Convolutional Neural Networks

Shan-Hung Wu
shwu@cs.nthu.edu.tw

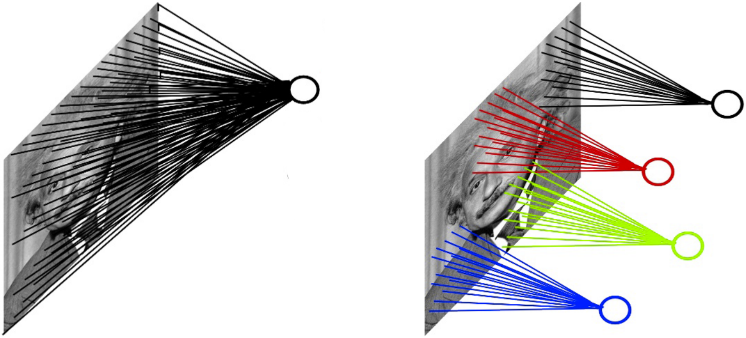
Department of Computer Science,
National Tsing Hua University, Taiwan

Machine Learning

Outline

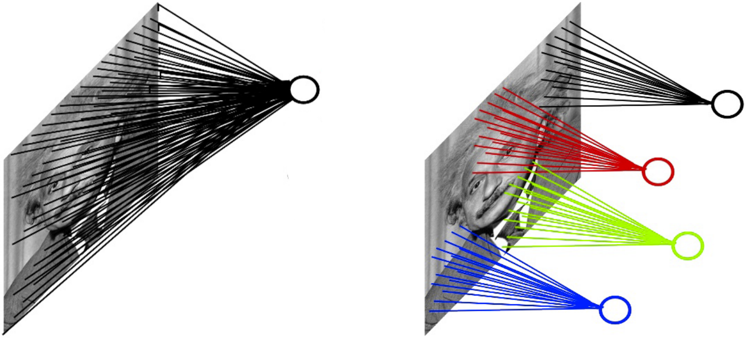
Outline

CNNs for Image Data



- Convolutional Neural Networks (CNNs) as ***regularized*** NNs
  - Incorporate image-specific prior knowledge in model design

CNNs for Image Data



- Convolutional Neural Networks (CNNs) as ***regularized*** NNs
  - Incorporate image-specific prior knowledge in model design
- What prior knowledge?

Image-Specific Priors (1/2)

- Patterns in an images are location independent

Image-Specific Priors (1/2)

- Patterns in an images are location independent
- Drives development of ***convolution layers***
  - ***Pruned*** and ***tied*** weights between neurons at successive layers
  - Significantly improves learning efficiency

Image-Specific Priors (1/2)

- Patterns can be zoomed (more specifically, downsampled)

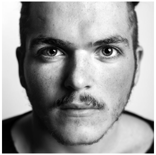
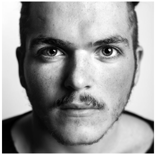


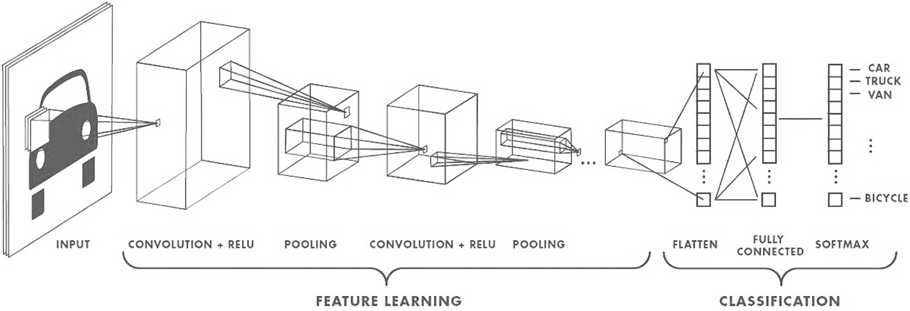
Image-Specific Priors (1/2)

- Patterns can be zoomed (more specifically, downsampled)
- Drives development of ***pooling layers***
  - ***Reduce #neurons*** at deeper layers to save computation

Typical CNN Architecture

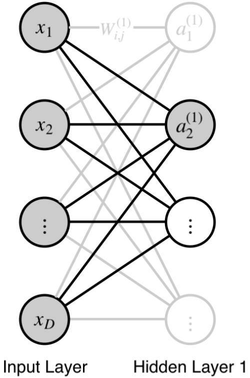
- Convolution and pooling layers learn hidden representations
  - Capture local image patterns
- Fully connected layers in the end
  - Serve as classifier, regressor, etc.

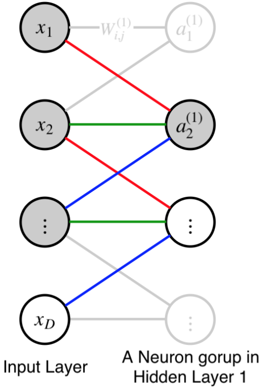


Outline

Capturing location Independent Patterns

- In a fully connected layer, each neuron is connected to all pixels
- A convolution layer:
  - Divides neurons into groups
  - Lets each group of neurons capture a particular local pattern at different locations
- ***Pruned*** and ***tied*** weights; ***zero-padding*** at image boundary

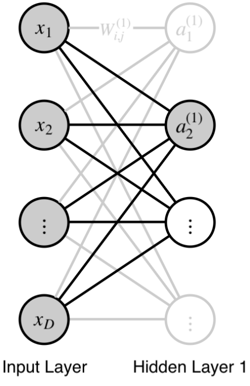


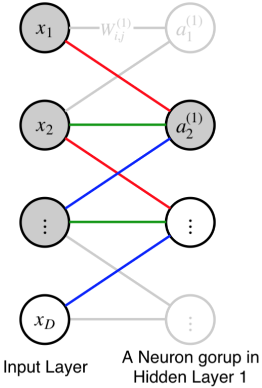


Filters and Feature Maps

$$a_i^{(l)} = \text{act}^{(l)}\left([a_{i-K^{(l)}/2}^{(l-1)}, \cdots, a_i^{(l-1)}, \cdots, a_{i+K^{(l)}/2}^{(l-1)}] \begin{bmatrix} w_1 \\ \cdots \\ w_{K^{(l)}} \end{bmatrix} + b^{(l)}\right)$$

- Tied weights $[w_1, \ldots, w_{K^{(l)}}]$ for a group are called a ***filter*** or ***kernel***
- Activations per group are called ***feature map*** or ***activation map***



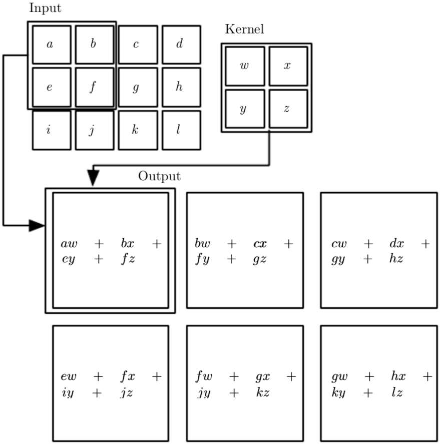


2D Convolution

- 1D input $*$ 1D filter $\rightarrow$ 1D activation map ($*$ as "scanned by")
  - $\boldsymbol{x} \in \mathbb{R}^D * \boldsymbol{w} \in \mathbb{R}^K \rightarrow \boldsymbol{a} \in \mathbb{R}^D$

2D Convolution

- 1D input $*$ 1D filter $\rightarrow$ 1D activation map ($*$ as "scanned by")
  - $\boldsymbol{x} \in \mathbb{R}^D * \boldsymbol{w} \in \mathbb{R}^K \rightarrow \boldsymbol{a} \in \mathbb{R}^D$
- 2D input $*$ 2D filter $\rightarrow$ 2D activation map
  - $\boldsymbol{X} \in \mathbb{R}^{W \times H} * \boldsymbol{W} \in \mathbb{R}^{K \times K} \rightarrow \boldsymbol{A} \in \mathbb{R}^{W \times H}$



Why Called Convolution? (1/2)

- Suppose we are tracking the location $y(t)$ of a car with a GPS sensor at time t
  - Let $x(t)$ be a GPS reading at time t

[1] In this example, w should satisfy $\int w(t-s)ds = 1$ and $w(t-s) = 0$ if $s < 0$

Why Called Convolution? (1/2)

- Suppose we are tracking the location $y(t)$ of a car with a GPS sensor at time t
  - Let $x(t)$ be a GPS reading at time t
  - If GPS sensor is noisy, we can obtain an estimate of $y(t)$ by the ***convolution*** of the function $x(\cdot)$ with a weighting function $w(\cdot)$:[1]

$$\begin{aligned} y(t) &\approx (x * w)(t) \\ &= \int x(s)w(t-s)ds \text{ (continous time), or} \\ &= \sum_s x(s)w(t-s) \text{ (discrete time),} \end{aligned}$$

[1] In this example, w should satisfy $\int w(t-s)ds = 1$ and $w(t-s) = 0$ if $s < 0$

Why Called Convolution? (1/2)

- Suppose we are tracking the location $y(t)$ of a car with a GPS sensor at time t
  - Let $x(t)$ be a GPS reading at time t
  - If GPS sensor is noisy, we can obtain an estimate of $y(t)$ by the ***convolution*** of the function $x(\cdot)$ with a weighting function $w(\cdot)$:[1]

$$\begin{aligned} y(t) &\approx (x * w)(t) \\ &= \int x(s)w(t-s)ds \text{ (continous time), or} \\ &= \textstyle\sum_s x(s)w(t-s) \text{ (discrete time),} \end{aligned}$$

  - $x(\cdot)$ is called ***input***
  - $w(\cdot)$ is called ***kernel***

[1] In this example, w should satisfy $\int w(t-s)ds = 1$ and $w(t-s) = 0$ if $s < 0$

Why Called Convolution? (2/2)

- Given discrete time and 2D input, we have

$$(x*w)(i,j)=\sum_u\sum_v x(u,v)w(i-u,j-v)$$

Why Called Convolution? (2/2)

- Given discrete time and 2D input, we have

$$(x*w)(i,j)=\sum_u\sum_v x(u,v)w(i-u,j-v)$$

- Convolution is commutative, thus

$$(x*w)(i,j)=\sum_u\sum_v x(i-u,j-v)w(u,v)$$

Why Called Convolution? (2/2)

- Given discrete time and 2D input, we have

$$(x*w)(i,j)=\sum_u\sum_v x(u,v)w(i-u,j-v)$$

- Convolution is commutative, thus

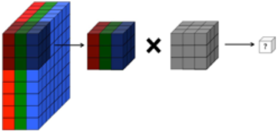
$$(x*w)(i,j)=\sum_u\sum_v x(i-u,j-v)w(u,v)$$

- 2D convolution in CNNs:
  - $x(i-u,j-v)$ as image pixels
  - $w(u,v)$ as a filter dimension $W_{u,v}$
  - $(x*w)(i,j)$ as an activation $A_{i,j}$ in a feature map
  - Operator $*$ as a scanning step

Supporting Color Images

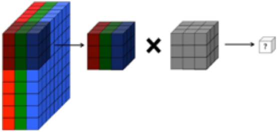
- A color image has multiple channels
- How to extend 2D convolution?
  1. 3D input $*$ 2D filter $\rightarrow$ 3D activation map
     $\mathbf{X} \in \mathbb{R}^{W \times H \times 3} * \mathbf{W} \in \mathbb{R}^{K \times K} \rightarrow \boldsymbol{A} \in \mathbb{R}^{W \times H \times 3}$
  2. 3D input $*$ 3D filter $\rightarrow$ 2D activation map
     $\mathbf{X} \in \mathbb{R}^{W \times H \times 3} * \mathbf{W} \in \mathbb{R}^{K \times K \times 3} \rightarrow \boldsymbol{A} \in \mathbb{R}^{W \times H}$

Supporting Color Images



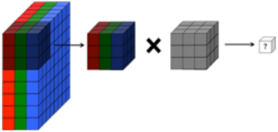
- A color image has multiple channels
- How to extend 2D convolution?
  1. 3D input $*$ 2D filter $\rightarrow$ 3D activation map
     $\mathbf{X} \in \mathbb{R}^{W \times H \times 3} * \mathbf{W} \in \mathbb{R}^{K \times K} \rightarrow \boldsymbol{A} \in \mathbb{R}^{W \times H \times 3}$ X
  2. 3D input $*$ 3D filter $\rightarrow$ 2D activation map
     $\mathbf{X} \in \mathbb{R}^{W \times H \times 3} * \mathbf{W} \in \mathbb{R}^{K \times K \times 3} \rightarrow \boldsymbol{A} \in \mathbb{R}^{W \times H}$ ✓
- Because humans ***see through channels***

Supporting Color Images



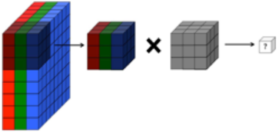
- A color image has multiple channels
- How to extend 2D convolution?
  1. 3D input $*$ 2D filter $\rightarrow$ 3D activation map
     $\mathbf{X} \in \mathbb{R}^{W \times H \times 3} * \mathbf{W} \in \mathbb{R}^{K \times K} \rightarrow \boldsymbol{A} \in \mathbb{R}^{W \times H \times 3}$ X
  2. 3D input $*$ 3D filter $\rightarrow$ 2D activation map
     $\mathbf{X} \in \mathbb{R}^{W \times H \times 3} * \mathbf{W} \in \mathbb{R}^{K \times K \times 3} \rightarrow \boldsymbol{A} \in \mathbb{R}^{W \times H}$ ✓
- Because humans ***see through channels***
- Still called 2D convolution because the scanning moves in 2D

Supporting Color Images



- A color image has multiple channels
- How to extend 2D convolution?
  1. 3D input $*$ 2D filter $\to$ 3D activation map
     $\mathbf{X} \in \mathbb{R}^{W \times H \times 3} * \mathbf{W} \in \mathbb{R}^{K \times K} \to \boldsymbol{A} \in \mathbb{R}^{W \times H \times 3}$ X
  2. 3D input $*$ 3D filter $\to$ 2D activation map
     $\mathbf{X} \in \mathbb{R}^{W \times H \times 3} * \mathbf{W} \in \mathbb{R}^{K \times K \times 3} \to \boldsymbol{A} \in \mathbb{R}^{W \times H}$ ✓
- Because humans ***see through channels***
- Still called 2D convolution because the scanning moves in 2D
- When to use 3D convolution?

