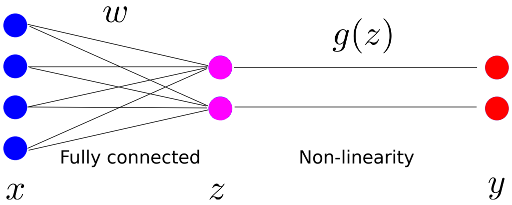
Convolutional Neural Networks

Alasdair Newson

LTCI, Télécom Paris, IP Paris
alasdair.newson@telecom-paris.fr

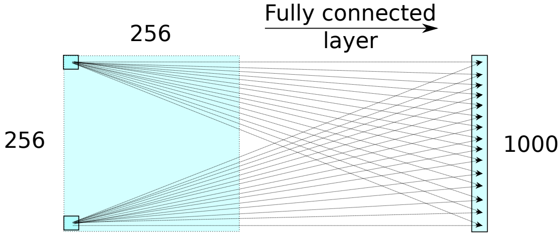
Introduction

- Neural networks provide a highly flexible way to model complex dependencies and patterns in data
- In the previous lessons, we saw the following elements :
  - MLPs : fully connected layers, biases
  - Activation functions : sigmoid, soft max, ReLU
  - Optimisation : gradient descent, stochastic gradient descent
  - Regularisation : weight decay, dropout, batch normalisation
  - RNNs : for sequential data



Introduction

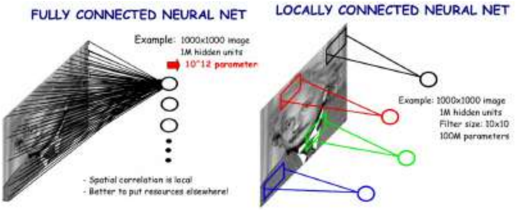
- In MLPs each layer of the network contained **fully connected** layers
- Unfortunately, there are great drawbacks with such an approach



- Each hidden unit is connected to each input unit
- There is high redundancy in these weights :
  - In the above example, 65 million weights are required

Introduction

- For many types of data with **grid-like topological structures** (eg. images), it is not necessary to have so many weights
- For these data, the **convolution** operation is often extremely useful
- Reduces the number of parameters to train
  - Training is faster
  - Convergence is easier : smaller parameter space



- A neural network with convolution operations is known as a **Convolutional Neural Network** (CNN)

Introduction - some history

- "Neocognitron" of Fukushima* : first to incorporate notion of receptive field into a neural network, based on work on animal perception of Hubert and Weisel†
- Yann LeCun first to propose **back-propagation** for training convolutional neural networks‡
  - Automatic learning of parameters instead of hand-crafted weights
  - However, training was very long : required 3 days (in 1990)

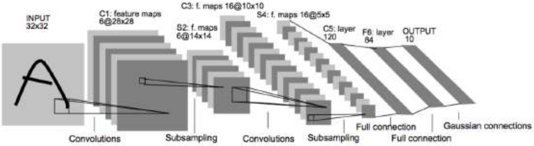


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

* ***Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position***, *Fukushima, K., Biological Cybernetics, 1980*

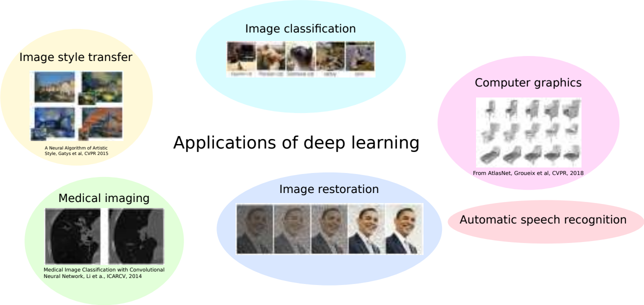
† ***Receptive fields and functional architecture of monkey striate cortex***, *Hubel, D. H. and Wiesel, T. N, 1968*

‡ ***Backpropagation Applied to Handwritten Zip Code Recognition***, *LeCun, Y. et al., AT&T Bell Laboratories*

Introduction - some history

- In the years 1998-2012, research continued on shallow and deep neural networks, but other machine learning approaches were preferred (GMMs, SVMs etc.)
- In 2012, Alex Krizhevsky et al. used **Graphics Processing Units** (GPUs) to carry out backpropagation on a very deep convolutional neural network
  - Greatly outperformed classic approaches in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
- GPUs turned out to be very efficient for training neural nets (lots of parallel computations)
- Signalled the beginning of deep learning revolution

- Since 2012, CNNs have completely revolutionised many domains
- CNNs produce competetive/best results for most problems in image processing and computer vision



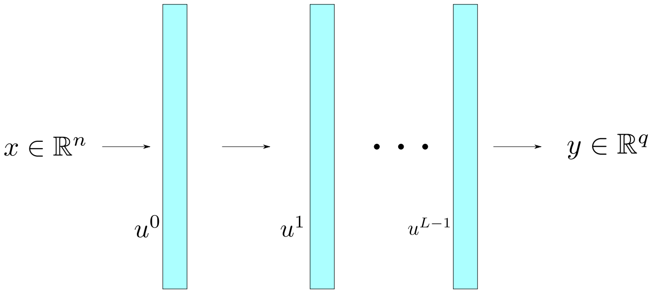
- Being applied to an ever-increasing number of problems

Summary

1. Introduction, notation
2. Convolutional Layers
3. Down-sampling and the receptive field
4. CNN details and variants
5. CNNs in practice
6. Image datasets, well-known CNNs, and applications
   - Applications of CNNs
7. Interpreting CNNs
   - Visualising CNNs
   - Adversarial examples

Notations

- $x \in \mathbb{R}^n$: input vector
- $y \in \mathbb{R}^q$: output vector
- u_ℓ : feature vector at layer ℓ
- θ_ℓ : network parameters at layer ℓ



Neural network with L layers

Introduction

- A "**Convolutional Neural Network**" (CNN) is simply a concatenation of :
  1. Convolutions (filters)
  2. Additive biases
  3. Down-sampling ("Max-Pooling" etc.)
  4. Non-linearities

- In this lesson, we will be mainly concentrating on **convolutional** and **down-sampling** layers

Summary

1. Introduction, notation
2. Convolutional Layers
3. Down-sampling and the receptive field
4. CNN details and variants
5. CNNs in practice
6. Image datasets, well-known CNNs, and applications
   - Applications of CNNs
7. Interpreting CNNs
   - Visualising CNNs
   - Adversarial examples

Convolution operator

Let f and g be two integrable functions. The **convolution operator** $*$ takes as its input two such functions, and outputs another function $h = f * g$, which is defined at any point $t \in \mathbb{R}$ as :

$$h(t) = (f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau.$$

- Intuitively, the function h is defined as the inner product between f and a *shifted* version of g

Convolutional Layers

- In many practical applications, in particular for CNNs, we use the **discrete convolution** operator, which acts on discretised functions;

Discrete convolution operator

Let f_n and g_n be two summable series, with $n \in \mathbb{Z}$. The discrete convolution operator is defined as :

$$(f * g)(n) = \sum_{i=-\infty}^{+\infty} f(i)g(n-i)$$

- Intuitively, the function h is defined as the inner product between f and a *shifted* version of g
- In practice, the filter is of small spatial support, around 3×3, or 5×5
- Therefore, only a **small number** of parameters need to be trained (9 or 25 for these filters)

Properties of convolution

1. Associativity : $(f * g) * h = f * (g * h)$
2. Commutativity : $f * g = g * f$
3. Bilinearity : $(\alpha f) * (\beta g) = \alpha\beta(f * g)$, for $(\alpha, \beta) \in \mathbb{R} \times \mathbb{R}$
4. Equivariance to translation : $(f * (g + \tau))\,(t) = (f * g)(t + \tau)$

Associativity, commutativity

- Associativity+commutativity implies that we can carry out convolution in any order
- There is no point in having two or more consecutive convolutions
    - This is true in fact for any linear map

Equivariance to translation

- Equivariance implies that the convolution of any shifted input $(f+\tau)*g$ contains the **same information** as $f*g$ †
- This is useful, since we want to detect objects **anywhere in the image**

† if we forget about border conditions for a moment

Convolutional Layers - 2D Convolution

- Most often, we are going to be working with **images**
- Therefore, we require a 2D convolution operator : this is defined in a very similar manner to 1D convolution :

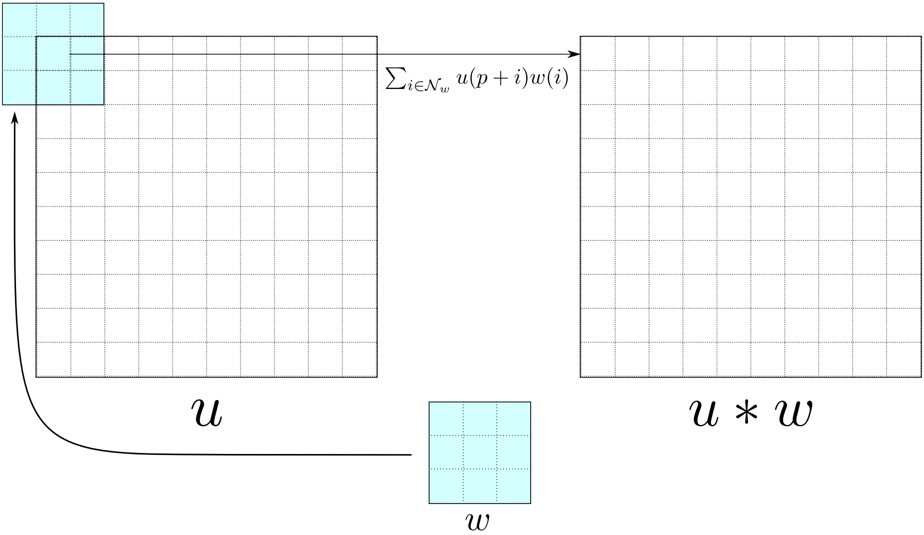
2D convolution operator

$$(f * g)(s,t) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f(i,j)g(s-i, t-j)$$

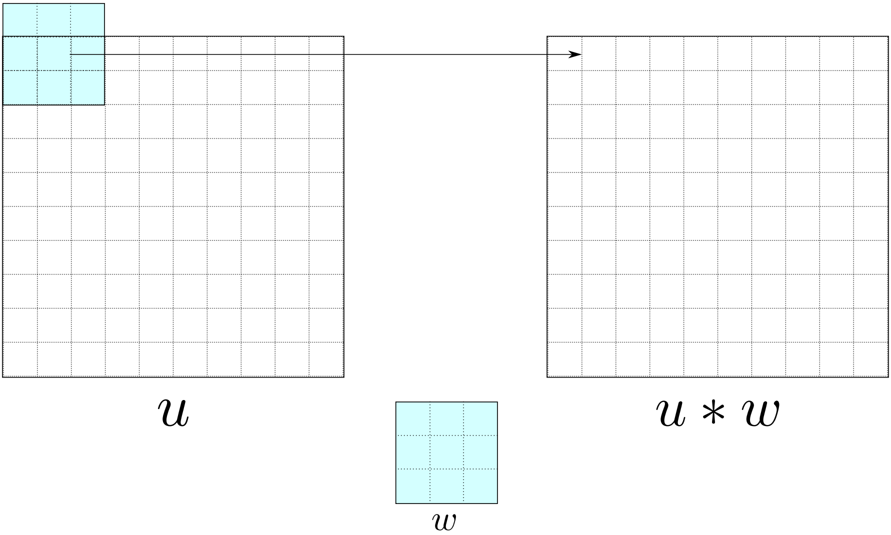
Important remarks for the rest of the lesson!

