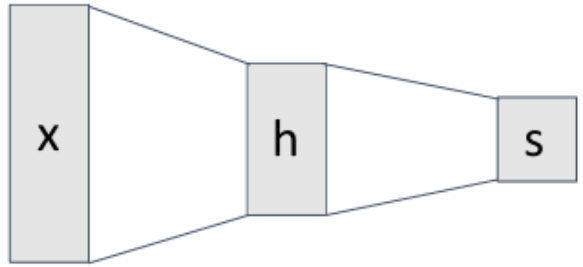
Convolution Layers



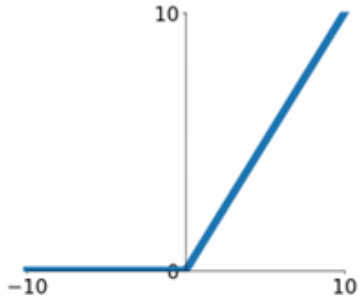
Pooling Layers



Fully-Connected Layers



Activation Function



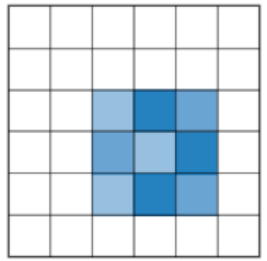
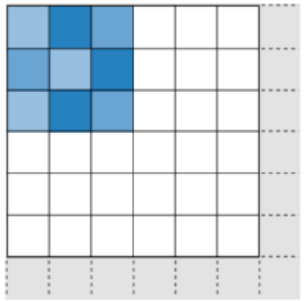
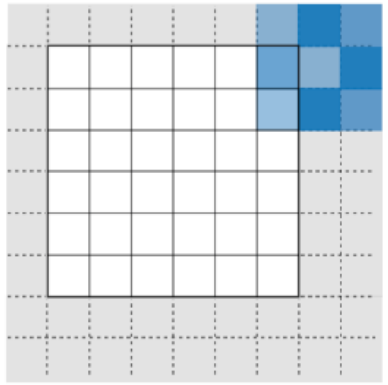
Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Hyper-Parameters for CNN

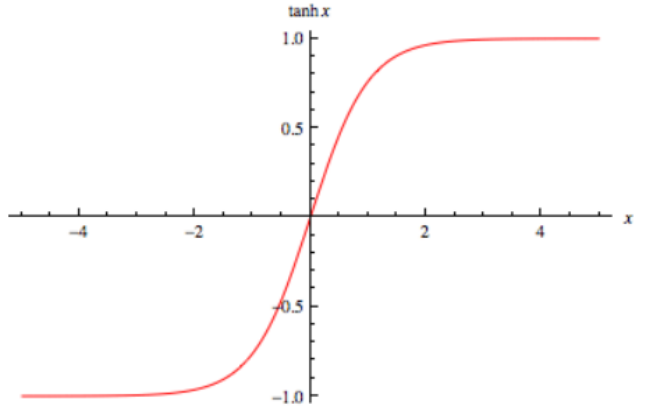
- Network Structure
  - #Filters
  - Filter Size
  - Stride
  - Zero-Padding
  - Pooling Size
  - Activation Functions
- Training
  - Dropout Rate
  - Learning rate
  - Batch Size
  - Optimizer: SGD, SGD with momentum, Adam, AdaGrad, RMSprop
  - Weight Initialization
    - Random Initialization
    - Xavier / Glorot Initialization
    - He Initialization

Padding

Mode	Valid	Same	Full
Value	$P = 0$	$P_{\text{start}} = \left\lfloor \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rfloor$ $P_{\text{end}} = \left\lceil \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$	$P_{\text{start}} \in [\![0, F-1]\!]$ $P_{\text{end}} = F - 1$
Illustration			
Purpose	• No padding • Drops last convolution if dimensions do not match	• Padding such that feature map size has size $\left\lceil \frac{I}{S} \right\rceil$ • Output size is mathematically convenient • Also called 'half' padding	• Maximum padding such that end convolutions are applied on the limits of the input • Filter 'sees' the input end-to-end

1.1 Initialization

- Bad initialization results in convergence to bad regions [1]
  - Because of the nonconvexity, global minimum cannot be attained.
- Rule of thumb :
  - The trainable parameters (e.g., the filters of ConvNet) are randomly initialized
  - All-zero and all-one initializations are bad ideas. (all gradients are the same)
  - Bad scaling leads to terrible results. Small random numbers don't work for deep netwo
  - Pretrained parameters can be very good initialization.
- Xavier initialization, also known as Glorot initialization [2]
  - Make sure variance of output = variance of input
  - This method only works on the assumption that the activations are zero-centered.
    - $f(x) = -f(-x), f'(0) = 1$
  - This is the case for tanh and softsign activation functions.
  - Invalid for sigmoid and ReLU (activations collapse to zero)



tanh graphed on wolfram alpha

torch.nn.init.xavier_uniform_(tensor, gain=1.0, generator=None)

$$W^i \sim U(-a, a), \quad a = \beta \times \sqrt{\frac{6}{n_i + n_{i+1}}}$$

torch.nn.init.xavier_normal_(tensor, gain=1.0, generator=None)

$$W^i \sim N(0, \sigma^2), \quad \sigma = \beta \times \sqrt{\frac{2}{n_i + n_{i+1}}}$$

[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016. Page 301

[2] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2010.

1.1 Initialization

- Nowadays, ReLu has become the activation function of choice in many architectures, and it certainly does not have a mean of 0.
- Kaiming Initialization [1], also known as He/MSRA initialization:
  - Designed for CNNs, but also can be applied to other DNNs

torch.nn.init.kaiming_uniform_(*tensor*, *a=0*, *mode='fan_in'*, *nonlinearity='leaky_relu'*, *generator=None*) [SOURCE]

Fill the input *Tensor* with values using a Kaiming uniform distribution.

The method is described in *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* - He, K. et al. (2015). The resulting tensor will have values sampled from $\mathcal{U}(-\text{bound}, \text{bound})$ where

$$\text{bound} = \text{gain} \times \sqrt{\frac{3}{\text{fan\_mode}}}$$

Also known as He initialization.

Parameters

- **tensor** (*Tensor*) – an n-dimensional *torch.Tensor*
- **a** (*float*) – the negative slope of the rectifier used after this layer (only used with `'leaky_relu'`)
- **mode** (*str*) – either `'fan_in'` (default) or `'fan_out'`. Choosing `'fan_in'` preserves the magnitude of the variance of the weights in the forward pass. Choosing `'fan_out'` preserves the magnitudes in the backwards pass.
- **nonlinearity** (*str*) – the non-linear function (*nn.functional* name), recommended to use only with `'relu'` or `'leaky_relu'` (default).
- **generator** (*Optional*[*Generator*]) – the torch Generator to sample from (default: None)

torch.nn.init.kaiming_normal_(*tensor*, *a=0*, *mode='fan_in'*, *nonlinearity='leaky_relu'*, *generator=None*) [SOURCE]

Fill the input *Tensor* with values using a Kaiming normal distribution.

The method is described in *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* - He, K. et al. (2015). The resulting tensor will have values sampled from $\mathcal{N}(0, \text{std}^2)$ where

$$\text{std} = \frac{\text{gain}}{\sqrt{\text{fan\_mode}}}$$

Also known as He initialization.

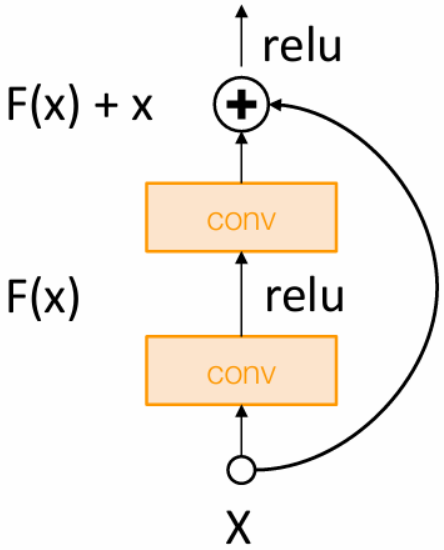
Parameters

- **tensor** (*Tensor*) – an n-dimensional *torch.Tensor*
- **a** (*float*) – the negative slope of the rectifier used after this layer (only used with `'leaky_relu'`)
- **mode** (*str*) – either `'fan_in'` (default) or `'fan_out'`. Choosing `'fan_in'` preserves the magnitude of the variance of the weights in the forward pass. Choosing `'fan_out'` preserves the magnitudes in the backwards pass.
- **nonlinearity** (*str*) – the non-linear function (*nn.functional* name), recommended to use only with `'relu'` or `'leaky_relu'` (default).
- **generator** (*Optional*[*Generator*]) – the torch Generator to sample from (default: None)

[1] He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.

1.1 Initialization

- Nowadays, ReLu has become the activation function of choice in many architectures, and it certainly does not have a mean of 0.
- Kaiming Initialization [1], also known as He/MSRA initialization:
  - Designed for CNNs, but also can be applied to other DNNs
- For residual networks, at the start Var(F(x))=Var(x), but then Var(F(x)+x)>Var(x)
  - Variance grows with each block!



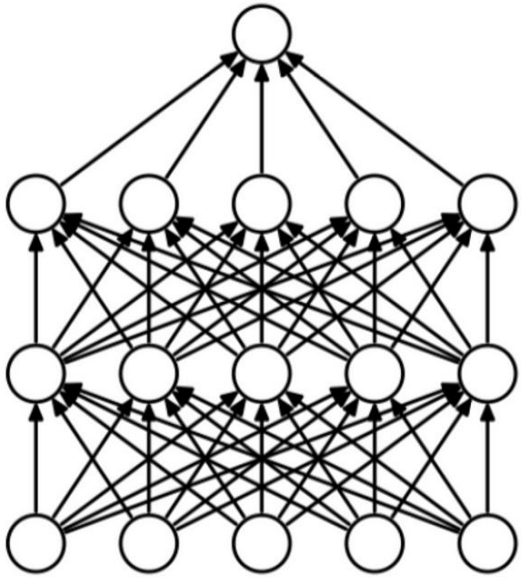
Residual Block

- **Solution:** Initialization first conv with Kaiming initialization, then initialize second conv to zero. Thus Var(x+F(x))=Var(x)
- Proper initialization is an active area of research.

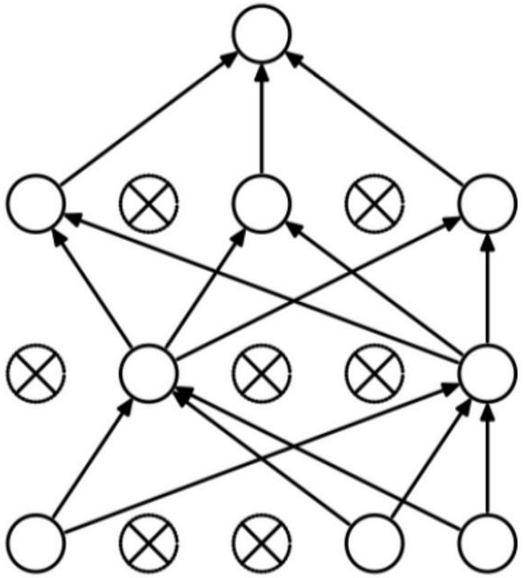
[1] He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.

1.2 Dropout for Regularization

- Train
  - In each iteration of training (1 forward + 1 backward), randomly mask 50% (or an arbitrary percentage) of the neurons.
- Prediction
  - No dropout.
  - Use all the parameters



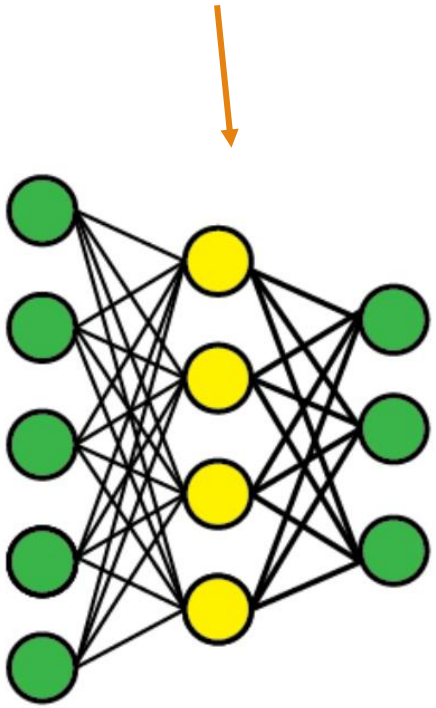
(a) Standard Neural Net



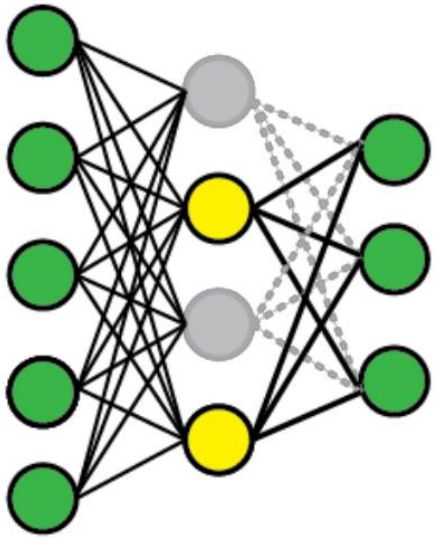
(b) After applying dropout.

1.2 Dropout Implementation

Dropout for this layer



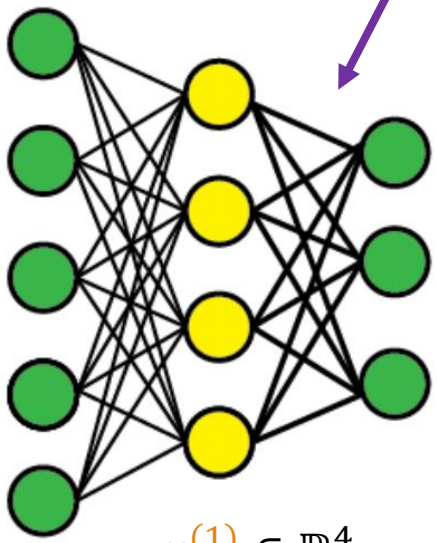
- For a batch of training samples
  - Randomly choose 50% of the neurons
  - Set the selected neurons to zeros
  - Multiply the unselected neurons by 2 (rescale)
- For another batch, do an independent random sampling.



1.2 Dropout Implementation

Regularize this layer

$x^{(0)} \in \mathbb{R}^5$



$x^{(1)} \in \mathbb{R}^4$

- **Input:** vector $x^{(0)} \in \mathbb{R}^5$
- $\mathrm{z}^{(1)} = \mathbf{W}^{(0)} x^{(0)} \in \mathbb{R}^4$
- $x^{(1)} = \max\{0, \mathrm{z}^{(1)}\} \in \mathbb{R}^4$
- **Add a dropout layer**
- $\mathrm{z}^{(2)} = \mathbf{W}^{(1)} x^{(1)} \in \mathbb{R}^3$
- **Output:** SoftMax $\left(\mathrm{z}^{(2)}\right) \in \mathbb{R}^3$

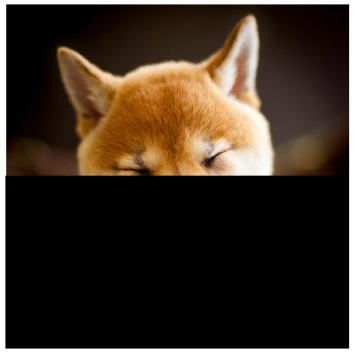
- $m \in \mathbb{R}^4$ is a random vector (Each entry is 0 or 2, with probability 50%)
- Apply m to $x^{(1)}$:

$$\tilde{x}^{(1)} = m \circ x^{(1)}$$

where is matrix entry-wise multiplication

1.2 Dropout works because …

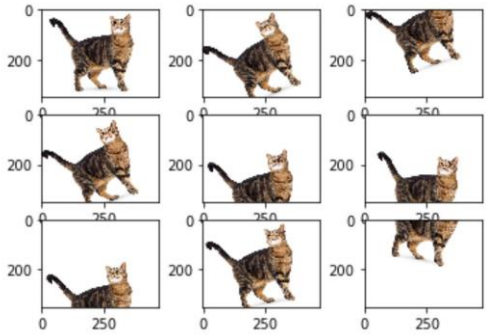
- During training, dropout forces the network to make decision based on part of the features
- Dropout is a regularization [1]
  - Alleviate overfitting
  - Like the L1 and L2 norm regularizations
  - Forces the network to have a redundant representation; prevents co-adaptation of features
- Another interpretation:
  - Dropout is training a large ensemble of models (sharing parameters)
- Later architectures use global average pooling instead of fully-connected layers, so they don't use dropout at all!

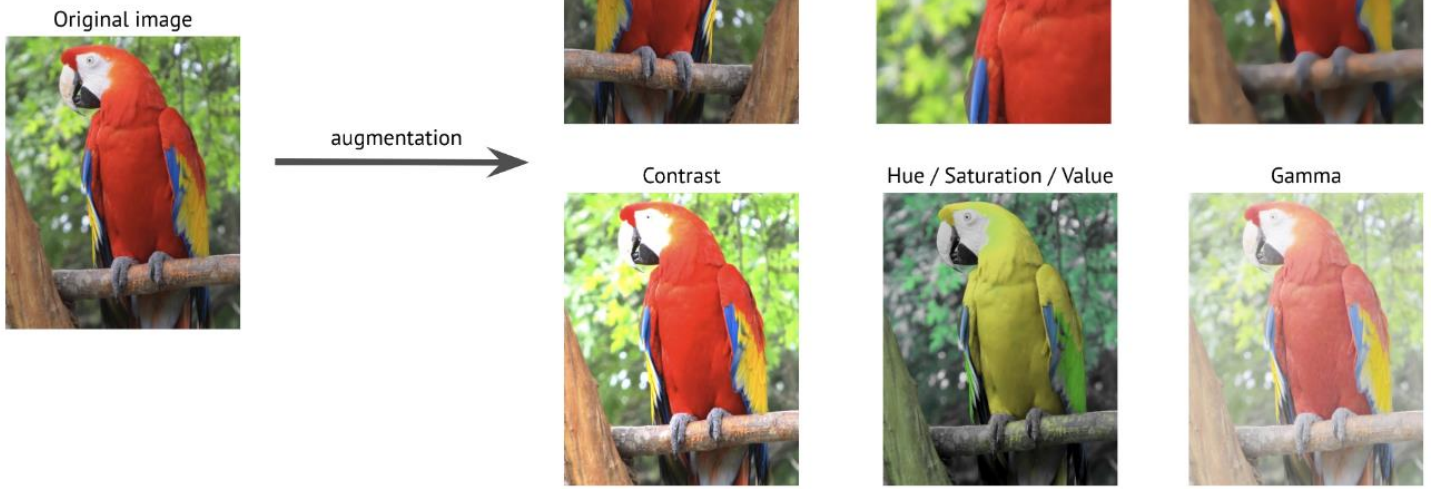


[1]. Wager, Wang, & Liang. Dropout Training as Adaptive Regularization. In *NIPS*, 2013.

1.3 Data Augmentation

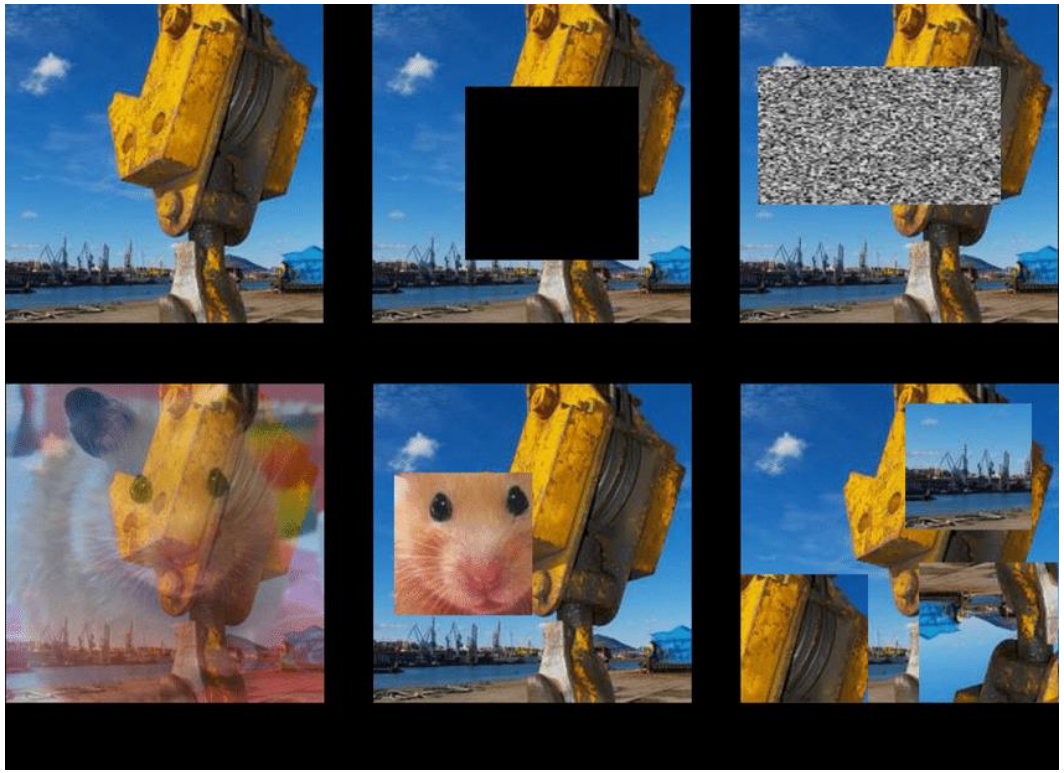
- Data augmentation: generating more training samples from existing training data
  - E.g., flip, rotation, crop, shift, cutout, add random noise



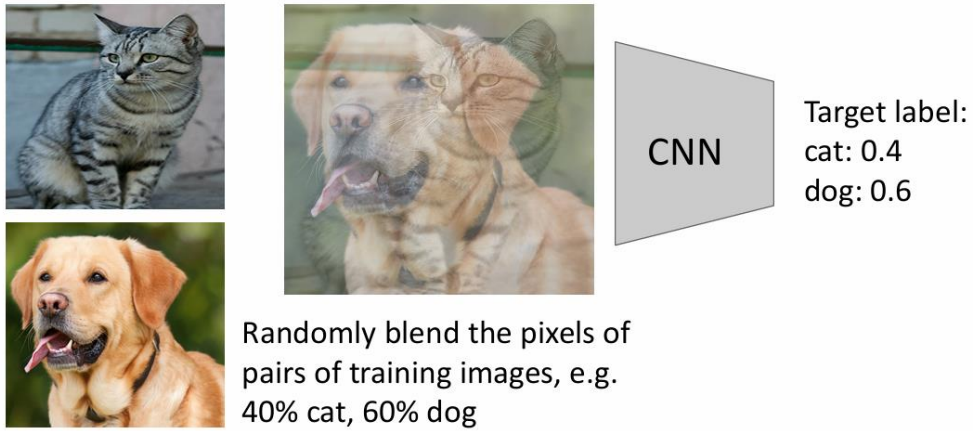


1.3 Data Augmentation

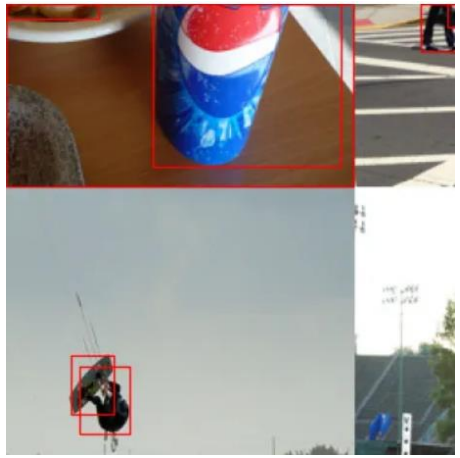
- Some advanced augmentation by adding noise



From Top-left to Bottom-right: Input image, cutout, random erasing, mixup, CutMix, Local Augment



Mixup



Mosaic

1.3 Data Augmentation

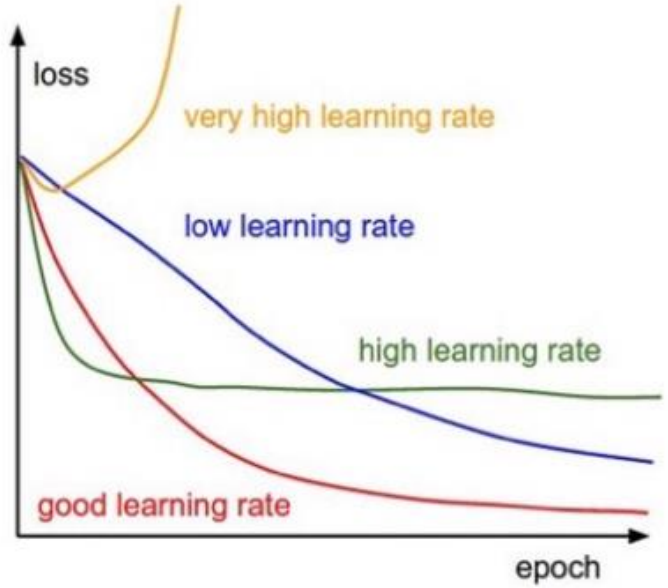
The table shows Top-1 accuracy (%) on the ImageNet validation set; higher is better.

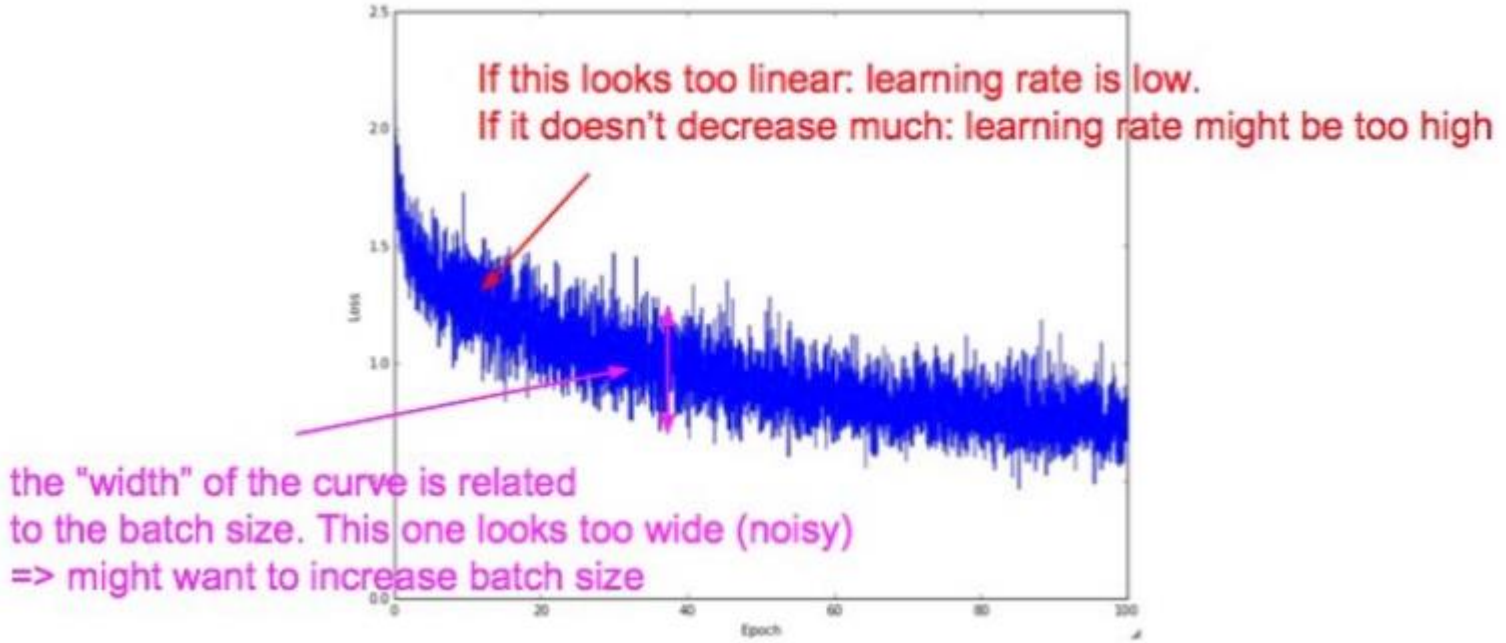
Model	Base augmentations	AutoAugment augmentations
ResNet-50	76.3	77.6
ResNet-200	78.5	80.0
AmoebaNet-B (6,190)	82.2	82.8
AmoebaNet-C (6,228)	83.1	83.5

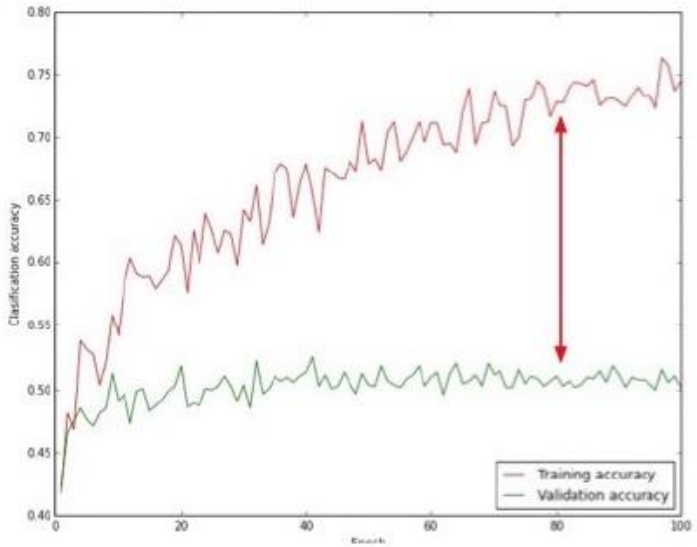
- Adding noise to the input: a special kind of augmentation
- Be careful about the transformation applied -> label preserving
  - Example: classifying ‘b’ and ‘d’; ‘6’ and ‘9’
- Recommended Libraries
  - Albumentation [1]
  - Torchvision
  - Imgaug

[1] Buslaev, Alexander, et al. "Albumentations: fast and flexible image augmentations." Information 11.2 (2020): 125.

