Convolutional Neural Networks

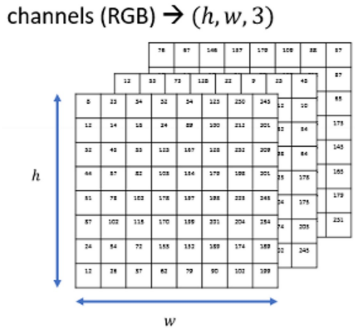
Sunday, December 12, 2021 12:37 PM

12. CNN

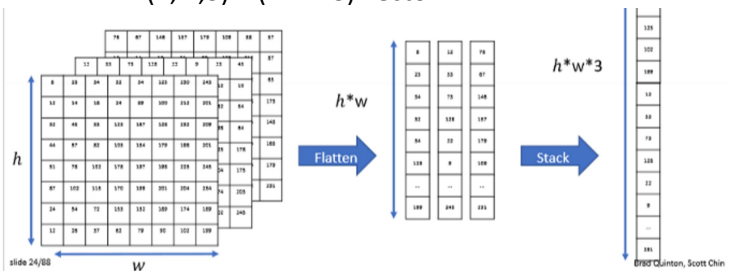
- CNN = class of NN used for image analysis and comp vision

Image Classification w/ Fully-Connected NN

- **Structured data**
  - each feature = piece of info
  - data can be tabulated
  - eg. housing data, feature = house price
- **Unstructure data**
  - no relationship b/w samples of same feature
  - eg. image data: each pixel = feature
- color image data
  - 3 channels (RGB) = (h, w, 3)



  - convert to feature vector, concatenate:
    - (h,w,3) = (h*w*3) vector



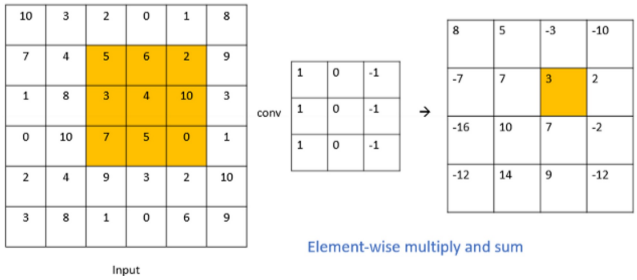
- **Problem 1: throwing away spatial information**
- eg. for 32x32 img, 32*32*3=3,072
  - if 1st layer has 1000 units, W[1] is (1000,3072) matrix
  - unit = neuron to train?
- **Problem 2: Too many params**
  - for 200x200 px, FCN (fully connected network) = too many params
  - which connections to eliminate?
- **Problem 3: Pixels that form visual feature are local**
  - every unit trying to make sense of entire img
  - spacial correlation is local
  - eg. pixel of eye has nothing to do with pixel of foreground
  - **Solution: have each unit connect only to small region**
    - eg. 11x11 px region, sparsity, fewer params per unit!, regions are contiguous, able to vectorize!
    - good for face detection
    - No tolerance to translation: If trained on roughly centered, but supplied non centered

face, wont activate

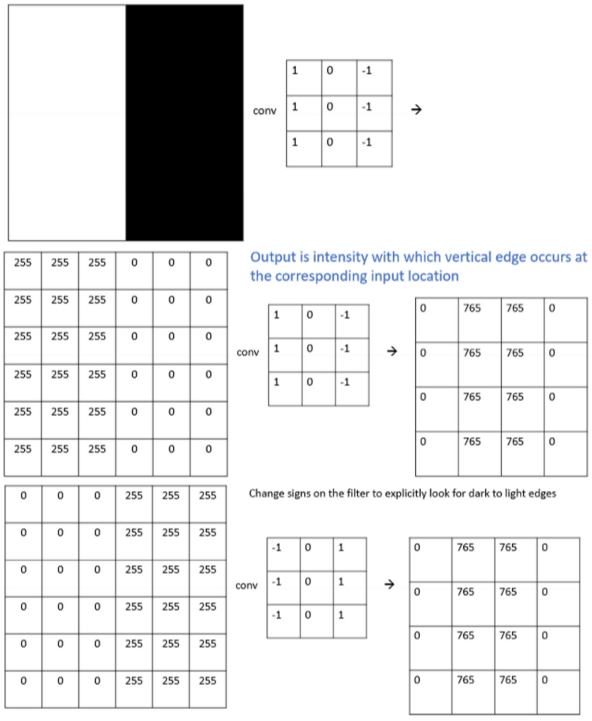
- **Problem: Redundant Neurons**
  - eg. for patches of hair, redundant neurons activating many other places of img, learning the same thing
  - **Solution: Shared parameters**
    - neurons at diff locations share params
    - one neuron scans looking for specific feature, filter (kernel)
    - use multiple filters, eahc looks for diff feature, convolutional filters

Convolutional Filters

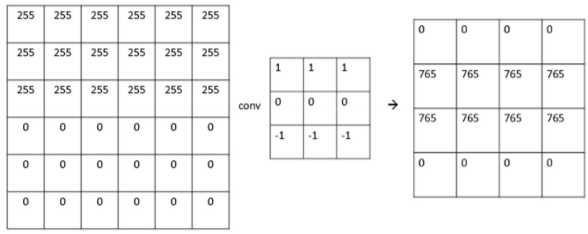
- **Convolution Operation**
  - edge detection



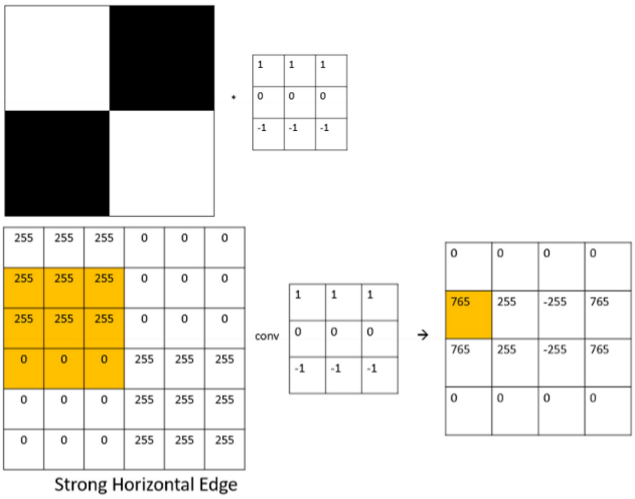
- **Vertical Edge detection**



- **Horizontal Edge detection**



- **Horizontal and Vertical Edges**



- **Choosing Filter Values**

1	0	-1
1	0	-1
1	0	-1

Vertical Edge Filter

1	0	-1
2	0	-2
1	0	-1

Sobel Filter

3	0	-3
10	0	-10
3	0	-3

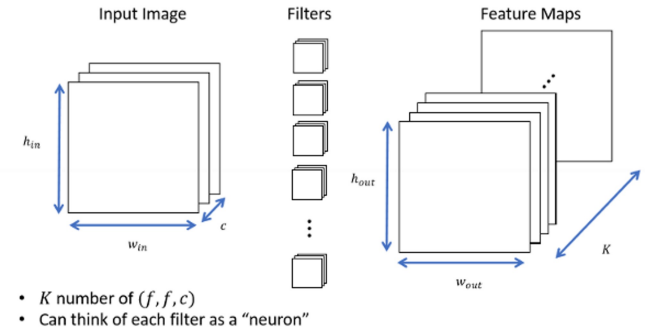
Scharr Filter

  - let deep learning model decide
  - treat filter vals as learnable params, supply data, let model learn best vals

