Advanced Machine Learning (Semester 1 2023)

Convolutional Neural Networks

Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

May 22, 2023

Recap

Workflow for Developing Neural Network Applications

1. understand and define the problem:
   - what are the inputs?
   - what are the expected outputs? (classification (binary, multiclass, multilabel); regression)
   - are inputs sufficently informative to determine outputs?
2. define the data set
   - prepare data: normalization, PCA
3. choose a metric to measure success
4. Define evaluation methodology
   - e.g. cross-validation: holdout if plenty of data, otherwise K-fold
5. develop mode
   - last-layer activation function
   - loss function
   - optimization
6. Train model to point of overfitting
7. Tune the model: regularization
8. Test the model
9. If sufficient: apply the model

Recap

Software Frameworks

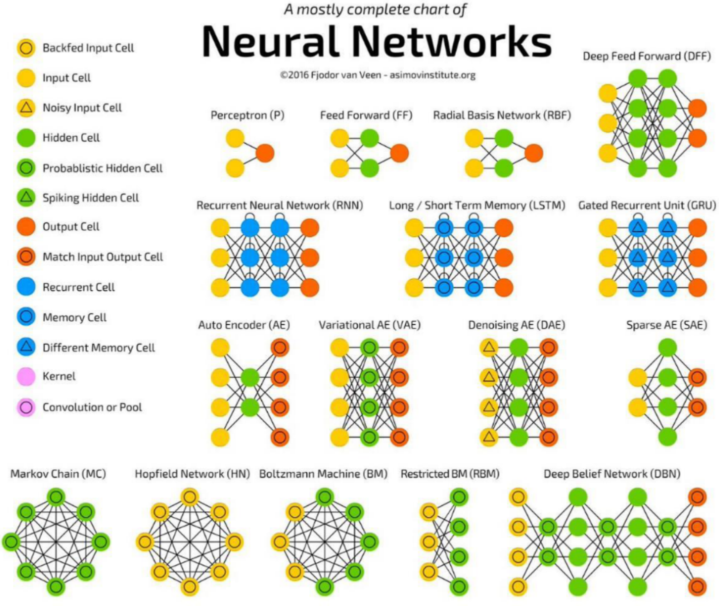
TensorFlow

astroNN

PyTorch

Keras

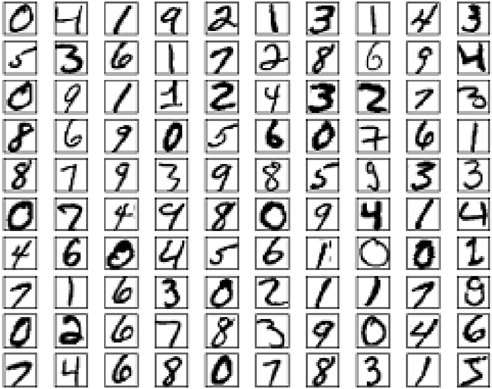
Neural Network Architectures



Sasen Cain (@spectralradius)

Neural Network Architectures

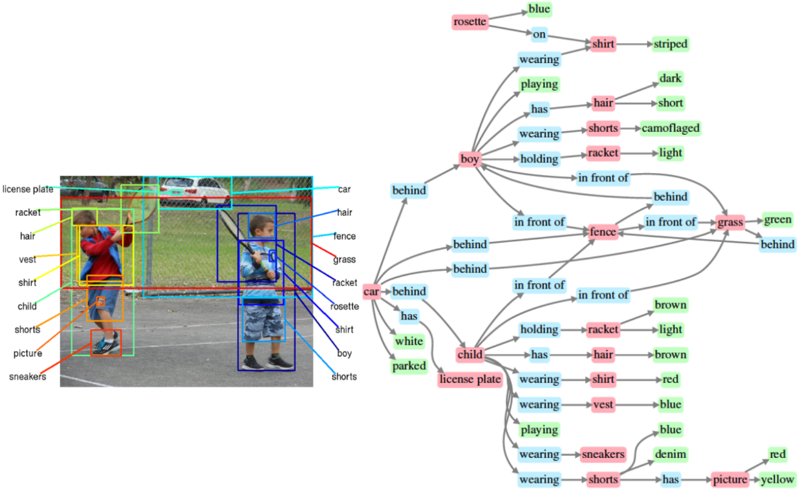
'classic' trained feed-forward neural networks can handle object recognition



A subset of the MNIST Handwritten Digits dataset.

Neural Network Architectures

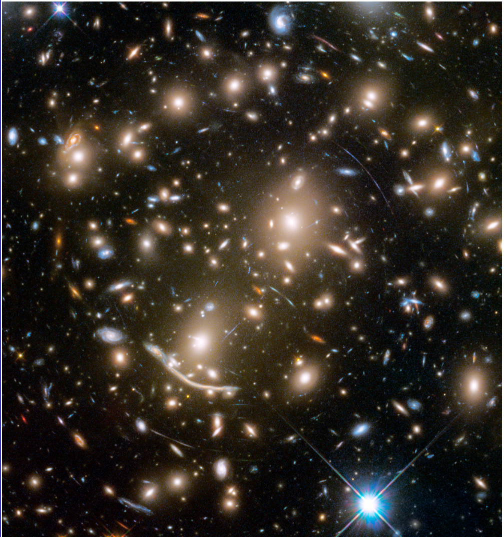
but we want not only object recognition...



Johnson *et al.*, “Image Retrieval using Scene Graphs”, CVPR 2015

Neural Network Architectures

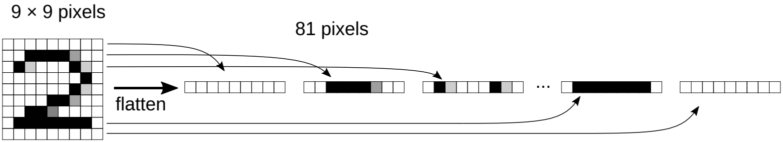
but we want not only object recognition...



interacting galaxies
galaxy clusters
blending
gravitational lenses
foreground stars
...

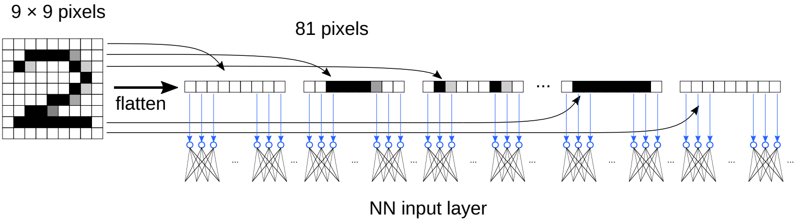
Neural Network Architectures

classic approach to **computer vision**:
treat image pixels as individual features to feed into the deep neural network (Multi-Layer Perceptron)



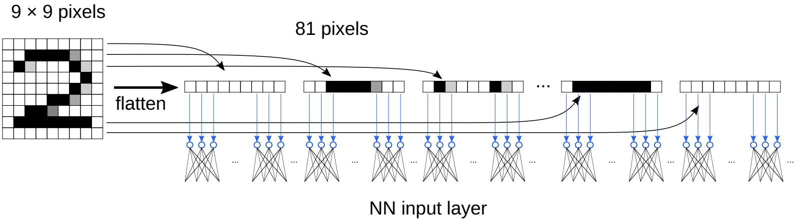
Neural Network Architectures

classic approach to **computer vision**:
treat image pixels as individual features to feed into the deep neural network (Multi-Layer Perceptron)



Neural Network Architectures

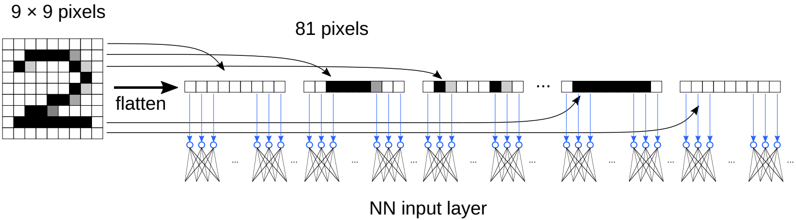
classic approach to **computer vision**:
treat image pixels as individual features to feed into the deep neural network (Multi-Layer Perceptron)



⇨ okay for simple and low-resolution images

Neural Network Architectures

classic approach to **computer vision**:
treat image pixels as individual features to feed into the deep neural network (Multi-Layer Perceptron)



⇨ flattening the image removes spatial information

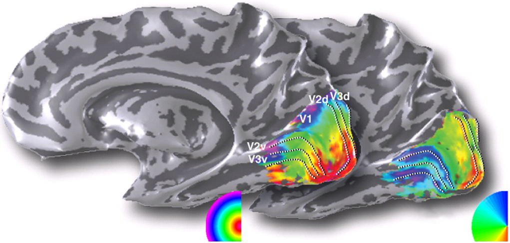
for complex and high-resolution dataset: Convolutional Neural Network (CNN) is the solution

Artificial Vision

Convolutional Neural Networks (CNN) were not invented overnight

topographical mapping of the visual cortex (in humans and animals):

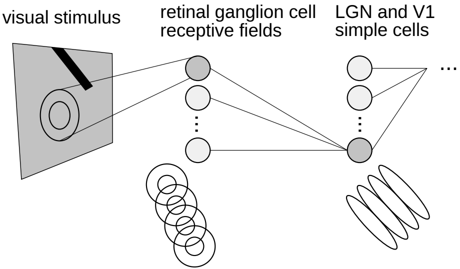
nearby cells in the cortex represent nearby regions in the visual field



credit: Wandell et al. (2007)

Artificial Vision

the visual cortex is organized hierarchically:



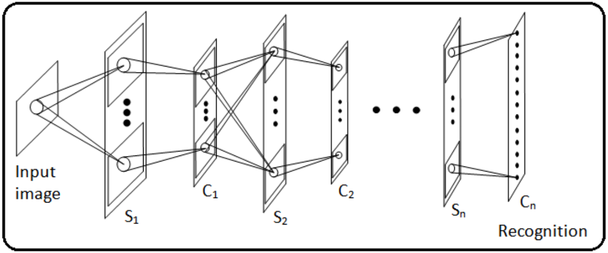
adapted from: Lane MJcIntosh, CS231n (2017)

simple cells: respond to light orientation
complex cells: respond to light orientation and movement
hypercomplex cells: respond to movement with an end point

Artificial Vision

Neocognitron (Fukushima 1980)

uses a "sandwich" architecture of simple cells and complex cells where simple cells contain modifiable parameters, complex cells perform pooling operation



⇨ the first CNN

Convolutional Neural Networks - The Idea

Convolutional neural networks (CNNs) are similar to feedforward neural networks: A convolutional neural network consists of an input layer, hidden layers for feature learning and an output layer for classification.
The difference lies in the operations performed by the hidden layers: In a CNN, the hidden layers include layers that perform convolutions.
These networks harness principles from linear algebra, particularly matrix multiplication, to **identify patterns within an image**.