Supporting Color Images



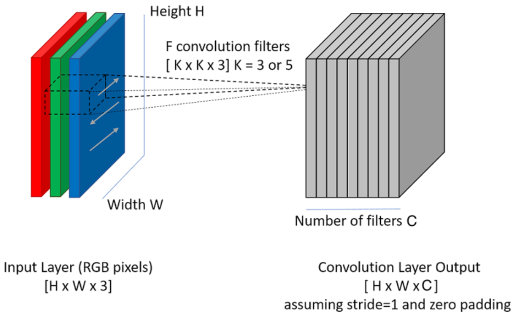
- A color image has multiple channels
- How to extend 2D convolution?
  1. 3D input $*$ 2D filter $\rightarrow$ 3D activation map
     $\mathbf{X} \in \mathbb{R}^{W \times H \times 3} * \mathbf{W} \in \mathbb{R}^{K \times K} \rightarrow \boldsymbol{A} \in \mathbb{R}^{W \times H \times 3}$ X
  2. 3D input $*$ 3D filter $\rightarrow$ 2D activation map
     $\mathbf{X} \in \mathbb{R}^{W \times H \times 3} * \mathbf{W} \in \mathbb{R}^{K \times K \times 3} \rightarrow \boldsymbol{A} \in \mathbb{R}^{W \times H}$ ✓
- Because humans ***see through channels***
- Still called 2D convolution because the scanning moves in 2D
- When to use 3D convolution? E.g., videos as 4D input
  - To detect spatially & temporally local pattern
  - 4D input $*$ 4D filter $\rightarrow$ 3D activation map

Detecting Multiple Local Patterns

- Suppose we use $C^{(l)}$ filters to detect local patterns at layer l
- After convolution, we have $C^{(l)}$ feature maps: $\{\boldsymbol{A}^{(l,1)}, \cdots, \boldsymbol{A}^{(l,C)}\}$
- How to form the input for the next layer $l+1$?

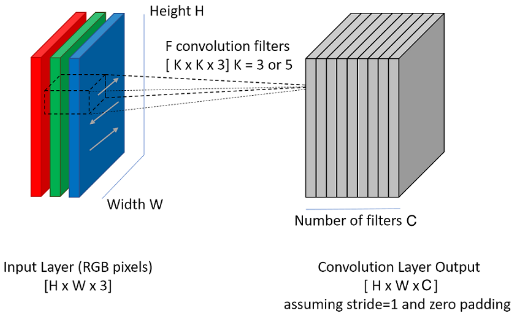
Detecting Multiple Local Patterns

- Suppose we use $C^{(l)}$ filters to detect local patterns at layer l
- After convolution, we have $C^{(l)}$ feature maps: $\{\boldsymbol{A}^{(l,1)}, \cdots, \boldsymbol{A}^{(l,C)}\}$
- How to form the input for the next layer $l+1$?
- Stack up the activation maps *along the channel dimension*
  - $\mathbf{A}^{W \times H \times C^{(l)}}$ as the input



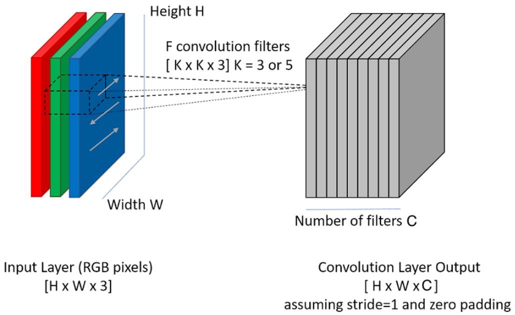
Detecting Multiple Local Patterns

- Suppose we use $C^{(l)}$ filters to detect local patterns at layer l
- After convolution, we have $C^{(l)}$ feature maps: $\{\boldsymbol{A}^{(l,1)}, \cdots, \boldsymbol{A}^{(l,C)}\}$
- How to form the input for the next layer $l+1$?
- Stack up the activation maps ***along the channel dimension***
  - $\mathbf{A}^{W \times H \times C^{(l)}}$ as the input
- Dimension of a filter in the next layer?



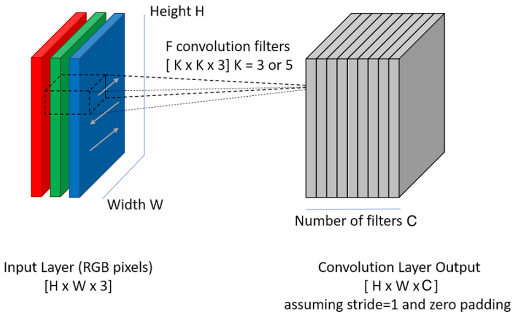
Detecting Multiple Local Patterns

- Suppose we use $C^{(l)}$ filters to detect local patterns at layer l
- After convolution, we have $C^{(l)}$ feature maps: $\{\boldsymbol{A}^{(l,1)}, \cdots, \boldsymbol{A}^{(l,C)}\}$
- How to form the input for the next layer $l+1$?
- Stack up the activation maps ***along the channel dimension***
  - $\mathbf{A}^{W \times H \times C^{(l)}}$ as the input
- Dimension of a filter in the next layer? $K^{(l+1)} \times K^{(l+1)} \times C^{(l)}$



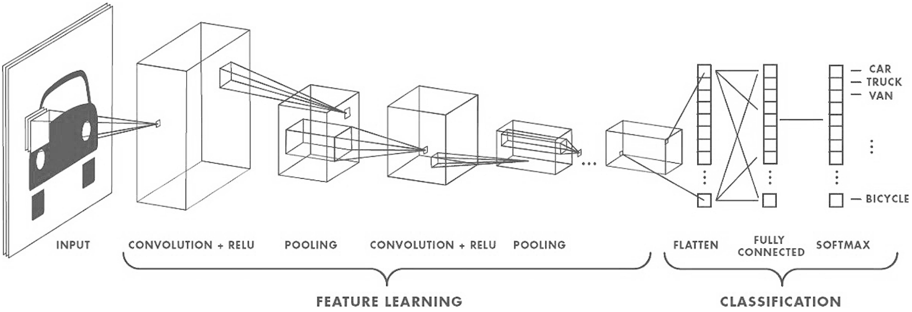
Detecting Multiple Local Patterns

- Suppose we use $C^{(l)}$ filters to detect local patterns at layer l
- After convolution, we have $C^{(l)}$ feature maps: $\{\boldsymbol{A}^{(l,1)}, \cdots, \boldsymbol{A}^{(l,C)}\}$
- How to form the input for the next layer $l+1$?
- Stack up the activation maps ***along the channel dimension***
  - $\mathbf{A}^{W \times H \times C^{(l)}}$ as the input
- Dimension of a filter in the next layer? $K^{(l+1)} \times K^{(l+1)} \times C^{(l)}$
- What does the filter in the next layer learn?



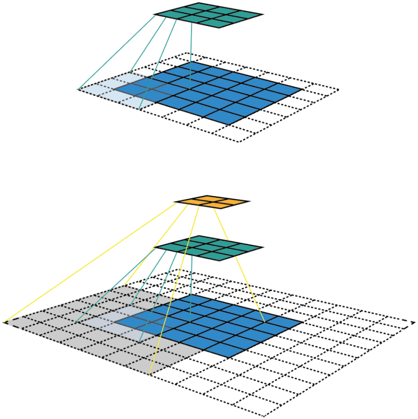
Learning Deep Local Patterns (1/3)

- Feature map is ***equivariant*** to translation
  - A function f is equivariant to a function g if $f(g(x)) = g(f(x))$
  - $g(x)$ as translated input
  - $f(x)$ as feature map
- Stacking up feature maps along channel dimension allows a deep filter to ***see through patterns in the same local region*** to detect new patterns



Learning Deep Local Patterns (2/3)

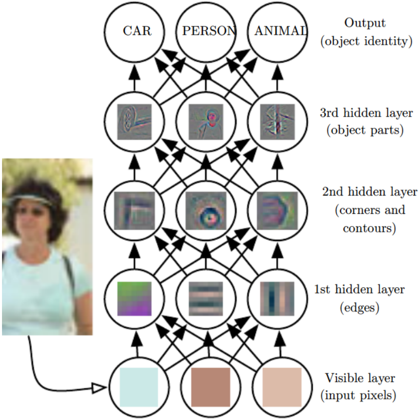
- Deep filter has larger ***receptive field*** than shallow filter
  - Receptive field of a neuron is the pixels that can activate the neuron
- I.e., deep filter sees (indirectly) more pixels



Learning Deep Local Patterns (3/3)

- A deep filter
  - sees through patterns in the same region
  - has enlarged receptive field

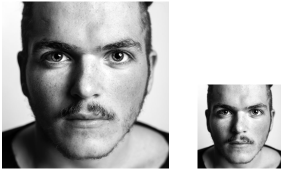
  to detect new patterns
- Together, these capabilities allow more complex patterns such as objects to be detected

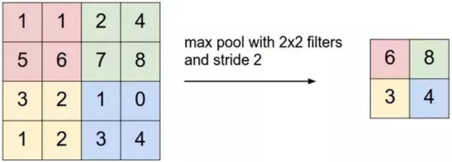


Outline

Downsampling Feature Maps

- Idea: to add a ***max pooling*** layer after each feature map
  - $\tilde{a}_{i,j} = \max\{$activation values scanned by a filter each time$\}$
  - Usually with a large ***stride*** (i.e., amount of filter shift during scanning)
- A max pooling layer downsamples a feature map ***without significantly changing "how it looks"***
  - Reduces #neurons (input of the next layer) and speeds up scanning

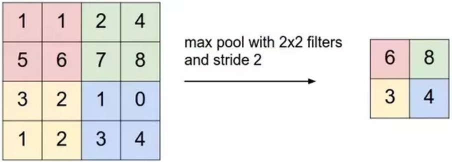




Downsampling Feature Maps

- Idea: to add a ***max pooling*** layer after each feature map
  - $\tilde{a}_{i,j} = \max\{$activation values scanned by a filter each time$\}$
  - Usually with a large ***stride*** (i.e., amount of filter shift during scanning)
- A max pooling layer downsamples a feature map ***without significantly changing "how it looks"***
  - Reduces #neurons (input of the next layer) and speeds up scanning
- Max pooling vs. average pooling?
  - Max: better for detecting edges or textures
  - Average: better for detection brightness or contrast





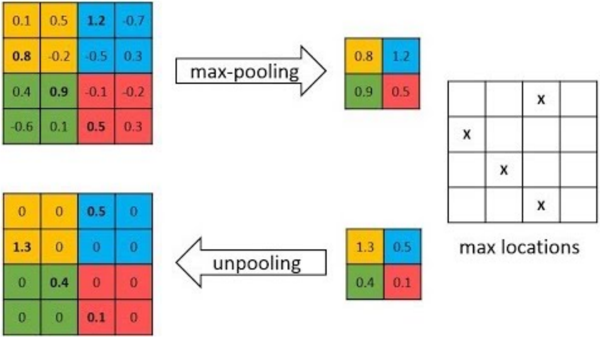
How to Train CNNs with Max Pooling Layers?

- A (max) pooling layer does not introduce new weights to learn
- However, the max function is non-differentiable
- How to backprop trough a max pooling layer?

How to Train CNNs with Max Pooling Layers?

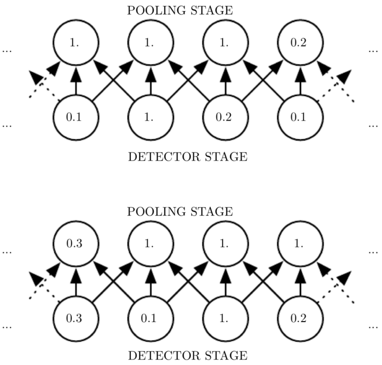
- A (max) pooling layer does not introduce new weights to learn
- However, the max function is non-differentiable
- How to backprop trough a max pooling layer?