1.4 Learning Rate Strategy







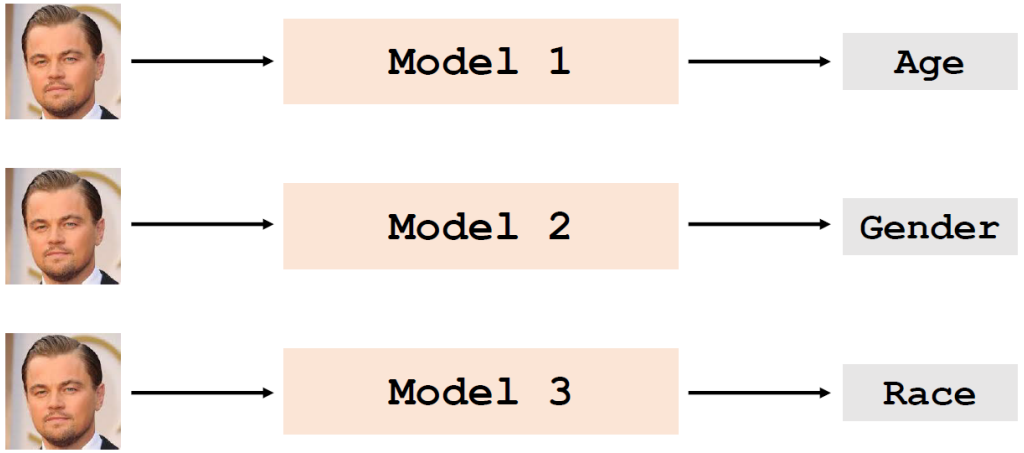
big gap = overfitting
=> increase regularization strength

no gap
=> increase model capacity

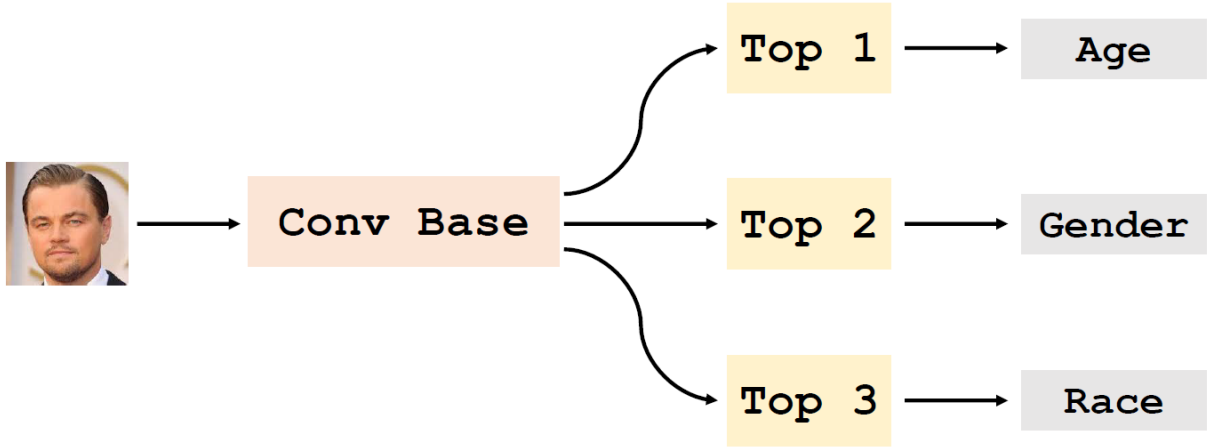
- Start with big learning rate and decay over time
- How long to train? Early Stopping and always keep track of the model snapshot that worked best on validation set

1.5 Multi-Task Learning

- Without Parameter Sharing



- Multi-Task Learning



$\text{Loss1} = (\text{AgeLabel} - \text{AgePred})^2$ Regression

$\text{Loss2} = \text{dist}(\text{GenderLabel}, \text{GenderPred})$ Binary Classification

$\text{Loss3} = \text{dist}(\text{RaceLabel}, \text{RacePred})$ Multi-class Classification

$$\text{Loss1} + \lambda \cdot \text{Loss2} + \gamma \cdot \text{Loss3}$$

1.6 Batch Normalization

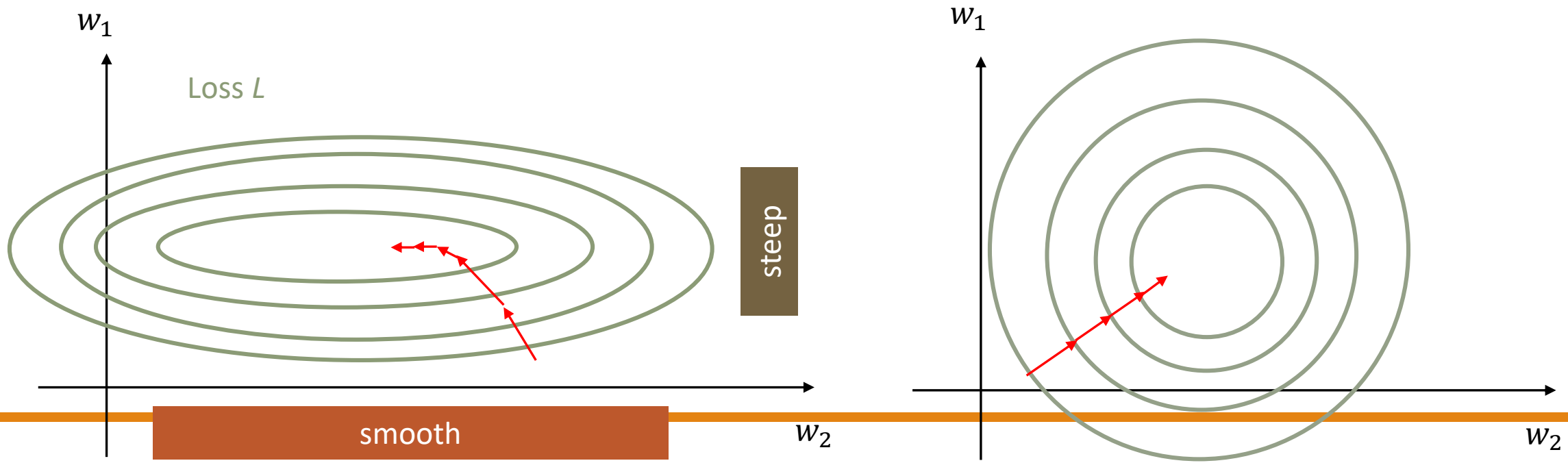
Feature Scaling for Linear Models

- E.g., person's feature vector: $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in \mathbb{R}^2$
  - The 1st dimension is one person's income (in dollars)
    - Assume it is randomly from the Gaussian distribution $N(3000, 400^2)$
  - The 2nd dimension is one person's height (in inch)
    - Assume it is randomly from the Gaussian distribution $N(69, 3^2)$
- Imagine a simple linear model: $\boldsymbol{y} = w_1\boldsymbol{x}_1 + w_2\boldsymbol{x}_2 + b$

1.6 Batch Normalization

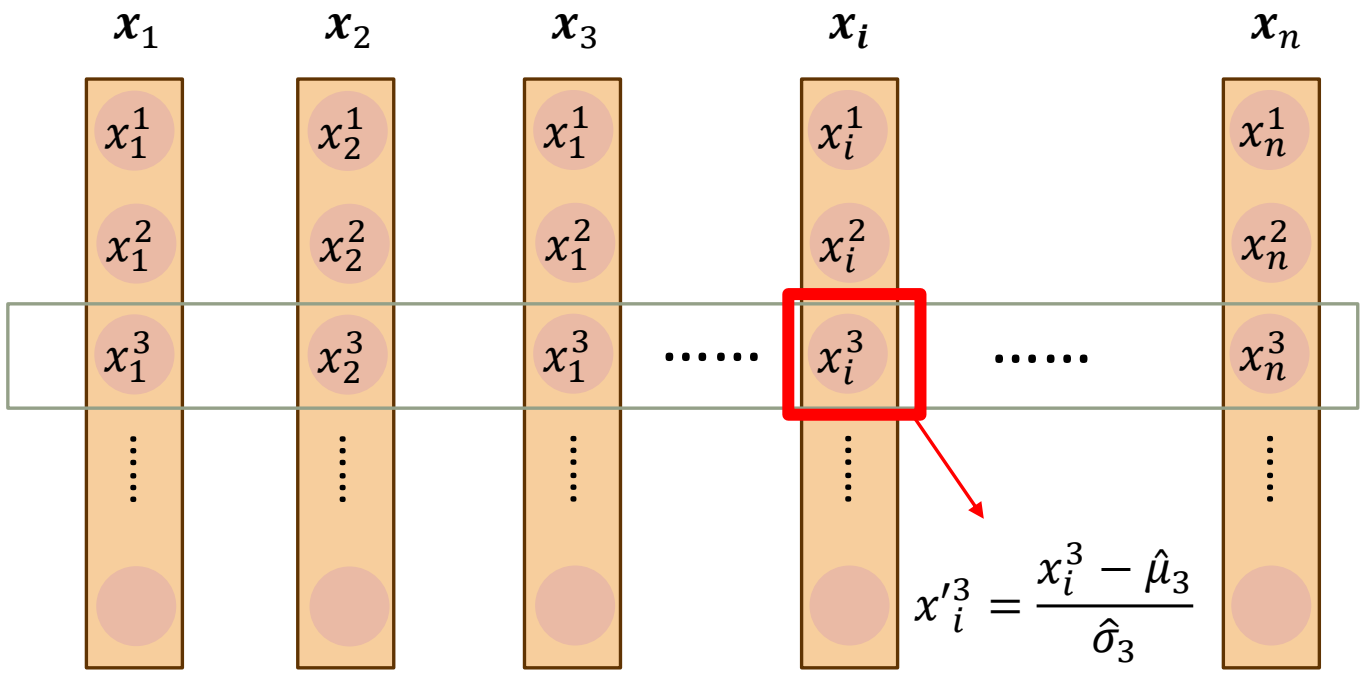
Error Surface (Loss Landscape)

- The error surface is a visualization of the loss function (L)
- **Gradient Descent Struggle**
  - The gradient with respect to w_1 will be much larger than that with respect to w_2
  - This leads to:
    - Oscillations: Gradient descent might "bounce around" excessively in the w_1 direction
    - Slow Convergence: Need a small learning rate to avoid overshooting



1.6 Batch Normalization

- Feature Scaling to the Rescue
  - **Standardization:** $x_i' = \frac{x_i - \hat{\mu}}{\hat{\sigma}}$
    - $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$ is the sample mean
    - $\hat{\sigma} = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x}_i - \hat{\mu})^2$ is the sample variance
    - After the scaling, the scaled samples $x_1', \cdots, x_n'$ have zero mean and unit variance.



1.6 Batch Normalization

Problem: What is the issue of zero-mean, unit variance for all input data?

Standardization of Hidden Layers [1]

- Let $\mathbf{x}^{(k)} \in \mathbb{R}^{n \times d}$ be the output of the *k*-th hidden layer.
- $\hat{\mu}_j \in \mathbb{R}^d$: sample mean of $\mathbf{x}^{(k)}$ evaluated on a batch of samples (per channel mean).
- $\hat{\sigma}_j \in \mathbb{R}^d$: sample std of $\mathbf{x}^{(k)}$ evaluated on a batch of samples (per channel std).
- $\gamma_j \in \mathbb{R}^d$: scaling parameter (trainable)
- $\beta_j \in \mathbb{R}^d$: shifting parameter (trainable)
- **Standardization:** $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$
  - For $j = 1, \cdots, d$.

Learning $\gamma = \sigma, \beta = \mu$ will recover the identity function!

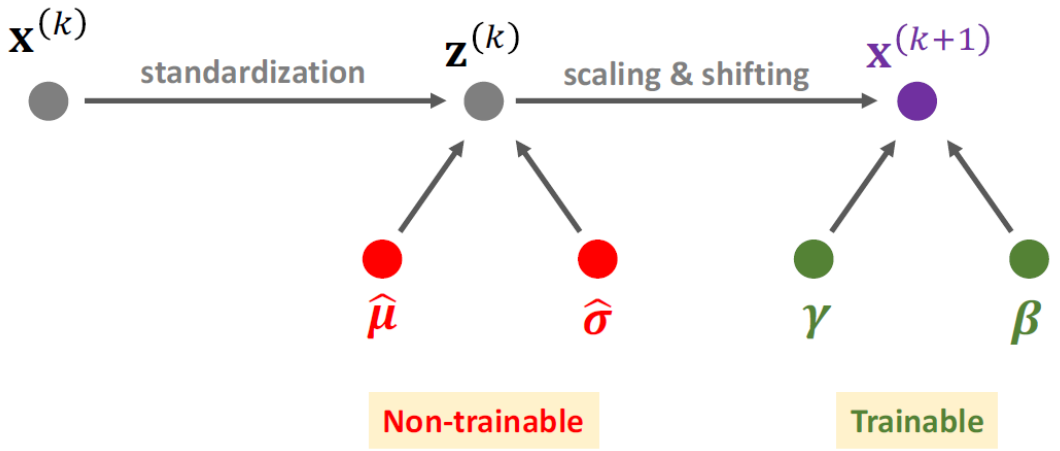
- **Scale and shift:** $x_j^{(k+1)} = z_j^{(k)} \cdot \gamma_j + \beta_j$, for $j = 1, \cdots, d$

[1] Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

1.6 Batch Normalization

Backpropagation for Batch Normalization Layer

- Let $\mathbf{x}^{(k)} \in \mathbb{R}^{n \times d}$ be the output of the *k*-th hidden layer.
- $\hat{\boldsymbol{\mu}} \in \mathbb{R}^{1 \times d}$: Non-trainable. Just record them in the forward pass;
- $\hat{\boldsymbol{\sigma}} \in \mathbb{R}^{1 \times d}$: Use them in the backpropagation.
- $\boldsymbol{\gamma} \in \mathbb{R}^{1 \times d}$: scaling parameter (trainable)
- $\boldsymbol{\beta} \in \mathbb{R}^{1 \times d}$: shifting parameter (trainable)
- **Standardization:** $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$
  - For $j = 1, \cdots, d$.
- **Scale and shift:** $x_j^{(k+1)} = z_j^{(k)} \cdot \gamma_j + \beta_j$
  - For $j = 1, \cdots, d$.



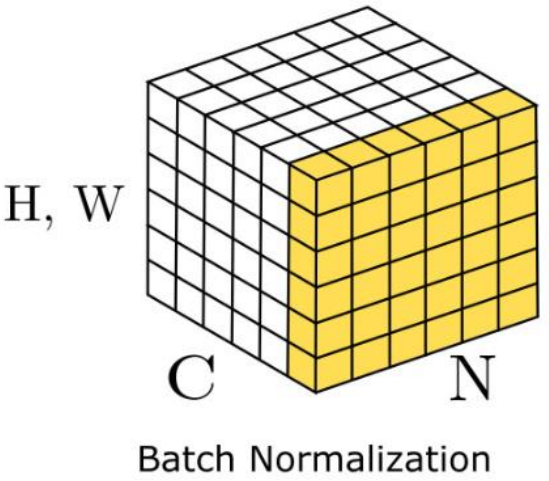
1.6 Batch Normalization

Test-Time

- Let $\mathbf{x}^{(k)} \in \mathbb{R}^{n \times d}$ be the output of the *k*-th hidden layer.
- $\hat{\mu}_j \in \mathbb{R}^d$: (Running) average of values seen during training (per channel mean).
- $\hat{\sigma}_j \in \mathbb{R}^d$: (Running) average of values seen during training (per channel std).

Batch Normalization for ConvNets (Spatial BatchNorm, BatchNorm2D)

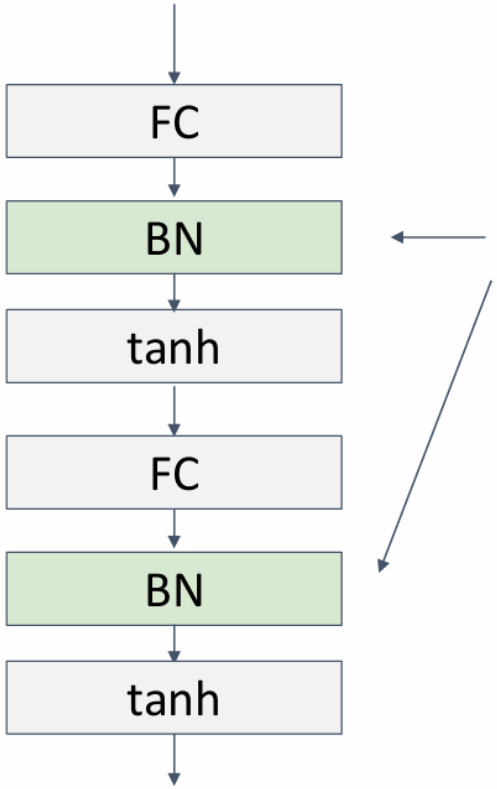
- Let $\mathbf{x}^{(k)} \in \mathbb{R}^{n \times d \times H \times W}$ be the output of the *k*-th hidden layer.
- $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}} \in \mathbb{R}^{1 \times d \times 1 \times 1}$: per channel mean and std.
- $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^{1 \times d \times 1 \times 1}$: per channel re-scaling and re-shifting parameters.



Batch Normalization

1.6 Batch Normalization

- Design of layers



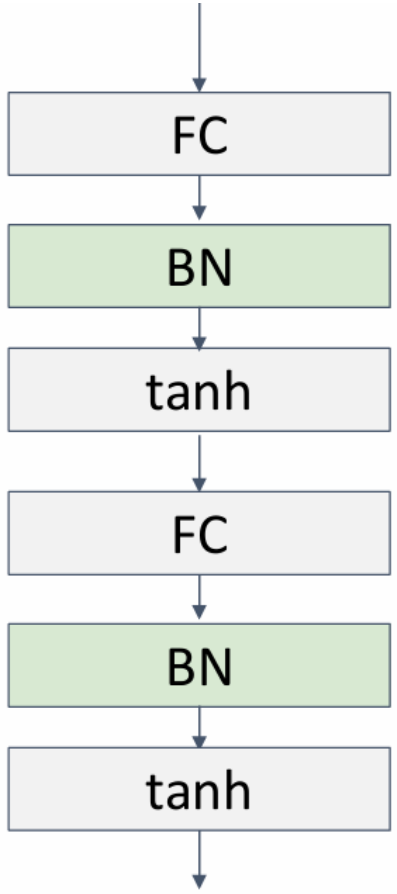
Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

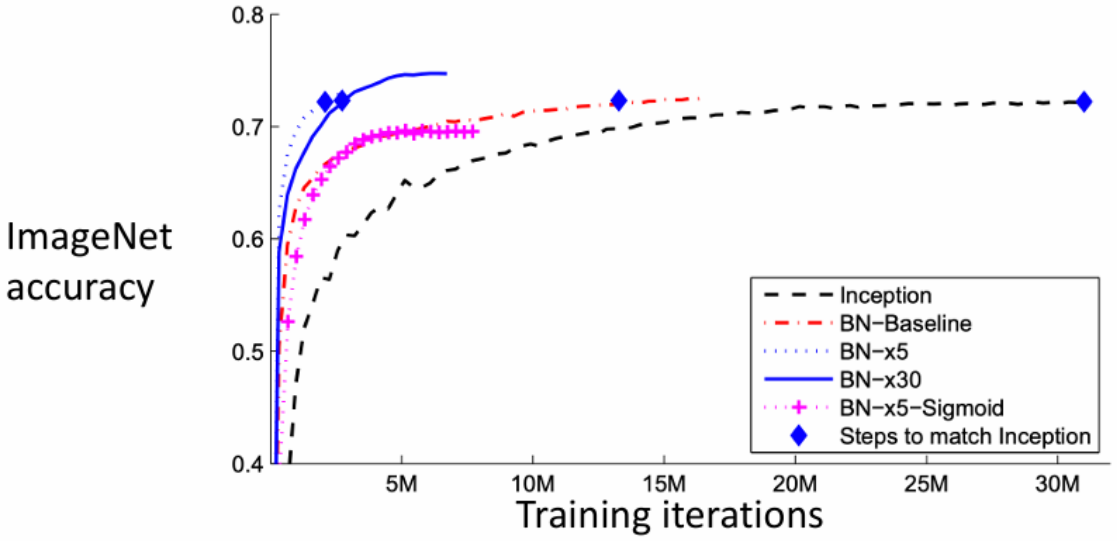
Ioffe and Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, ICML 2015

1.6 Batch Normalization

- Remarks



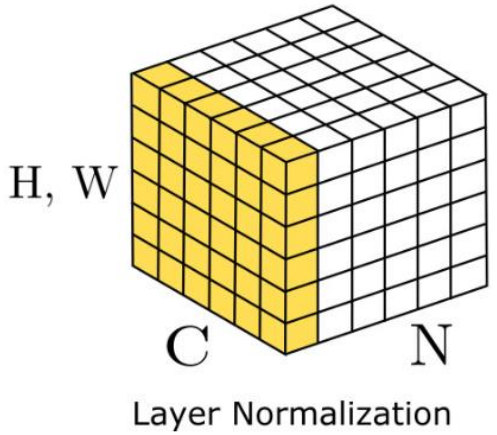
- Makes deep networks **much** easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!



Loffe and Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, ICML 2015

1.7 Layer Normalization

- Layer Normalization for fully-connected layers[1] (Used in RNNs, Transformers)
  - Let $\mathbf{x}^{(k)} \in \mathbb{R}^{n \times d}$ be the output of the *k*-th hidden layer.
  - $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\sigma}} \in \mathbb{R}^{n \times 1}$: per sample mean and std.
  - $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^{1 \times d}$



Layer Normalization

- Batch Normalization
  - Let $\mathbf{x}^{(k)} \in \mathbb{R}^{n \times d \times H \times W}$ be the output of the *k*-th hidden layer.
  - $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}} \in \mathbb{R}^{1 \times d \times 1 \times 1}$: per channel mean and std.
  - $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^{1 \times d \times 1 \times 1}$: per channel re-scaling and re-shifting parameters.

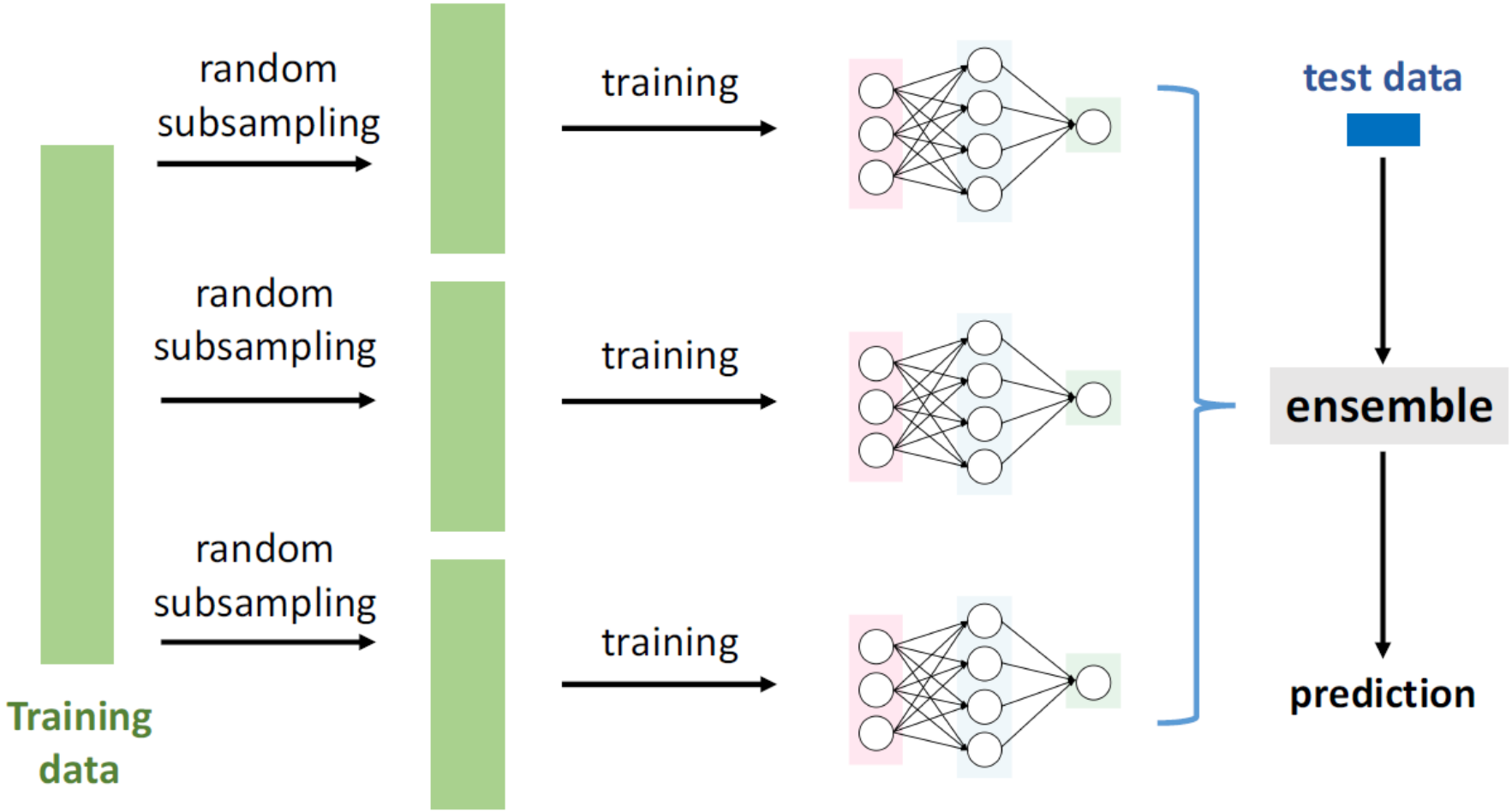
[1] Ba, Kiros, and Hinton, "Layer Normalization", arXiv 2016

1.7 Layer Normalization

Aspect	BatchNorm	LayerNorm
Normalization Scope	Normalizes across the batch for each feature. It computes the mean and variance for each feature/channel across the mini-batch.	Normalizes across features for each individual sample. It computes the mean and variance across all features for each sample.
Use Cases	Effective in CNNs and fully connected layers where batch sizes are consistent.	Suitable for recurrent neural networks (RNNs) and cases where batch size is variable or one.
Dimension Calculation	For a feature map of size [BatchSize, Channels, Height, Width], mean and variance are calculated per channel.	For an input of size [SampleSize, TokenSize, FeatureSizePerToken], mean and variance are calculated across all features for each sample.
Meaning	The transformation is trainable and specific to each neuron: different neurons can have their inputs normalized to different ranges after re-scaling and re-shifting.	The transformation is consistent across a layer: all inputs are normalized to the same range, which might limit model capacity if features differ significantly (e.g., color and size).
Parameter Learning	Each feature/channel has its own pair of trainable parameters (γ, β) for re-scaling and re-shifting.	A single pair of trainable parameters (γ, β) is shared across all features for re-scaling and re-shifting.
Advantages	• Improves gradient flow through the network. • Reduces the risk of vanishing gradient. Allows use of higher learning rates. • Reduces the sensitivity to initialization	• Decouples the batch size from the training process • Especially beneficial in models where temporal or sequential information is important.

