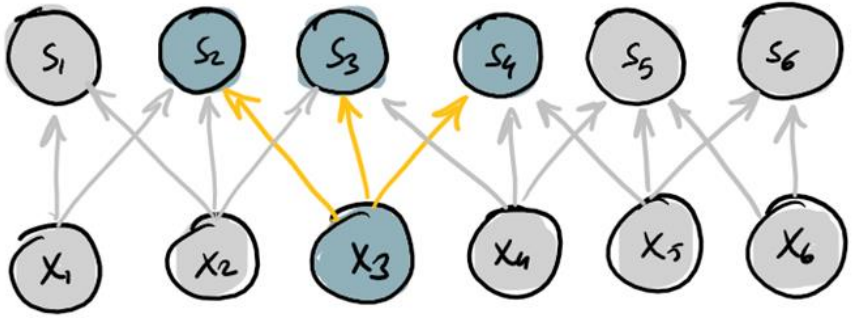
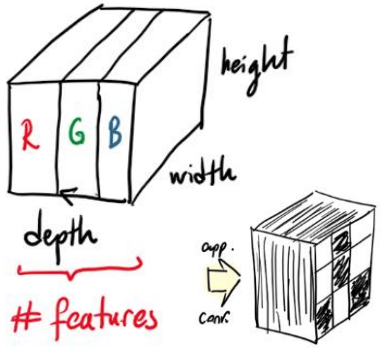
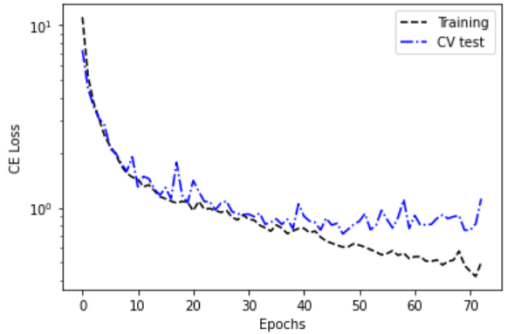
Data Driven Engineering II: Advaced Topics

Image processing and analysis

Institute of Thermal Turbomachinery
Prof. Dr.-Ing. Hans-Jörg Bauer







Outline of the week :

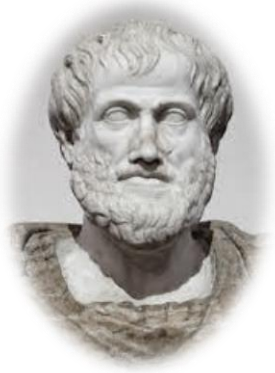
There are popular architectures that are pre-trained. We can download them and use it for personal purposes. Usually comeptitions are win by embedding of this trained models. You just download and apply the transfer learning methodology.

Karlsruhe Institute of Technology

Conv. Neural Networks

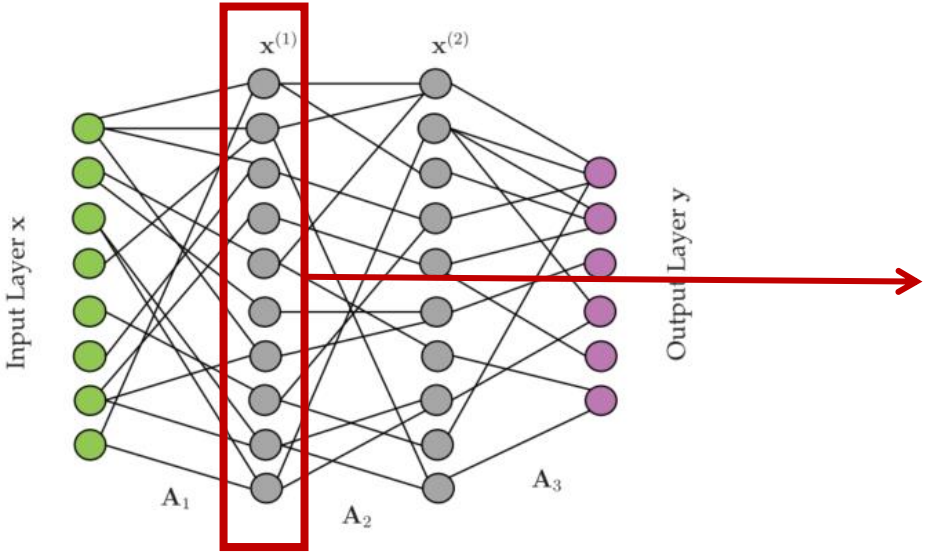
* What is CNN ?
* Why convolution is useful ?
* Where is it useful ?
* CNN – How does it work ?
* "Hall of Fame" : Popular Arch.
* Transfer Learning with CNN

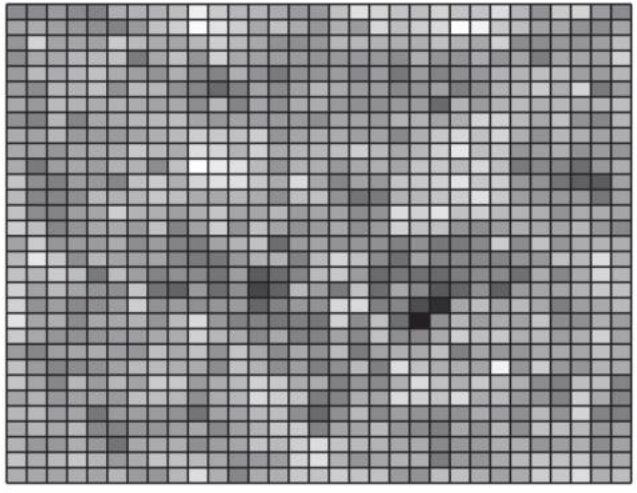
} Next week



"The soul never thinks without a picture."

Connectivity in ANN





It sounds like your professor is giving an overview of the fundamental ideas that underpin Convolutional Neural Networks (CNNs). Let's try to weave together the key words you provided:

At the core of a neural network is the idea that we want to learn meaningful representations of our input data that will help us solve a particular task. This starts with taking our input and "projecting" it into a different space, typically a higher-dimensional space where the task becomes easier to solve.

The term "projection" in this context often refers to a mathematical transformation of the data. A simple example of a projection is a dot product, which you can think of as measuring how much of one vector lies in the direction of another. This can be interpreted as the cosine of the angle between the two vectors, hence the mention of "cosine projection."

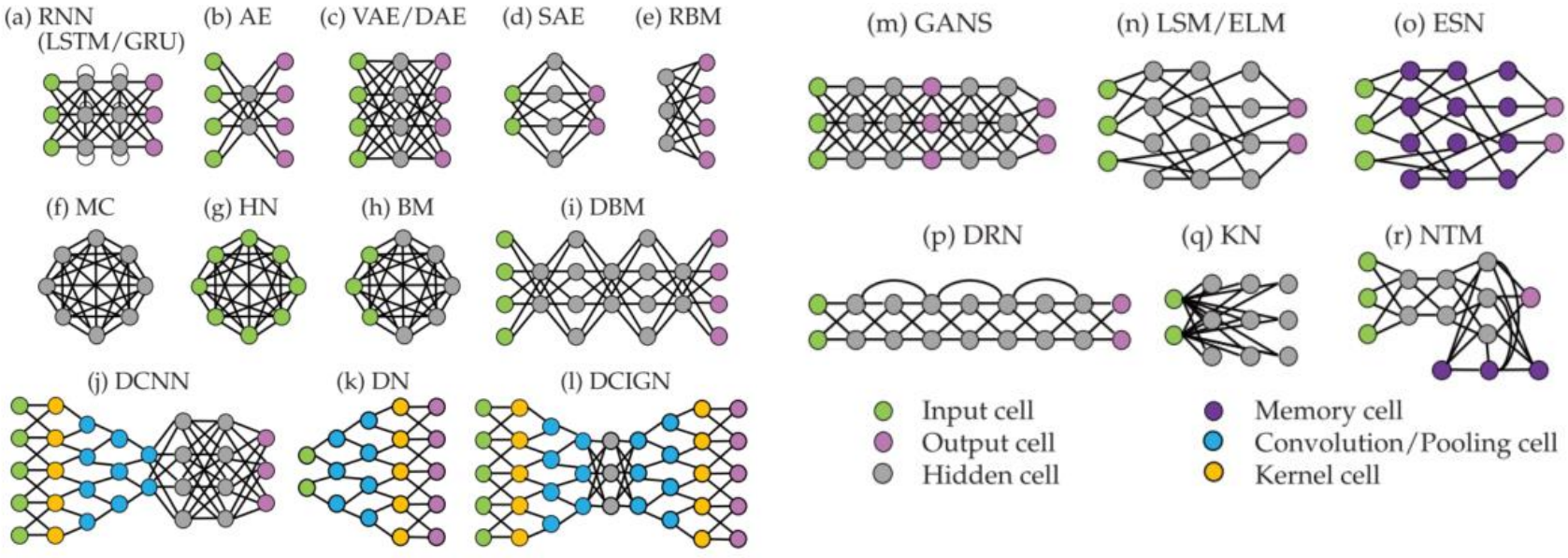
Each layer of a neural network applies its own transformation to the data it receives, which can be thought of as a projection followed by an activation function. The activation function, acting like a filter, introduces non-linearity and allows the network to learn complex patterns.

These transformations or projections can be thought of as a form of automated feature engineering. Each layer learns to represent the data in a way that makes the task easier for the subsequent layers.

Eventually, through successive layers, we reach a point where the representations are suitable for our task, and we use a final layer to make predictions based on these representations. The architecture of the connections between these layers determines the type of neural network we have.

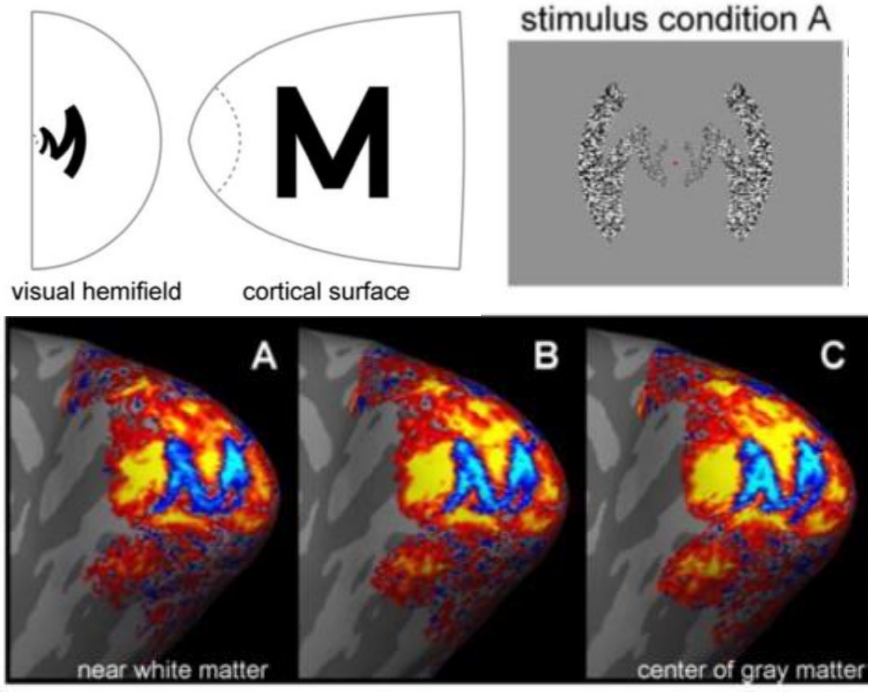
Convolutional Neural Networks (CNNs) are one such architecture that is particularly well-suited for tasks involving spatial data, such as images. Instead of connecting every neuron to every neuron in the next layer (like in a fully connected layer), a CNN uses convolutions over the input layer to compute the output, effectively focusing on local, contiguous sections of the input data. This local focus and sharing of weights allows CNNs to excel at tasks such as image recognition, where the relationship between neighboring pixels is critically important.

