CS-570 Computer Vision

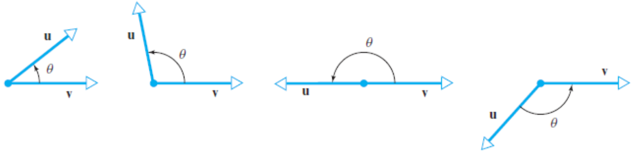
Nazar Khan

Department of Computer Science
University of the Punjab

11. Convolutional Neural Networks

A Neuron as a Detector

- A neuron can be viewed as a detector.
- When it fires, the input must have been similar to its weights.
  - Firing $\implies$ $\mathbf{w}^T\mathbf{x}$ was high $\implies$ $\mathbf{w}$ was similar to $\mathbf{x}$
- So neuron firing indicates detection of something similar to its weights.



$$\mathbf{u}^T\mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta$$

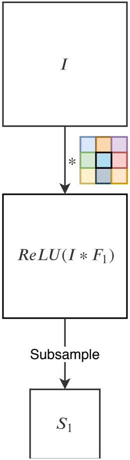
- Since $-1 \leq \cos\theta \leq 1$, $\mathbf{u}^T\mathbf{v}$ is highest when $\cos\theta = 1$
- That happens when $\theta = 0$
- That happens when vectors **u** and **v** point in the same direction.

Convolutional Neural Networks

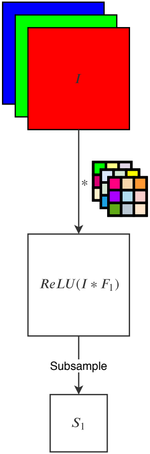
- Now we will look at networks that produce neuronal output via convolution.
- Known as Convolutional Neural Networks (CNNs).
- Most frequently used network architecture.
- Exploits local correlation of inputs.