1. During forward pass, remember the index of max
2. Then backprop only through the index



Side Effect: Translation Invariance

- Max pooling makes feature map ***invariant*** to input translation
  - Every value in the bottom row has changed
  - But only half of the values in the top row have changed

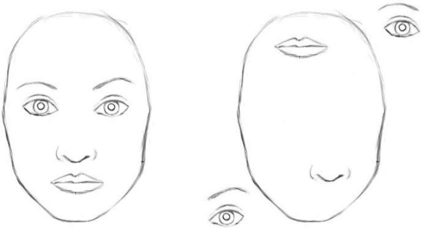


Equivalence vs. Invariance (1/2)

- Raw feature map: ***equivariant*** to input translations
- Max pooled feature map: ***invariant*** to input translations
- How does it impact the learning?

Equivariance vs. Invariance (1/2)

- Raw feature map: ***equivariant*** to input translations
- Max pooled feature map: ***invariant*** to input translations
- How does it impact the learning?
- Given a face recognition task:
    - Humans: left ✓, right X
    - CNNs with max pooling layers: left ✓, right ✓



Equivariance vs. Invariance (2/2)

- Use max pooling only when you care more about ***whether some feature is present*** than exactly where it is
  - Enough for CNNs to perform many image tasks well

Equivariance vs. Invariance (2/2)

- Use max pooling only when you care more about ***whether some feature is present*** than exactly where it is
  - Enough for CNNs to perform many image tasks well
- However, there are also many opposite cases:
  - Image segmentation: weighted average pooling in Mask R-CNNs [6]
  - Predicting class labels of objects from unseen angles: Capsule Net [19, 8]

Exercise: #weights and #neurons at each layer?

Softmax output: 10 classes

$\uparrow$

Fully connected

$\uparrow$

Flatten

$\uparrow$

Max pooling: $K = 4$, stride$= 4$

$\uparrow$

Convolution: 32 filters, $K = 4$, zero padding

$\uparrow$

Max pooling: $K = 4$, stride$= 4$

$\uparrow$

Convolution: 16 filters, $K = 4$, zero padding

$\uparrow$

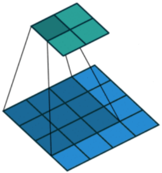
Input image: $256 \times 256 \times 3$

Exercise: #weights and #neurons at each layer?

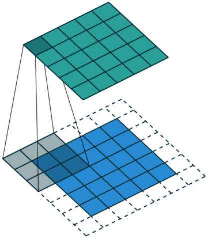
Softmax output: 10 classes	#units: 10
$\uparrow$	#weights: 0
Fully connected	#units: 10
$\uparrow$	#weights: 8192×10
Flatten	#units: 8192
$\uparrow$	#weights: 0
Max pooling: $K = 4$, stride$= 4$	#units: $16 \times 16 \times 32$
$\uparrow$	#weights: 0
Convolution: 32 filters, $K = 4$, zero padding	#units: $64 \times 64 \times 32$
$\uparrow$	#weights: $4 \times 4 \times 16 \times 32$
Max pooling: $K = 4$, stride$= 4$	#units: $64 \times 64 \times 16$
$\uparrow$	#weights: 0
Convolution: 16 filters, $K = 4$, zero padding	#units: $256 \times 256 \times 16$
$\uparrow$	#weights: $4 \times 4 \times 3 \times 16$
Input image: $256 \times 256 \times 3$	

Outline

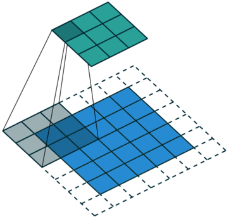
Variants of Convolution



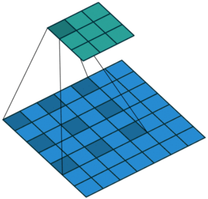
No padding, stride=1.



Zero padding, stride=1.

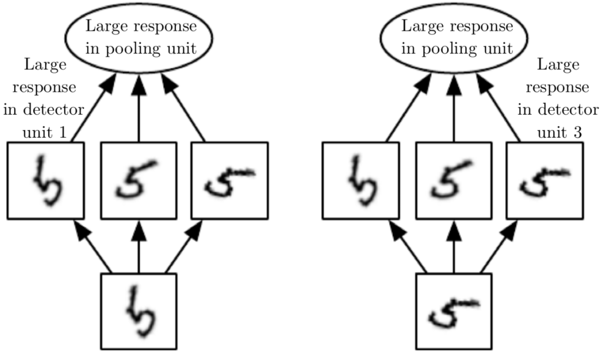


Zero padding, stride=2.



No padding, stride=1, dilation=2.

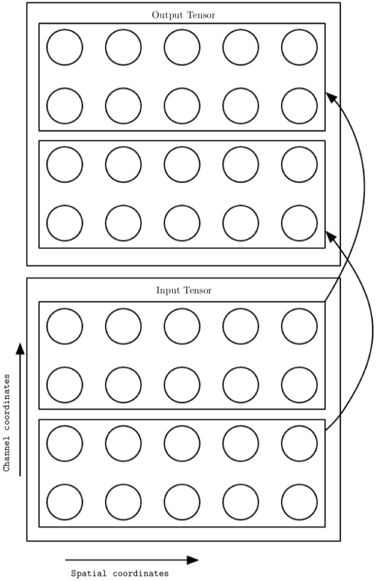
Cross-Filter Feature Pooling



- We may apply max pooling to features of ***different filters***
- Creates invariance to other transformations of the input
    - E.g., rotations

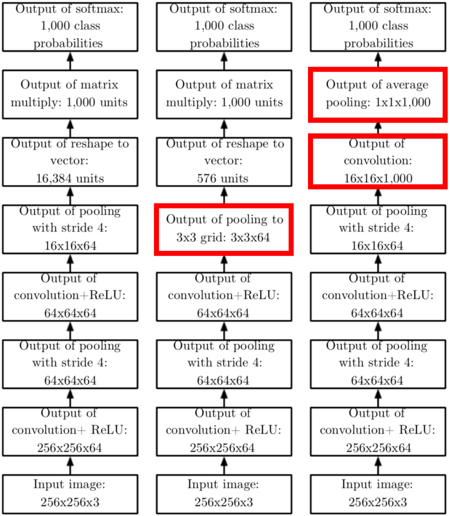
Channel Grouping

- An output channel may only connect to few input channels
- Limits #sub-patterns in a pattern (a prior)
  - May improve learning efficiency
- Further reduces #weights and computation



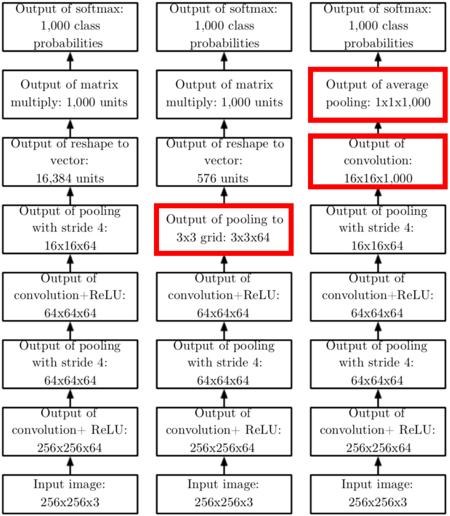
Input and Output

- Left: for images with fixed size



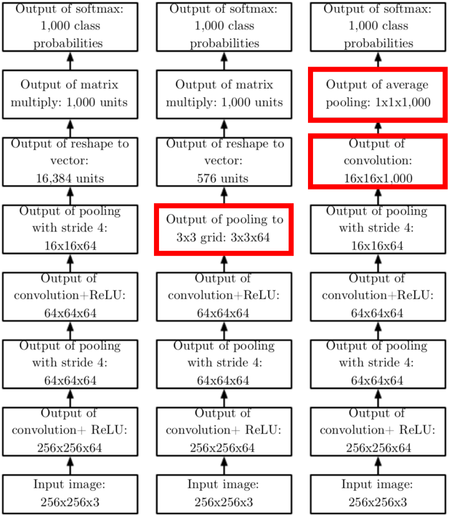
Input and Output

- Left: for images with fixed size
- Center: for images with variable size

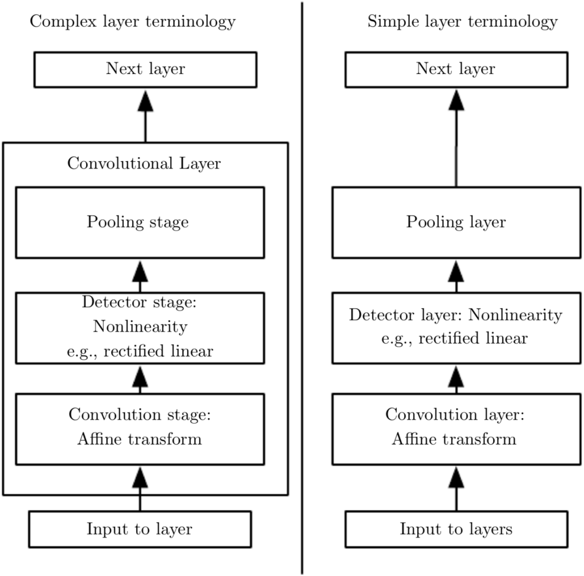


Input and Output

- Left: for images with fixed size
- Center: for images with variable size
- Right: classes as local patterns
  - Called ***fully convolutional networks*** (FCNs)

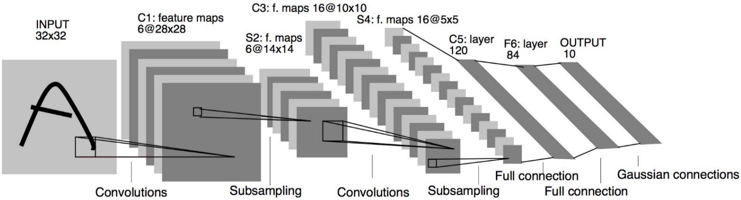


Layer Terminology



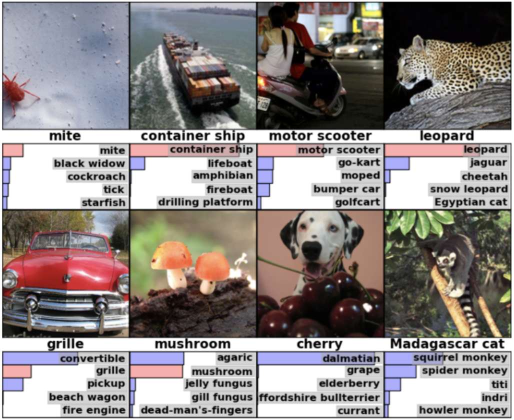
Case Study: LeNet (1998)

- Developed by Yann LeCun [12]
- Lays fundamentals of modern CNNs
  - Convolution → Pooling → Convolution → Pooling → ...



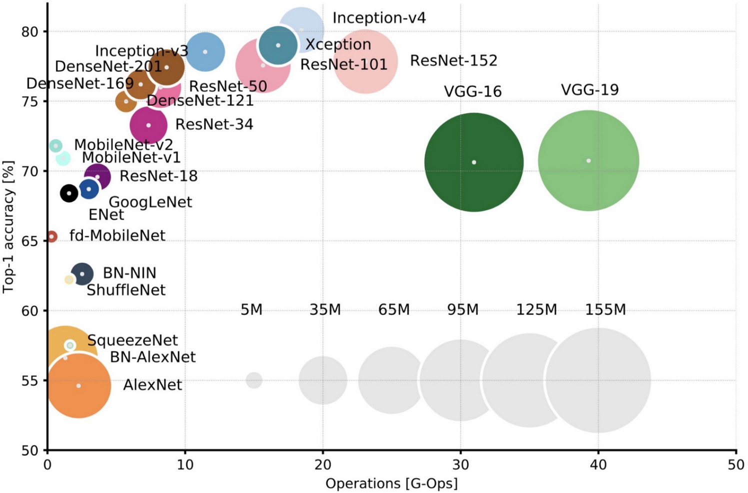
Arrival of Big Visual Data (2009)

- ImageNet [1] is an image database organized according to the WordNet [15] nouns hierarchy
  - Over five hundred images per word
  - Each image is labeled multiple words



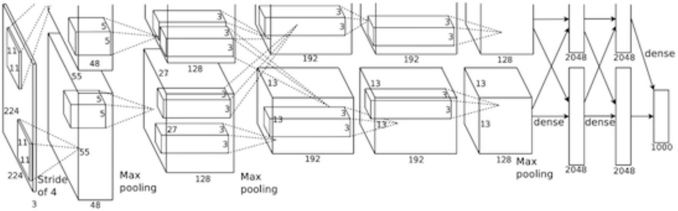
ImageNet Large Scale Visual Recognition Competition (ILSVRC)

- Drives AlexNet, VGG, GoogleLeNet, ResNet, DenseNet...



