Convolutional Neural Networks

Master BDMA course

Tom Dupuis

Master BDMA, CentraleSupélec

October 18, 2023

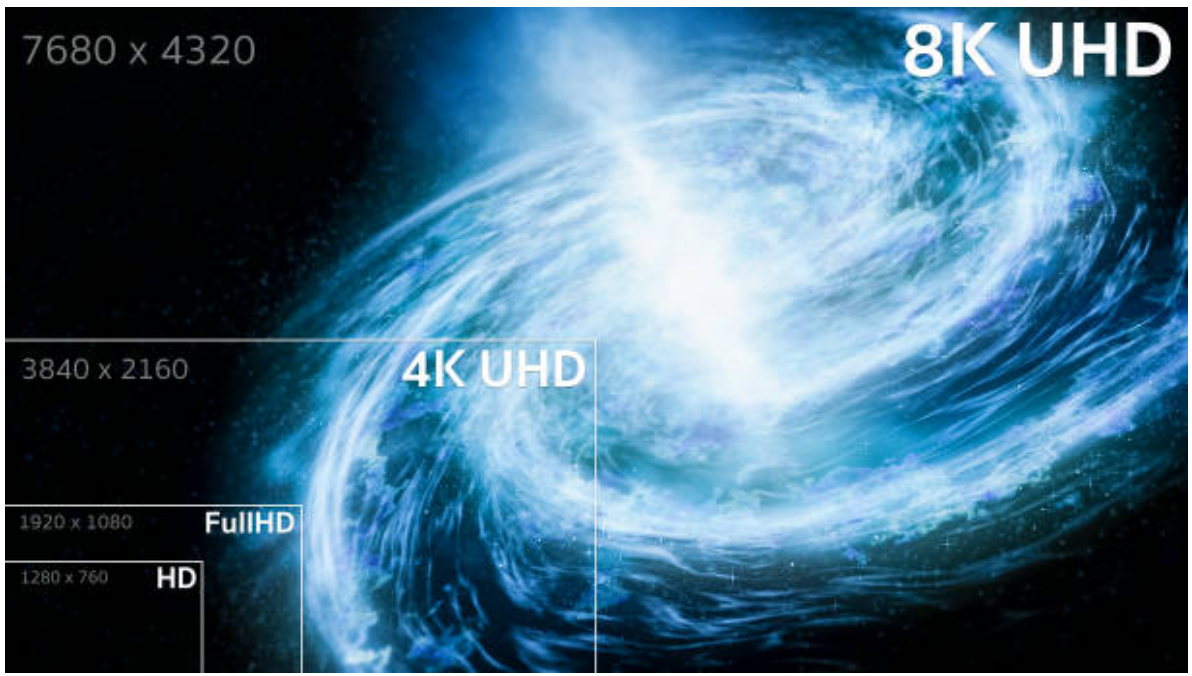
Plan

Resources

- This course is based on several full course resources on Computer Vision that have been summarized for one course of 1h30
- The main used resources are:
  - Dive into Deep Learning: https://d2l.ai/index.html
  - Deep Learning Book: https://www.deeplearningbook.org
  - The diverse references we provided
- If you want to go in-depth of this large topic, we strongly recommend you to also look at these resources

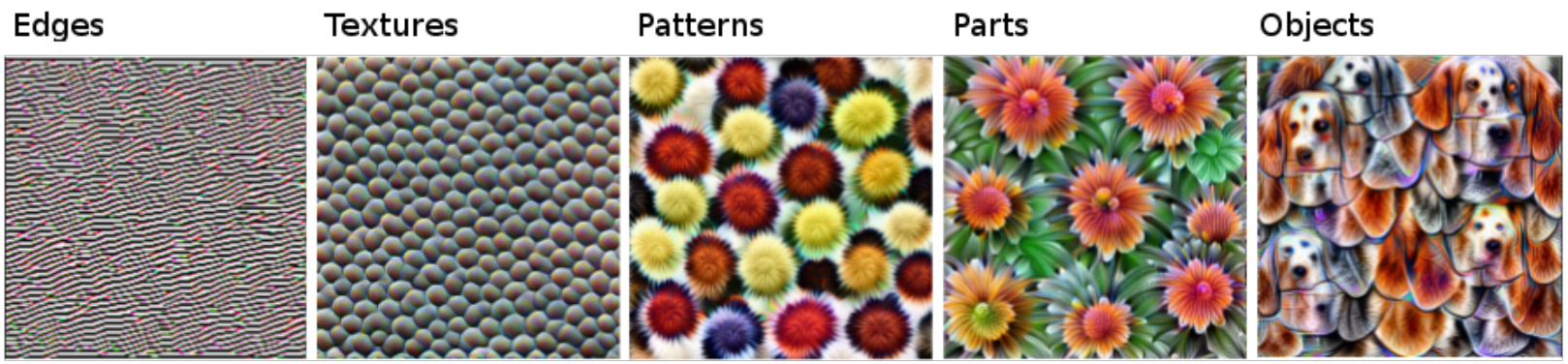
MLP limitations

- Dealing with data with a high number of features
  - Let be a dataset of images at full HD resolution
  - The feature size is 1920 * 1080 * 3 = 6 220 800
  - Learning a single layer to reduce the dimension to 1000 requires $\sim 600 * 1K \approx 10^9$ parameters
  - This is very hard to train



MLP limitations

- Use the knowledge of the input data modality to shape the network
  - For images, our network should be equivariant to translation e.g detect a repeated pattern independently of where it is located

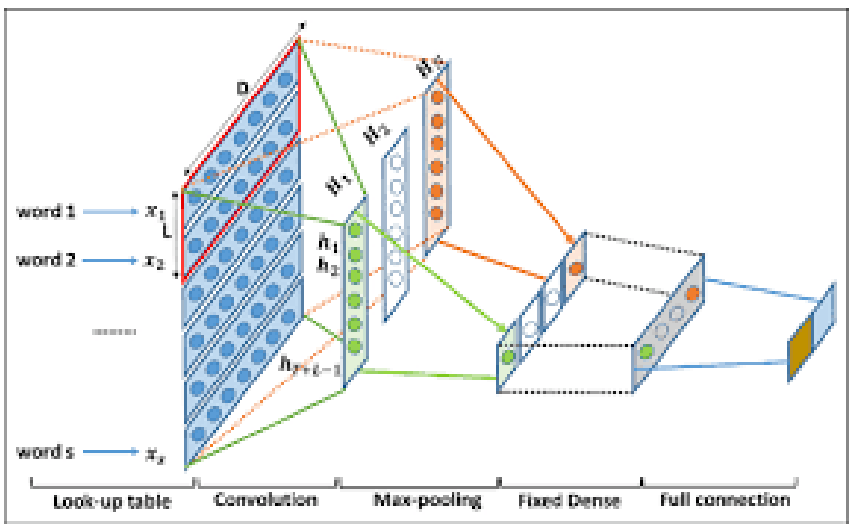


What are CNNs ?

Definition

Convolutional neural networks are neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

- Convolutional Neural Networks deal with data arranged according to a grid
  - Temporal data (audio, financial data, ...), images, videos, ...



History

- First work on CNNs in 1998, LeCun et al. 1998
- Theoretical advances to make Deep Neural Networks converge, Hinton et al. 2006
- Access to large datasets such as ImageNet, Deng et al. 2009
- Advances in harward technology to scale learning (CPU, GPU)
- Win of the 2012 ImageNet challenge by AlexNet, Krizhevsky et al. 2012

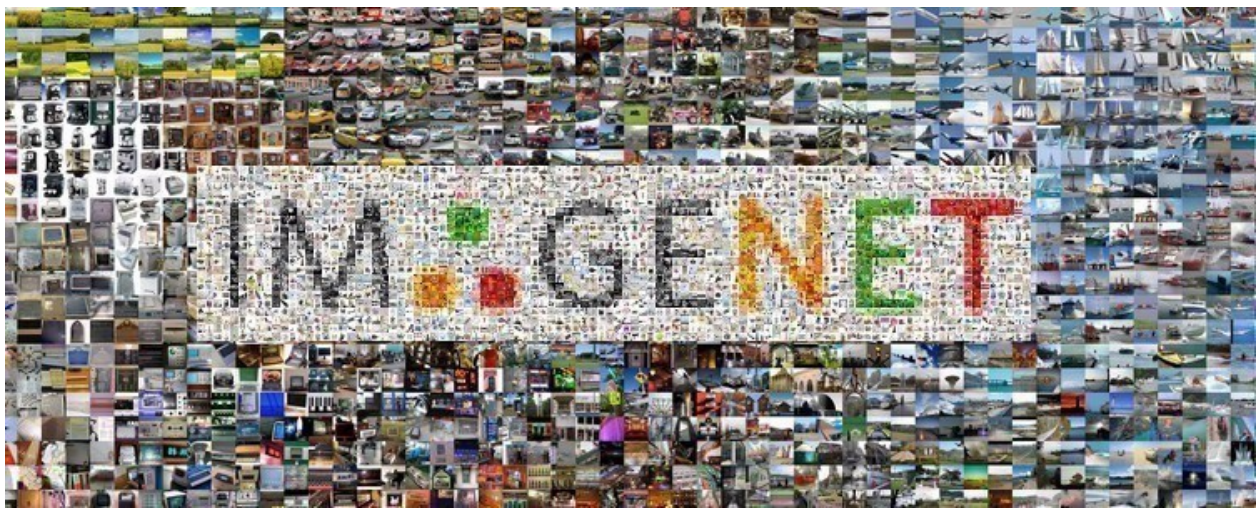


Figure: ImageNet challenge

Plan

Definition

Convolution

For $f, g : \mathbb{R} \to \mathbb{R}$

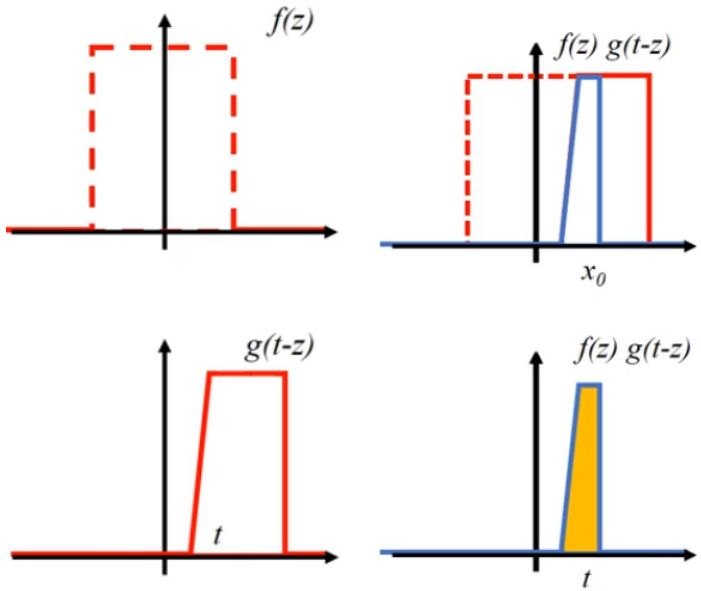
$$(f * g)(x) = \int_{-\infty}^{\infty} f(z)g(x - z)\, dz$$

For discrete functions (here in 2D):

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

- f is the **input** signal
- w is the **kernel**, also called filter
- $f * w$, the convolution output, is the **feature map**

Illustration and properties

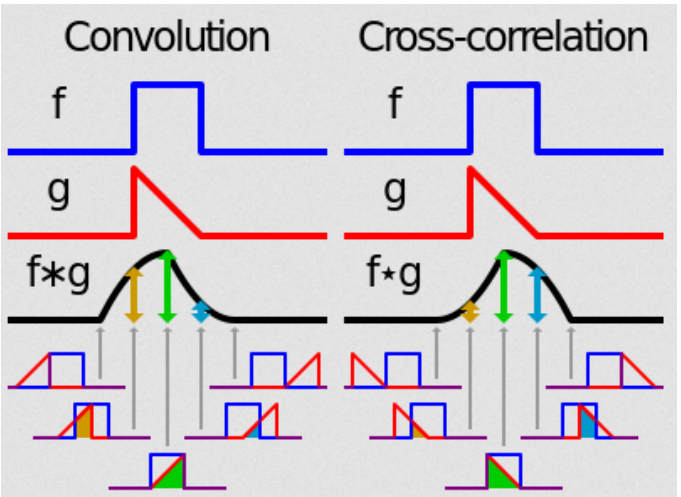


- Commutativity: $(f * g)(x) = (g * f)(x)$
- Distributivity: $(f * (g + h))(x) = (f * g)(x) + (f * h)(x)$
- Associativity: $((f * g) * h)(x) = (f * (g * h))(x)$

Cross-Correlation

- In practice CNNs do not use convolutions but **cross-correlations**
- Cross-correlation lose the commutativity property that is not required to train a Neural Network

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



- By convention, we use the term "convolution" to describe cross-correlations

Illustration of a Convolution

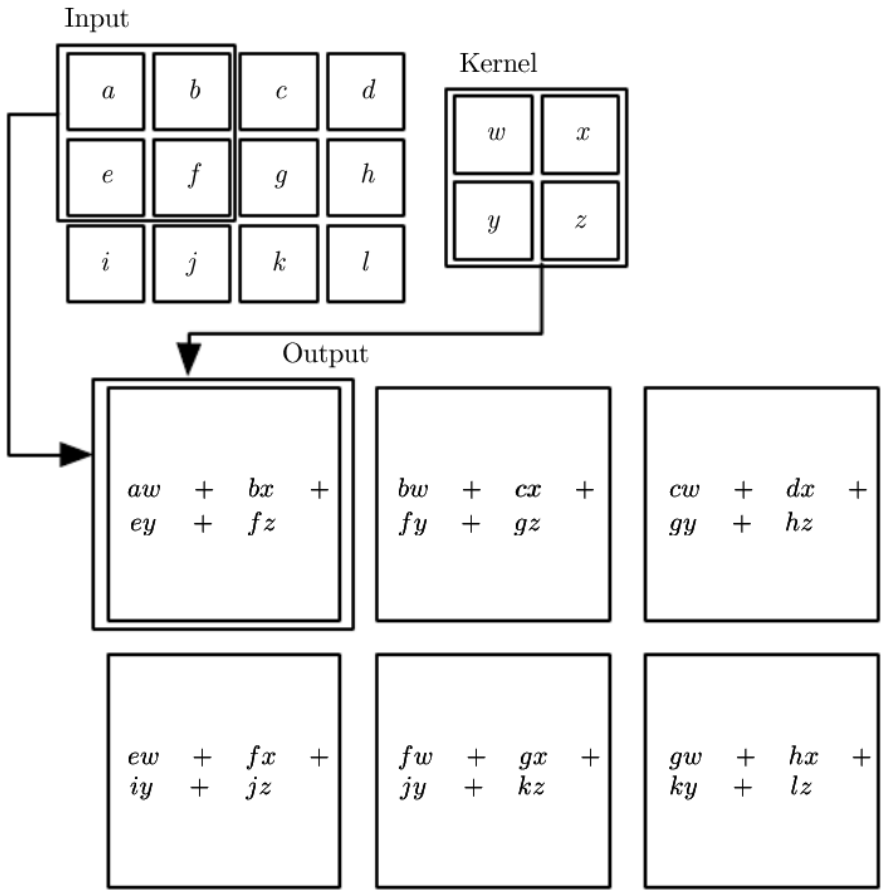


Figure: Convolution of a 3×4 input with a 2×2 kernel[1]

[1] Goodfellow et al. 2016.

