Machine Learning (CE 40717)

Fall 2024

Ali Sharifi-Zarchi

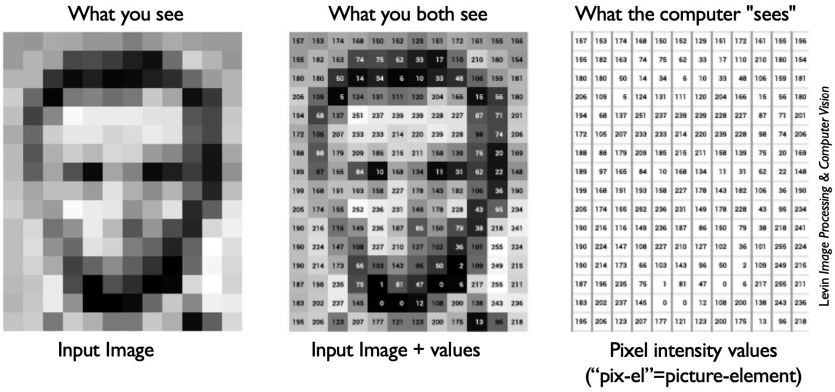
CE Department
Sharif University of Technology

October 27, 2024

How do humans see?

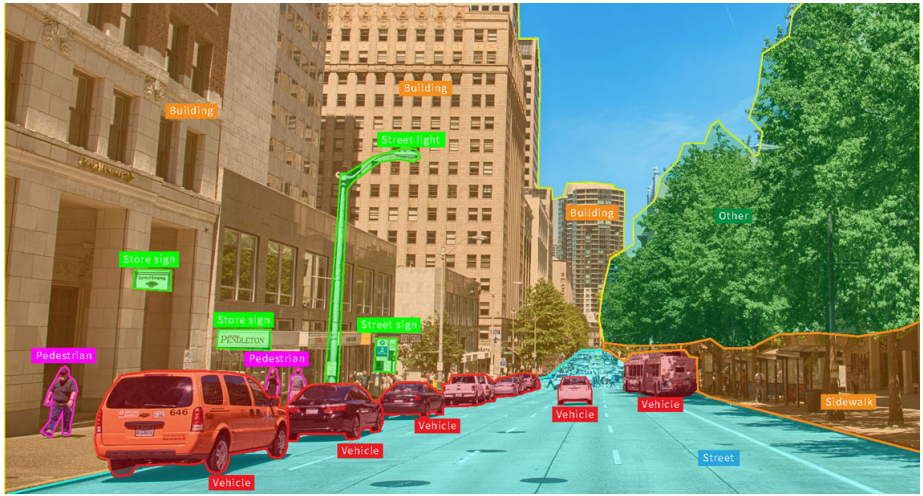
How can we enable computers to see?

What computers see: images as numbers



An image is just a matrix of numbers [0,255].i.e., $1080 \times 1080 \times 3$ for an RGB image.
Question: is this Lincoln? Washington? Jefferson? Obama? How can the computer answer this question?

What is computer vision? (1)



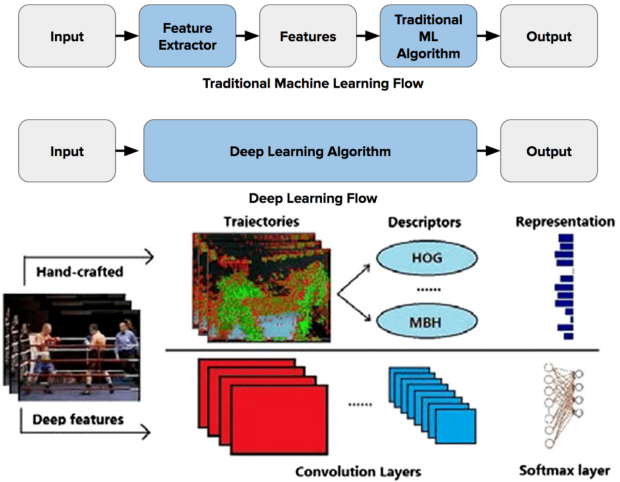
What is computer vision? (2)

Noisy Image

Denoised Image

What is computer vision? (3)

Hand-crafted features: before deep learning features (1)



Two types of features used for action recognition, i.e., hand-crafted features and deep learning features

Hand-crafted features: before deep learning features (2)

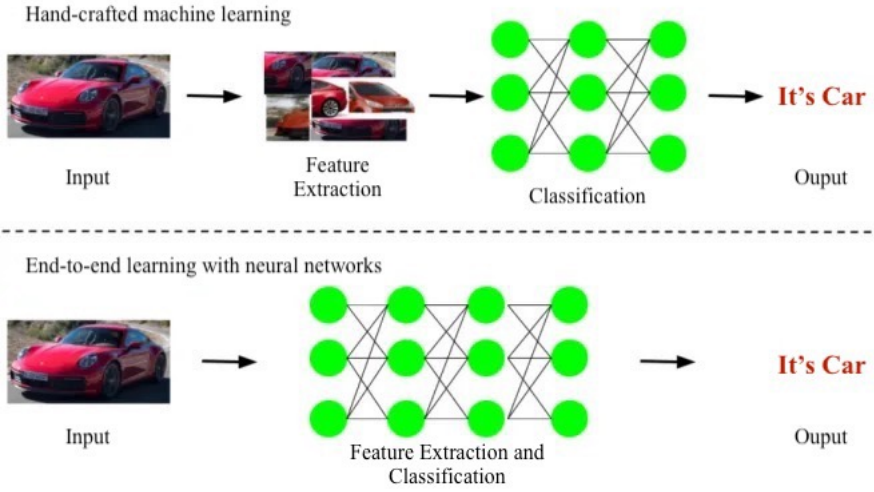
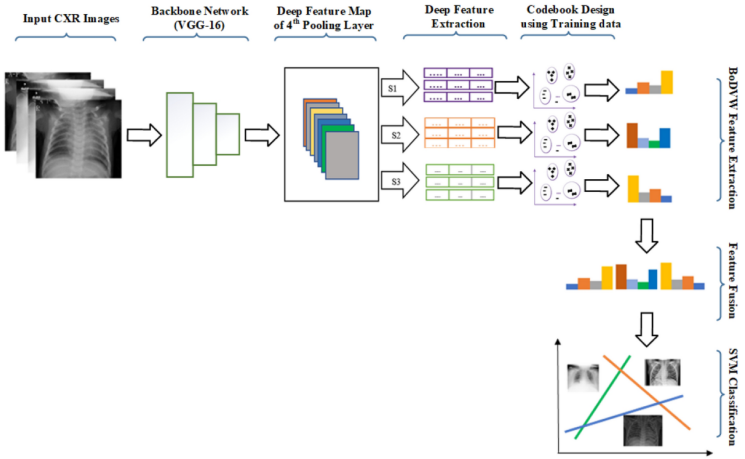
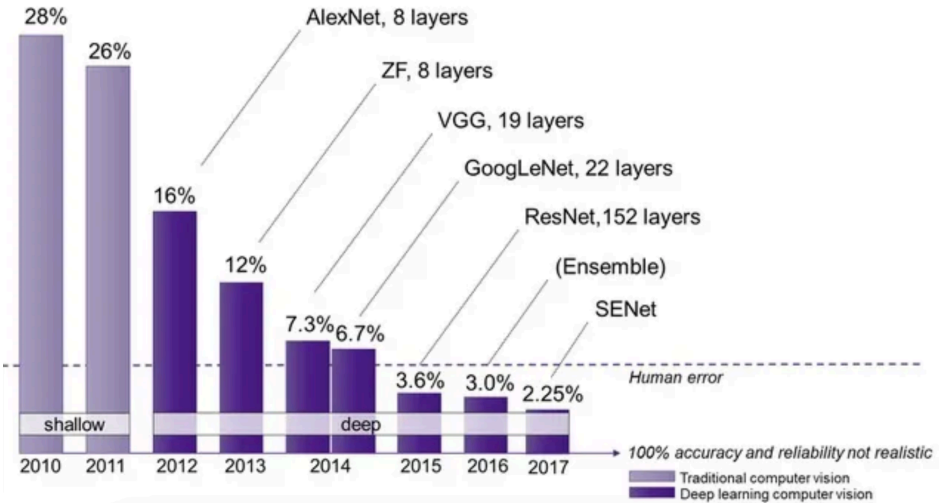