Case Study: AlexNet (2012) [11]

- Scales up LeNet
- Use of rectified linear units (ReLU)
- Use of dropout technique to avoid overfitting
- Uses GPU to get 10x faster training time

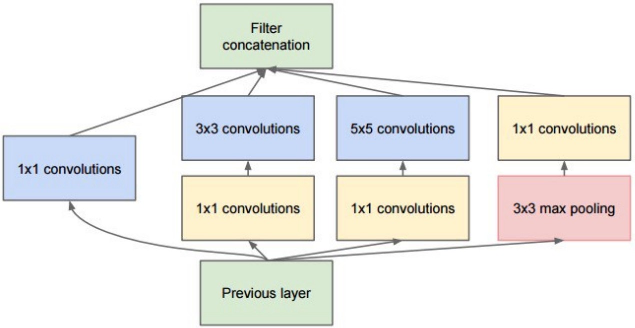


Case Study: VGG (2014) [21]

- Replace a filter with a large K (9 or 11) with ***a sequence of filters with smaller K*** (3)
- Take advantage of enlarged receptive fields at deep layers

Case Study: GoogleLeNet (2015) [22]

- Proposes ***inception module*** consisting of filters of different K's
  - Learns patterns from sub-patterns of different sizes
- Use 1×1 convolution as ***bottleneck layer*** to save computing



Bottleneck layer

- Conventional:
  - Input $(W \times H \times 256) * 256$ filters $(3 \times 3 \times 256 \times 256)$
    $\rightarrow$ Feature maps $(W \times H \times 256)$
  - #multiplies: $(W \times H) \times (3 \times 3 \times 256) \times 256 \approx (W \times H) \times 590K$

Bottleneck layer

- Conventional:
  - Input $(W \times H \times 256) * 256$ filters $(3 \times 3 \times 256 \times 256)$
    $\rightarrow$ Feature maps $(W \times H \times 256)$
  - #multiplies: $(W \times H) \times (3 \times 3 \times 256) \times 256 \approx (W \times H) \times 590K$
- Bottleneck layer:
  1. Input $(W \times H \times 256) * 64$ filters $(1 \times 1 \times 256 \times 64)$
     $\rightarrow$ Feature maps $(W \times H \times 64)$
  2. Input $(W \times H \times 64) * 64$ filters $(3 \times 3 \times 64 \times 64)$
     $\rightarrow$ Feature maps $(W \times H \times 64)$
  3. Input $(W \times H \times 64) * 256$ filters $(1 \times 1 \times 64 \times 256)$
     $\rightarrow$ Feature maps $(W \times H \times 256)$

  - #multiplies:
    $(W \times H) \times [(1 \times 1 \times 256) \times 64 + (3 \times 3 \times 64) \times 64 + (1 \times 1 \times 64) \times 256] \approx (W \times H) \times 70K$

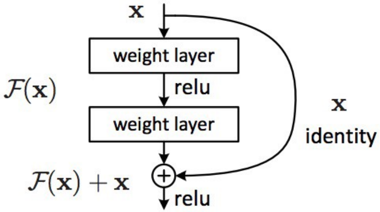
Bottleneck layer

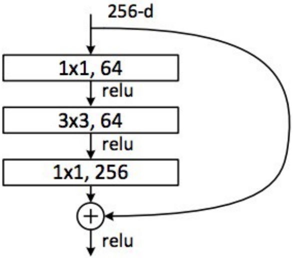
- Conventional:
  - Input $(W \times H \times 256) * 256$ filters $(3 \times 3 \times 256 \times 256)$
    $\rightarrow$ Feature maps $(W \times H \times 256)$
  - #multiplies: $(W \times H) \times (3 \times 3 \times 256) \times 256 \approx (W \times H) \times 590K$
- Bottleneck layer:
  1. Input $(W \times H \times 256) * 64$ filters $(1 \times 1 \times 256 \times 64)$
     $\rightarrow$ Feature maps $(W \times H \times 64)$
  2. Input $(W \times H \times 64) * 64$ filters $(3 \times 3 \times 64 \times 64)$
     $\rightarrow$ Feature maps $(W \times H \times 64)$
  3. Input $(W \times H \times 64) * 256$ filters $(1 \times 1 \times 64 \times 256)$
     $\rightarrow$ Feature maps $(W \times H \times 256)$

  - #multiplies:
    $(W \times H) \times [(1 \times 1 \times 256) \times 64 + (3 \times 3 \times 64) \times 64 + (1 \times 1 \times 64) \times 256] \approx (W \times H) \times 70K$
- At little cost of performance drop!
  - It helps by ***combining the features at the same position*** before learning new patterns from their relative positions

Case Study: ResNet (2016) [7]

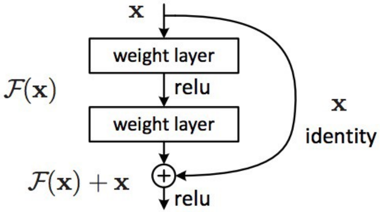
- Use of batch normalization
- An FCN: omits fully-connected layers at the end
- Very deep—152 layers
- Shortcut from current to the next-2 layer (or next bottleneck layer)

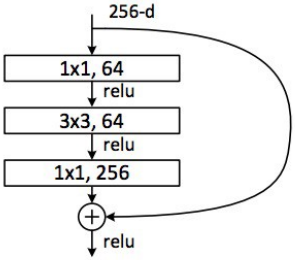




Case Study: ResNet (2016) [7]

- Use of batch normalization
- An FCN: omits fully-connected layers at the end
- Very deep—152 layers
- Shortcut from current to the next-2 layer (or next bottleneck layer)
- Idea: let a deeper CNN perform as least as well as a shallower one
  - Can "skip" layers if they are not helpful





Unraveled ResNet

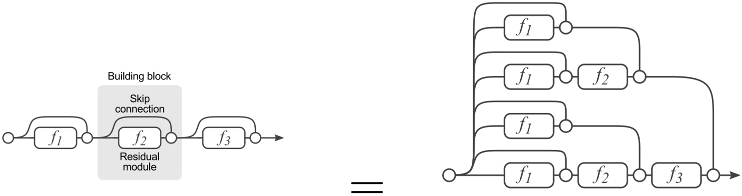
- Why to the next-***2*** layer?

Unraveled ResNet

- Why to the next-***2*** layer?
  - Empirically, does not improve performance in case of 1
  - One layer is not enough to learn "residual" patterns

Unraveled ResNet

- Why to the next-*2* layer?
  - Empirically, does not improve performance in case of 1
  - One layer is not enough to learn "residual" patterns
- ResNet can also be seen as an ensemble of small networks [25]
- ResNet usually operates on blocks of relatively low depth (20–30 layers)
  - Blocks act in parallel, rather than serially

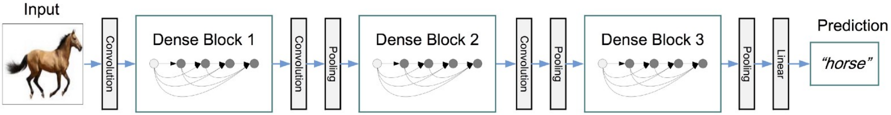


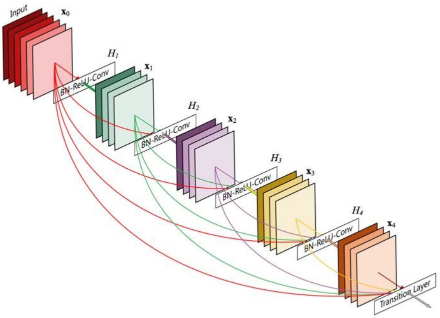
(a) Conventional 3-block residual network

(b) Unraveled view of (a)

Case Study: DenseNet (2017) [9]

- Idea: let a filter in a block see through features in ***all*** previous blocks





Outline

1. Design
   - Convolution Layers
   - Pooling Layers
   - Variants & Case Studies
2. **Visualization**
   - Visualizing Activations
   - Visualizing Filters/Kernels
   - Visualizing Gradients
   - Dreaming and Style Transfer
3. Beyond Image Classification
   - Segmentation and Localization
   - Object Detection
   - More Applications

What does a CNN lean?

- Common criticism: the decisions of NNs are not interpretable

What does a CNN lean?

- Common criticism: the decisions of NNs are not interpretable
- Modeling image priors also helps people understand CNNs!
- Two common approaches:
  - Given f and input $\boldsymbol{x}$, ***find out parts of*** $\boldsymbol{x}$
  - Given f, ***synthesis input*** $\boldsymbol{x}$

  that mostly activate $\hat{y}_j = f_j^{(L)}(\cdots f^{(1)}(\boldsymbol{x}))$ or $a_{i,j,c}^{(l)} = f_{i,j,c}^{(l)}(\cdots f^{(1)}(\boldsymbol{x}))$

Outline

Retrieving Raw Images

- On simple way to understand how $\hat{y}_j$ or $a_{i,j,c}$ is made is to retrieve (from external database) the images that mostly activate it

Retrieving Raw Images

- On simple way to understand how $\hat{y}_j$ or $a_{i,j,c}$ is made is to retrieve (from external database) the images that mostly activate it
- Images can be cropped by the receptive field of $a_{i,j,c}$
- E.g., maximally activating patches for neurons in AlexNet

Conditioned Retrieval

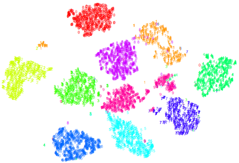
- Given an input $\boldsymbol{x}$, we can find images most similar to $\boldsymbol{x}$ ***in feature space***

Conditioned Retrieval

- Given an input $\boldsymbol{x}$, we can find images most similar to $\boldsymbol{x}$ ***in feature space***
- E.g., NNs at the deepest fully-connected layer of AlexNet
  - Semantically consistent; despite pixel-level diversity

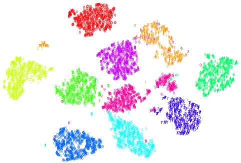
Clustering Raw Images

- Cluster images (e.g., using t-SNE [14]) based on their similarity ***in feature space***



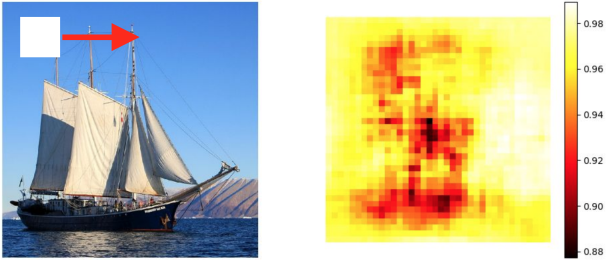
Clustering Raw Images

- Cluster images (e.g., using t-SNE [14]) based on their similarity ***in feature space***
- E.g., 2D t-SNE space reduced from the activation space of the deepest fully-connected layer of AlexNet



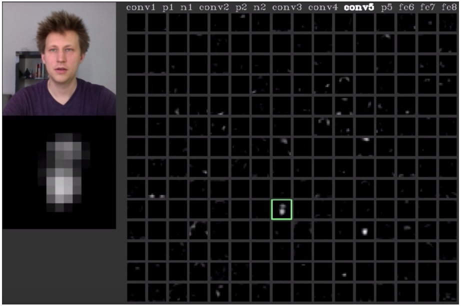
Occluding Parts of an Image

- Mask part of a given image $\boldsymbol{x}$ before feeding to f
  - Occlusion area corresponds to the receptive field of a neuron
- Draw heatmap of neuron activation at each mask location



Visualizing Activation Maps as Images

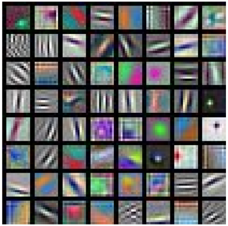
- Treat each feature map as a grayscale image
  - Smaller at deeper layers
- E.g., feature maps of AlexNet at 1st and 5th layers
  - A feature map at layer 1 detects verticals
  - Another at layer 5 detects faces



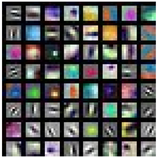
Outline

Visualizing Filters

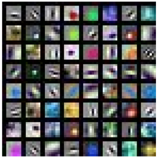
- The filters ($K \times K \times 3$) at the first layer can be viewed as a color image
  - Help us understand what pixels are detected
  - ***Not*** specific to an image
- E.g., 64 first-layer filters in different CNNs



AlexNet:
64 x 3 x 11 x 11



ResNet-18:
64 x 3 x 7 x 7

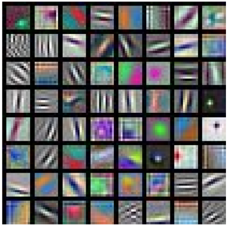


ResNet-101:
64 x 3 x 7 x 7

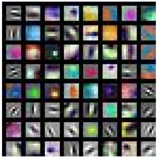
DenseNet-121:
64 x 3 x 7 x 7

Visualizing Filters

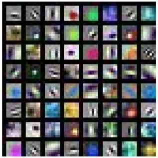
- The filters ($K \times K \times 3$) at the first layer can be viewed as a color image
  - Help us understand what pixels are detected
  - ***Not*** specific to an image
- E.g., 64 first-layer filters in different CNNs
- However, this method cannot be applied to filters at deep layers
  - Filters scan sub-patterns, not pixels
  - We need a way to recursively map patterns to sub-patterns... to pixels



AlexNet:
64 x 3 x 11 x 11



ResNet-18:
64 x 3 x 7 x 7

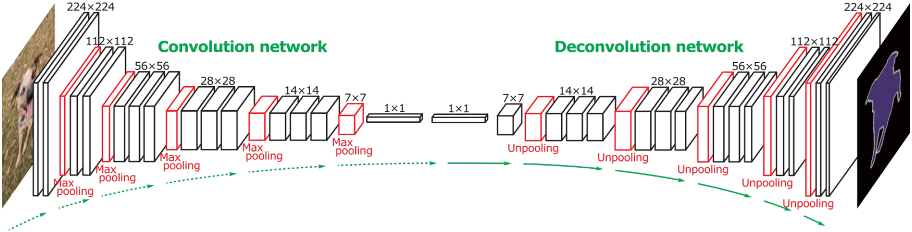


ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7

Deconvolution Network (DeconvNet) [28, 27]