Motivation to use Convolutions 1/3: Sparse interactions

- MLPs makes an interaction between each input neurons and each output neurons
- CNNs have **sparse interactions** thanks to kernels
- Kernels have a small number of parameters
- It significantly reduces the number of parameters

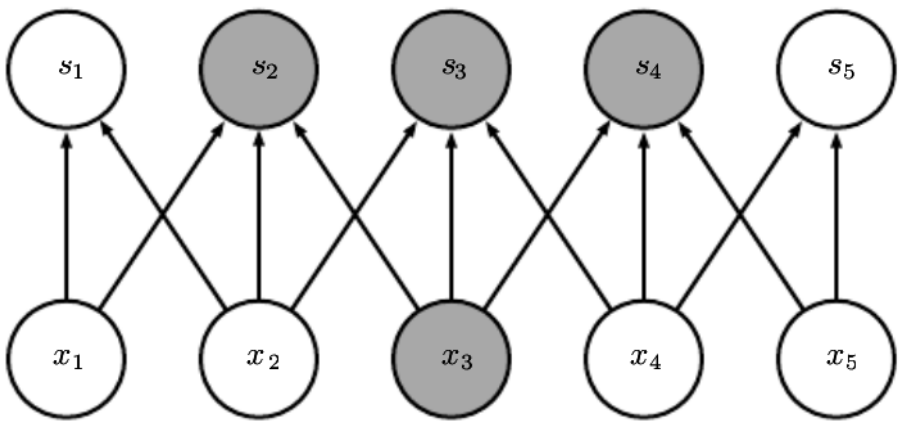


Figure: Sparse connectivity: for a kernel of width 3, an input unit affects **only** 3 output units

Motivation to use Convolutions 2/3: Parameter sharing

- In MLP, there is one weight per connection between the input and the output
- CNNs apply the **same kernels** to different patch of an input
- It significantly reduces the number of parameters

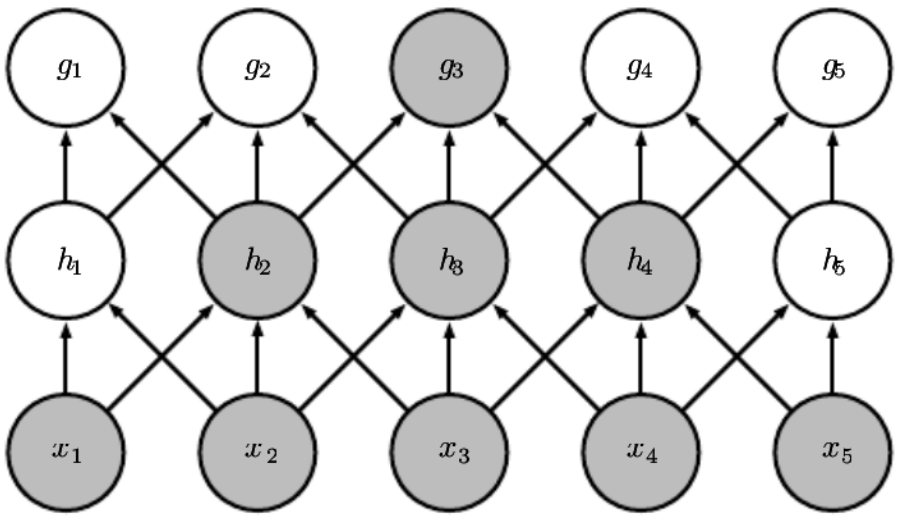


Figure: The same kernel is applied to each input patch through the black arrows

Motivation to use Convolutions 3/3: Equivariant translation

Equivariance

A function f is said to be equivariant to a transform g if for any input x:

$$f(g(x)) = g(f(x))$$

- Because of parameter sharing, the CNNs are **equivariant to translation**
- Particularly useful for some modalities such as image for which we aim to detect several objects in the same manner

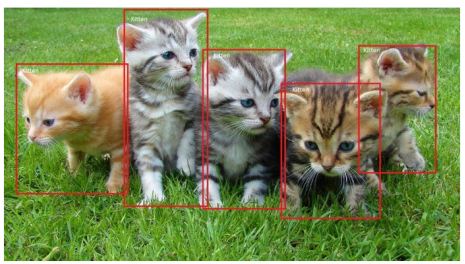


Figure: Equivariance translation, if the input is shifted, the convolution computes the same representation.

Channels

- Images, and some other modalities such as videos, are composed of **channels**: Red, Green and Blue
- Kernels are convoluted with all channels of the input

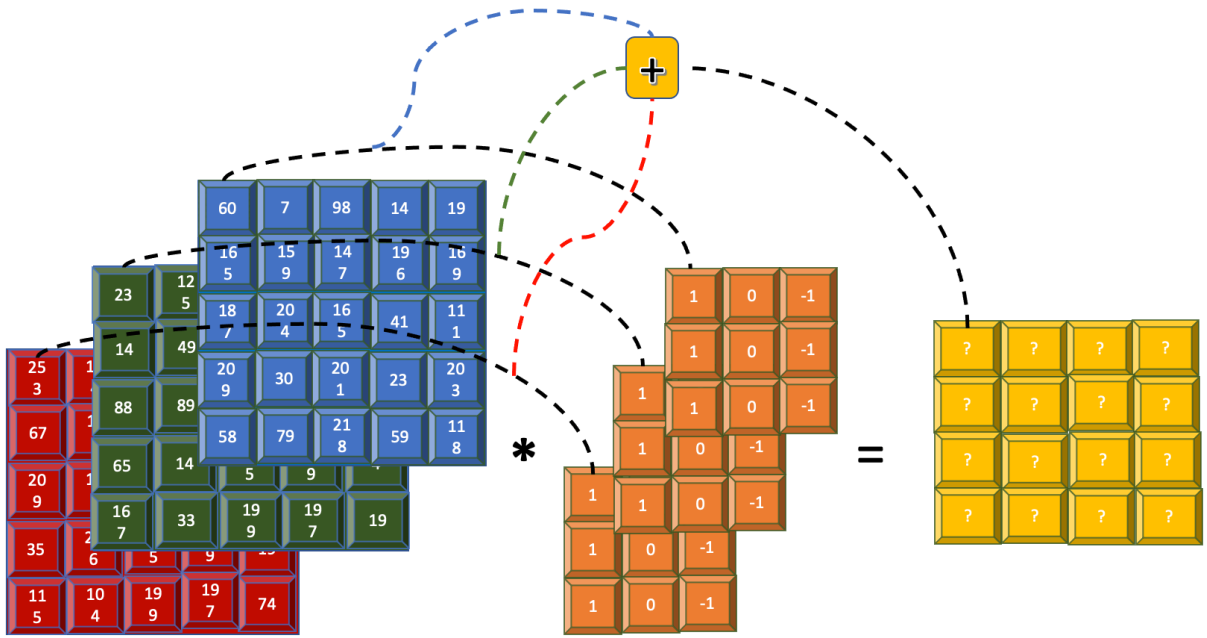


Figure: Convolution of a 3-channel input with a 3-channel kernel

Equivariance to zoom, rotation, ...

- Convolutions are **not equivariant** and **not invariant** to rotation, zoom, etc
- You might want to be invariant to such transformations for classification for example
- To make the network invariant to such transformations:
  - Transform the data with some data augmentations, such as zooming or rotating
  - Feed the neural network with transformed data

Figure: Rotation of an image.

Convolution reduces input size

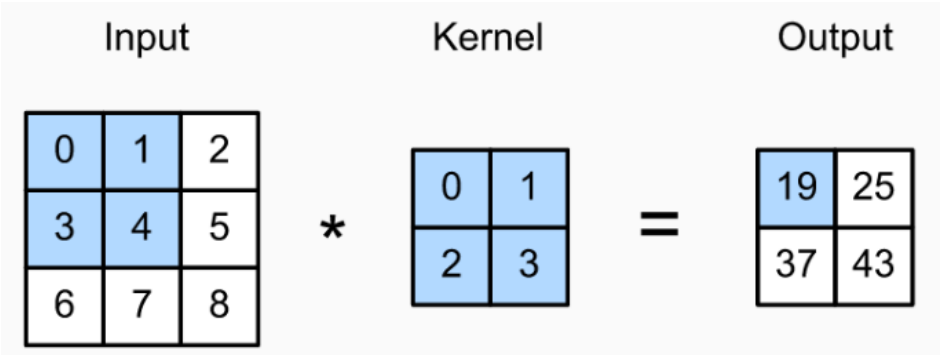


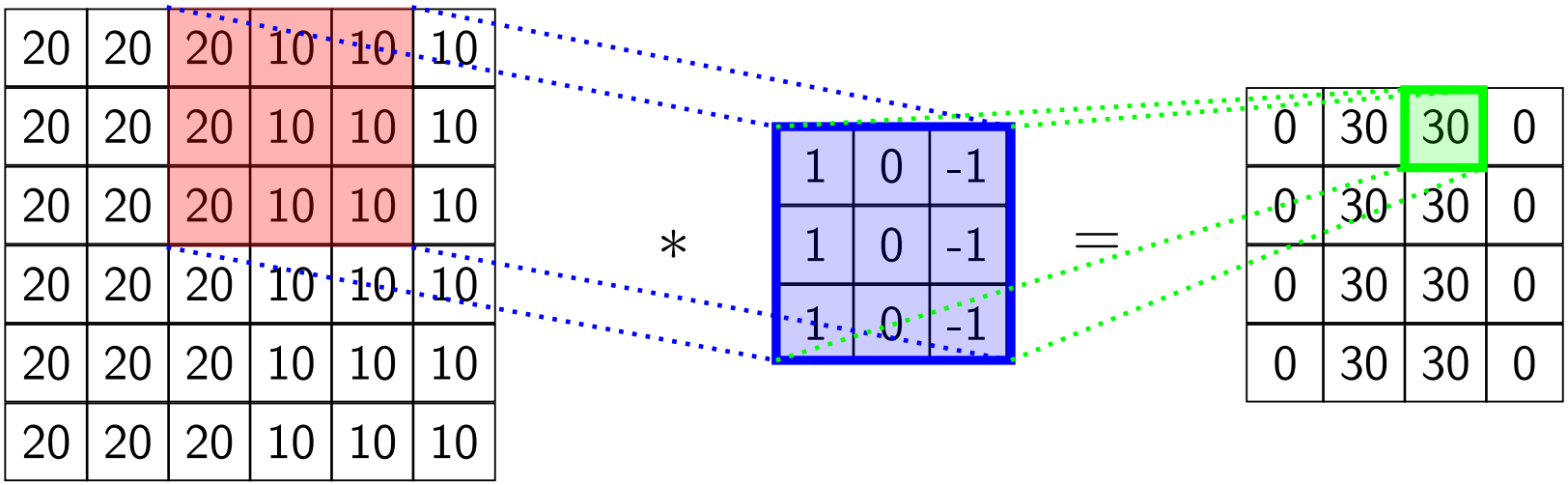
Figure: Convolution of a 3×3 with a 2×2 kernel. The output is 2×2

- The output size is lower than the input size if the kernel is larger than 1×1
- This is due to the necessity to **fit the whole image**
- The output size of the convolution of a $n_h \times n_w$ with a kernel of size $k_h \times k_w$ is

$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

- Later we will see how to keep the same size by using padding

Weights in kernels: fixed



- In this example, we make our kernel look for vertical edges and fixed the weights ourselves

Weights in kernels: learned

- Learning a CNN backbone through Deep Learning is **learning the weights of the kernels via backpropagation**
- The filters seek to learn the best representation to fit its training objective given via the *training loss*
- Using different kernels allow to detect **various patterns**
- CNNs contain thousands of Kernels and are grouped in **hierarchical layers** to learn from low-level features to high-level features