Image features



Fusion of multi-scale bag of deep visual words features of chest X-ray images to detect COVID-19 infection

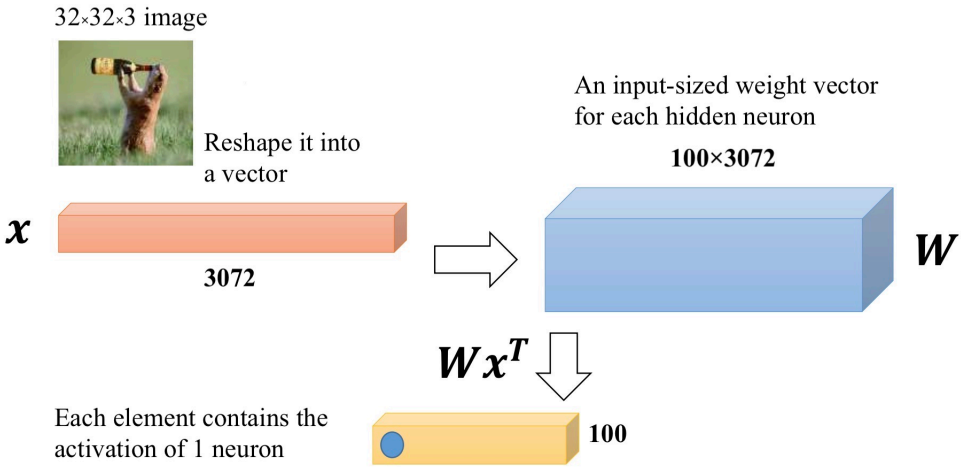
Effect of dl



Comparison between Deep Learning and Traditional Models

FCNs on images

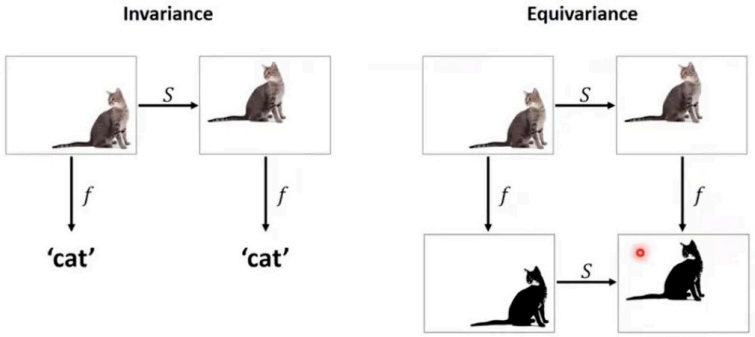
Dense Vector Multiplication



Fully connected layers

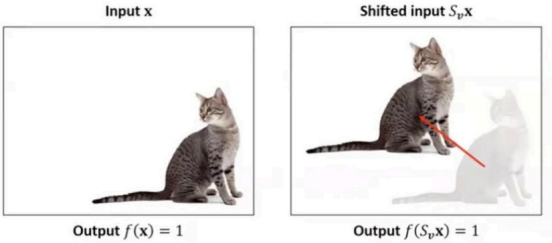
- Neurons in a single layer operate independently and do not share connections, even for inputs.
- To be shift-invariant, many samples (in various time or locations) must be shown to them.
- **Regular Neural Nets dont scale well to full images**
  - Parameters would add up quickly!
  - Full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Invariance vs equivariance



- **Invariance:** The output remains the same when the input undergoes a transformation.
- **Equivariance:** The output varies predictably when the input undergoes a transformation.

Shift-invariance



- *Cat detector* $f : \mathbb{R}^d \to \mathbb{R}$
- *Shift operator* $S_v : \mathbb{R}^d \to \mathbb{R}^d$ shifting the image by vector v

Shift invariance: $f(\mathbf{x}) = f(S_v\mathbf{x})$

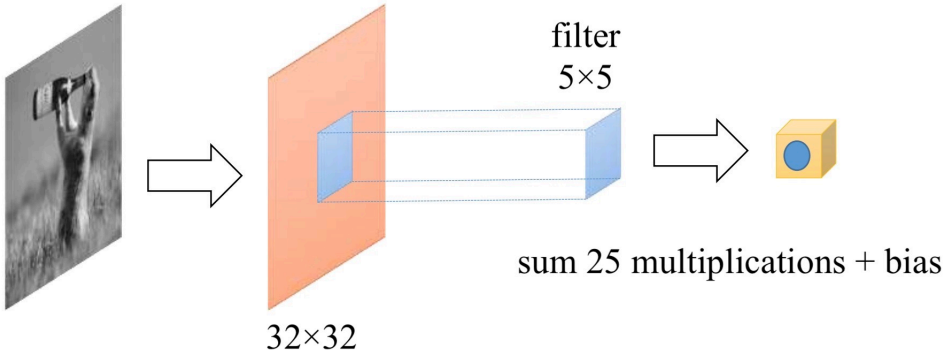
Will an MLP that recognizes the left image as a cat also recognize the shifted image on the right as a cat?