- We are going to denote the **filters with** w
- For lighter notation, we write $w(i) =: w_i$ (and the same for x_i etc.)

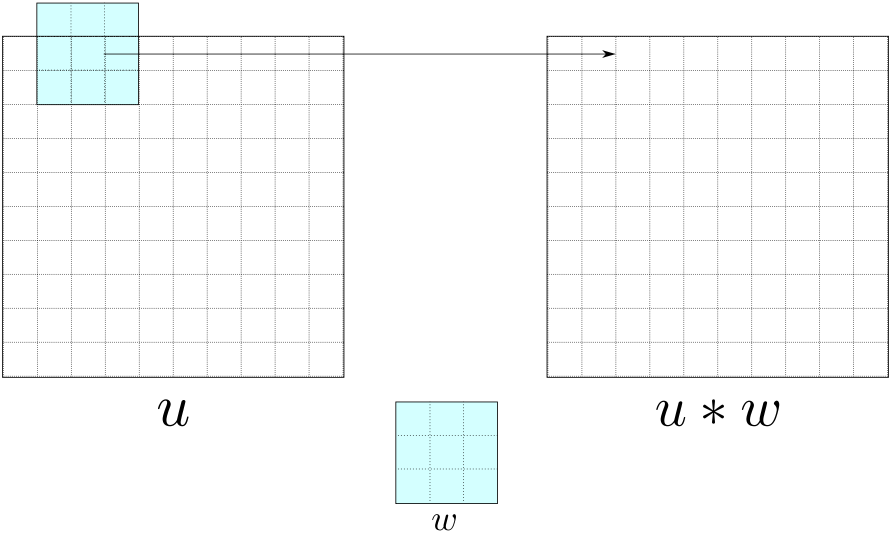
Convolutional Layers : Visual Illustration



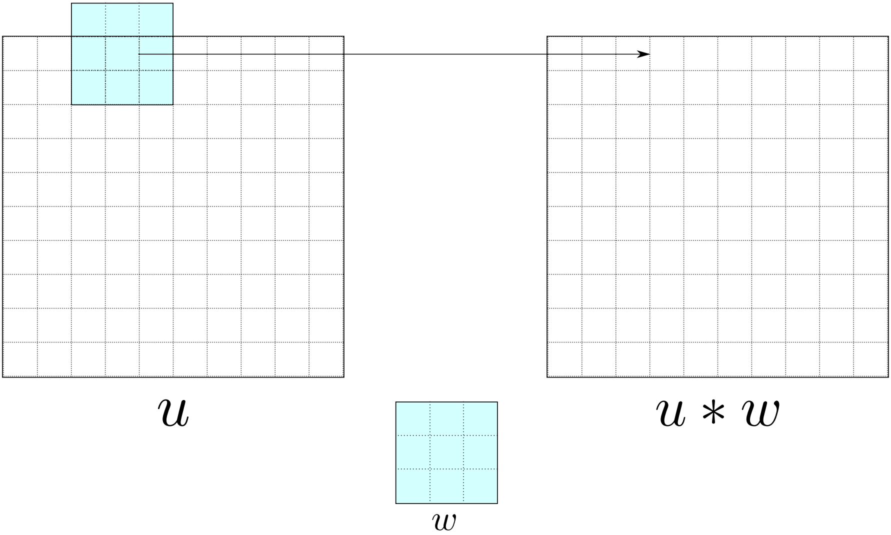
Convolutional Layers : Visual Illustration



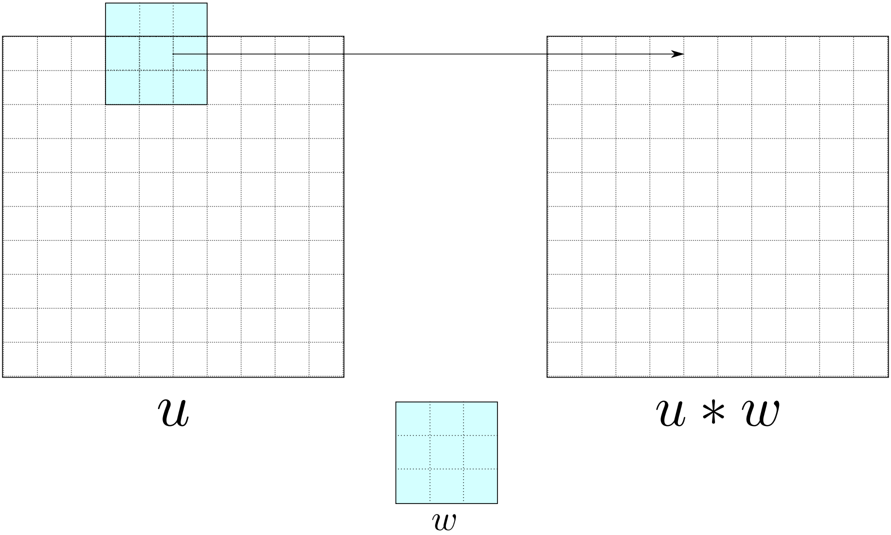
Convolutional Layers : Visual Illustration