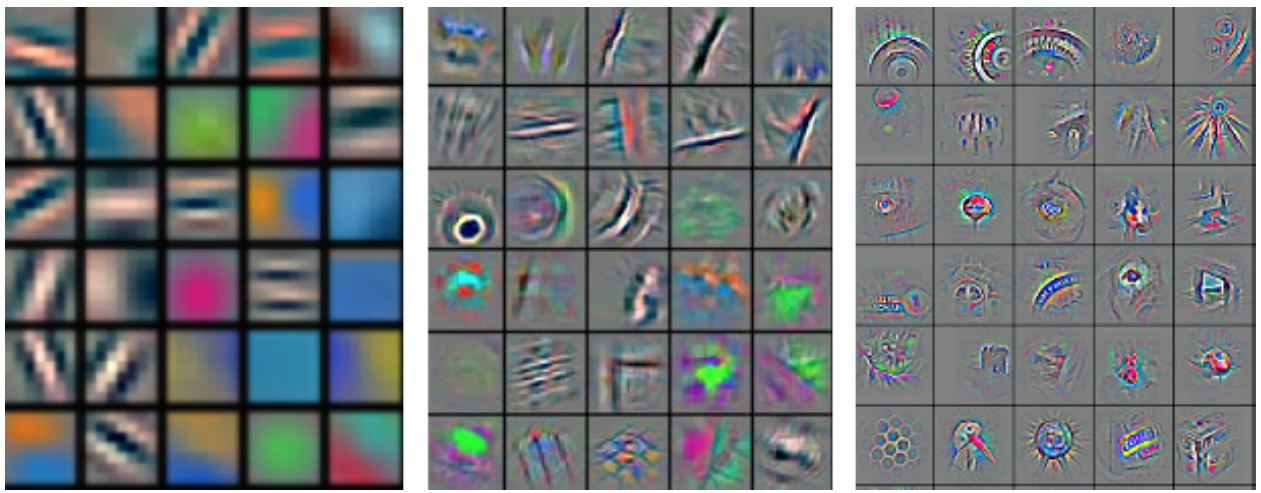


Figure: Visualization of learned kernels through several CNN layers

Learn several filters

- To compute various features such as different edge detections (horizontal, vertical, diagonal, ...), we need **several** filters
- For an input of size size $C \times n \times n$ and F filters of size $C \times m \times m$, with C channels, the feature map produced has F channels and is of size

$$F \times (n - m + 1) \times (n - m + 1)$$

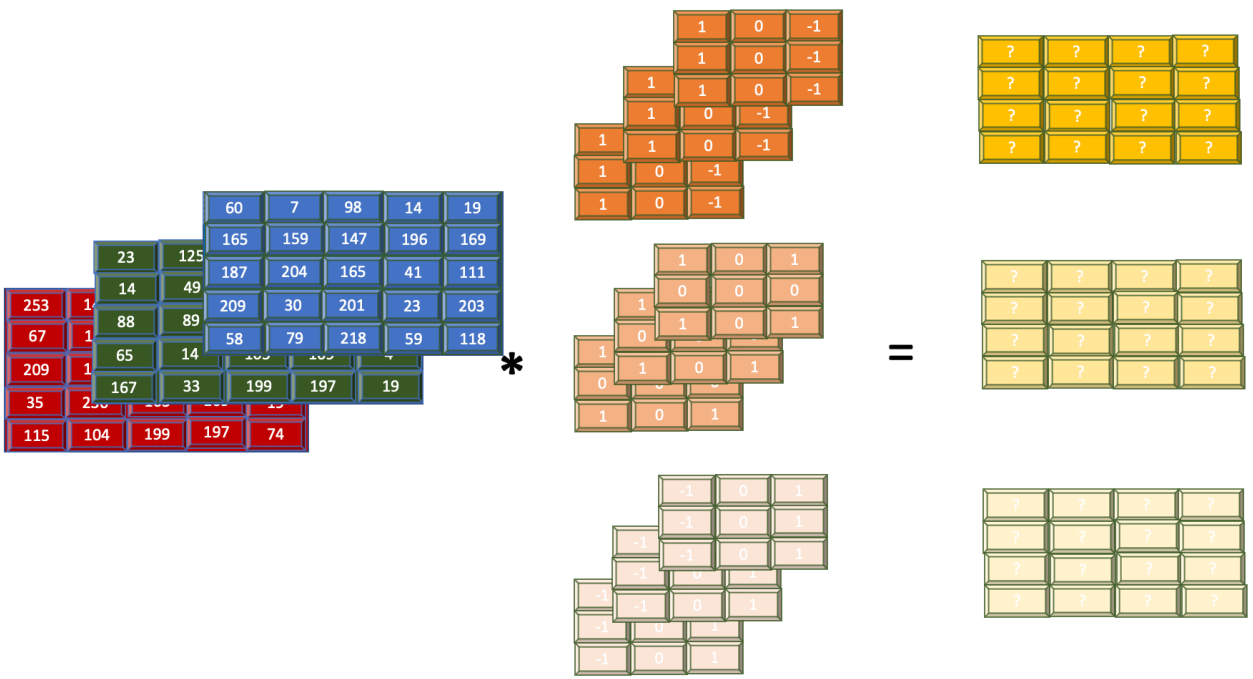


Figure: Convolution of a 3-channel input with 3 3-channel kernels

Receptive field

Receptive field

The **receptive field** is defined as the region in the input space that a particular CNN's feature is looking at (i.e. be affected by):
$r = \sum_{l=1}^{L}(k_l - 1)\prod_{i=1}^{l-1} s_i) + 1$

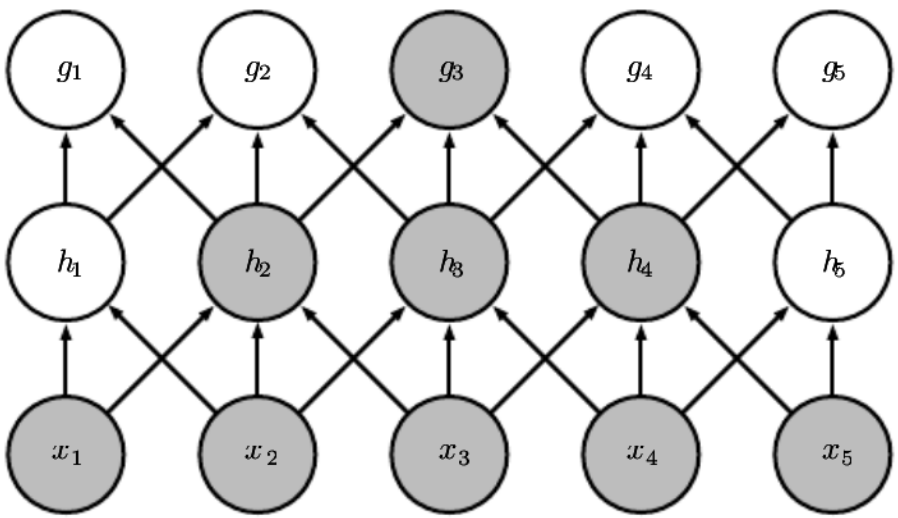
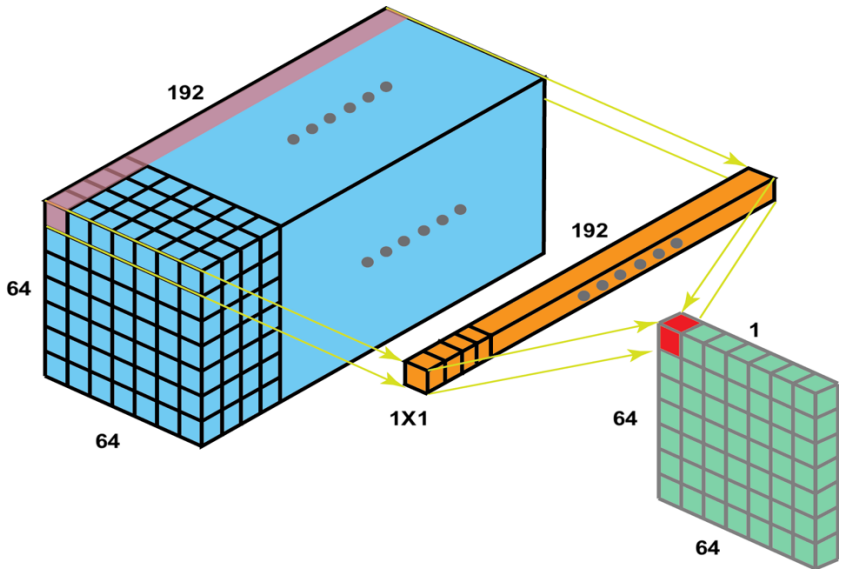


Figure: Deeper convolutions increase the receptive field of the output unit

Convolution 1x1



- Linear combination of all channels for each pixel
- Allow to **learn** to reduce or increase the number of filters
- If input of size $F_{in} \times H \times W$ and filter of size $F_{out} \times H \times W$
  - If $F_{out} < F_{in}$: dimension reduction
  - If $F_{out} > F_{in}$: increase dimension
- Can replace Fully Connected Layers (example classification if $F_{out} = N_{classes}$)

Plan

Sides problem

- Convolution is applied on **fixed size patch** of data
- Convolution can only be shifted until there are no pixels
- Two potential problems:
  - Output size is reduced
  - Lost information on the sides

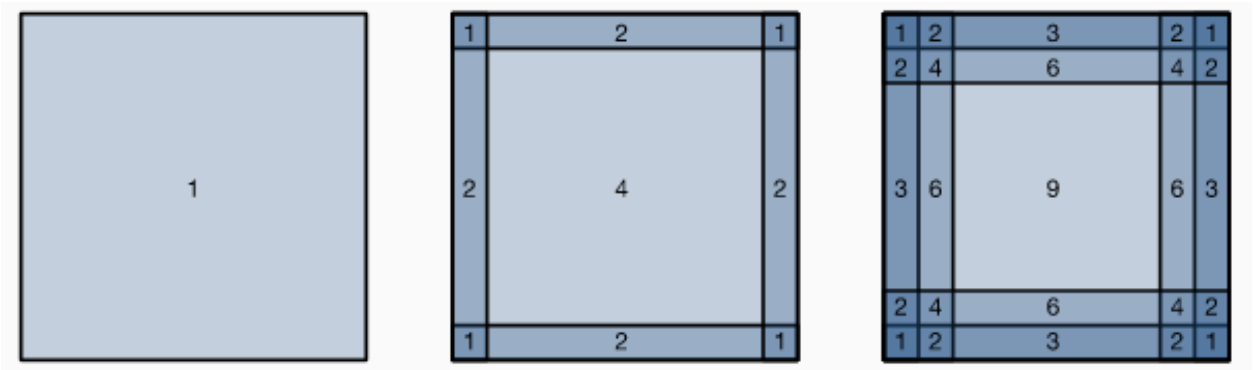


Figure: Pixel utilisation from an image by convolution of kernels of size 1×1, 2×2 and 3×3

Padding

- To avoid edge problem: add data on each side of the image
- There are various possible padding strategies
- In practice, generally *zero* is employed

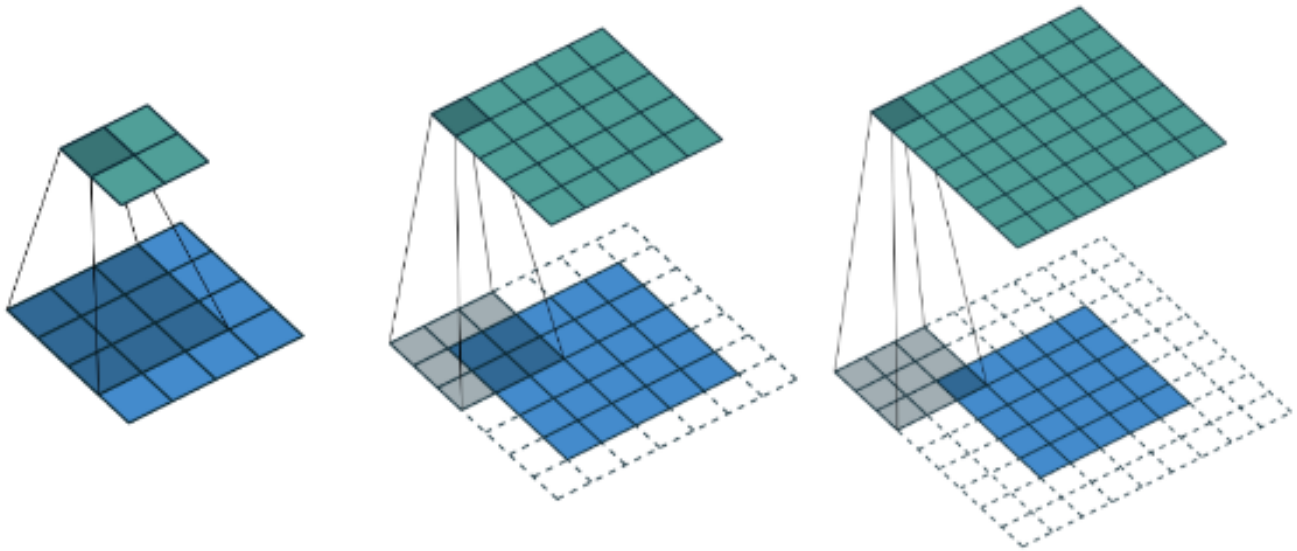
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	0	0
0	0	6	7	8	9	10	0	0
0	0	11	12	13	14	15	0	0
0	0	16	17	18	19	20	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
6	6	6	7	8	9	10	10	10
11	11	11	12	13	14	15	15	15
16	16	16	17	18	19	20	20	20
16	16	16	17	18	19	20	20	20
16	16	16	17	18	19	20	20	20

13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8
3	2	1	2	3	4	5	4	3
8	7	6	7	8	9	10	9	8
13	12	11	12	13	14	15	14	13
18	17	16	17	18	19	20	19	18
13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8

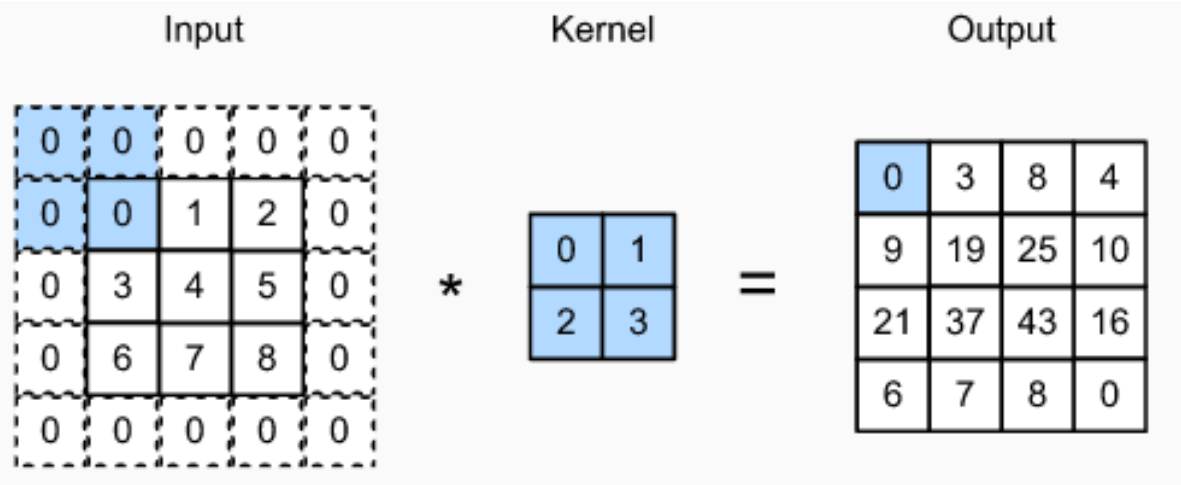
Figure: Zero, replicate and mirror padding

Zero Padding



- Various "modes" of zero padding
- For a padding of k pixels
  - "valid": no padding
  - **"same"**: add $\frac{k-1}{2}$ zeros on each side
  - "full": add k zeros on each side

Feature map after padding



- The output size of the convolution of a $n_h \times n_w$ with a kernel of size $k_h \times k_w$ is

$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

- If we add p_h rows and p_w columns of padding the output size becomes

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- Generally we want to have $p_h = k_h - 1$ and $p_w = k_w - 1$ which is achieved by applying **same zero padding**

Stride

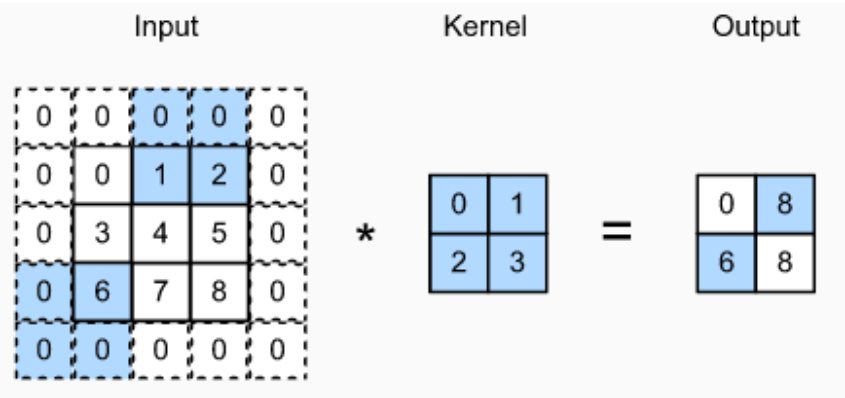


Figure: Convolution with height stride of 3 and width stride of 2

- Currently, we slide the kernel across all locations right and down
- We can **skip** some spatial locations
  - It reduces the feature map output
  - It reduces the computational cost
  - It reduces the accuracy of the representations
- If we add s_h, s_w for the height and width strides the feature map size is

$$(n_h - k_h + p_h + 1 + s_h)/s_h \times (n_w - k_w + p_w + 1 + s_w)/s_w$$

Pooling

Pooling

Pooling statistically summarizes a neighborhood

- A pooling layer takes a window of values and output one value
- Several pooling types
  - **Max pooling**: take the maximum value of a window
  - **Average pooling**: take the mean value of a window
  - l^2 norm, weighted average pooling, ...