- Given an activation value at deep layer
- Goal: to "undo" the effect of convolution, ReLU, and max pooling to synthesis image



Undoing Pooling

Nearest Neighbor

1	2
3	4

Input: 2 x 2

→

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Output: 4 x 4

"Bed of Nails"

1	2
3	4

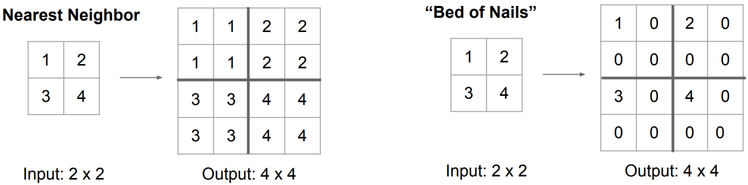
Input: 2 x 2

→

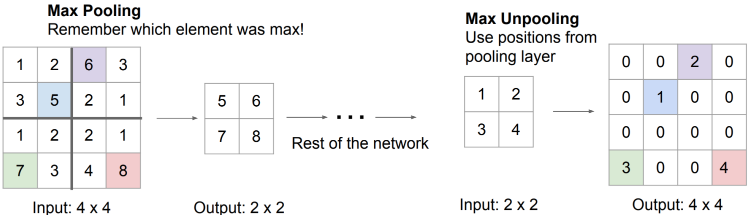
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output: 4 x 4

Undoing Pooling



- If x is available: remember which element was max during the forward pass (called ***max unpooling***)

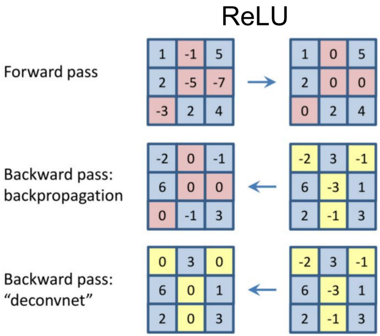


Undoing ReLU

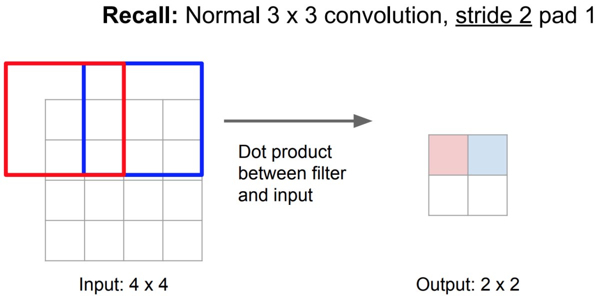
- If $\boldsymbol{x}$ is available: rectify using the binary mask remembered during the feed-forward ReLU operation
- DeconvNet [27]: simply use ordinary ReLU

Undoing ReLU

- If $\boldsymbol{x}$ is available: rectify using the binary mask remembered during the feed-forward ReLU operation
- DeconvNet [27]: simply use ordinary ReLU
    - Feature maps (and final pixels) are always positive
    - Gives more clear results empirically



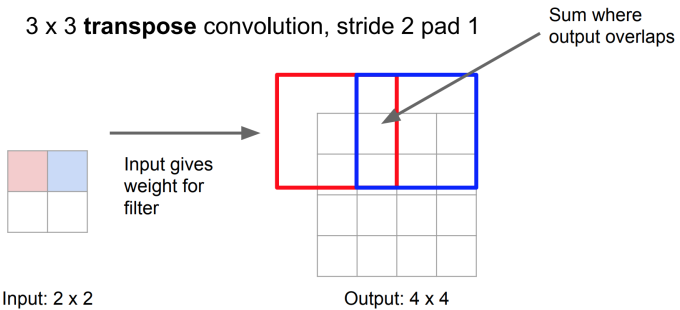
Undoing Convolution (1/2)



- Denoting a convolution op by $*$, we have ($K=2$):

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} x & y \\ z & w \end{bmatrix} = ax+by+cz+dw$$

Undoing Convolution (2/2)

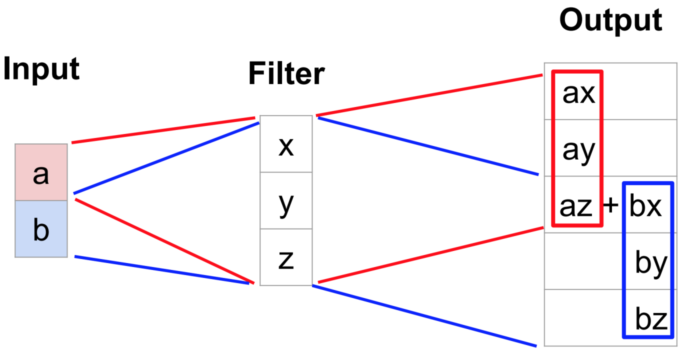


- Denoting a ***transposed convolution*** op by $*^\top$, we have $(K = 2)$:

$$a' *^\top \begin{bmatrix} w & z \\ y & x \end{bmatrix} = \begin{bmatrix} a'w & a'z \\ a'y & a'x \end{bmatrix}$$

Why Called Transposed Convolution? (1/2)

- Example: 1D convolution with $K = 3$, stride=1, and padding=1



Why Called Transposed Convolution? (2/2)

- 1D convolution ($K=3$, ***stride=1***, padding=1):

$$\boldsymbol{a} * \boldsymbol{w} = \boldsymbol{W}\boldsymbol{a} = \begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay+bz \\ ax+by+cz \\ bx+cy+dz \\ cx+dy \end{bmatrix}$$

- 1D transposed convolution ($K=3$, stride=1, padding=0):
  - $\boldsymbol{W}^\top$ denotes a regular convolution ***with reversed filter weights***

$$\boldsymbol{a}' *^\top \boldsymbol{w} = \boldsymbol{W}^\top \boldsymbol{a} = \begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix} = \begin{bmatrix} a'x \\ a'y+b'x \\ a'z+b'y+c'x \\ b'z+c'y+d'x \\ c'z+d'y \\ d'z \end{bmatrix}$$

General Cases

- 1D convolution ($K = 3$, stride=2, padding=1):

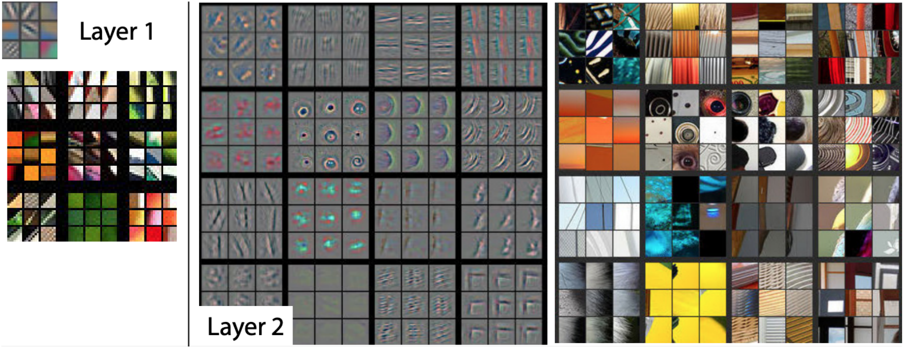
$$\boldsymbol{a} * \boldsymbol{w} = \boldsymbol{W}\boldsymbol{a} = \begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

- 1D transposed convolution:
  - $\boldsymbol{W}^\top$ ***does not*** represent a convolution any more

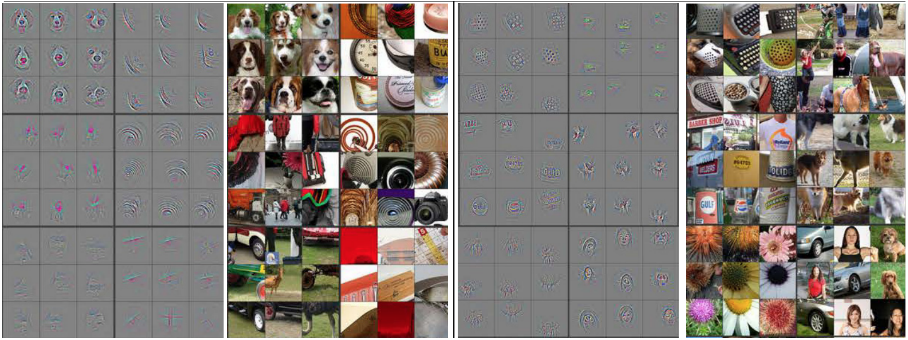
$$\boldsymbol{a} *^\top \boldsymbol{w} = \boldsymbol{W}^\top \boldsymbol{a} = \begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

DeconvNet Visualization: Layer 2

- Input image $\boldsymbol{x}$ is given
- Uses max unpooling

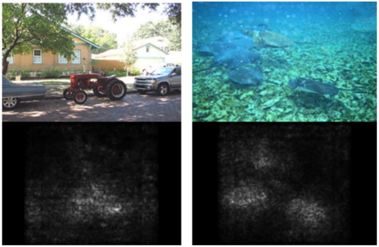


DeconvNet Visualization: Layers 4 and 5



Outline

Saliency Maps



1. Given an image $\boldsymbol{x}$, compute gradient of unnormalized class score (the logit) with respect to image pixels at $\boldsymbol{x}$:

$$\frac{\partial \hat{y}_j}{\partial \boldsymbol{x}} = \frac{\partial f_j(\boldsymbol{x};\Theta)}{\partial \boldsymbol{x}}$$

   - Network weights Θ are fixed now

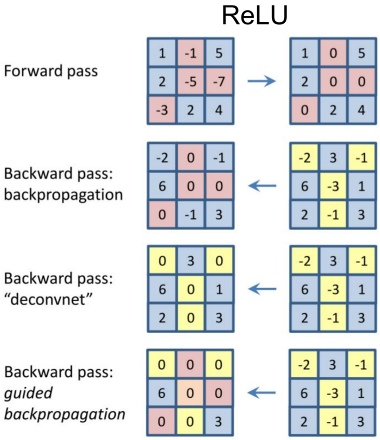
2. Take absolute value and max over RGB channels

Guided Backprop (1/2)

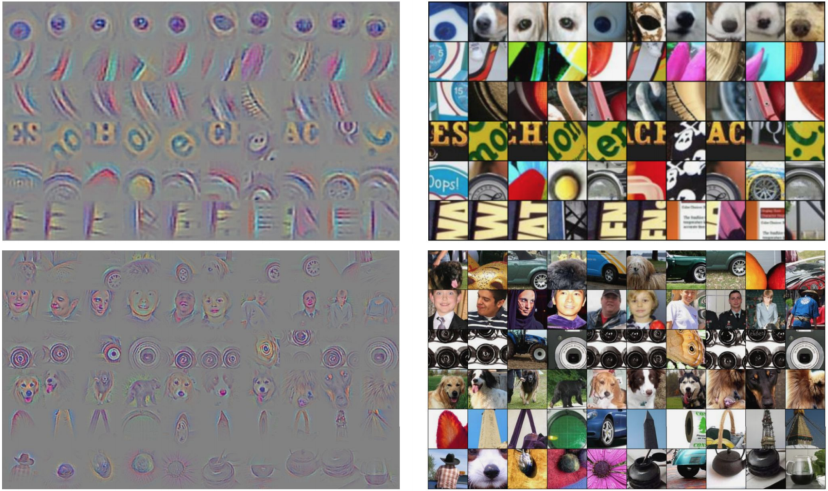
- Similarly, we can compute gradient for a neuron:

$$\frac{\partial a_{i,j,c}}{\partial \boldsymbol{x}}$$

- Trick to get clear visualization: ***guided backprop***
  - Gradient = forward term $\times$ backward term
  - Only keep the positive part of the gradient



Guided Backprop (2/2)



Gradient Ascent

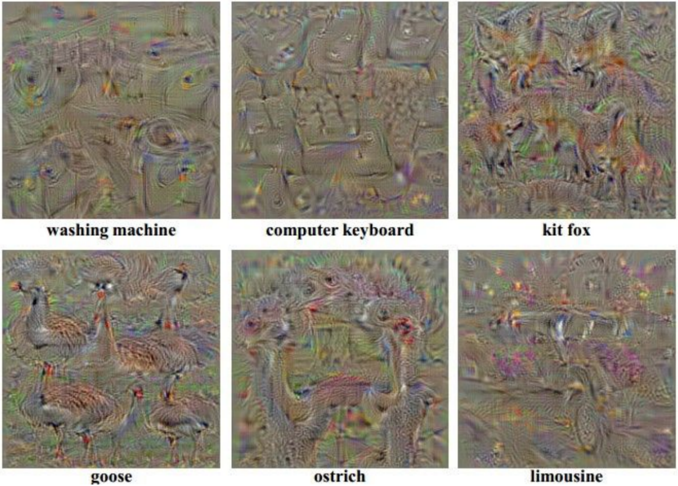
- Guided backprop requires $\boldsymbol{x}$ to be given
- ***Gradient ascent*** synthesizes $\boldsymbol{x}$ from scratch:

$$\arg\max_{\boldsymbol{x}} J(\boldsymbol{x};\Theta) = \arg\max_{\boldsymbol{x}} f(\boldsymbol{x};\Theta) - \Omega(\boldsymbol{x})$$