1.8 Ensemble Methods

- Bagging (aka Bootstrap Aggregating)



1.8 Why Ensemble?

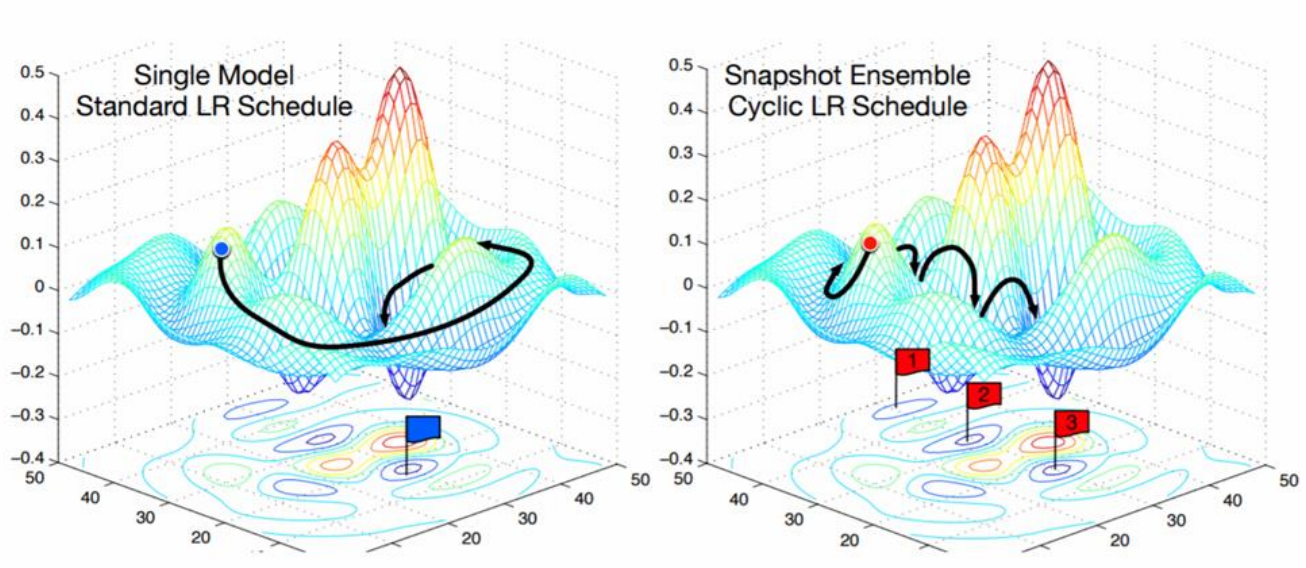
- Deep neural networks are unstable
  - Sensitive to hyper-parameters
  - Ransom initialization
  - Stochastic gradient

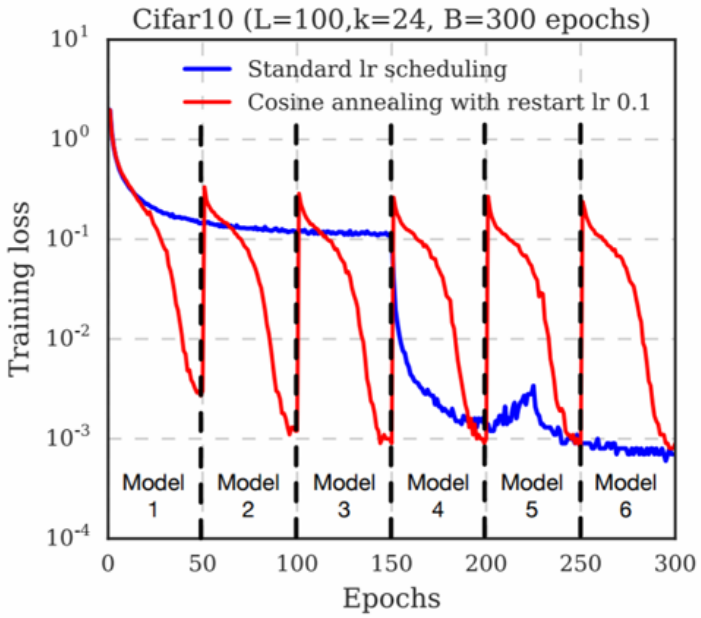
  } Different local optima

- Ensemble methods reduce variance

1.8 Model Ensembles

- Train multiple independent models
- At test time average their results
  - Take average of predicted probability distributions, then choose argmax
- Enjoy 2% extra performance
- Instead of training independent models, use multiple snapshots of a single model during training [1][2]





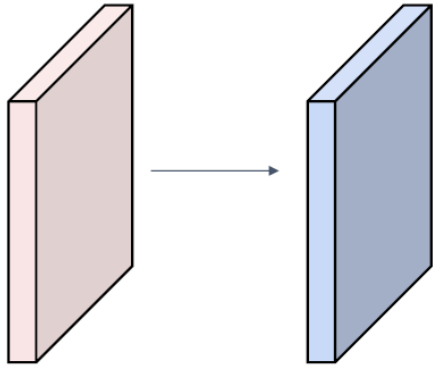
[1] Loshchilov and Hutter, “SGDR: Stochastic gradient descent with restarts”, arXiv 2016
[2] Huang et al, “Snapshot ensembles: train 1, get M for free”, ICLR 2017

Convolution Operation

- Convolution Recap

Input volume: 3x32x32
We have 10 filters (5x5 kernel size) with stride 1, pad 2

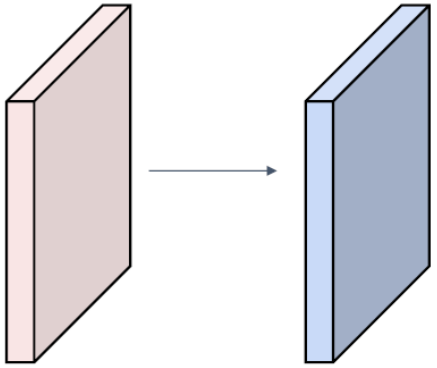
Output volume size: ?



Convolution Operation

Convolution Recap

Input volume: 3x32x32
We have 10 filters (5x5 kernel size) with stride 1, pad 2



Output volume size:
(32+2x2-5)/1 + 1 = 32 spatially,
So 10x32x32

Number of learnable parameters?

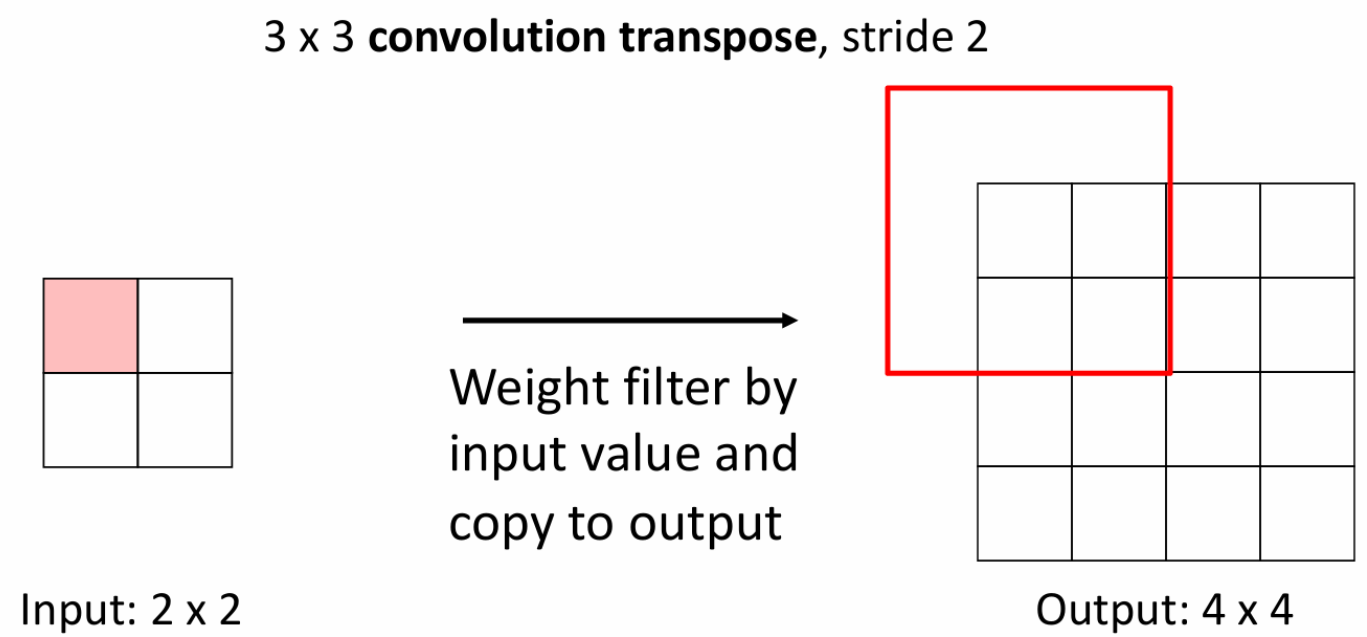
Parameters per filter: 3x5x5 + 1 (for bias) = 76
So for 10 filters, the total parameters is 10 x 76 = 760

Number of operations (multiply-add)?

There are 10x32x32=10,240 outputs
Each output is the inner product of two 3x5x5 tensors (75 elements)
So the total operations: 75x10240=**768K**

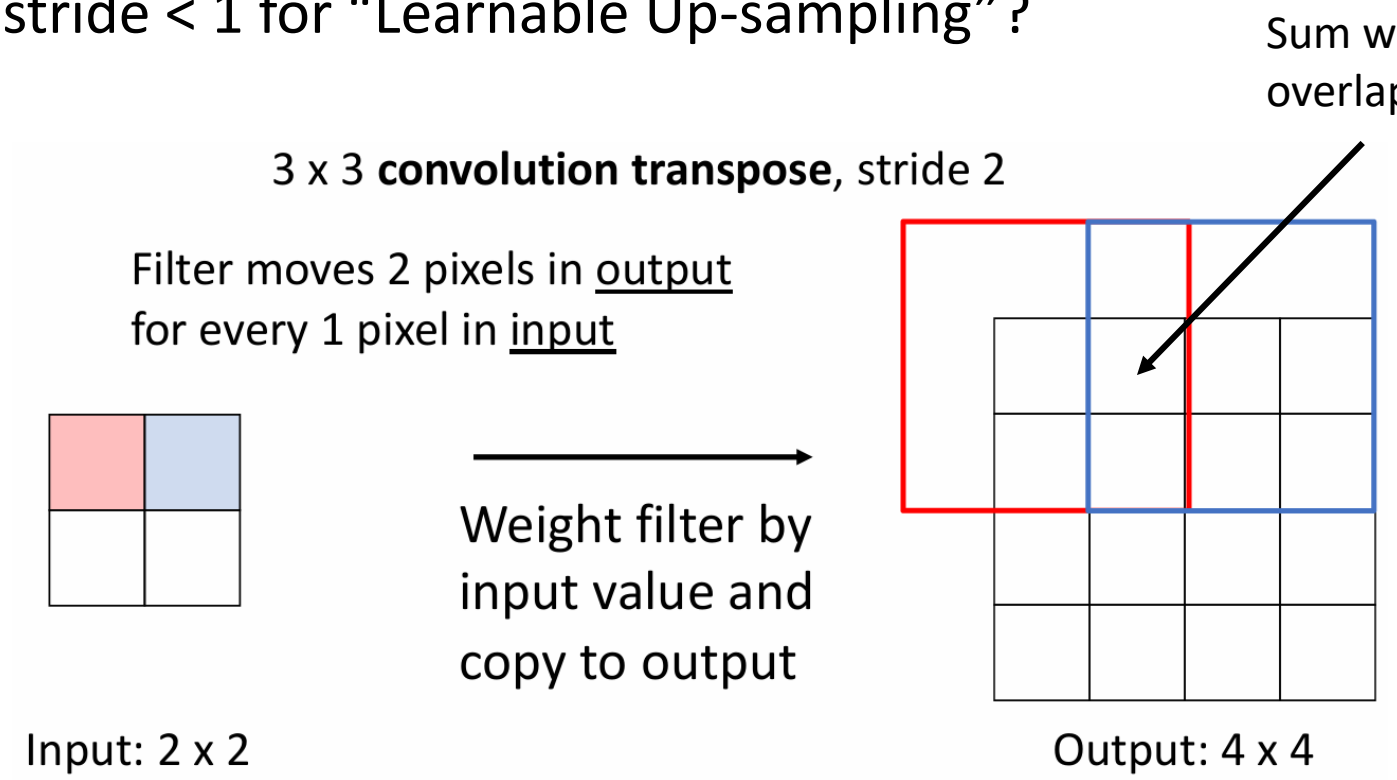
2.1 Transposed Convolution

- Convolution with stride > 1 is “Learnable Down sampling”
- Can we use stride < 1 for “Learnable Up-sampling”?



2.1 Transposed Convolution

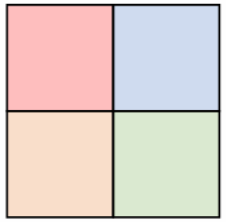
- Convolution with stride > 1 is “Learnable Down sampling”
- Can we use stride < 1 for “Learnable Up-sampling”?



2.1 Transposed Convolution

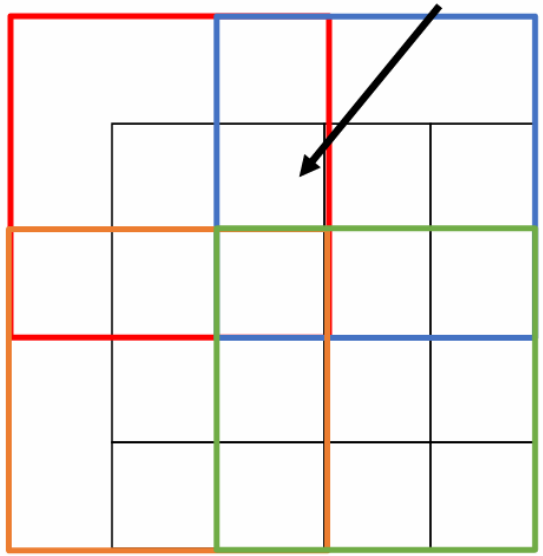
3 x 3 **convolution transpose**, stride 2

This gives 5x5 output – need to trim one pixel from top and left to give 4x4 output



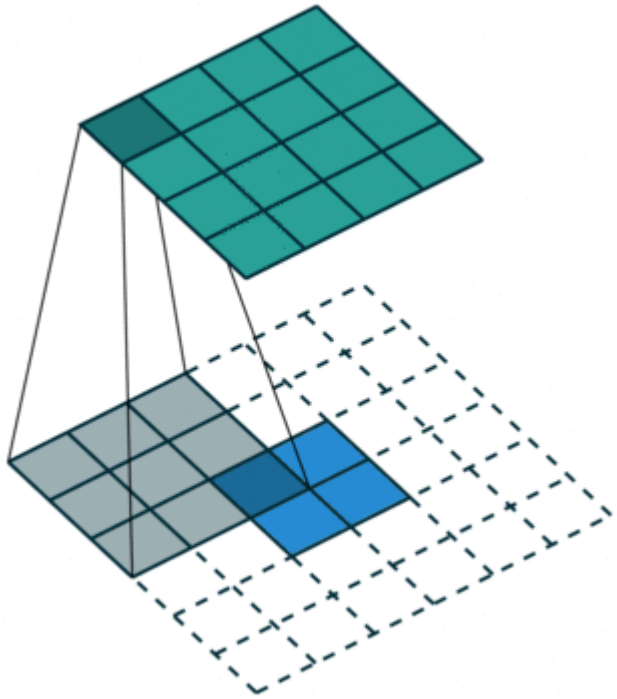
Weight filter by input value and copy to output

Sum where output overlaps



Input: 2 x 2

Output: 4 x 4



Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016).

2.1 Transposed Convolution

- Transposed Convolution, also called fractionally-strided convolution or a deconvolution

$$Output\ Size = (I - 1) \times S + K - 2P$$

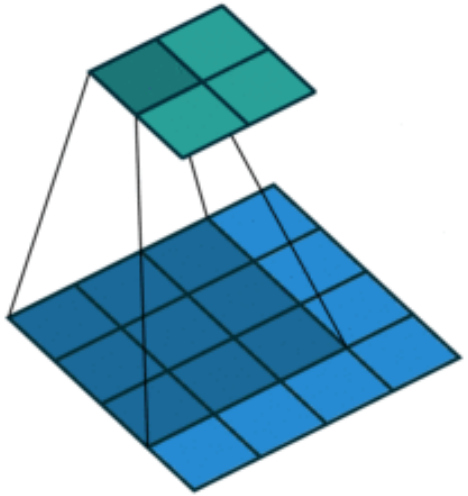
Where:

- I is the size of the input (height or width)
- S is the stride (how much the filters should shift over the input matrix)
- K is the kernel size
- P is the padding
  - Padding P is often set to 0 for transposed convolution

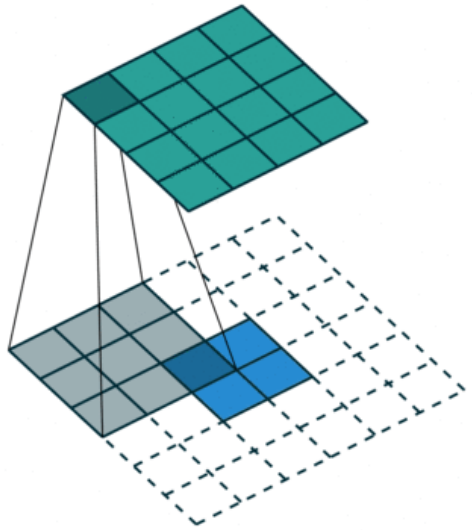
Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016).

2.1 Transposed Convolution

- Transposed convolution is not the inverse operation of convolution
- It is also a direct convolution (used for **upsampling**, in semantic segmentation and generative adversarial network)
- Blue maps are inputs, cyan maps are outputs



Conv
Padding = 0
Strides = 1



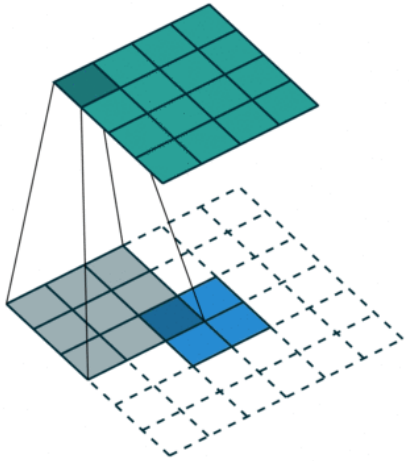
Transposed Conv
Padding = 0
Strides = 1

- The padding here refers to is the amount of cropping applied to the output of the naïve operation.
- A Padding of 0 means no cropping, and thus, the output will be expanded to its full possible size based on the kernel and stride settings.
- Note the differences between input and output padding. We normally denote the output padding the real padding value in the setting in transposed convolution.

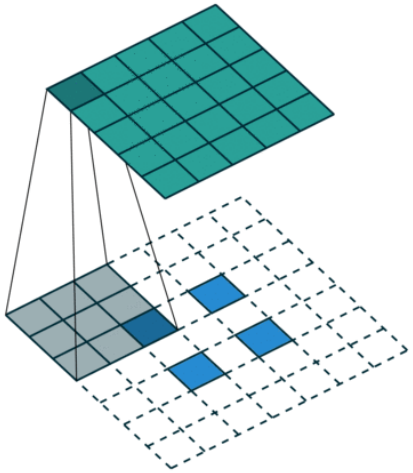
2.1 Transposed Convolution

1. Padding s-1 rows and columns between elements of the input feature map with zeros
2. Padding k-p-1 rows and columns around the input feature map with zeros
3. Flip the convolution kernel parameters vertically and horizontally (transpose)
4. Proceed normal convolution operation (padding 0, shift is always 1, regardless of the stride value s)

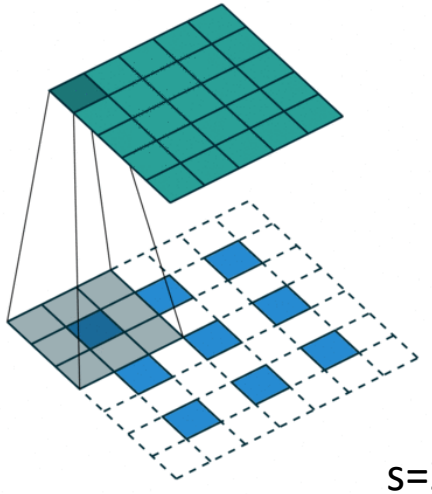
$$Output\ Size = (I - 1) \times S + K - 2P$$



s=1,p=0,k=3



s=2,p=0,k=3

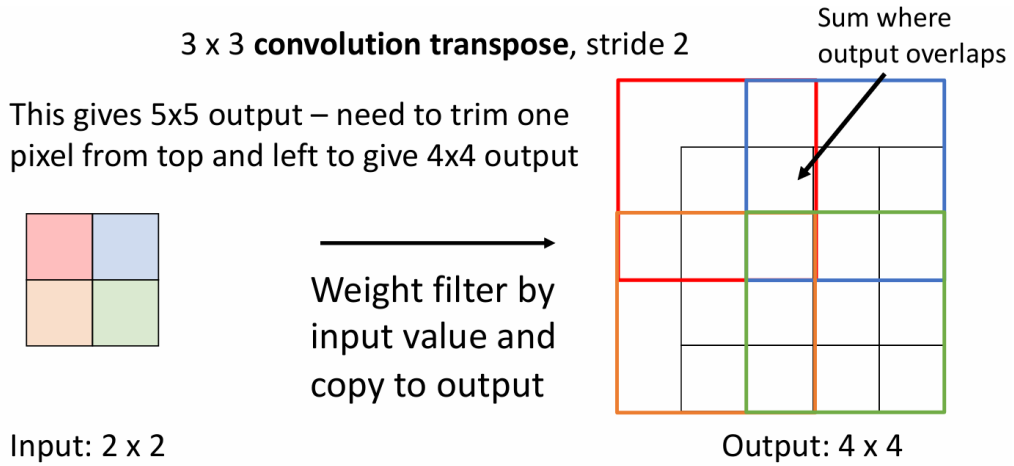


s=2,p=1,k=3

https://github.com/vdumoulin/conv_arithmetic

Why trimming for output?

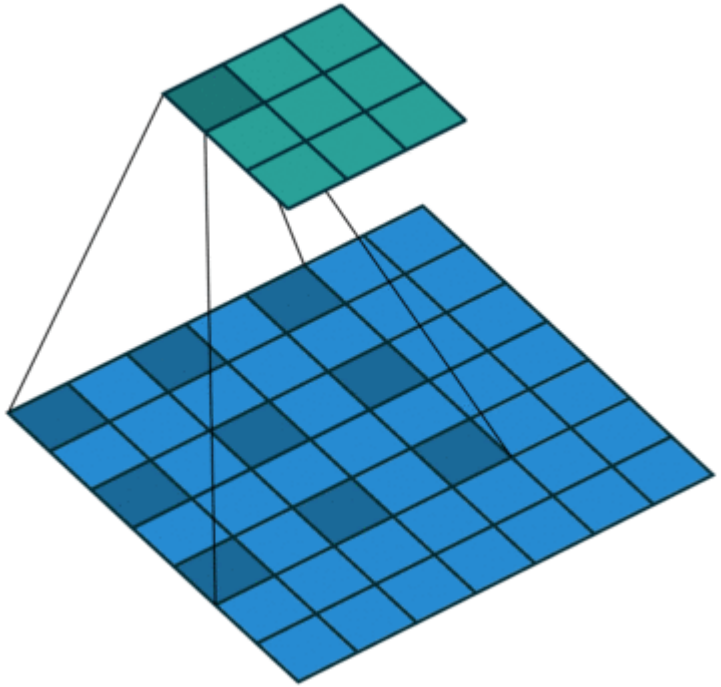
- Real Application consideration (not covered in the theory)
- Edge Effect and Cropping
  - This adjustment is effectively an intentional trimming of the expanded output to fit a desired size or to discard boundary effects that are not useful for the model.
  - In neural networks, often specific output dimensions are required for compatibility with subsequent layers . For example, 2x2 → 4x4 → 8x8
- Avoid Boundary Artifacts
  - Kernel applications that partially lie outside the input domain can introduce artifacts that might be undesirable for learning features accurately.

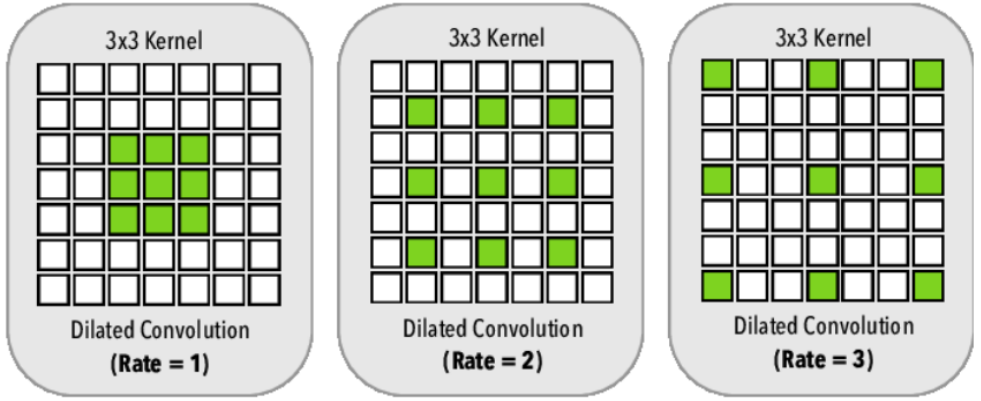


2.2 Dilated Convolution

➢ Also called Atrous convolution

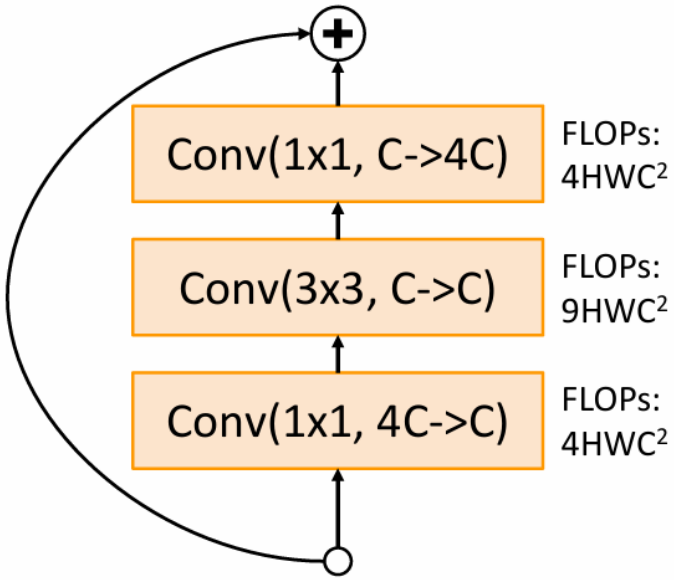
- Dilated Convolutions are a type of convolution that “inflate” the kernel by inserting holes between the kernel elements.
  - An additional parameter l (dilation rate) indicates how much the kernel is widened.
  - There are usually spaces inserted between kernel elements
- Advantages
  - Increase receptive field without adding additional parameters/calculations
  - Capture multi-scale context information
- Disadvantages
  - Gridding issue
  - Making pooling less robust





2.3 Grouped Convolution

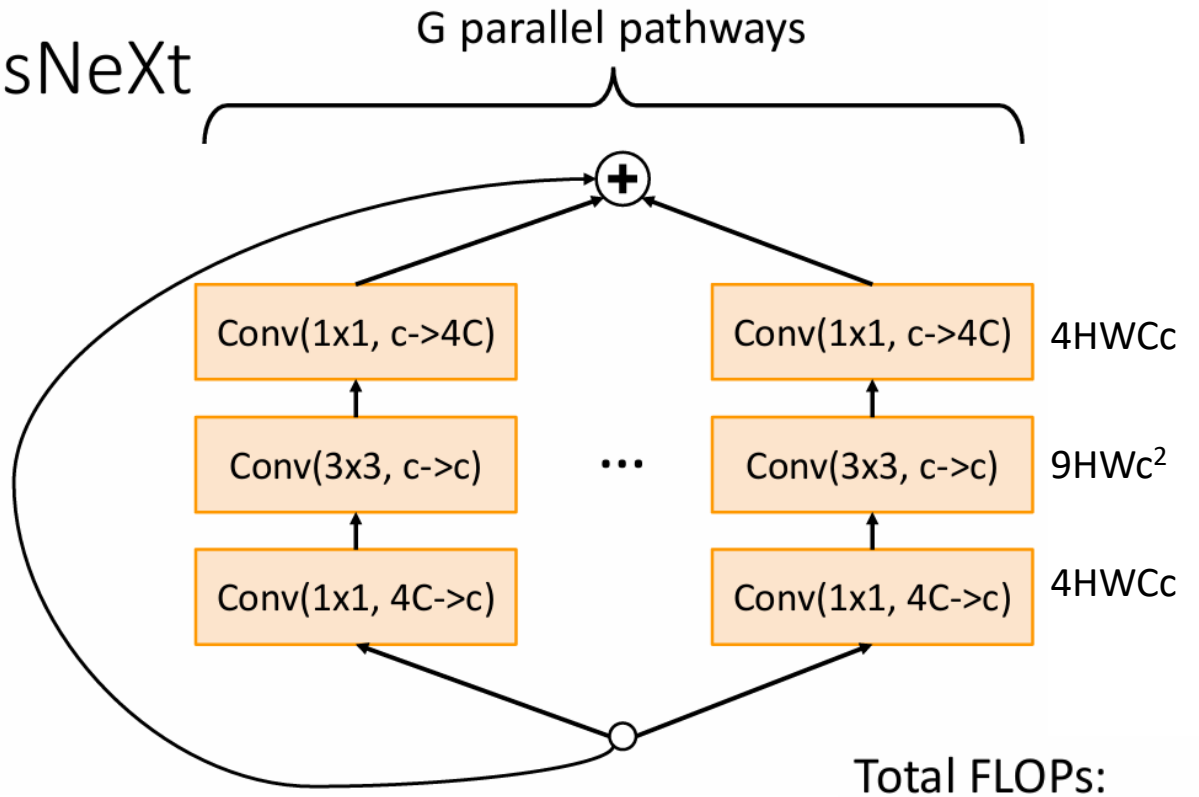
Improving ResNets



"Bottleneck" Residual block

Total FLOPs: $17HWC^2$

ResNeXt



Total FLOPs: $(8Cc + 9c^2)*HWG$

2.3 Grouped Convolution

Convolution with group=1

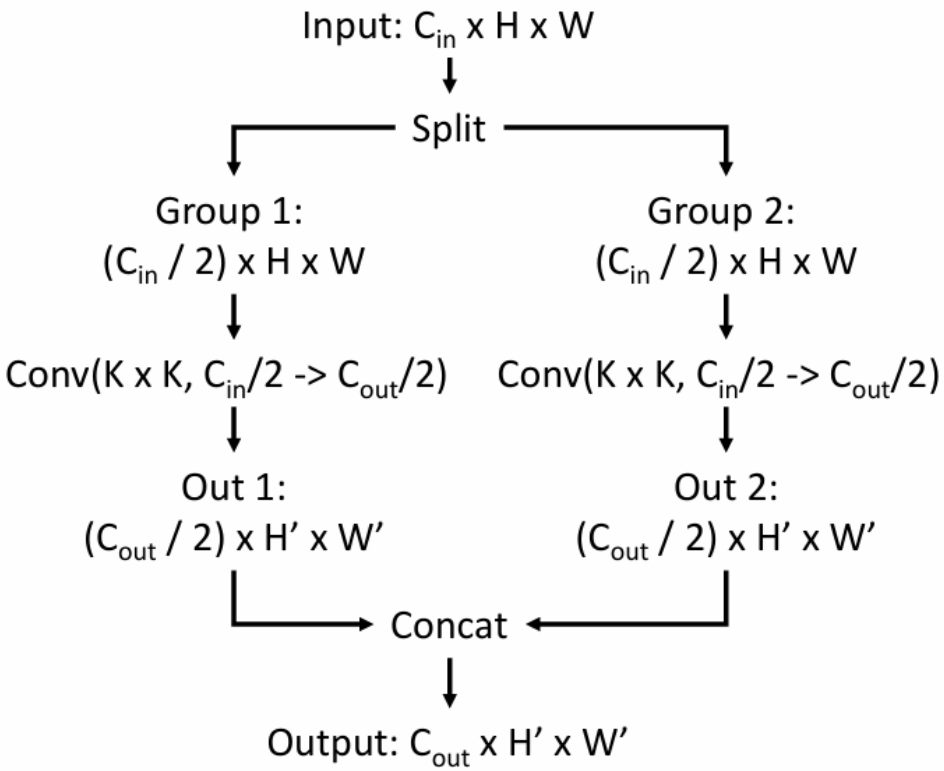
- Input: $C_{in} \times H \times W$
- Weight: $C_{out} \times C_{in} \times K \times K$
- Output: $C_{out} \times H' \times W'$
- FLOPS: $C_{out} C_{in} K^2 HW$