Convolutional Neural Networks - The Idea

Convolutional neural networks (CNNs) are similar to feedforward neural networks: A convolutional neural network consists of an input layer, hidden layers for feature learning and an output layer for classification.
The difference lies in the operations performed by the hidden layers: In a CNN, the hidden layers include layers that perform convolutions.
These networks harness principles from linear algebra, particularly matrix multiplication, to **identify patterns within an image**.

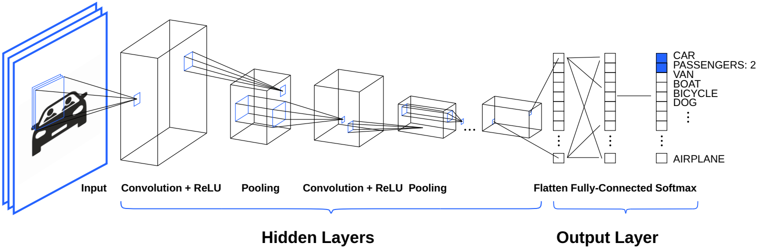
Basic underlying idea:
An image is a matrix of pixels. Important parts on the image (objects, edges) are within close distances. Individual neurons respond to input only in a restricted region, like in the *visual cortex* of the brain.

Recap: Tensors

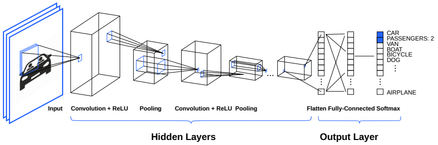
name	dimensionality (rank, order)	example
scalar	0	42
vector	1	$(6.6\ 8\ 2)$
matrix	2	$\begin{pmatrix} 1 & 2 & 3 \\ 55 & 0.6 & 1 \end{pmatrix}$
cube	3	$\begin{pmatrix} (6.7\ 7.5\ 2) & (4\ 8.4\ 5.1) & (55.7\ 1\ 2.5) \\ (7.5\ 3\ 2) & (3\ 8\ 9.1) & (6.5\ 9.4\ 2) \\ (2.7\ 2\ 2.5) & (1.4\ 8\ 1.5) & (6.2\ 8.3\ 2) \end{pmatrix}$

Convolutional Neural Networks - The Idea

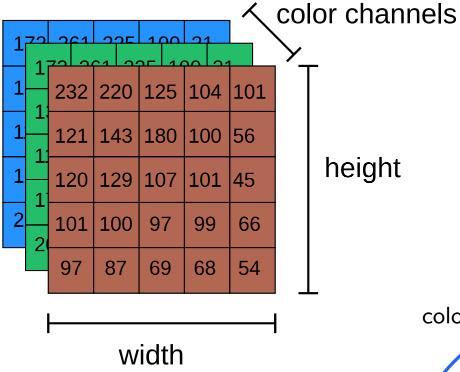
A typical CNN architecture consists of alternating **Convolutional layers** and **Pooling layers** that extract features. Finally, their output is flattened into a one-dimensional feature layer that is passed into a Multi-Layer Perceptron (fully-connected layers) to obtain a prediction.



The Input



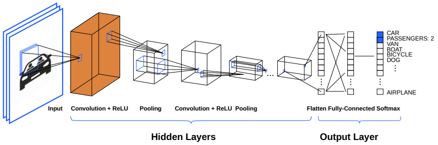
preserving the input image's spatial and temporal dependencies



color image movie

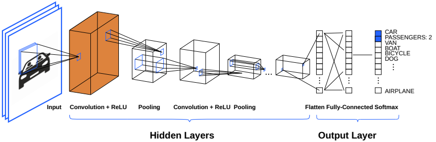
In a CNN, the input is a tensor with shape (input height) $\times$ (input width) $\times$ (input channels) $\times$ (number of inputs).

Convolution Layer



The objective of the Convolution Layer is to extract the high-level features, such as edges, from the input image.

Convolution Layer



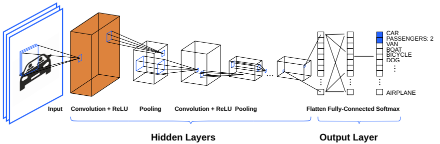
If you have used image/ photo editing, you likely have already applied convolutions, whether you realize it or not:

blurring/smoothing

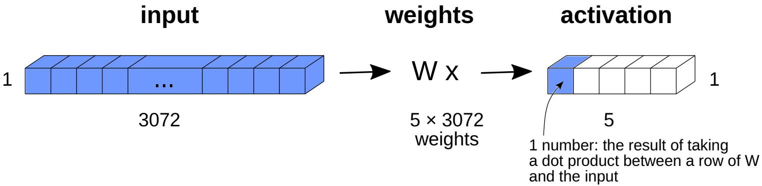
sharpening

edge detection

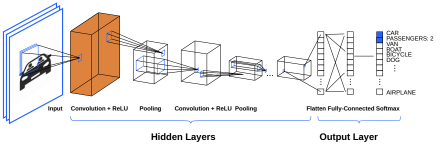
Convolution Layer



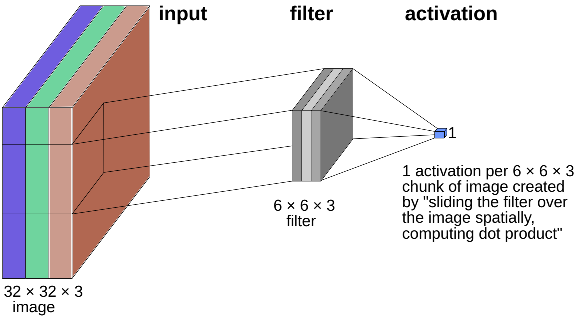
in feed-forward neural network:
a 32 $\times$ 32 pixel $\times$ 3 channels image is flattened



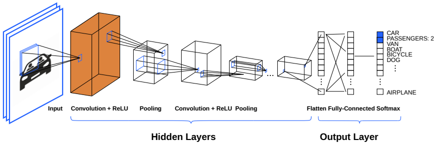
Convolution Layer



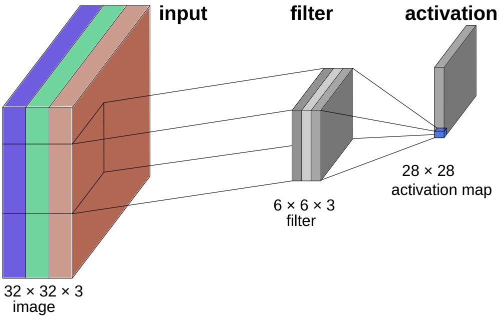
in a CNN:
processing a 32 $\times$ 32 pixel $\times$ 3 channels image as tensor



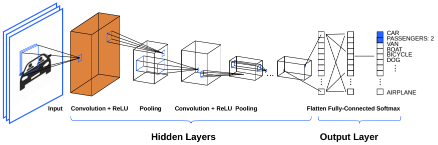
Convolution Layer



in a CNN:
processing a 32 $\times$ 32 pixel $\times$ 3 channels image as tensor



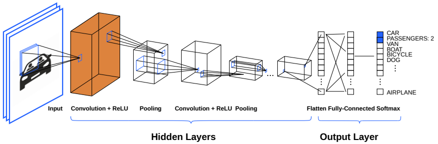
Convolution Layer



What happens internally:
convolution is an element-wise multiplication of two matrices followed by summation

The operation performed in CNN is technically a **cross-correlation**, not a convolution:

Convolution Layer



What happens internally:
convolution is an element-wise multiplication of two matrices followed by summation

The operation performed in CNN is technically a **cross-correlation**, not a convolution:

Convolution:

$$y_{i,j} = \sum_l \sum_k \overset{\text{input}}{x_{i-l,j-m}} \overset{\text{filter}}{w_{l,m}}$$

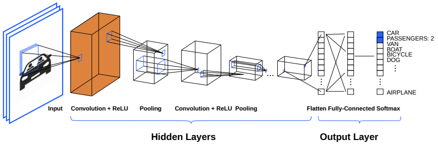
Cross-Correlation (as carried out here):

$$y_{i,j} = \sum_l \sum_k x_{i+l,j+m} w_{l,m}$$

the difference is for cross-correlation we don't have to ‘flip‘ the filter's kernel relative to the input

both can be interchanged through a simple rotation operation

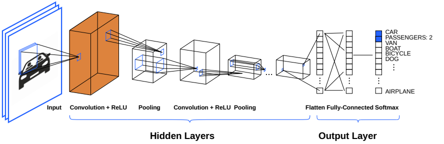
Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

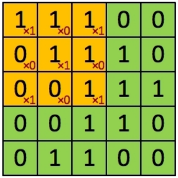
We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

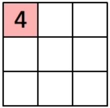
kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

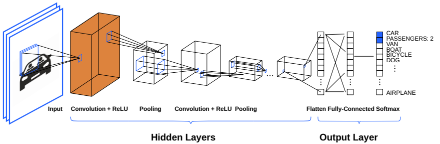
image



convolved feature



Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

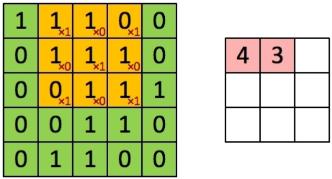
We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

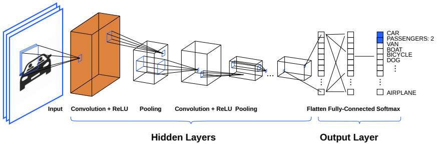
kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

image **convolved feature**



Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

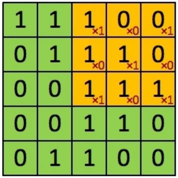
We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