  - $f(\boldsymbol{x})$: a prediction score or activation value
  - $\Omega(\boldsymbol{x})$: regularizer that makes the image more natural
- Solved by gradient ascent algorithm: $\boldsymbol{x}^{(t)} \leftarrow \boldsymbol{x}^{(t-1)} + \lambda \frac{\partial J(\boldsymbol{x};\Theta)}{\partial \boldsymbol{x}}$

Natural Image Regularizer (1/2)

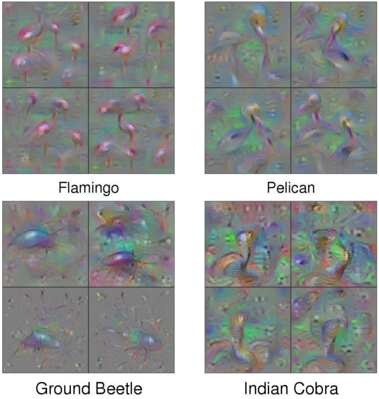
- $\Omega(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2$ [20]



washing machine / computer keyboard / kit fox / goose / ostrich / limousine

Natural Image Regularizer (2/2)

- $\Omega(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2$
- During gradient ascent optimization, periodically do followings: [26]
  - Gaussian blur image
  - Clip pixels with small values to 0
  - Clip pixels with small gradients to 0



Multi-Faceted Gradient Ascent (1/2)

- A class or a feature may be multi-faceted [16]
- Cluster images that mostly activate a neuron
  - Each cluster represents a facet
- Set the initial $\boldsymbol{x}^{(0)}$ as an image close to a clusterhead

Reconstructions of multiple feature types (facets) recognized by the same “grocery store” neuron

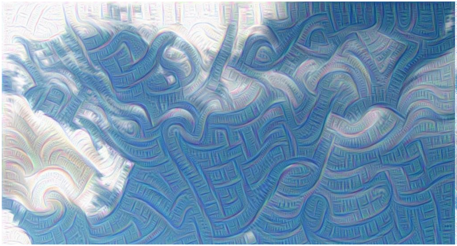
Corresponding example training set images recognized by the same neuron as in the "grocery store" class

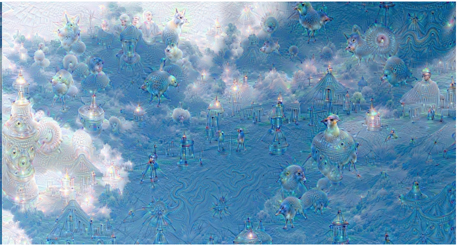
Multi-Faceted Gradient Ascent (2/2)

Outline

DeepDream

- To amplify the neuron activations at some layer
- Gradient ascent, but
  - Start at a given image
  - Maximize activations of all neurons in a layer



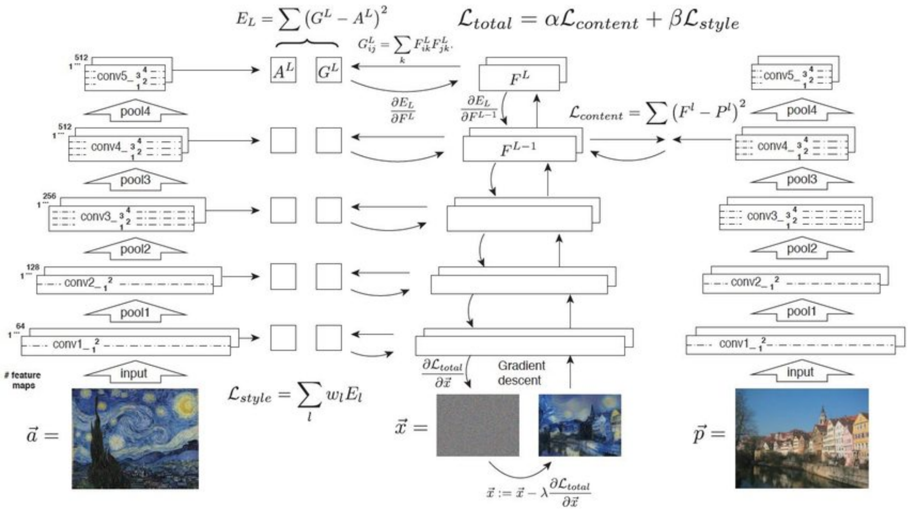


Style Transfer [3]

- Given a content image $\boldsymbol{p}$ and style image $\boldsymbol{a}$
- Synthesis an image $\boldsymbol{x}$ with content in $\boldsymbol{p}$ and style in $\boldsymbol{a}$

Network Architecture: VGG (Fixed Weights)

- Gradient descent finds $\boldsymbol{x}$ minimizing both L_{content} and L_{style}

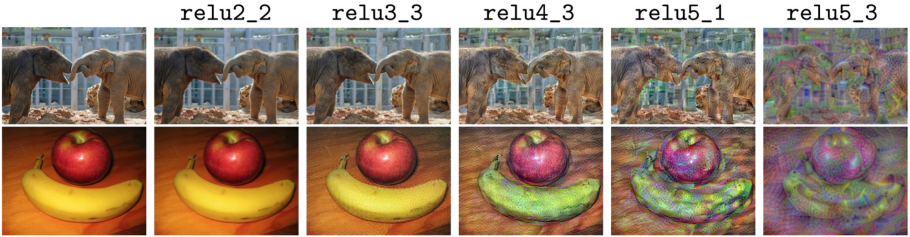


Content Loss

- The content loss:

$$L_{\text{content}} = \sum_{c} \|f^{(l)}_{\cdot,\cdot,c}(\boldsymbol{x}) - f^{(l)}_{\cdot,\cdot,c}(\boldsymbol{p})\|_F^2$$

aligns the feature maps of $\boldsymbol{x}$ and $\boldsymbol{p}$ at a particular layer l

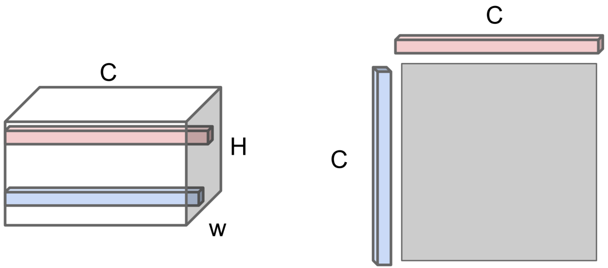


Style Loss (1/2)

- The style loss:

$$L_{\text{style}} = \sum_l \|\boldsymbol{G}_{\boldsymbol{x}}^{(l)} - \boldsymbol{G}_{\boldsymbol{p}}^{(l)}\|_F^2$$

  aligns the ***Gram matrices*** $\boldsymbol{G} \in \mathbb{R}^{C\times C}$ of $\boldsymbol{x}$ and $\boldsymbol{p}$ at all layers
- $\boldsymbol{G} = \boldsymbol{F}\boldsymbol{F}^\top$, where $\boldsymbol{F} \in \mathbb{R}^{C\times WH}$ is the reshaped feature map
  - $C_{s,t} = \sum_{i,j} a_{i,j,m} a_{i,j,n}$ captures the correlation of sub-patterns m and n at different locations

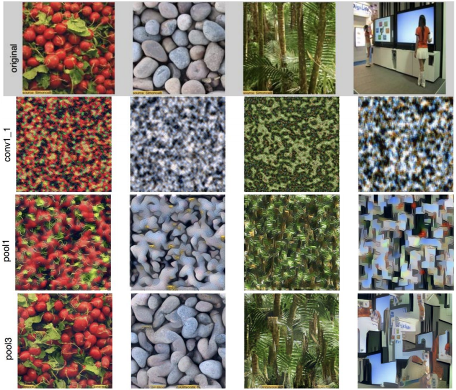


Style Loss (2/2)

- The style loss:

$$L_{\text{style}} = \sum_{l} \|\boldsymbol{G}_{\boldsymbol{x}}^{(l)} - \boldsymbol{G}_{\boldsymbol{p}}^{(l)}\|_F^2$$

- Layer-by-layer effect: deeper layer $\rightarrow$ more coarse-grained texture [2]



Fast Style Transfer

- Problem: gradient descent is run to generate an image
  - Very slow!
- Fast style transfer for, e.g., videos?

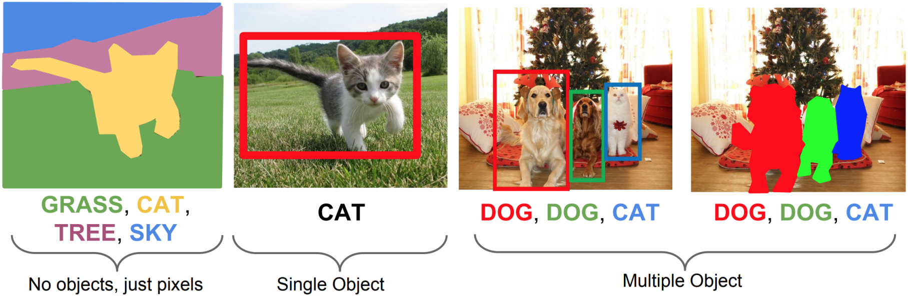
Fast Style Transfer

- Problem: gradient descent is run to generate an image
  - Very slow!
- Fast style transfer for, e.g., videos?
- Idea: train another network to perform style transfer [10, 24]

Outline

More Supervised Image Tasks

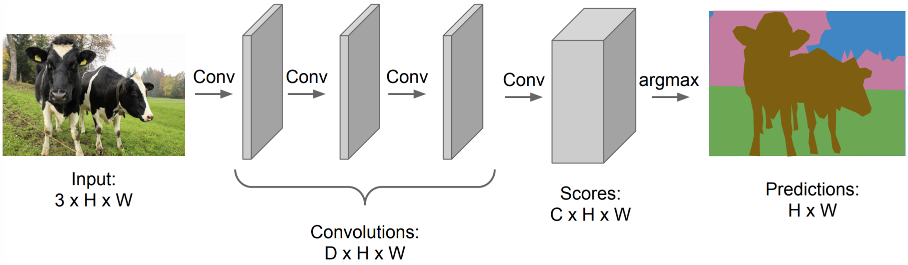
- ***Semantic segmentation***: label $\boldsymbol{y}$ contains segment ID of each pixel
- Classification + ***localization***: label $\boldsymbol{y}$ contains class ID and location info (e.g., bounding box) of an object
- ***Object detection***: label $\boldsymbol{y}$ contains class IDs and location info of multiple objects
- ***Instance segmentation***: object detection with pixel masks



Outline

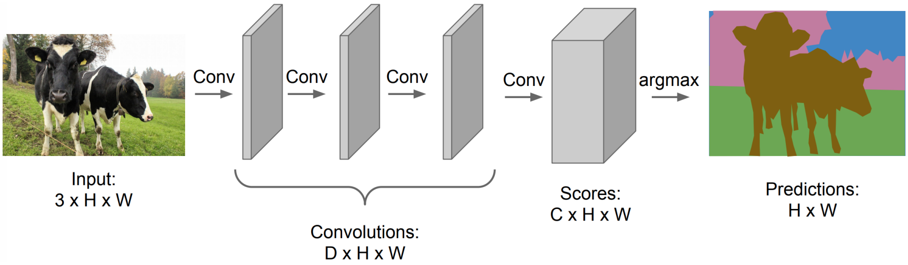
Semantic Segmentation: FCN

- Use an FCN to predict the class of each pixel



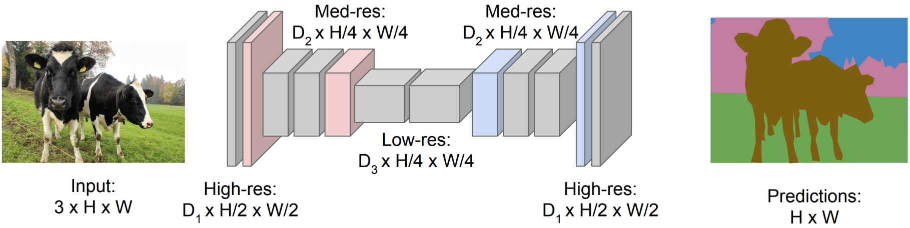
Semantic Segmentation: FCN

- Use an FCN to predict the class of each pixel
- However, convolutions at original image resolution are very expensive



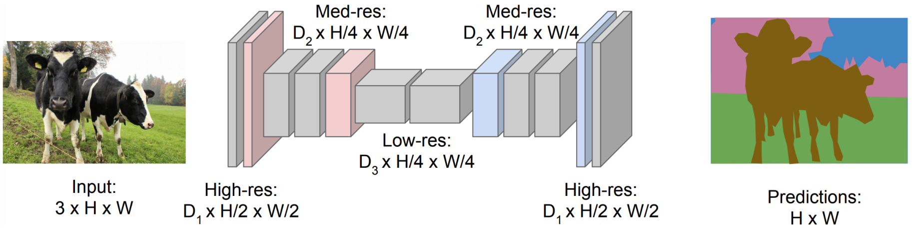
Semantic Segmentation: FCN + DeconvNet

- Use DeconvNet to do downsampling and upsampling inside the network



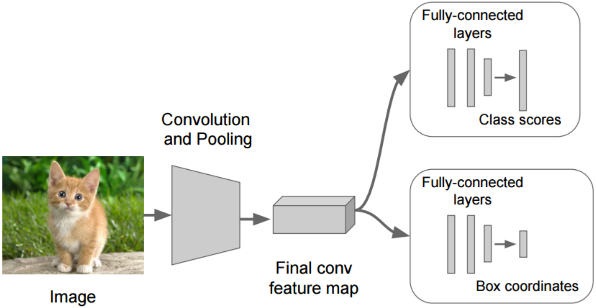
Semantic Segmentation: FCN + DeconvNet

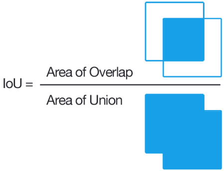
- Use DeconvNet to do downsampling and upsampling inside the network
- The weights of a transpose convolution can be ***tied*** with weights of a corresponding convolution or ***re-trained***