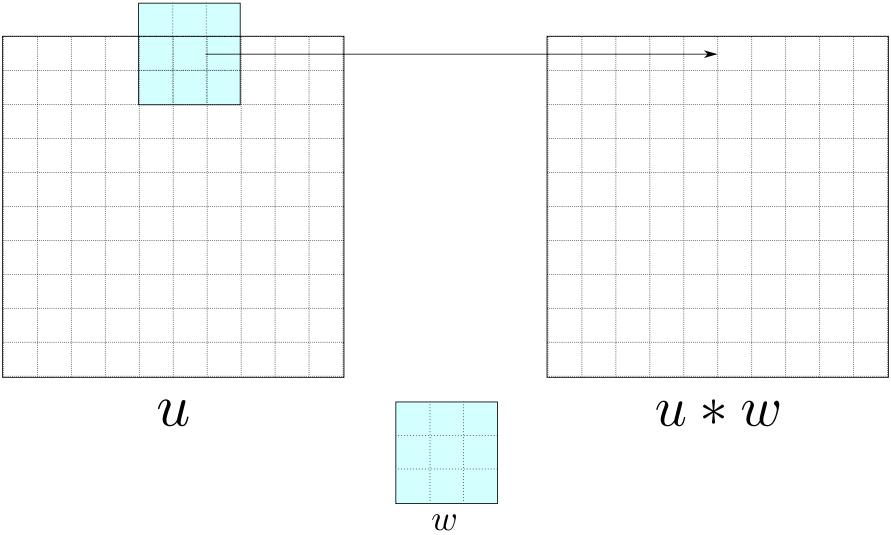


Convolutional Layers : Visual Illustration

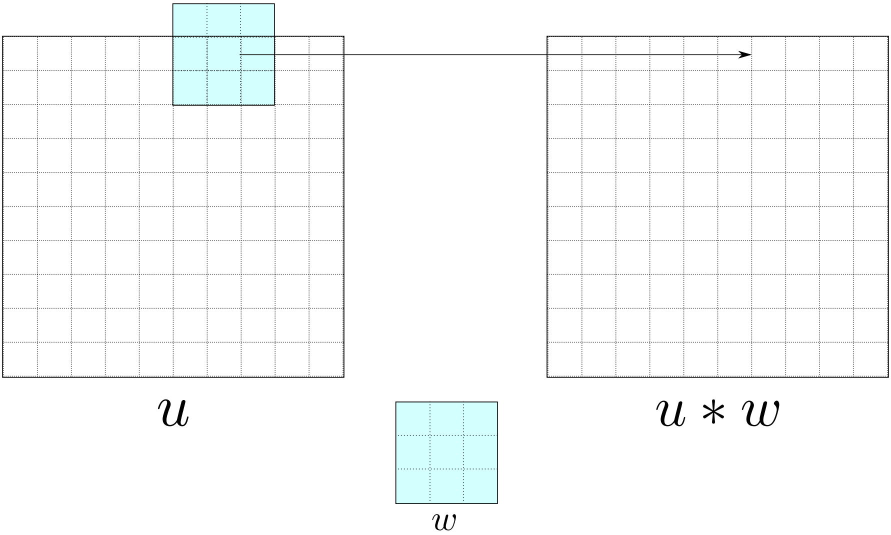


Convolutional Layers : Visual Illustration

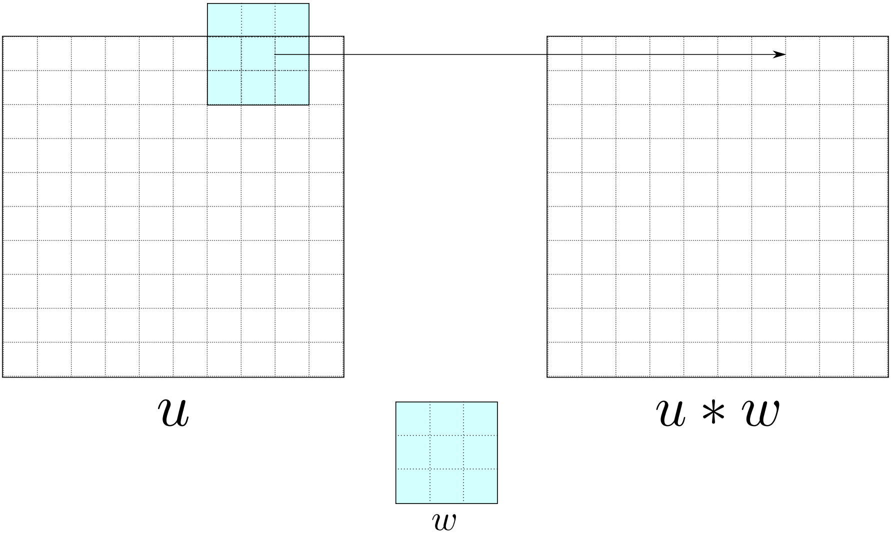



u
$u * w$
w

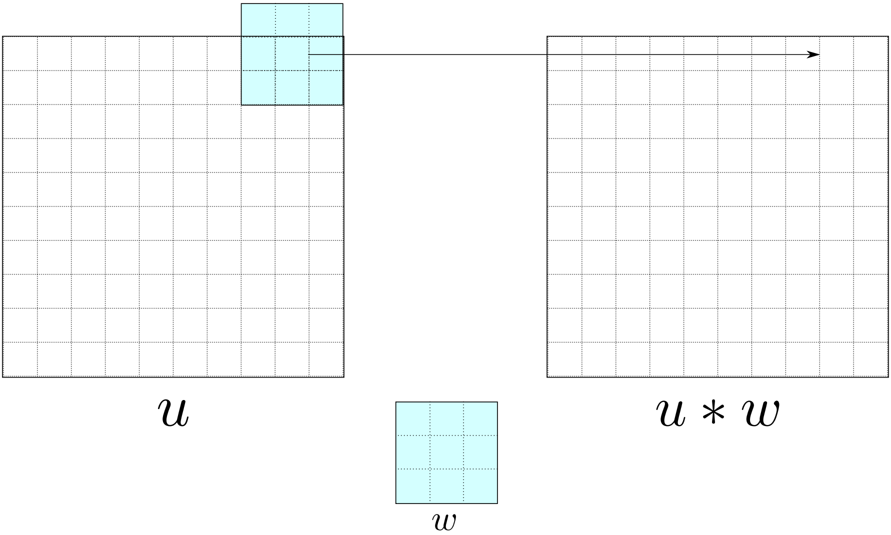
Convolutional Layers : Visual Illustration

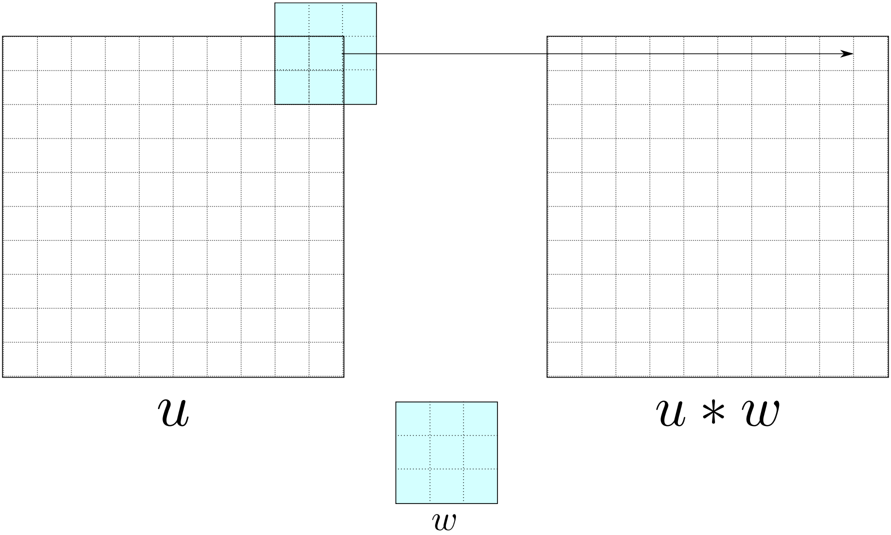


Convolutional Layers : Visual Illustration

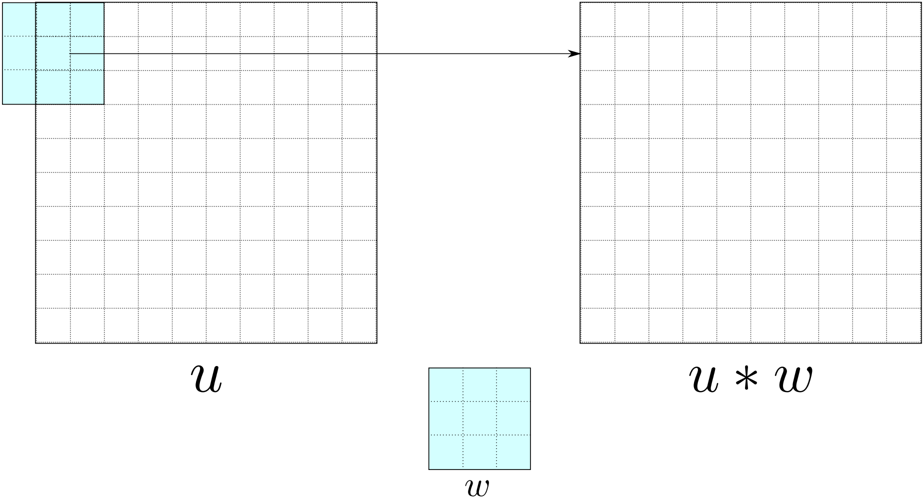


Convolutional Layers : Visual Illustration

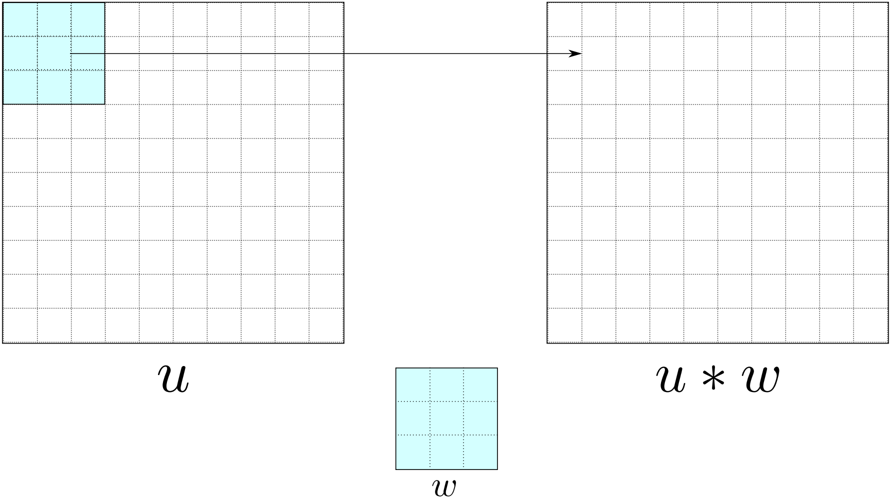


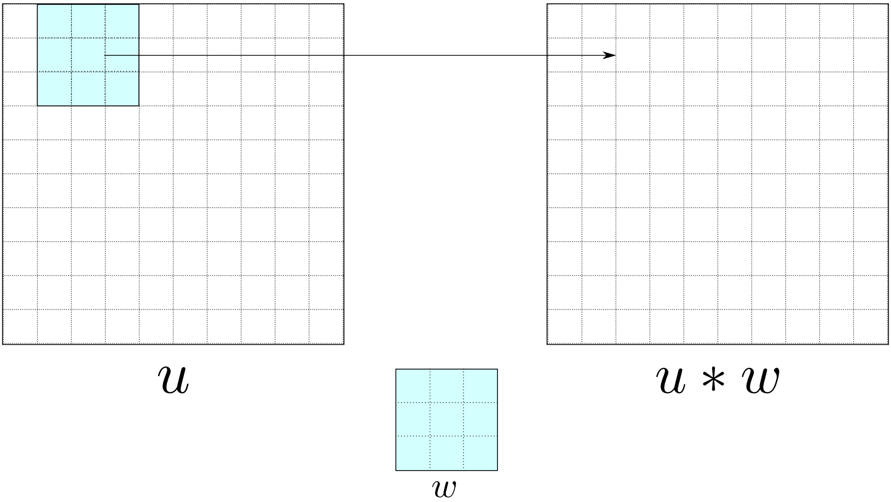

u
w
$u * w$

Convolutional Layers : Visual Illustration



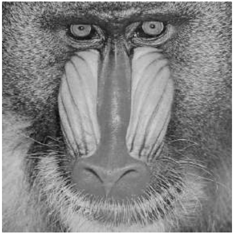
Convolutional Layers : Visual Illustration





u
$u * w$
w

- The filter weights w_i determine what type of “feature” can be detected by convolutional layers;
- Example, sobel filters :



Horizontal edge

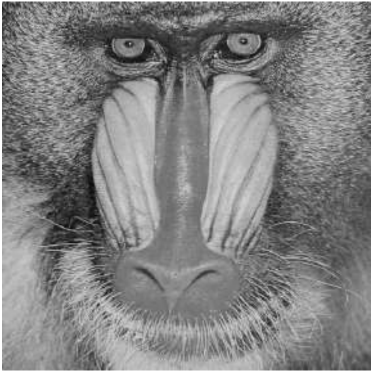
$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Vertical edge

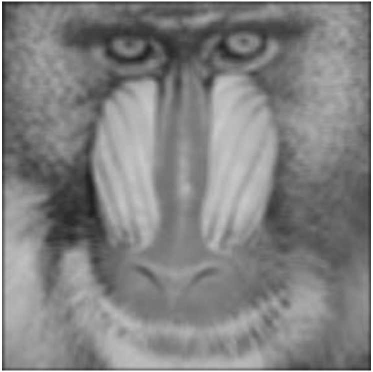
$$\begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & -2 \\ -1 & 0 & 1 \end{bmatrix}$$