Connectivity in ANN



- CNN := Special case of partially connected ANNs

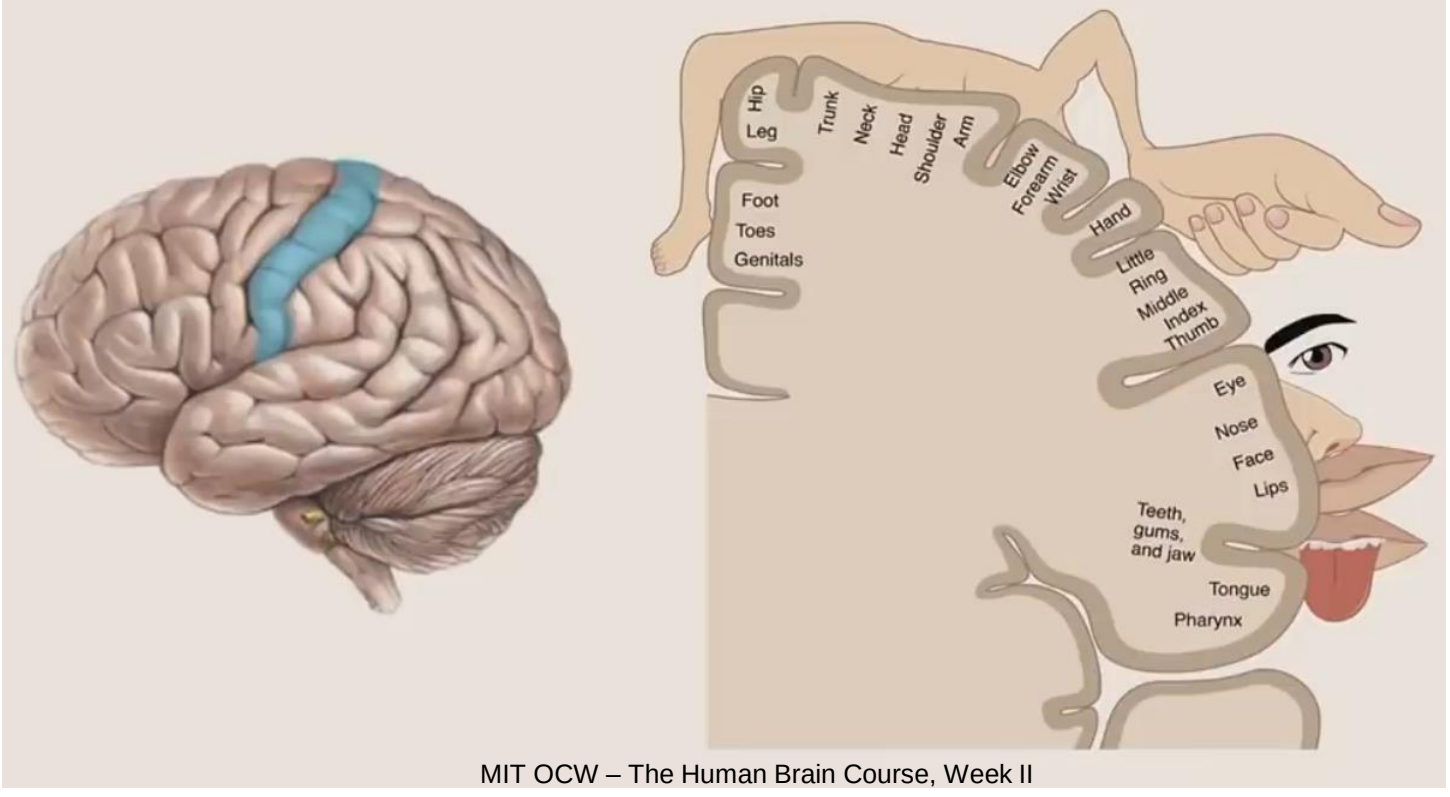
Concept of receptive field: Primary Sensory Cortex



- Sensory regions are organized via maps
- Concept of receptive field:
  - A “place” in the state space makes a particular neuron file (relatively)
  - Spatially organized
  - State: location, color, shape, direction of motion

MIT OCW – The Human Brain Course, Week II; 10.1016/j.neuroimage.2010.05.005

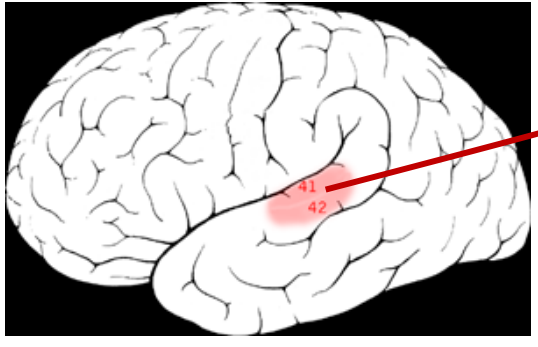
Primary Sensory Cortex: Touch maps

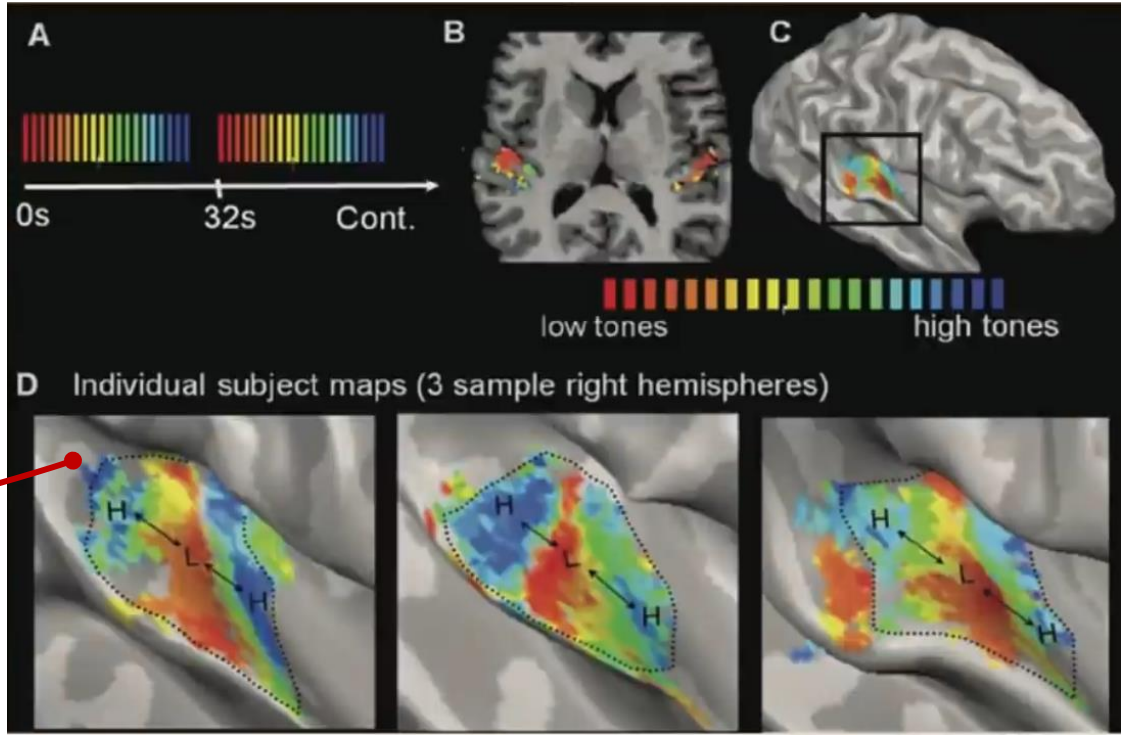


MIT OCW – The Human Brain Course, Week II

Primary Sensory Cortex: Auditory maps

- Sensory regions are organized via maps:
  - Frequency-based

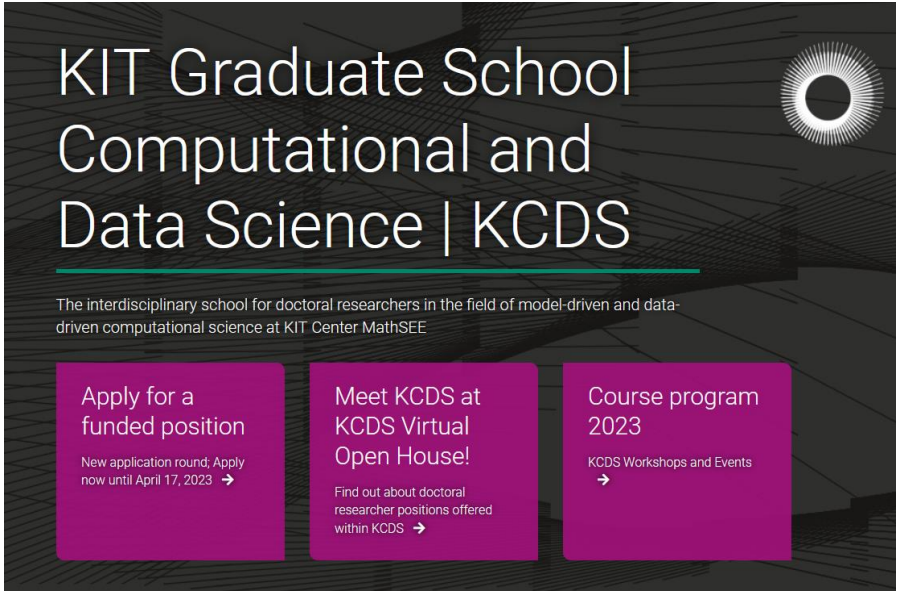




MIT OCW – The Human Brain Course, Week II

KCDS Talk

What makes our brain efficient is that when we are trying to solve a problem, an specific part of the neurons are activated, not all the connections at once.

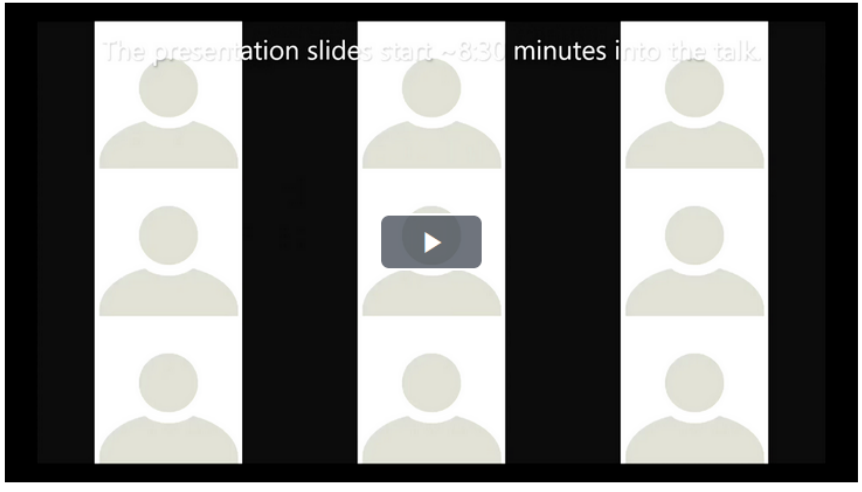


KCDS Talk: "How the brain works and why adaptive models matter" (Dr. Cihan Ates)

Hühnerfuß, Angela [Hrsg.] [1]; Ates, Cihan [2]

[1] KIT-Zentrum Mathematik in den Natur-, Ingenieur- und Wirtschaftswissenschaften (KIT-Zentrum MathSEE), Karlsruher Institut für Technologie (KIT)

[2] Institut für Thermische Strömungsmaschinen (ITS), Karlsruher Institut für Technologie (KIT)



Conv. Neural Networks : Basics

Karlsruhe Institute of Technology

Convolutional Neural Network

Specialized

- Organized, Grid-like data
  - 1D
  - 2D
  - 3D
  - N-D

Training

- Labelled data
- "Backprop."