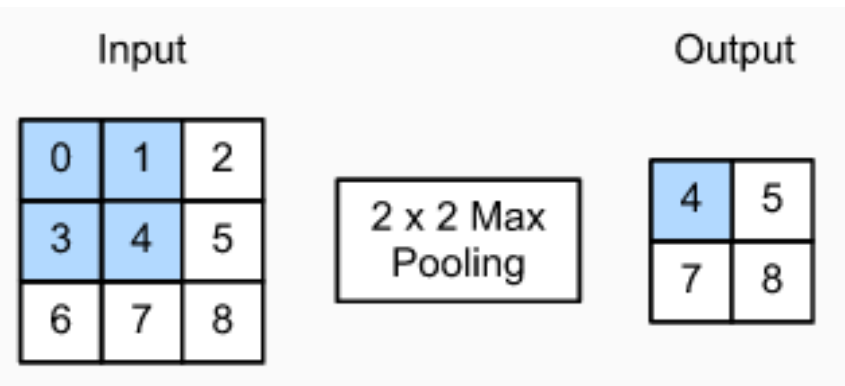


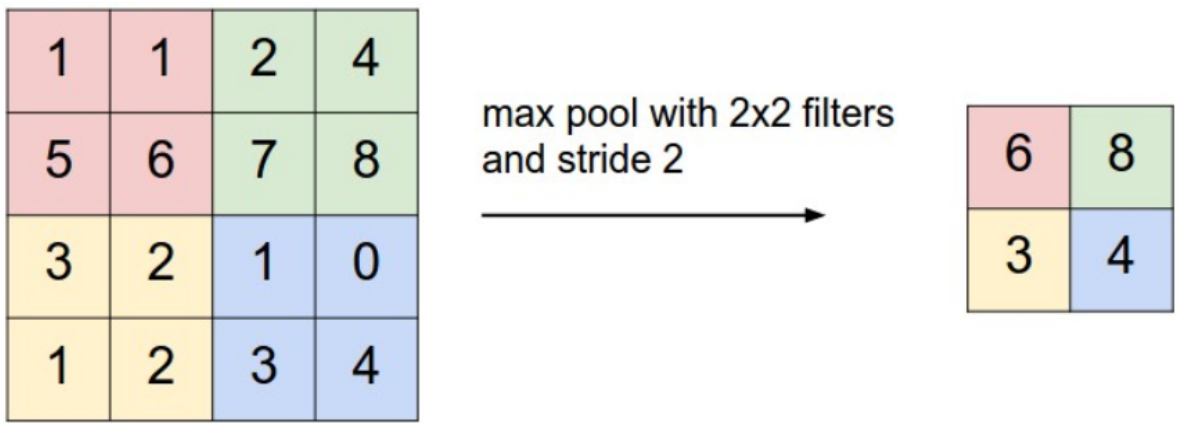
Figure: Max pooling of size 2x2

Why use pooling ?

- Make our representations invariant to **minor translations**
- For deeper layers minor modifications can mean great changes in the input image
- It reduces the amount of parameters and computation
- Help to **reduce overfitting**
- It sets up a strong prior: invariance to local translation of learned kernels
- Can be degrading for some tasks where precise locations is required

Stride and padding for pooling layers

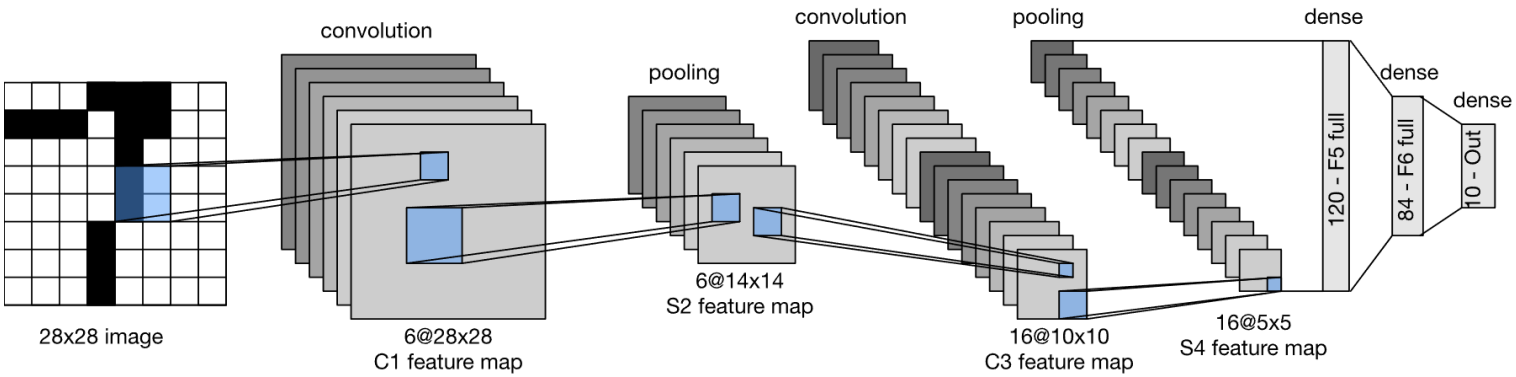
- Pooling layers can use the same tools as kernels
- Stride and padding are both used in pooling layers to learn adequate representations



Plan

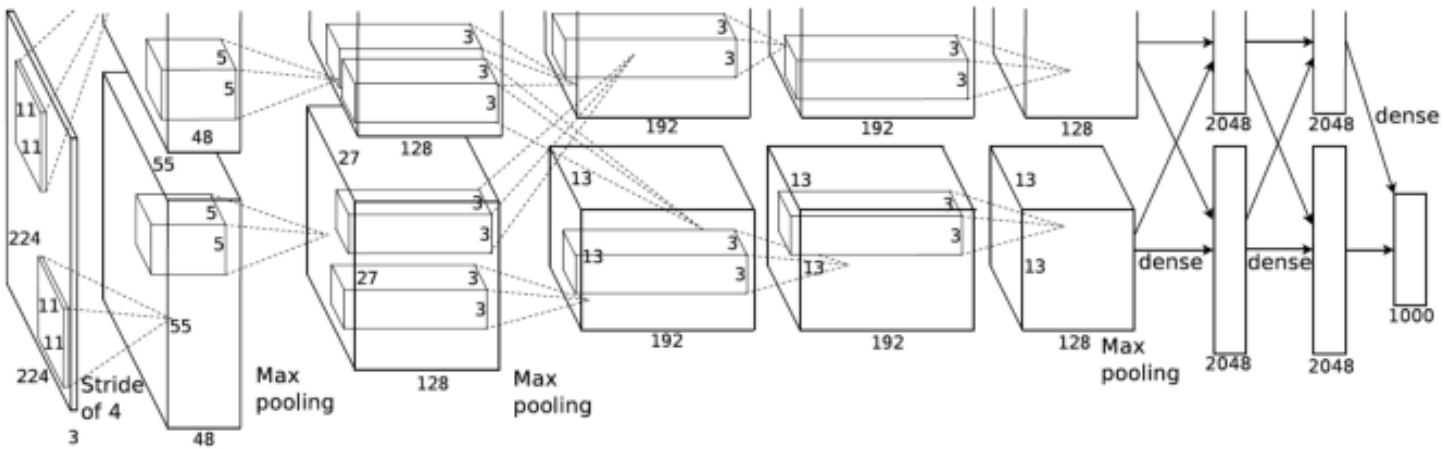
LeNet, LeCun et al. 1998

- Designed after 10 years of working on handwritten bank checks



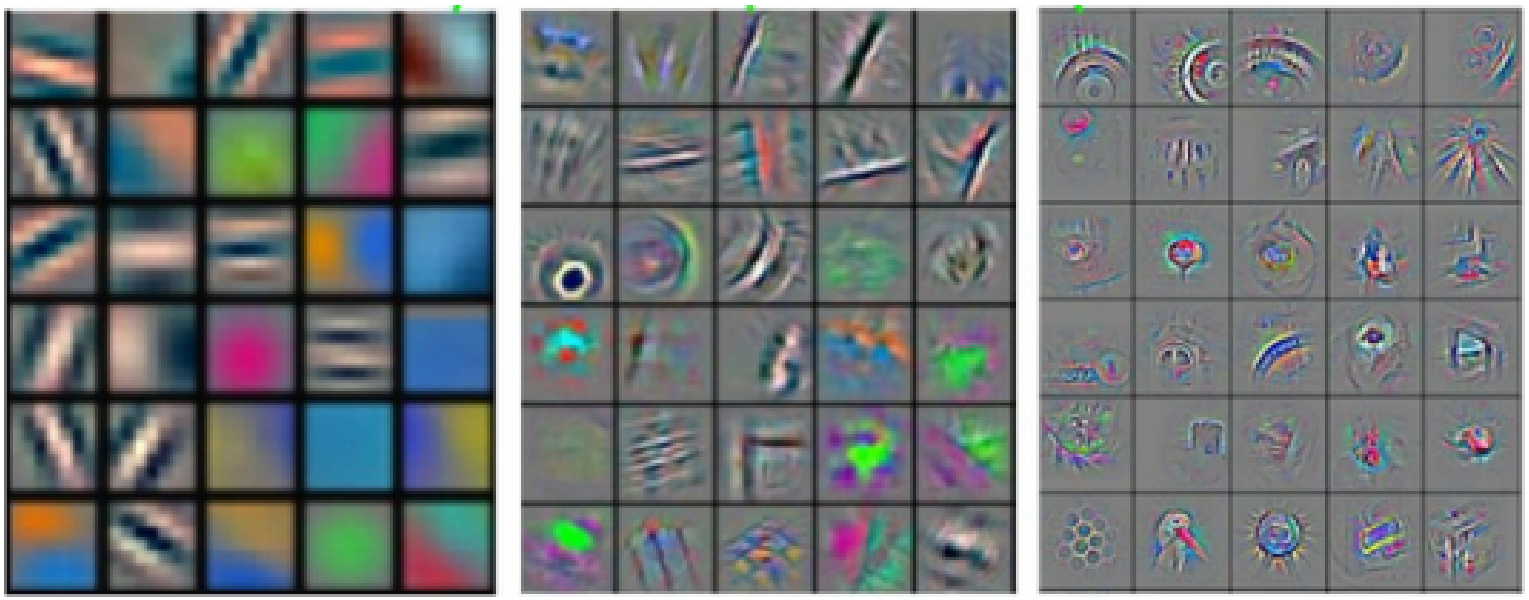
- Paved the way to basics of Deep Learning
  - **Feature extraction** by convolution layers
  - **Classification** by MLP layers
  - Layer of **convolution** followed by **pooling** (avg) followed by **non-linearity** (sigmoïd and tanh)
  - SGD optimizer to perform backpropagation

AlexNet, Krizhevsky et al. 2012



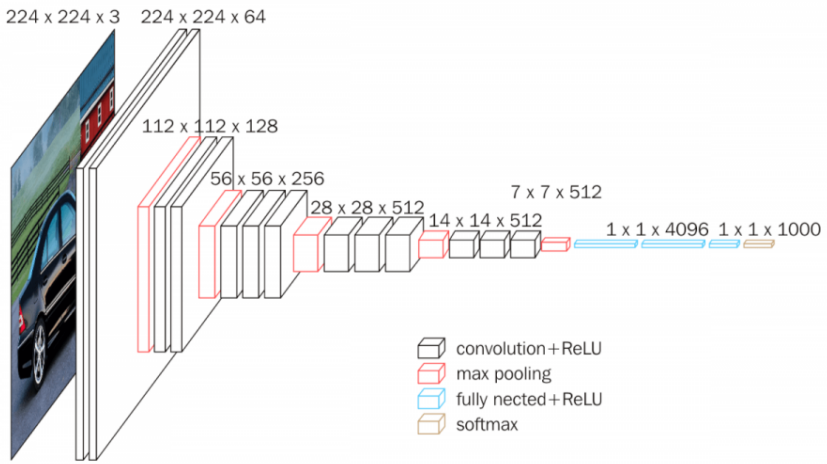
- First **Deep Neural Network**
- Won the 2012 ImageNet challenge that caused a new Deep Learning area
- Extended LeNet to larger images with wider convolutions
- Use ReLUs and max pooling
- Use dropout