Convolutional Layers

- Convolutional filters can also act as low-pass/smoothing filters



Input image



Low-pass filtered image

Convolutional Layers

- We can also write convolution as a matrix/vector product, as in the case of fully connected layers

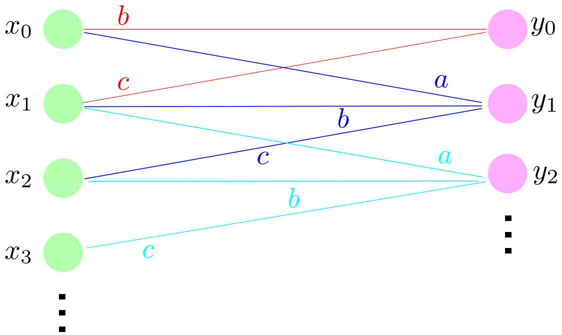
Example : discrete Laplacian operator

$$w = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \rightarrow A_w = \mathsf{K} \Big\downarrow \overset{\xrightarrow{\hspace{8em}\mathsf{n}\hspace{8em}}}{\begin{pmatrix} 4 & -1 & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} & & \\ -1 & 4 & -1 & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} & \\ 0 & -1 & 4 & -1 & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ -1 & & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} & -1 & 4 \end{pmatrix}}$$

- This further illustrates the drastic reduction in weight parameters (9 instead of Kn)
- Can be useful to view convolution in this manner (we will see this later)

Convolutional Layers

- At this point, it is good to have a more "neural network"-based illustration of CNNs



- We can see two of the main justifications for CNNs
  1. Sparse connectivity
  2. Weight sharing

Convolutional Layers

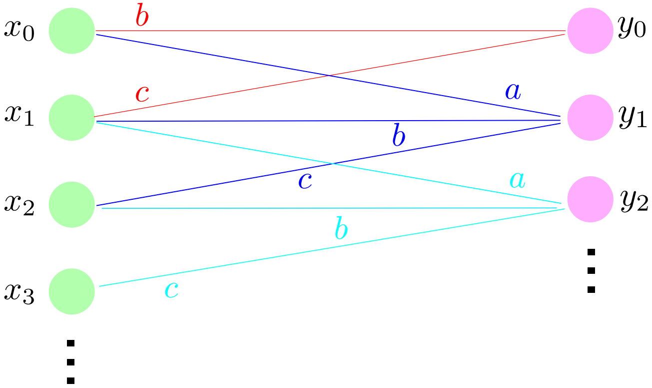
- Now that we understand convolution, how do we **optimize** a neural network with convolutional layers ? **Back-propagation**
- Consider a layer with just a convolution with w

$$x \xrightarrow{\quad x * w \quad} y$$

- We have the derivatives $\frac{\partial \mathcal{L}}{\partial y_i}$ available
- We want to calculate the following quantities :
  - $\frac{\partial \mathcal{L}}{\partial x_k}$ (for further back-propagation) and
  - $\frac{\partial \mathcal{L}}{\partial w_k}$
- We shall use the abbreviation $\frac{\partial \mathcal{L}}{\partial y_i} =: dy_i$

Convolutional Layers

- Before considering the general case, let's take an example from the illustration from above



- Say we want to calculate $dx_1 := \frac{\partial \mathcal{L}}{\partial x_1}$

Convolutional Layers

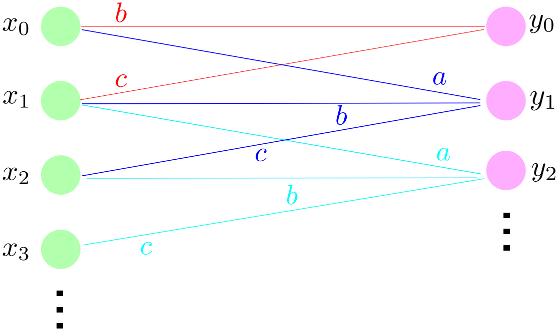
- Each element y_i depends on the input x_i and the weight w_k
- Therefore, we can consider that the loss is a function of several variables :

$$\mathcal{L} = f\left(x_1, \ldots, x_n, w_1, \ldots, w_K, y_1(x., w.), \ldots, y_m(x., w.)\right)$$

- We use the multi-variate chain rule

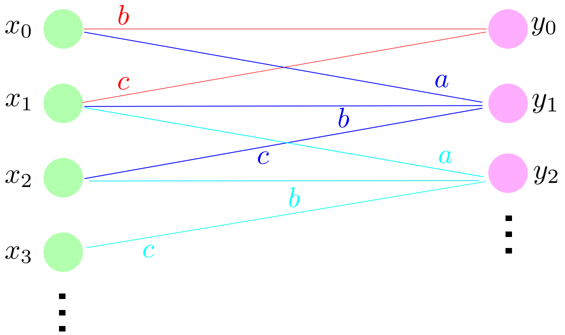
$$dx_1 = \sum_i \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial x_1}$$

Convolutional Layers



$$dx_1 = ???$$

Convolutional Layers



$$dx_1 = dy_0 \frac{\partial y_0}{\partial x_1} + dy_1 \frac{\partial y_1}{\partial x_1} + dy_1 \frac{\partial y_2}{\partial x_1} = dy_0 c + dy_1 b + dy_2 a$$

- As we can see, the order of the weights is flipped

Convolutional Layers

- Now, let us calculate $\frac{\partial \mathcal{L}}{\partial x_k}$ for any k

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_k} &= \sum_i dy_i \frac{\partial y_i}{\partial x_k} \\
&= \sum_i dy_i \frac{\partial (x * w)_i}{\partial x_k} \\
&= \sum_i dy_i \frac{\partial \left(\sum_j x_j w_{i-j} \right)}{\partial x_k} \\
&= \sum_i dy_i w_{i-k} = \sum_i dy_i w_{-(k-i)}
\end{aligned}
$$

multi-variate chain rule

- More compactly : $dx_k = (dy * \text{flip}(w))_k$

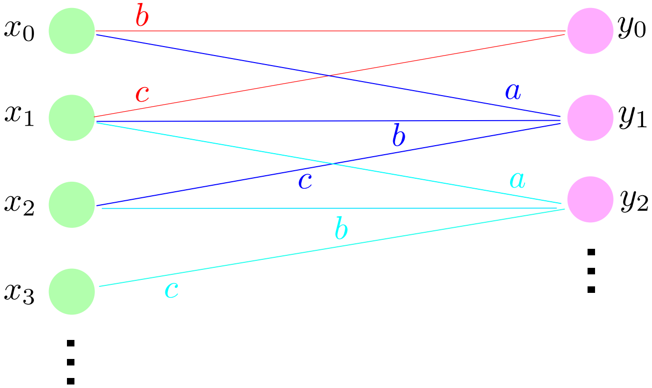
Convolutional Layers

- Recall that the convolution operator can be written $y = A_w x$, with A_w the convolution matrix
- The flipping of the weights corresponds to a transpose of A

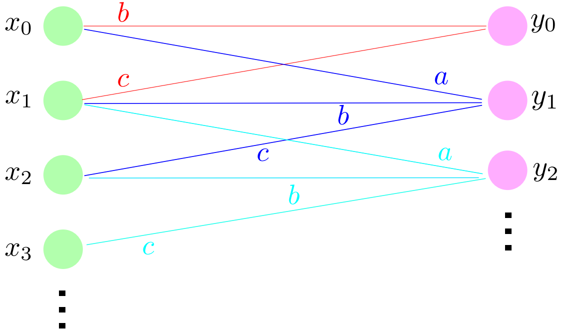
$$dx = {A_w}^\intercal dy \tag{1}$$

- This gives an easy method of backpropagation in convolutional layers
  - Although you will not actually have to implement this

- Now for the second part : $\frac{\partial \mathcal{L}}{\partial w_k}$



- Again, we use the chain rule. For example $da = \sum_i \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial a}$



- We have $y_i = ax_{i-1} + bx_i + cx_{i+1}$

$$da = \sum_i dy_i \; x_{i-1}$$

- In the general case, we have:

$$\frac{\partial \mathcal{L}}{\partial w_k} = \sum_i dy_i \frac{\partial y_i}{\partial w_k} \qquad \text{multi-variate chain rule}$$

$$= \sum_i dy_i \frac{\partial (x * w)_i}{\partial w_k}$$

$$= \sum_i dy_i \frac{\partial \left(\sum_j x_j w_{i-j} \right)}{\partial w_k}$$

$$= \sum_i dy_i x_{i-k} = \sum_i dy_i x_{-(k-i)} \qquad k = i - j$$

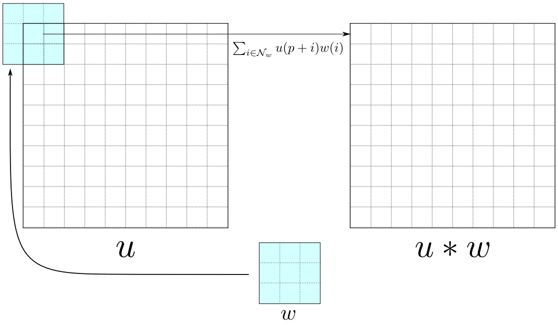
- More compactly : $dw_k = (dy * \text{flip}(x))_k$

Convolutional Layers

- Note : optimisation of loss w.r.t one parameter w_k **involves entire image**
- **Weights are "shared" across the entire image**
- This notion of **weight sharing** is one of the main justifications of using CNNs
- In practice, we do not calculate dw_k and dx_k ourselves, we use the **automatic differentiation** tools of Tensorflow, Pytorch etc.