CONV2D

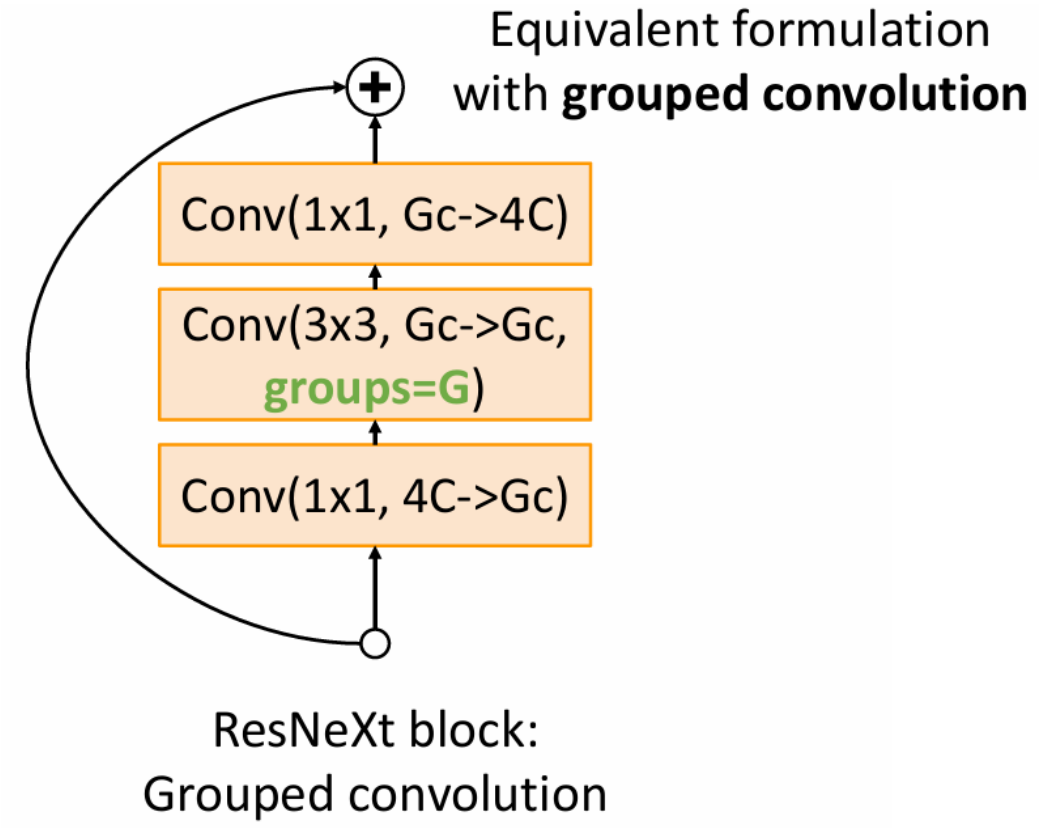
CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

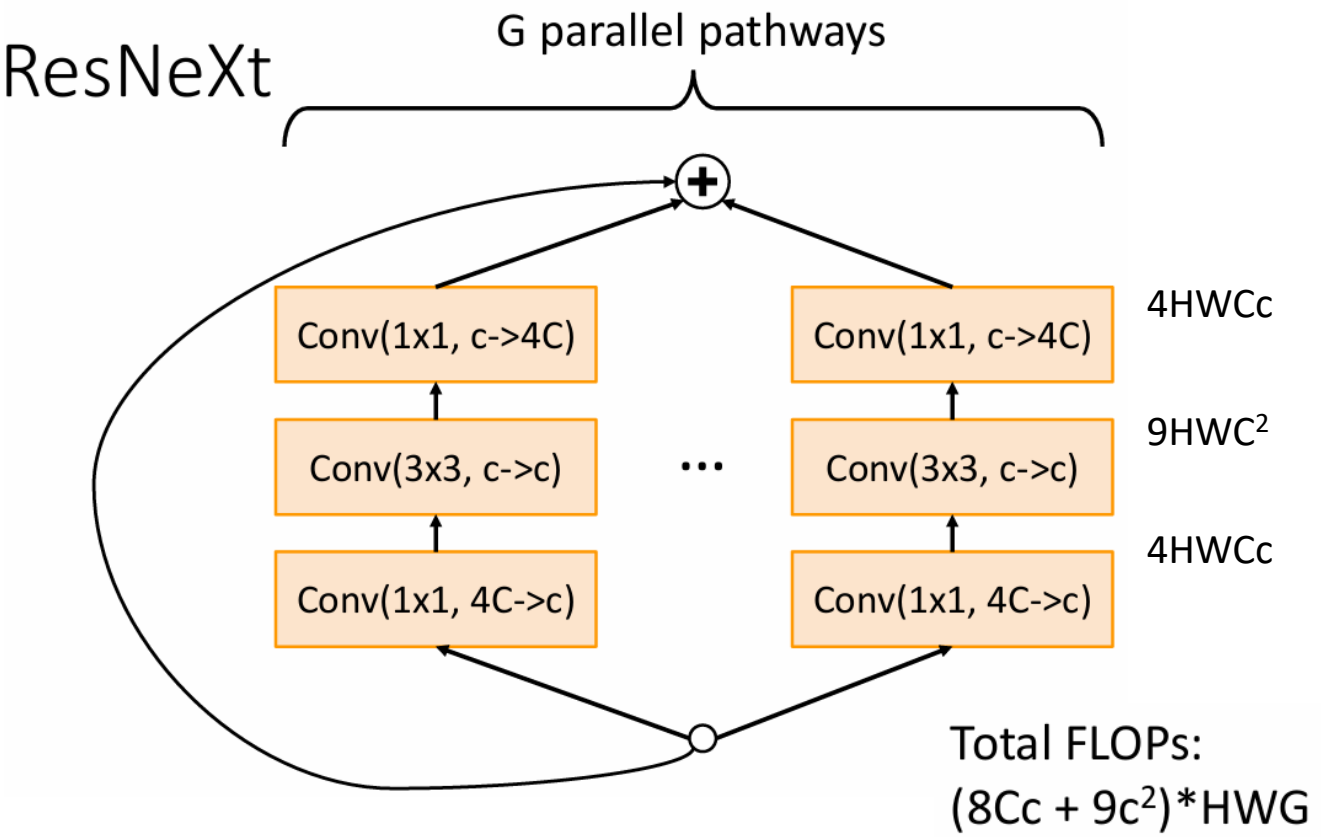
Convolution with group=2

- Two parallel convolution layers that work on half channels



2.4 ResNeXt



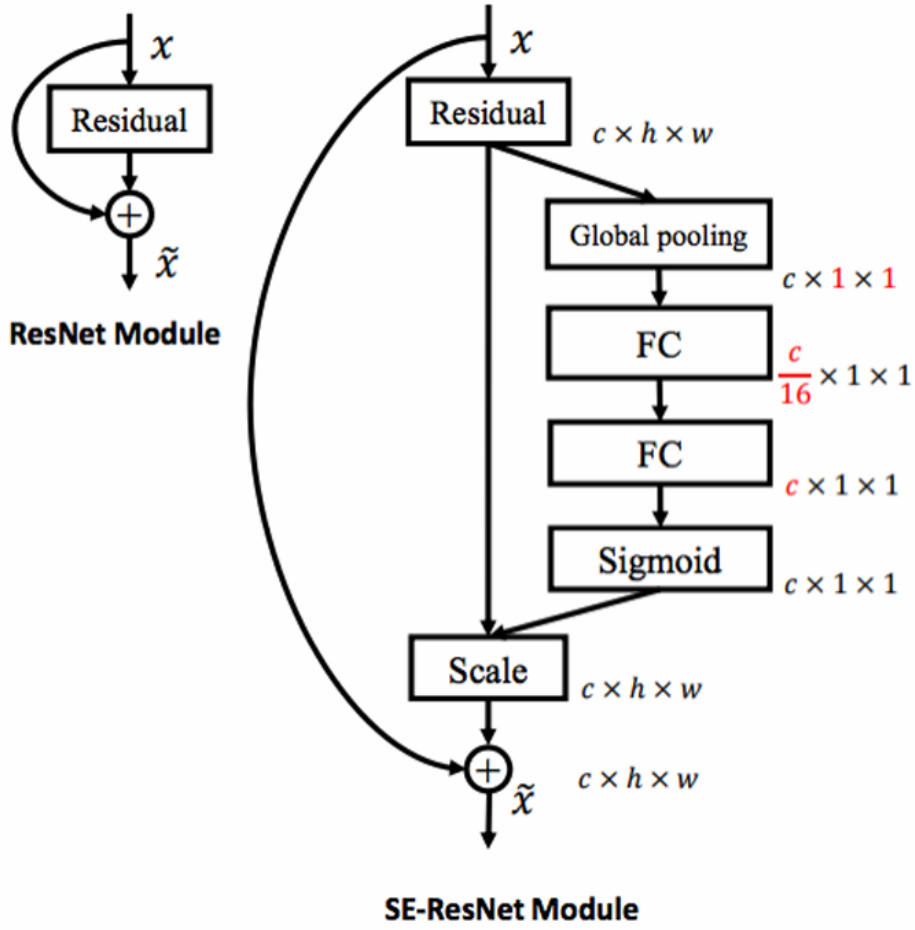


- Adding groups improves performance with the same computational complexity

Xie et al, “Aggregated residual transformations for deep neural networks”, CVPR 2017

2.5 Squeeze-and-Excitation Networks

- Adds a ”Squeeze-and-excite” (SE) branch to each residual block that performs global pooling, full-connected layers, and multiplies back onto feature map
- Adds global context to each residual block!
  - Won ILSVRC 2017 with ResNeXt-152-SE



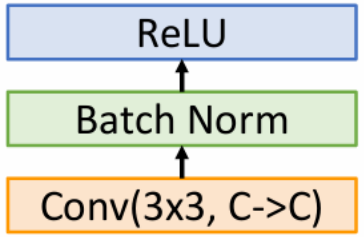
Hu et al, “Squeeze-and-Excitation networks”, CVPR 2018

2.6 MobileNets (Tiny Networks)

- Motivations
  - Small enough to fit in an phone (just 4M parameters)
- Key idea: Depthwise separable convolution

Standard Convolution Block

Total cost: $9C^2HW$

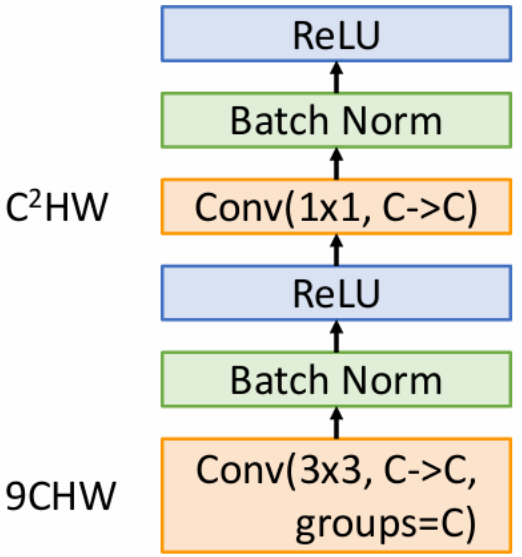


$9C^2HW$

Speedup = $9C^2/(9C+C^2)$
= $9C/(9+C)$
=> 9 (as C->inf)

Depthwise Separable Convolution

Total cost: $(9C + C^2)HW$



C^2HW

"Pointwise Convolution"

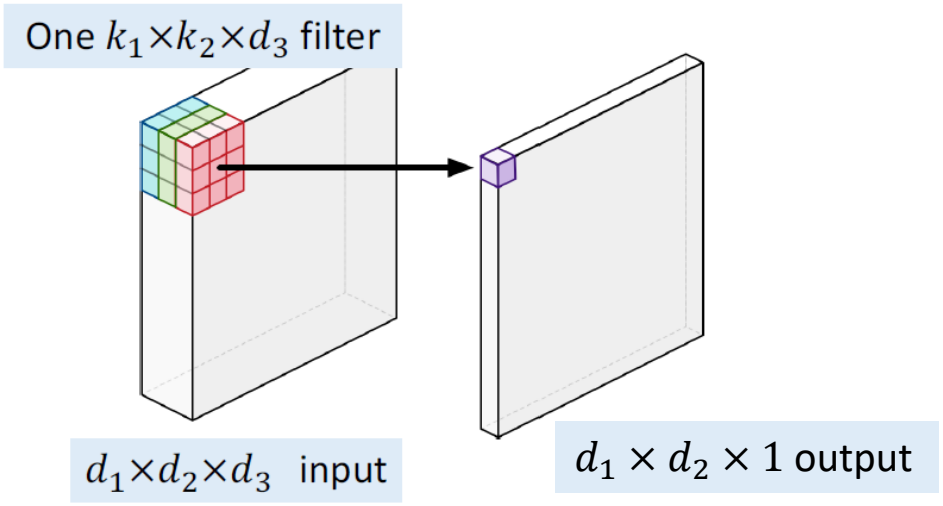
$9CHW$

"Depthwise Convolution"

Howard et al, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017

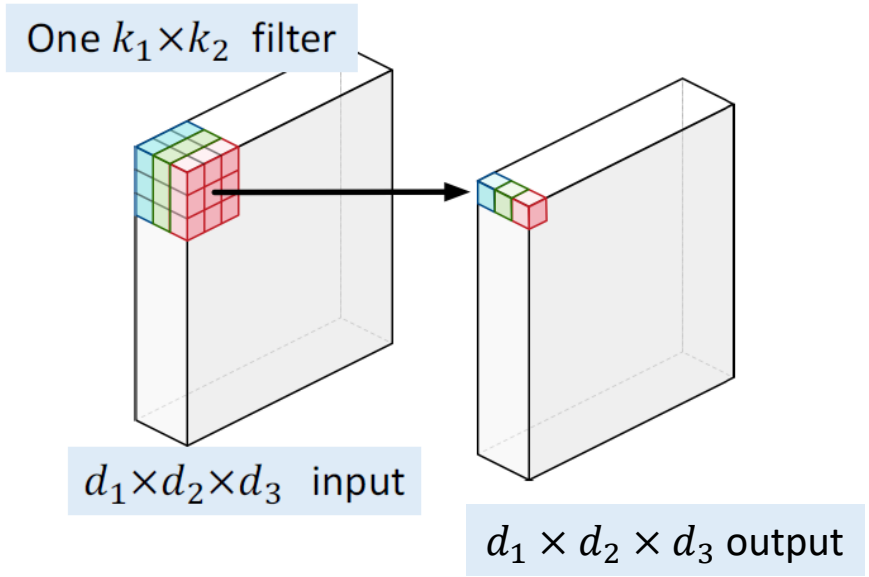
2.6 MobileNets (Tiny Networks)

Convolution



In this example, $d_3 = 3$.

Depthwise Convolution



For $i = 1$ to d_3 (each of the slices):

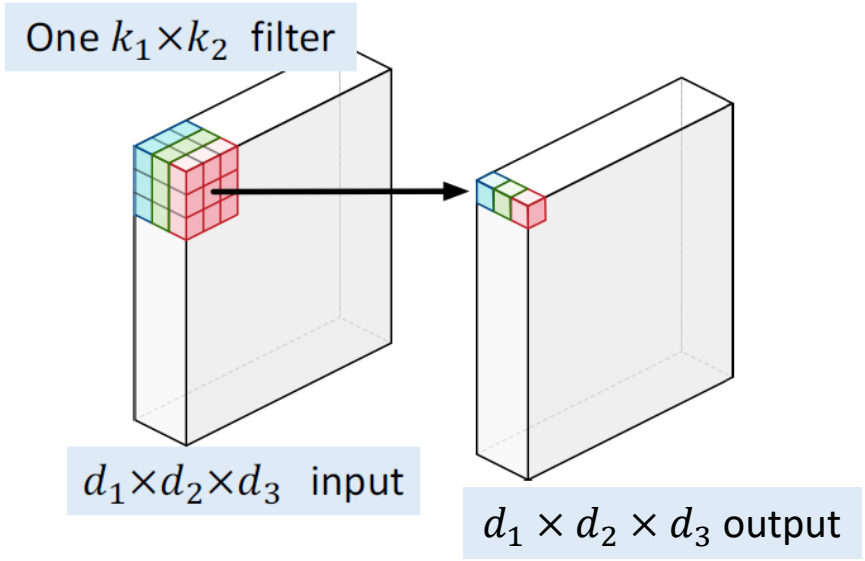
- Compute the convolution of the *i*-th slice (shape $d_1 \times d_2$) and the filter (shape $k_1 \times k_2$).
- Output: a matrix (shape $d_1 \times d_2$).

Howard et al, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017

2.6 MobileNets (Tiny Networks)

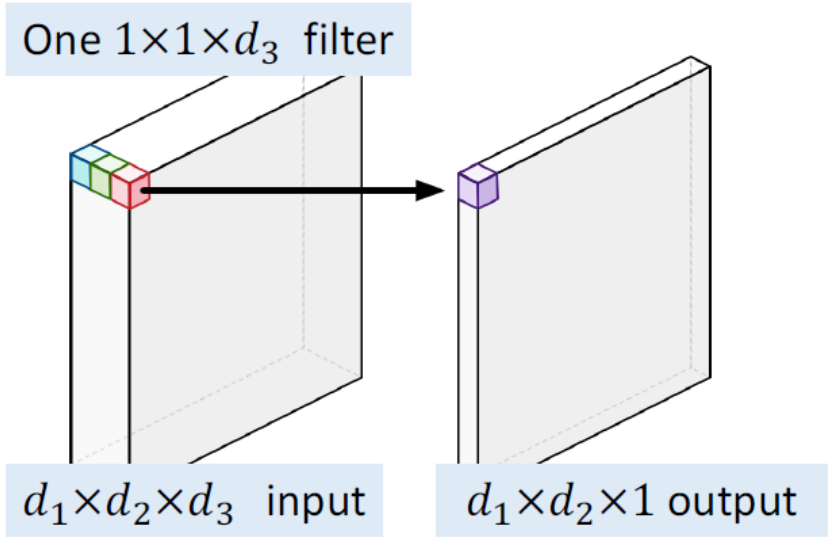
- MobileNet: Depthwise Conv + 1×1 Conv

Depthwise Convolution



$k_1 \times k_2 \times d_3$ parameters

1×1 Convolution

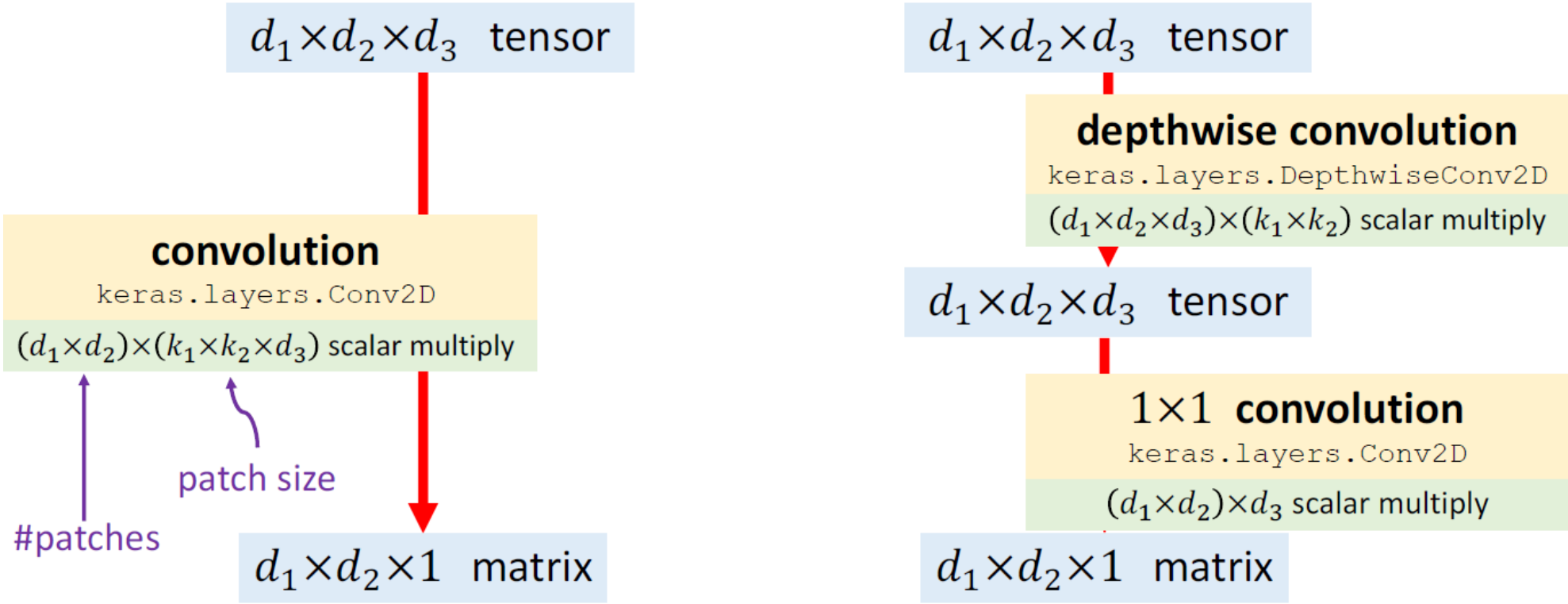


$k_1 \times k_2 + d_3$ parameters

- ShuffleNet: Zhang et al, CVPR 2018
- MobileNetV2: Sandler et al, CVPR 2018
- ShuffleNetV2: Ma et al, ECCV 2018

2.6 MobileNets (Tiny Networks)

- MobileNet: Depthwise Conv + 1×1 Conv

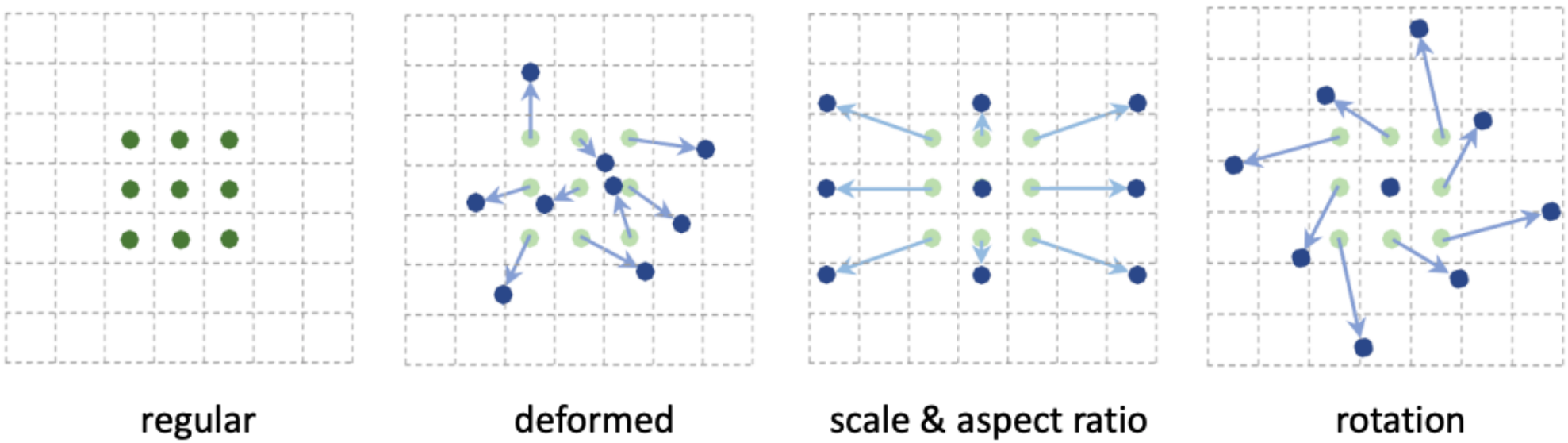


Does not save computation though

- ShuffleNet: Zhang et al, CVPR 2018
- MobileNetV2: Sandler et al, CVPR 2018
- ShuffleNetV2: Ma et al, ECCV 2018

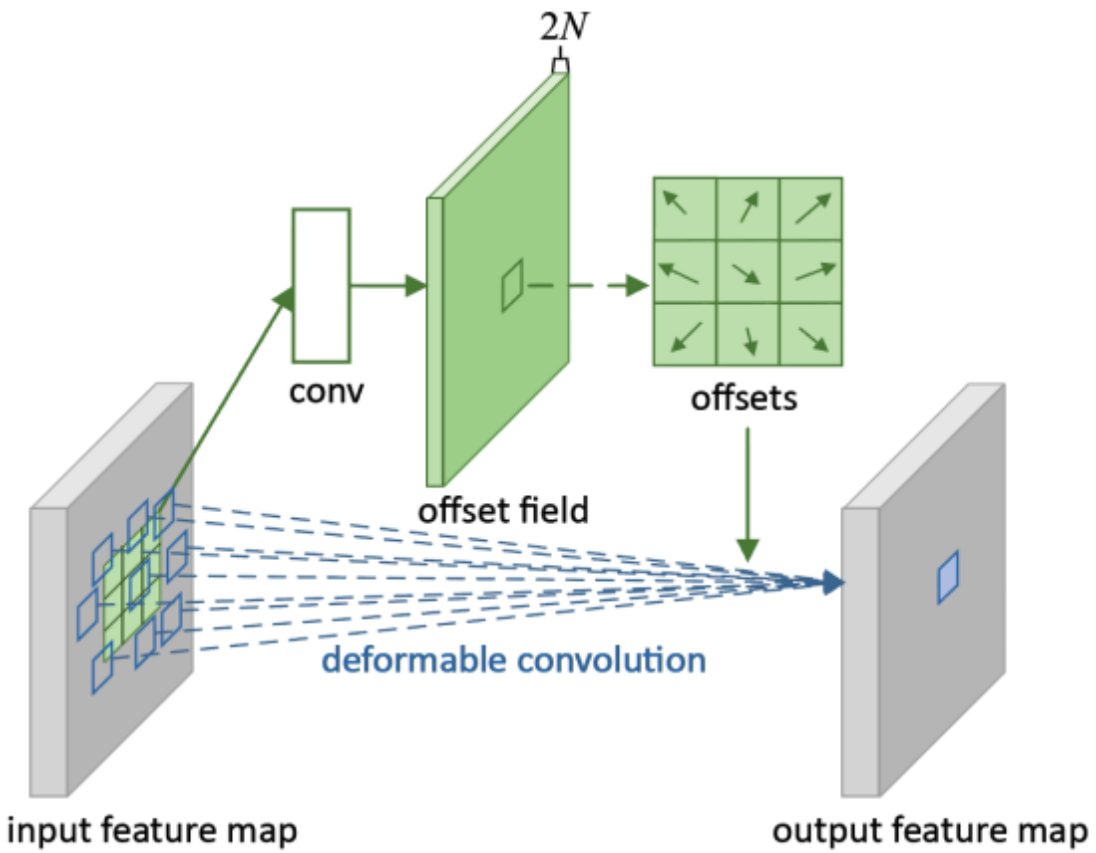
2.7 Deformable Convolution

- DCN for spatial transformations networks [1,2]
  - Regular CNNs are inherently limited to model large unknown transformations
  - Deformable convolution provides local, dense, non-parametric transformation
    - Learn to deform the sampling locations in the convolution/ROI pooling modules



[1] Jifeng Dai, et al. Deformable Convolutional Networks, ICCV 2017.
[2] Xizhou Zhu, et al. Deformable ConvNets v2: More Deformable, Better Results, CVPR 2018.