Building Blocks of CNNs
Viewing convolution as neurons

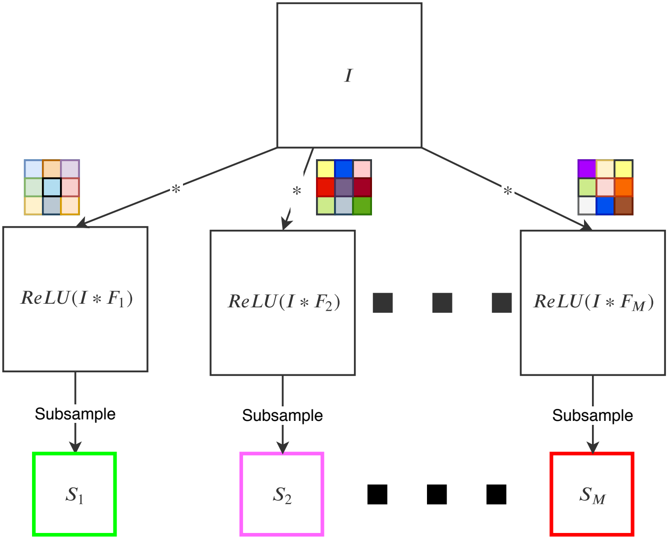
Single channel input



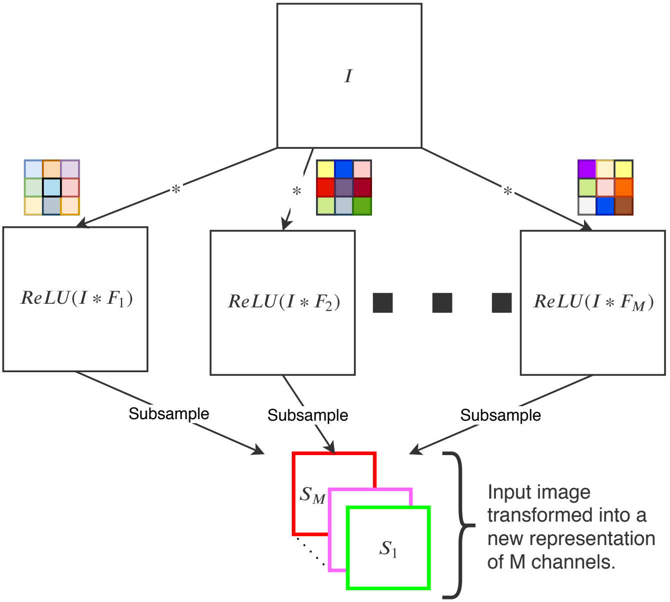
Multichannel input



Building blocks of CNNs

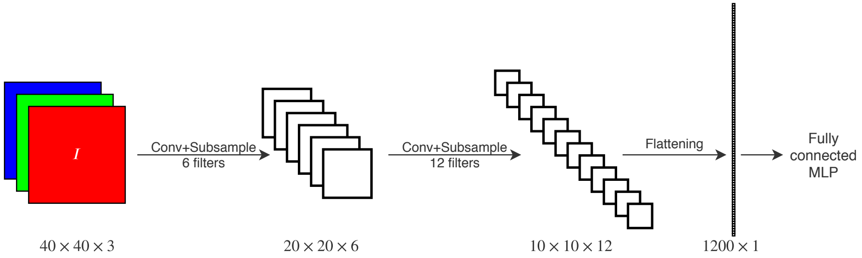


Building blocks of CNNs



CNN

- Convolution by M filters produces M channels.
- They represent an M-channel transformation of the input image I.
- This M-channel image can now be transformed further via additional convolution filters.
- Convolution-subsampling block can be repeated multiple times.
- $I \rightarrow M_1$ channels $\rightarrow M_2$ channels $\rightarrow \cdots \rightarrow M_b$ channels $\rightarrow$ flattening $\rightarrow$ MLP.



Convolutional layer

- Consists of multiple arrays of neurons. Each such array is called a *slice* or more accurately *feature map*.
- Each neuron in a feature map
  - is connected to only few neurons in the previous layer, but
  - uses the same weight values as all other neurons in that feature map.
- So within a feature map, we have both
  - local receptive fields, and
  - shared weights.

Convolutional layer

- Example: A feature map may have
  - 100 neurons placed in a 10×10 array, with
  - each neuron getting input from a 5×5 patch of neurons in the previous layer (receptive field), and
  - the same $26 (= 5 \times 5 + 1)$ weights shared between these 100 neurons.
- **Viewed as detectors, all 100 neurons detect the same 5×5 pattern but at different locations of the previous layer**.
- Different feature maps will learn[1] to detect different kinds of patterns.
  - For example, one feature map might learn to detect horizontal edges while others might learn to detect vertical or diagonal edges and so on.

[1] based on their learned weights

Convolutional layer

- To compute activations of the 100 neurons, a dot-product is computed between the same shared weights and different 5×5 patches of previous layer neurons.
- This is equivalent to **sliding a window of weights over the previous layer and computing the dot-product at each location of the window**.
- Therefore, activations of the feature map neurons are computed via *convolution* of the previous layer with a *kernel* comprising the shared weights. Hence the name of this layer.

Subsampling layer

- Reduces the spatial dimensions of the previous layer by downsampling. Also called *pooling* layer.
- Example: downsampling previous layer of $n \times n$ neurons by factor 2 yields a pooled layer of $\frac{n}{2} \times \frac{n}{2}$ neurons.
- No adjustable weights. Just a fixed downsampling procedure.
- Reduces computations in subsequent layers.
- Reduces number of weights in subsequent layers. This reduces overfitting.

Subsampling

- Options: From non-overlapping 2×2 patches
  - pick top-left (standard downsampling by factor 2)
  - pick average (*mean-pooling*)
  - pick maximum (*max-pooling*)
  - pick randomly (*stochastic-pooling*)
- Fractional max-pooling: pick pooling region randomly.