Convolutional Layers - border conditions

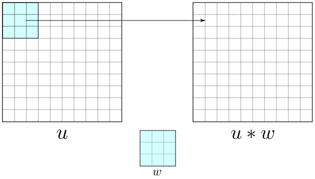
- The convolution operator poses a problem at the borders
- Theoretically, we consider functions defined over an infinite domain, but which have compact support
- In reality, we only have finite vectors/matrices to work on



Two common approaches to border conditions

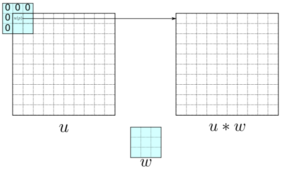
"VALID" approach

- Only take shift/dot products that do not extend beyond $\mathrm{Supp}(u)$
- Output size : $m - |w| + 1$



"SAME" approach

- Keep output size m
- Need to choose values outside of $\mathrm{Supp}(u)$: zero-padding



2D+feature convolution

- Several filters are used per layer, let us say K filters : $\{w_1, \dots, w_K\}$
- The resulting vectors/images are then stacked together to produce the next layer's input $u^{\ell+1} \in \mathbb{R}^{m \times n \times K}$

$$u^{\ell+1} = [u * w_1, \dots, u * w_K]$$

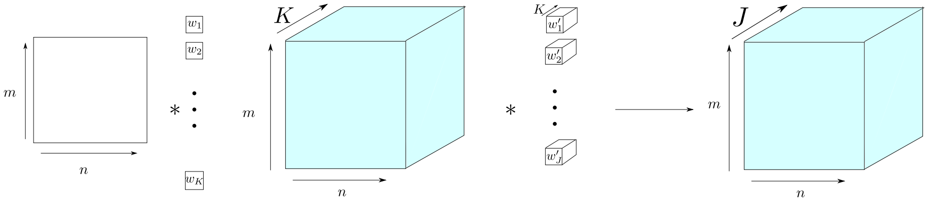
- Therefore, the next layer's weights must have a depth of K. The 2D convolution with an image of depth K is defined as

$$(u * w)_{y,x} = \sum_{i,j,k} u(i,j,\mathbf{k})\, w(y-i, x-j, \mathbf{k})$$

Useful explanation : *https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215*

Convolutional layers

- Illustration of several consecutive convolutional layers with different numbers of filter

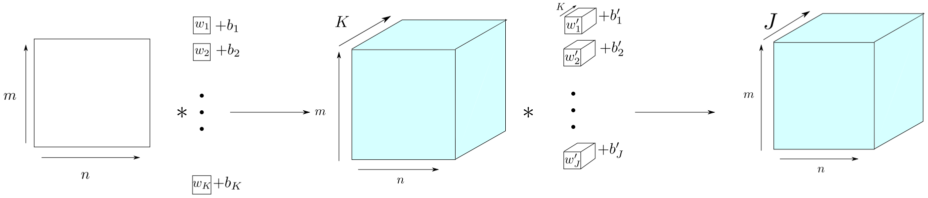


- Each layer contains “image” with a depth, where each channel corresponds to a different filter response
- Each layer is a concatenation of several features : rich information

Useful explanation : *https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215*

Convolutional layers - a note on Biases

- A note on biases in neural networks : **each output layer is associated with one bias**
- There is **not** one bias per pixel
- This is coherent with the idea of **weight sharing** (bias sharing)



Convolutional Layers

- In many cases, we are primarily interested in **detection**;
- We would like to detect objects **wherever they are in the image**

- Formally, we would like to have some **shift invariance property**;
- This is done in CNNs by using **subsampling**, or some variant :
  - Strided convolutions
  - Max pooling
- We explain these now

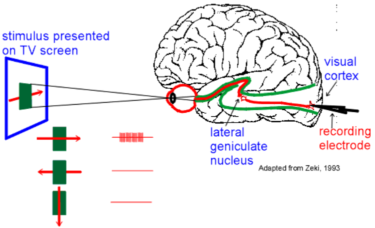
Summary

1. Introduction, notation
2. Convolutional Layers
3. Down-sampling and the receptive field
4. CNN details and variants
5. CNNs in practice
6. Image datasets, well-known CNNs, and applications
   - Applications of CNNs
7. Interpreting CNNs
   - Visualising CNNs
   - Adversarial examples

Down-sampling and the receptive field

The Receptive Field

- Neural networks were initially inspired by the brain's functioning
- Hubel and Weisel† showed that the visual cortex of cats and monkeys contained cells which individually responded to different small regions of the visual field
- The region which an individual cell responds to is known as the "receptive field" of that cell



† ***Receptive fields and functional architecture of monkey striate cortex***, *Hubel, D. H.; Wiesel, T. N, 1968 Illustration from :* `http://www.yorku.ca/eye/cortfld.htm`

- This idea was imitated in convolutional neural networks by adding **down-sampling** operations

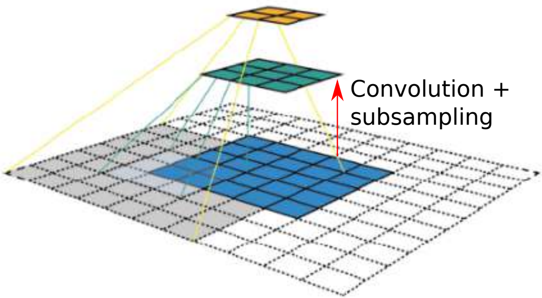


Illustration from : Applied Deep Learning, Andrei Bursuc, https://www.di.ens.fr/~lelarge/dldiy/slides/lecture_7/

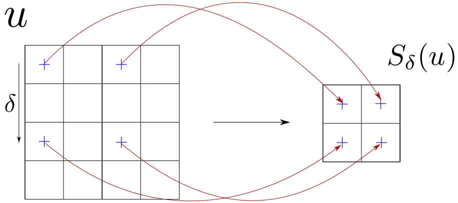
Strided convolution

- **Strided convolution** is simply convolution, followed by **subsampling**

Subsampling operator (for 1D case)

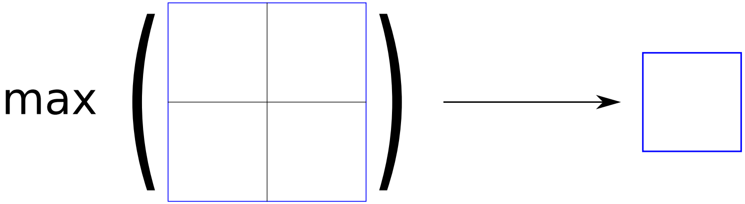
Let $x \in \mathbb{R}^n$. We define the subsampling step as $\delta > 1$, and the subsampling operator $S\delta : \mathbb{R}^n \rightarrow \mathbb{R}^{\frac{n}{\delta}}$, applied to x, as

$$S_\delta(x)\ (t) = x(\delta t),\ \ \text{for}\ t = 0 \ldots \frac{n}{\delta} - 1$$



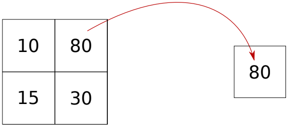
Max pooling

- **Max pooling** subsampling consists in taking the **maximum** value over a certain region
- This maximum value is the new subsampled value
- We will indicate the max pooling operator with S_m

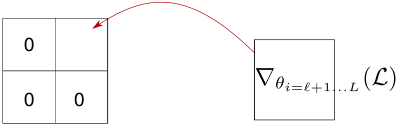


Max pooling

- Back propagation of max pooling only passes the gradient through the **maximum**



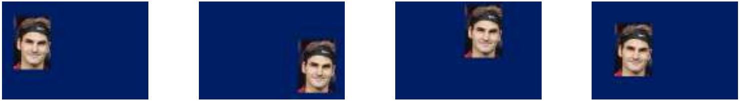
Max pooling



Back propagation

Down-sampling

- Conclusion : cascade of convolution, non-linearities and subsampling produces **shift-invariant** classification/detection
- We can detect Roger wherever he is in the image !



$u * w$

Convolution + non-linearity +max pooling

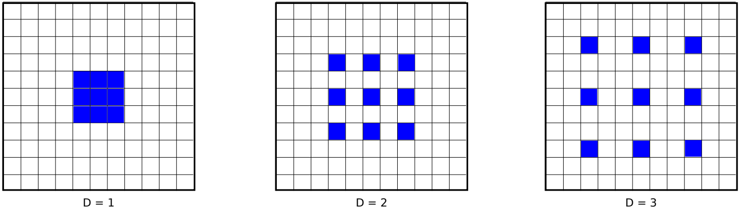
✓ ✓ ✓ ✓

Summary

1. Introduction, notation
2. Convolutional Layers
3. Down-sampling and the receptive field
4. CNN details and variants
5. CNNs in practice
6. Image datasets, well-known CNNs, and applications
   - Applications of CNNs
7. Interpreting CNNs
   - Visualising CNNs
   - Adversarial examples

Dilated Convolution

- There is a variant of convolution called *dilated convolution*[*]
- Increase spatial extent of convolution without adding parameters
  - Add a space D between each point in the convolution

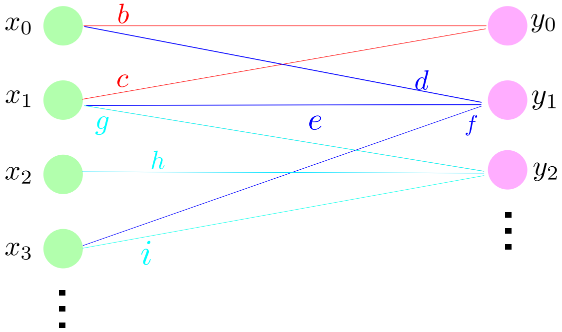


$$(u * v)(y,x) = \sum_{i,j,k} u(i,j,k) v(y - Di, x - Dj, k) \tag{2}$$

[*] ***Multi-Scale Context Aggregation by Dilated Convolution**, Yu, F, Kolten, V, ICLR 2016*

Locally connected layers / unshared convolution

- We might wish for a mix of a dense layer and a convolutional layer
- One possibility : **locally-connected layers** (sometimes called "unshared convolution")
  - Local connectivity but no weight sharing



- Number of weights increases linearly with the number of pixels, rather than quadratically (for MLPs)

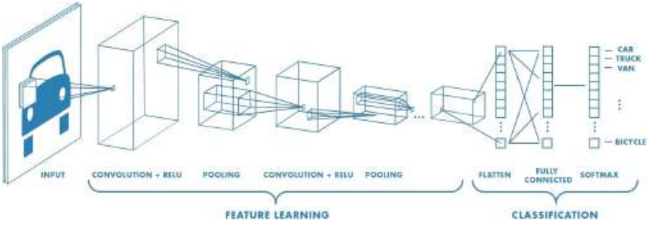
Summary

1. Introduction, notation
2. Convolutional Layers
3. Down-sampling and the receptive field
4. CNN details and variants
5. CNNs in practice
6. Image datasets, well-known CNNs, and applications
   - Applications of CNNs
7. Interpreting CNNs
   - Visualising CNNs
   - Adversarial examples

How to build your CNN ?