A problem

- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern matters matters
- Traditional MLPs are sensitive to pattern location
  - Moving it by one component results in an entirely different input that the MLP wont recognize
- **Requirement:** Network must be *shift invariant*

Convolution on images

Convolution (Refresher)



Matrix input preserving spatial structure

Fully connected or convolution

Question: Can I just do classification on an flatten image (e.g., a 1080x1080x3 image) with fully connected layers?

Answer: No, using fully connected layers on such a large image is inefficient due to:

- Loss of spatial structure
- High parameter count

Why Use Convolution:

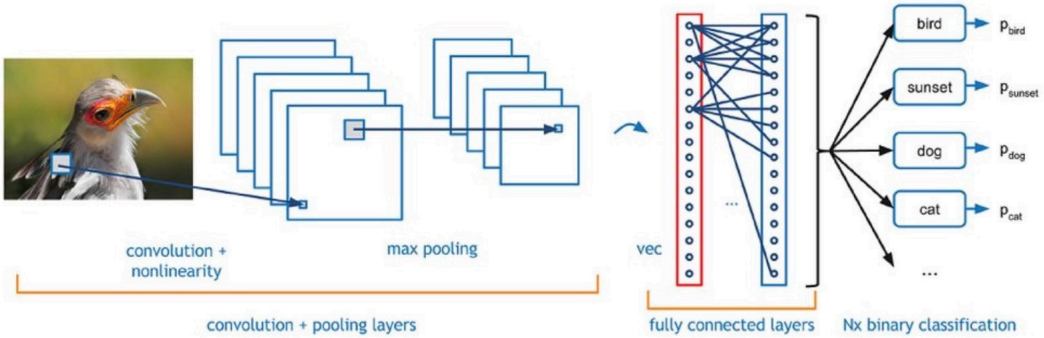
- **Exploit spatial structure:** Convolution sees local patterns by using filters over small patches of the image.
- **Reduce parameters:** By sharing weights across the image, convolution reduce the number of parameters.
- **Build hierarchical features:** Convolution starts with small patterns, then combines them to recognize more complex patters.

What is cnn?

- It is a class of deep learning.
- Convolutional neural network (ConvNets or CNNs) is one of the main categories to do image recognition, image classifications, object detection, recognition of faces, etc.
- It is similar to the basic neural network. CNNs also have learnable parameters like weights and biases, similar to neural networks.
- CNN is heavily used in computer vision.
- There are 3 basic components to define CNN:
  - The Convolution Layer
  - The Pooling Layer
  - The Output Layer (or) Fully Connected Layer

Architecture of CNN

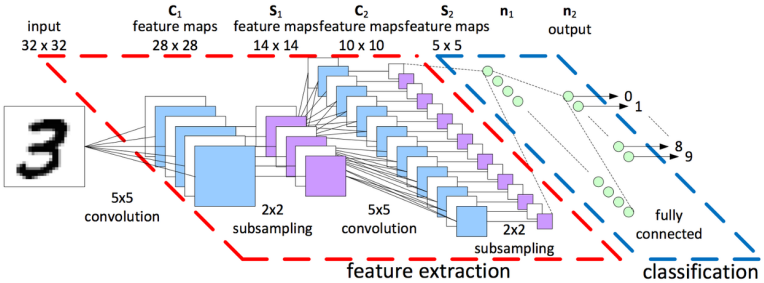
The basic idea of Convolutional Neural Networks (CNNs) is similar to Backpropagation Neural Networks (BPNNs) but differs in implementation.



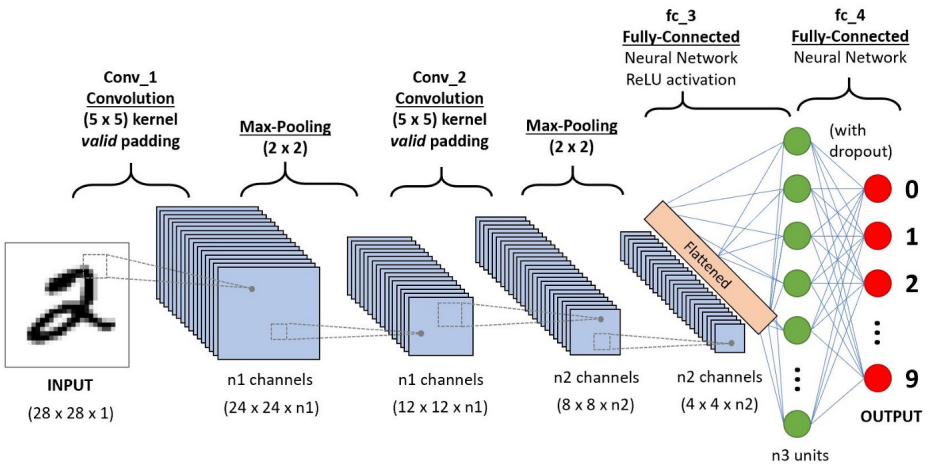
After vectorized (vec), the 2D arranged inputs become ID vectors. Then the network is just like a BPNN (Back propagation neural networks)

The basic structure (1)

- Alternating Convolution (C) and subsampling layer (S)
- Subsampling allows flexible positioning of features.



The basic structure (2)



The basic structure (3)

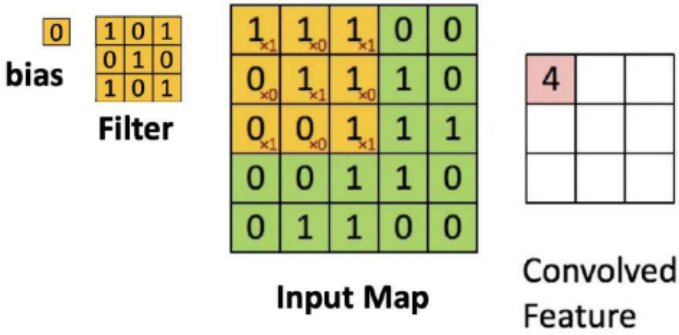
Three Main Types of Layers

- **Convolutional Layer**
  - Output of neurons are connected to local regions in the input.
  - Applying the same filter across the entire image.
  - CONV layers parameters consist of a set of learnable filters.
- **Pooling Layer**
  - Performs a downsampling operation along the spatial dimensions.
- **Fully-Connected Layer**
  - Typically used in the final stages of the network to combine high-level features and make predictions.

Cnn vs. FCN

- **Fully Connected Networks (FCNs):**
  - High number of parameters, leading to overfitting on large inputs (e.g., images).
  - Lack of spatial awareness, making it sensitive to the exact positioning of patterns.
  - Suitable for structured data but inefficient for image processing.
- **Convolutional Neural Networks (CNN):**
  - Uses filters to process only small parts of the input at a time (locality).
  - Weight sharing and local connectivity reduce the number of parameters.
  - Can recognize patterns independent of their location within the image (shift invariance).
  - Efficient for image, video, and speech data.