Classification + Localization

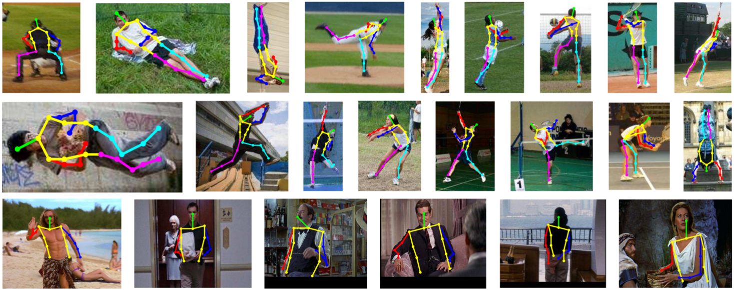
- Two losses:
  - Classification loss: $l(\hat{\boldsymbol{y}},\boldsymbol{y})$
  - Regression loss, e.g., $\|[\hat{x},\hat{y},\hat{w},\hat{h}]-[x,y,w,h]\|_2^2$ or (1 - IoU)
    - ***Intersection over Union*** (IoU) takes the size of ROI (region of interest) into account and gives an relative error





Pose Estimation

- Just like classification + localization
- Bounding boxes replaced by joint positions
  - Head, neck, shoulder, elbow, hand, hip, knee, foot, etc.
- Regression loss for each joint position



Outline

How to Detect Multiple Objects?

How to Detect Multiple Objects?

- Naive idea: let CNN detect one object in a sliding window
  - Assuming ***fixed #classes***: cat, dog, and background

How to Detect Multiple Objects?

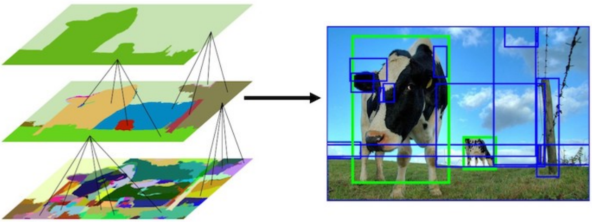
- Naive idea: let CNN detect one object in a sliding window
  - Assuming ***fixed #classes***: cat, dog, and background
- Problem: too many windows!
  - at different locations and scales

Region Proposals

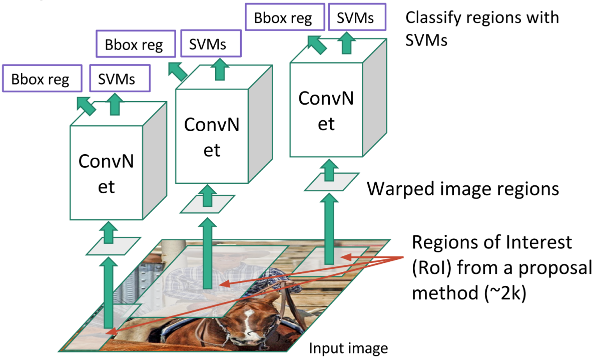
- Use a region proposal algorithm that outputs bounding boxes likely to contain objects
- E.g., selective search [23]
  - Low precision; ***high recall***

Region Proposals

- Use a region proposal algorithm that outputs bounding boxes likely to contain objects
- E.g., selective search [23]
  - Low precision; ***high recall***
- Repeat:
  - Group adjacent pixels/segments based on similarity
  - Propose a bounding box for each new segment
- Deterministic and fast: 1000+ region proposals in a few seconds on CPU

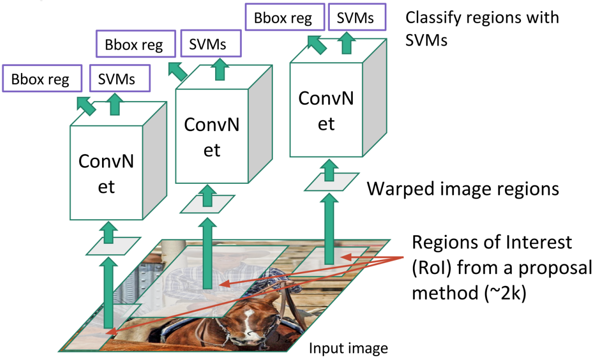


R-CNN [5]


Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Classify regions with SVMs
ConvNet
ConvNet
ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

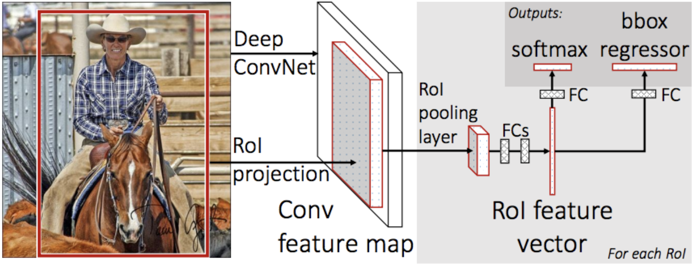
R-CNN [5]

- Multi-stage training:
  1. CNN (AlexNet) with softmax classifier (log loss)
  2. SVMs (hinge loss)
  3. Regressors (least square loss)
- Storage: 2K feature tensors
- Slow at test time
  - 2K feed-forward passes in CNN



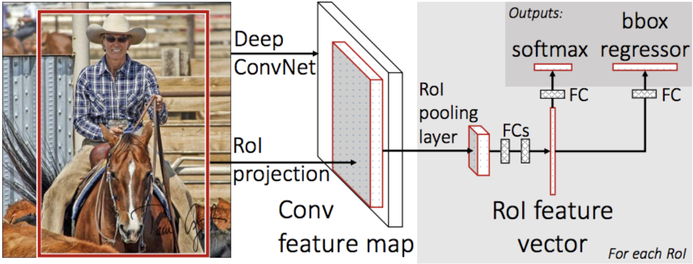
Fast R-CNN [4]

- Single network: end-to-end prediction and training
- Shared CNN computation
  - Input: 1 entire image
  - Output: 1 feature tensor (storage consumption)
- Then apply classification/regression network to each RoI ***in the feature space***
  - ***RoI pooling***: warp different projected RoIs to the same 7×7 grid



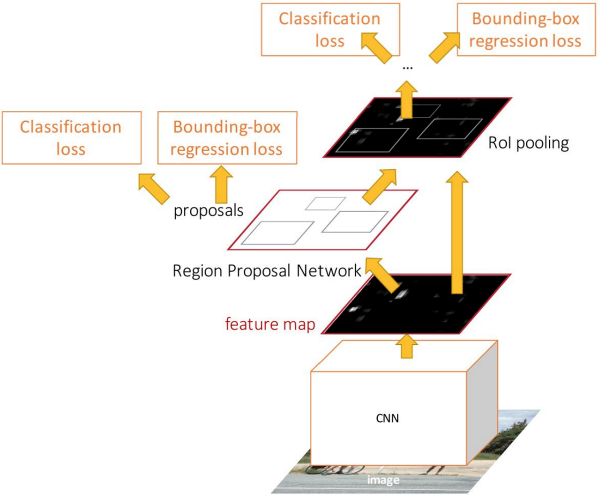
Fast R-CNN [4]

- Single network: end-to-end prediction and training
- Shared CNN computation
  - Input: 1 entire image
  - Output: 1 feature tensor (storage consumption)
- Then apply classification/regression network to each RoI ***in the feature space***
  - ***RoI pooling***: warp different projected RoIs to the same 7×7 grid
- Test time: 1 feed-forward pass in CNN for all predictions
  - So fast such that the region proposal algorithm becomes a bottleneck



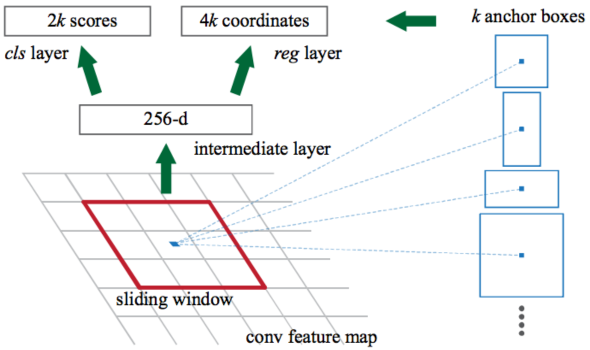
Faster R-CNN [18]

- Jointly learn a ***region proposal network*** (RPN)



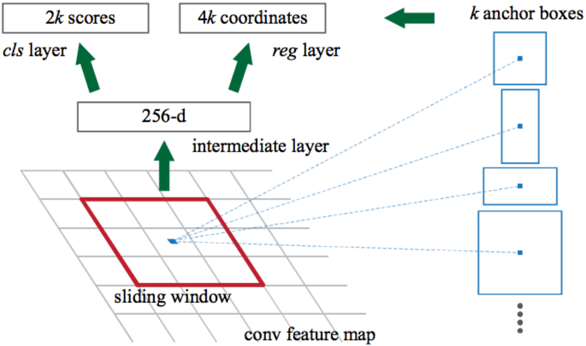
Region Proposal Network (1/2)

- Slides a window over the feature map of the CNN
- At each window location, the network outputs for each anchor box:
  - A binary score: if the anchor box contains an object or not
  - Corrections of coordinates of the anchor box
- Anchor boxes represent common aspect ratios at different scales



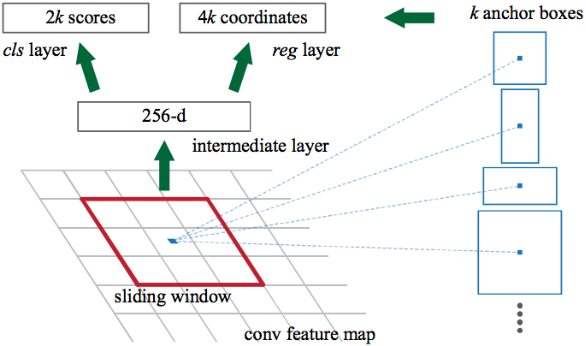
Region Proposal Network (2/2)

- Why anchor boxes?
  - Regularizing/limiting correction length allows network to learn proposals of different sizes



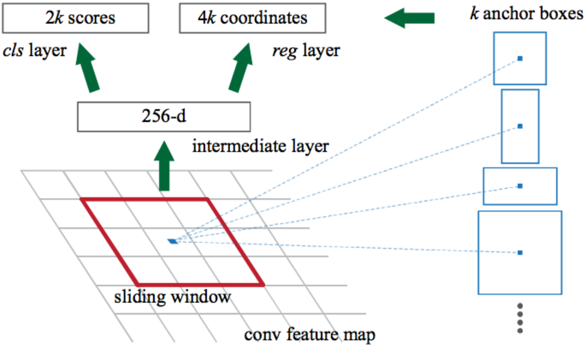
Region Proposal Network (2/2)

- Why anchor boxes?
  - Regularizing/limiting correction length allows network to learn proposals of different sizes
- How to generate training labels?



Region Proposal Network (2/2)

- Why anchor boxes?
  - Regularizing/limiting correction length allows network to learn proposals of different sizes
- How to generate training labels?
  - For each RoI in the ground truth, assign positive label (and corrections) to the anchor box with the highest IoU score at a window location



Single-Shot Detectors

- Faster R-CNN gives close to real-time test performance
  - ~ 0.2 sec per image

Single-Shot Detectors

- Faster R-CNN gives close to real-time test performance
  - ~ 0.2 sec per image
- Still not fast enough for detecting objects in videos
  - Bottleneck: repeated computation for each RoI

Single-Shot Detectors

- Faster R-CNN gives close to real-time test performance
  - ~ 0.2 sec per image
- Still not fast enough for detecting objects in videos
  - Bottleneck: repeated computation for each RoI
- Single-shot object detectors:
  - You Only Look Once (YOLO) [17]
  - Single-Shot Detector (SSD) [13]

YOLO (1/2)

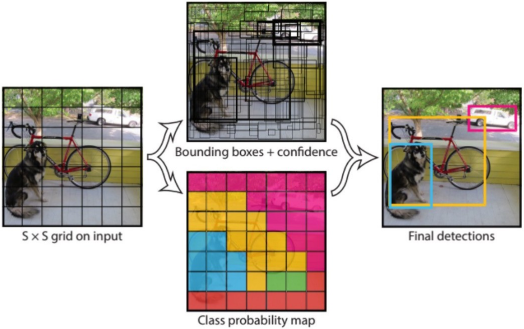
- In Faster R-CNN:
  - Region proposal network generates region proposals
  - Classification/regression network accepts/rejects proposals and makes adjustments

YOLO (1/2)

- In Faster R-CNN:
  - Region proposal network generates region proposals
  - Classification/regression network accepts/rejects proposals and makes adjustments
  - Why not use the region proposal network to directly generate final bounding boxes?