AlexNet: filters learned at each depth



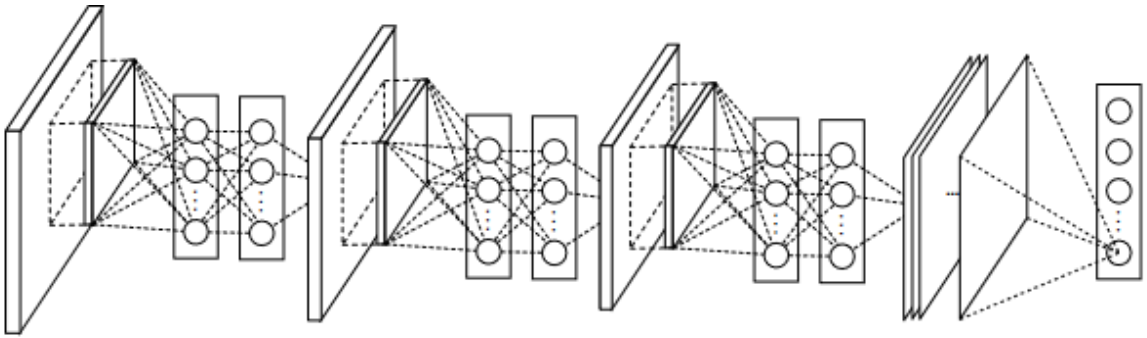
- As layers are deeper, the receptive field is higher
- First layers learn basic **patterns**
- Deeper layers learn more complicated shapes

VGG: Visual Geometry Group, Simonyan et al. 2015



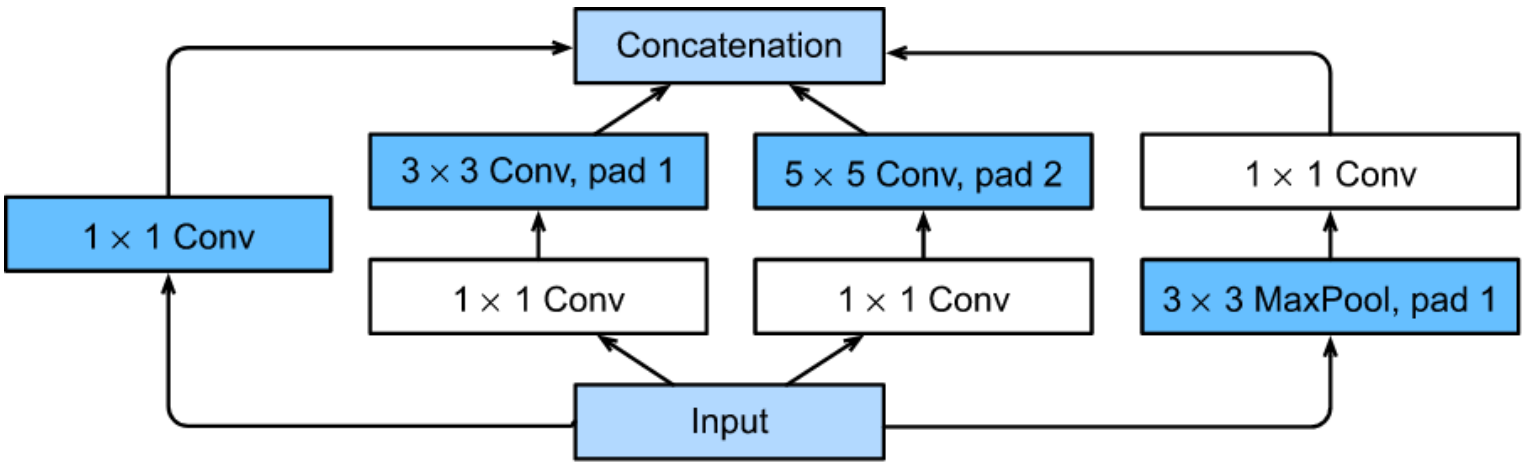
- First network with **blocks**
  - Convolution layer with padding to keep same resolution
  - Non linearity
  - Pooling layer to reduce resolution
- Problem: maintaining enough resolution to have lots of blocks
- Solution: more convolutions at low resolution instead of few convolutions at high resolution
- Increases the number of non linearity and reduce the number of parameters for equal receptive field

NiN: Network in Network, Lin et al. 2013



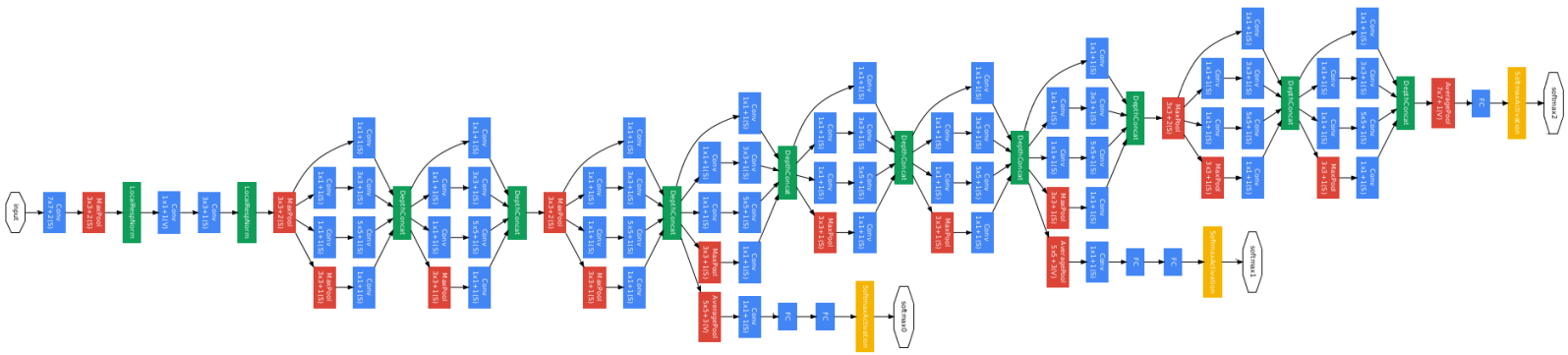
- Previous architectures improved each others by enhancing the number of convolutions (**width**) and deepening the network (**depth**)
- Two problems:
  - Last fully connected layers consume large number of parameters
  - How to add non linearity without fully connected layer that would destroy the spatial structure
- Network in Network improvements:
  - Use 1×1 convolutions to increase the number of non-linearity
  - Global average pool at the end to remove large fully connected layer

Inception Blocks



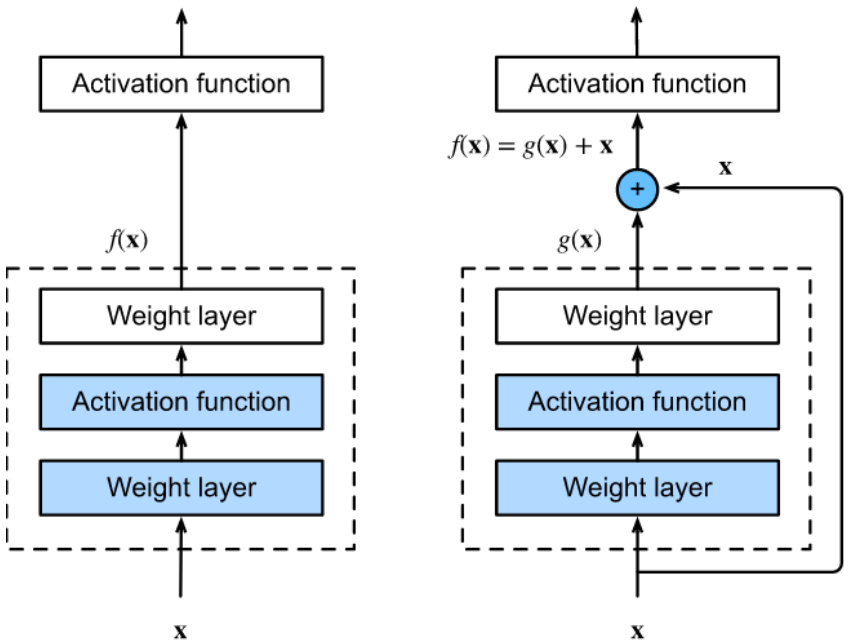
- Google faced a problematic: **fast deployment and inference**
- Large convolutions with lot of channels increase inference time
- Solution: reduce number of channels, do convolution, dilate the number of channels

GoogleNet, Szegedy et al. 2015



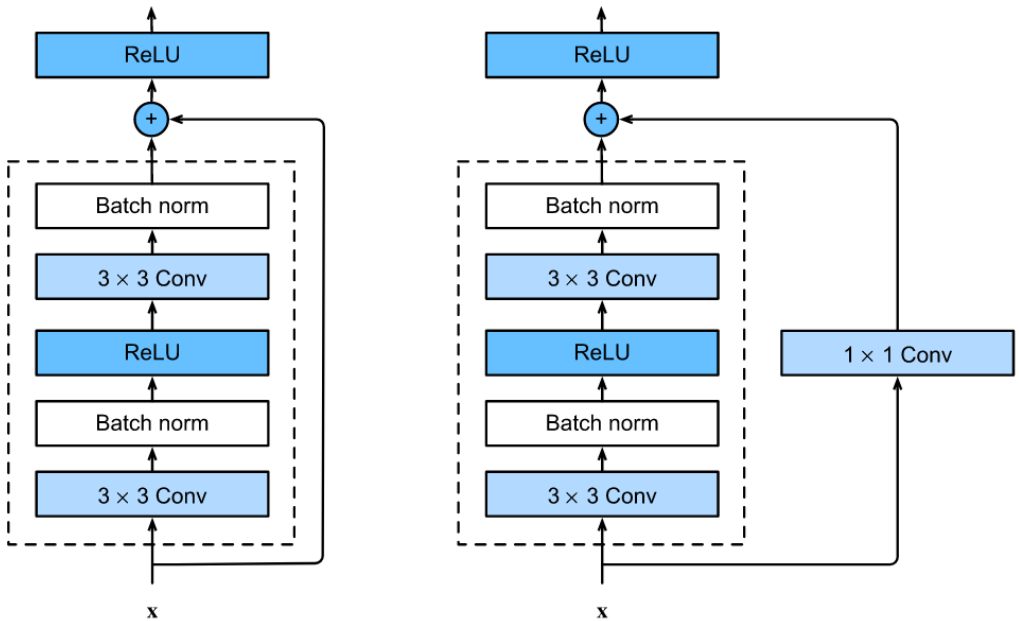
- Based on Inception blocks
- Use several classifiers throughout training to enhance discriminative features at first layers and reduce vanishing gradient

Residual Blocks



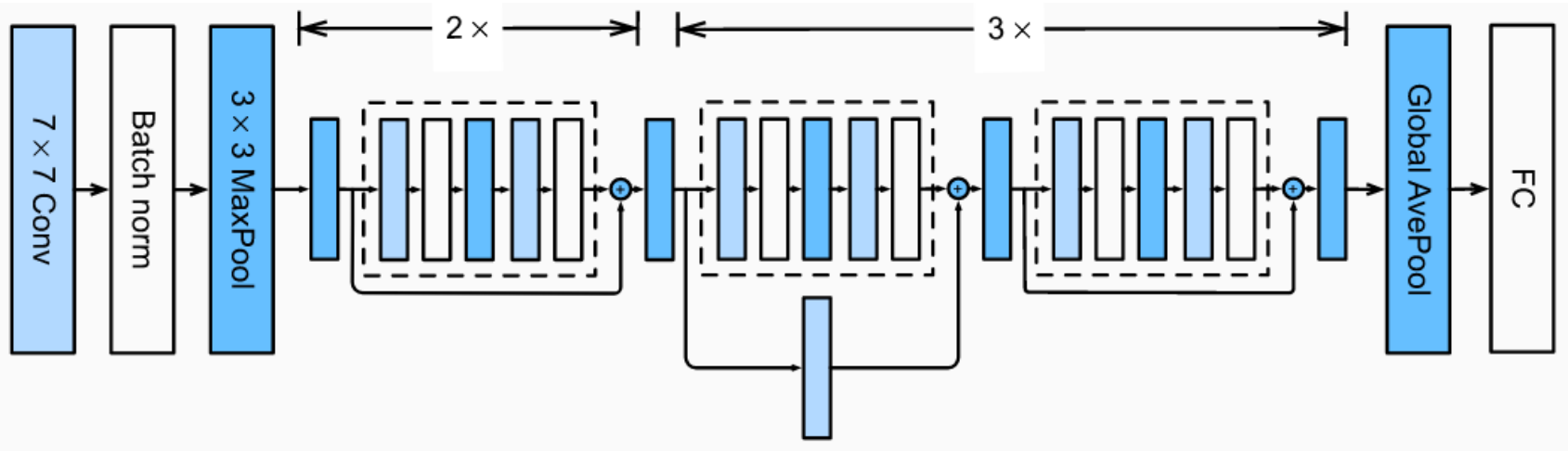
- Residual blocks make use of **residual connection** (or **shortcut connection**) to keep the input information
- Make the identity function easier to learn
- Reduce vanishing or exploding gradient problems

ResNet blocks



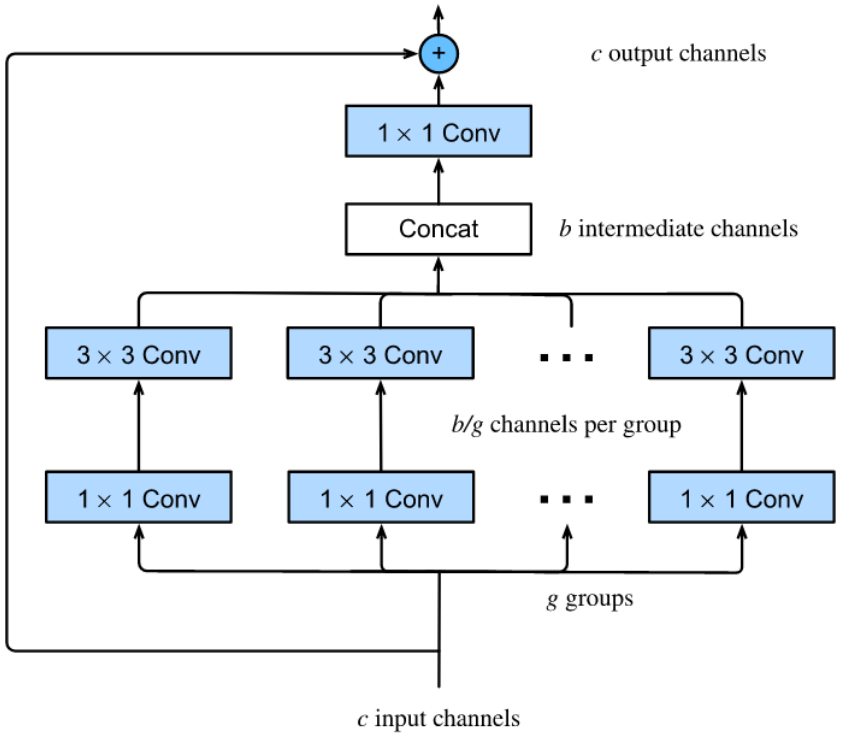
- As VGG, ResNet blocks use 3×3 convolutions
- It adds batch normalization to stabilize training
- Residual connection are concatenated with the result of two convolutions