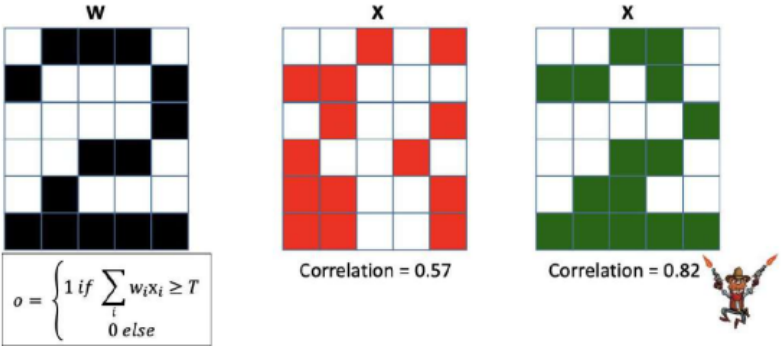
What is a convolve



Scanning an image with a "filter"

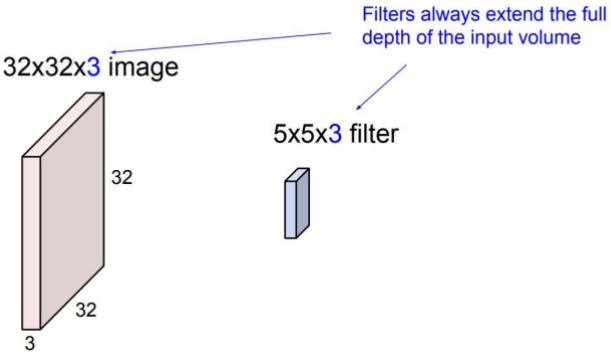
- A filter is really just a perceptron, with weights and a bias.
- At each location, the filter is multiplied component-wise with the underlying map values, and the products are summed along with the bias.

Weights showing correlation



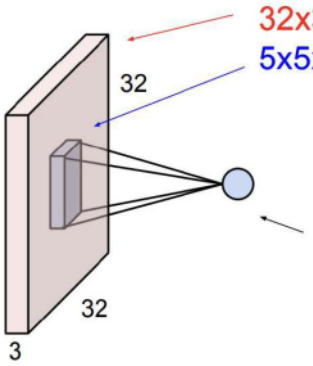
- The weights of the filter represent the appearance of the number "2".
- The green has a higher correlation with the filter compared to the red.
- The green pattern is more likely to represent the number "2".

What is convolution



- **Convolve:** Slide over the image spatially, computing dot products.
- This allows us to preserve the spatial structure of the input.

What is the output

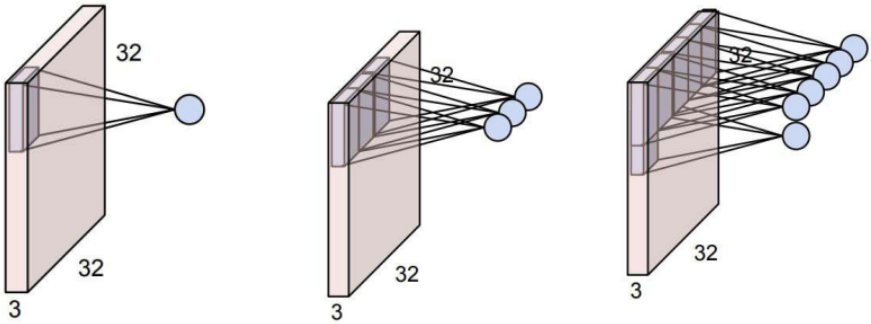


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

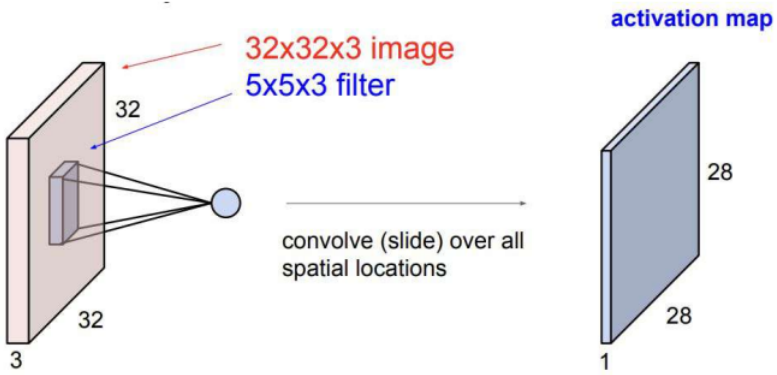
$$w^T x + b$$

It's simply a neuron with local connectivity!

Convolution process (1)

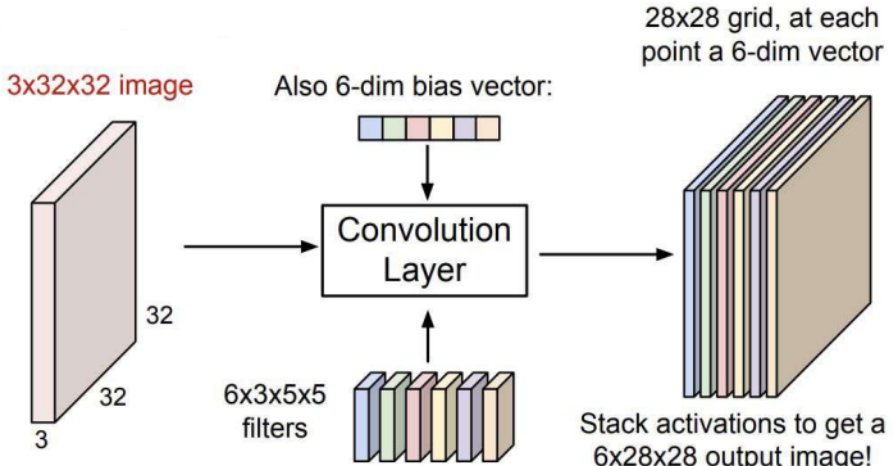


Convolution process (2)

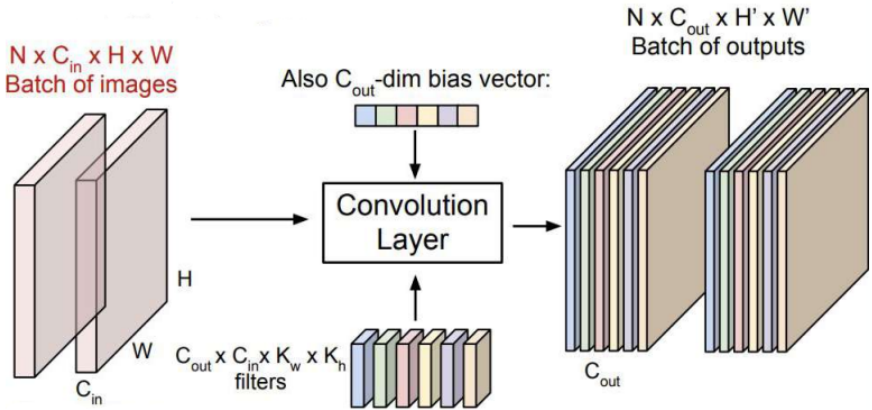


If we consider the image to be of size $n \times m \times b$ and the filter to be of size $n' \times m' \times b$, we will have an activation map of size $(n - n' + 1) \times (m - m' + 1) \times 1$ for each filter.