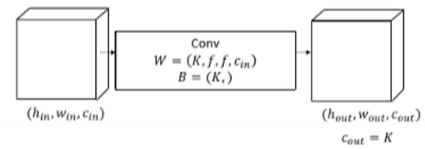
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

13. CNN

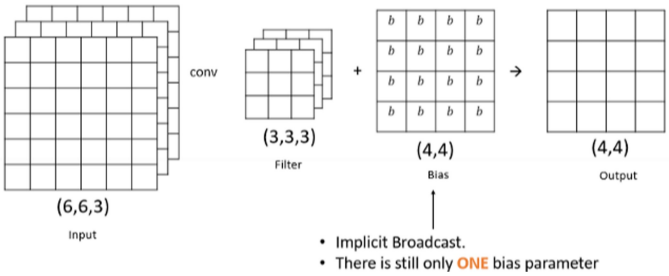
- need same amount of filters as feature Maps outputs



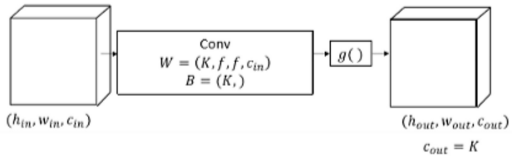
- collection of filters = single (K,f,f,c) weight tensor
- still need bias params and activations for convolutional layer



- for each filter neuron, need 1 bias/filter



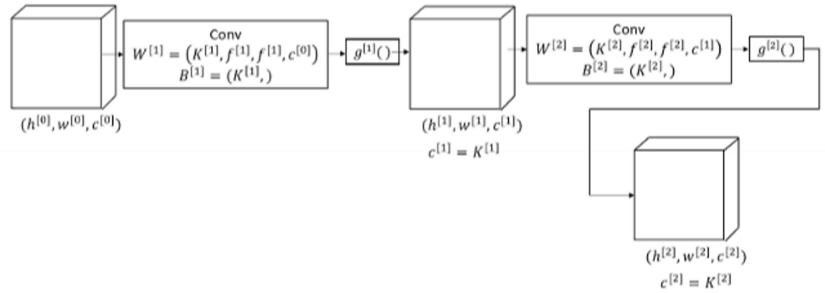
- defined "linear" part of layer (x=WX+B), need nonlinear activation



- Weight parameters: K*f*f*c_in, K = bias
- Total: K*(f*f*c_in + 1)

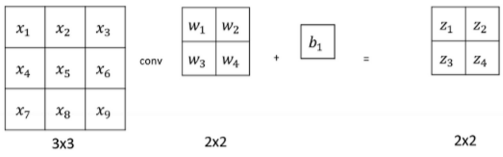
Stacking Convolution Layers

- 2 convolutional layers stacked:



Relationship Bw convolutional and FCL

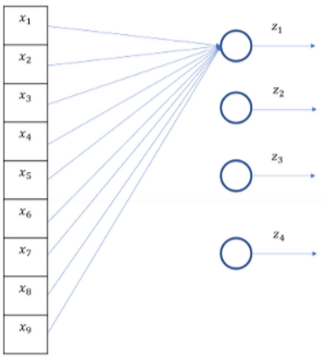
- FCN lead to a lot of params, doesn't use locality of visual features spatial correlation
- Conv Layer would have redundant neurons, solve with param sharing
- **Transform FCL to ConvLayer**
  - eg. 1 filter:



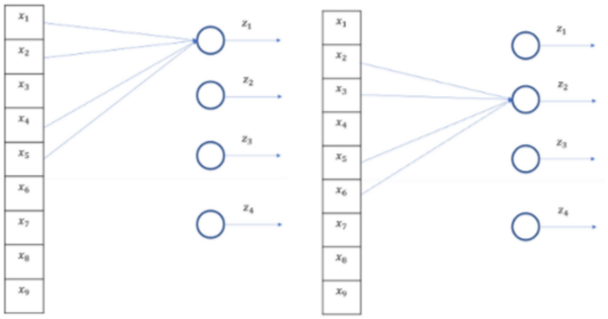
$$z_1 = w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5 + b_1 \qquad z_3 = w_1x_4 + w_2x_5 + w_3x_7 + w_4x_8 + b_1$$

$$z_2 = w_1x_2 + w_2x_3 + w_3x_5 + w_4x_6 + b_1 \qquad z_4 = w_1x_5 + w_2x_6 + w_3x_8 + w_4x_9 + b_1$$

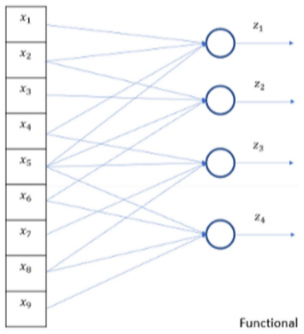
  - For FCL with 4 units:



  - FCL Sparsely Connected: use 4 connections instead of 9

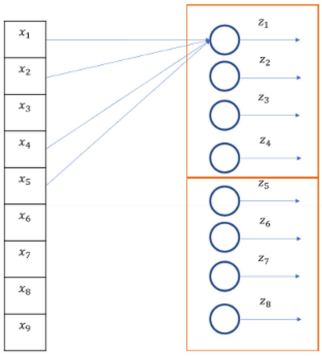


  - Parameter Sharing: share same 3 weights and 1 bias param



$$z_1 = w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5 + b_1$$
$$z_2 = w_1x_2 + w_2x_3 + w_3x_5 + w_4x_6 + b_1$$
$$z_3 = w_1x_4 + w_2x_5 + w_3x_7 + w_4x_8 + b_1$$
$$z_4 = w_1x_5 + w_2x_6 + w_3x_8 + w_4x_9 + b_1$$

- convolutional layer is just a FCL with sparse connectivity and parameter sharing!