2.7 Deformable Convolution



Regular convolution

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n)$$

Deformable convolution

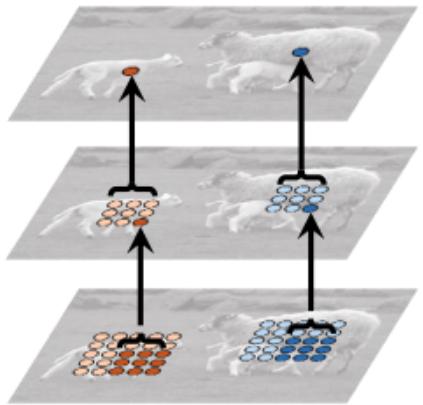
$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)$$

where $\Delta\mathbf{p}_n$ is generated by a sibling branch of regular convolution

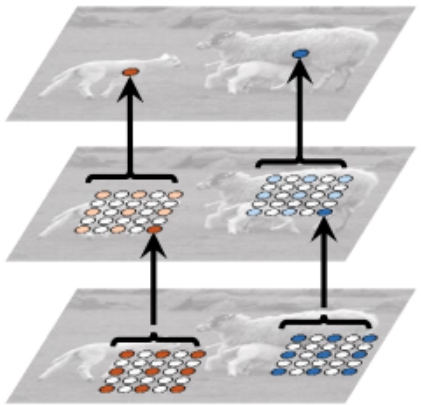
[1] Jifeng Dai, et al. Deformable Convolutional Networks, ICCV 2017.
[2] Xizhou Zhu, et al. Deformable ConvNets v2: More Deformable, Better Results, CVPR 2018.

2.7 Deformable Convolution

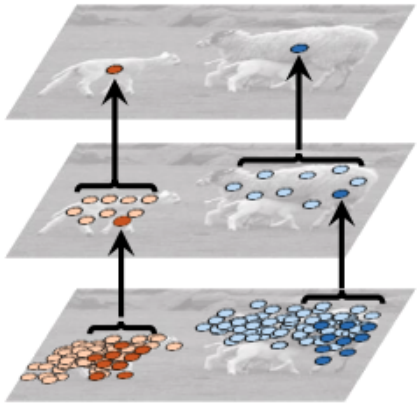
Deformable modules	DeepLab mIoU@V/@C	Class-aware RPN mAP@0.5/@0.7	Faster R-CNN mAP@0.5/@0.7	R-FCN mAP@0.5/@0.7
Dilated convolution (2, 2, 2) (default)	69.7 / 70.4	68.0 / 44.9	78.1 / 62.1	80.0 / 61.8
Dilated convolution (4, 4, 4)	73.1 / 71.9	72.8 / 53.1	78.6 / 63.1	80.5 / 63.0
Dilated convolution (6, 6, 6)	73.6 / 72.7	73.6 / 55.2	78.5 / 62.3	80.2 / 63.5
Dilated convolution (8, 8, 8)	73.2 / 72.4	73.2 / 55.1	77.8 / 61.8	80.3 / 63.2
Deformable convolution	**75.3 / 75.2**	**74.5 / 57.2**	78.6 / 63.3	81.4 / 64.7
Deformale RoI pooling	N.A	N.A	78.3 / 66.6	81.2 / 65.0
Deformale convolution & RoI pooling	N.A	N.A	**79.3 / 66.9**	**82.6 / 68.5**



regular convolution



dilated convolution



deformable convolution

[1] Jifeng Dai, et al. Deformable Convolutional Networks, ICCV 2017.
[2] Xizhou Zhu, et al. Deformable ConvNets v2: More Deformable, Better Results, CVPR 2018.

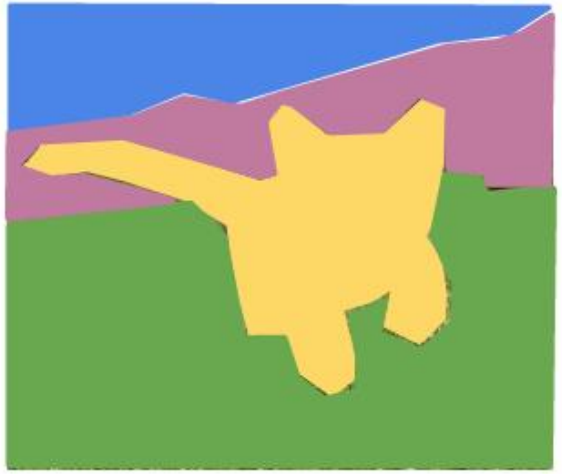
Computer Vision Tasks

Classification

CAT

No spatial extent

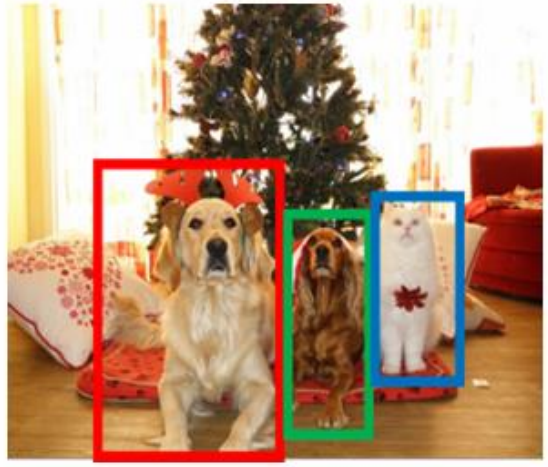
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



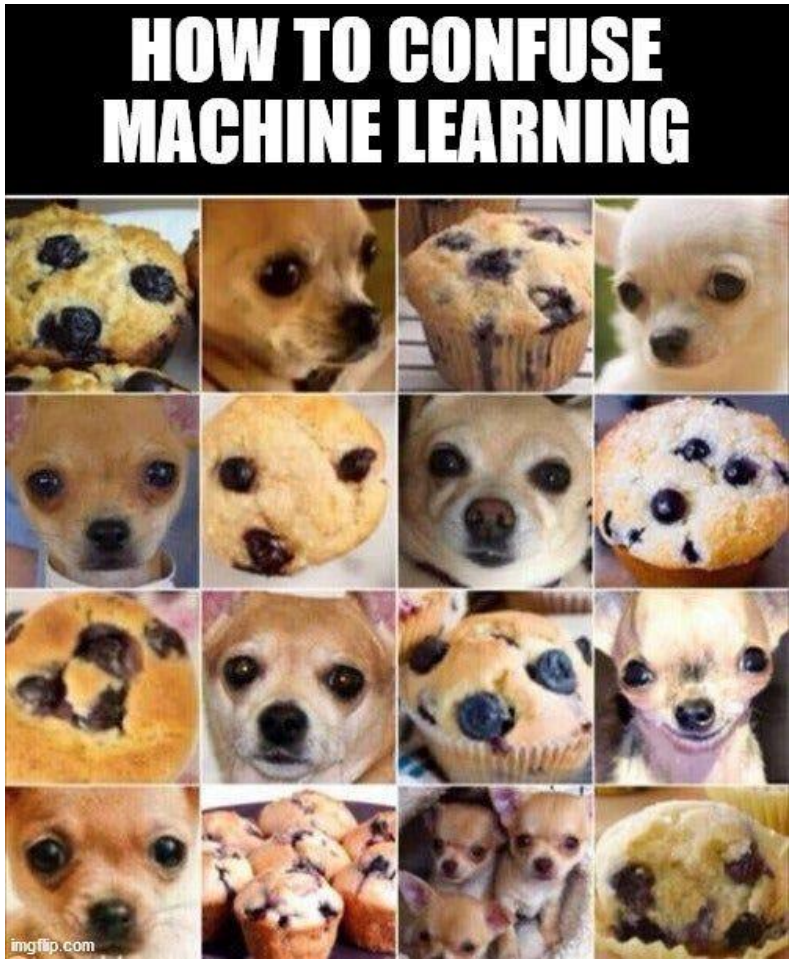
DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

Multiple Objects

Computer Vision is Challenging



3.1 What is Object Detection?

Definition

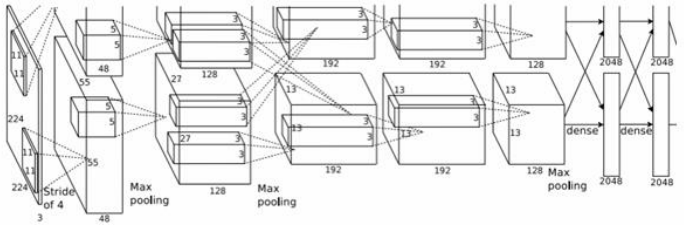
- **Input:** Single RGB image
- **Output:** A set of detected objects
  - For each object predict:
    1. Category label (from fixed, predefined set of categories)
    2. Bounding box (four numbers: x, y, width, height)

Challenges

- **Multiple outputs:** Need to output variable numbers of objects per image
- **Multiple types of output:** Need to predict “what” (category label) as well as “where” (bounding box)
- **Large images:** Classification works at 224x224; need higher resolution for detection, often above 800x600

Solution: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO
Cat? NO
Background? YES

3.1 What is Object Detection?

Definition

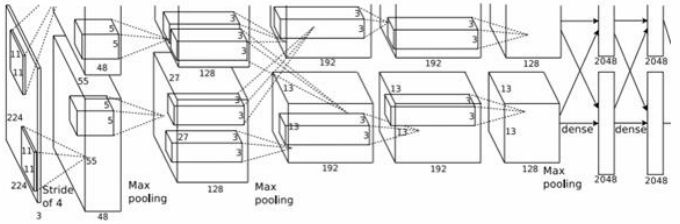
- **Input:** Single RGB image
- **Output:** A set of detected objects
  - For each object predict:
    1. Category label (from fixed, predefined set of categories)
    2. Bounding box (four numbers: x, y, width, height)

Challenges

- **Multiple outputs:** Need to output variable numbers of objects per image
- **Multiple types of output:** Need to predict “what” (category label) as well as “where” (bounding box)
- **Large images:** Classification works at 224x224; need higher resolution for detection, often above 800x600

Solution: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

3.1 What is Object Detection?

Definition

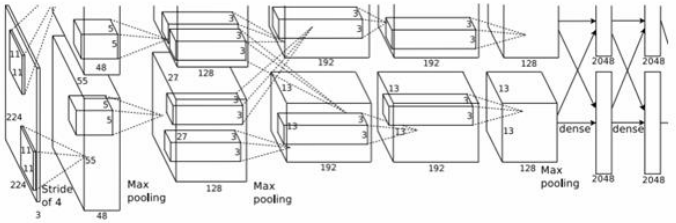
- **Input:** Single RGB image
- **Output:** A set of detected objects
  - For each object predict:
    1. Category label (from fixed, predefined set of categories)
    2. Bounding box (four numbers: x, y, width, height)

Challenges

- **Multiple outputs:** Need to output variable numbers of objects per image
- **Multiple types of output:** Need to predict “what” (category label) as well as “where” (bounding box)
- **Large images:** Classification works at 224x224; need higher resolution for detection, often above 800x600

Solution: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

3.1 What is Object Detection?

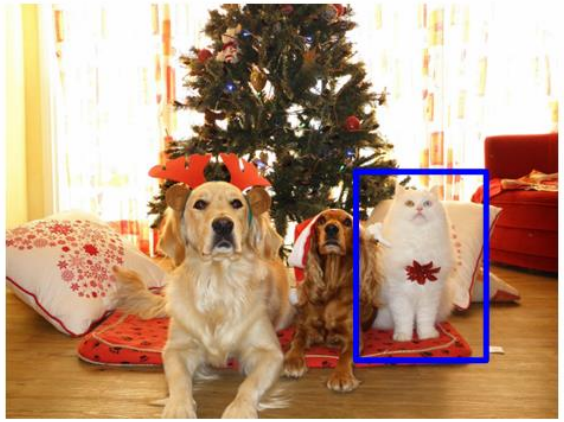
Definition

- **Input:** Single RGB image
- **Output:** A set of detected objects
  - For each object predict:
    1. Category label (from fixed, predefined set of categories)
    2. Bounding box (four numbers: x, y, width, height)

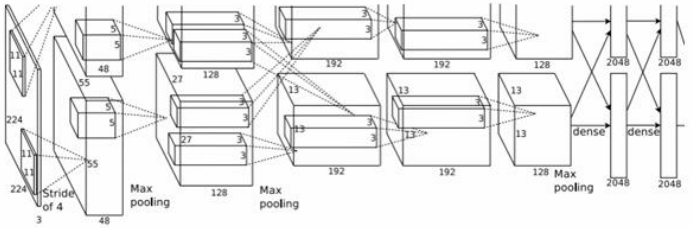
Challenges

- **Multiple outputs:** Need to output variable numbers of objects per image
- **Multiple types of output:** Need to predict “what” (category label) as well as “where” (bounding box)
- **Large images:** Classification works at 224x224; need higher resolution for detection, often above 800x600

Solution: Sliding Window



Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

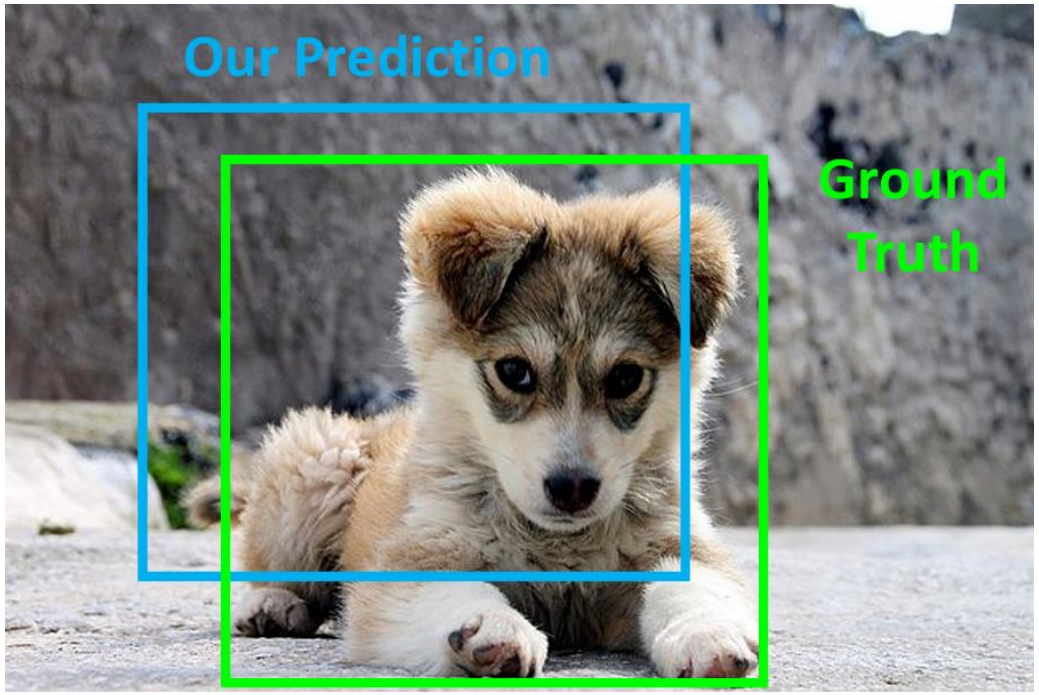


Dog? NO
Cat? YES
Background? NO

3.2 Comparing Boxes: IoU

Intersection over Union (IoU)

- How can we compare our prediction to the ground-truth?



3.2 Comparing Boxes: IoU

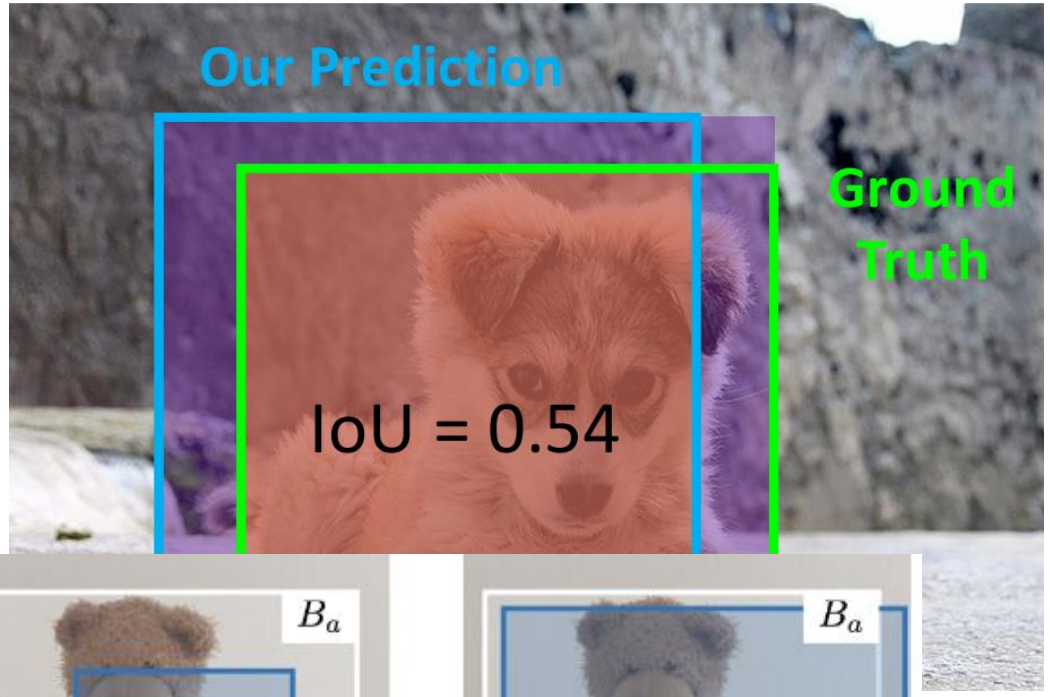
Intersection over Union (IoU)

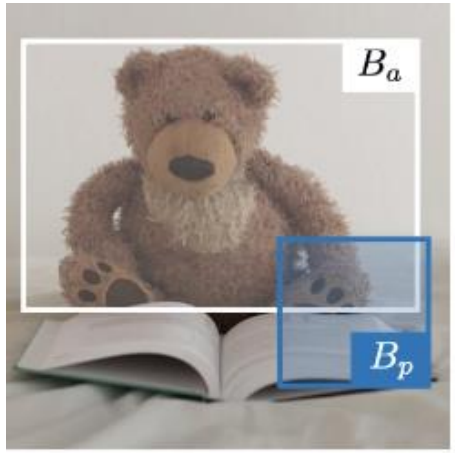
- How can we compare our prediction to the ground-truth?

- Intersection over Union (IoU)
  - Also called “Jaccard similarity”

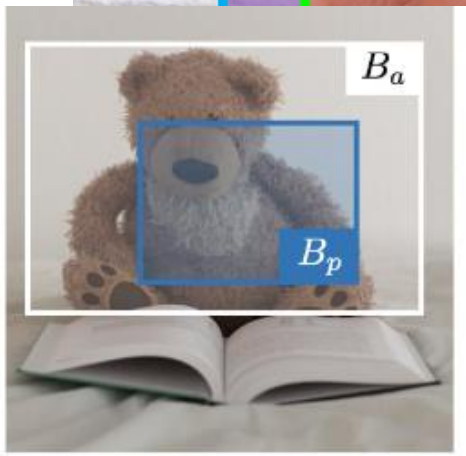
$$\frac{\text{Area of Intersection}}{\text{Area of Union}}$$

IoU > 0.5 is “decent”
IoU > 0.7 is “good”
IoU > 0.9 is “almost perfect”

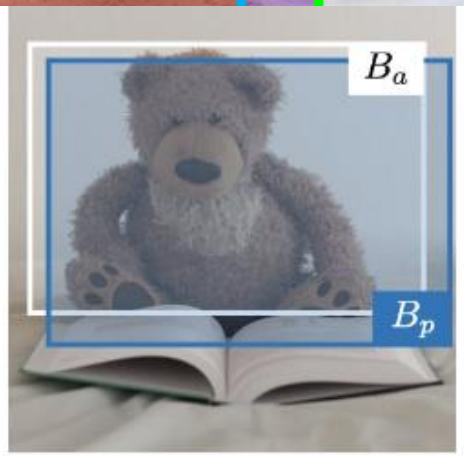




$\text{IoU}(B_p, B_a) = 0.1$



$\text{IoU}(B_p, B_a) = 0.5$



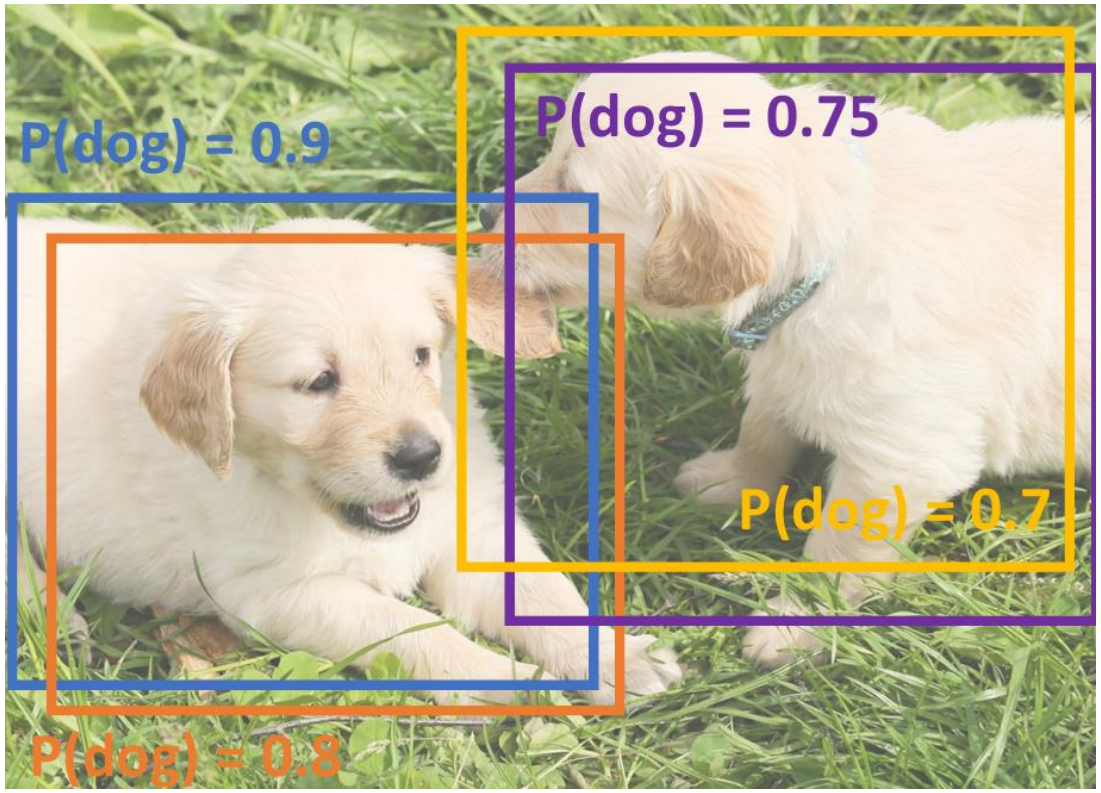
$\text{IoU}(B_p, B_a) = 0.9$

3.3 Non-Max Suppression

Overlapping Boxes

- **Problem:** Object detectors often output many overlapping detections.
- **Solution:** Post-process raw detections using Non-Max Suppression (NMS).

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1



IoU(,) = **0.78**
IoU(,) = 0.05
IoU(,) = 0.07

IoU(,) = **0.74**

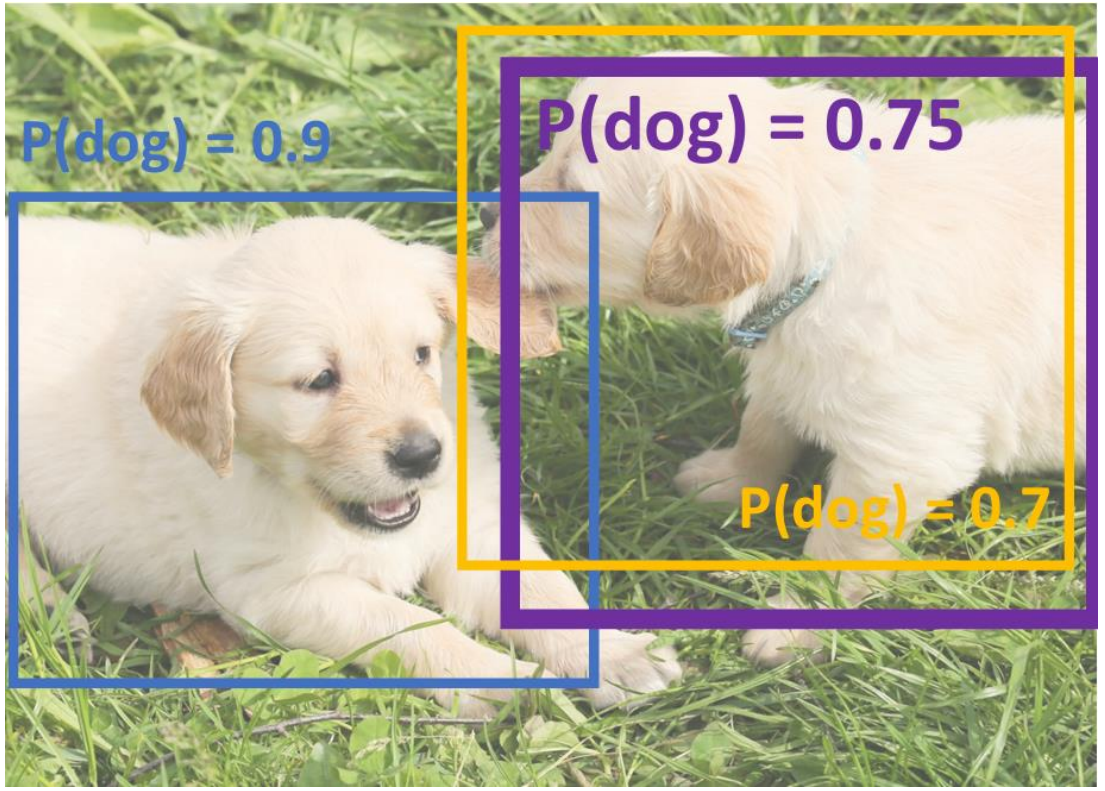
3.3 Non-Max Suppression

Overlapping Boxes

- **Problem:** Object detectors often output many overlapping detections.
- **Solution:** Post-process raw detections using Non-Max Suppression (NMS).

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

IoU(,) = **0.74**

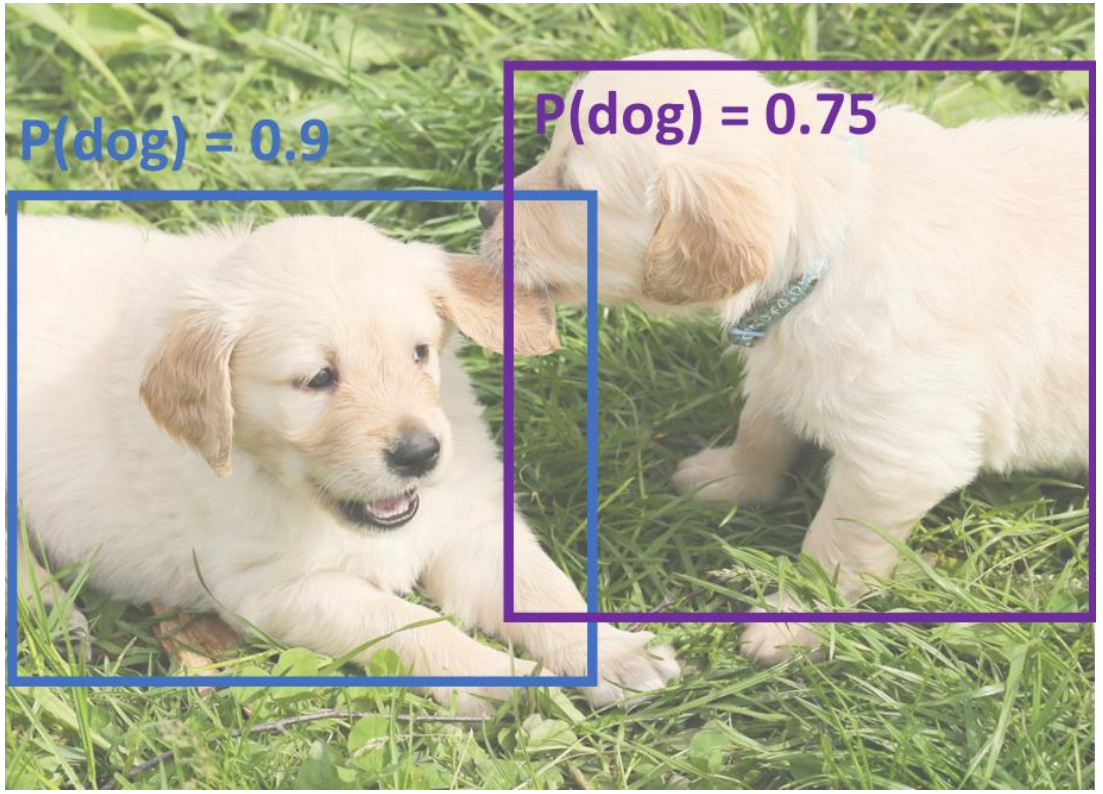


3.3 Non-Max Suppression

Overlapping Boxes

- **Problem:** Object detectors often output many overlapping detections.
- **Solution:** Post-process raw detections using Non-Max Suppression (NMS).