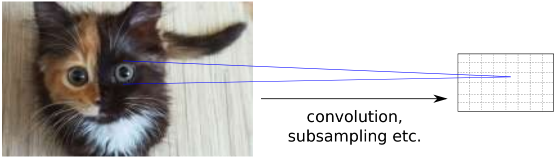
- We have looked at the following operations : convolutions, additive biases, non-linearities
- All of these elements make up convolutional neural networks
- However, how do we put these together to create our own CNN ?
  - **Architecture ?**
  - Programming tools ?
  - Datasets ?

- Simple classification CNN architecture often consists of a **feature learning section**
  - Convolution $\rightarrow$ biases $\rightarrow$ non-linearities $\rightarrow$ subsampling
  - This continues until a fixed subsampling is achieved
- After this, a **classification section** is used
  - Fully connected layer $\rightarrow$ non-linearity



Architecture

- Central question : how to choose number of layers ?
- Complicated, very little theoretical understanding, currently a hot topic of research
- However : there are a few rules of thumb to follow
  - Receptive field of the **deepest layer should encompass what we consider to be a fundamental brick** of the objects we are analysing



- Set number of layers and subsampling factors according to the problem

CNN programming frameworks

- Caffe
  - Open source, developed by University of California, Berkley
  - Network created in separate specific files
  - Somewhat laborious to use, less used than other frameworks
- Theano
  - Open source, created by the Université de Montréal
  - Unfortunately, to be discontinued due to strong competition
- Tensorflow
  - Open source, developed by Google
  - Implements a wide range of deep learning functionalities, widely used
- Pytorch
  - Open source, developed by Facebook
  - Implements a wide range of deep learning functionalities, widely used

Summary

1. Introduction, notation
2. Convolutional Layers
3. Down-sampling and the receptive field
4. CNN details and variants
5. CNNs in practice
6. Image datasets, well-known CNNs, and applications
   - Applications of CNNs
7. Interpreting CNNs
   - Visualising CNNs
   - Adversarial examples

MNIST dataset

- MNIST is a dataset of 60,000 28×28 pixel grey-level images containing hand-written digits
- The digits are centred in the images and scaled to have roughly the same size
- Although quite a “simple” dataset, still used to display performance of modern CNNs

Caltech 101

- Produced in 2003, first major object recognition dataset
- 9,146 images, 101 object categories, each category contains between 40 and 800 images
- Annotations exist for each image : bounding box for the object and a human-drawn outline

ImageNet dataset

- Dataset created in 2009 by researchers from Princeton unverisity
- Very large dataset : 14,197,122 images, hand-annotated
- Used for the ImageNet Large Scale Visual Recognition Challenge, an annual benchmark competition for object recognition algorithms

- Created by Yann LeCun in 1989, goal : to recognise handwritten digits
- Able to classify digits with 98.9% accuracy, used by U.S. government to automatically read digits

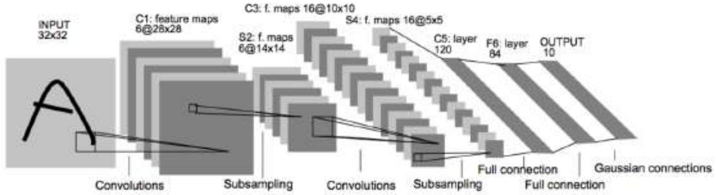


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Illustration from : ***Gradient-based Learning Applied to Document Recognition****, LeCun, Y. Bottou, L., Bengio, Y. and Haffner, Proceedings of the IEEE, 1989*

AlexNet (2012)

- AlexNet : created by Alex Krizhevsky in 2012
- Improved accuracy of ImageNet Large Scale Visual Recognition Challenge competition by **10 percentage points** (16.4%)
- First **truly deep neural network**
- Signaled beginning of dominance of deep learning in image processing and computer vision

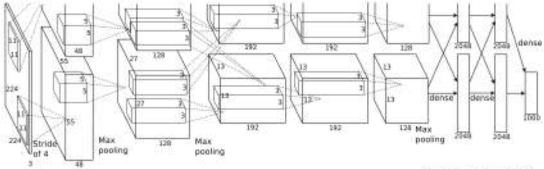
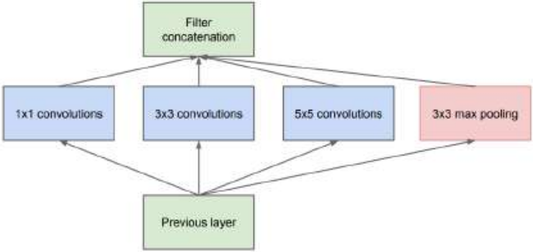


*Illustration from : **Imagenet classification with deep convolutional neural networks**, Krizhevsky, A., Sutskever, I. and Hinton, G. E, NIPS, 2012*

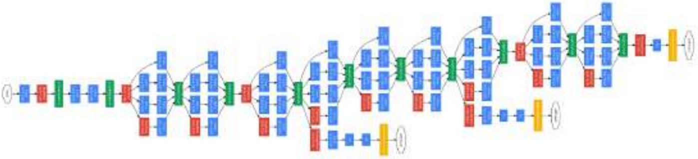
- In 2014/2015, Google introduced the "Inception" architecture/module
- Major attempt at reducing total number of parameters
- No fully connected layers, only convolutional
  - 2 million instead of 60 million for AlexNet
- Novel idea : have variable receptive field sizes in one layer



Going deeper with convolutions, *Szegedy et al, CVPR, 2015*

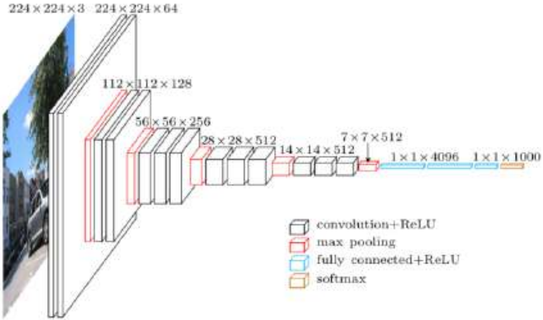
GoogLeNet (2015)

- Created by Google in 2014, GoogLeNet is a specific implementation of the "inception" architecture
- 6.6% test error rate on ImageNet (human error rate 5%)



***Going deeper with convolutions**, Szegedy et al, CVPR, 2015*

- VGG16 is a 16-layer network, with small receptive fields (3×3 filters, with less subsampling)
- Around 7.5% test error on ILSVRC



Very Deep Convolutional Networks for Large-Scale Image Recognition, *Simonyan, K. and Zisserman, A., ICLR, 2015*
Illustration from Mathieu Cord,
https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/

Summary of advances in CNNs

Network	LeNet (1998)	AlexNet (2012)	GoogLeNet (2014)	VGG16 (2015)
Image size	28×28	$256 \times 256 \times 3$	$256 \times 256 \times 3$	$224 \times 224 \times 3$
Layers	3	8	22	16
Parameters	60,000	60 million	2 million	138 million

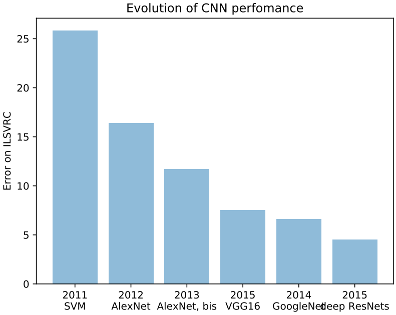


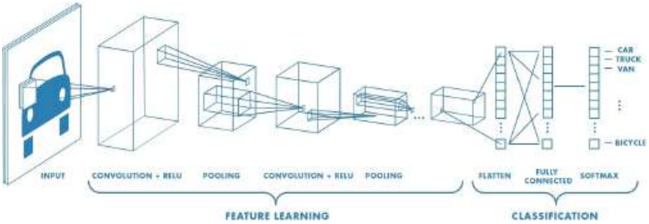
Image classification

- As we mentioned before, CNNs make sense for data with **grid-like** structures
- In particular, images are most often the target of CNNs
- Arguably the most common application of CNNs is to **image classification**
- Why is image classification important ? Closely linked to :
  - Object detection
  - Tracking
  - Image search (in large databases for example)
- In recent years, the best performing classification algorithms have been using neural networks

Image classification

- Why is image classification difficult ?
  - Images can vary in size, shape, position
  - We need to deal with variable lighting conditions, occlusions etc.
- Let us look at a standard CNN classificaton network

- We have input datapoints x, which we wish to classify into several, predefined classes $\{c_i\}, i = 1 \ldots K$, where K is the number of classes
- As we have seen, convolution, non-linearities, subsampling allow for robust classification that is invariant to many perturbations



- Vast majority of CNN classification networks follow this general architecture

Resiudal architectures : ResNET

- ResNET* (2016) uses **skip connections** to mitigate the **vanishing gradient** problem
  - Similar to LSTM, except propagates through network layers, rather than time
- Residual mechanism used in many subsequent architectures
- Latest residual archtiture gives 87.54% accuracy on ImageNet

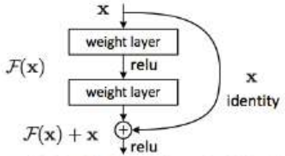


Figure 2. Residual learning: a building block.