Filter 1

$$z_1 = w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5 + b_1$$
$$z_2 = w_1x_2 + w_2x_3 + w_3x_5 + w_4x_6 + b_1$$
$$z_3 = w_1x_4 + w_2x_5 + w_3x_7 + w_4x_8 + b_1$$
$$z_4 = w_1x_5 + w_2x_6 + w_3x_8 + w_4x_9 + b_1$$

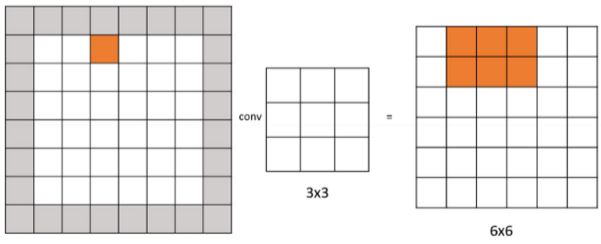
Filter 2

$$z_5 = w_5x_1 + w_6x_2 + w_7x_4 + w_8x_5 + b_2$$
$$z_6 = w_5x_2 + w_6x_3 + w_7x_5 + w_8x_6 + b_2$$
$$z_7 = w_5x_4 + w_6x_5 + w_7x_7 + w_8x_8 + b_2$$
$$z_8 = w_5x_5 + w_6x_6 + w_7x_8 + w_8x_9 + b_2$$

- Fully-Connected Neural Network Layer:
  - $a^{[l]} = g(W^{[l]} \cdot a^{[l-1]} + b^{[l]})$
- Convolutional Neural Network Layer
  - $a^{[l]} = g(conv(W^{[l]}, a^{[l-1]}) + b^{[l]})$

Convolutional Layers: Padding

- spatial dimensions h,w shrinks with every conv layer
- # output activations depends on how many times filter fits in input
- **Problem 1: Data shrinks**
  - want to use conv layer output as input, but output volume's spatial dim shrinks
  - limits how deep CNN
- **Problem 2: Data at edge**
  - input data at edge influence fewer output vals than input data in middle
- **Solution: Padding**
  - pad perimeter of input volume before conv operation
  - eg. 6x6 img with 1 element padding becomes 8x8
  - output is now 6x6, this preserves original spatial dim
  - pad with value of 0



- Input volume: (h, w, c)
- Filter size: (f, f, c)
- Output volume without padding: $(h - f + 1, w - f + 1)$
- With padding $(h + 2p - f + 1, W + 2p - f + 1)$

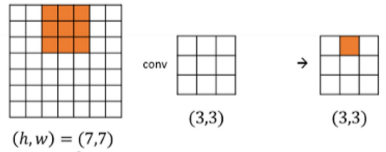
- **No padding: Valid**
- **Pad so that output volume is same as input volume: Same**

- Input is (h, w), output is $(h + 2p - f + 1, w + 2p - f + 1)$
- What to set p so that input is equal to output?
- $h = h + 2p - f + 1 \rightarrow p = \frac{f-1}{2}$
- $w = w + 2p - f + 1 \rightarrow p = \frac{f-1}{2}$
    - filters usually odd sized and square
    - p should be = integer

Stride

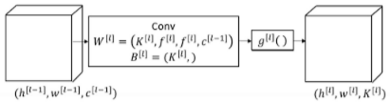
- can slide filter by larger steps, another hyperparameter of CNN
- eg. stride = 2
- compression/downsampling of the feature map, shrinks volumes in controlled fashion, control size

- Input Volume: (h, w, c)
- Output Volume = $\left(\frac{h+2p-f}{s} + 1, \frac{w+2p-f}{s} + 1, K\right)$



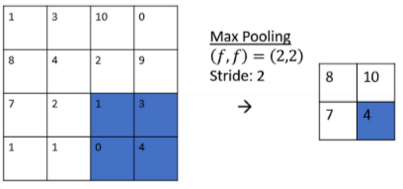
$(h, w) = (7,7)$
$p = 0$
$s = 2$

- Hyperparameters
    - Number of filters K
    - Filter size (f, f)
    - Stride s
    - Padding p
- Input Volume
    - $(h^{[l-1]}, w^{[l-1]}, c^{[l-1]})$
- Output Volume
    - $\left(\frac{h^{[l-1]}+2p-f}{s} + 1, \frac{w^{[l-1]}+2p-f}{s} + 1, K\right)$
- # Learned parameters
    - $K * (f * f * c^{[l-1]} + 1)$



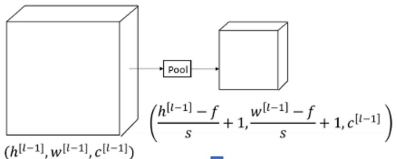
Final Layes of a CNN

- Flatten final volume
- use 1+ FCL
- Final volume must be manageable size
- conv layers = feature extractors
- **Control volume size**
    - final volume depends on stride, padding of conv layers
- **Pooling Layer**
    - output is max value of each region
    - compress data, discard all but the strongest signal
    - adds flexibility to feature detection, tolerance to translation (jitter)



Pooling

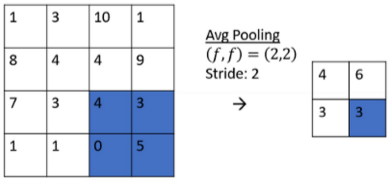
- Hyperparameters
    - Pooling function
    - Pool size (f, f)
    - Stride s



Usually, $f = s$

- No learned parameters!
- Reduces spatial dimensions but NOT channel dimension $\left(\frac{h^{[l-1]}}{s}, \frac{w^{[l-1]}}{s}, c^{[l-1]}\right)$

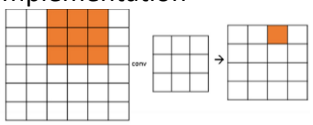
- **Average Pooling**
    - average value within each region



14. Convolutional Layer Implementation Details

- Non vectorized implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$
Weight is tensor (K, f, f, c_{in})
Output is tensor (h_{out}, w_{out}, K, m)

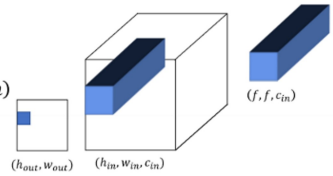



```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    for x in range(f):
     for y in range(f):
      for c in range(cin):
       output[…] += input[…] * weight[…]
```

Convolution of filter with part of input

- Partially vectorized:

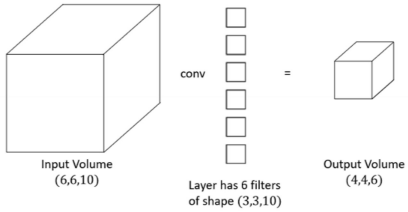
Input is tensor $(h_{in}, w_{in}, c_{in}, m)$
Weight is tensor (K, f, f, c_{in})
Output is tensor (h_{out}, w_{out}, K, m)