Convolution process (3)

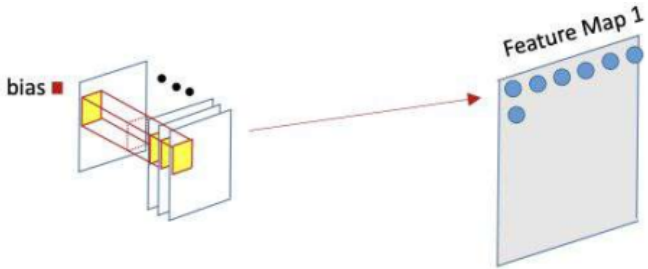


Convolution process (4)



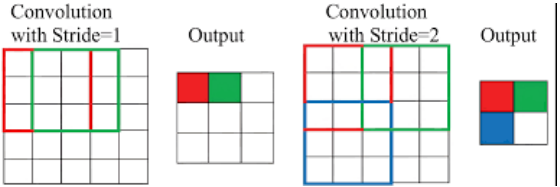
Batching input images can provide a more generalized representation, as described above.

A different view



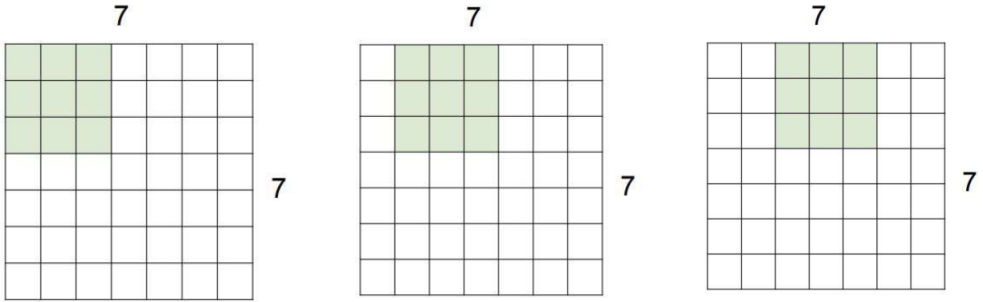
$$z(i,j,s) = \underbrace{\sum_{p} \underbrace{\sum_{k=1}^{L} \sum_{l=1}^{L} w(k,l,p,s)\, a(i+l-1, j+k-1, p)}_{\text{One map}} + b(s)}_{\text{All maps}}$$

Stride (1)



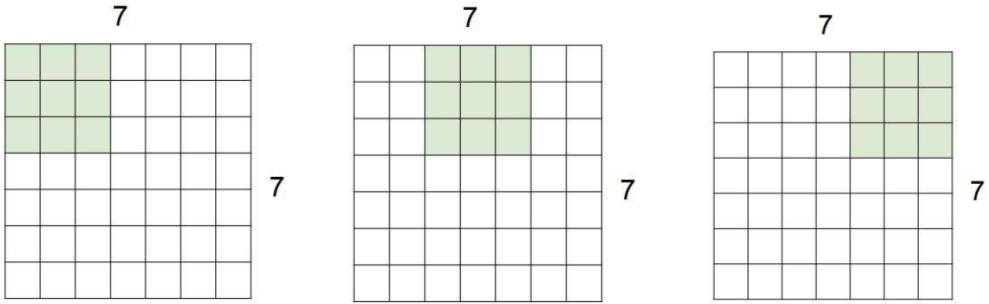
- The scans of the individual "filters" may advance by more than one pixel at a time
  - The "stride" may be greater than 1
  - Effectively increasing the granularity of the scan
    - Saves computation, sometimes at the risk of losing information
- This results in a reduction in the size of the resulting maps
  - They will shrink by a factor equal to the stride
- This can happen at any layer

Stride (2)



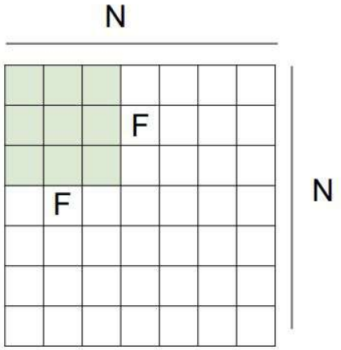
- Stride: 1
- output dimension : 5×5

Stride (3)



- stride : 2
- output dimension : 3×3

Stride (4)

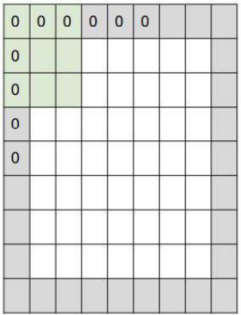


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

- We can't choose any stride blindly!
- The output dimensions must be integers.
- This is why padding comes into the picture.

Padding (1)



In practice, it is common to zero-pad the border. **Recall:** The original formula for convolution without padding:

$$\frac{N-F}{\text{stride}}+1$$

Now, we should adjust the formula considering padding P:

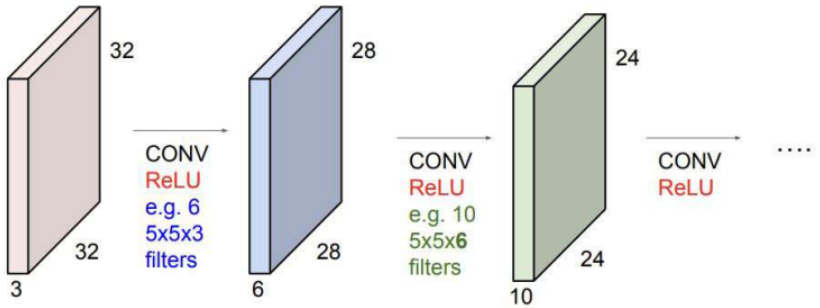
$$\frac{N+2P-F}{\text{stride}}+1$$

Padding (2)

- Zero-padding is used not only for stride > 1, but also to prevent a reduction in output size even when $S = 1$.
- For stride > 1, zero padding is adjusted to ensure that the size of the convolved output is $\left\lceil \frac{N}{S} \right\rceil$

  This is achieved by zero padding the image with $P = S \left\lceil \frac{N}{S} \right\rceil - N$
- For an F width filter:
  - **Odd** F: Pad on both left and right with $\frac{F-1}{2}$ columns of zeros
  - **Even** F: Pad one side with $\frac{F}{2}$ columns of zeros, and the other with $\frac{F}{2} - 1$ columns of zeros
  - The resulting image is width $N + F - 1$
- The top/bottom zero padding follows the same rules to maintain map height after convolution

Convnet (1)



- **ConvNet:** sequence of convolution layers, interspersed with activation functions.