- ☑ Flag ship of Deep Learning
- ☑ Image / Video processing
  - Benchmark datasets
  - Best solutions ~ weekly monthly

Conv. Neural Networks : Basics

Karlsruhe Institute of Technology

Datasets

3,749 Machine Learning Datasets

„paperswithcode"

CIFAR-10

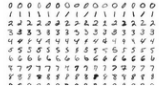
The CIFAR-10 dataset (Canadian Institute for Advanced Research, 10 classes) is a subset of the Tiny Images dataset and consists of 60000 32x32 color images. The images are labelled with...

5,634 PAPERS • 44 BENCHMARKS

ImageNet

The ImageNet dataset contains 14,197,122 annotated images according to the WordNet hierarchy. Since 2010 the dataset is used in the ImageNet Large Scale Visual Recognition Challenge...

5,565 PAPERS • 56 BENCHMARKS



MNIST

The MNIST database (Modified National Institute of Standards and Technology database) is a large collection of handwritten digits. It has a training set of 60,000 examples, and a test set of...

3,881 PAPERS • 36 BENCHMARKS

COCO (Microsoft Common Objects in Context)

The MS COCO (Microsoft Common Objects in Context) dataset is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of 328K im-...

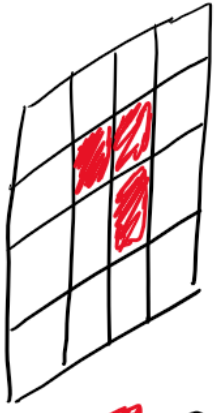
3,804 PAPERS • 58 BENCHMARKS

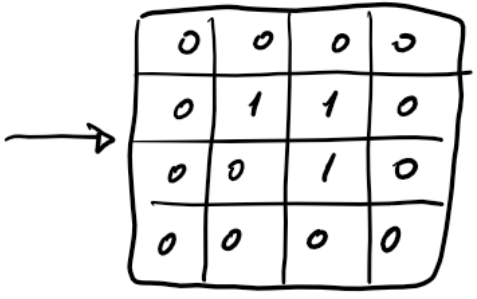
Under the hood: CNN layer

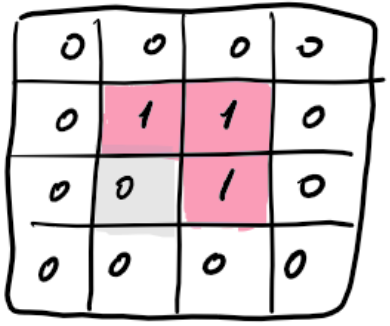
Conv. Layer * "Grid-like" // spatial structure

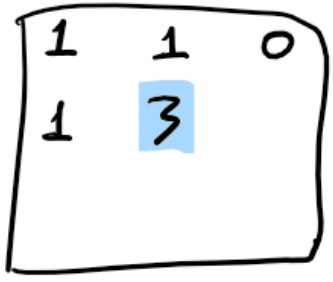
Book Keeping

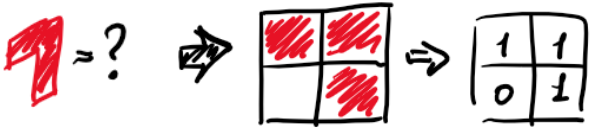
Eg:





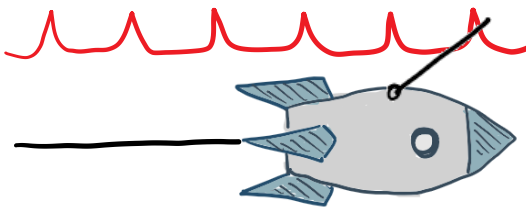






~ Projection score of pattern

What is convolution operation?



Light emitter

$x(t)$: position

t	x

$$s(t) = \int x(a)\, w(t-a)\, da$$

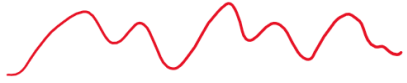
weighted

history

"Convolution"

W := wisdom

noise



↳ Averaging ~ integration

Recent data is more relevant !

↳ weighted averaging

Convolution in CNN:

Karlsruhe Institute of Technology

Term:

- x := input
- w := kernel
- s := feature map

Data

- Discrete; $\int \rightarrow \sum \Rightarrow s = \sum x(a)\, w(t-a)$
- Multi-dim; 2D $\Rightarrow s(i,j) = \sum_m \sum_n I(m,n)\, K(i-m, j-n)$

$I(m+i, n+j)\, K(m,n)$ *

$(a+b = b+a)$

Wait a minute... Is it you MLP?

* What is special here?

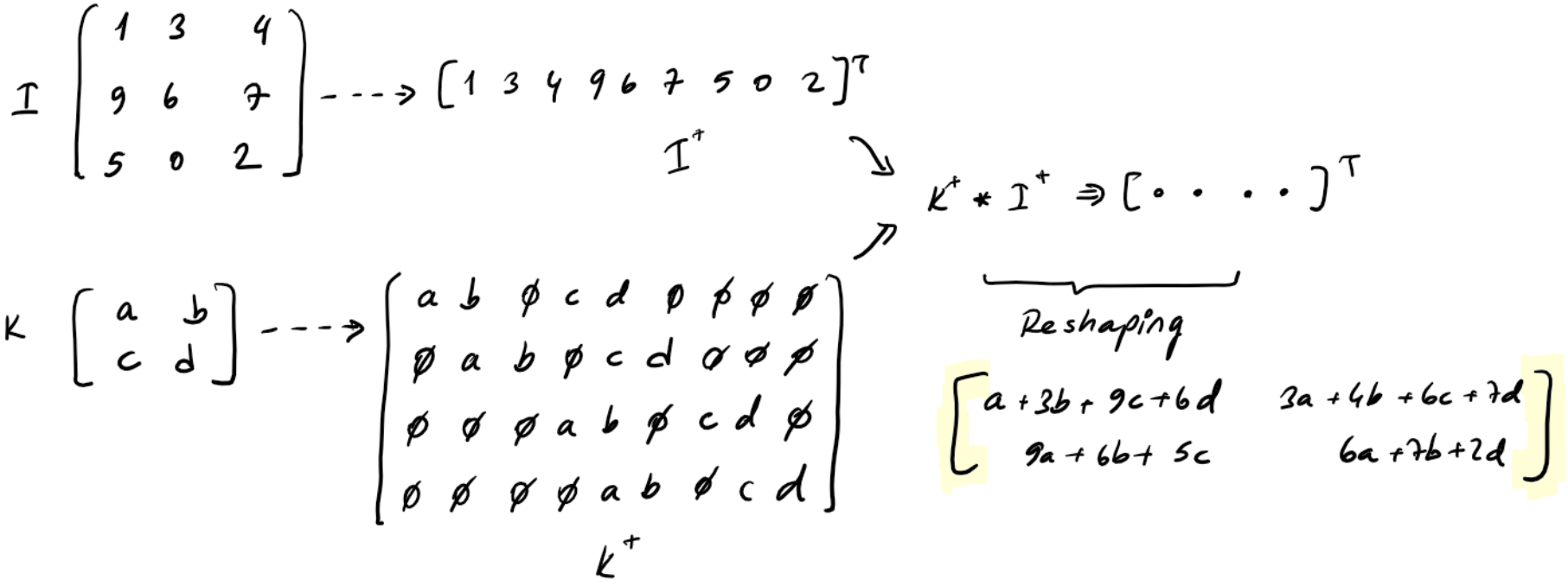
Operation ⇒ " Multiplication by a matrix "

* Kernel is smaller than input

$m \times n \Longrightarrow k \times n \; ; \; k \ll m$

$$\underbrace{\begin{bmatrix} 1 & 3 & 4 \\ 9 & 6 & 7 \\ 5 & 0 & 2 \end{bmatrix}}_{I} * \underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{K} = \begin{bmatrix} a+3b+ & 3a+4b+ \\ 9c+6d & 6c+7d \\ 9a+6b+ & 6a+7b+ \\ 5c+\emptyset & \emptyset+2d \end{bmatrix}$$

Wait a minute... Is it you MLP?

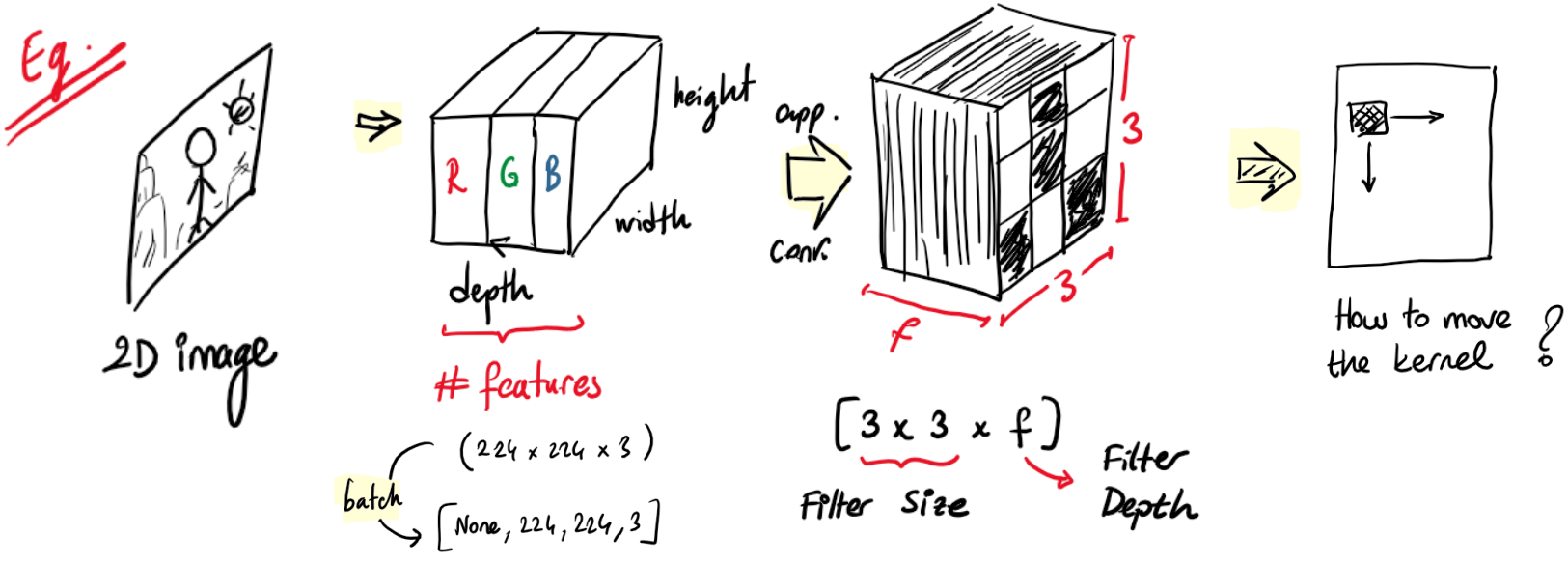


Under the hood : CNN layer

If we are talking about images as in put case, we may have multiple features in it. So a filter can be put it on an image, and you then have three filters on it; RGB. So every pixel value, has a vector of three. Why we do that? Because CNNs expect structured data.