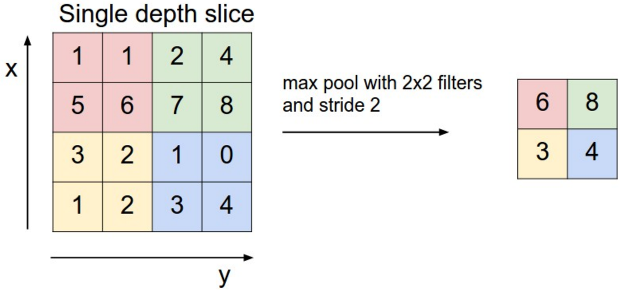


Figure: Max-pooling with 2×2 receptive fields, and stride of 2 neurons. Source: http://cs231n.github.io/convolutional-networks/

Subsampling

- The options in the last slide discard 75% of the data.
- They correspond to
  - neurons with 2×2 receptive fields, and
  - *stride* of 2 neurons.
- This is the most commonly used configuration. Other options exist but note that pooling with larger receptive fields discards too much data.
- Subsampling layer can be skipped if convolution layers uses stride>1 since that also produces a subsampled output.

Subsampling

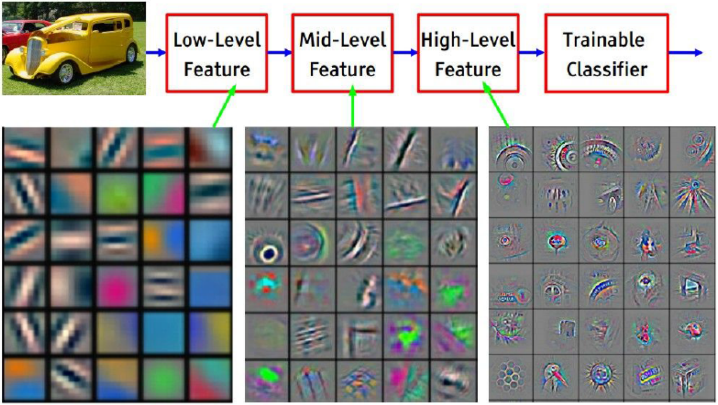
A pooling layer

- with $F \times F$ receptive field and stride S,
- "looking at" a $W_1 \times H_1 \times D_1$ input volume,
- produces a $W_2 \times H_2 \times D_2$ output volume, where
  - $W_2 = \frac{W_1 - F}{S} + 1$
  - $H_2 = \frac{H_1 - F}{S} + 1$
  - $D_2 = D_1$.

Fully Connected Layers

- After flattening, fully connected layers(s) can be used.
- The last layer has
  - neurons equal to the desired output size, and
  - activation functions based on the problem to be solved.
- The flattened layer can therefore be viewed as a transformation $\phi(\mathbf{x})$ that is fed into a sub-network of fully connected layers.
- Similarly, outputs of earlier layers are *intermediate representations* of the input.

Intermediate Representations

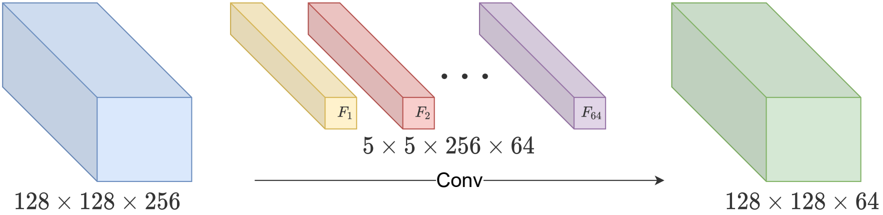


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Intermediate feature representations. Early layers form simple, low-level representations of the input. They are used to incrementally form more complex, high-level representations.