Convnet (2)

Question: What do convolutional filters at different levels of a ConvNet learn?

Answer:

- Filters in the early layers typically detect simple features, such as edges, textures, and basic shapes.
- As we move deeper into the network, the filters learn more complex and abstract features, such as specific parts of objects (e.g., a nose, eyes, or other high-level patterns).

Example-1

Input volume: 32x32x3
10 5x5 filters with stride 1, pad 2

Output volume size:

$$\left(\frac{32+2\times 2-5}{1}\right)+1=32 \text{ spatially, so } 32x32x10$$

Example-2

Input volume: 32x32x3
10 5x5 filters with stride 1, pad 2

How many parameters are in this layer?
Each filter has 5*5*3 + 1 = 76 params *(+1 for bias)*
⇒ 76*10 = **760**

Parameter setting (1)

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyper-parameters:
  - Number of filters K
  - Their spatial extent F
  - The stride S
  - The amount of zero padding P
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = \left(\frac{W_1 - F + 2P}{S}\right) + 1$
  - $H_2 = \left(\frac{H_1 - F + 2P}{S}\right) + 1$
  - $D_2 = K$

Common settings:

- K = powers of 2 (e.g., 32, 64, ...)
- $F = 3,\ S = 1,\ P = 1$
- $F = 5,\ S = 1,\ P = 2$
- $F = 5,\ S = 2,\ P =$ whatever fits!
- $F = 1,\ S = 1,\ P = 0$

Parameter setting (2)

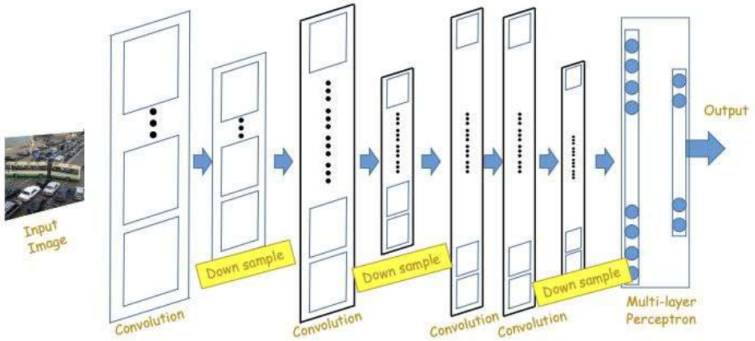
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by the d-th bias.

Review

Three Main Types of Layers

- **Convolutional Layer**
  - Output of neurons are connected to local regions in the input.
  - Applying the same filter across the entire image.
  - CONV layers parameters consist of a set of learnable filters.
- **Pooling Layer**
  - Performs a downsampling operation along the spatial dimensions.
- **Fully-Connected Layer**
  - Typically used in the final stages of the network to combine high-level features and make predictions.

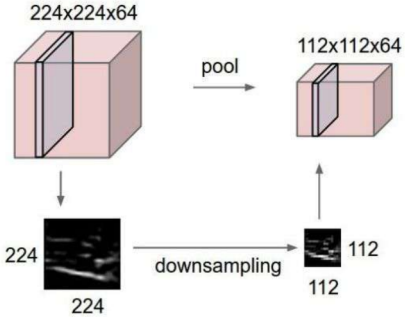
Pooling (1)



Convolution & activation layers are followed intermittently by pooling layers.

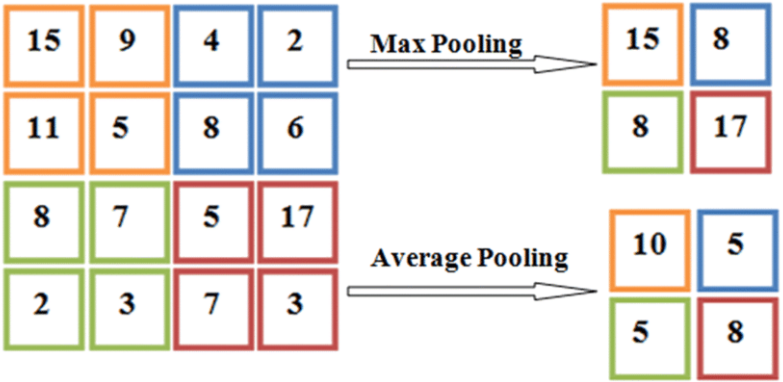
- Often, they alternate with convolution, though this is not necessary.

Pooling (2)



- Reduce the spatial size of the representation.
  - To reduce the number of parameters and computational demands within the network.
  - To control overfitting.
- Help the network to become invariant to small translations or distortions.
- Operates over each activation map independently.

Pooling (3)



Two Primary Types of Pooling:

- **Max Pooling:** Selects the maximum value from each patch of the feature map.
- **Average Pooling:** Computes the average of the values in each patch of the feature map.

Parameter setting (1)

Question:
How does a pooling layer affect the dimensions of the input image?

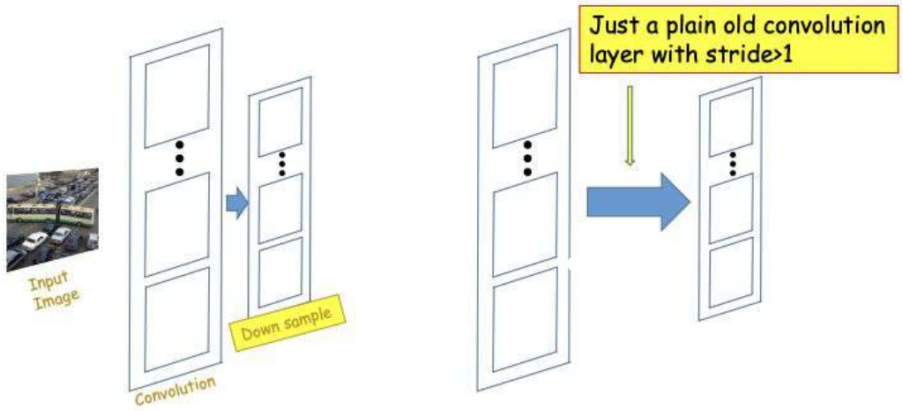
Answer:

- An $N \times N$ picture compressed by a $P \times P$ pooling filter with stride S results in an output map of side $\left\lceil \frac{(N-P)}{S} \right\rceil + 1$.
- Typically, pooling layers do not use zero-padding.