Karlsruhe Institute of Technology

Conv. Layer * "Grid-like" // spatial structure



We are doing the weighted averaging by sliding the filter over the dataset

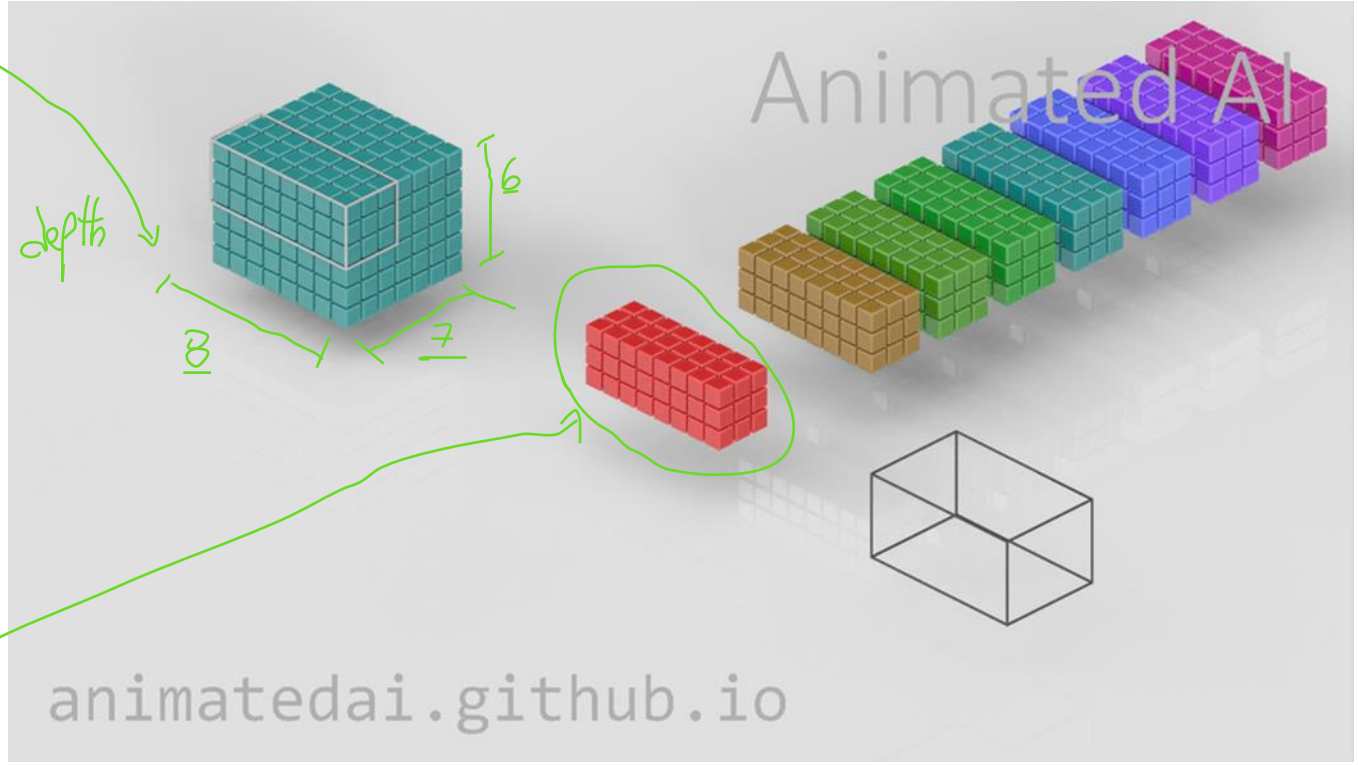
Under the hood : CNN layer

Eg

- # features ⇒ 8
- Spatial domain ⇒ 6 x 7
- # filters applied ⇒ 8
  ! → color coded ~

Filter array:

[3 x 3 x 8]



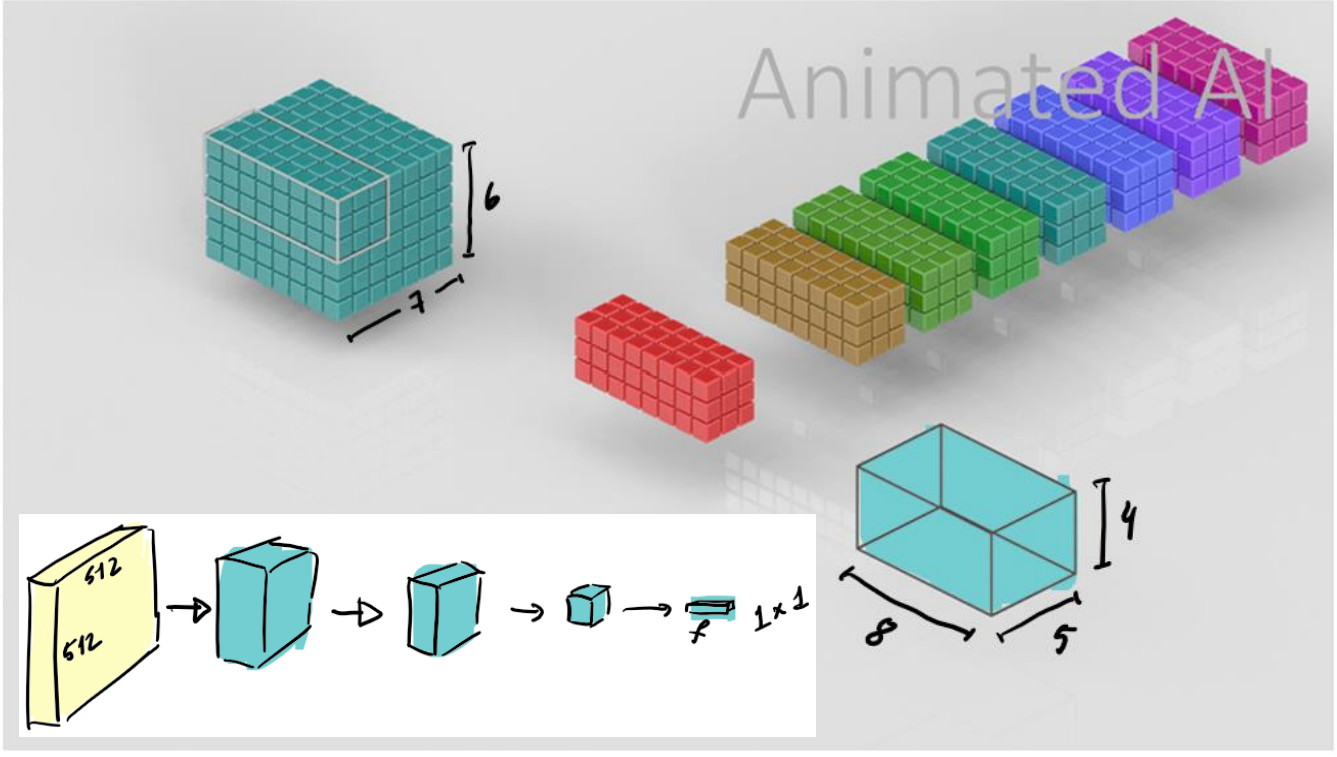
Under the hood : CNN layer

Eg.

- # features ⇒ 8
- Spatial domain ⇒ 6 x 7
- # filters applied ⇒ 8
  ! → color coded ~

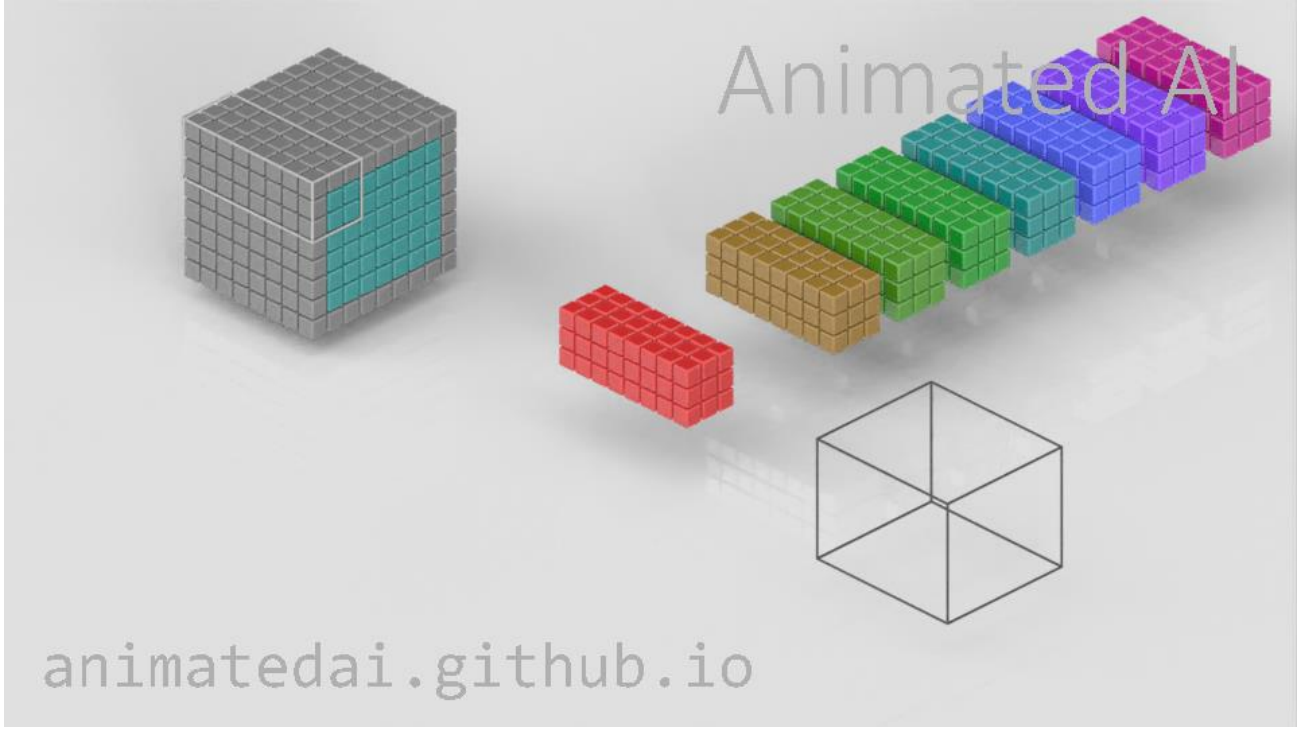
Filter array:

[3 x 3 x 8]