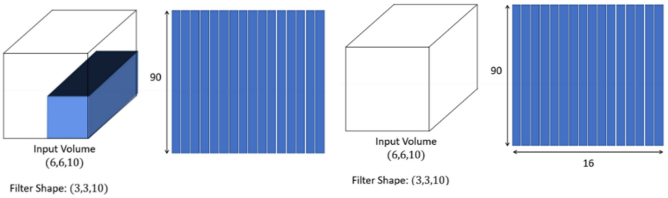

```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    output[…]=conv(input[…], weight[…])
```

Showing One filter, one sample

- **Vectorized implementation**
  - implemented as matrix mult
  - Transforming convolution into matrix mult
  - eg. conv layer

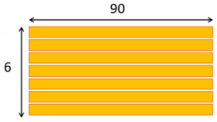


1. Transform input volume into 2D matrix:



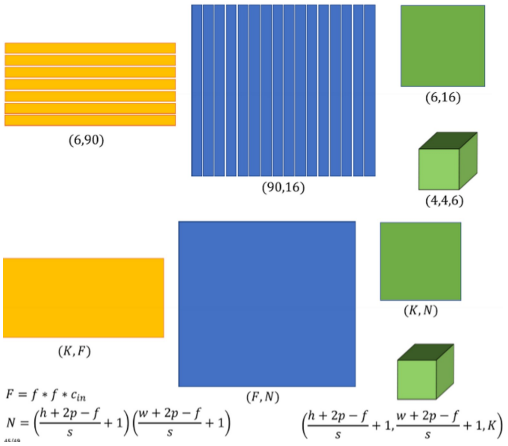
2. Transform Filters in Conv Layer into 2D Matrix

Layer has 6 filters of shape $(3,3,10)$



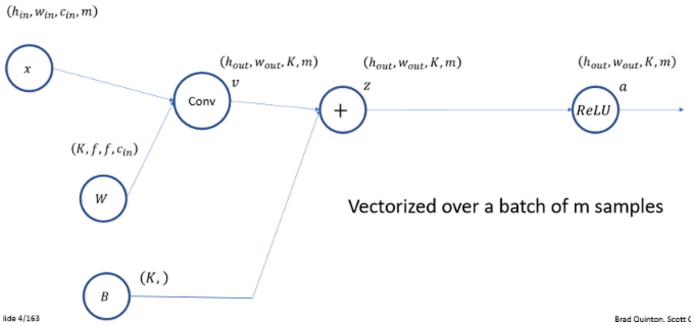
- Reshape each filter into a vector
- Each filter is a row in new matrix

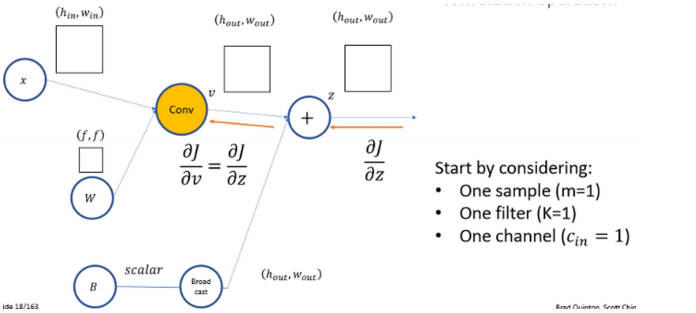
3. Reshape output



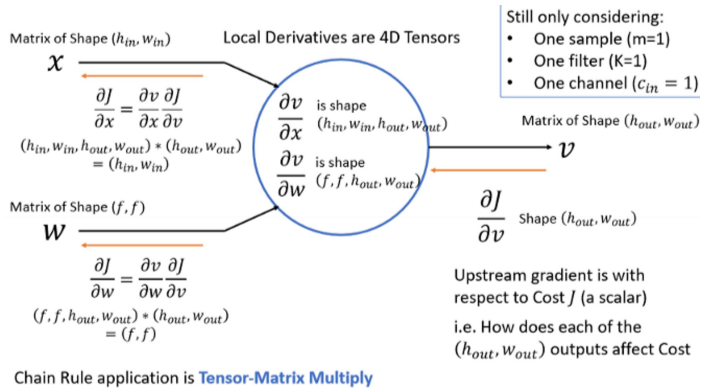
- transform input volume using **im2col**
- transform weight matrix using **w.reshape(K, -1)**
- transform final output using same reshape func

15. CNN Back Propagation





- **Back propagation for x and w:**
  - **Compute backprop through conv operation**
    - Local derivatives = 4D tensors
    - Chain rule: Tensor-Matrix Mult



Chain Rule application is **Tensor-Matrix Multiply**

$(h_{in}, w_{in}) = (3,3)$

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}

conv

$(f, f) = (2,2)$

w_{11}	w_{12}
w_{21}	w_{22}

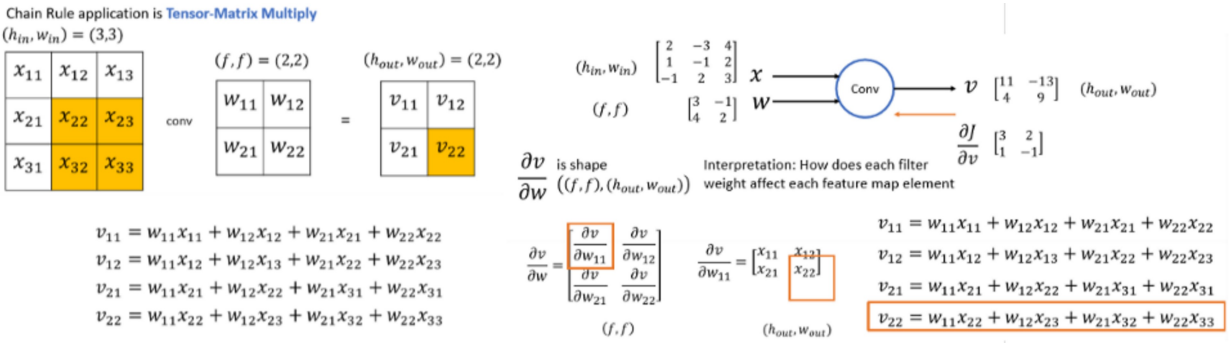
=

$(h_{out}, w_{out}) = (2,2)$

v_{11}	v_{12}
v_{21}	v_{22}

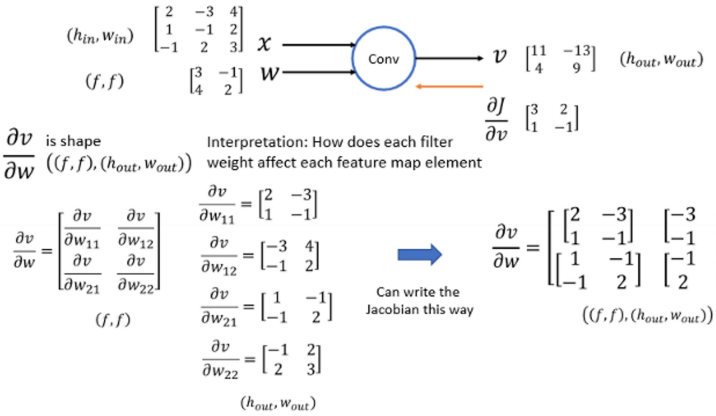
$$v_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$$
$$v_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$$
$$v_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{31}$$
$$v_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}$$