Parameter setting (2)

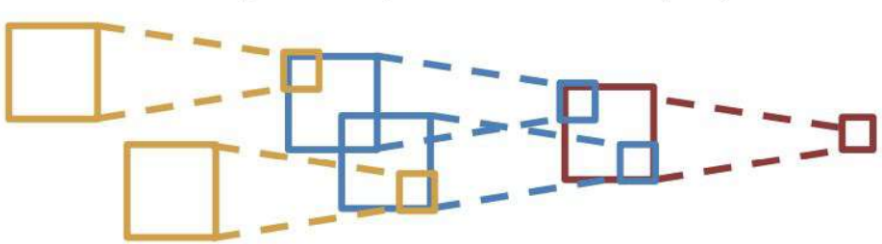
- Pooling accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - Their spatial extent F,
  - The stride S.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = \frac{(W_1 - F)}{S} + 1$
  - $H_2 = \frac{(H_1 - F)}{S} + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input.
- It is uncommon to use zero-padding for pooling layers.

Alternative to pooling



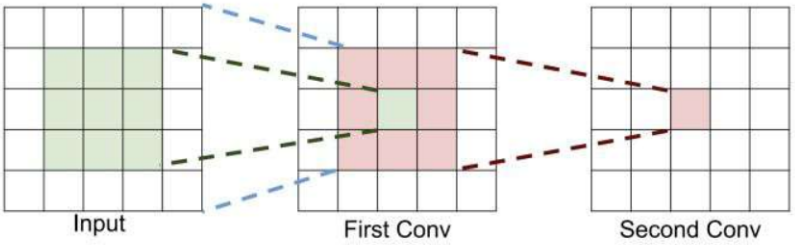
Downsampling be done by a simple convolution layer with stride larger than 1,
Replacing the max pooling layer with a convolution layer.

Receptive field (1)



- **Receptive Field :** How big of a region in the input or previous layer does a neuron on the n-th conv-layer see?
- For convolution with kernel size K, each element in the next layer depends on a $K \times K$ **receptive field** in the previous layer.

Receptive field (2)



- Units in the deeper layers can be indirectly connected to most of the input image.
- Each successive convolution adds $K-1$ to the receptive field size. With L layers, the receptive field size is $1+L\cdot(K-1)$.
- **Problem:** For large images, we need many layers for each output to "see" the whole image.
  - **Solution:** Downsample inside the network using strides and pooling.

Power of small filters

Suppose the input is $H \times W \times C$ and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W).

one CONV with 7 x 7 filters

Number of weights
$= C \times (7 \times 7 \times C) = \mathbf{49}C^2$

three CONV with 3 x 3 filters

Number of weights
$= 3 \times C \times (3 \times 3 \times C) = \mathbf{27}C^2$

Both options achieve a receptive field of 7; however, using multiple smaller filters reduces the number of parameters, introduces more nonlinearity, and generally leads to a more efficient, expressive model.

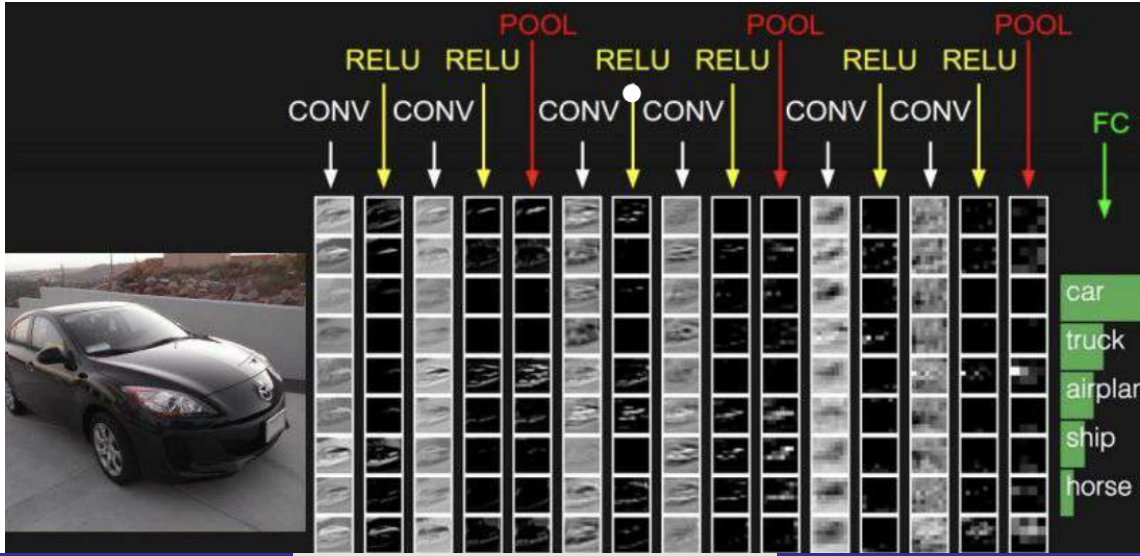
Inductive bias in CNNS

Inductive Bias:
Refers to the assumptions a model incorporates to generalize from training data to unseen data.

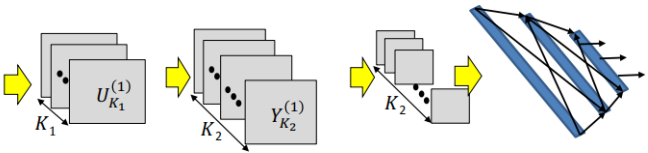
Key Features of Inductive Bias in CNNs:

- **Weight Sharing:**
  A single filter is applied across different regions of the input, significantly reducing the number of parameters.
- **Locality:**
  CNNs use small filters (e.g., 3×3) that focus on local regions, aligning well with image data where local structures are important.
- CNNs are more sample-efficient than FCNs due to their inductive biases.
- CNNS reduce sample complexity from $O(d^2)$ in FCNs to $O(\log^2 d)$.