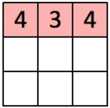
kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

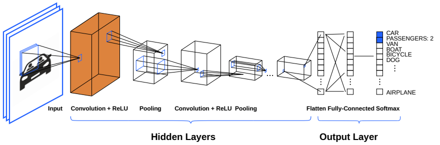
image



convolved feature



Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

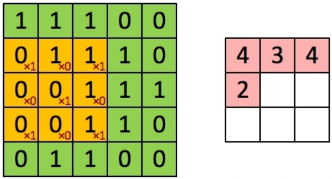
We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

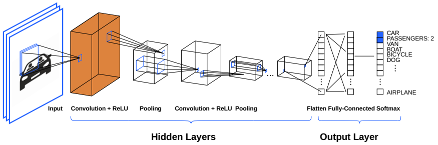
kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

image **convolved feature**



Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

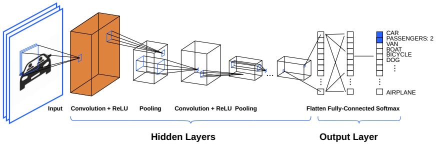
image

1	1	1	0	0
0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0
0	$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1
0	$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0
0	1	1	0	0

convolved feature

4	3	4
2	4	

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

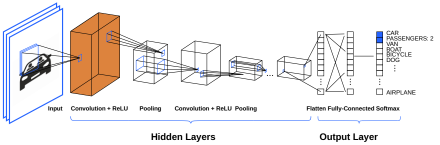
image

1	1	1	0	0
0	1	$1_{\times 1}$	$1_{\times 0}$	$0_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$
0	0	$1_{\times 1}$	$1_{\times 0}$	$0_{\times 1}$
0	1	1	0	0

convolved feature

4	3	4
2	4	3

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

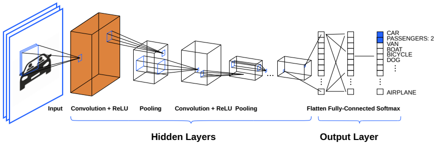
image

1	1	1	0	0
0	1	1	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
$0_{\times 0}$	$0_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0

convolved feature

4	3	4
2	4	3
2		

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

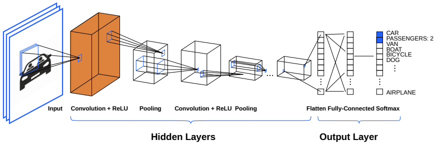
image

1	1	1	0	0
0	1	1	1	0
0	$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	1
0	$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	0
0	$1_{\times 1}$	$1_{\times 0}$	$0_{\times 1}$	0

convolved feature

4	3	4
2	4	3
2	3	

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a $5\times5\times1$ image with a $3\times3\times1$ kernel to get a $3\times3\times1$ feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

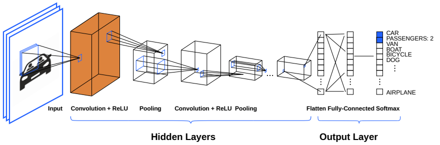
image

1	1	1	0	0
0	1	1	1	0
0	0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$0_{\times 0}$
0	1	$1_{\times 1}$	$0_{\times 0}$	$0_{\times 1}$

convolved feature

4	3	4
2	4	3
2	3	4

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

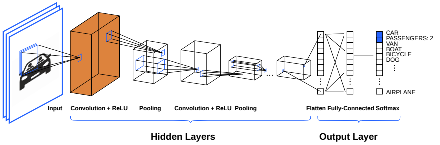
image

1	1	1	0	0
0	1	1	1	0
0	0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$0_{\times 0}$
0	1	$1_{\times 1}$	$0_{\times 0}$	$0_{\times 1}$

convolved feature

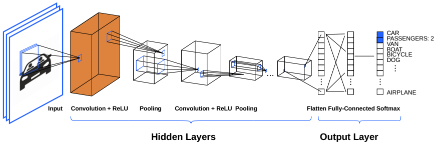
4	3	4
2	4	3
2	3	4

$\Rightarrow$ The kernel K shifts 9 times because of Stride Length $= 1$ (Non-Strided)



Design consideration: **The size of the convolution output.**

Convolution Layer

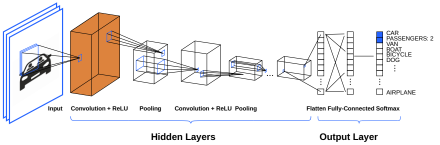


Design consideration: **The size of the convolution output.**

The stride of the convolution (how many pixel the filter jumps each time) is not necessarily 1 pixel.

The size of the convolution output depends on implementation factors and might not be identical to the input.

Convolution Layer



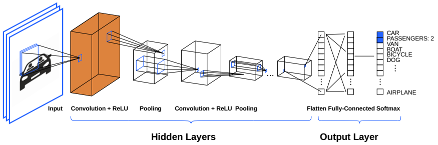
Design consideration: **The size of the convolution output.**

Image size: $I \times I$
Filter size: $F \times F$
Stride: S

the **floor** operator: the smallest integer less than or equal to the input

Output size (in each dimension): $\lfloor (I - F)/S \rfloor + 1$
assuming the filter is not allowed to go beyond the edge

Convolution Layer



Design consideration: **The size of the convolution output.**

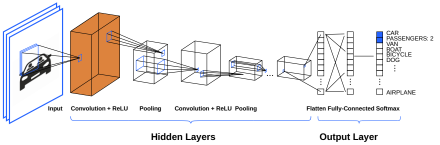
Image size: $I \times I$
Filter size: $F \times F$
Stride: S

the **floor** operator: the smallest integer less than or equal to the input

Output size (in each dimension): $\lfloor (I - F)/S \rfloor + 1$
assuming the filter is not allowed to go beyond the edge

The result is a reduction in the output size, even for S=1
If this is not acceptable: Use **padding**.

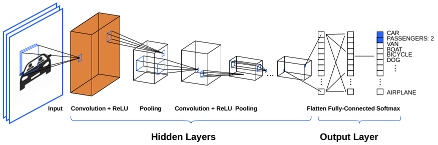
Convolution Layer



The solution: **Zero Padding**

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Convolution Layer



The solution: **Zero Padding**

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

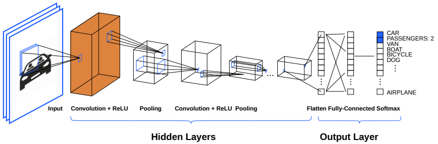
For an F width filter:

odd F: pad on left, right, top, bottom with a width of $(F-1)/2$ zeros
even F: pad one side and top with $F/2$ zeros, and the other side and bottom with a width of $L/2-1$ zeros

The resulting image width is $I+F-1$, the result of the convolution has width I

For a stride $S>1$, zero padding is adjusted to ensure that the result of the convolution has width $\lceil I/S \rceil$: first zero-padding the image with $S\lceil I/S \rceil - I$ zeros and then apply the rules above.

Convolution Layer

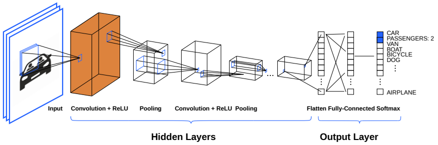


The **Computational Cost** of carrying out the convolution (cross-correlation) operation:

the cost of scanning an $I \times I$ image with a $F \times F$ filter is $\mathcal{O}(I^2 F^2)$: we have F^2 multiplications at each of I^2 image pixels in the equation

$$y_{i,j} = \sum_{l} \sum_{k} x_{i+l,j+m} \; w_{l,m}$$

Convolution Layer



The **Computational Cost** of carrying out the convolution (cross-correlation) operation:

the cost of scanning an $I \times I$ image with a $F \times F$ filter is $\mathcal{O}(I^2 F^2)$:
we have F^2 multiplications at each of I^2 image pixels in the equation

$$y_{i,j} = \sum_l \sum_k x_{i+l,j+m} \; w_{l,m}$$

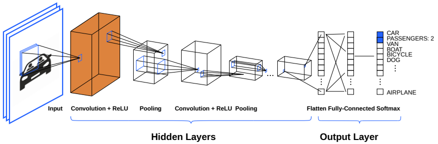
expensive for large filters

a solution: convolutions using **Discrete Fourier Transforms (DFTs)**:

$$Y = IDFT2(DFT2(X) \circ conj(DFT2(W)))$$

where IDFT2 is the **second order inverted discrete Fourier transform**
computational cost $\mathcal{O}(I^2 \log F)$

Convolution Layer



The **Computational Cost** of carrying out the convolution (cross-correlation) operation:

the cost of scanning an $I \times I$ image with a $F \times F$ filter is $\mathcal{O}(I^2 F^2)$: we have F^2 multiplications at each of I^2 image pixels in the equation

$$y_{i,j} = \sum_{l} \sum_{k} x_{i+l,j+m} \; w_{l,m}$$

⇨ expensive for large filters

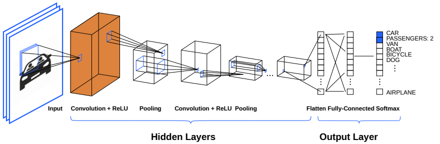
a solution: convolutions using **Discrete Fourier Transforms (DFTs)**:

$$Y = IDFT2(DFT2(X) \circ conj(DFT2(W)))$$

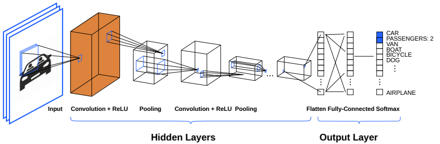
where IDFT2 is the **second order inverted discrete Fourier transform**
computational cost $\mathcal{O}(I^2 \log F)$

⇨ significant reduction in computational cost

Convolution Layer

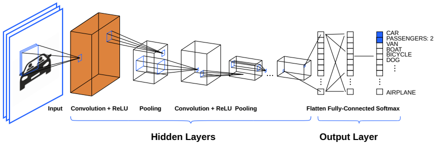


How to **set** the kernel values of the filters?