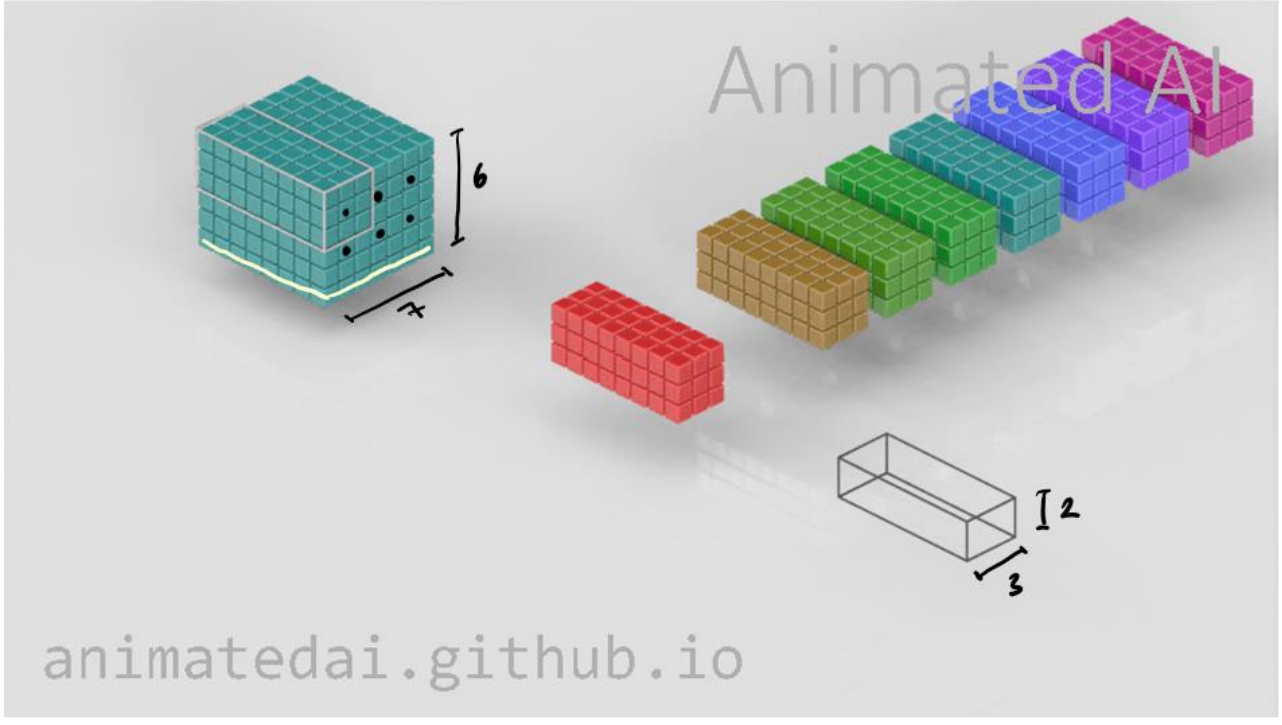
Padding in ConvNets

- Add imagmary pixels; TF := zero
- TF ⇨ (i) same; (ii) valid.
  ! → add to all edges as equally as possible
- Valid (border); rightmost is in the image.



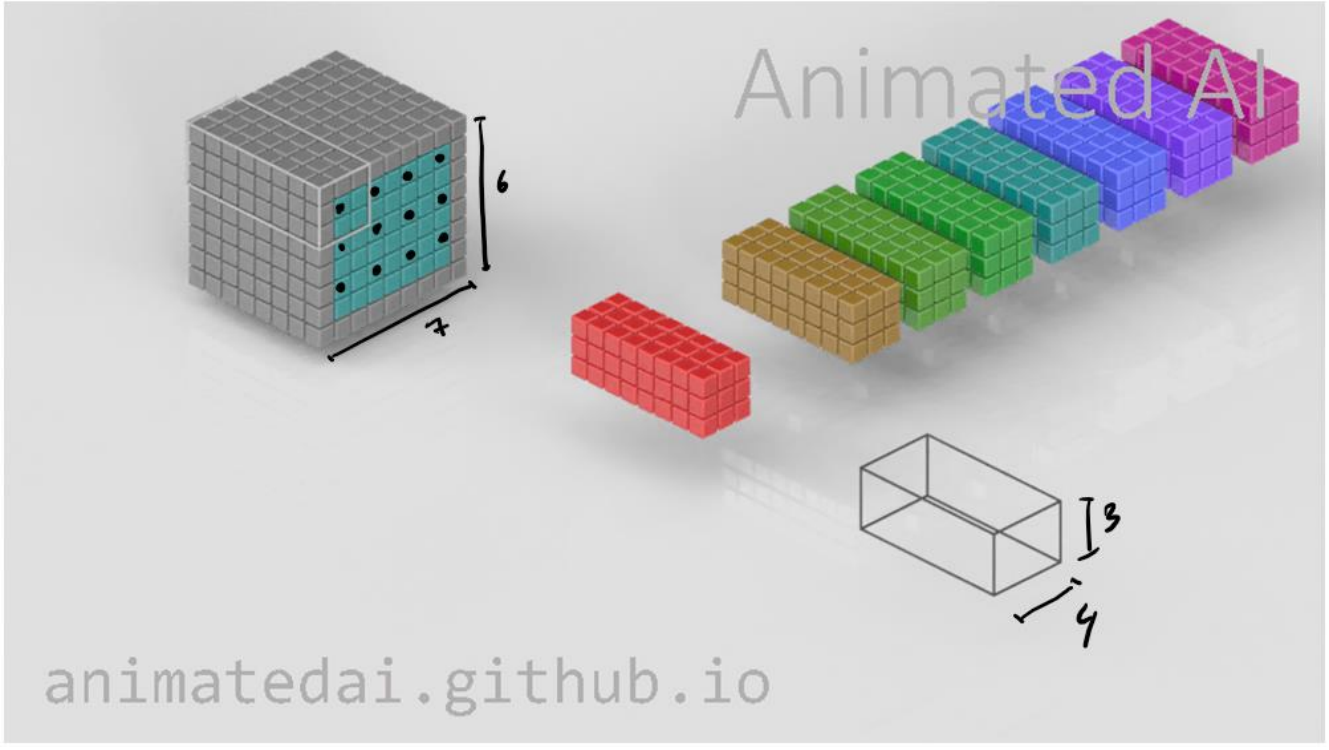
Downsampling option I: Strides

- How to slide the filters
- Filter size ↔ Stride
- padding



Downsampling option I: Strides

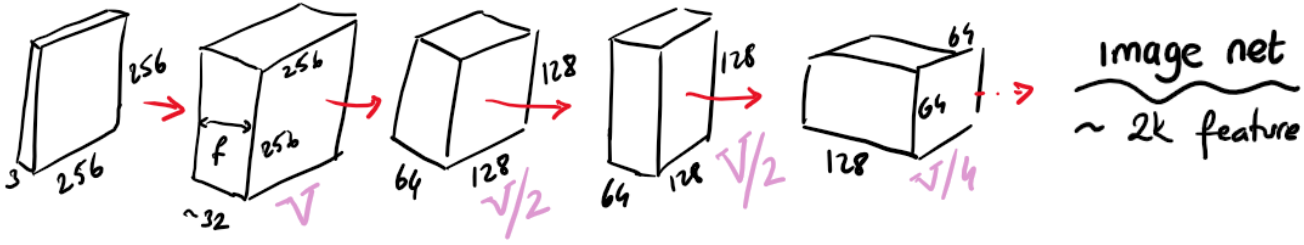
- How to slide the filters
- Filter size ↔ Stride
- padding



Notes on how to build a CNN

- Feature filters are learnt via back prop.
- First layer ~ 3×3, 5×5, 7×7
- Hidden layers ~ 3×3; sometimes 1×1
- As you go deep; #filters ↑ while "resolution" ↓
  - padding
  - strides
  - pooling

  ! ---> usually ⇒ volume is halved



Under the hood : CNN layer

Pooling

* Obj : subsample the input data → comp. load, memory, # parameters

* Replace the output with summary statistics

- ☑ Max.
- ☑ Mean
- ☑ l_2 norm.
- ☑ weighted operations (eg. distance based)

The pooling operation used in convolutional neural networks is a big mistake, and the fact that it works so well is a disaster.

Geoffrey Hinton

Under the hood: CNN layer

Pooling

Max Pooling

$$I = \begin{bmatrix} 6 & 3 & 4 & 4 & 5 & 0 & 3 \\ 4 & 7 & 4 & 0 & 4 & 0 & 4 \\ 7 & 0 & 2 & 3 & 4 & 5 & 2 \\ 3 & 7 & 5 & 0 & 3 & 0 & 7 \\ 5 & 8 & 1 & 2 & 5 & 4 & 2 \\ 8 & 0 & 1 & 0 & 6 & 0 & 0 \\ 6 & 4 & 1 & 3 & 0 & 4 & 5 \end{bmatrix}$$

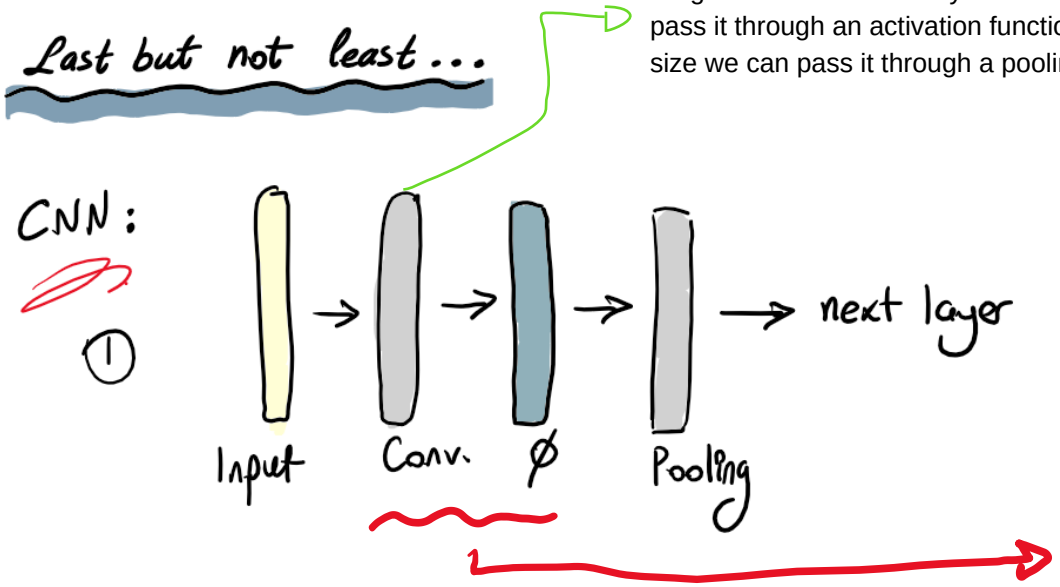
$(3\times3),\ s=1$

$$\Rightarrow \begin{bmatrix} 7 & 7 & 5 & 5 & 5 \\ 7 & 7 & 5 & 5 & 7 \\ 8 & 8 & 5 & 5 & 7 \\ 8 & 8 & 6 & 6 & 7 \\ 8 & 8 & 6 & 6 & 6 \end{bmatrix}$$