Source: `http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf`

Cost of Convolution Layer

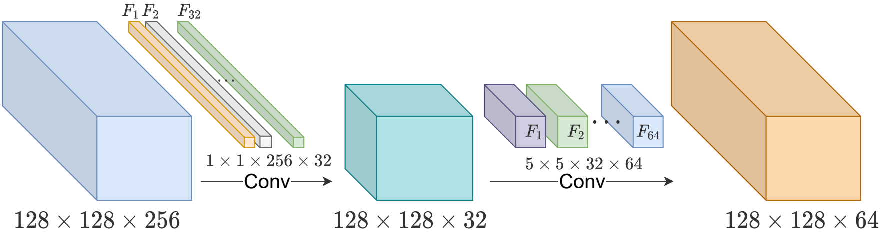


$$\text{Cost} = \#\text{ multiplications} = \underbrace{(128 \times 128 \times 64)}_{\text{Output neurons}} \times \underbrace{(5 \times 5 \times 256)}_{\text{Cost per neuron}}$$

$$= 6710886400$$

$$= 6.7 \text{ billion}$$

1×1 convolution



$$\text{Cost} = \left(\underbrace{(128 \times 128 \times 32)}_{\text{Output neurons}} \times \underbrace{(1 \times 1 \times 256)}_{\text{Cost per neuron}}\right) + \left(\underbrace{(128 \times 128 \times 64)}_{\text{Output neurons}} \times \underbrace{(5 \times 5 \times 32)}_{\text{Cost per neuron}}\right)$$

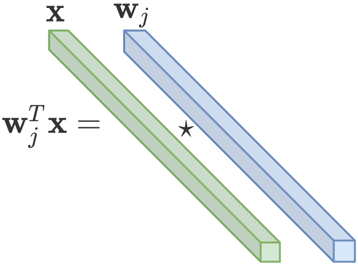
$$= 134217728 + 838860800$$

$$= 973078528 = 0.97 \text{ billion}$$

Almost 7 times reduction in number of multiplications to produce output volume of the same size.

1×1 convolution

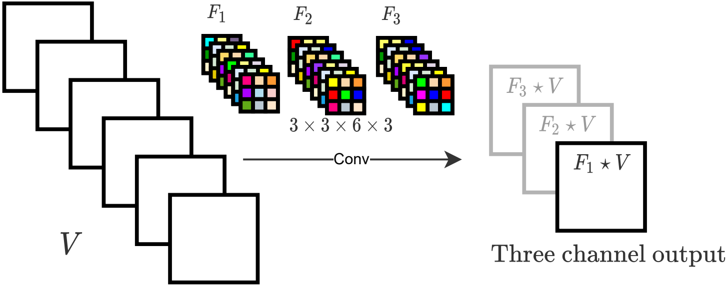
- A 1×1 convolution is just a linear combination of the input channels.
- The fully connected layer of a traditional MLP can also be represented via 1×1 convolutions.



Depthwise Separable Convolution

What happens in standard convolution?

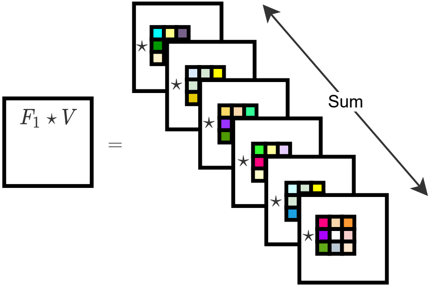
Consider the case of standard convolution using 3 filters.



Number of weights to produce 3 channel output $= 3 \times 3 \times 6 \times 3 = 162$.

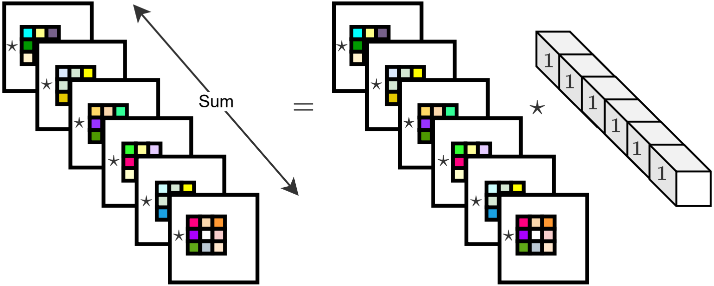
Depthwise Separable Convolution
What happens in standard convolution?

The first output channel is produced by 6 channel-wise convolutions that are then added together.