ResNet: Residual Neural Networks, He et al. 2016



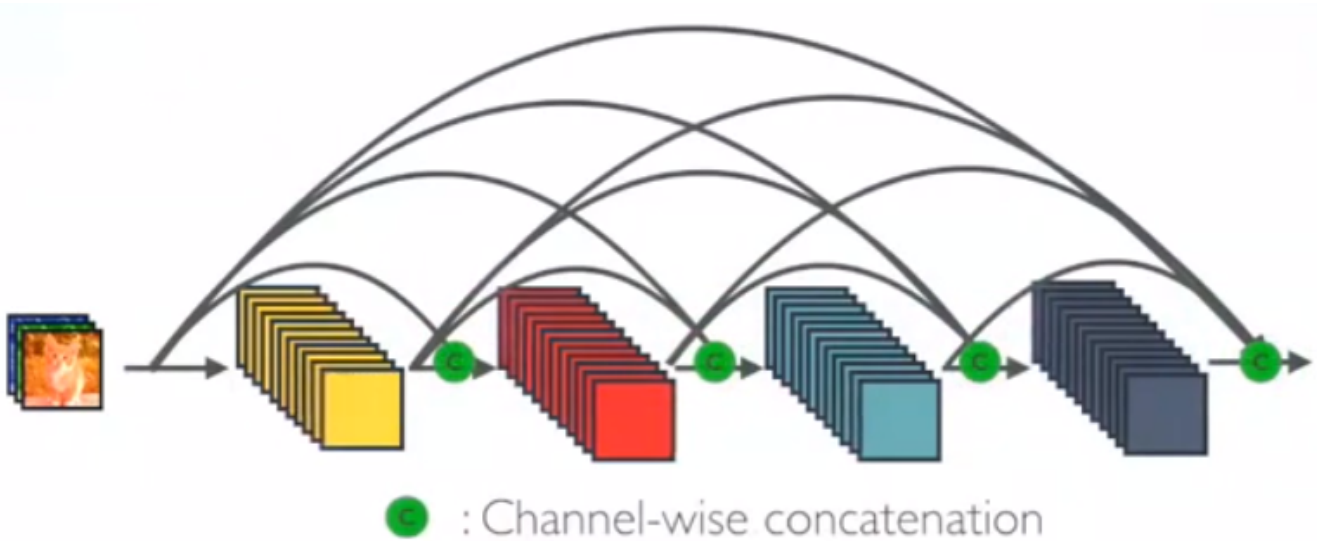
- ResNet is composed of **blocks of residual blocks**
- There are different ResNet depending on how many blocks are used (18 is illustrated)
- It is a widely used backbone until this day

ResNexts: Group convolution, Xie et al. 2017



- To add more nonlineary in ResNets we could: add more layers, increase the width convolutions or the number of channels
- Problem: increasing number of channels add quadratic complexity ($F_{in} \times F_{out}$)
- Group convolution solve this problem

DenseNet, Huang et al. 2017



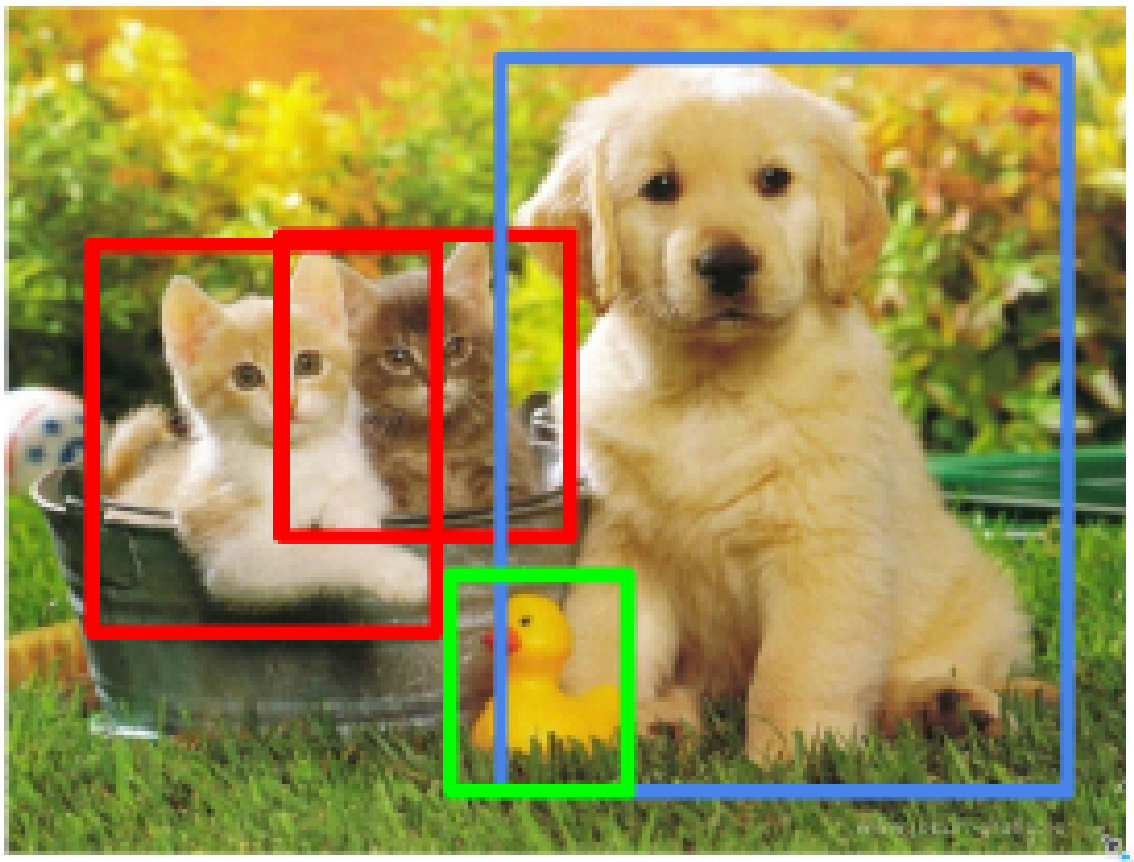
- Based on ResNet blocks
- Instead of adding input and result of convolution, it concatenates both of them plus the results of each Dense blocks on the channel dimension
- Use of **transition layers**: 1×1 convolutions after concatenation to reduce feature dimension

More architectures

- For deployment on mobiles: MobileNets, Howard et al. 2017, that separate convolution in two operations
- EfficientNet, Tan et al. 2019, that grid search without computational budget how to modify the architecture at best (width, depth or input resolution)
- ...
- Convolutional Neural Networks slowly being replaced in some cases by **Visual Transformers** (future class about transformers)

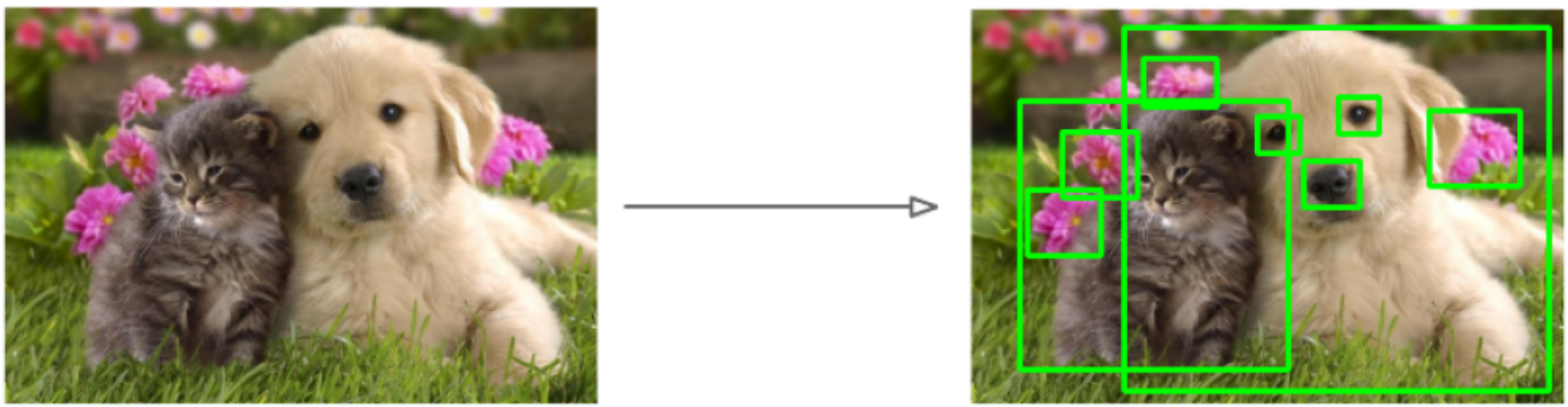
Goal of Object Detection

- Find a **bounding box** that contains an object of interest
- Two objectives:
  - Regression to shape the box
  - Classification to identity the object



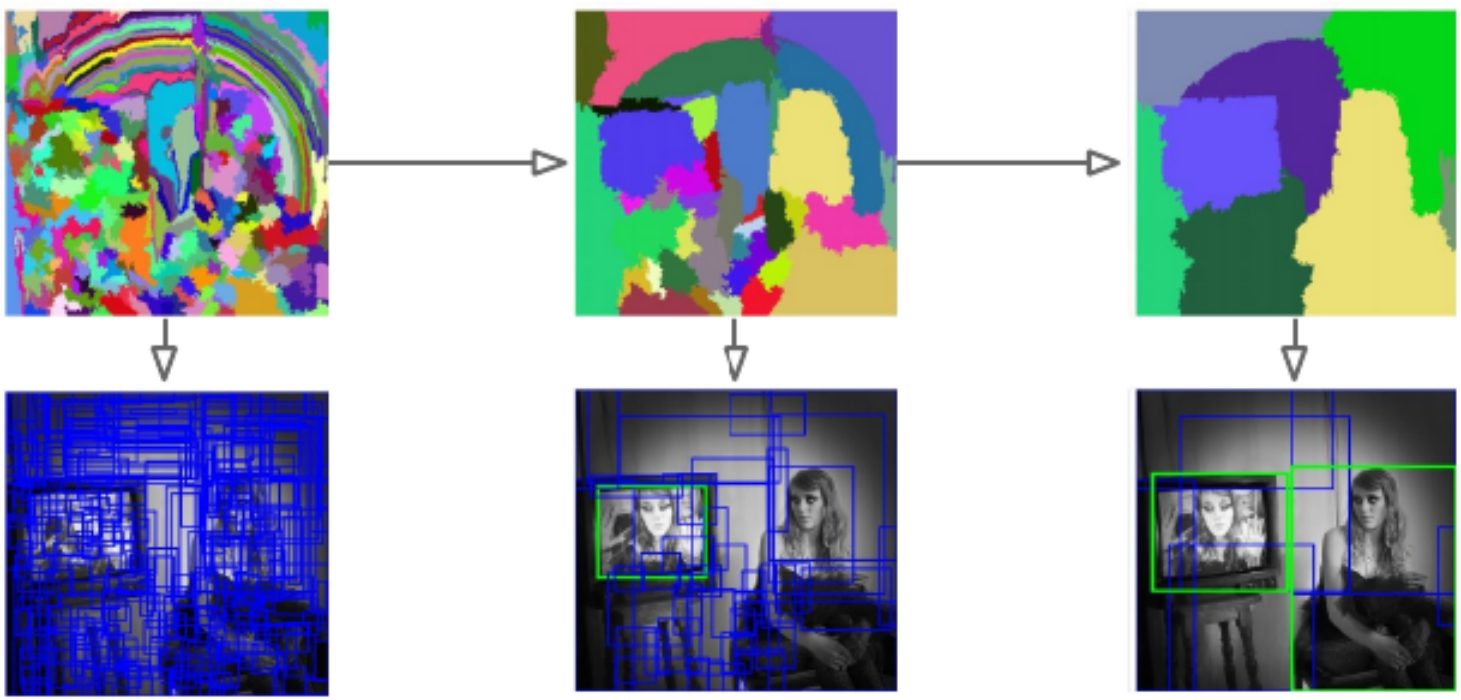
How to consider the detection problem ?

- As a classification problem ?
  - Test various positions and scales to propose various **anchor boxes**.
  - Scalable only if the classifier is fast enough
- However CNNs are computationally expensive
- **Solution**: select a subset of boxes
- find blobs likely to contain objects without class consideration (objectness)



Region proposal: selective search, Uijlings et al. 2013

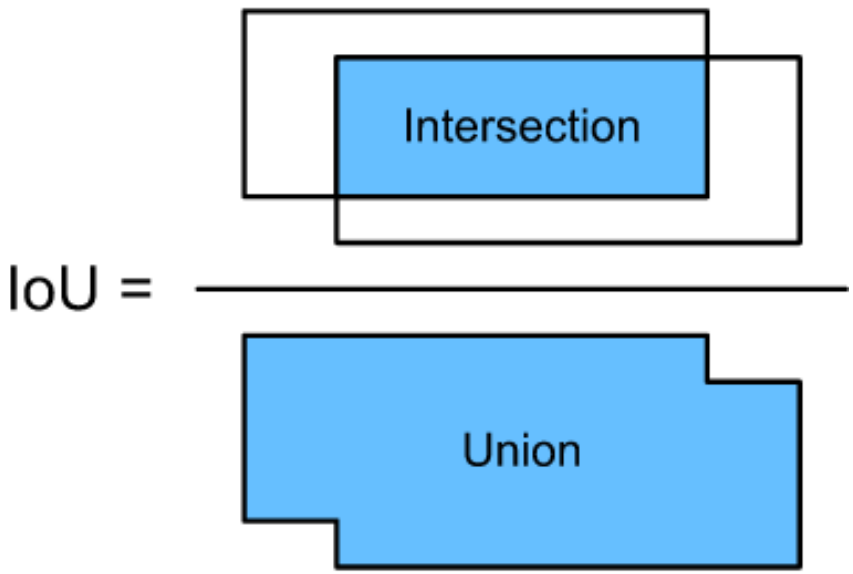
- Selective search is a widely used tool to provide region of interest in an image
- The regions are computed following this pipeline:
  - Ascendant segmentation
  - Fuse regions at different scales
  - Convert regions in potential boxes of interest



Intersection over Union (IoU)

- For two sets A and B the Jaccard index or **intersection over union (IoU)** is the ratio of the intersection area to their union area