Summary of concepts

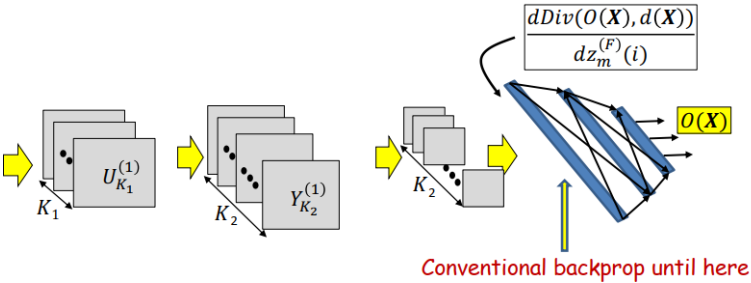


Training (1)



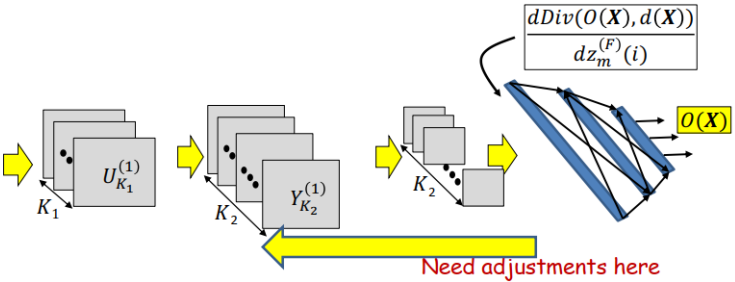
- Training is as in the case of the regular MLP
- Training examples of (Image, class) are provided
- The difference between the desired and actual network output for any input
- **Network parameters are optimized using gradient descent variants**
- **Gradients are computed through backpropagation**

Training (2)



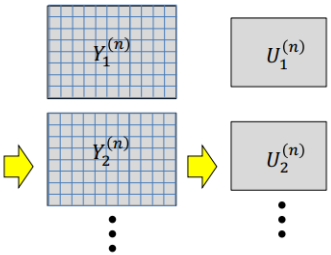
- Backpropagation continues in the usual manner until the computation of the derivative of the divergence w.r.t the inputs to the first "flat" layer

Training (3)



- Backpropagation from the flat MLP requires special consideration of:
  - The pooling layers, particularly Maxout
  - The shared computation in the convolution layers

Backpropagation maxout (1)

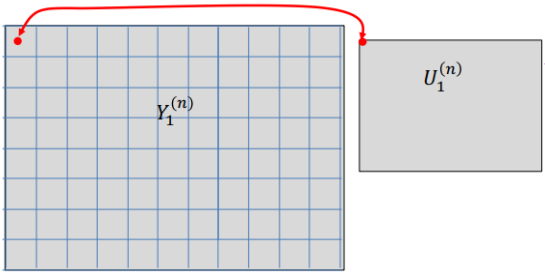


$$P_m^{(n)}(i,j) = \underset{\substack{k\in\{(i-1)d+1,\,id\},\\ l\in\{(j-1)d+1,\,jd\}}}{\operatorname{argmax}} Y_m^{(n)}(k,l)$$

$$U_m^{(n)}(i,j) = Y_m^{(1)}(P_m^{(n)}(i,j))$$

- The derivative w.r.t $U_m^{(n)}(i,j)$ can be computed via backprop.
- But this cannot be propagated backwards to compute the derivative w.r.t. $Y_m^{(n)}(k,l)$.
- **Max and argmax are not differentiable.**

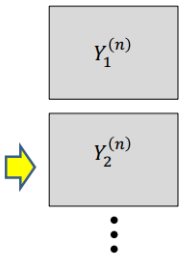
Backpropagation maxout (2)



- $$\frac{dDiv(O(X),d(X))}{dY_m^{(n)}(k,l)} = \begin{cases} \frac{dDiv(O(X),d(X))}{dU_m^{(n)}(i,j)} & \text{if } (k,l) = P_m^{(n)}(i,j) \\ 0 & \text{otherwise} \end{cases}$$

- **Approximation:** Derivative w.r.t the Y terms that did not contribute to the maxout map is 0.

Backpropagation weights

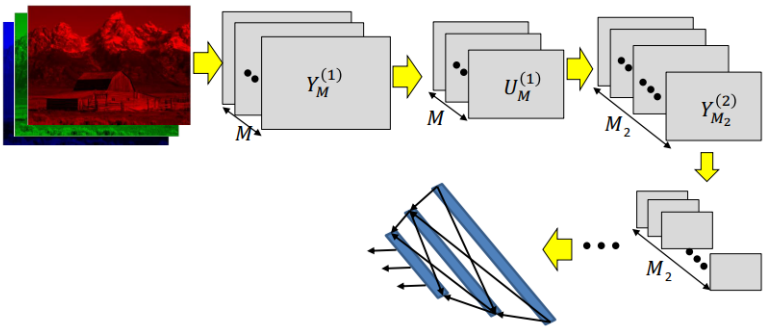


$$z_m^{(n)}(i,j) = \sum_{r=1}^{M_{n-1}} \sum_{k=1}^{L_n} \sum_{l=1}^{L_n} w_m^{(n)}(r,k,l) U_r^{(n-1)}(i+k, j+l)$$

$$Y_m^{(n)}(i,j) = f\left(z_m^{(n)}(i,j)\right)$$

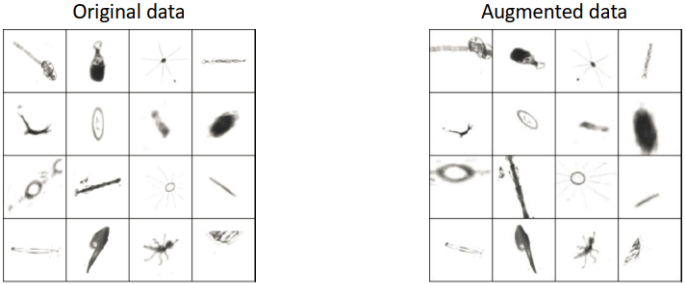
- **Note:** each weight contributes to *every* position in the map at the output of the convolutional layer.
- **Every position will contribute to the derivative of the weight**
  - Updating shared parameters

Learning the network



- Derived the divergence concerning every intermediate output and free parameter (filter weights).
- Can now be embedded in the gradient descent framework to learn the network.

A problem



Question: How can generalization be improved? **Answer:** Rotation, Translation, Rescaling, Flipping, etc.

Design choices

- **Number of convolutional and downsampling layers**
  - Arrangement (order in which they follow each other)
- **For each convolutional layer:**
  - Number of filters d^l
  - Spatial extent of filter $F^l \times F^l$
    - The "depth" of the filter is fixed by the number of filters in the previous layer d^{l-1}
  - The stride S^l
- **For each downsampling or pooling layer:**
  - Spatial extent of filter $P^l \times P^l$
  - The stride S^l
- **For the final multi-layer perceptron (MLP):**
  - Number of layers, and number of neurons in each layer

Typical architecture for CNNS

- `[(CONV-RELU)*N  POOL?]*M  (FC-RELU)*K  SOFTMAX`
  - Where N is usually up to ~ 5
  - M is a large number
  - $0 \leq K \leq 2$

But recent advances such as ResNet and GoogleNet challenge this paradigm!

Conclusion

- Highly structured nature of image data
- Local correlations are important
- CNNs have local and shared connections
- CNNs introduce strong inductive biases
- To leverage the two-dimensional structure of image data and introduce inductive biases, four interrelated concepts can be used:
  - Hierarchy
  - Locality
  - Invariance

content...