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

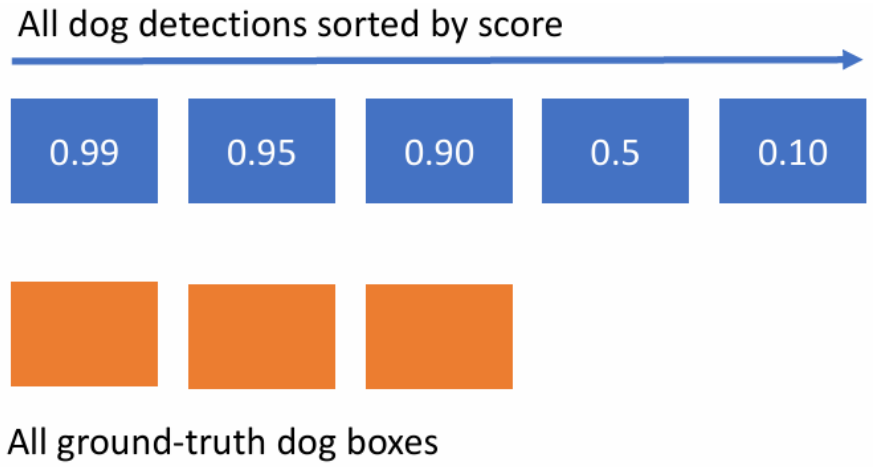


Remarks: NMS might eliminate “good” boxes when objects are highly overlapping…

3.4 Evaluating Object Detectors

Mean Average Precision (mAP)

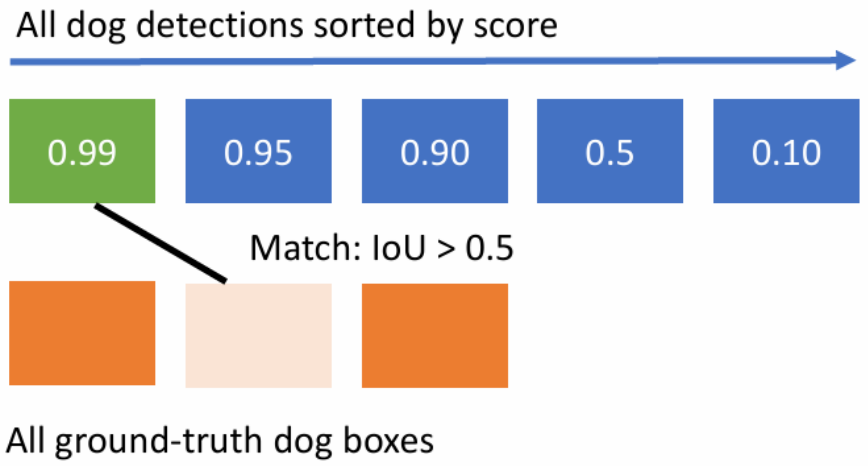
- Run object detector on all test images (with NMS)
- For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)

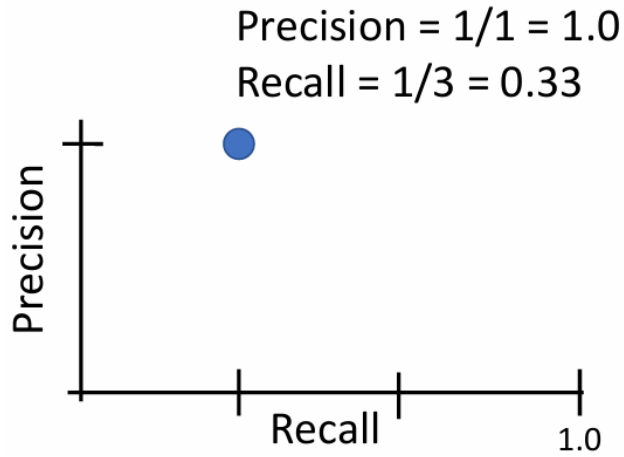


3.4 Evaluating Object Detectors

Average Precision (AP)

- Run object detector on all test images (with NMS)
- For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        - If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        - Otherwise mark it as negative
        - Plot a point on PR Curve

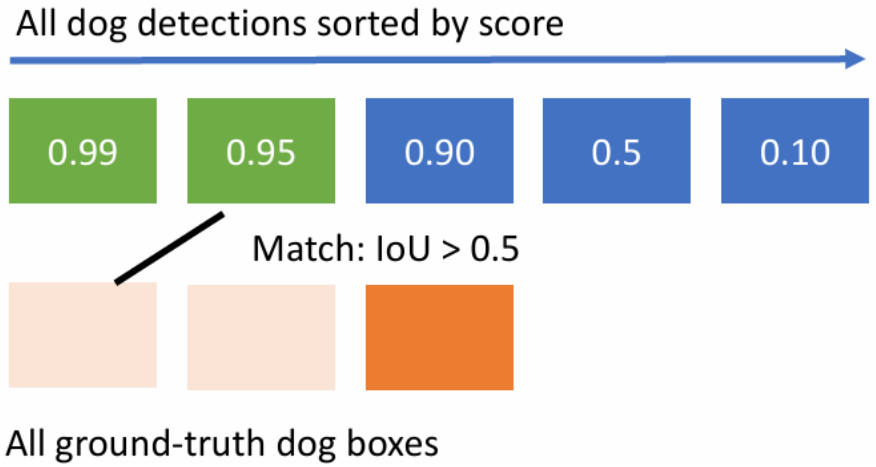


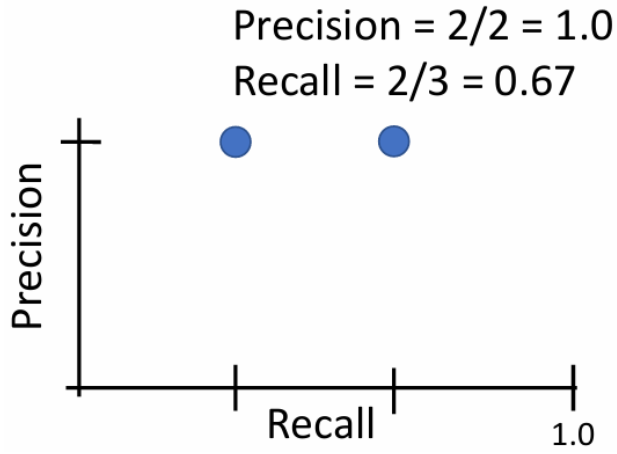


3.4 Evaluating Object Detectors

Average Precision (AP)

- Run object detector on all test images (with NMS)
- For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        - If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        - Otherwise mark it as negative
        - Plot a point on PR Curve

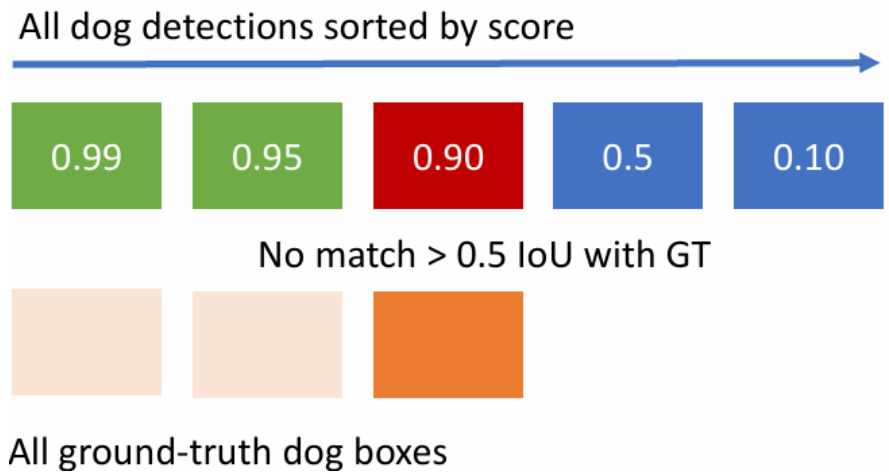


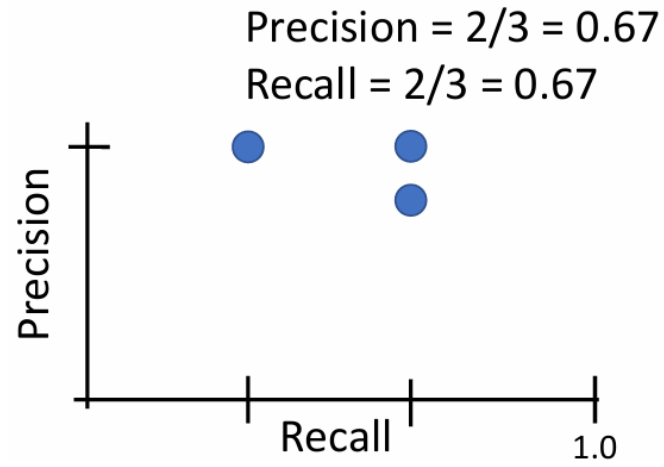


3.4 Evaluating Object Detectors

Average Precision (AP)

- Run object detector on all test images (with NMS)
- For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        - If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        - Otherwise mark it as negative
        - Plot a point on PR Curve

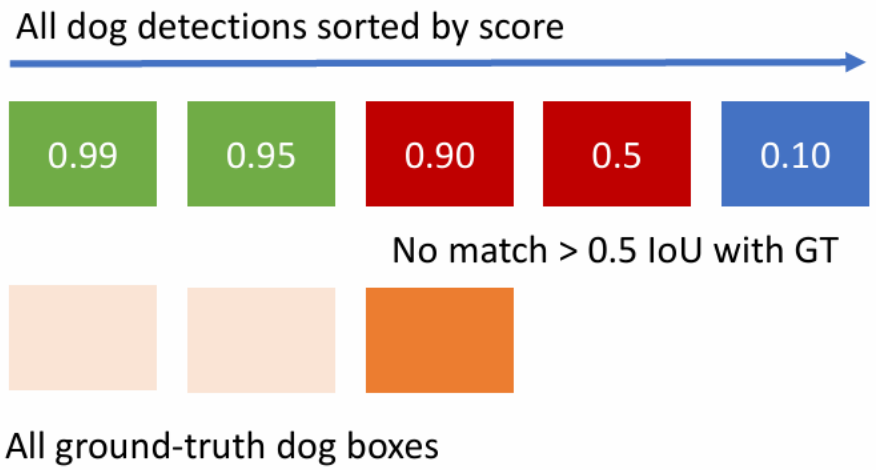


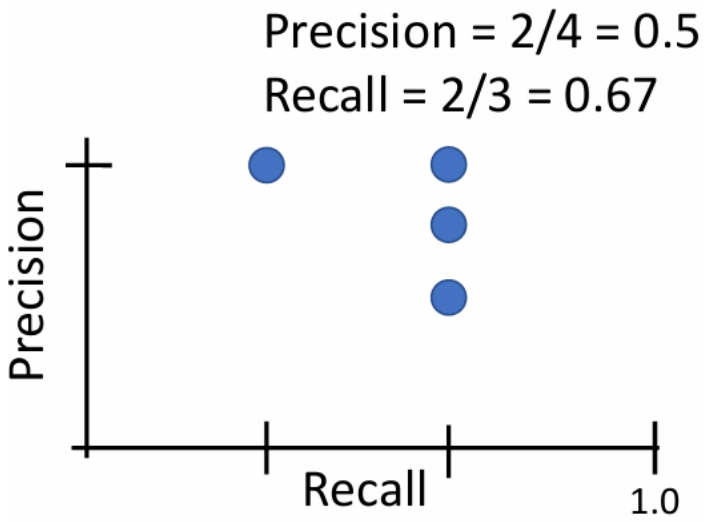


3.4 Evaluating Object Detectors

Average Precision (AP)

- Run object detector on all test images (with NMS)
- For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
  1. For each detection (highest score to lowest score)
     - If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
     - Otherwise mark it as negative
     - Plot a point on PR Curve

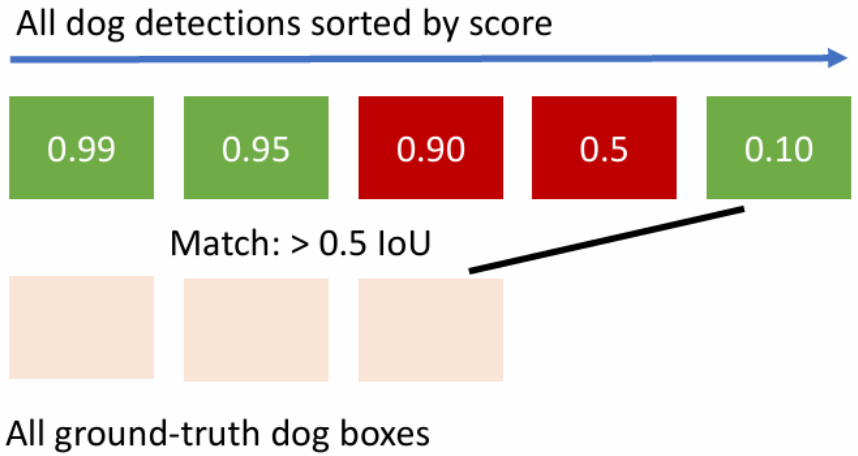


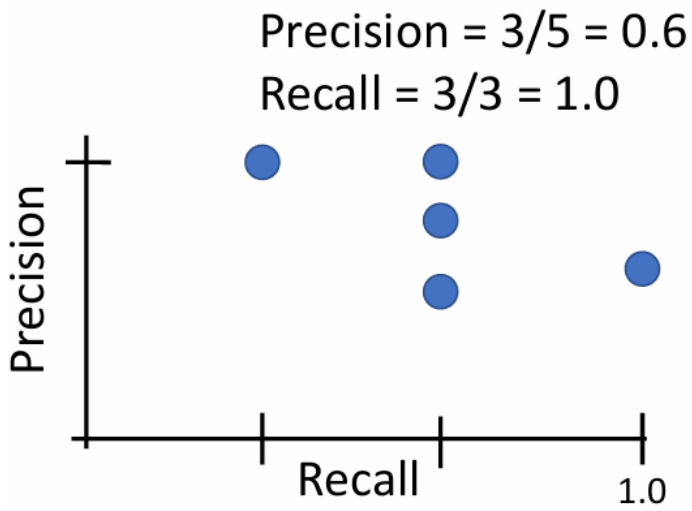


3.4 Evaluating Object Detectors

Average Precision (AP)

- Run object detector on all test images (with NMS)
- For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        - If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        - Otherwise mark it as negative
        - Plot a point on PR Curve





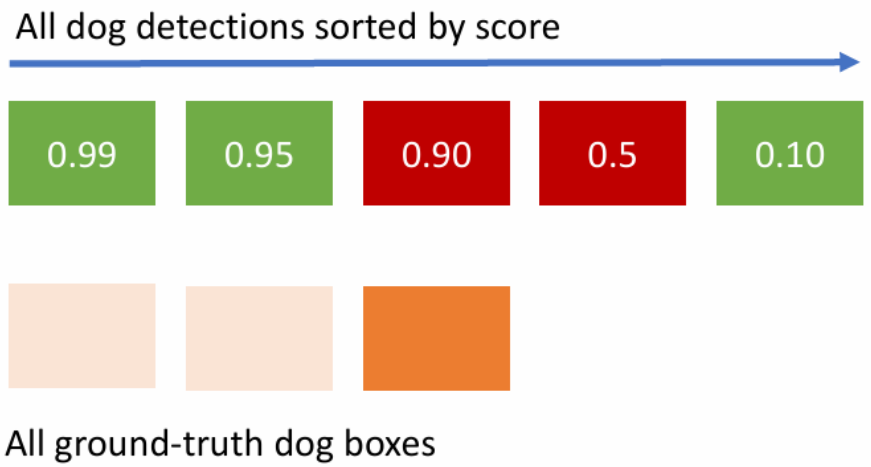
3.4 Evaluating Object Detectors

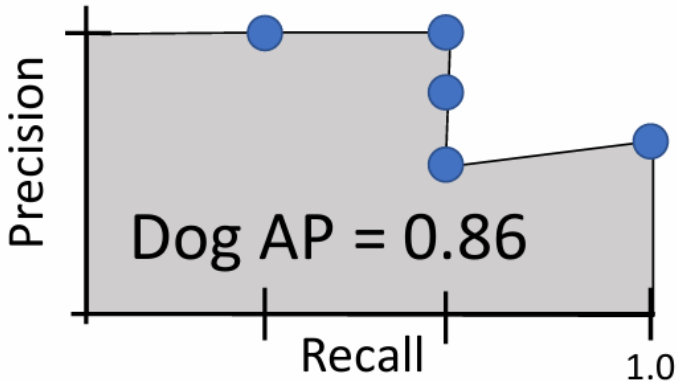
Average Precision (AP)

- Run object detector on all test images (with NMS)
- For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
  1. For each detection (highest score to lowest score)
     - If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
     - Otherwise mark it as negative
     - Plot a point on PR Curve
  2. Average Precision (AP) = area under PR curve

How to get AP=1.0?

Hit all GT boxes with IoU > 0.5, and have no "false positive" detections ranked above any "true positives"





3.4 Evaluating Object Detectors

Mean Average Precision (mAP)

- Run object detector on all test images (with NMS)
- For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        - If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        - Otherwise mark it as negative
        - Plot a point on PR Curve
    2. Average Precision (AP) = area under PR curve
- Mean Average Precision (mAP) = average of AP for all categories
- For "COCO mAP": Computer mAP@thresh for each IoU threshold (0.5, 0.55, 0.6, …, 0.95) and take average

Car AP = 0.65
Cat AP = 0.80
Dog AP = 0.86

mAP@0.5 = 0.77
mAP@0.55 = 0.71
mAP@0.60 = 0.65
…
mAP@0.95 = 0.2

COCO mAP = 0.4

3.5 Single-Stage Object Detection

YOLO V1

Run CNN backbone to get features aligned to input image

Classify each object as one of C categories (or background)[1]

Input Image
(e.g. 3x640x480)

CNN

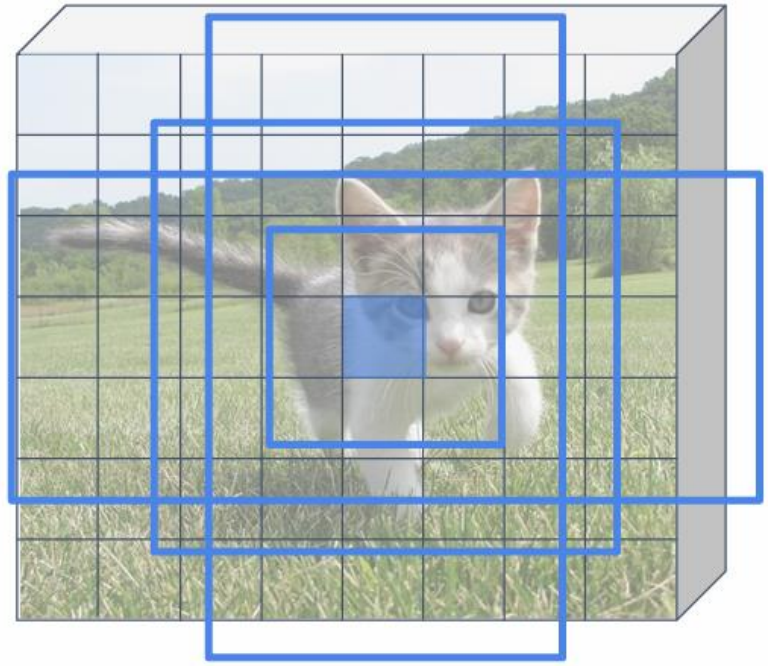


Image Features
(e.g. 512x20x15)

Conv

Anchor category
(C+1)xKx20x15

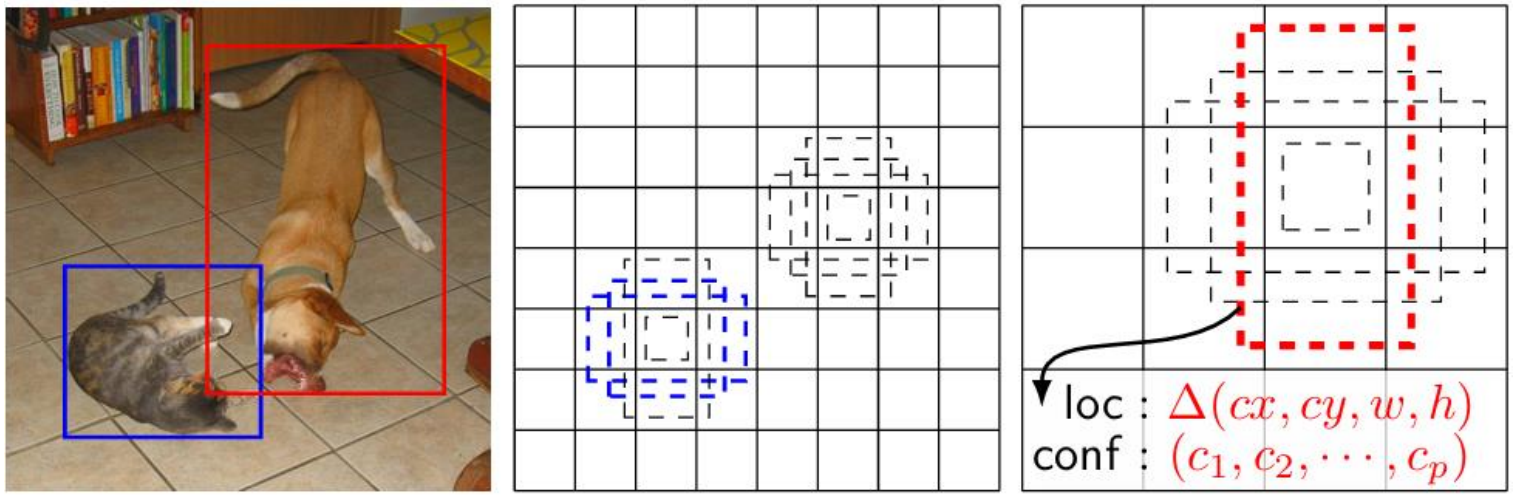
Box transforms
Cx4Kx20x15

K anchors at each position in the image feature map

[1] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

3.5 Single-Stage Object Detection

- Single Shot Multibox Detector (SSD)



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Box regression localization loss

Class confidence loss

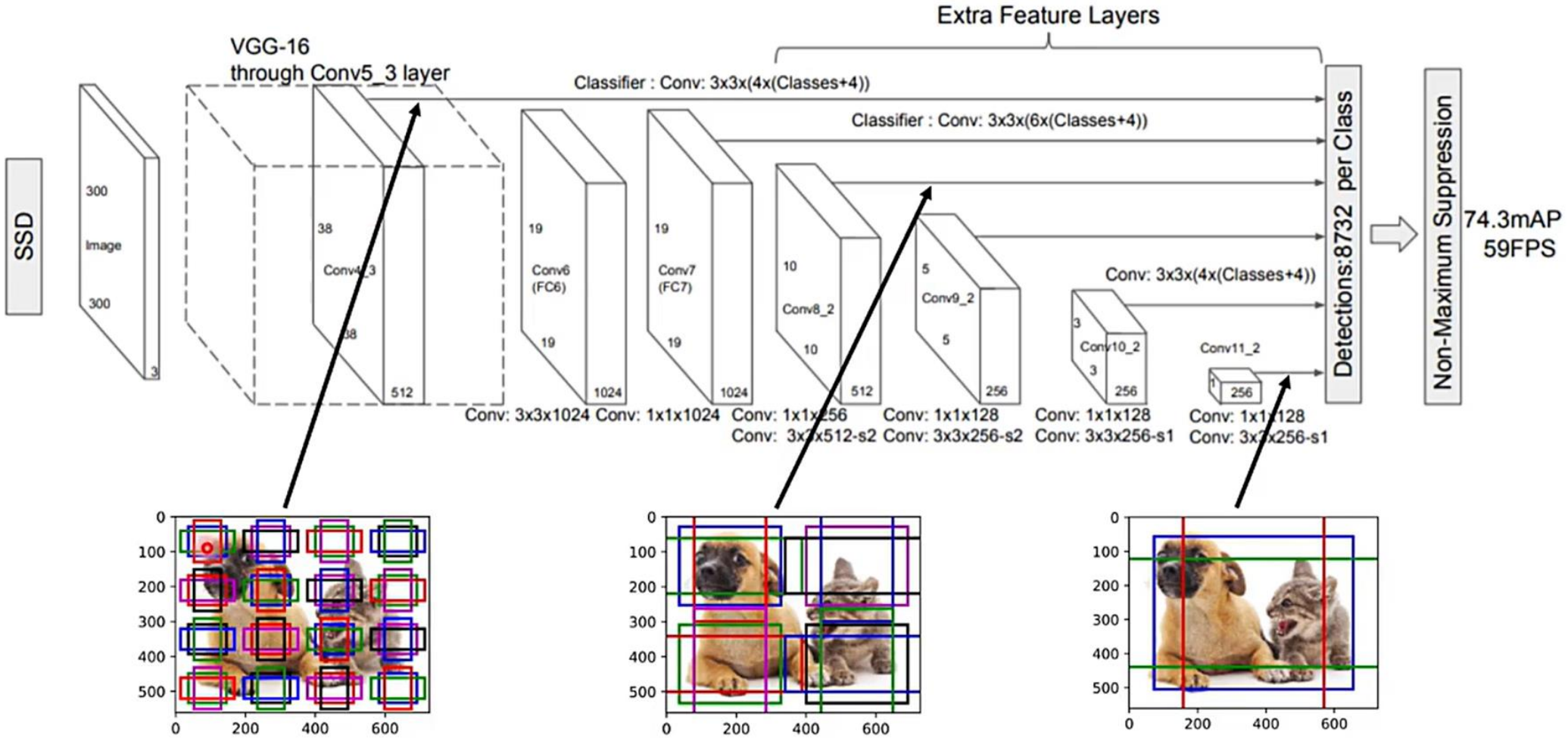
Anchor boxes spread on the grids with different ratios (1, 2, ½)

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

[1] Liu, Wei, et al. "Ssd: Single shot multibox detector." Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14. Springer International Publishing, 2016.

3.5 Single-Stage Object Detection

- Single Shot Multibox Detector (SSD)

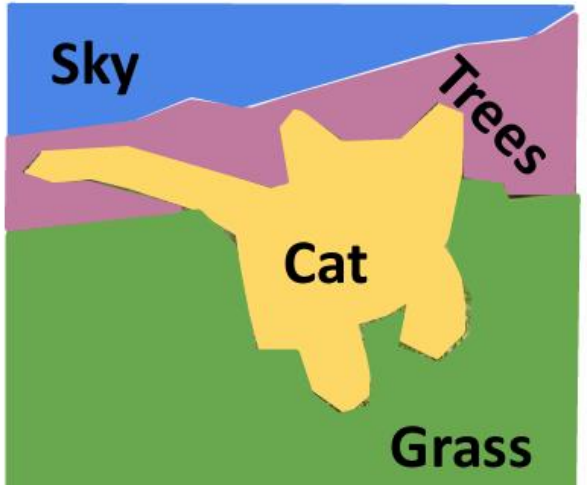


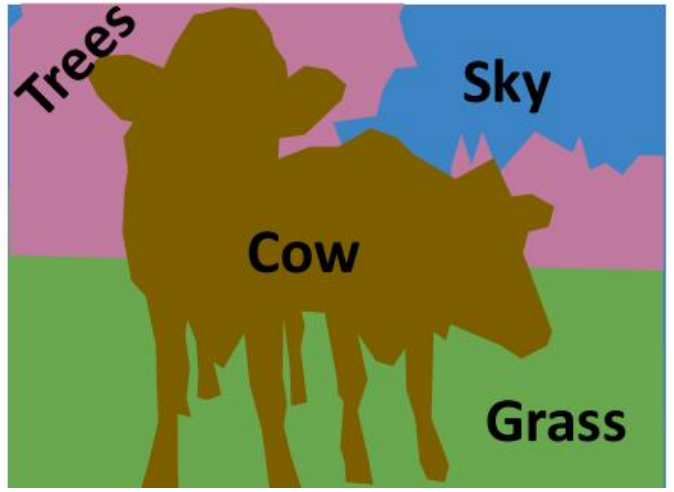
[1] Liu, Wei, et al. "Ssd: Single shot multibox detector." Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14. Springer International Publishing, 2016.