How to **set** the kernel values of the filters?

In the Convolutional Neural Network, these kernel values in the filter are **essentially weights to be learned and trained** to automatically capture spatial edge patterns in images.

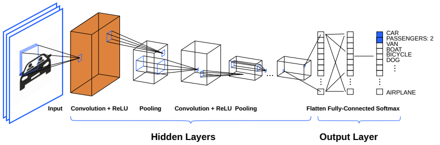


How to **set** the kernel values of the filters?

In the Convolutional Neural Network, these kernel values in the filter are **essentially weights to be learned and trained** to automatically capture spatial edge patterns in images.

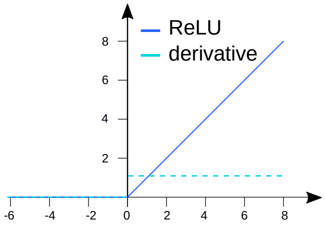
In addition to its ability to capture complex patterns, the small number of weights (filters are usually small relative to the images) in each layer makes the CNN very efficient to train.

Convolution Layer



After each convolution operation, a **Rectified Linear Unit (ReLU) transformation** is applied to the feature map, introducing nonlinearity to the model.
This is the activation function we have seen before.

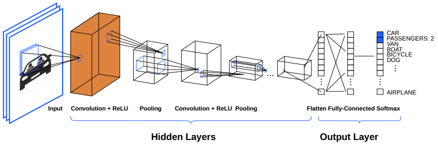
ReLU effectively removes negative values from an activation map by setting them to zero.



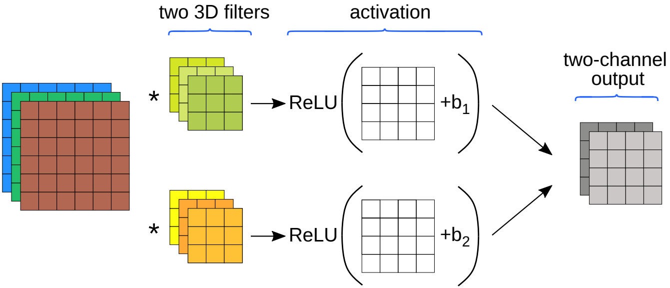
$$\varphi(z) = \max(0, z)$$

$$\varphi'(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

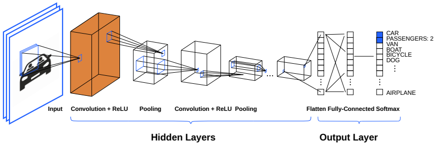
Convolution Layer



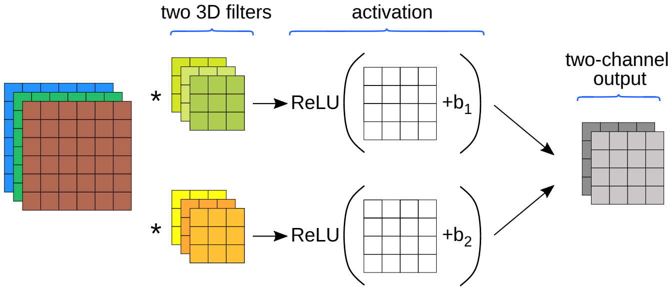
A hypothetical, simple Convolutional Layer:



Convolution Layer



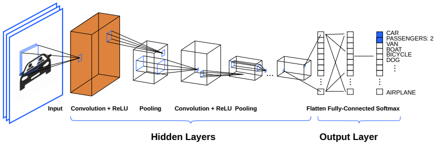
A hypothetical, simple Convolutional Layer:



The number of channels in a Convolutional Layer output equals the number of 3D (length $\times$ width $\times$ channels) filters used in the layer.

During the 3D convolutions, each output channel, as a 2D matrix, has additional bias broadcasted to each element, before a ReLU activation function is applied.

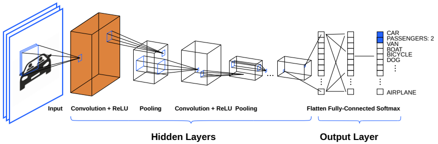
Convolution Layer



Three **hyperparameters** control the size of the output volume of the convolutional layer:

1. The **filter depth** affects the depth of the output. E.g.: three distinct filters would yield three different feature maps.

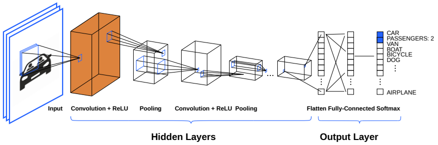
Convolution Layer



Three **hyperparameters** control the size of the output volume of the convolutional layer:

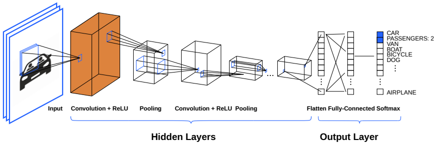
1. The **filter depth** affects the depth of the output. E.g.: three distinct filters would yield three different feature maps.
2. **Stride** is the number of pixels that the kernel moves over the input matrix. A larger stride means smaller overlap of receptive fields and smaller spatial dimensions of the output volume.

Convolution Layer



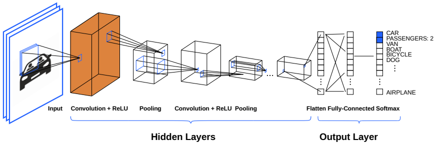
Three **hyperparameters** control the size of the output volume of the convolutional layer:

1. The **filter depth** affects the depth of the output. E.g.: three distinct filters would yield three different feature maps.
2. **Stride** is the number of pixels that the kernel moves over the input matrix. A larger stride means smaller overlap of receptive fields and smaller spatial dimensions of the output volume.
3. **Zero-padding** is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero.



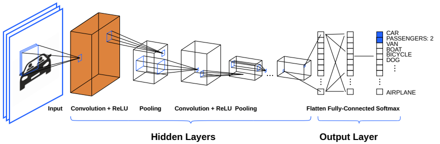
The total **number of parameters to be trained**:

Convolution Layer



The total **number of parameters to be trained**:

Considering if there are 10 filters that are $3 \times 3 \times 3$ in dimension, then the total number of parameters (weights and biases) in the Convolutional Layer would be: Total Parameters $= (3 \times 3 \times 3 + 1) \times 10 = 280$

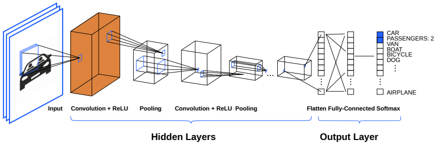


The total **number of parameters to be trained**:

Considering if there are 10 filters that are $3 \times 3 \times 3$ in dimension, then the total number of parameters (weights and biases) in the Convolutional Layer would be: Total Parameters $= (3 \times 3 \times 3 + 1) \times 10 = 280$

The total number of parameters usually increase in later Convolutional Layers, along with the typical increase in channel size and the number of filters.

Convolution Layer



The total **number of parameters to be trained**:

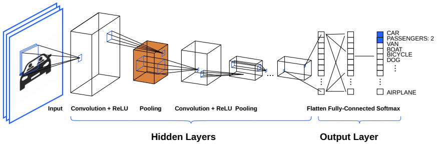
Considering if there are 10 filters that are $3 \times 3 \times 3$ in dimension, then the total number of parameters (weights and biases) in the Convolutional Layer would be: Total Parameters $= (3 \times 3 \times 3 + 1) \times 10 = 280$

The total number of parameters usually increase in later Convolutional Layers, along with the typical increase in channel size and the number of filters.

computing the total number of parameters in a layer:

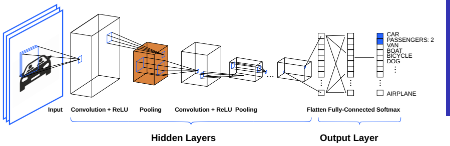
$$\left.\begin{array}{l} I \text{ input size} \\ F \text{ filter size} \\ P \text{ padding size} \\ S \text{ stride size} \\ n \text{ number of filters} \\ n_{prev} \text{ number of filters in previous layer} \end{array}\right\} \begin{array}{l} \text{output dimensions: } \left\lfloor \frac{I+2P-F}{S} + 1 \right\rfloor^2 \\ \text{filter dimension: } F^2 \times n_{prev} \\ \text{number of weights: } F^2 \times n_{prev} \times n \\ \text{number of biases } n \end{array}$$

Pooling Layer



Each Convolutional Layer is followed by a Pooling Layer.
Similar to the Convolution Layer, the Pooling Layer reduces the spatial size of the Convolved Feature. It suppresses noise and extracts dominant features.

Pooling Layer

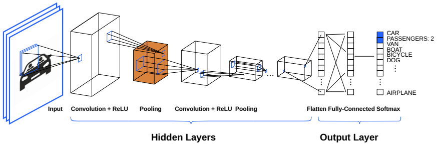


Each Convolutional Layer is followed by a Pooling Layer.
Similar to the Convolution Layer, the Pooling Layer reduces the spatial size of the Convolved Feature. It suppresses noise and extracts dominant features.

In detail, a pooling layer carries out the following tasks:

- Extract the most important (relevant) features by getting the maximum number or averaging the numbers.
- Reduce the dimensionality (number of pixels) of the output returned from previous convolutional layers.
- Reduce the number of parameters in the network.
- Remove any noise present in the features extracted by previous convolutional layers.
- Increase the accuracy of CNNs.

Pooling Layer



The pooling layer contains three elements: Feature Map, Filter and Pooled Feature Map.
The **Pooling Operation** happens between a section of the feature map and the filter. It outputs the pooled feature map which is a downsampled version of the feature map.