Chain Rule application is Tensor-Matrix Multiply



$$v_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$$
$$v_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$$
$$v_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{31}$$
$$v_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}$$

- **Find Jacobian Matrix**
  - each jacobian slice is a sliding window over input x



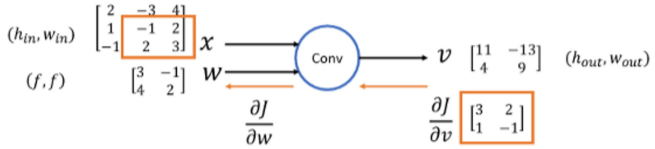
- **Apply chain rule to compute weight gradients**

$$\frac{\partial J}{\partial w} = \frac{\partial v}{\partial w}\frac{\partial J}{\partial v} = \begin{bmatrix} \begin{bmatrix} 2 & -3 \\ 1 & -1 \end{bmatrix} & \begin{bmatrix} -3 & 4 \\ -1 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} & \begin{bmatrix} -1 & 2 \\ 2 & 3 \end{bmatrix} \end{bmatrix} \begin{bmatrix} 3 & -2 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 12 & -20 \\ 2 & -8 \end{bmatrix}$$

$((f, f), (h_{out}, w_{out}))$ (h_{out}, w_{out}) (f, f)

$$\frac{\partial J}{\partial w_{22}} = -1 * 3 + 2 * (-2) + 2 * 1 + 3 * (-1) = -8$$

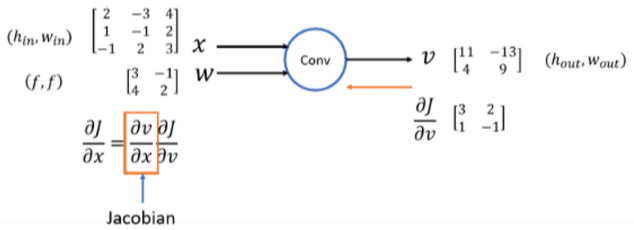
- **Can be done without jacobian, using a convolution operation**

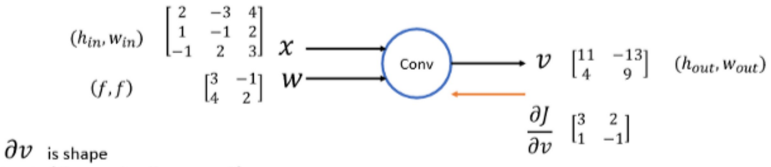


Instead of using Jacobian

$$\frac{\partial J}{\partial w} = x \text{ conv } \frac{\partial J}{\partial v} = \begin{bmatrix} 12 & -20 \\ 2 & -8 \end{bmatrix}$$

- **For dJ/dx:**





$\frac{\partial v}{\partial x}$ is shape $((h_{in}, w_{in}), (h_{out}, w_{out}))$

$$\frac{\partial v}{\partial x} = \begin{bmatrix} \frac{\partial v}{\partial x_{11}} & \frac{\partial v}{\partial x_{12}} & \frac{\partial v}{\partial x_{13}} \\ \frac{\partial v}{\partial x_{21}} & \frac{\partial v}{\partial x_{22}} & \frac{\partial v}{\partial x_{23}} \\ \frac{\partial v}{\partial x_{31}} & \frac{\partial v}{\partial x_{32}} & \frac{\partial v}{\partial x_{33}} \end{bmatrix} \quad \frac{\partial v}{\partial x_{22}} = \begin{bmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{bmatrix}$$

(h_{in}, w_{in}) (h_{out}, w_{out})

$$\begin{aligned} v_{11} &= w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22} \\ v_{12} &= w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23} \\ v_{21} &= w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{31} \\ v_{22} &= w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33} \end{aligned}$$

- **This is another convolution:**

$$\frac{\partial J}{\partial x} = \frac{\partial v}{\partial x}\frac{\partial J}{\partial v} = \begin{bmatrix} \begin{bmatrix} w_{11} & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} w_{12} & w_{11} \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & w_{12} \\ 0 & 0 \end{bmatrix} \\ \begin{bmatrix} w_{21} & 0 \\ w_{11} & 0 \end{bmatrix} & \begin{bmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{bmatrix} & \begin{bmatrix} 0 & w_{22} \\ 0 & w_{12} \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ w_{21} & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ w_{22} & w_{21} \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & w_{22} \end{bmatrix} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial v_{11}} & \frac{\partial J}{\partial v_{12}} \\ \frac{\partial J}{\partial v_{21}} & \frac{\partial J}{\partial v_{22}} \end{bmatrix} =$$

$$\begin{bmatrix} w_{11}\frac{\partial J}{\partial v_{11}} & w_{12}\frac{\partial J}{\partial v_{11}} + w_{11}\frac{\partial J}{\partial v_{12}} & w_{12}\frac{\partial J}{\partial v_{12}} \\ w_{21}\frac{\partial J}{\partial v_{11}} + w_{11}\frac{\partial J}{\partial v_{21}} & w_{22}\frac{\partial J}{\partial v_{11}} + w_{21}\frac{\partial J}{\partial v_{12}} + w_{12}\frac{\partial J}{\partial v_{21}} + w_{11}\frac{\partial J}{\partial v_{22}} & w_{22}\frac{\partial J}{\partial v_{12}} + w_{12}\frac{\partial J}{\partial v_{22}} \\ w_{21}\frac{\partial J}{\partial v_{21}} & w_{22}\frac{\partial J}{\partial v_{21}} + w_{21}\frac{\partial J}{\partial v_{22}} & w_{22}\frac{\partial J}{\partial v_{22}} \end{bmatrix}$$