$3\times3,\ s=2$

$$\begin{bmatrix} 7 & 5 & 5 \\ 8 & 5 & 7 \\ 8 & 6 & 6 \end{bmatrix}$$

weighted sum. Element by element multiplication. This is linear, if we want to make it non-linear we can pass it through an activation function like an MLP. Then if we want to reduce the size we can pass it through a pooling layer,




```python
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

Understanding CNN layers

In a convolutional neural network, a key operation is the convolution of the input data with a set of filters or kernels. A filter can be thought of as a small matrix of weights. This filter is systematically moved across the input data (also a matrix if it's an image), performing an element-wise multiplication with the part of the input it currently covers, and then summing up the results into a single output pixel. This operation is then repeated for every position the filter can reach on the input data, resulting in a new matrix called a feature map or convolutional layer.

The reason that the output is smaller than the input is that the filter needs to be able to fit squarely within the input. If you imagine sliding a small square across a larger one, the small square won't fit in the larger one once it gets close to the edges. Hence, depending on the size of your filter, a certain amount of the input near the edges may not be included in the output. This is why convolution can reduce the spatial dimensions (width and height) of the image.

Understanding Filters

Filters are designed to detect specific types of features in the input data. For instance, in image processing, one filter might detect horizontal edges in the image, while another might detect vertical edges. The number of filters in a layer of a convolutional network represents the number of features it's trying to learn. A CNN does not necessarily need to have the same number of filters as input features; this is just a design choice based on what you think the complexity of your data might be. Having more filters allows the network to learn more complex representations.

Understanding Padding

Padding is a technique used to preserve the spatial dimensions of the input through the convolutional layers. There are two common types of padding: 'valid' and 'same'. 'Valid' padding means no padding, which may result in the spatial dimensions of the output being smaller than the input. 'Same' padding means the padding is adjusted so that the output has the same spatial dimensions as the input.

Understanding Strides

Strides refer to how much the filter moves across the input matrix on each step. A stride of 1 means it moves across and down one pixel at a time. Using a stride of more than 1 (like 2 or 3) will make the filter jump over pixels, which will result in a smaller output size and faster computation at the cost of possibly losing some information. In the case of padding, you can apply padding such that you can move the filter across the whole input including the last layer.

General Concepts

-> Feature filters learned via Backpropagation: Just like the weights in a fully connected layer, the values in the filters are learned via backpropagation. Initially, they are set to random values, but they are updated based on the gradient of the error with respect to the filter values during training.

-> Small size kernels: Small kernels such as 3x3 or 1x1 are used often in practice because they reduce the computational complexity and thus make the network faster and less memory-intensive. Furthermore, they can learn complex patterns with fewer parameters due to the hierarchical nature of CNNs.

-> Depth and resolution: As you move deeper into a CNN, typically the resolution (width and height) of your layers decreases while the number of filters (depth) increases. This is because as you move further from the input, the network tends to learn more abstract features, which can be combined in many ways (hence the increase in depth), and the exact spatial relationships matter less (hence the decrease in resolution).

-> Pooling: As I was saying, pooling layers are used to reduce the dimensions of the data. Pooling works by applying a down-sampling operation along the spatial dimensions (width, height). Common types of pooling include max pooling, which takes the largest value in the area covered by the pool size, and average pooling, which takes the average value. This has the effect of both reducing computational load and helping to make the learned representations invariant to small shifts or distortions.

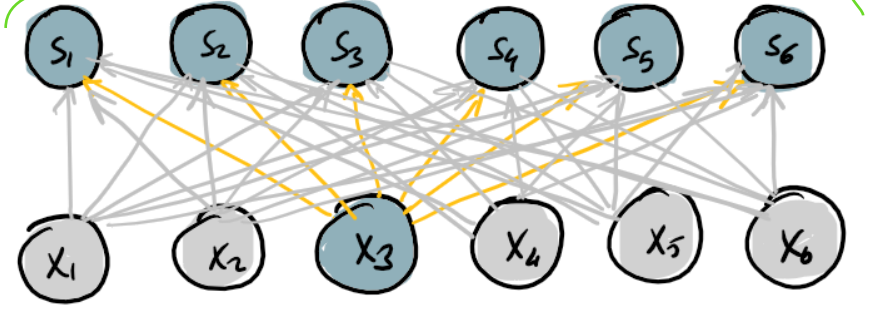
For example, if the pooling size is 2x2, max pooling would take the maximum value from the 2x2 grid it is applied to, and average pooling would take the average. The pooling layer then slides over the input in a similar manner to the convolutional layer to produce a downsampled output.

-->Shrinking and transforming: This essentially describes the process that happens as you go through the layers of a CNN. Initially, the network starts with the raw input, such as an image with pixel intensity values. As you pass through a layer (which involves convolution with filters, applying an activation function, and possibly also pooling), you "transform" this data into a different representation, and typically "shrink" its spatial dimensions.
As you go deeper into the network, the representations become more abstract and less directly connected to the raw input, but more informative about the class or feature that the network is trying to predict or detect.
--> Bias and Batch Normalization: In the context of CNNs, bias is an additional parameter that is added to the output of the convolution operation, before the activation function is applied. Having a separate bias for each filter allows the network more flexibility in fitting to the data.
Batch normalization, on the other hand, is a technique to help improve the speed, performance, and stability of neural networks. It normalizes the activations of the neurons in a layer for each batch, i.e., for each mini-batch of input, it adjusts and scales the activations to have mean 0 and variance 1.