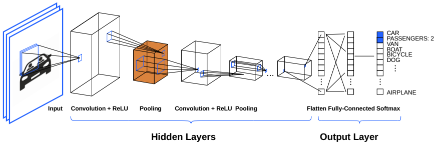
Example: Maximum Pooling here: with a 2×2 filter and $S = 2$

7	1	5	8
4	6	5	1
9	0	7	5
3	2	1	0

$\longrightarrow$

7	8
9	7

Pooling Layer



alternatives:

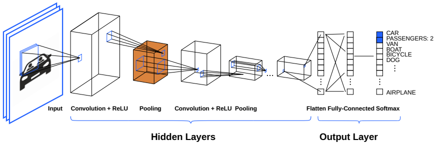
p-Norm Pooling
here: with a 2×2 filter and $S = 2$, $p = 5$

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

$$\xrightarrow{\sqrt[p]{\frac{1}{p^2}\sum_{i,j} x_{ij}^p}}$$

4.86	8
2.38	3.16

Pooling Layer



alternatives:

p-Norm Pooling

here: with a 2×2 filter and $S = 2$, $p = 5$

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

$$\xrightarrow{\sqrt[p]{\frac{1}{p^2}\sum_{i,j} x_{ij}^p}}$$

4.86	8
2.38	3.16

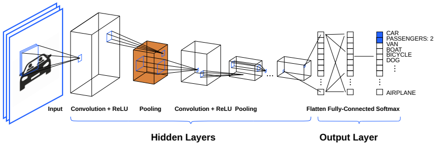
Average Pooling

here: with a 2×2 filter and $S = 2$

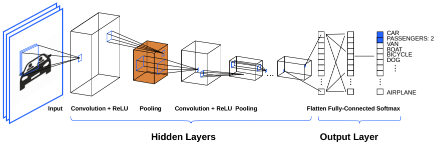
7	1	5	8
4	6	5	1
9	0	7	5
3	2	1	0

$\longrightarrow$

4.5	4.75
3.5	3.25



Other than convolutional filters which are learned, pooling filters are typically not learned.

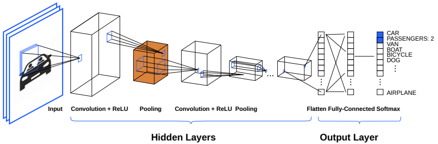


Other than convolutional filters which are learned, pooling filters are typically not learned.

But it is still possible to do so!

In this case, the same network is applied on each block when moving the pooling filter.

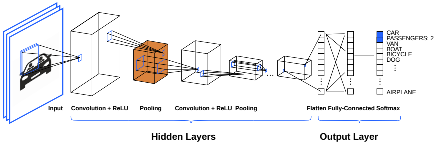
Pooling Layer



Which pooling should we chose?

The most common pooling layer is **Max Pooling**, with a pooling window size of 2 and stride of 2 - resulting in a output dimension that is halved. By picking the most activated value in the pooling window, the Max Pooling layer removes redundant features and helps in preventing over-fitting.

Pooling Layer



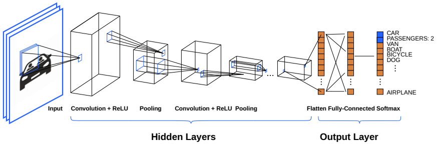
Recap
Motivation
Convolutional Neural Networks
Convolution Layer
Pooling Layer
Training
CNNs in Astronomy
CNN Architectures
Visualizing Features & Fooling CNNs
Outlook
Outlook

Which pooling should we chose?

The most common pooling layer is **Max Pooling**, with a pooling window size of 2 and stride of 2 - resulting in a output dimension that is halved. By picking the most activated value in the pooling window, the Max Pooling layer removes redundant features and helps in preventing over-fitting.

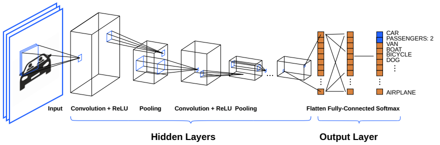
On the other hand, **Average Pooling** typically happens between the Convolutional Layers and the Fully Connected Layers, in place of the Flatten operation. Average Pooling is sometimes preferred compared to the Flatten operation due to it having more less features passed into the Fully Connected Layers, and may reduce over-fitting.

Fully Connected Layers



The fully connected layers classify the detected features in the image into a class label. These are the final layers in a CNN.

Fully Connected Layers

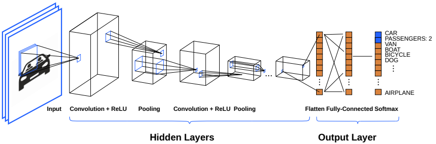


The fully connected layers classify the detected features in the image into a class label. These are the final layers in a CNN.

As input, the previous layer is flattened. There can be multiple fully connected layers. The final layer carries out the classification.

The ReLU activation is used in each fully connected layer except in the final layer in which we use the Softmax activation for multiclass classification.

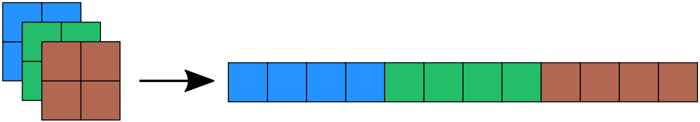
Fully Connected Layers



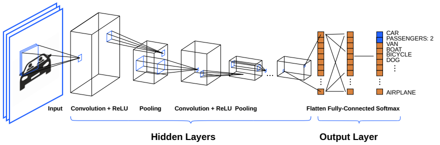
A **Flatten Operation** is necessary:

In a CNN, the final pooled feature map is fed to a Multilayer Perceptron (MLP) to classify the feature map it into a class label.

MPLs only accept one-dimensional data - this requires flattening the final pooled feature map into a single column. Unlike flattening the original image, important pixel dependencies are retained when pooled maps are flattened.



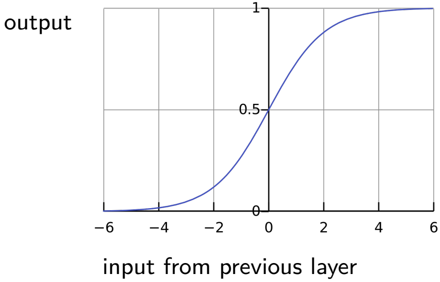
Fully Connected Layers



The Fully Connected Layers learn non-linear combinations of the high-level features as represented by the output of the convolutional layer.

Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks.

The **activation function** is usually a softmax function:



Summary: The Building Blocks of a CNN

The CNN is a built along how the vision is processed in mammals.

Summary: The Building Blocks of a CNN

The CNN is a built along how the vision is processed in mammals.

It includes:

- convolutional layers compromising learned filters that scan the outputs of previous layers
- pooling layers that downsample outputs from convolutional layers
- an output layer that works on flattened data from the last pooling layer.

Summary: The Building Blocks of a CNN

The CNN is a built along how the vision is processed in mammals.

It includes:

- convolutional layers compromising learned filters that scan the outputs of previous layers
- pooling layers that downsample outputs from convolutional layers
- an output layer that works on flattened data from the last pooling layer.

Convolution can change the size of the output. This can be controlled with zero padding.

Summary: The Building Blocks of a CNN

The CNN is a built along how the vision is processed in mammals.

It includes:

- convolutional layers compromising learned filters that scan the outputs of previous layers
- pooling layers that downsample outputs from convolutional layers
- an output layer that works on flattened data from the last pooling layer.

Convolution can change the size of the output. This can be controlled with zero padding.

Pooling layers may perform operations such as max, p-norm, or learned downsampling networks.

Training CNN

As in the case of regular MLP, training happens through backpropagation. The only difference comes from the structure of the neural network.

Training CNN

As in the case of regular MLP, training happens through backpropagation. The only difference comes from the structure of the neural network.

As in the case of a regular MLP:

- a training sample is provided
- the difference between the desired ouput and the true output is defined and calculated in response to any output of the training sample
- network parameters are trained through backpropagation with gradient descent

Training CNN

CNNs give us two key benefits: local invariance and compositionality.

Training CNN

CNNs give us two key benefits: local invariance and compositionality.

In the final **multi-layer perceptron**, all weights and biases for classification are learned to make predictions based on detected higher-level features.

Training CNN

CNNs give us two key benefits: local invariance and compositionality.

In the final **multi-layer perceptron**, all weights and biases for classification are learned to make predictions based on detected higher-level features.

In the **convolutional layers**, the filters must learned. In the context of image classification, our CNN may learn to:

- Detect edges from raw pixel data in the first layer.
- Use these edges to detect shapes (i.e., *blobs*) in the second layer.
- Use these shapes to detect higher-level features such as car parts, lensed galaxies etc.

Training CNN

CNNs give us two key benefits: local invariance and compositionality.

In the final **multi-layer perceptron**, all weights and biases for classification are learned to make predictions based on detected higher-level features.

In the **convolutional layers**, the filters must learned. In the context of image classification, our CNN may learn to:

- Detect edges from raw pixel data in the first layer.
- Use these edges to detect shapes (i.e., *blobs*) in the second layer.
- Use these shapes to detect higher-level features such as car parts, lensed galaxies etc.

The **number of parameters**:
Let each layer j have K_j maps.
Let the filters in the jth layer be of size $L_j \times L_j$
For the jth layer we will require $K_j(K_{j-1}L_j^2 + 1)$ filter parameters.
The total number of parameters for each convolutional layer is then

$$\sum_j K_j(K_{j-1}L_j^2 + 1)$$

Training CNN

Computing the Loss

Given a training set with input-output pairs:
$(x_1, \hat{y}_1), (x_2, \hat{y}_2), ..., (x_T, \hat{y}_T)$

The loss on the ith instance is $\Delta(x_i, \hat{y}_i)$.

The total loss is $E = \frac{1}{T}\sum_{i=1}^{T}\Delta(x_i, \hat{y}_i)$.

The loss is minimized w.r.t. the weights and biases $\boldsymbol{\theta}$

Training CNN

Computing the Loss

Given a training set with input-output pairs: $(x_1, \hat{y}_1), (x_2, \hat{y}_2), ..., (x_T, \hat{y}_T)$

The loss on the ith instance is $\Delta(x_i, \hat{y}_i)$.

The total loss is $E = \frac{1}{T} \sum_{i=1}^{T} \Delta(x_i, \hat{y}_i)$.