- **(why do we rotate conv twice?)**

0	0	0	0
0	$\frac{\partial J}{\partial v_{11}}$	$\frac{\partial J}{\partial v_{12}}$	0
0	$\frac{\partial J}{\partial v_{21}}$	$\frac{\partial J}{\partial v_{22}}$	0
0	0	0	0

conv

w_{22}	w_{21}
w_{12}	w_{11}

=

$$\begin{bmatrix} w_{11}\frac{\partial J}{\partial v_{11}} & w_{12}\frac{\partial J}{\partial v_{11}} + w_{11}\frac{\partial J}{\partial v_{12}} & w_{12}\frac{\partial J}{\partial v_{12}} \\ w_{21}\frac{\partial J}{\partial v_{11}} + w_{11}\frac{\partial J}{\partial v_{21}} & w_{22}\frac{\partial J}{\partial v_{11}} + w_{21}\frac{\partial J}{\partial v_{12}} + w_{12}\frac{\partial J}{\partial v_{21}} + w_{11}\frac{\partial J}{\partial v_{22}} & w_{22}\frac{\partial J}{\partial v_{12}} + w_{12}\frac{\partial J}{\partial v_{22}} \\ w_{21}\frac{\partial J}{\partial v_{21}} & w_{22}\frac{\partial J}{\partial v_{21}} + w_{21}\frac{\partial J}{\partial v_{22}} & w_{22}\frac{\partial J}{\partial v_{22}} \end{bmatrix}$$

0	0	0	0
0	$\frac{\partial J}{\partial v_{11}}$	$\frac{\partial J}{\partial v_{12}}$	0
0	$\frac{\partial J}{\partial v_{21}}$	$\frac{\partial J}{\partial v_{22}}$	0
0	0	0	0

conv

w_{22}	w_{21}
w_{12}	w_{11}

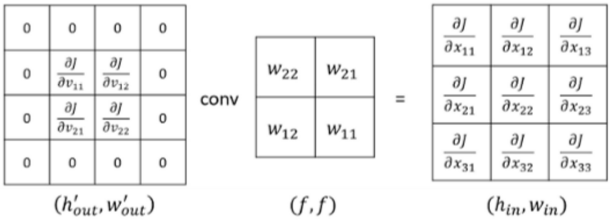
=

$\frac{\partial J}{\partial x_{11}}$	$\frac{\partial J}{\partial x_{12}}$	$\frac{\partial J}{\partial x_{13}}$
$\frac{\partial J}{\partial x_{21}}$	$\frac{\partial J}{\partial x_{22}}$	$\frac{\partial J}{\partial x_{23}}$
$\frac{\partial J}{\partial x_{31}}$	$\frac{\partial J}{\partial x_{32}}$	$\frac{\partial J}{\partial x_{33}}$

$$\frac{\partial J}{\partial x} = pad\left(\frac{\partial J}{\partial v}\right) \text{ conv rot180}(w)$$

- No Jacobian require at all!
- This convolution yields same result as doing the full 4D-Tensor Jacobian and upstream gradient matrix multiply!

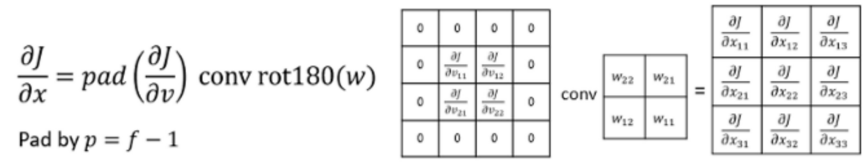
- How much padding do we want?



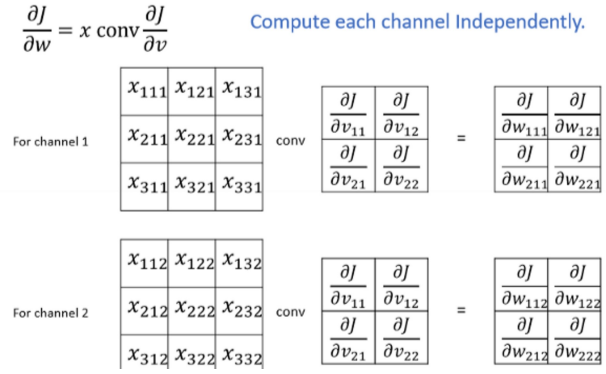
$$h_{out} = h_{in} - f + 1 \qquad w_{out} = w_{in} - f + 1$$
$$h'_{out} = h_{out} + 2p_h \qquad w'_{out} = w_{out} + 2p_w$$
$$h_{in} = h'_{out} - f + 1 \qquad w_{in} = w'_{out} - f + 1$$

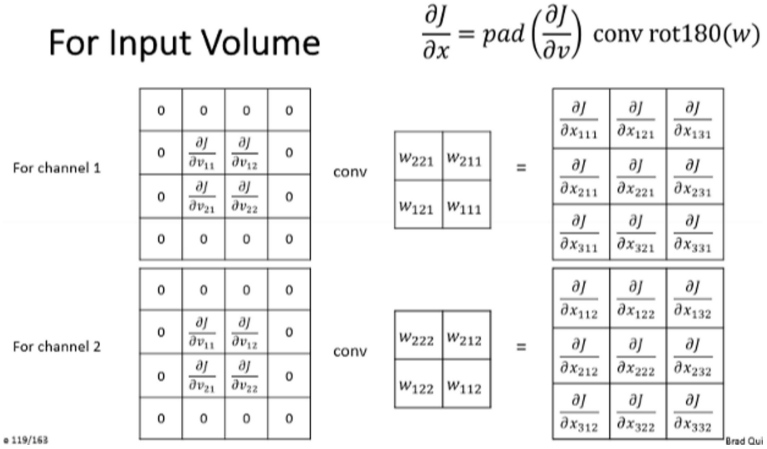
$$p_h = f - 1 \qquad p_w = f - 1$$

- **SUMMARY (m=1, K=1, c_in=1)**

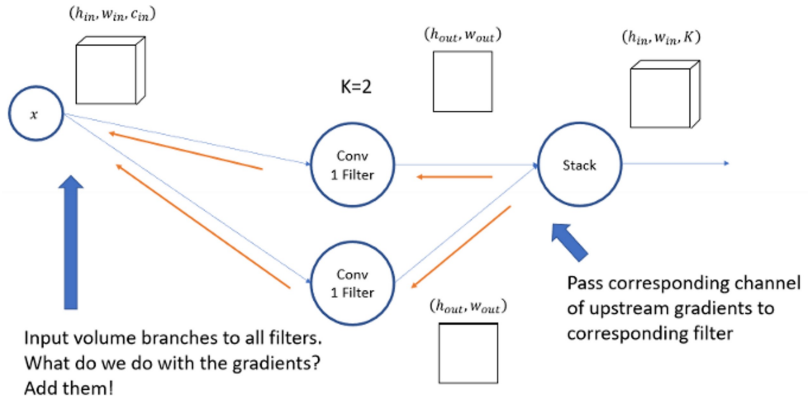


- **For multiple Channels (c_in != 1):**
  - each channel of weights operate independently on each channel of input volume
  - compute each channel independently