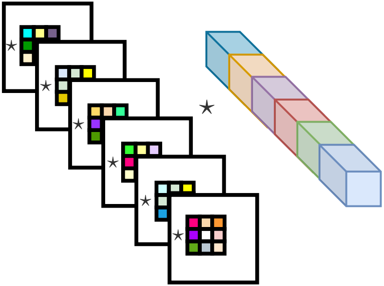
Depthwise Separable Convolution
What happens in standard convolution?

Summation of per-channel results corresponds to 1×1 convolution with a volume of 1s.



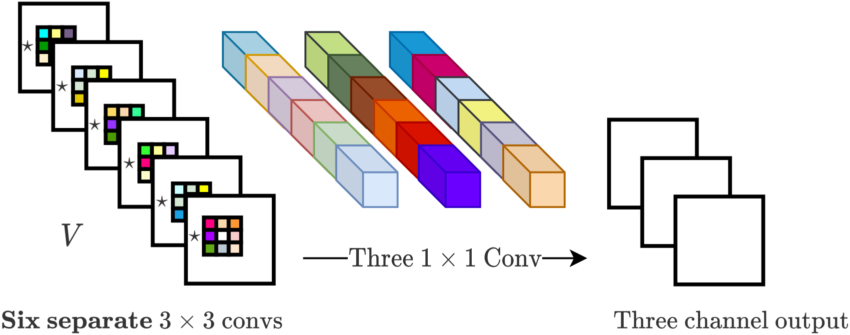
Depthwise Separable Convolution

Replace sum by a linear combination. This is called a *depthwise separable convolution*.



Depthwise Separable Convolution

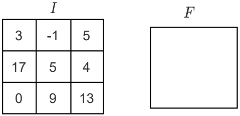
Multiple linear combinations lead to multiple output channels.



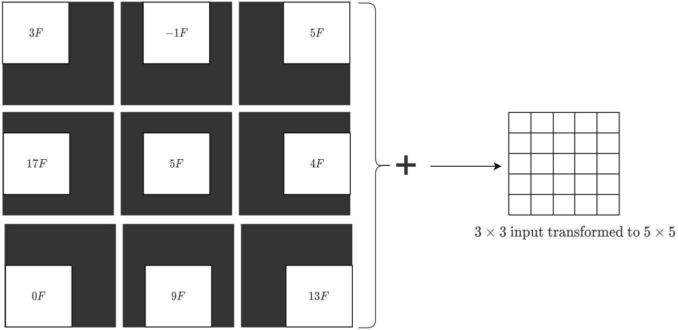
Number of weights to produce 3 channel output $= (3 \times 3 \times 6) + (6 \times 3) = 72$.

Expensive convolution (excluding the summation) is performed only once.
Multiple channels are produced via cheap 1×1 convolution.

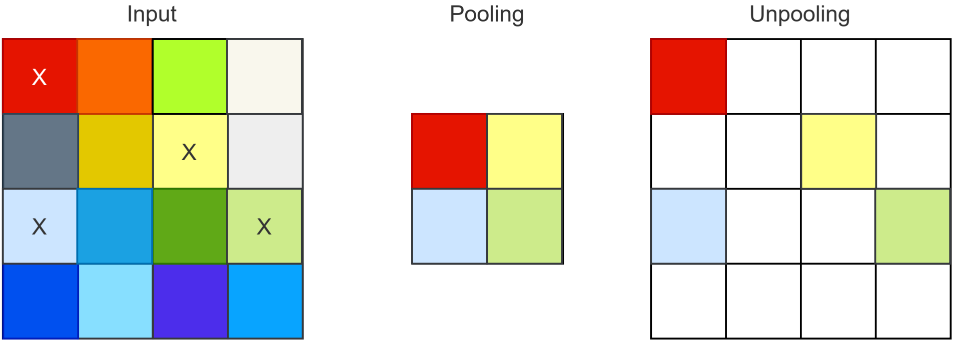
Transposed Convolution



A *transposed convolution* superimposes copies of the filter F scaled by the values in input I. Can be used to increase size.