The loss is minimized w.r.t. the weights and biases $\boldsymbol{\theta}$

Training through gradient descent

initialize all weights and biases $\boldsymbol{\theta}$

for every layer l for all filter indices m, until loss has converged update $\boldsymbol{\theta}_k \to \boldsymbol{\theta}_{k+1}$:

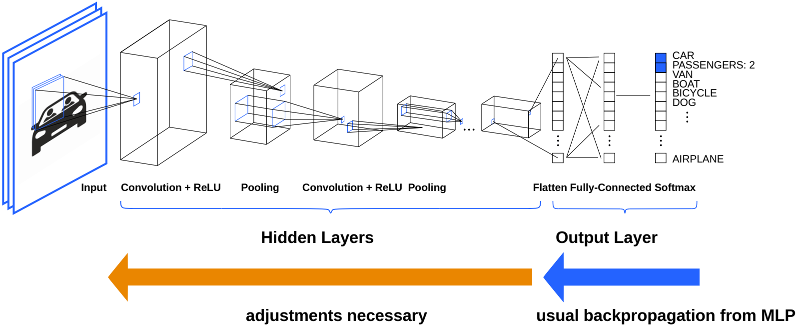
$$\boldsymbol{\theta}_{k+1}(l, m, j, x, y) = \boldsymbol{\theta}_k(l, m, j, x, y) - \eta \frac{\partial E}{\partial \boldsymbol{\theta}_k(l, m, j, x, y)}$$

Training CNN

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

Backpropagation from the flat MLP requires special consideration:

- the shared computation in the convolution layers
- the pooling layers



Training CNN

Both the Forward pass and the Backpropagation of a Convolutional layer are Convolutions.

So the gradients for backpropagation become, with filter F, layer output y_l and layer input $x = y_{l-1}$:

$$\frac{\partial E}{\partial F} = \mathrm{Conv}\left(\text{input } x, \text{loss gradient } \frac{\partial E}{\partial y}\right)$$

$$\frac{\partial E}{\partial x} = \mathrm{Conv}\left(180^\circ \text{ rotated filter } F', \text{ loss gradient } \frac{\partial E}{\partial y}\right)$$

Summary

CNN include convolutional layers comprising learned filters that scan the outputs of the previous layer.

Summary

CNN include convolutional layers comprising learned filters that scan the outputs of the previous layer.

Pooling layers operate on groups of outputs from the convolutional layer to reduce the network size.

Summary

CNN include convolutional layers comprising learned filters that scan the outputs of the previous layer.

Pooling layers operate on groups of outputs from the convolutional layer to reduce the network size.

The parameters of the CNN can be learned through backpropagation.

Applications of Convolutional Neural Networks

Advantages of Convolution Neural Networks:

- used for deep learning with few parameters
- less parameters to learn as compared to fully connected layer
- tool for image processing

Applications of Convolutional Neural Networks

Advantages of Convolution Neural Networks:

- used for deep learning with few parameters
- less parameters to learn as compared to fully connected layer
- tool for image processing

Disadvantages of Convolution Neural Networks:

- Comparatively complex to design and maintain
- Comparatively slow [depends on the number of hidden layers]

Applications of Convolutional Neural Networks

Advantages of Convolution Neural Networks:

- used for deep learning with few parameters
- less parameters to learn as compared to fully connected layer
- tool for image processing

Disadvantages of Convolution Neural Networks:

- Comparatively complex to design and maintain
- Comparatively slow [depends on the number of hidden layers]

General Applications of Convolution Neural Networks:

- Image processing
- Computer Vision
- Speech Recognition

Applications of Convolutional Neural Networks

Specific Applications in Astronomy:

1. extracting prominent features from astronomical images, e.g.
   - craters
   - gravitational lenses

Applications of Convolutional Neural Networks

Specific Applications in Astronomy:

1. extracting prominent features from astronomical images, e.g.
   - craters
   - gravitational lenses
2. classifying objects on astronomical images

Applications of Convolutional Neural Networks

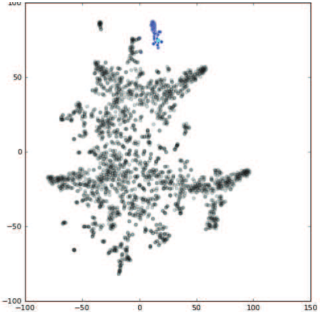
Specific Applications in Astronomy:

1. extracting prominent features from astronomical images, e.g.
   - craters
   - gravitational lenses
2. classifying objects on astronomical images
3. classifying time-series data based on a plot (light curve)

Applications of Convolutional Neural Networks

Specific Applications in Astronomy:

example:



left: the Hubble Heritage image of the Carina Nebula
right: the locations of each of the 2240 sub images in the clustering space

Convolutional Neural Networks in Astronomy, and Applications for Diffuse Structure Discovery, Peek, J. E. G.; Jones, Craig K.; Hargis, Jonathan, Astronomical Data Analysis Software and Systems XXVII, April 2020

Astronomy with CNN

Rotation-invariant convolutional neural networks for galaxy morphology prediction, S. Dieleman, K. W. Willett, J. Dambre (2015)

Measuring the morphological parameters of galaxies is a key requirement for studying their formation and evolution. The deep neural network model for galaxy morphology classification exploits translational and rotational symmetry.

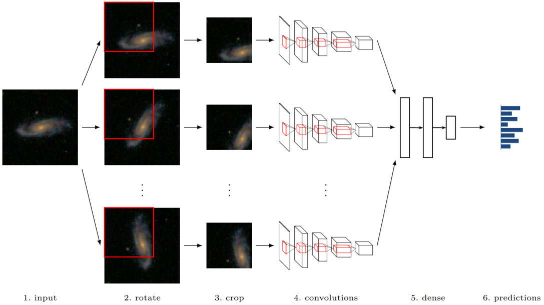
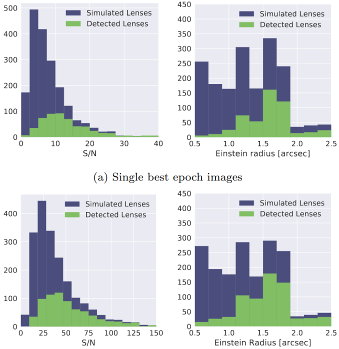


Figure 4: Schematic overview of the CNN architecture. The input (1) is rotated/flipped for different viewpoints (2), which are cropped to reduce redundancy (3). Processing is done by convolutional and pooling layers (4), their output is concatenated and processed by dense layers (5) to obtain predictions (6).

Astronomy with CNN

CMU DeepLens: Deep Learning For Automatic Image-based Galaxy-Galaxy Strong Lens Finding, F. Lanusse, Qu. Ma, N, Li, et al. (2017)

CMU DeepLens is a fully automated galaxy-galaxy lens finding method based on Deep Learning, trained and validated on 20,000 LSST-like mock observations including a range of lensed systems of various sizes and signal-to-noise ratios.



(a) Single best epoch images

(b) Stack images

Figure 7. S/N and Einstein radius distribution of the simulated and recovered lens populations, for our fiducial 1% FPR detection threshold.

Astronomy with CNN

Searching for exoplanets using artificial intelligence, K. A. Pearson, L. Palafox, C. A. Griffith (2018)

the ideal algorithm should be

- fast
- robust to noise
- capable of learning and abstracting highly nonlinear systems

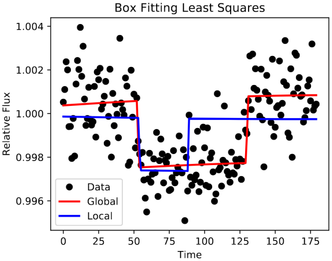
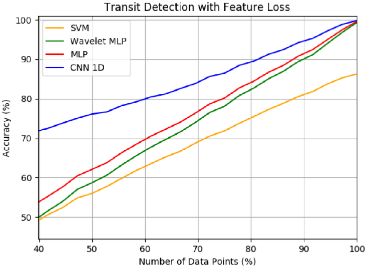


Figure 2. Past transit detection algorithms are accomplished by correlating the data with a simple box model through a least-squares optimization.



Astronomy with CNN

Finding Strong Gravitational Lenses in the Kilo Degree Survey with Convolutional Neural Networks, C. E. Petrillo, C. Tortora, S. Chatterjee et al. (2017)

A morphological classification method based on a Convolutional Neural Network (CNN) for recognizing strong gravitational lenses is applied to the 255 square degrees of the Kilo Degree Survey (KiDS), one of the current-generation optical wide surveys.

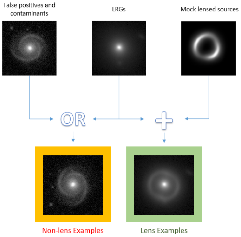
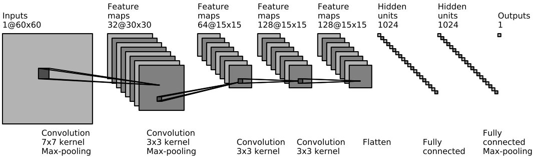


Figure 3. A schematic of the training-set creation.



Astronomy with CNN

Quasar microlensing light curve analysis using deep machine learning, G. Vernardos & G. Tsagkatakis (2019)

Objective: Use CNN to model quasar microlensing light curves and extract the size and temperature profie of the accretion disc.
The effects of microlensing are simulated by magnification maps: pixellated representations of the source plane caustics due to the microlenses.

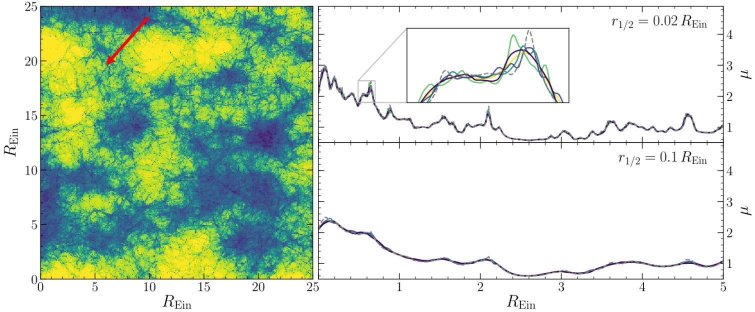


Figure 2. Mock light curve from magnification map. The light curve trajectory on the map is depicted as a red arrow.

CNN Architectures

How to construct a CNN?

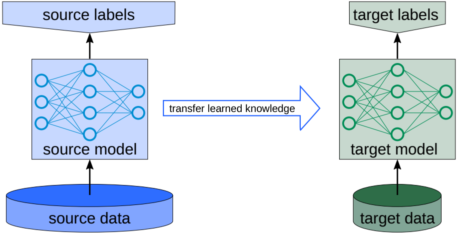
CNN Architectures

How to construct a CNN?