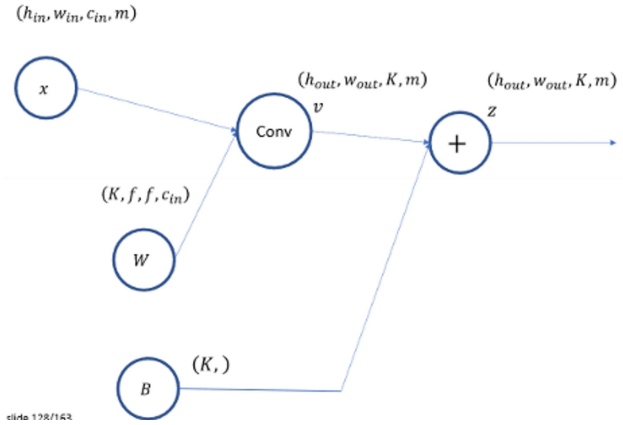


- **For multiple filters (K != 1)**



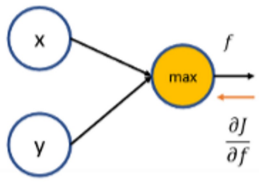
Multiple Samples

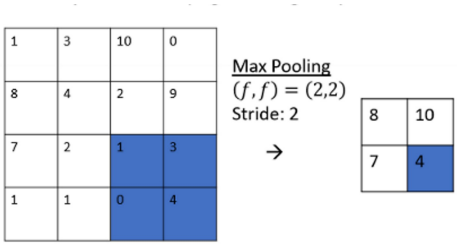
- add m for size param

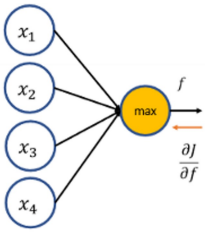


Back pro through maxpooling (max)

- only one input can affect the output at any time, gradient routed to largest var







Gradient Check for Back prop

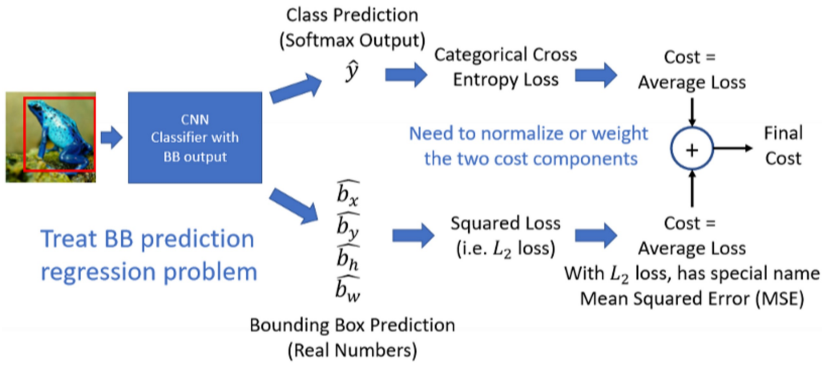
- easy to make mistakes when doing back prop
- write tests for code in software dev
- Can use numerical gradient checking instead of backprop tests
    - too slow for regular use, need to make 2 calls to fwd prop func for each gradient value

17. CNN Applications

Object Localization and Detection

- **Localization**

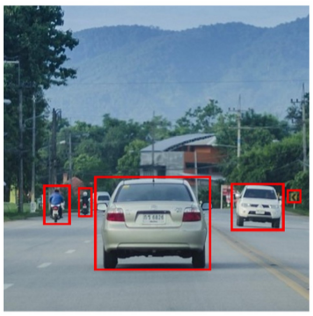
- Outputs:
  - class prediction
  - bounding box (b_x, b_y, b_w, b_h)
- Start with CNN Classifier (ZFNet)



- **Squared Loss (L2 loss)**
  - square of diff bw predictions and true vals

$$L(b_x, b_y, b_w, b_h, \widehat{b_x}, \widehat{b_y}, \widehat{b_w}, \widehat{b_h}) = (b_x - \widehat{b_x})^2 + (b_y - \widehat{b_y})^2 + (b_h - \widehat{b_h})^2 + (b_w - \widehat{b_w})^2$$

- **Landmark Detection**
  - want to output x,y coord (with h, w) of bounding box
  - **Face detection**
    - FC layer predicts 2 num (x,y) for each landmark
- **Pose Detection**
  - define a landmark for each joint
- **Classification and localization of zero + objects**
  - cant use directly localization approach, since this is for a fixed number of obj
  - **Sliding window**
    - start with trained CNN classifier that knows about these classes and a "none of the above" class
    - supply crops of the img to CNN via sliding window



  - What window shapes/stride?

Sliding Window Locations for one window of shape (b_h, b_w) in an image of shape (H, W):

$$(H - b_h + 1) \cdot (W - b_w + 1)$$

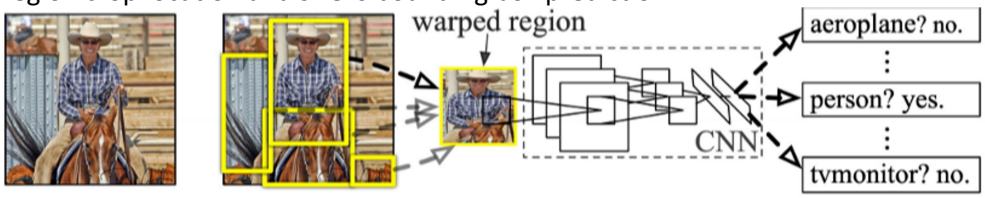
Repeat for all possible window shapes

$$\sum_{b_h=1}^{H} \sum_{b_w=1}^{W} (H - b_h + 1) \cdot (W - b_w + 1)$$

    - infeasible to look at all possible window sizes at all locations iteratively

Regions with CNN Features (R-CNN)

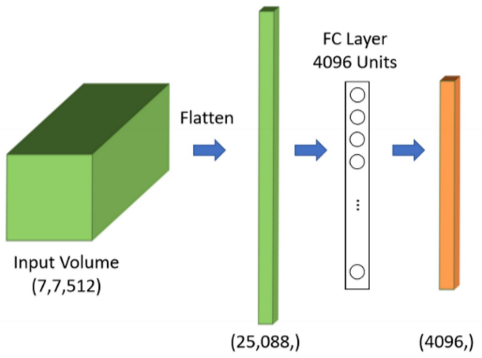
- Use region proposal alg to find manageable number of regions (crops) that could have an obj
- send region crops to classifier
- region crop location and size is bounding box prediction