3.6 Semantic Segmentation

Label each pixel in the image with a category label

Don’t differentiate instances, only care about pixels



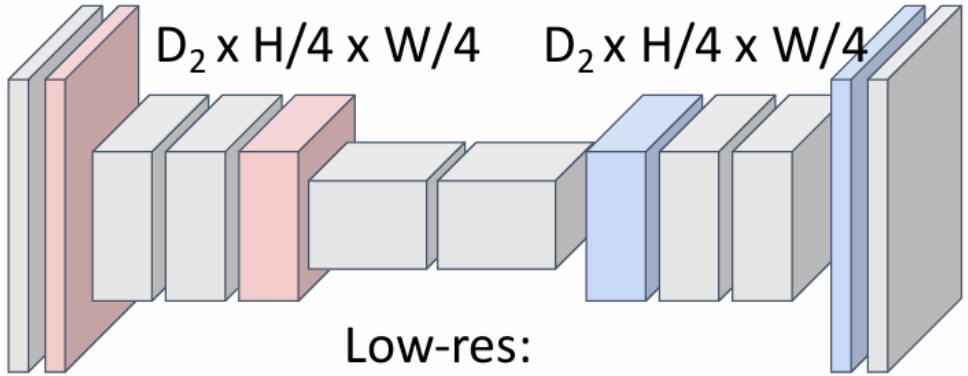


3.6 Semantic Segmentation

Semantic Segmentation: Fully Convolutional Network

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W



Predictions:
H x W

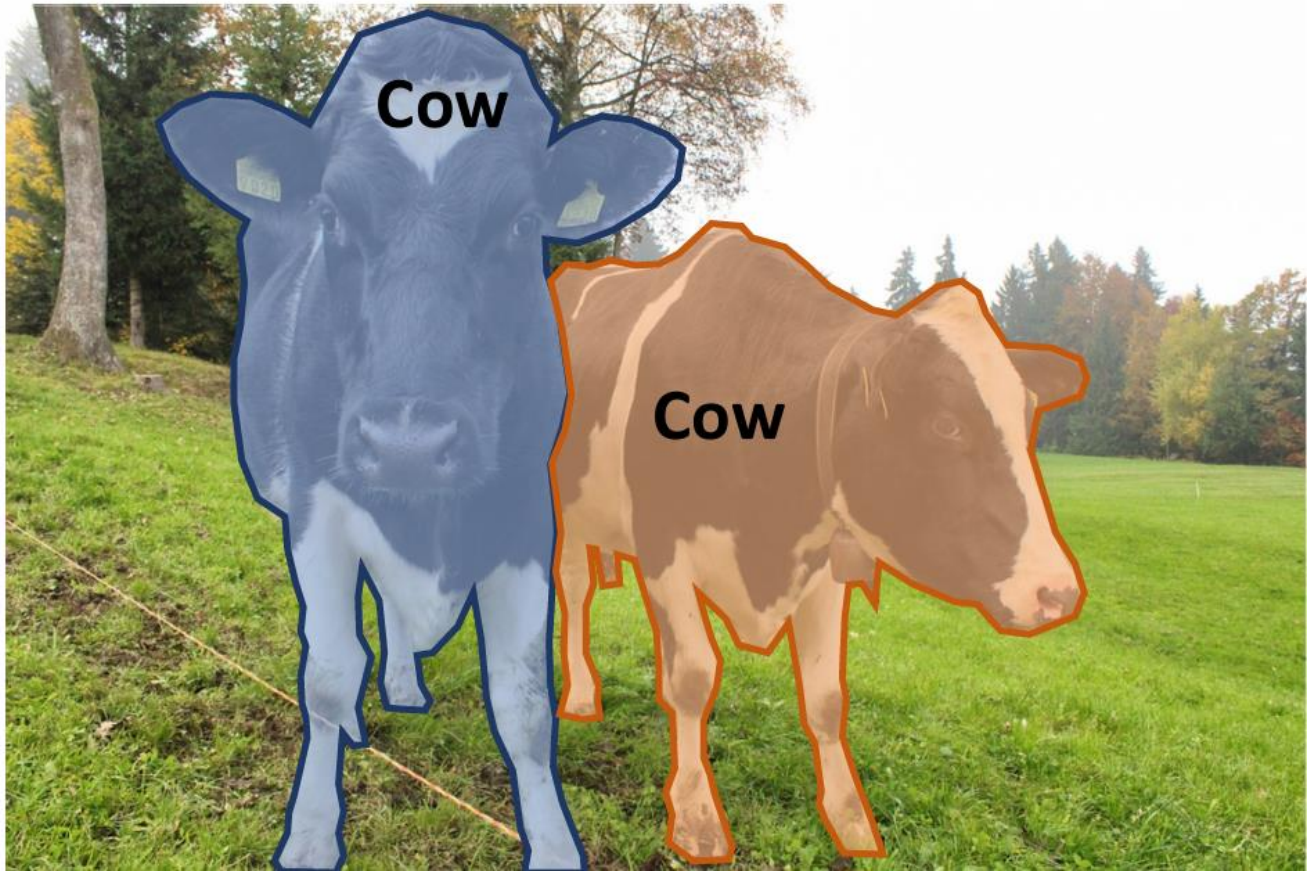
Loss function: Per-pixel cross-entropy

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

3.7 Instance Segmentation

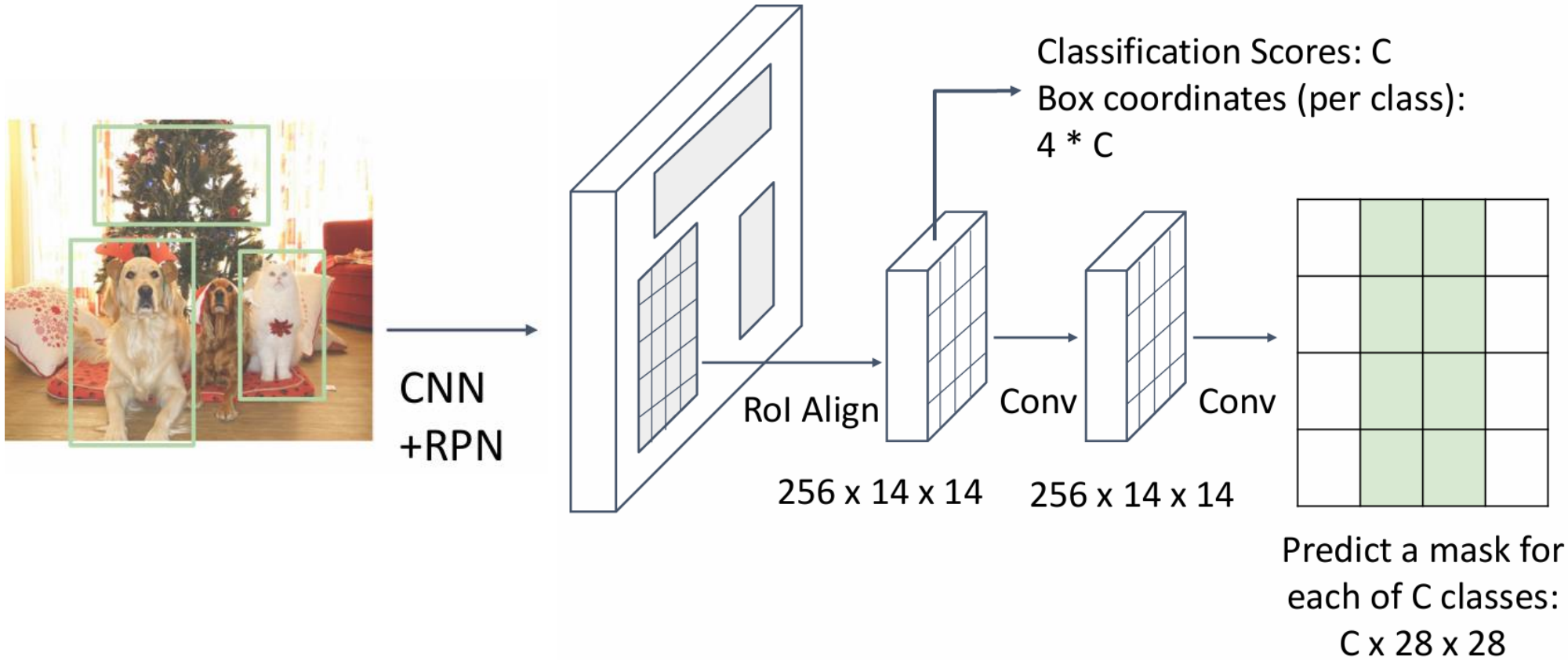
Instance Segmentation: Detect all objects in the image, and identify the pixels that belong to each object (Only things!)

Approach: Perform object detection, then predict a segmentation mask for each object!



3.7 Instance Segmentation

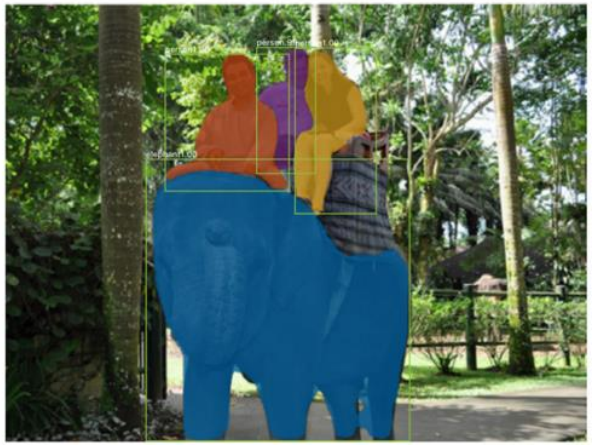
- Mask R-CNN



He et al, “Mask R-CNN”, ICCV 2017

3.7 Instance Segmentation

- Mask R-CNN Results



He et al, “Mask R-CNN”, ICCV 2017

3.8 Face Recognition

Question: Why not using Softmax classifier (as in image recognition)?

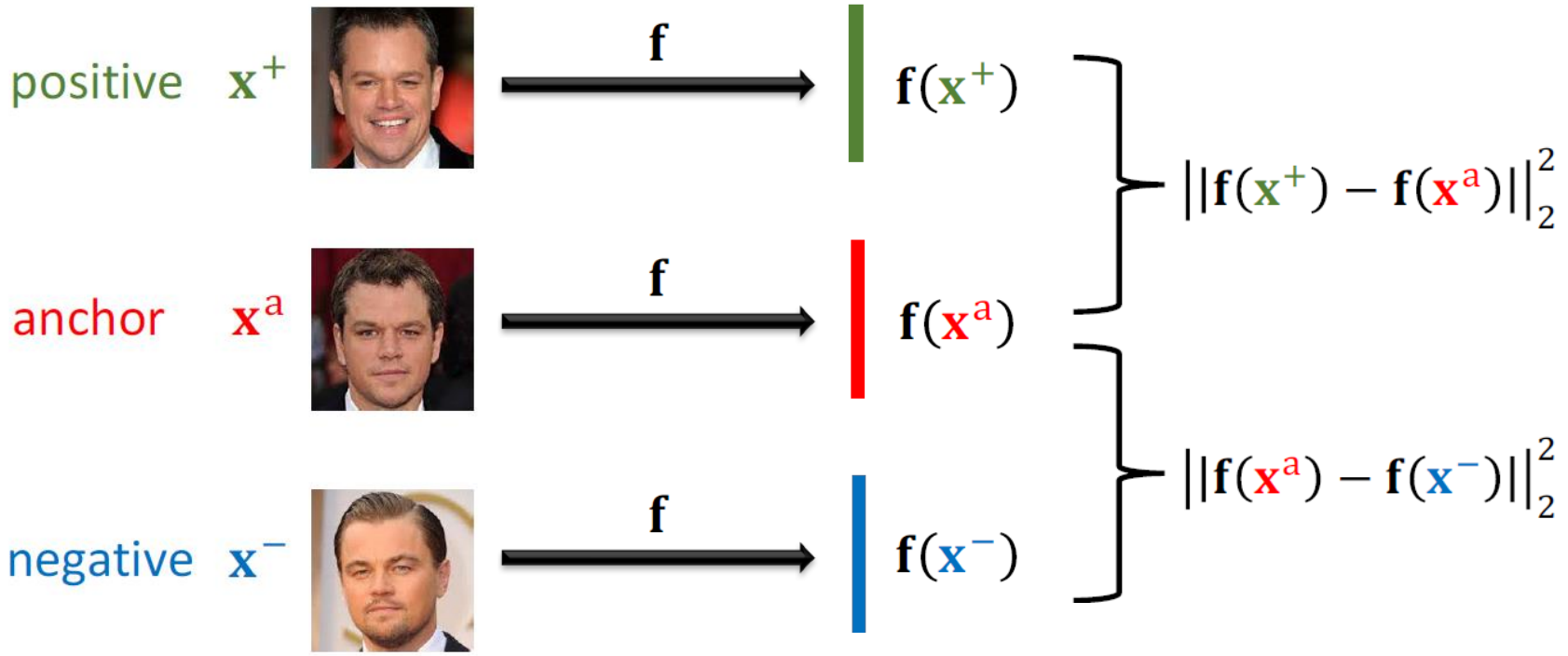
3.8 Face Recognition

Question: Why not using Softmax classifier (as in image recognition)?

- Softmax classifier is a dense layer with Softmax activation.
- #class is large (can be millions or billions).
- #parameters in the output layer is huge!
  - Suppose the input shape of Softmax classifier is $1K$.
  - Suppose #class is $10M$.
  - Then #parameters $= 1K \times 10M = 10G$.
- Faces of the same person can vary significantly due to changes in pose, lighting conditions, facial expressions, occlusions, etc. Softmax classifier might struggle to generalize well to such variability, especially when the training data for each class (person) is limited.

3.8 Face Recognition

- Training



randomly sample a triplet | Feature extraction with CNN | Loss functions

3.8 Face Recognition

- Loss Functions

  - N triplets: $(\mathbf{x}_1^{\mathrm{a}}, \mathbf{x}_1^{+}, \mathbf{x}_1^{-}), (\mathbf{x}_2^{\mathrm{a}}, \mathbf{x}_2^{+}, \mathbf{x}_2^{-}), \cdots, (\mathbf{x}_N^{\mathrm{a}}, \mathbf{x}_N^{+}, \mathbf{x}_N^{-})$.
  - N can be much larger than n (#samples).
  - Optimization model:

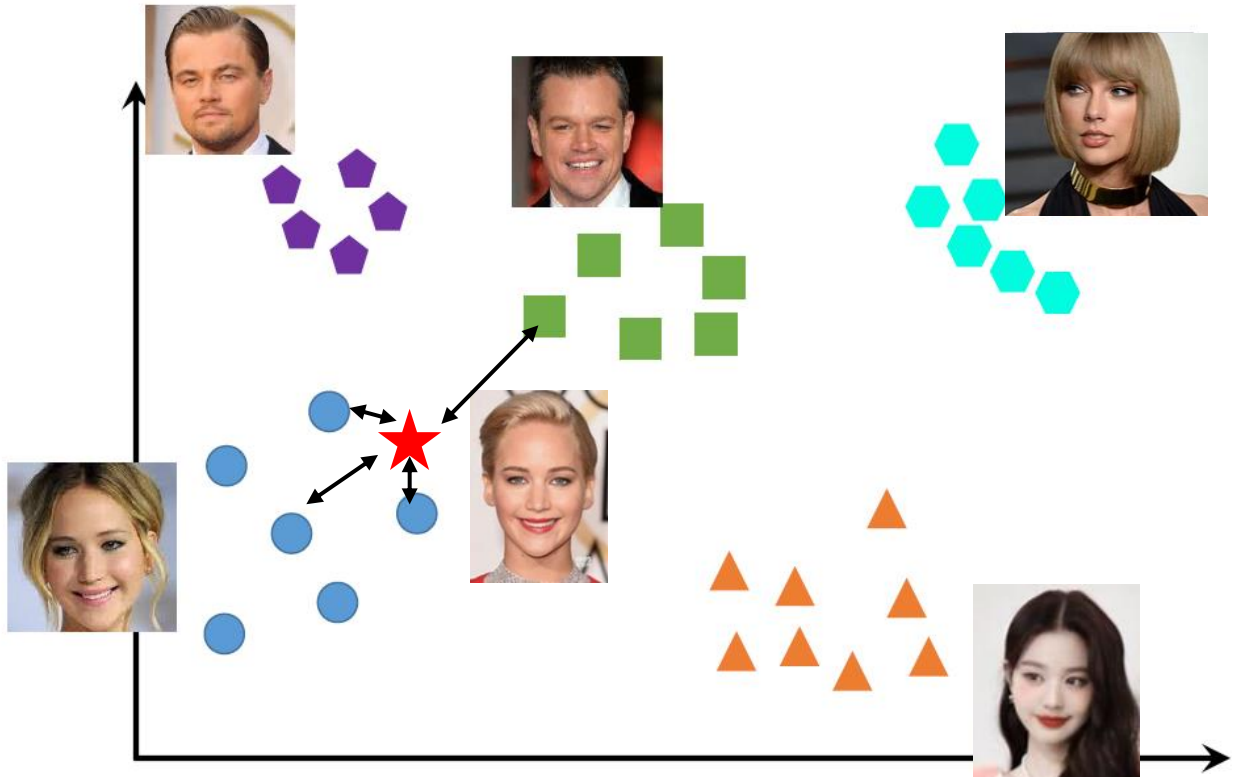
$$\min \frac{1}{N}\sum_{i=1}^{N}\left[\left\|\mathbf{f}(\mathbf{x}_i^{+})-\mathbf{f}(\mathbf{x}_i^{\mathrm{a}})\right\|_2^2-\left\|\mathbf{f}(\mathbf{x}_i^{\mathrm{a}})-\mathbf{f}(\mathbf{x}_i^{-})\right\|_2^2+\boxed{\alpha}\right]_+.$$

  - Here, $\alpha > 0$, and $[z]_+ = \max\{z, 0\}$.

3.8 Face Recognition

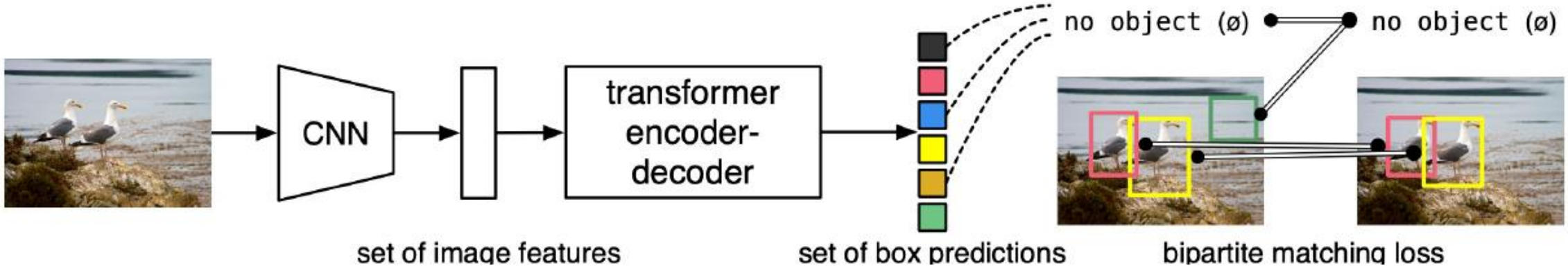
- Prediction

1. Feature extraction for all Training photos
2. Feature extraction for Test sample
3. KNN Classifier



4.1 DETR: Object Detection with Transformers

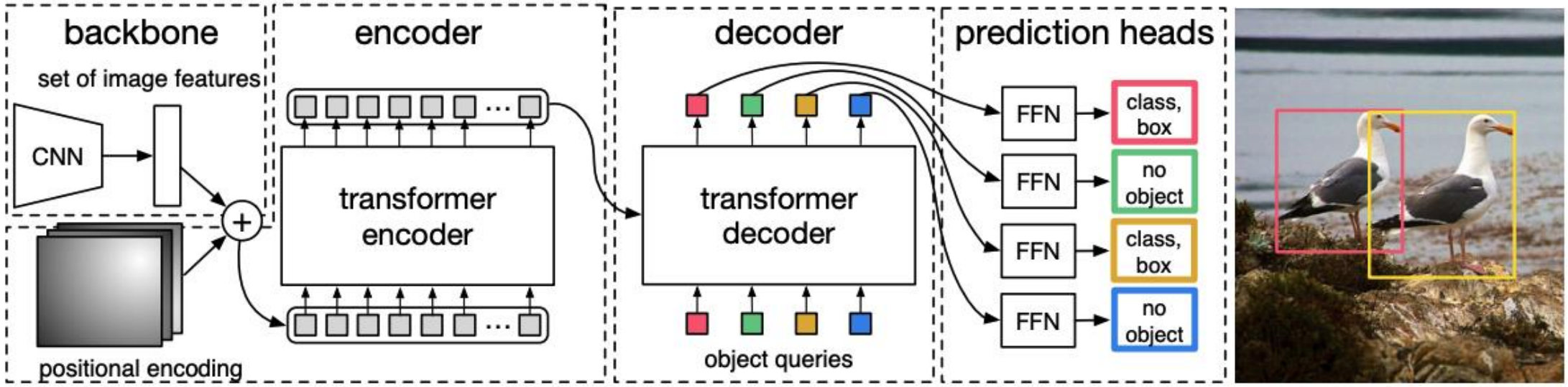
- DETR[1]: end-to-end object detection with transformers directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture.
- It does NOT rely on the many hand-designed components like in FasterRCNN.



- Propose a direct set prediction approach to bypass the surrogate tasks
- Adopt an encoder-decoder architecture based on transformers
- Self-attention models all pairwise interactions between elements in a sequence to remove duplicate predictions
- Predicts all objects at once in end-to-end manner with a set loss function which performs bipartite matching between predicted and GT objects

[1] Nicolas, et al. End-to-End Object Detection with Transformers, ECCV 2020.

4.1 DETR: Object Detection with Transformers



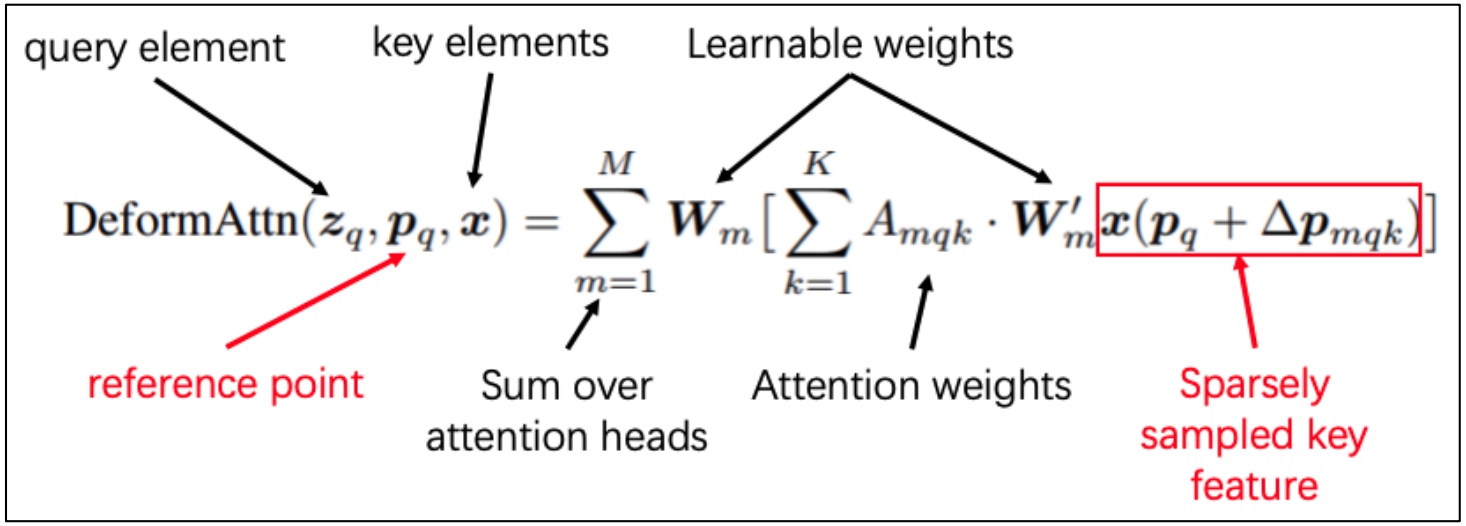
- Object Query
  - We set prior number of object query as 100
  - Object query as input to search its unique bounding box and class.
- Encoder
  - Token (pixel) looks for highly related other tokens
  - Acquire global information
- Decoder
  - Each object query (also pixel as the potential object ”center of mass”) learns the range of its focused bounding box

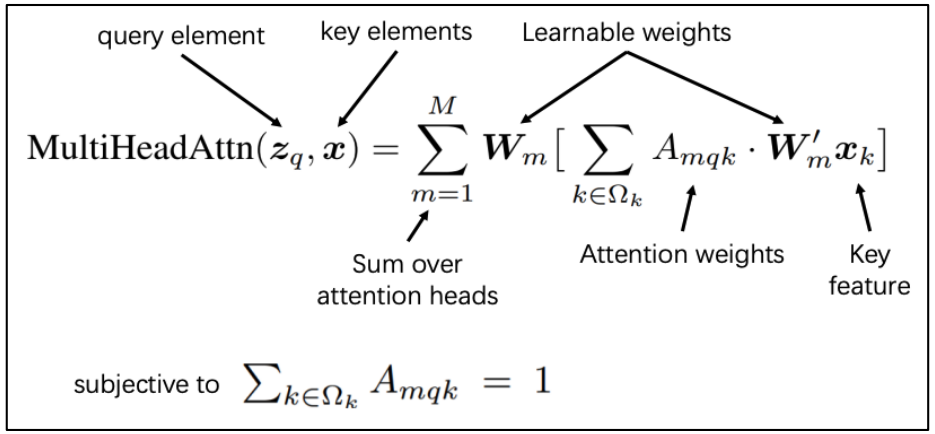
4.1 DETR: Object Detection with Transformers

- The takeaway from DETR:
  - DETR showed significant promise of generalizability, e.g., the same model easily applied to panoptic segmentation in a unified manner.
  - DETR achieved comparable performance to Faster R-CNN, but not on par with more recent detectors (especially on small objects), also requiring extra-long training schedule and auxiliary decoding losses
  - Hard to converge (requires very long training time)
- So deformable DETR is proposed to resolve the above issues
  - There is no need to dense attention to look over all possible spatial locations
  - Deformable convolution is effective on image recognition, but lacks the element relation modeling mechanism
  - So use deformable attention to only attend to a small set of points around the reference points to reduce computation complexity
  - Add in multi-scale feature maps for better small object recognition

4.2 Deformable DETR

Deformable Attention

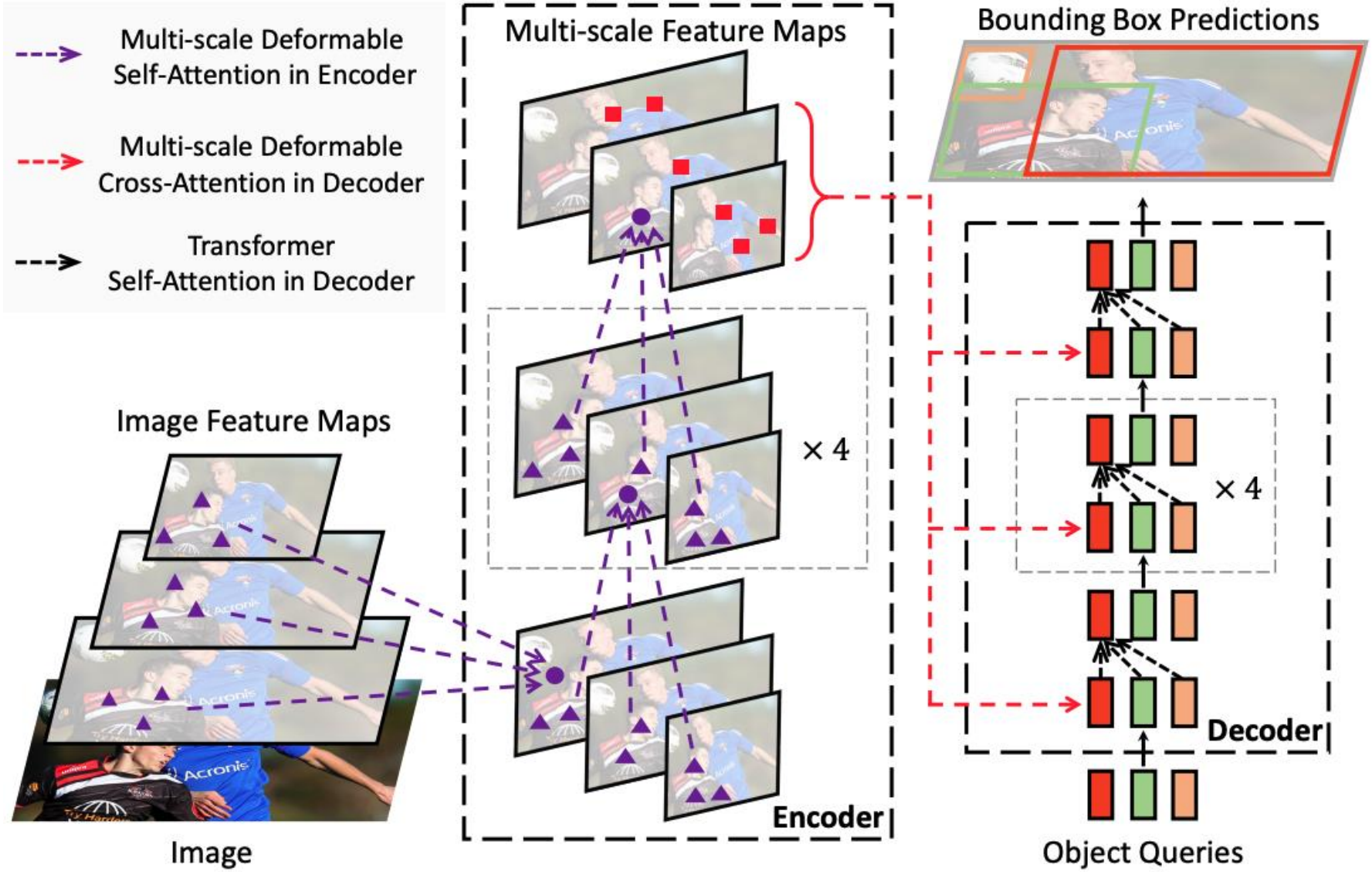




- It only attends to a small set of key sampling points around a reference point, regardless of the spatial size of the feature maps
- K is the total sampled key number ($K \ll HW$)
- In encoder, for each query pixel, the reference point $\hat{p}_q$ is itself
- In decoder, the reference point $\hat{p}_q$ is predicted from its object query embedding via a learnable linear projection followed by a sigmoid function

[1] Xizhou Zhu. Deformable DETR: Deformable Transformers for End-to-End Object Detection. ICCV 2021.

4.2 Deformable DETR



[1] Xizhou Zhu. Deformable DETR: Deformable Transformers for End-to-End Object Detection. ICCV 2021.