⇨ There are several common CNN architectures we can adapt to solve our science questions.

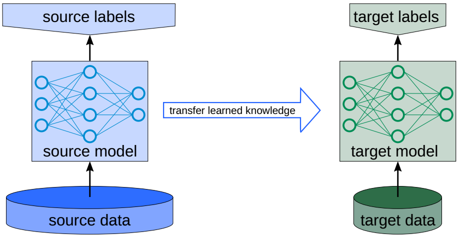
Transfer Learning for Images

The **basic concept** of transfer learning is training on a large dataset and transfer the knowledge to a smaller dataset. It is based on the idea that convolutional layers extract general features applicable across images.



Transfer Learning for Images

The **basic concept** of transfer learning is training on a large dataset and transfer the knowledge to a smaller dataset. It is based on the idea that convolutional layers extract general features applicable across images.

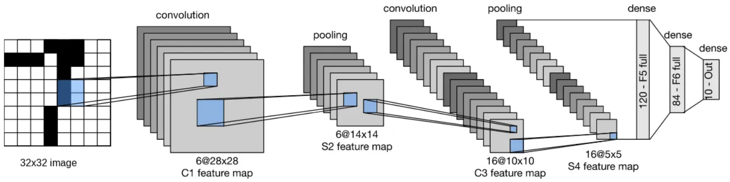


Following is the general outline for transfer learning for object recognition:

1. Load in a pre-trained CNN model trained on a large dataset.
2. Freeze parameters (weights) in model's lower convolutional layers.
3. Add custom classifier with several layers of trainable parameters.
4. Train classifier layers on training data available for task.
5. Fine-tune hyperparameters and unfreeze more layers as needed.

LeNet-5

Lenet-5 was proposed by Y. LeCun (1998) in **Gradient-Based Learning Applied to Document Recognition**. This CNN is used to predict handwritten numeric characters in greyscale images.



Excluding pooling, LeNet-5 consists of 5 layers:

- 2 convolution layers with kernel size 5×5, followed by
- 3 fully connected layers.

Tanh activation function is used except for the last layer (which has softmax). LeNet-5 has 60,000 trainable parameters.
Input: 32×32 pixel image. Largest character is 20×20. Pixel values are normalized (mean of pixels $= 0$, std of pixels $=1$).

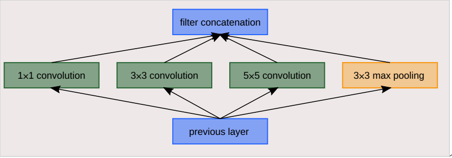
GoogLeNet (Inception-v1)

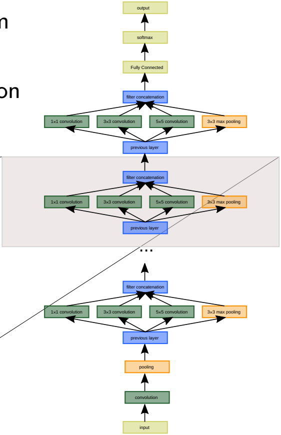
GoogLeNet, also called Inception-v1 (Szegedy et al. 2014) is built on the concept of a **inception module**: design a good local network topology and then **stack** these modules on top of each other (network within a network)

a naive inception module:

apply parallel filter operations on the input from previous layer:

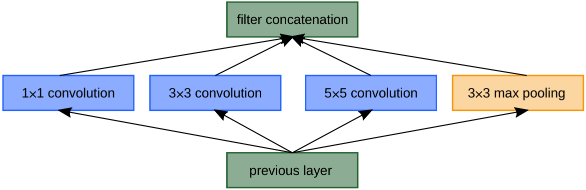
- multiple receptive field sizes for convolution (1×1, 3×3, 5×5)
- pooling operation (3×3)





GoogLeNet (Inception-v1)

the problem: **computational complexity**

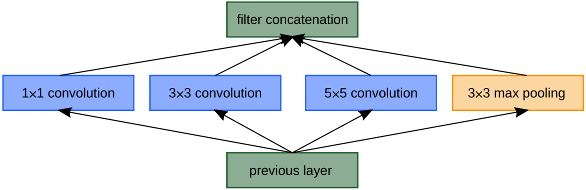


the **convolutional operations**:

e.g. with a module input of $28 \times 28 \times 256$, we get:
1×1 convolution: $28 \times 28 \times 128 \times 1 \times 1 \times 256$ operations
3×3 convolution: $28 \times 28 \times 192 \times 3 \times 3 \times 256$ operations
5×5 convolution: $28 \times 28 \times 96 \times 5 \times 5 \times 256$ operations

GoogLeNet (Inception-v1)

the problem: **computational complexity**

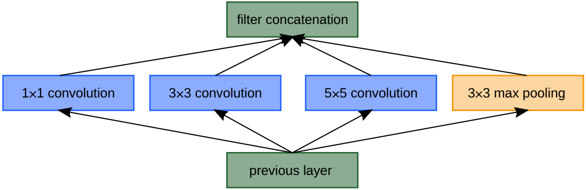


the **convolutional operations**:

e.g. with a module input of $28 \times 28 \times 256$, we get:
1×1 convolution: $28 \times 28 \times 128 \times 1 \times 1 \times 256$ operations
3×3 convolution: $28 \times 28 \times 192 \times 3 \times 3 \times 256$ operations
5×5 convolution: $28 \times 28 \times 96 \times 5 \times 5 \times 256$ operations
$\Rightarrow$ total 854×10^6 operations

GoogLeNet (Inception-v1)

the problem: **computational complexity**



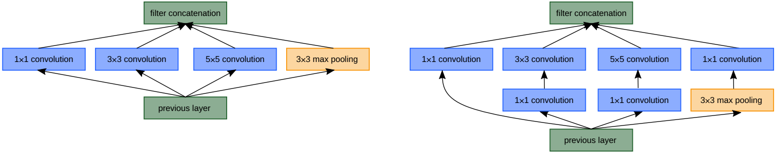
the **convolutional operations**:

e.g. with a module input of $28 \times 28 \times 256$, we get:
$1{\times}1$ convolution: $28 \times 28 \times 128 \times 1 \times 1 \times 256$ operations
$3{\times}3$ convolution: $28 \times 28 \times 192 \times 3 \times 3 \times 256$ operations
$5{\times}5$ convolution: $28 \times 28 \times 96 \times 5 \times 5 \times 256$ operations
$\Rightarrow$ total 854×10^6 operations

plus: **pooling layer** preserves the feature depth which means total depth after concatenation can only grow at every layer

GoogLeNet (Inception-v1)

Solution (Szegedy et al. 2014): **bottleneck layers** that use 1×1 convolutions to reduce feature depth



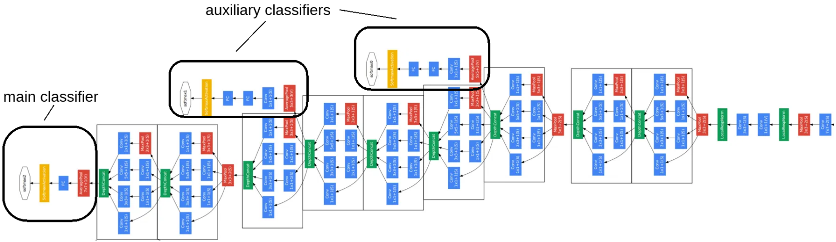
naive vs. with dimensionality reduction

computationally more efficient:

e.g. with a module input of $28 \times 28 \times 256$, we get:
total 358×10^6 operations (compared to 854×10^6 operations for the naive version)

GoogLeNet (Inception-v1)

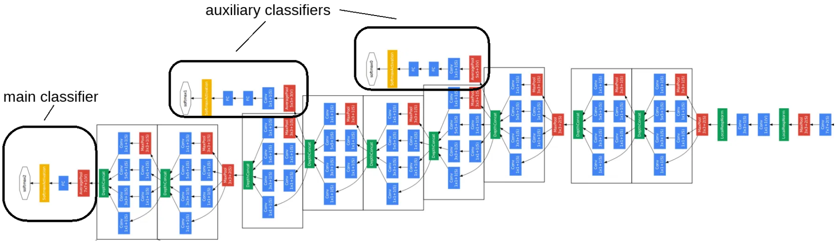
adding more layers comes with a **drawback**: exploding/vanishing gradients



GoogLeNet (Inception-v1)

adding more layers comes with a **drawback**: exploding/vanishing gradients

GoogLeNet solves this by adding **auxiliary classifiers** after the 3rd and 6th inception module to increase the gradient signal that gets propagated back. During training, their loss is added to the network's total loss with a 0.3 weight. At inference time, these auxiliary networks are discarded.

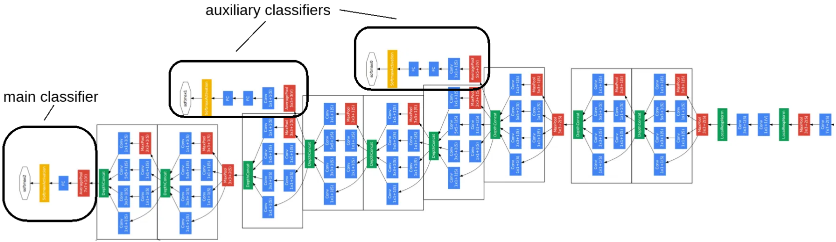


GoogLeNet (Inception-v1)

adding more layers comes with a **drawback**: exploding/vanishing gradients

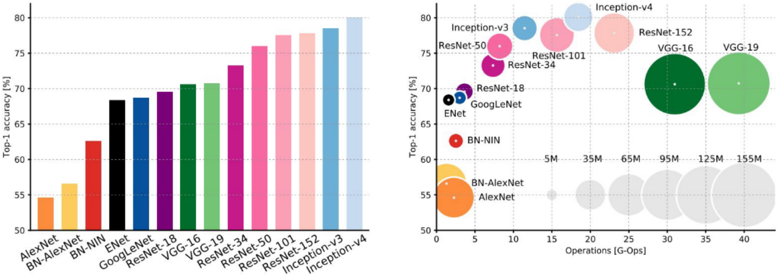
GoogLeNet solves this by adding **auxiliary classifiers** after the 3rd and 6th inception module to increase the gradient signal that gets propagated back. During training, their loss is added to the network's total loss with a 0.3 weight. At inference time, these auxiliary networks are discarded. The auxiliary classifiers, as well as the last layer of the main classifier, use a softmax activation function (all others use ReLU).