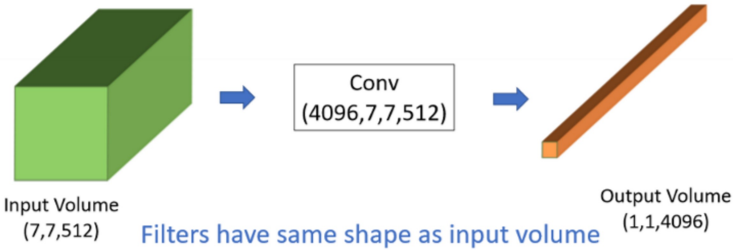
- **Faster R-CNN**
  - R-CNN
    - Propose regions. Evaluate one region at a time
  - Fast R-CNN
    - Classify all proposed regions at once
  - Faster R-CNN
    - Uses a CNN to propose regions

You only look once (YOLO)

- sliding window via convolution
- can evaluate all sliding window locations in one pass, Fast!
- restrictions on stride and size of sliding window
- **Building FC w conv layer:**
  - **FC Layer:**

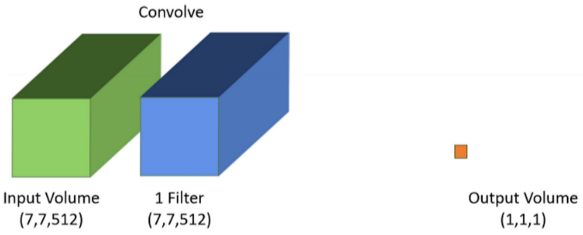


  - **Conv Layer (same result w conv):**



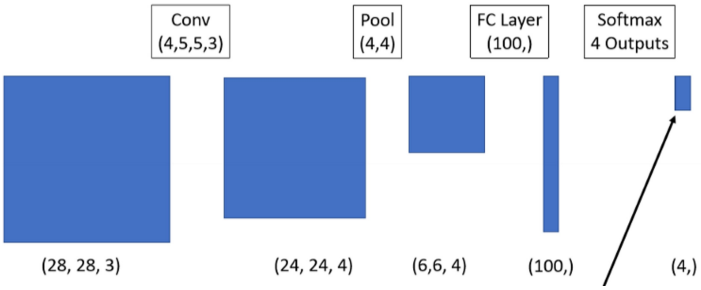
Filters have same shape as input volume

- eg. 1 filter



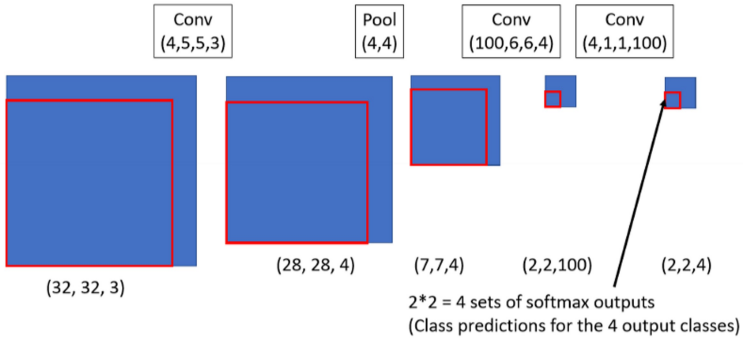
- **Sliding Window via Conv FC Layers**
  - start with CNN classifier
  - convert FC layers to use conv equivalent implementation
  - supply larger img for obj detection
  - each sliding window location is potential bounding box for obj
  - eg.

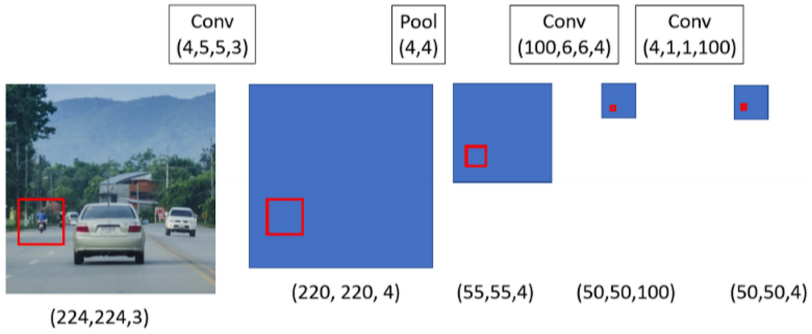
- CNN Classifier with 4 softmax outputs to predict
  - Car
  - Motorcycle
  - Street sign
  - Other
- Input is 28x28 color image



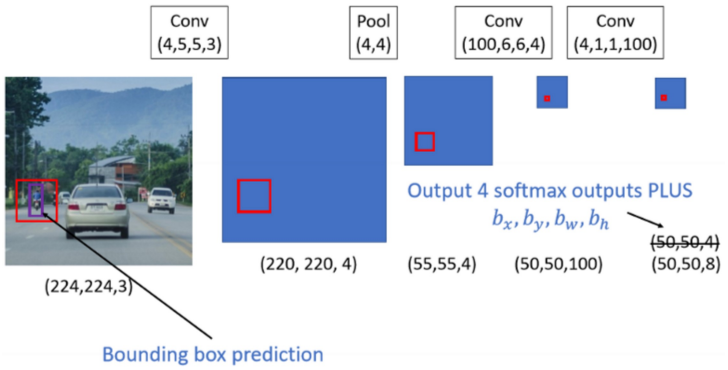
  - One set of softmax outputs (class predictions for 4 output classes)

- **Convert FC layers to use conv impl**





- Problem:
  - objects may not fit perfectly inside sliding window, inaccurate bounding box predictions
- Solution:
  - instead of applying a CNN classifier at each sliding window location, apply CNN classifier+localizer
  - outputs a bb prediction + class prediction



- Detecting Multiple Objects in Same Sliding Window location
  - so far, can only detect 1 obj at each sliding window, doesn't work well for obj larger than window
  - YOLO has Anchor Boxes
    - change localizer to predict up to X obj at each lcoation with predefined bb shapes

Applications beyond Classification and Detection

- **Image Retrieval**
  - use final flattened vol as signature of img
  - find similar img with similar signatures
  - given new img, find img w/ smalles euclidian dist bw signature vectors
- **Visualization Feature Vectors**
- **Saliency Maps**
- **Saliency via Occlusion**
- **Segmentation with Saliency Maps**