Deep Residual Learning for Image Recognition, Kaiming, H. et al, CVPR, 2016
Illustration from https://becominghuman.ai/resnet-convolution-neural-network-e10921245d3d

Attention mechanism in image networks

- Recall the **attention mechanism** in RNNs : addresses problem of long range dependency
  - Networks exist with attention only : **transformer***
- Also used in image network architectures (usually **self-attention**)

$$Attention(Q, K, V) = \mathrm{Softmax}(QK^T)V \tag{3}$$

- Q, queries: what is the importance of these elements
- K, keys: we use these elements for comparison (weighting)
- V, values: we use these to "reconstruct" the queries
- Often Q, K, V are the same, image patches

* ***Attention is all you need***, *Vaswani et al, NIPS, 2017*

Attention mechanism in image networks

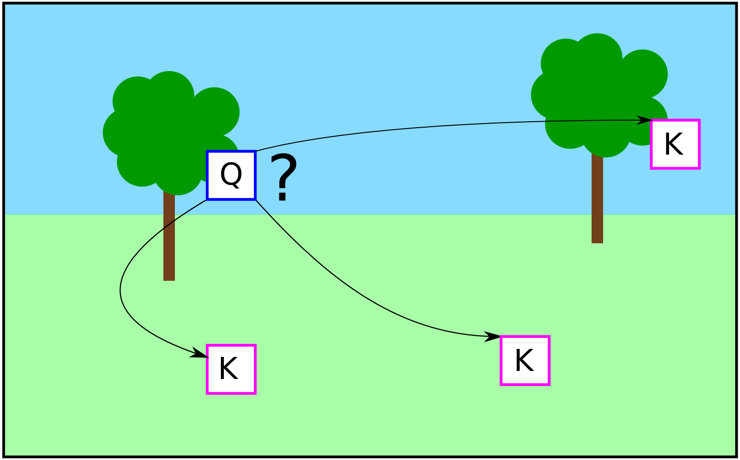
- Recall the **attention mechanism** in RNNs : addresses problem of long range dependency
  - Networks exist with attention only : **transformer**[*]
- Also used in image network architectures (usually **self-attention**)

$$Attention(Q, K, V) = \mathrm{Softmax}(QK^T)V \tag{4}$$

- This equation says that the attention is a weighted version of V
- The weights are given by a softmax of the dot products between patches in Q and those in K

[*] ***Attention is all you need***, *Vaswani et al, NIPS, 2017*

Attention mechanism in image networks



* ***Attention is all you need***, *Vaswani et al, NIPS, 2017*

Attention mechanism in image networks

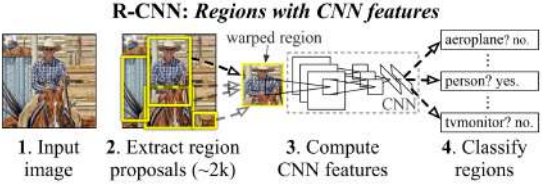
- Combined attention/convolution archtictures present the best accuracies on ImageNeT (to date*)
  - CoAt-Net7: 90.88% accuracy on ImageNet

Rank	Model	Top 1 Accuracy	Top 5 Accuracy	of params	Training Data	Paper	Code	Result	Year	Tags
1	CoAtNet-7	90.88%		2440M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	ConvTransformer JFT-3B
2	ViT-G/14	90.45%		1843M	✓	Scaling Vision Transformers			2021	Transformer JFT-3B
3	CoAtNet-6	90.45%		1470M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	ConvTransformer JFT-3B
4	ViT-MoE-15B (Every-2)	90.35%		14700M	✓	Scaling Vision with Sparse Mixture of Experts			2021	Transformer JFT-3B
5	Meta Pseudo Labels (EfficientNet-L2)	90.2%	98.8	480M	✓	Meta Pseudo Labels			2020	EfficientNet JFT-300M

* https://paperswithcode.com/sota/image-classification-on-imagenet

Image classification

- We can also **detect the position** of objects in images
- RNN* proposes a simple approach :
  1. Propose a list of bounding boxes in the image
  2. Pass the resized sub-images through a powerful classification network
  3. Classify each sub-image with your favourite classifier

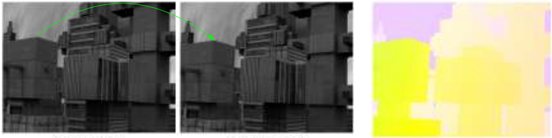


- Many variants on this work (Fast R-NN, Faster R-CNN) etc.

* ***Rich feature hierarchies for accurate object detection and semantic segmentation***, *Girschik, R. et al. CVPR 2014*

- **Motion estimation** is a central task for many image processing and computer vision problems : tracking, video editing
- **Optical flow** involves estimating a vector field $(u, v) : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ where each vector points to the displacement of pixel (x, y) from an image I_1 to I_2

$$I_1(x, y) = I_2(x + u(x, y), y + v(x, y))$$

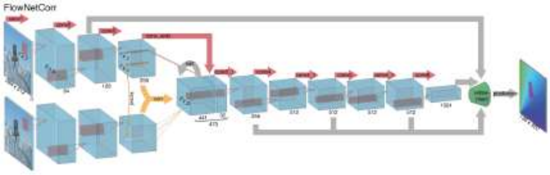


(a) Frame 1 (b) Frame 2 Optical flow

Illustration from : ***BriefMatch: Dense binary feature matching for real-time optical flow estimation****, Eilertsen, G, Forssén, P-E, Unger, J., Scandinavian Conference on Image Analysis, 2017*

Motion estimation with CNNs

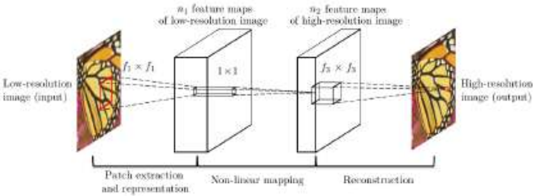
- A major challenge of optical flow estimation is to handle both fine and large-scale motions
  - This is difficult to do with classical, variational approaches
- CNNs have this multi-scale architecture already built in
- Example : FlowNet* uses this, first extracting meaningful features from the images (in parallel) and then combining them to create the optical flow

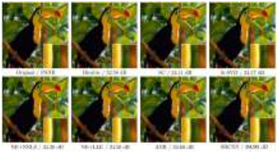


* ***FlowNet: Learning Optical Flow with Convolutional Networks***, *Fischer et al, ICCV 2015*

Super-resolution

- Image super-resolution : go from a low-resolution image to a higher-resolution one
- Relatively straightforward approach with a CNN*





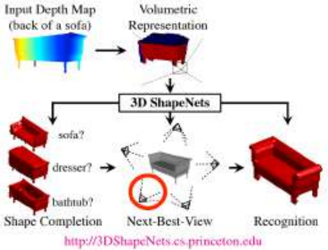
- Drawback, highly dependent on degradation used in lower-resolution images in database

* ***Learning a deep convolutional network for image super-resolution***, *Chao et al, ECCV 2014*

Point clouds

- CNNs require regular grids. Point cloud data are not in this format
- Nevertheless, ways have been found to deal with this

- ShapeNet* splits a volume up into sub-regions that are processed by CNNs
- Each region is a Bernoulli random variable representing the probability of this voxel belonging to a shape
- This general approach (using voxels) is followed in many other approaches



* ***3d shapenets: A deep representation for volumetric shapes***, *W. Zhirong et al. CVPR, 2015*

Summary

1. Introduction, notation
2. Convolutional Layers
3. Down-sampling and the receptive field
4. CNN details and variants
5. CNNs in practice
6. Image datasets, well-known CNNs, and applications
   - Applications of CNNs
7. Interpreting CNNs
   - Visualising CNNs
   - Adversarial examples

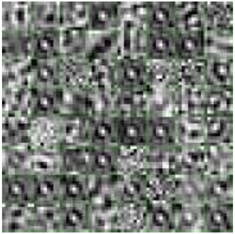
Adversarial examples

- As is often the case in deep learning, it is very difficult to understand what is going on in CNNs
- Much research is being dedicated to understanding these networks
  - Explainable AI (XAI) Darpa project*
- We discuss two topics related to interpretability
  - Visualising CNNs
  - Adversarial examples

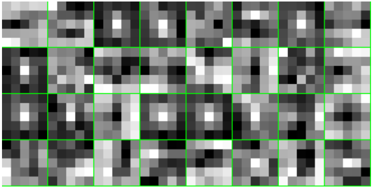
* *https://www.darpa.mil/program/explainable-artificial-intelligence*

Visualising CNNs

- We would like to **understand what CNNs are learning**
  - Unfortunately filters are difficult to interpret (especially deeper layers)



Layer 1 filters



Layer 3 filters

- Therefore, much research has been dedicated to **visualising** CNNs

Visualising CNNs

- Idea : "invert" CNN, find x to maximise the output of a certain layer
  - Understand what this layer is "seeing"
- This is **possible due to backpropagation**

Basic CNN visualisation algorithm

- Choose a layer ℓ to visualise
  - $x_0 \sim \mathcal{N}(0,1)$
  - For $i = 1 \dots N$
    - $x^i = x^{i-1} + \lambda \nabla_x \| u^\ell (x^{i-1}) \|$ Gradient ascent
- Return x^N

$$\xrightarrow[\text{Gradient ascent}]{x^i = x^{i-1} + \lambda \nabla_x \| u^\ell (x^{i-1}) \|}$$

Visualising features

- Generalisation: **maximise response to a given filter response**
- Choose layer ℓ, filter k and element ("pixel") (i, j)
- Random initialisation x_0, constrain norm of solution x

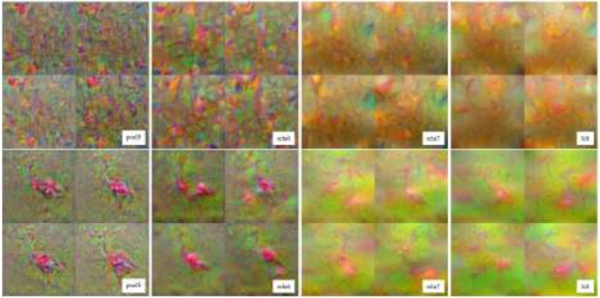
$$\hat{x} = \arg\max_{x} u^{\ell}_{i,j,k}$$
$$\text{with } \| x \| = \rho$$

- Optimisation : **gradient ascent**

† *Erhan, Bengio, Courville, Vincent,* ***Visualizing Higher-Layer Features of a Deep Network****, University of Montreal, 2009*