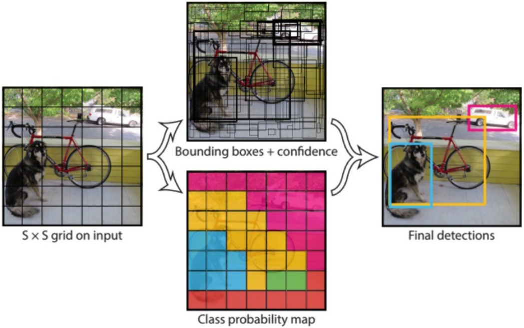
YOLO (2/2)

- YOLO [17] resembles to a region proposal network, except
  - Classifies window locations while proposing regions
  - Uses deterministic (non-parametric) algorithm to reject low-confidence boxes at test time



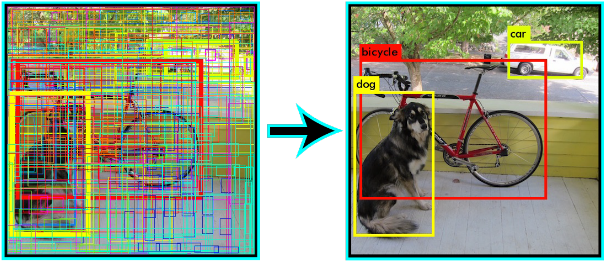
Network

- Output dimension: $S \times S \times (B \times 5 + C)$
  - S: #window locations
  - B: #anchor boxes
  - 5: corrections of box coordinates (4) + object confidence (1)
  - C: #classes (one-hot)
- End-to-end prediction and training
  - Each RoI in the ground truth is assigned to grid that contains RoI's midpoint and anchor box with highest IoU



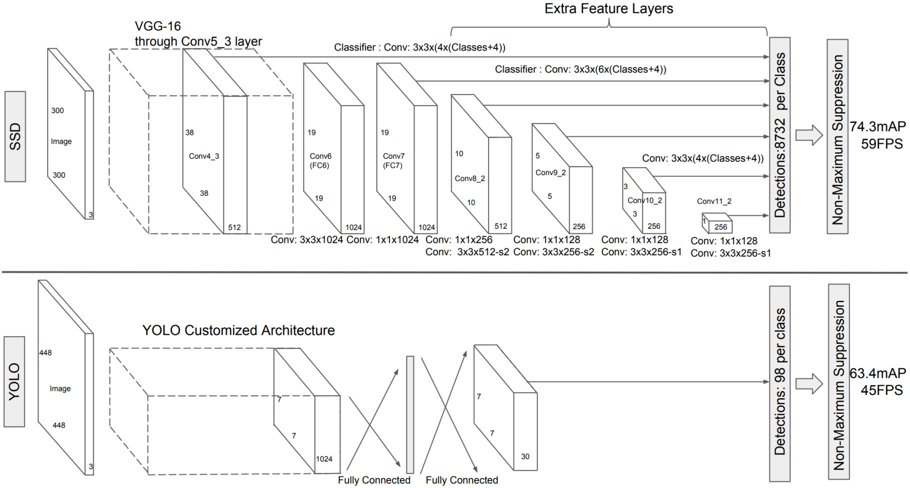
Reducing #Boxes at Test Time

1. Label each box by class score = class probability $\times$ object confidence
2. Discard boxes with low scores
3. ***Non-max suppression***: repeat until there is no box left
   - Output the box $\boldsymbol{b}$ with highest score
   - Discard any remaining box of the same class having IoU≥ 0.5 with $\boldsymbol{b}$



SSD [13]

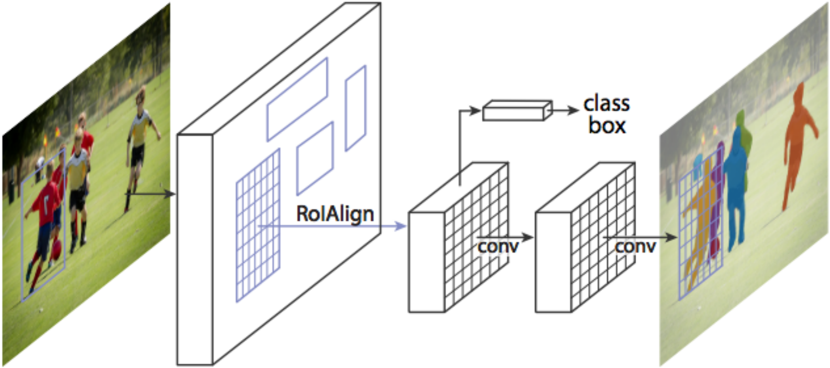
- FCN, no fully-connected layers
- Scans feature maps at multiple scales



Outline

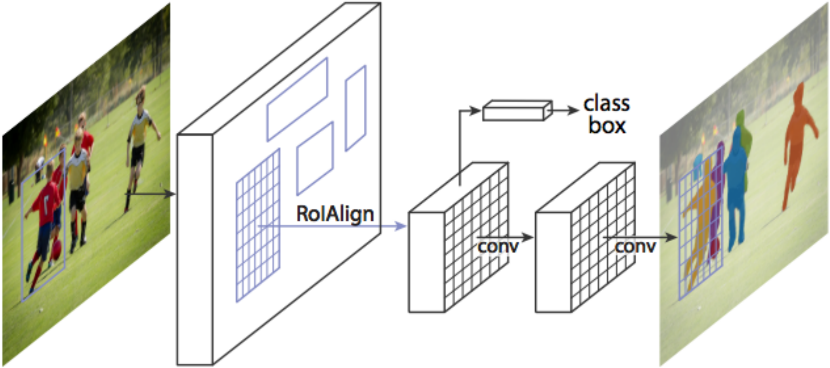
Mask R-CNN [6]

- Instance segmentation (object detection + segmentation)
- Add an FCN on top of the CNN features of Faster R-CNN
  - Outputs segmentation mask



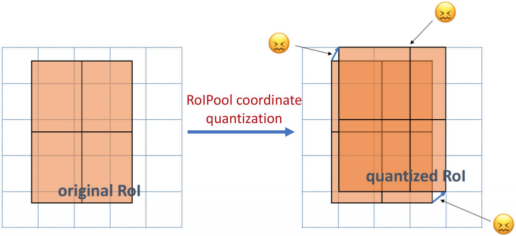
Mask R-CNN [6]

- Instance segmentation (object detection + segmentation)
- Add an FCN on top of the CNN features of Faster R-CNN
  - Outputs segmentation mask
- Replace RoI pooling with ***RoI align*** to achieve pixel-level precision in segmentation



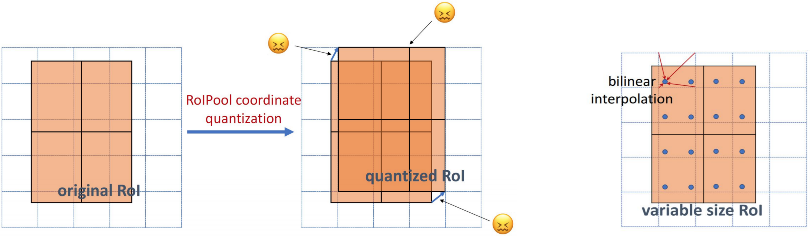
RoI Align

- RoI pooling in Fast/Faster R-CNNs creates translation invariance
  - Leads to errors in segmentation



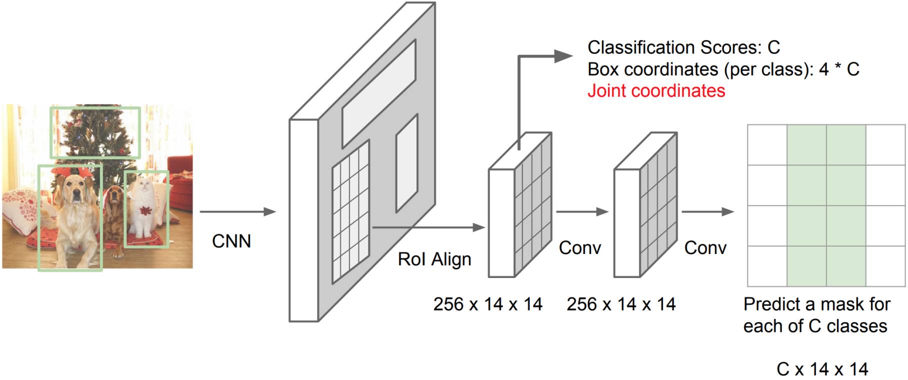
RoI Align

- RoI pooling in Fast/Faster R-CNNs creates translation invariance
  - Leads to errors in segmentation
- ***RoI align***: weighted average pooling
  - Computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map



Results

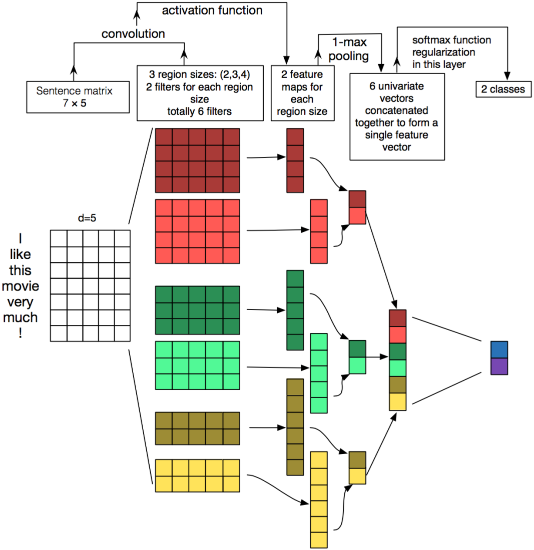
Mask R-CNN + Pose Estimation



Results

CNNs for Non-Image Tasks

- Example: sentiment analysis in NLP
  - Input: sentence/document
  - Output: positive or negative
- 1D convolution of words
  - Multiple filters ***of different sizes*** $(K = 2, \cdots, D)$ for 2-gram, ..., D-gram
- 1-max pooling



Reference I

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei.
Imagenet: A large-scale hierarchical image database.
In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[2] Leon Gatys, Alexander S Ecker, and Matthias Bethge.
Texture synthesis using convolutional neural networks.
In *Advances in Neural Information Processing Systems*, pages 262–270, 2015.

[3] Leon A Gatys, Alexander S Ecker, and Matthias Bethge.
Image style transfer using convolutional neural networks.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

Reference II

[4] Ross Girshick.
Fast r-cnn.
In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik.
Rich feature hierarchies for accurate object detection and semantic segmentation.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick.
Mask r-cnn.
In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

Reference III

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
Deep residual learning for image recognition.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst.
Matrix capsules with EM routing.
In *International Conference on Learning Representations*, 2018.

[9] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger.
Densely connected convolutional networks.
In *CVPR*, volume 1, page 3, 2017.

[10] Justin Johnson, Alexandre Alahi, and Li Fei-Fei.
Perceptual losses for real-time style transfer and super-resolution.
In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

Reference IV

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton.
Imagenet classification with deep convolutional neural networks.
In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner.
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278–2324, 1998.

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg.
Ssd: Single shot multibox detector.
In *European conference on computer vision*, pages 21–37. Springer, 2016.

[14] Laurens van der Maaten and Geoffrey Hinton.
Visualizing data using t-sne.
Journal of machine learning research, 9(Nov):2579–2605, 2008.

Reference V

[15] George A Miller.
Wordnet: a lexical database for english.
Communications of the ACM, 38(11):39–41, 1995.

[16] Anh Nguyen, Jason Yosinski, and Jeff Clune.
Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks.
arXiv preprint arXiv:1602.03616, 2016.

[17] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi.
You only look once: Unified, real-time object detection.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

Reference VI

[18] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun.
Faster r-cnn: Towards real-time object detection with region proposal networks.
In *Advances in neural information processing systems*, pages 91–99, 2015.

[19] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton.
Dynamic routing between capsules.
In *Advances in Neural Information Processing Systems*, pages 3857–3867, 2017.

[20] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman.
Deep inside convolutional networks: Visualising image classification models and saliency maps.
arXiv preprint arXiv:1312.6034, 2013.

Reference VII

[21] Karen Simonyan and Andrew Zisserman.
Very deep convolutional networks for large-scale image recognition.
arXiv preprint arXiv:1409.1556, 2014.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich.
Going deeper with convolutions.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[23] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders.
Selective search for object recognition.
International journal of computer vision, 104(2):154–171, 2013.

Reference VIII

[24] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky.
Texture networks: Feed-forward synthesis of textures and stylized images.
In *ICML*, pages 1349–1357, 2016.

[25] Andreas Veit, Michael J Wilber, and Serge Belongie.
Residual networks behave like ensembles of relatively shallow networks.
In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

[26] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson.
Understanding neural networks through deep visualization.
arXiv preprint arXiv:1506.06579, 2015.

Reference IX

[27] Matthew D Zeiler and Rob Fergus.
Visualizing and understanding convolutional networks.
In *European conference on computer vision*, pages 818–833. Springer, 2014.

[28] Matthew D Zeiler, Graham W Taylor, and Rob Fergus.
Adaptive deconvolutional networks for mid and high level feature learning.
In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2018–2025. IEEE, 2011.