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$



Label region proposal

- Region proposal anchor boxes are associated to ground-truth bounding boxes given their IoU (above a threshold)
- The class of an anchor box is the same as its associated bounding box
  - If the anchor box is not associated to a bounding box it is labeled as **background**
- The **offsets**, relative to the position of central coordinates between anchor box and bounding box, labels the regression

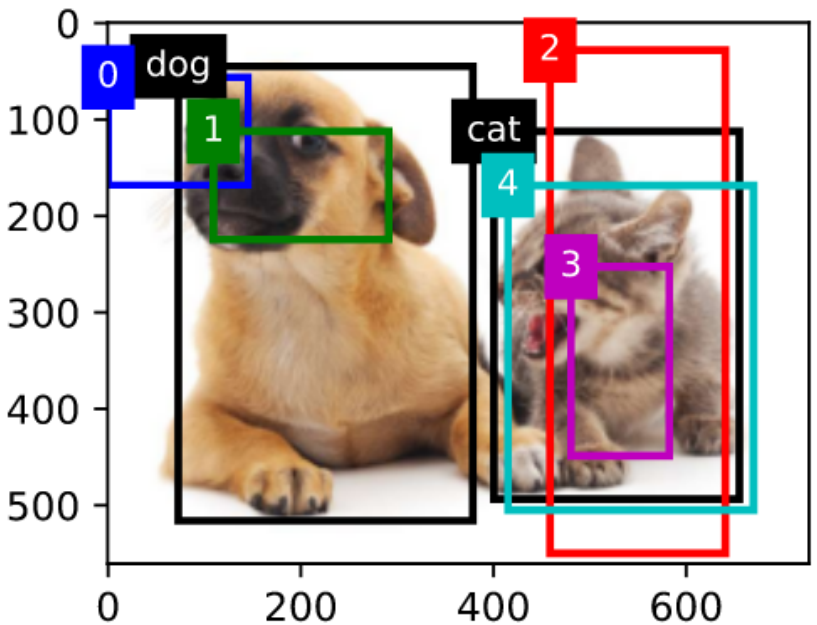
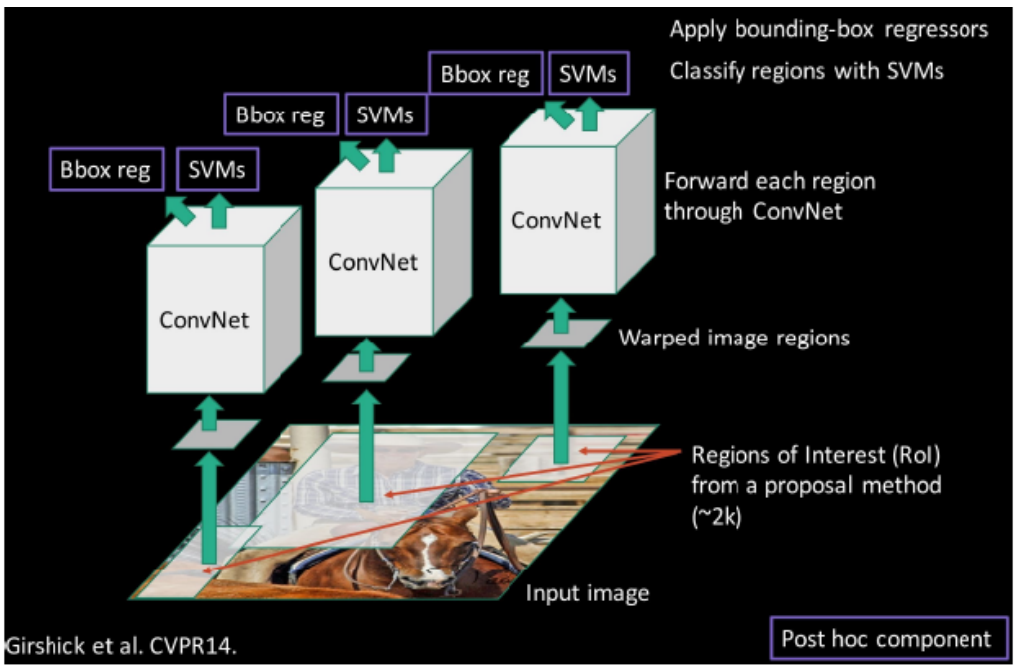


Figure: 1 is labeled dog, 2 3 4 labeled cat and 0 is labeled background

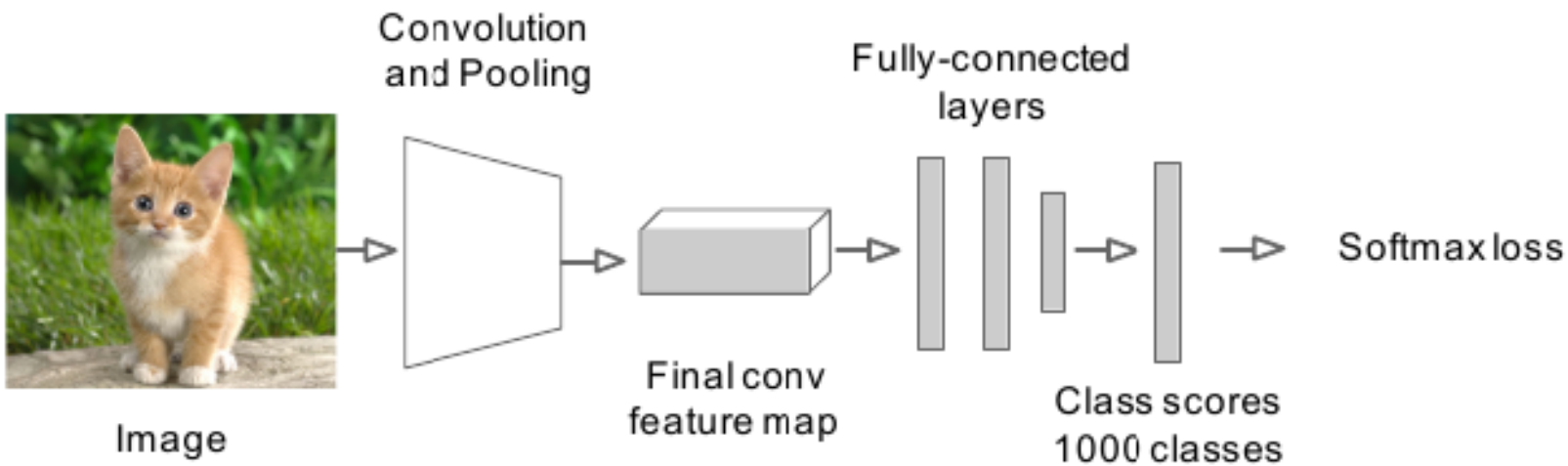
R-CNN, Girshick et al. 2014

- R-CNN takes region proposal as input
- It compute features for each of the regions
- The classification is done using several SVMs
- Regression of offsets is done by using a linear regression model



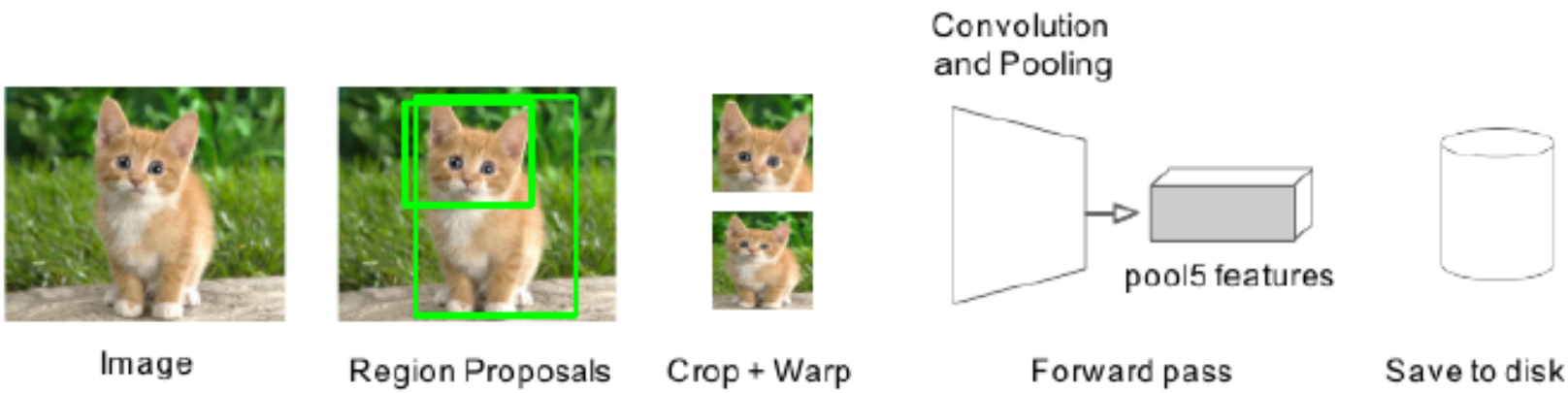
Training R-CNN 1/2

- Supervised **pretraining** of a CNN backbone (generally on ImageNet)
- Optional **fine-tuning** of the backbone to learn specialised features based on a classification objective
- The backbone is then **fixed**



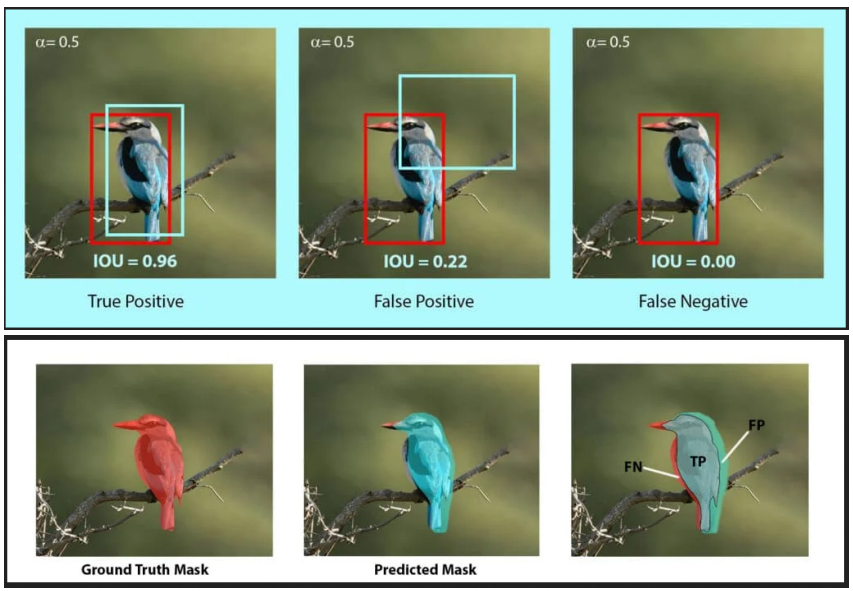
Training R-CNN 2/2

- For each input: crop and wrap proposal regions computed by selective search and compute the features
- Train a binary SVM for each class
- Apply linear regression to correct small offsets between the prediction and the actual bounding box



Evaluation of a detection model 1/2

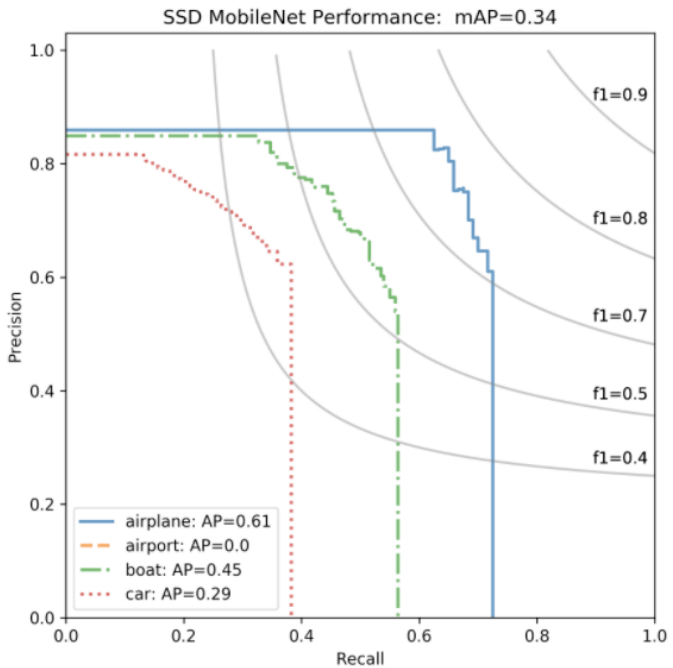
- True positive, false positive and false negative for detection problem is not the same as for Classification as detailed below
- The **threshold** for True and False positives/negatives is based on the IoU and a **threshold**



- $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$

Evaluation of a detection model 2/2

- **Average precision** (**AP**) is the area under the precision-recall curve



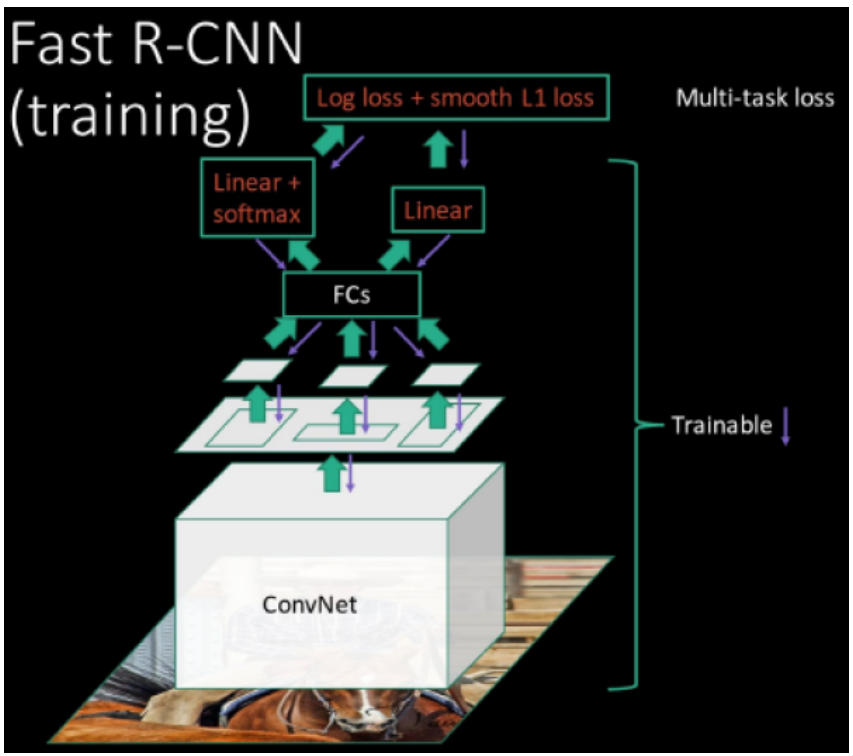
- **Mean Average Precision** (**mAP**) is the mean of Average Precision for various IoU thresholds