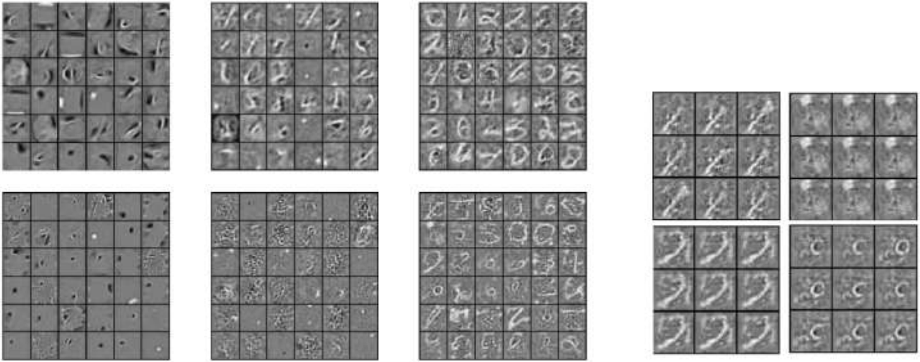
- More sophisticated approach: standard inverse problem with regularisation

$$\hat{x} = \arg\min_x \|f(x) - f_0\|_2^2 + \lambda \|x\|_2^2 + \mu \|\nabla x\|_2^2 \qquad (5)$$



† *Mahendran and Vedaldi* ***Understanding Deep Image Representations by Inverting Them****, Conference on Computer Vision and Pattern Recognition, 2014*

Visualising CNNs



Layer 1 Layer 2 Layer 3 Layer 4

Maximisation of different activations applied to MNIST dataset

† *Erhan, Bengio, Courville, Vincent,* ***Visualizing Higher-Layer Features of a Deep Network****, University of Montreal, 2009*

Visualising CNNs

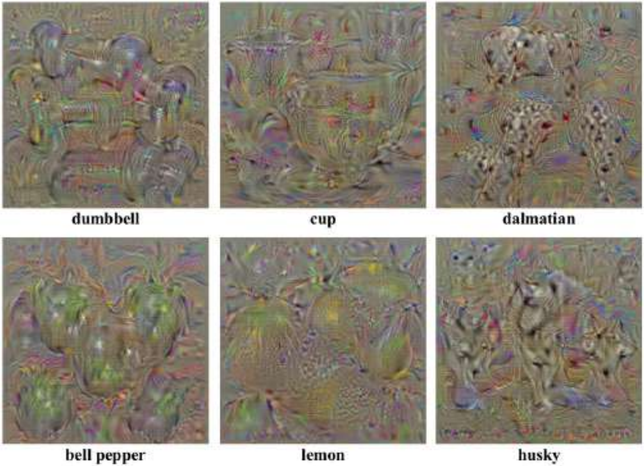
- Another approach of Simonyan et al.[†] proposes to see what images correspond to what classes
- Choose a class c, maximise the response of this class

$$\hat{x} = \arg\max_{x} f(x)_c - \lambda \|x\|_2^2$$

- Find an L_2-regularised image which maximises the score for a given class c
- Initialise with random input image x_0

[†] *Simonyan, Vedaldi, Zisserman* ***Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps****, arXiv preprint arXiv:1312.6034, 2013*

Visualising CNNs



Class *model* visualisation

† *Simonyan, Vedaldi, Zisserman* ***Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps****, arXiv preprint arXiv:1312.6034, 2013*

- Similar idea with Inception architecture of Google : "Deep Dream"
- Maximise a class from input image

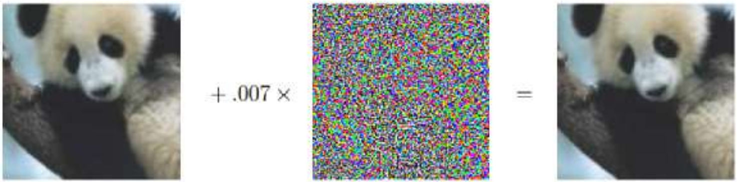
Input image

Maximising "dogs" category

Deepdream - a code example for visualizing neural networks, *Mordvintsev, A., Olah, C. and Tyka, M., Google Research, 2015*

Adversarial examples

- We often get the impression that CNNs are the end all and be all of AI
- Consistently produce state-of-the-art results on images
- However, CNNs **are not infallible** : adversarial examples[†] !



- How was this image created ???

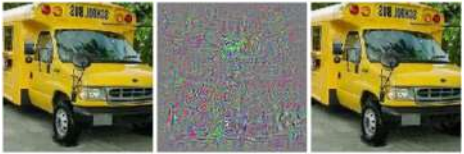
[†] ***Intriguing properties of neural networks***, *Szegedy, C. et al, arXiv preprint arXiv:1312.6199, 2013*

Adversarial examples

- Szegedy et al. propose[†] add a small perturbation r that fools the classifier network f into choosing the wrong class c for $\hat{x} = x + r$

$$\arg\min_r |r|_2^2, \;\; \text{s.t}\; f(x+r) = c, \; x + r \in [0,1]^n$$

- $\hat{x}$ is the closest example to x s.t $\hat{x}$ is classified as in class c
- Minimisation with box-constrained L-BFGS algorithm

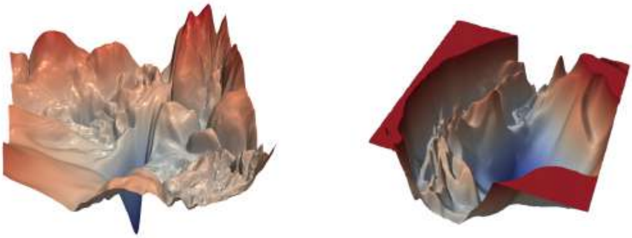


x $\quad$ r $\quad$ $\hat{x}$

[†] ***Intriguing properties of neural networks***, *Szegedy, C. et al, arXiv preprint arXiv:1312.6199, 2013*

Adversarial examples

- Common explanation : the space of images is very high-dimensional, and contains many areas that are unexplored during training time



Example of loss surfaces in commonly used networks (Res-Nets)

Illustration from ***Visualizing the Loss Landscape of Neural Nets****, Li, H et al, NIPS, 2018*

Adversarial examples

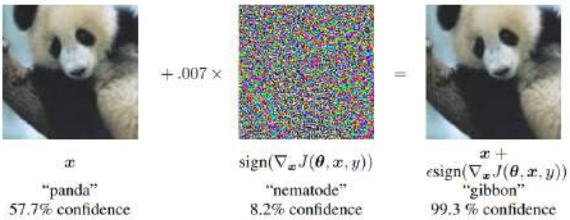
- Many approaches to adversarial examples exist. Goodfellow et al.[†] propose a principled way of creating these
- Consider the output of a fully connected layer $\langle w, \hat{x} \rangle = \langle w, x \rangle + \langle w, r \rangle$
- Let us set $r = \mathrm{sign}(w)$. What happens to $\langle w, \hat{x} \rangle$?
  - Increase by nm as dimension n increases (m is average value of w)
  - However, $|r|_\infty$ does not increase with n
- Conclusion : we can add a small vector r that increases the output response $\langle w, \hat{x} \rangle$

† ***Explaining and Harnessing Adversarial Examples***, *Goodfellow, I.J, Shlens, J. and Szegedy, C., ICLR 2015*

Adversarial examples

- Goodfellow et al. consider a local linearisation of the network's loss around θ
  - $\mathcal{L}(x_0) \approx f(x_0) + w\nabla_x \mathcal{L}(\theta, x_0, y_0)$
- Thus, the perturbation image $\hat{x}$ is set to

$$\hat{x} = x + \epsilon \text{sign}(\nabla_x \mathcal{L}(\theta, x, y))$$



† ***Explaining and Harnessing Adversarial Examples***, *Goodfellow, I.J, Shlens, J. and Szegedy, C., ICLR 2015*

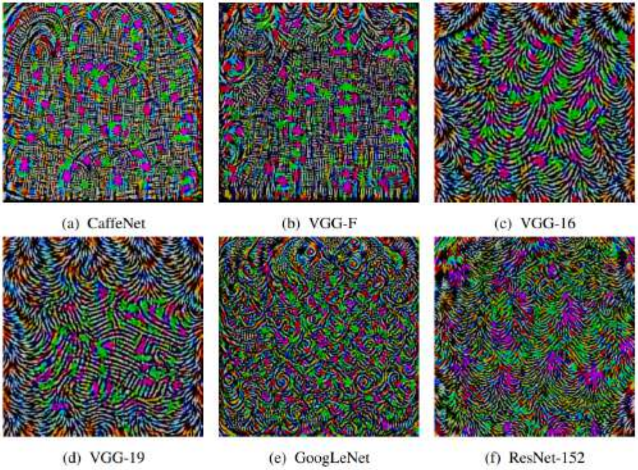
Adversarial examples

- Even worse, it is possible to create **universal adversarial** examples†
- Perturbations that fool a network for any image class

- Simple algorithm : initialise perturbation r, go through database adding specific perturbations to r, project onto set $\{\, r, ||r|| < \varepsilon\}$
- What do these perturbations look like ?

† ***Universal adversarial perturbations***, *Moosavi-Dezfooli, S-M, et al arXiv preprint (2017)*

Adversarial examples



† ***Universal adversarial perturbations***, *Moosavi-Dezfooli, S-M, et al arXiv preprint (2017)*

Adversarial examples

- Conclusion : CNNs are not necessarily robust
- Adversarial examples are a significant problem :
  - Even **printed photos** of adversarial examples work[†]

(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

- Explaining and resisting adversarial examples is currently a hot research topic

[†] ***Adversarial Examples in the Physical World***, *Kurakin, A., Goodfellow, I. J, Bengio, S. et al. ICLR workshop, 2017*

Summary

- CNNs represent the state-of-the art in many different domains/problems
- If you have an unsolved problem, there is a good chance CNNs will produce a good/excellent result
- **However : theoretical understanding is still relatively limited**
    - This leads to problems such as adversarial examples
    - It is not clear whether CNNs are truly robust/generalisable
    - This is a hot research topic, important if CNNs are to be used in industrial applications
- 21/10/2021 : last lab work, on CNNs