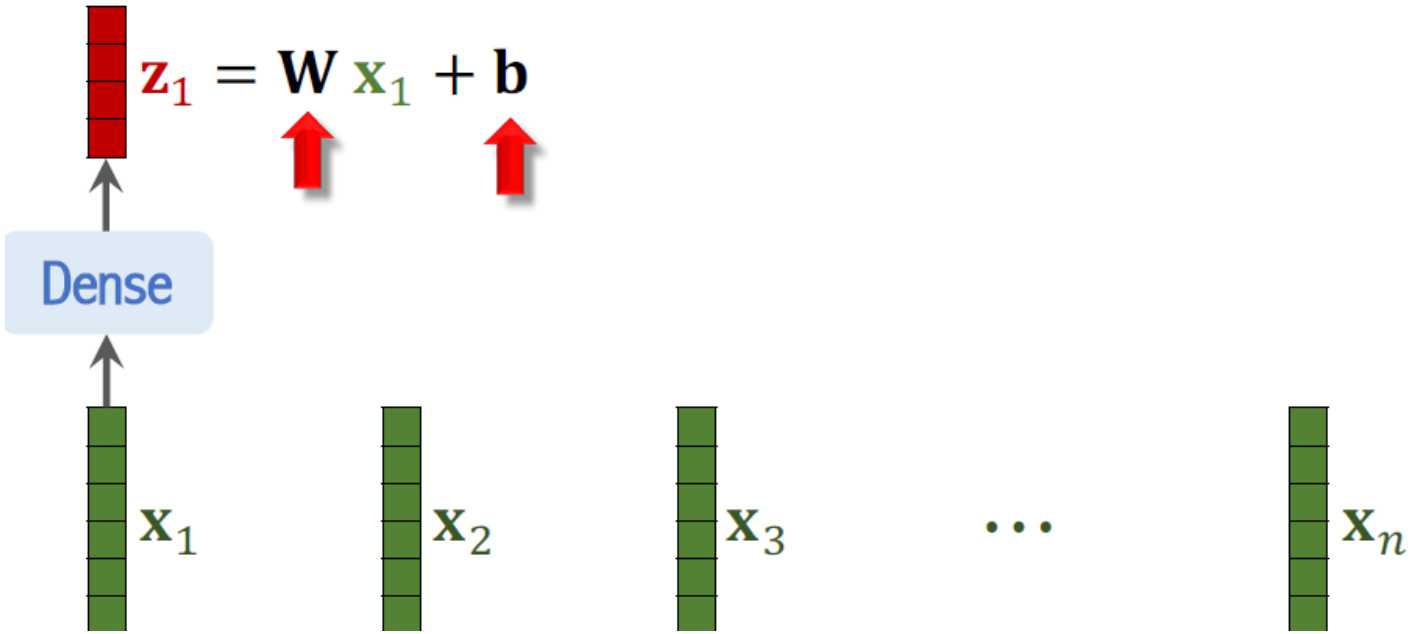
4.2 Visual Transformer

- Vision Transformer (ViT) [1] beats CNNs by a small margin, if the dataset for pretraining is sufficiently large (at least 100 million images)

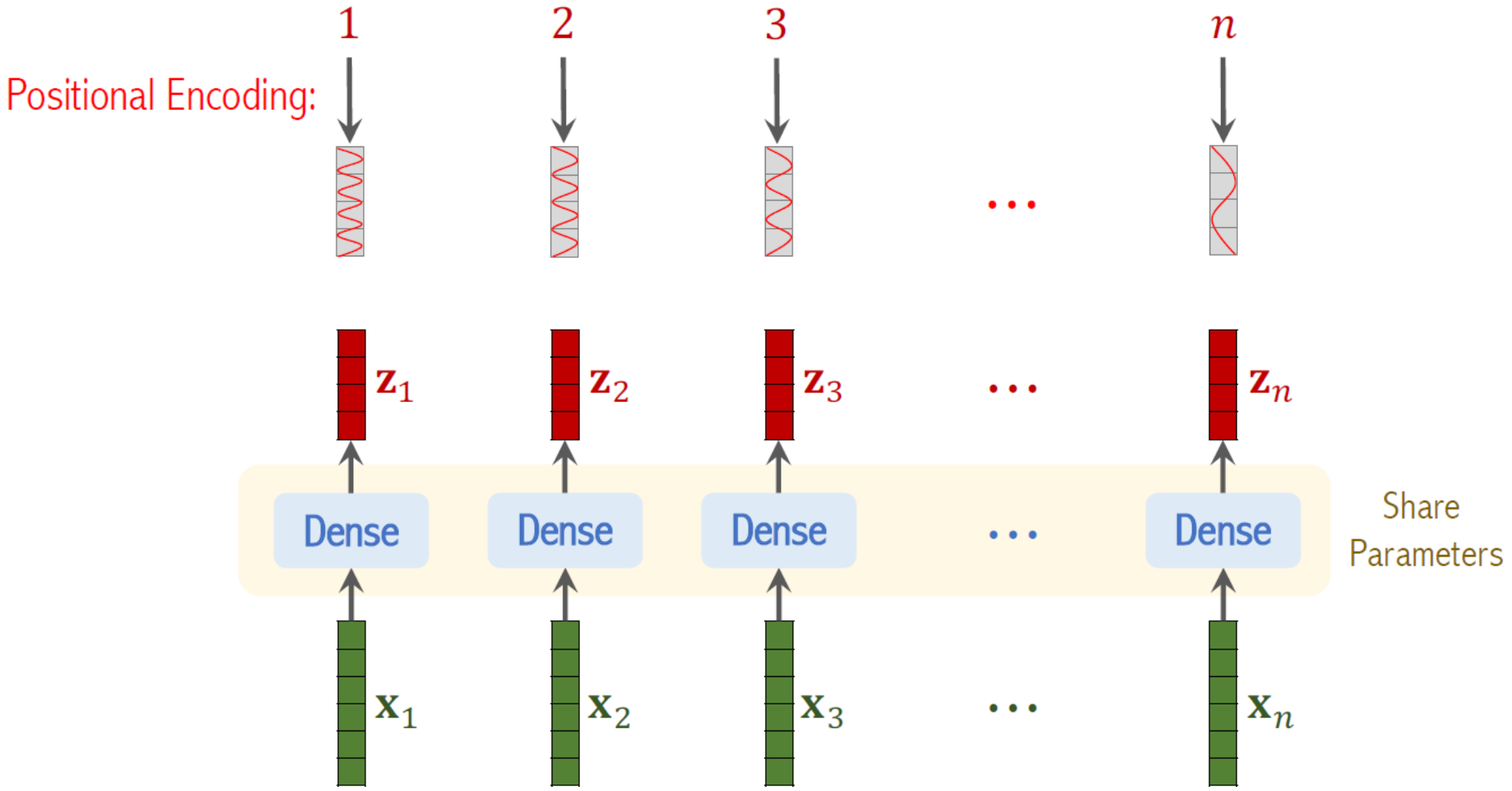
- Split image into patches (16x16)
- Vectorization
  - If the patches are $d_1 \times d_2 \times d_3$ tensors, then the vectors are $d_1 d_2 d_3 \times 1$

[1] Dosovitskiy et al. An image is worth 16×16 words: transformers for image recognition at scale. In ICLR, 2021.

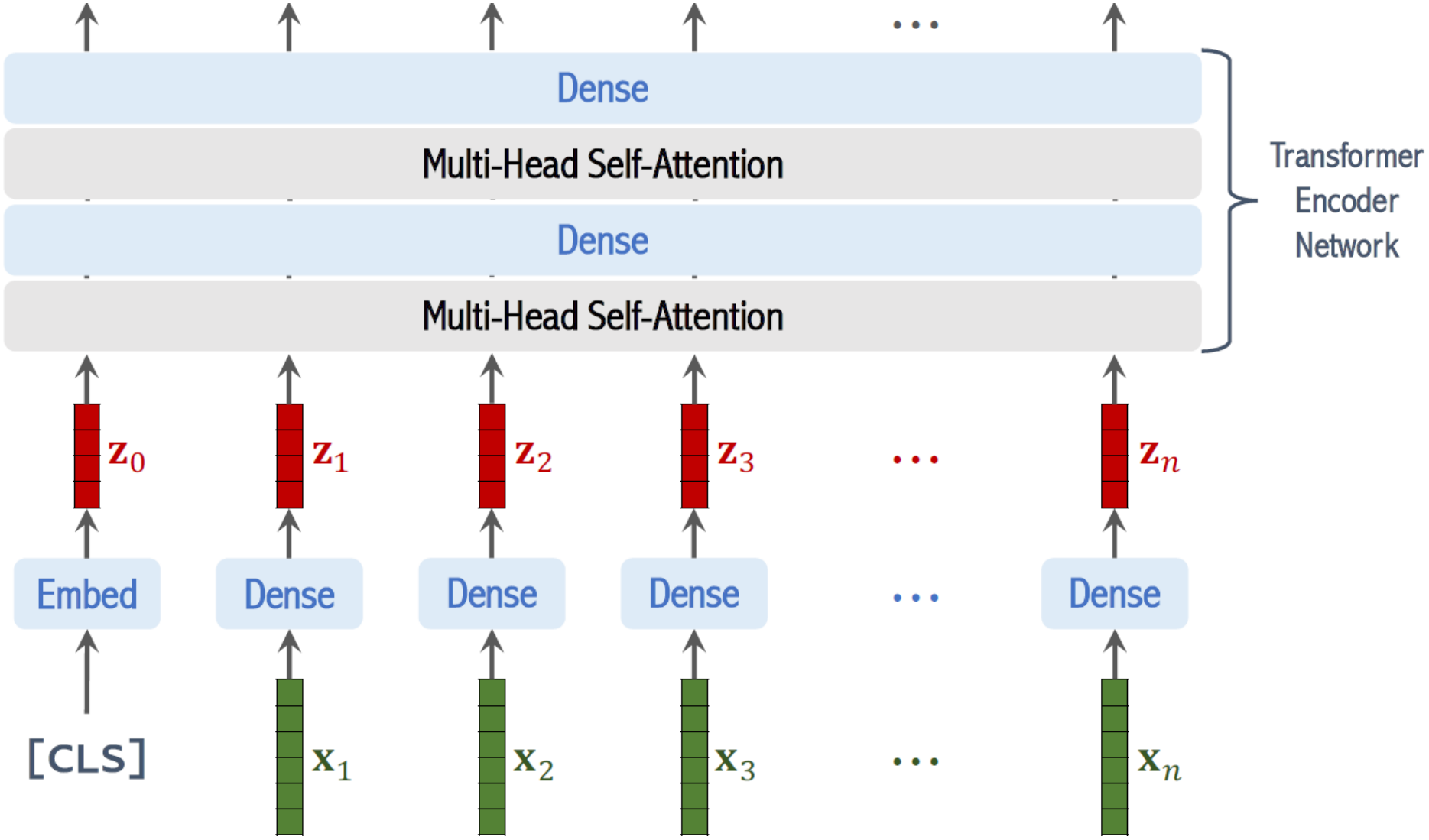
4.3 Visual Transformer



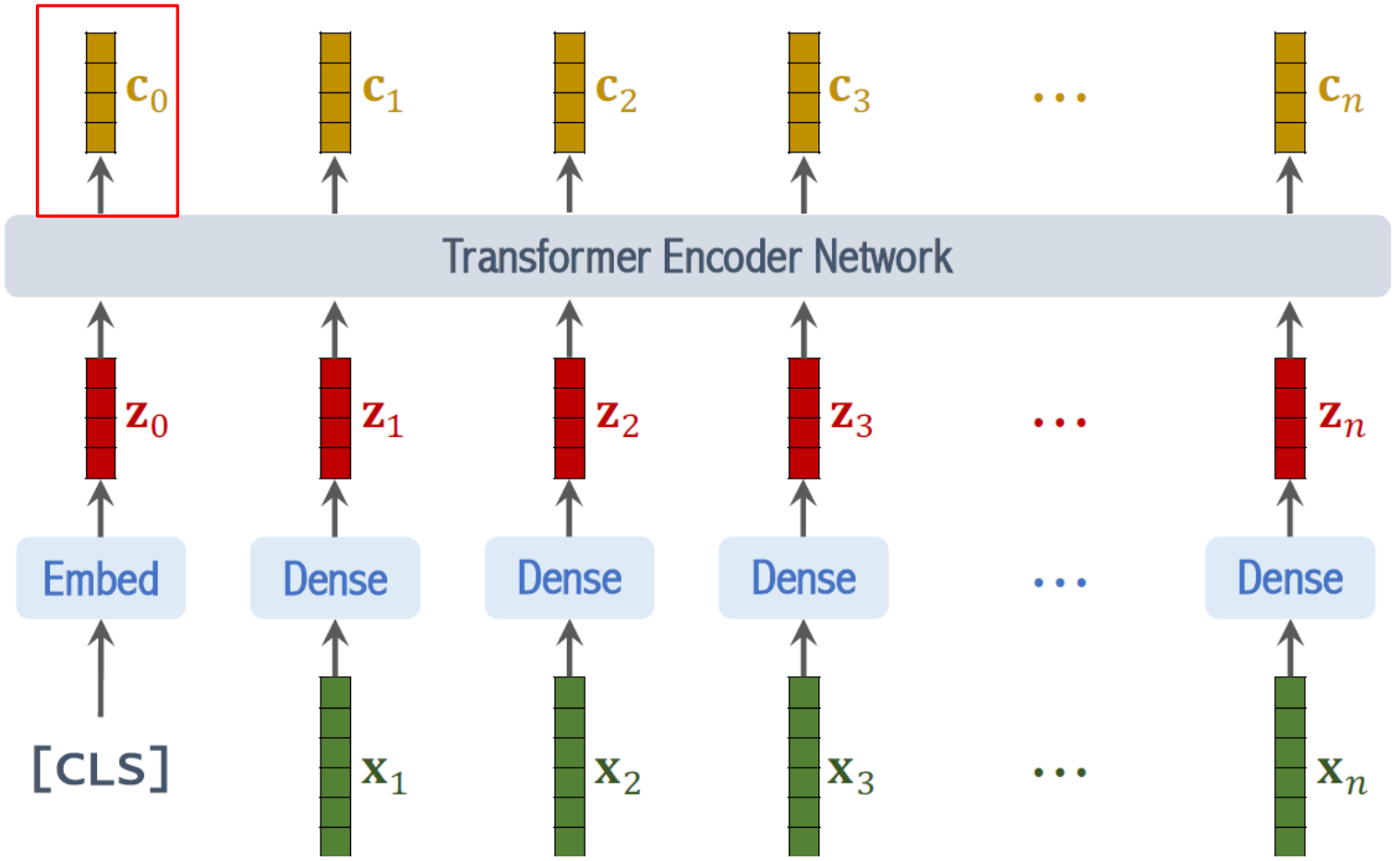
4.3 Visual Transformer



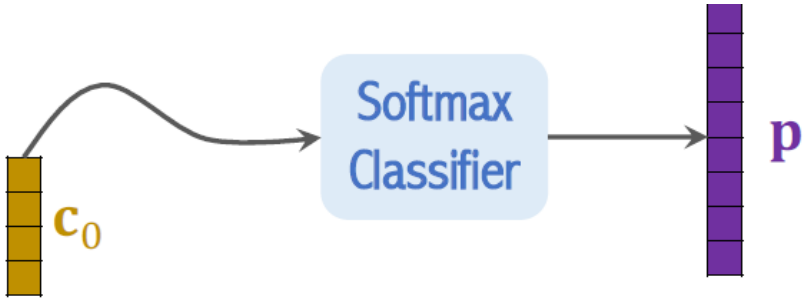
4.3 Visual Transformer

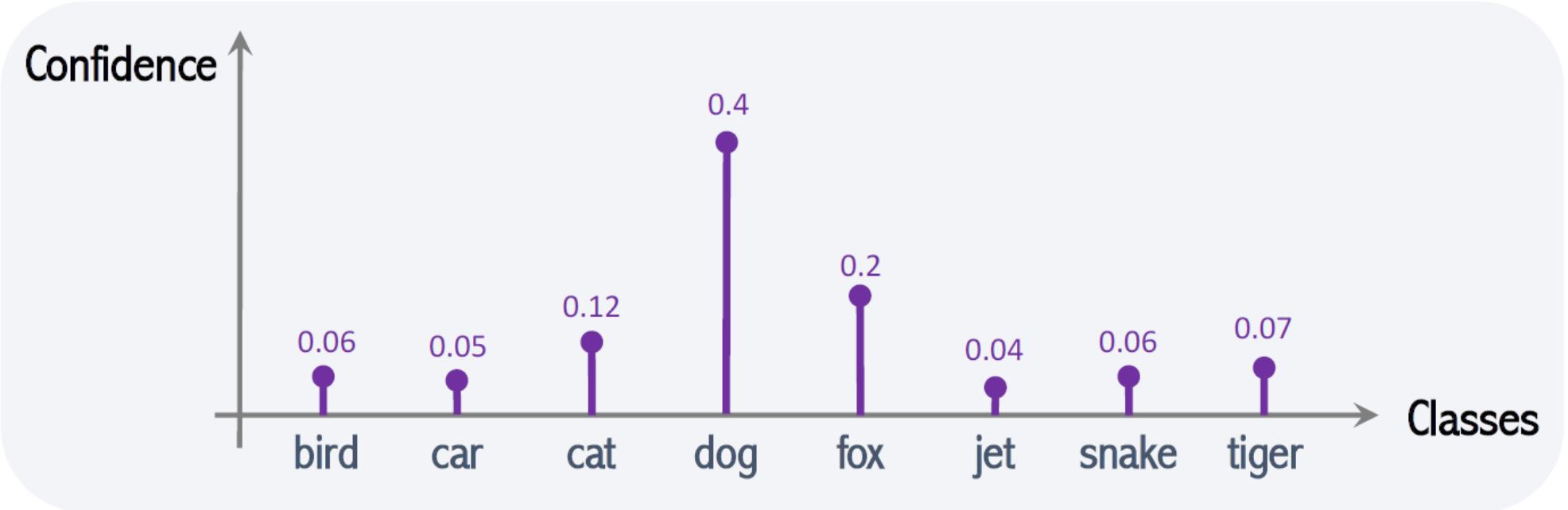


4.3 Visual Transformer



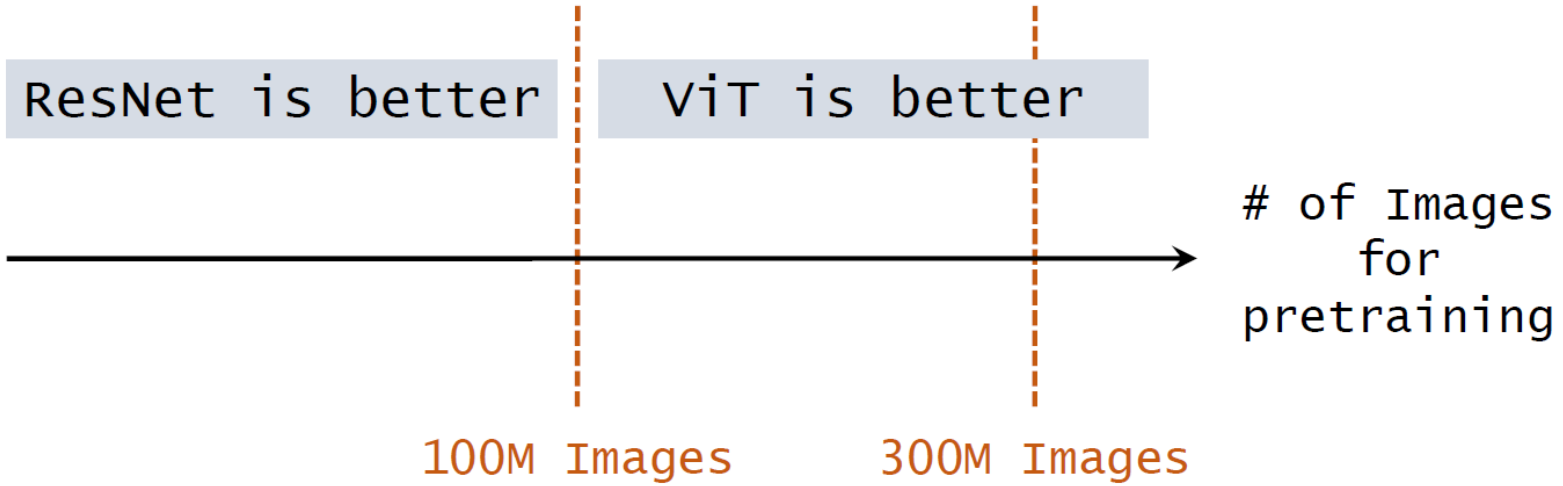
4.3 Visual Transformer



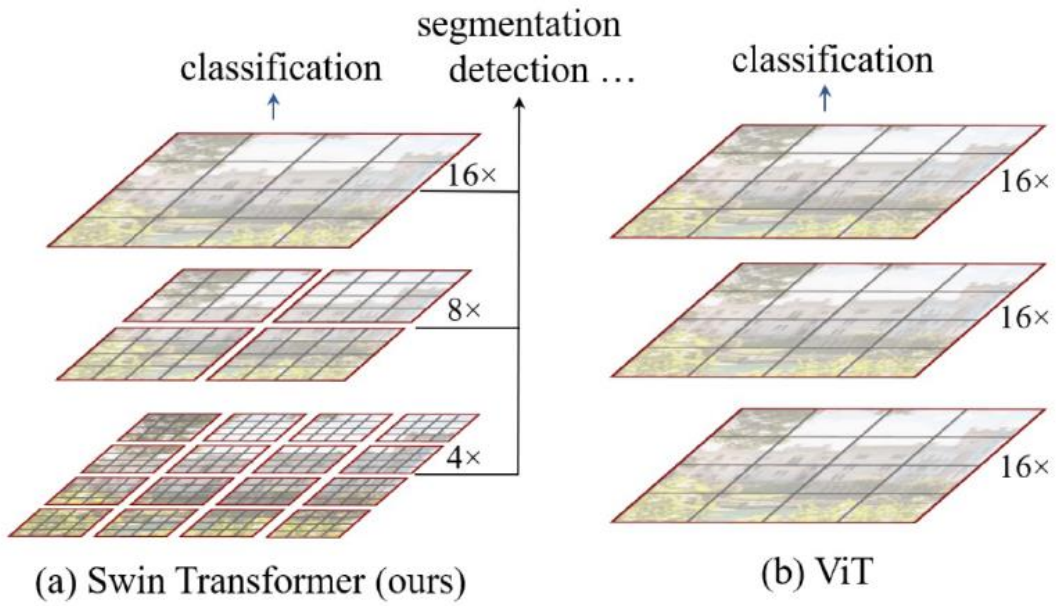


4.3 Visual Transformer

- Pretrained on ImageNet (small), ViT is slightly worse than ResNet.
- Pretrained on ImageNet-21K (medium), ViT is comparable to ResNet.
- Pretrained on JFT (large), ViT is slightly better than ResNet.



4.4 Swin Transformer (ICCV'21 best paper)



- Swin: hierarchical feature maps by merging image patches
  - Linear computation complexity to input image size due to computation of self-attention
- Only within each local window (using Shifted windows)

4.4 Swin Transformer (ICCV'21 best paper)

- Pipeline

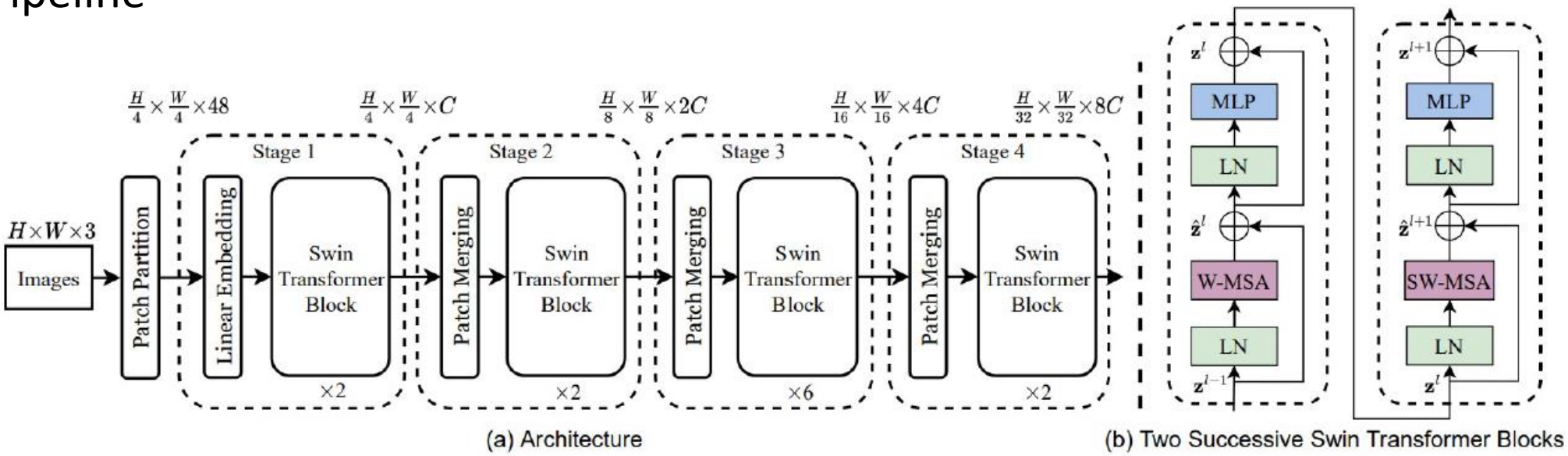
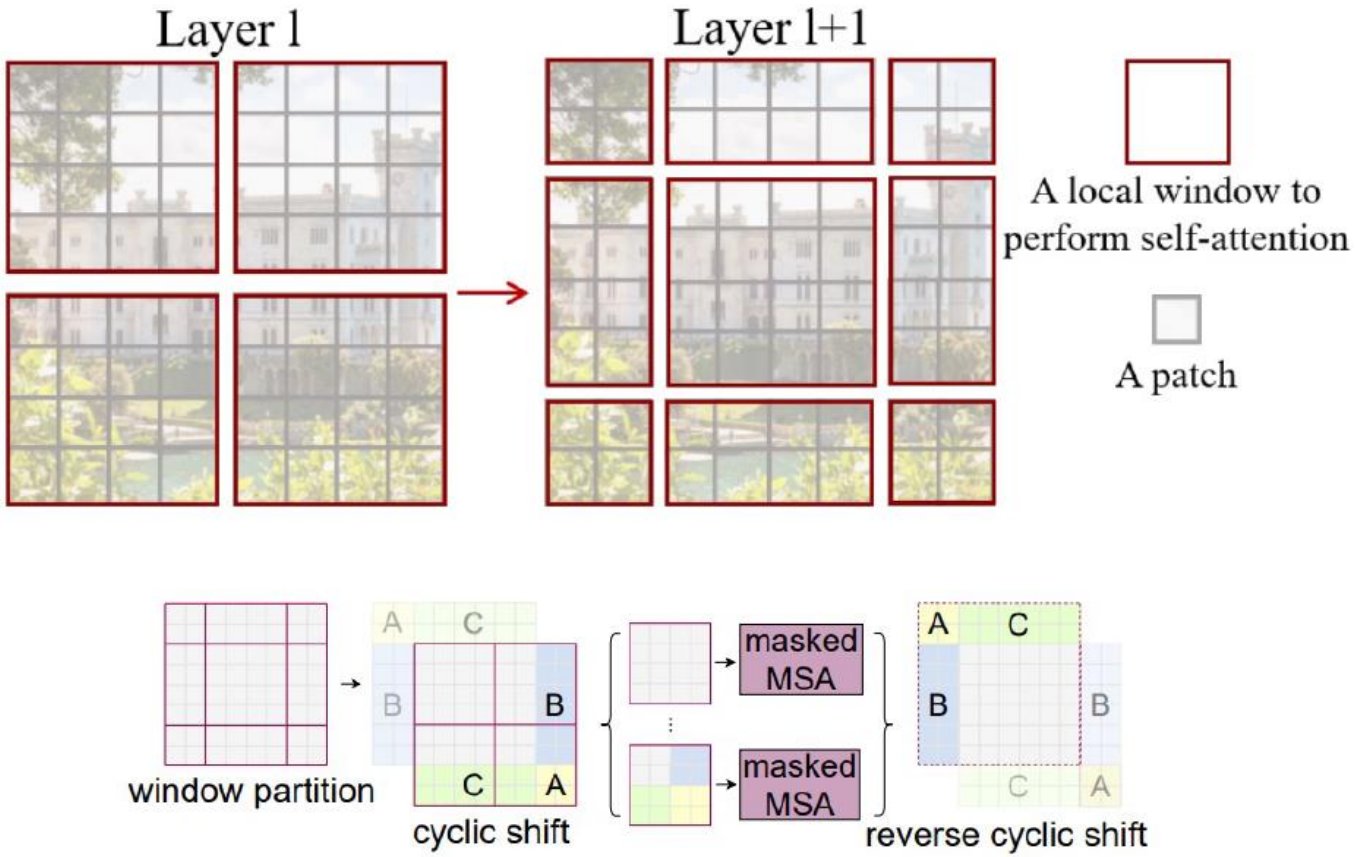


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

4.4 Swin Transformer (ICCV'21 best paper)

- Shifted Window



Thank you!

- Meta Learning and AutoML
  - NAS (network architecture search)
  - Hyperparameter auto-tuning (grid-search vs. random search)
- Model pruning
- Model quantization (PTQ, QAT, etc.)
- Knowledge Distillation
- …