Unpooling



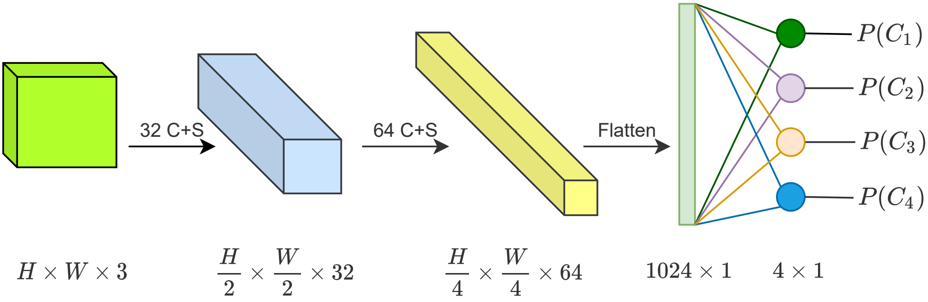
4×4 2×2 4×4

Reverses the size reduction effect of subsampling.

Fully Convolutional Networks (FCN)

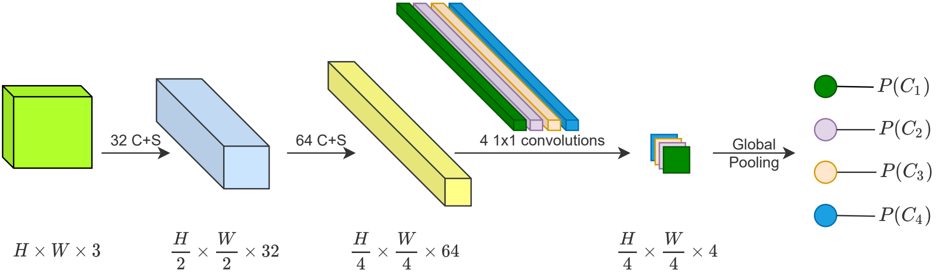
- An architecture for semantic segmentation.
- Only locally connected layers: convolution, pooling and upsampling.
- No fully connected layers (fewer parameters, faster training).
- Input image can be of any size.

The problem with fully connected layers



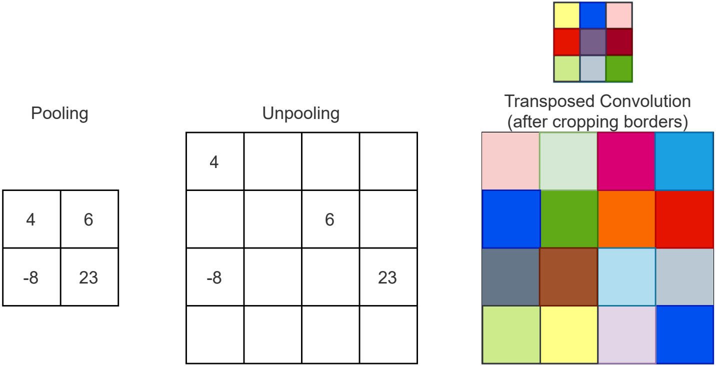
- K-class classification of an input image requires K softmax neurons at the output.
- 1024 neurons in fully connected layer imply that $H \times W$ must equal 256.
- So this can work with images of a certain size.

Fully Convolutional Networks



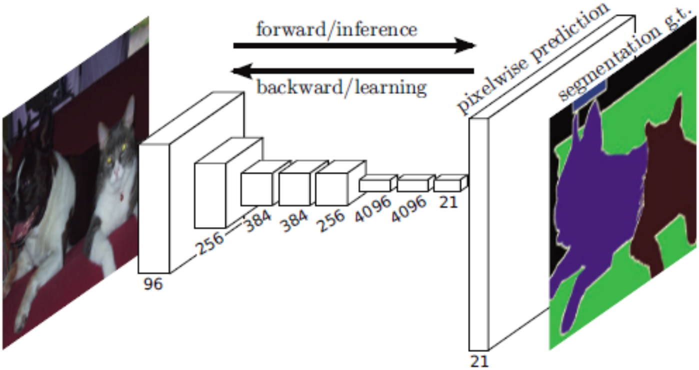
- K 1×1 convolutions corresponding to K classes.
- Followed by global pooling in each of the K channels.
- Followed by softmax.
- *Can work with images of any size.*

Image Generation via CNN



- Subsampled 2×2 result unpooled to a sparse 4×4 result that is then filled in via transposed convolution.
- Repeatedly upsample to obtain output of the same size as input.
- To generate images, use identity function at output.
- To generate pixel labels, use sigmoid or softmax.