This reduces the amount by which the distribution of activations changes during training, which can help to speed up training and reduce the likelihood of getting stuck in poor solutions. However, its use in CNNs is not as critical as in fully connected networks because convolutional layers are naturally more resistant to the issues that batch normalization helps to mitigate.

Finally, it's important to note that these descriptions represent some of the core ideas behind convolutional neural networks, but there are many additional techniques and variations that can be used. The design of a CNN can be highly problem-dependent, and there is a lot of ongoing research into how to build better networks.

Advantage #1 Sparse interactions

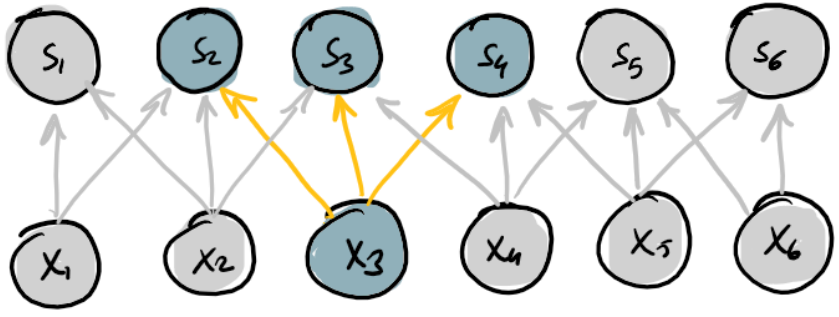
Here numbers of parameters will increase, memory requirement will increase, computational time will increase and back-propagation will be much more here.



MLP: Densly conected

$x_3 \rightarrow (s_1, \ldots s_n)$

$O(m \times n)$ runtime / example

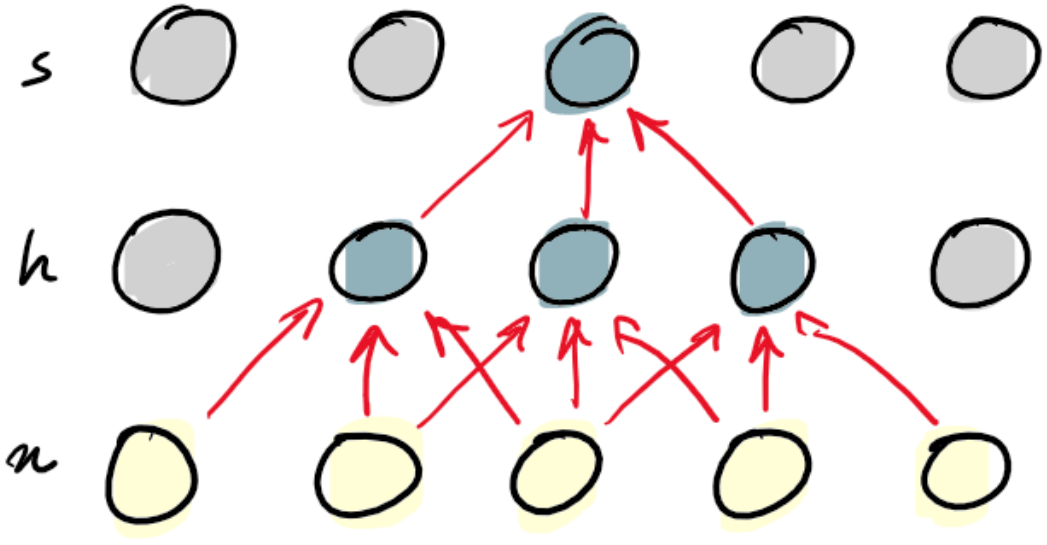


CNN $\Rightarrow$ $k = 3$ x_3 affects only (s_2, s_3, s_4)

$O(k \times n)$ runtime / example

Advantage #1 Sparse interactions

Receptive field in deeper layers

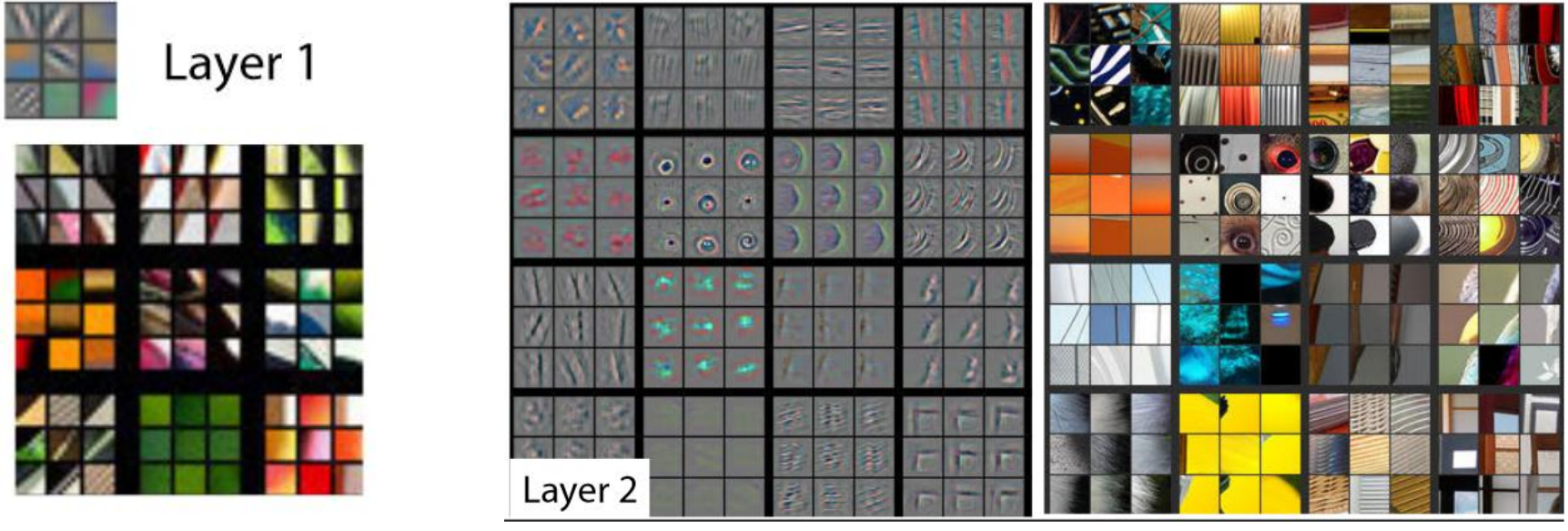


" indirectly connected to most of input layer "

Advantage #1 Sparse interactions

Receptive field in deeper layers

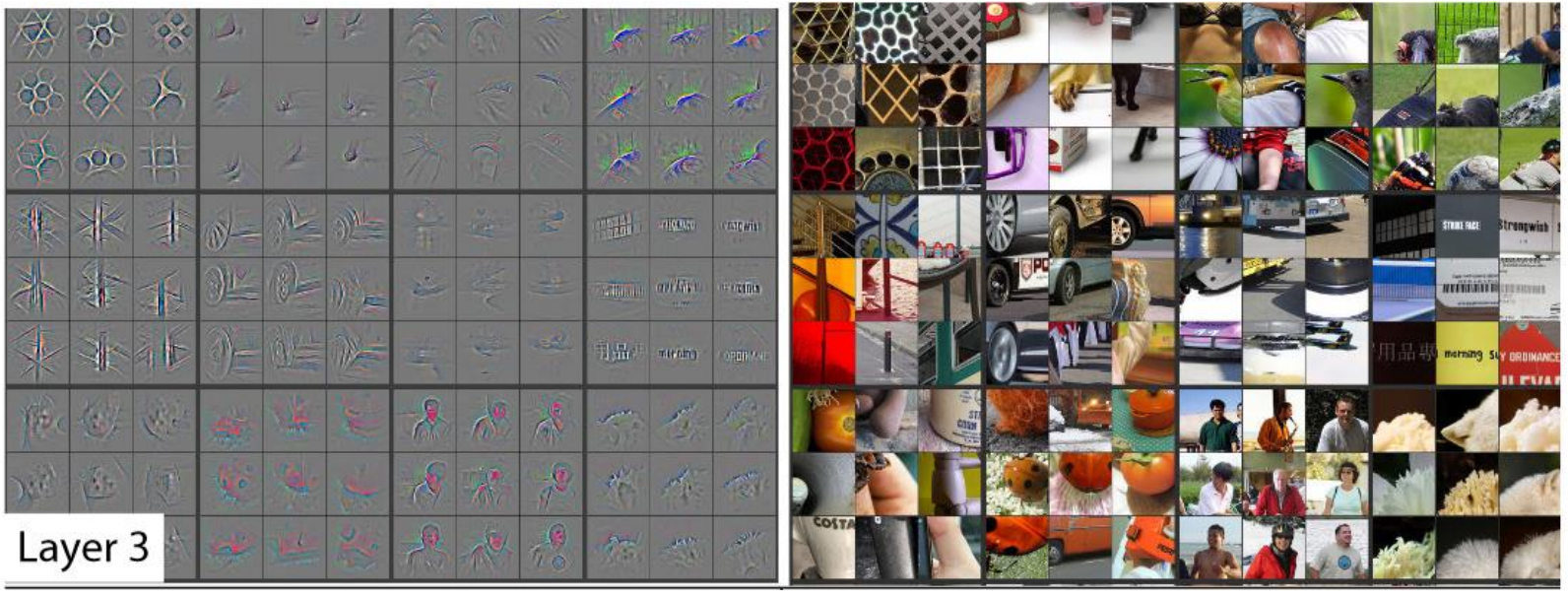
Visualizing and understanding convolutional networks



Advantage #1 Sparse interactions

Receptive field in deeper layers

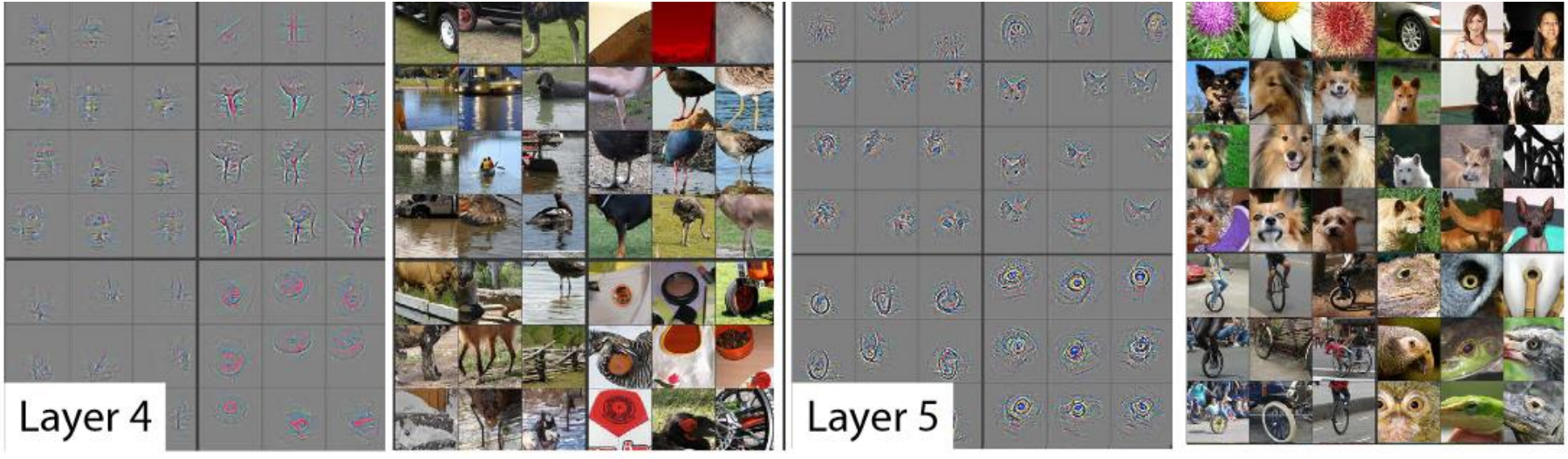
Visualizing and understanding convolutional networks



Advantage #1 Sparse interactions

Receptive field in deeper layers

Visualizing and understanding convolutional networks



Advantage # 2 Shared parameters

* In MLP; each element in matrix W is learnt & used only once

* In CNN; we learn filters (kernels)
  ↳ used at every position of the input

* ↑↑ memory eff. & statistical eff.

Eg
- Image of 280 x 320 px.
- k = 2

⇒ 280 x 319 px.

CNN; 280 x 319 x 3 operation (x x +)

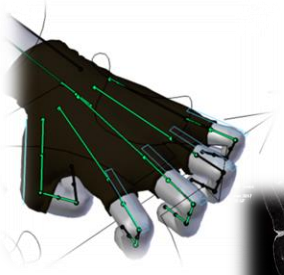
MLP; [280 x 320 x 280 x 319]

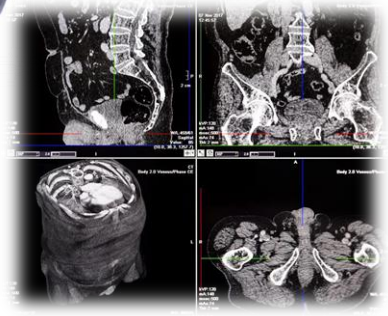
- CNN ~ 60k compt. more efficient

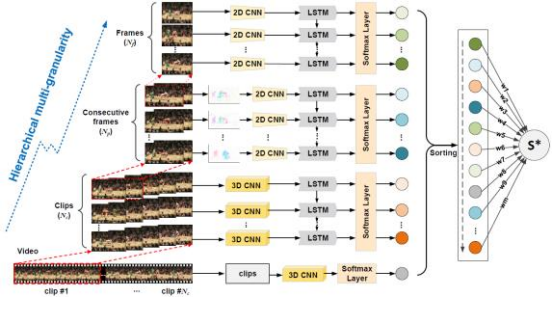
Applications of CNN :

Most common

#	Single feature	Multifeature
1D	Audio signal Univariate time-series	Joint-movement Multiv. time-series
2D	Audio + FFT (v, t) Images (intensity)	Color Image Analysis — RGB — St. Transport Phen.
3D	Medical Imaging Volumetric data	Color-video analy. Tr. Transport Phen.







Convolutional Neural Networks (CNNs) can be applied to time series data, and there are cases where this approach is very useful.

Time series data are essentially sequences of data points ordered in time, and thus, one-dimensional (1D) convolutions can be a good fit. In a 1D convolution, the convolutional kernel moves along one dimension only.

CNNs can learn to recognize patterns across time, as they do spatially within images. In the case of time series, these patterns can be trends or cycles that carry important predictive information.

There are several advantages of applying CNNs to time series data:

1. Automatic feature extraction: Just as CNNs can learn to identify useful spatial hierarchies in image data by starting with small local features (edges, colors) and gradually building them into larger, more abstract constructs (shapes, objects), the same principle can apply to time series data. For example, a network might learn to recognize local patterns like short-term trends or cyclical patterns, and then build those into higher-level features.
2. Translation invariance: A key property of CNNs is their translation invariance, which means they can learn a feature or pattern in one part of the data and recognize it elsewhere, no matter its location. This is useful in time series data where certain patterns can occur at different time intervals.
Handling multi-dimensional time series: If you have a time series that is multi-dimensional (i.e., there are multiple different quantities being measured over time), CNNs can handle this naturally, just as they handle color channels in an image.
3. Model complexity and overfitting: CNNs can be less prone to overfitting than other models when dealing with high-dimensional inputs, thanks to their parameter sharing architecture.
However, note that the applicability and effectiveness of CNNs can highly depend on the nature of the time series task at hand. For example, in cases where the series exhibit strong temporal dependencies, Recurrent Neural Networks (RNNs) and particularly their variants (like Long Short-Term Memory units, or LSTMs) can be more effective as they are specifically designed to handle such sequential data.
It's also not uncommon to see hybrid architectures (like CNN-LSTM) being used, where CNN layers are used for feature extraction, which are then fed into LSTM layers to model temporal dependencies.
99
As always in machine learning, the best approach will depend on the specifics of your dataset and the problem you're trying to solve.

The main job of a Convolutional Neural Network (CNN) is to learn and identify patterns in data. In the case of images, these patterns could be edges, curves, colors, textures, shapes, etc. In the case of time-series data, these patterns could be trends, seasonality, cycles, etc.

The concept of feature extraction is central to this learning process. A feature in this context is any measurable property or characteristic of the data being observed. For example, in an image, a feature might be a particular shape or pattern of colors. In a time series, a feature might be a repeating pattern of peaks and valleys, indicating some form of cyclical behavior.

CNNs perform feature extraction through their convolutional layers, where they learn to identify these local patterns in the input data. Here's a simplified explanation of how this works in the context of image analysis:

1. Convolutional Layer: The input image (or part of it) is passed through a set of filters or kernels, each of which is designed to detect a specific type of feature (e.g., an edge in a certain direction, a certain texture, etc.). Each filter is essentially a small matrix of weights that gets updated during the training process. The result of this convolution process is a set of feature maps, which are essentially transformed versions of the original image, emphasizing the detected features.

2. Activation Function: The output of the convolution is then passed through a non-linear activation function, usually a ReLU (Rectified Linear Unit). This introduces non-linearity into the model, enabling it to learn complex patterns.

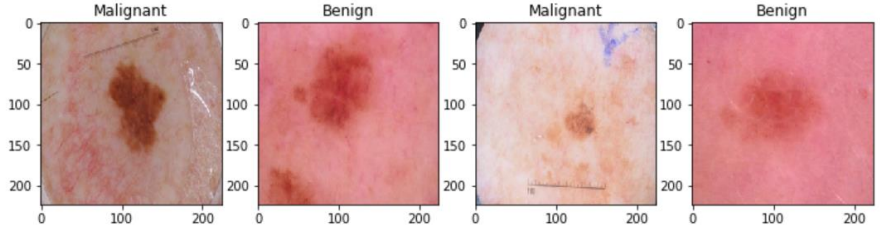
3. Pooling Layer: After the activation function, we often find a pooling operation (like max pooling or average pooling), which reduces the spatial dimensions (i.e., width and height) of the input volume. This operation helps to make the model more computationally efficient and control overfitting, while retaining the important features.

The output of the convolutional layers is a high-level representation of the input data, which emphasizes the features that the model has learned as being important for the task at hand. This high-level representation is then typically passed to one or more fully connected layers, which handle the task of making the final prediction or classification.

To sum up, the convolutional layers of a CNN are responsible for automatically learning useful features from the input data, which can then be used for the task the model is designed to perform. This is often more effective (and computationally efficient) than manually extracting features, as the model can learn to identify features that are specifically useful for its task.

Case study : Melanoma classification

- Annual case ~ ↑ 53% btw. 2008 - 2018
  - → UV exposure
- Diagnosis → visual examination by a dermatologist
  - → 65 ~ 80% accuracy
- CNN for binary classification

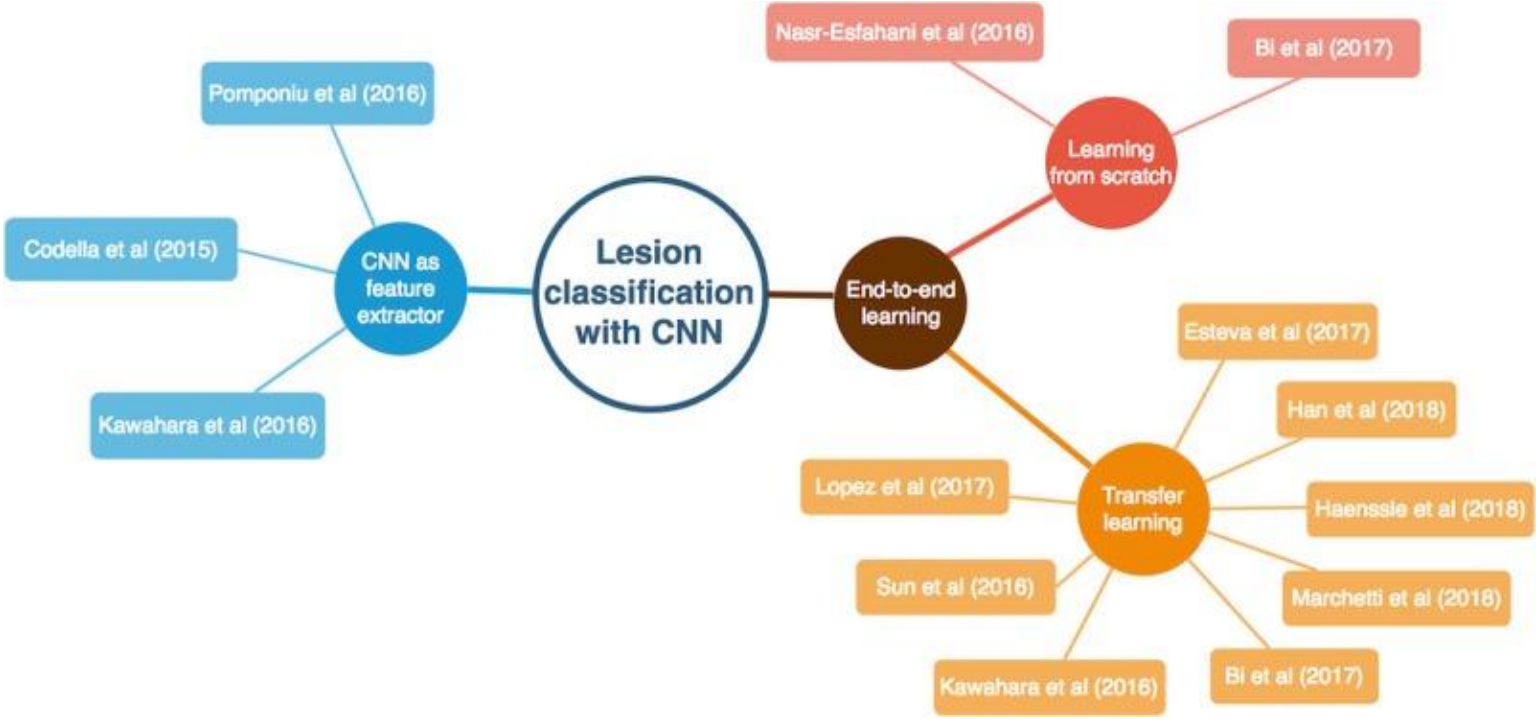


EARLY DETECTION MAKES A DIFFERENCE

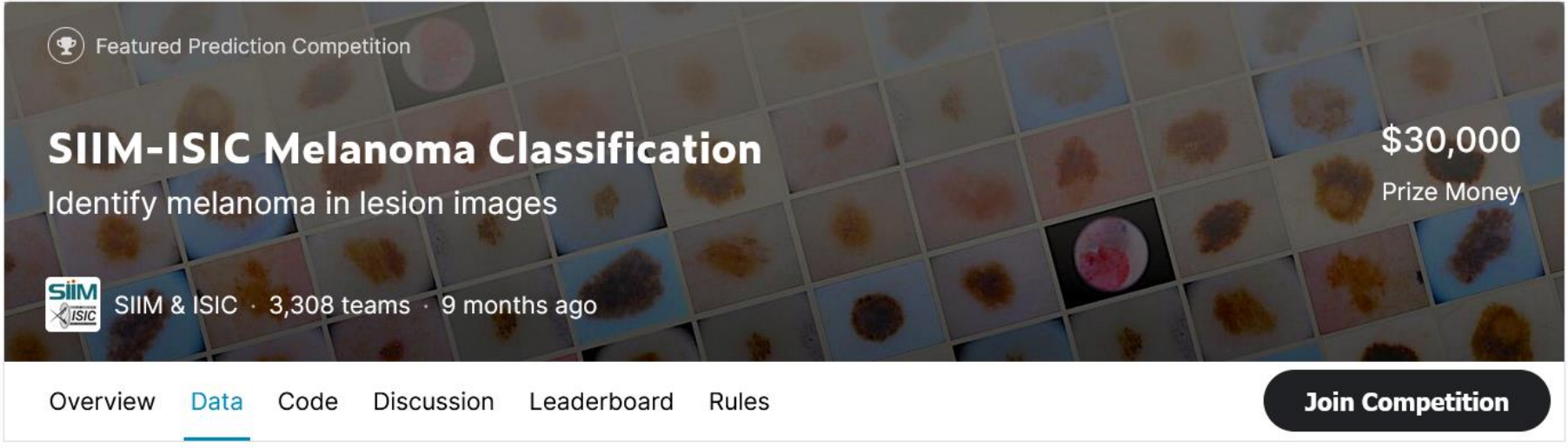
99%

5-year survival rate for patients in the U.S. whose melanoma is detected early. The survival rate drops to 66% if the disease reaches the lymph nodes and 27% if it spreads to distant organs.

Case study : Melanoma classification



an updated database



Outline of a ML Project

Basic Steps to Follow:

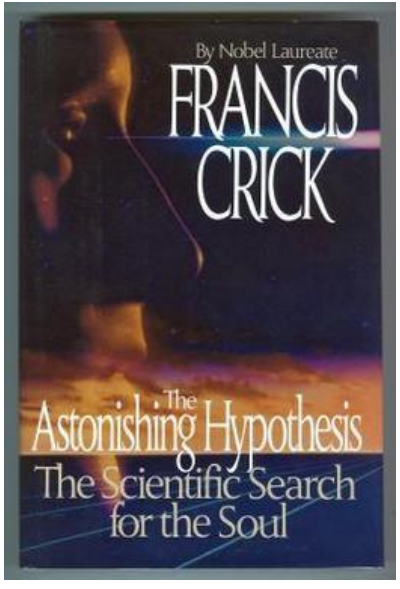
0.) Understand the business/task.
1.) Understand the data.
2.) Explore & prepare the data.
3.) Shortlist candidate models.
4.) Training the model
5.) Evaluate the model predictions.
6.) "Serve" the model!

} "Classification"

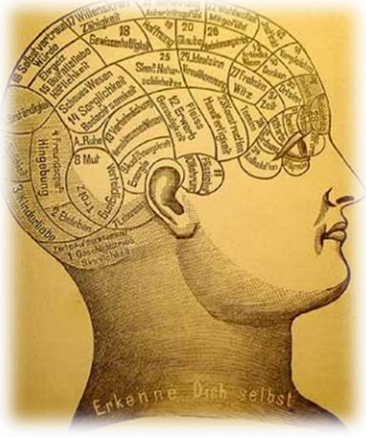
We will use a reduce version of the data of the previous file.

Ateliers & Saveurs in Montreal

colab



A Walk-through of the Mammalian Visual System



Imagery Debate: The Role of the Brain

Filters: Filters, also known as kernels, in convolutional neural networks (CNNs) are small matrices that move over the input data (such as an image) to perform a convolution operation. Each filter is used to detect a specific feature in the input, such as edges, corners, etc. For instance, a filter might be designed to detect vertical edges in an image by highlighting areas of the image with strong vertical contrast.

Padding: Padding is the process of adding extra pixels around the edge of the input image. It is used to preserve the spatial dimensions (height and width) of the input through the convolutional layer. This is particularly useful when you want your output image to have the same dimensions as the input image. There are two common types of padding: 'valid' and 'same'. 'Valid' padding means no padding (the input is left as is), and 'same' padding means padding is added so that the output has the same width and height as the original input.

Pooling: Pooling is a downsampling operation that reduces the size of the feature maps while retaining the most important information. This helps to reduce computational complexity and to control overfitting. The pooling layer summarizes the features present in a region of the feature map generated by a convolution layer. The two most common types of pooling are Max Pooling and Average Pooling. Max Pooling takes the maximum value in the pooling window, while Average Pooling takes the average.
Dropout: Dropout is a regularization technique used to prevent overfitting in neural networks. During training, randomly selected neurons in a layer are "dropped out" or temporarily removed along with all of their incoming and outgoing connections. This means that the neuron will not participate in the forward pass or backpropagation for that specific training iteration. This helps to make the model more robust and less reliant on any specific neurons, thereby preventing overfitting.
These components are often combined in the architecture of a convolutional neural network. A typical pattern might involve one or more convolutional layers (with filters and potentially padding) to detect features, followed by a pooling layer to reduce dimensionality. This pattern can be repeated multiple times to form a deep network. Dropout layers can be inserted in between these layers to control overfitting.

For example:

The input image is passed through a Convolutional layer with a certain number of filters.
Padding might be applied during the convolutional operation if we don't want to lose information at the borders of the image.
The resulting feature maps are then passed through a Pooling layer, which reduces their dimensions.
A Dropout layer could then be applied to randomly remove a fraction of the neurons in the network to prevent overfitting.
This pattern can be repeated multiple times to extract more complex features at each layer.
At the end, a fully connected layer (or layers) is usually used for classification or regression.