Auxiliary classifiers start with a 5×5 average-pooling, followed by a convolution layer with 1×1 kernel size and two fully connected layers.



CNN Architectures - Case Studies

comparing complexity



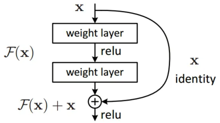
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

CNN Architectures - Summary

CNN Architecture	Default Input	Default Output	Number of Layers	Number of Parameters	Activation Function	specific properties
LeNet-5	$32 \times 32 \times 1$	10	5	60k	tanh	convolution layer
AlexNet	$244 \times 244 \times 3$	1000	8	60M	ReLU	local response normalization
VGG-16	$244 \times 244 \times 3$	1000	16	138M	ReLU	very deep but single-thread
GoogLeNet	$244 \times 244 \times 3$	1000	22	7M	ReLU	inception module, auxiliary classifiers
ResNet-50	$244 \times 244 \times 3$	1000	50	26M	ReLU	batch normalization, residual blocks

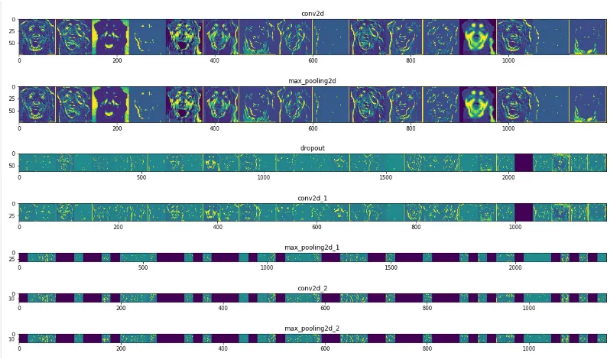
local response normalization:
This implements the idea of lateral inhibition where an excited neuron inhibits its neighbours, leading to contrast in that area.

residual blocks:
Traditionally, each layer feeds into the next layer. In a network with residual blocks, each layer feeds into the next layer and into the layers about 2-3 hops away.



Visualizing CNN Features

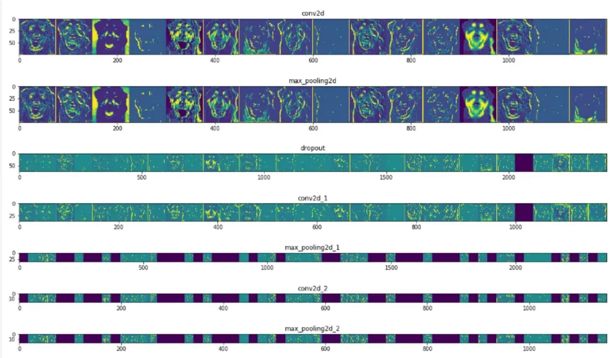
Visualizing how a CNN learns to identify different features present in images provides a deeper insight into the model.



Visualizing CNN Features

Visualizing how a CNN learns to identify different features present in images provides a deeper insight into the model.

This can be done with a process called **Gradient Ascent**.

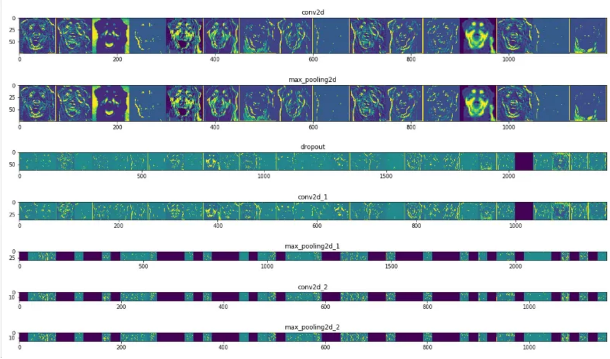


Visualizing CNN Features

Visualizing how a CNN learns to identify different features present in images provides a deeper insight into the model.

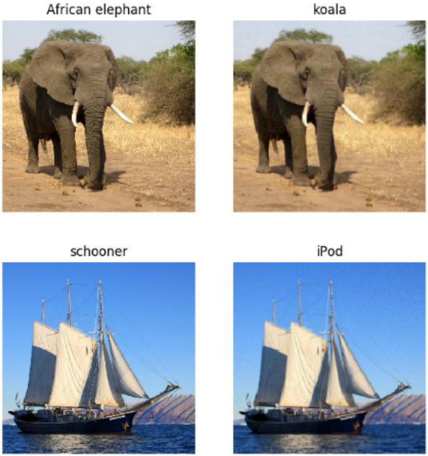
This can be done with a process called **Gradient Ascent**.

Gradient Ascent starts with a zero or random image and continuously backpropagates until an image is generated that maximizes the score for the particular class (e.g.: dog).



Fooling a CNN

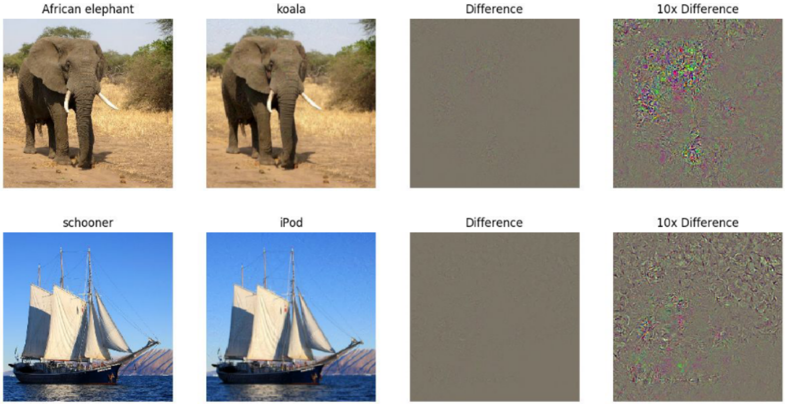
a CNN's classification:



?

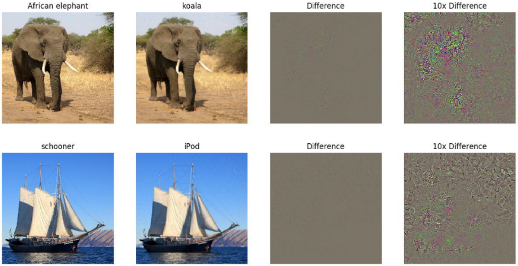
Fooling a CNN

Suppose we feed in an image to a CNN. Instead of maximizing the likelihood of the correct class (e.g. elepant) we maximize an incorrect class (e.g. koala) while making minimal changes to the image.



Fooling a CNN

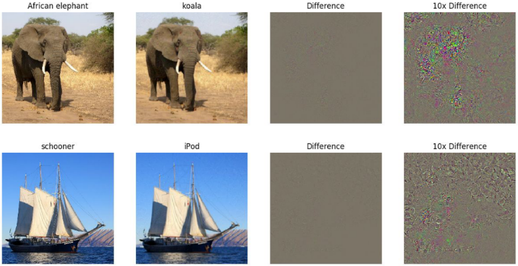
Suppose we feed in an image to a CNN. Instead of maximizing the likelihood of the correct class (e.g. elepant) we maximize an incorrect class (e.g. koala) while making minimal changes to the image.



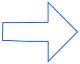
Intriguing properties of neural networks, Szegedy et al. (2013)
Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images, Nguyen, Yosinski, Clune (2014)

Fooling a CNN

Suppose we feed in an image to a CNN. Instead of maximizing the likelihood of the correct class (e.g. elepant) we maximize an incorrect class (e.g. koala) while making minimal changes to the image.



Intriguing properties of neural networks, Szegedy et al. (2013)
Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images, Nguyen, Yosinski, Clune (2014)



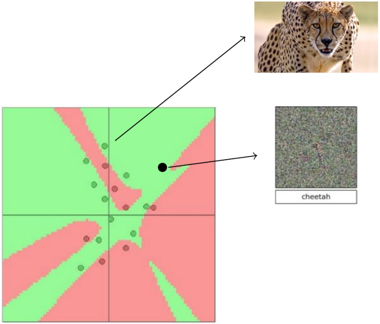
Notice that the changes are so minimal that the two images are indistinguishable to humans

Fooling a CNN

Why does this work?

Using the training images we fit some **decision boundaries** in the high-dimensional parameter space.
While doing so based on the **finite set of training images**, also decisions about many unseen points in parameter space are made.

As a result: Of the many points in this high-dimensional parameter space belonging to a certain class, only a few represent true images.



credit:
Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images, Nguyen, Yosinski, Clune (2014)

Google's *Deep Dream*

A fun way to use CNN:

Exaggerating feature attributes or textures using information that the bvlc_googlenet model learned during training.

The Mona Lisa with DeepDream effect using VGG16 network trained on ImageNet.

Google's *Deep Dream*

A fun way to use CNN:

1. Forward propagate a baseline image through your model. Compute the activations at a chosen layer.
2. To maximize the neuron activations in the layer we care about, set the gradient of chosen layer equal to its activation.
3. During the backward pass, compute the gradient on the loss w.r.t. the input image, as in the case of gradient ascent.
4. Execute the update rule to update your input image. We change the image so that these neurons fire even more.

you can try it out at: `https://deepai.org/`

Summary and Outlook

CNNs enable us to deal with images more naturally.

Summary and Outlook

CNNs enable us to deal with images more naturally.

Next time we look at another neural network architecture: **recurrent neural networks (RNN)**.

Summary and Outlook

June 5: **Presentation on project idea**

Presentation (15 minutes) should cover:

1. Clear explanation of the overall problem you want to solve

2. The dataset(s) you will use

2. Relationship to the topics covered in class: Which models/algorithms are you planning to use, how will you evaluate performance

4. List of papers you plan to read as references

5. How will you structure the project, rough timeline