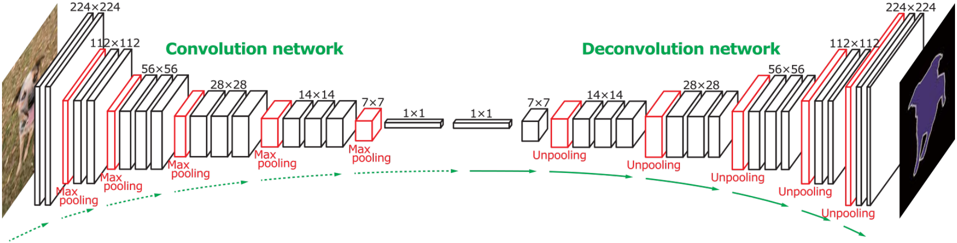
FCN for Semantic Segmentation[2]



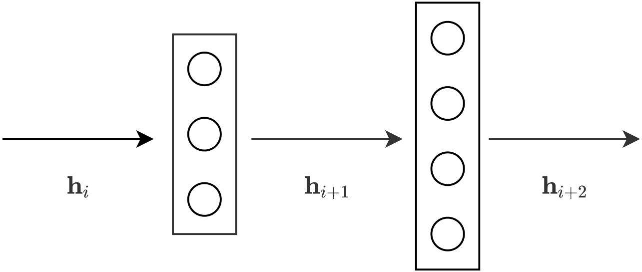
Each output pixel belongs to one of 21 classes.

[2]Segment image regions corresponding to different objects and find class of each object as well.

DeconvNet for Semantic Segmentation

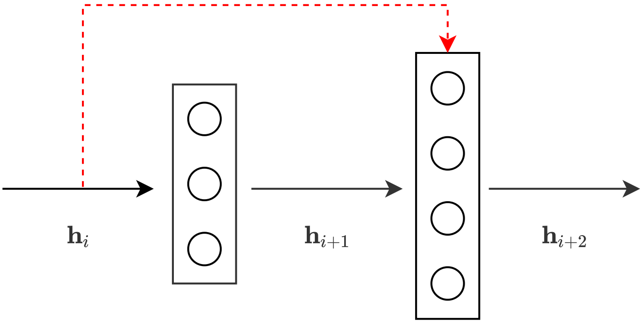


Residual Block



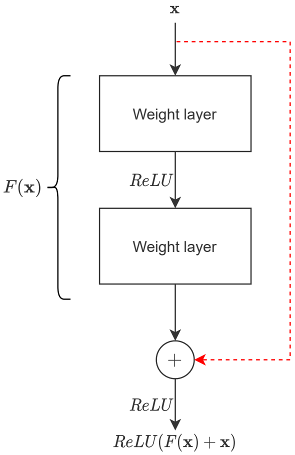
Standard propagation through two layers.

Residual Block



Skip connection between two layers.

Residual Block



If $F(\mathbf{x})$ approaches zero for any reason (e.g. due to weight regularization), the original input $\mathbf{x}$ can still be carried through.

Summary

- CNNs exploit local correlation of images.
- 1×1 convolutions reduce computations and can be used to control number of channels.
- Depthwise separable convolutions also reduce parameters and computations.
- Unpooling and transposed convolution can reverse the effects of subsampling to generate "images".
- Fully convolutional networks can operate on images of arbitrary size. No fully connected layers.
- Residual blocks avoid vanishing gradients.