R-CNN limits

- R-CNN is Slow in testing phase because there are as many passes in the CNN as the number of region proposals (around 2000)
- SVM and regression are a bit old school
- R-CNNs is therefore not usable for real world applications

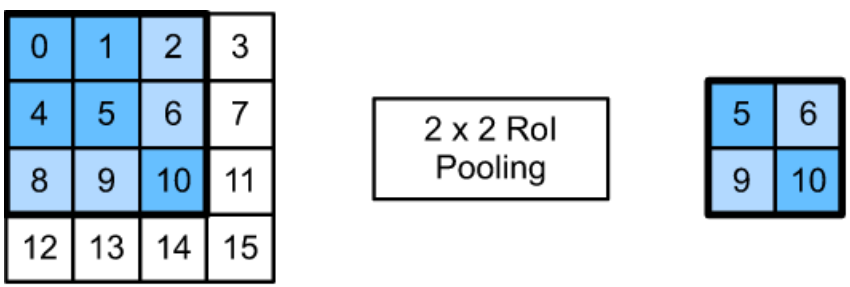
Fast R-CNN, Girshick 2015

- Fast R-CNN has been proposed to make R-CNN more efficient
- It passes the whole image only **once** in the CNN backbone
- The **CNN is trainable** and not fixed
- The proposed regions are **associated** with computed features from the output feature map



Fast R-CNN: Region of Interest Pooling

- Each region proposal can have a size different
- Selective search proposed regions are concatenated with features from the CNN to form the **Region of Interest (RoI) Pooling layer**
- This RoI pooling layer reshape the features to feed them to fully connected layers
- From these features the classes and the offsets are predicted



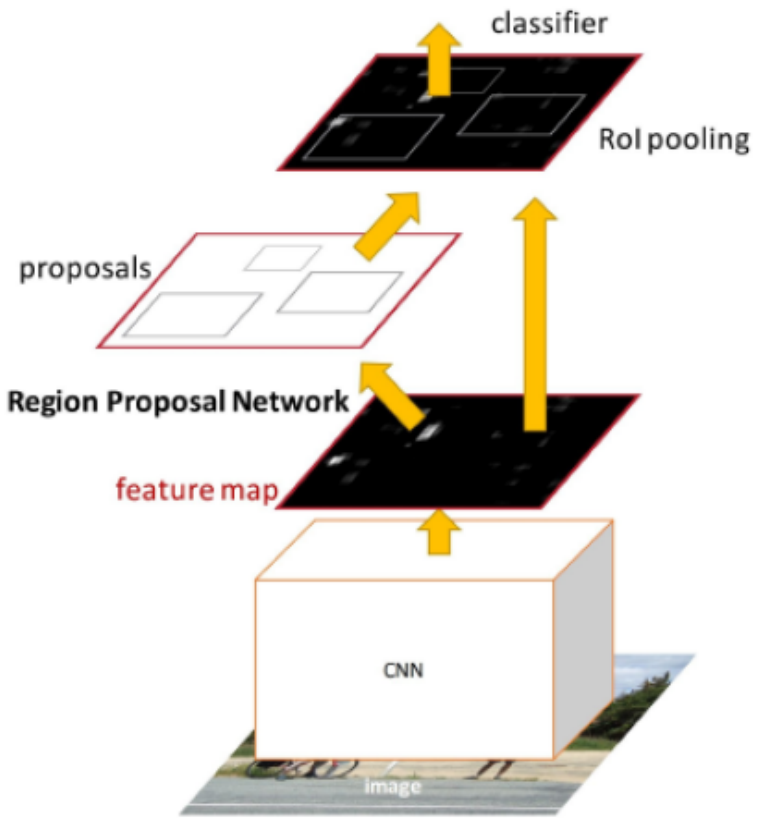
Speed-up thanks to Fast R-CNN

	R-CNN	**Fast R-CNN**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
Test time per image with Selective Search	50 seconds	**2 seconds**
(Speedup)	1x	**25x**

- Great speedup thanks to Fast R-CNN
- Problem: these results do not include the region proposal phase that is costly in time
- Faster R-CNN solve this issue

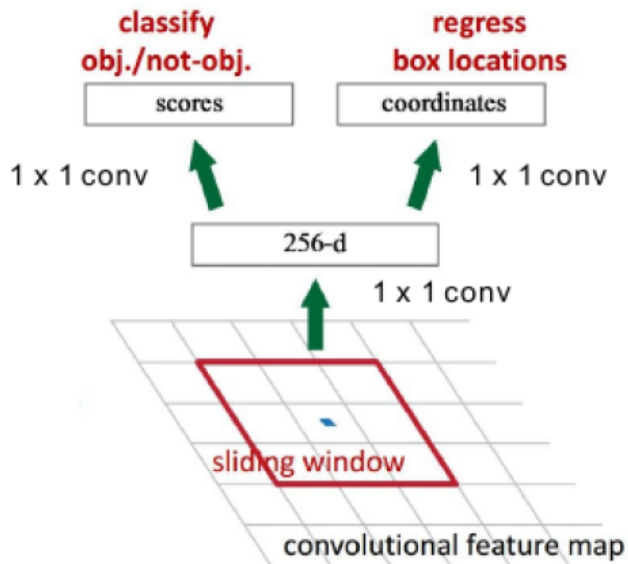
Faster R-CNN, Ren et al. 2017

- Selective search is replaced by a **learned** Region Proposal Network
- The rest is similar to Fast R-CNN



Faster R-CNN: Region Proposal Network (RPN)

- The **Region Proposal Network (RPN)** is a network learned to propose regions of interest
- It slides a window on the feature size
- At each localisation of the window it makes a prediction for k **anchors** (propositions)
  - These anchors are sampled by varying scale and aspect ratio
- A small network predict if there is an object
- A Small network predict offsets with the bounding box



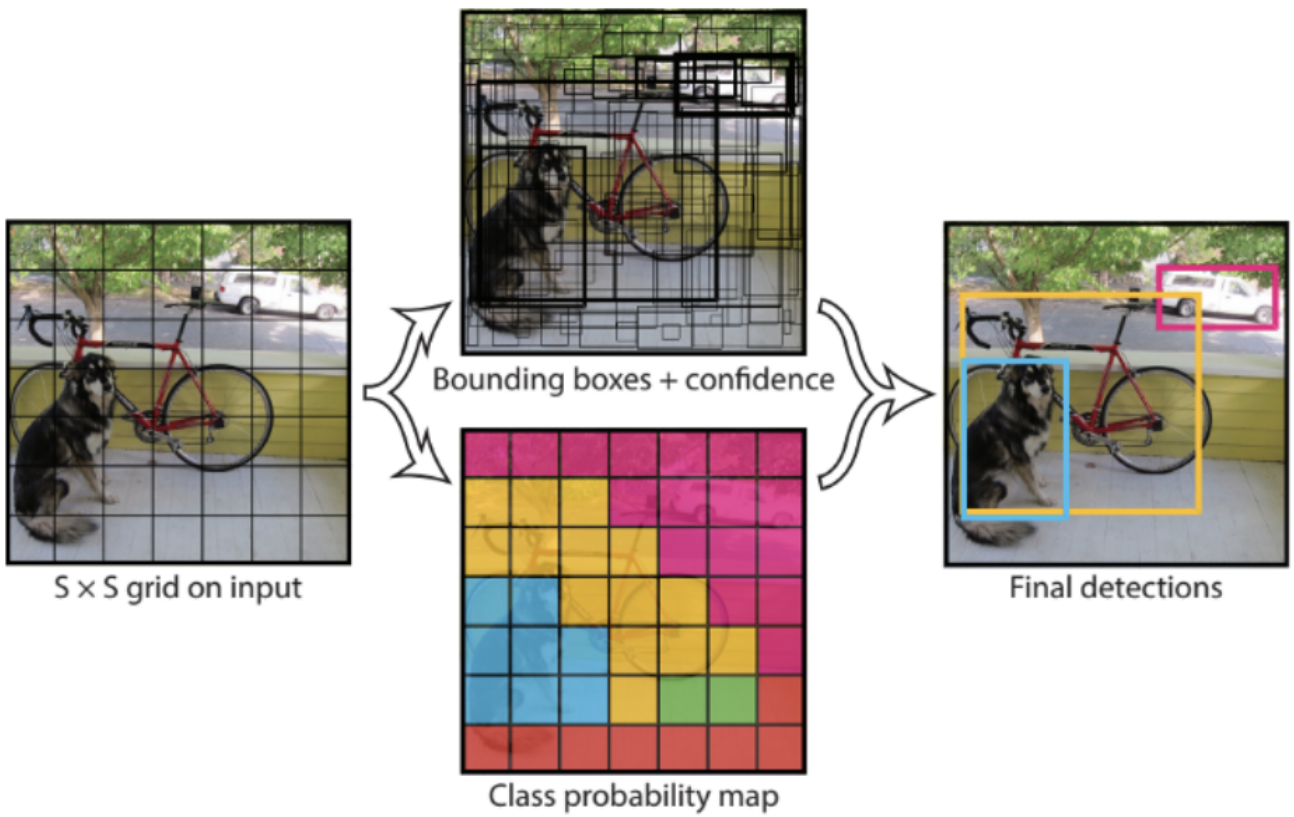
Speed-up thanks to Faster R-CNN

	R-CNN	**Fast R-CNN**	**Faster R-CNN**
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

- Close to CNN state-of-the art but rather slow
- R-CNN are **two-stage** detectors:
  - extraction of regions
  - classification and refining of localisation

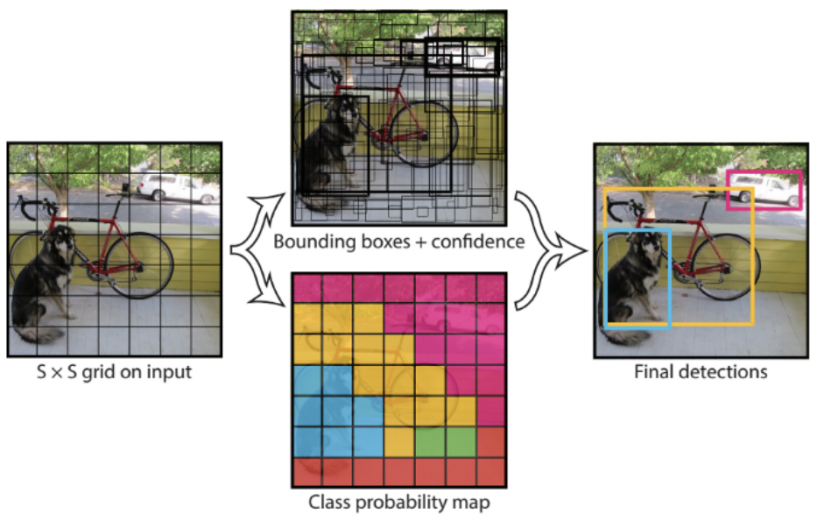
You Only Look Once (YOLO) Redmon et al. 2016

- Single stage dector: one network to predict the bounding boxes and the class probabilities
- First real time detector close to Faster R-CNN performance



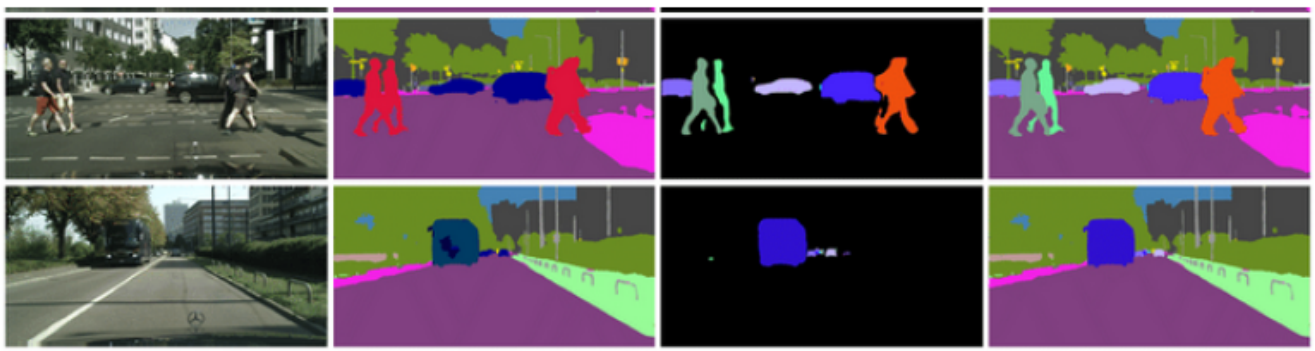
YOLO

- Architecture of 24 convolutional layers and input dimension of 448×448
- For each input, it cuts it in 7×7 grid of cells
- Each cell computes the **objectness** and **anchor boxes**
  - The object is considered at the center of the cell
  - The boxes can reach out the cell
- YoLo has been enhanced to YoLo v2, v3, v4, v5 ...



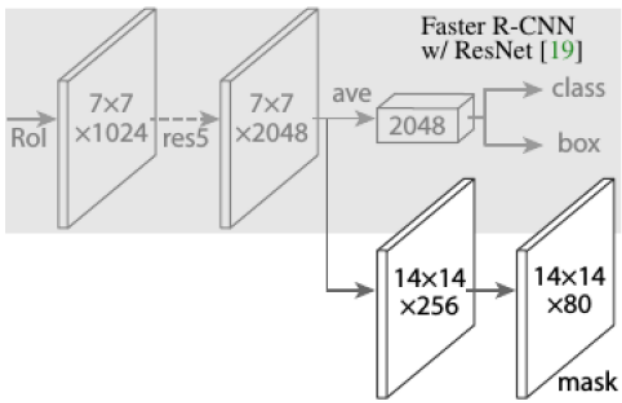
Semantic Segmentation

- Find an object class for each pixel
- Several problems
  - **Semantic segmentation**: Associate each pixel to a specific class
  - **Instance segmentation**: Associate each pixel a specific class and identify various instances from same class
  - **Panoptic segmentation**: Associate each pixel to one class and prevent overlapping segments



Mask R-CNN, He et al. 2017

- Basically: Faster R-CNN with a third branch
- The third branch outputs the **object mask**
- It can use various backbones such as ResNet, DenseNet, ...



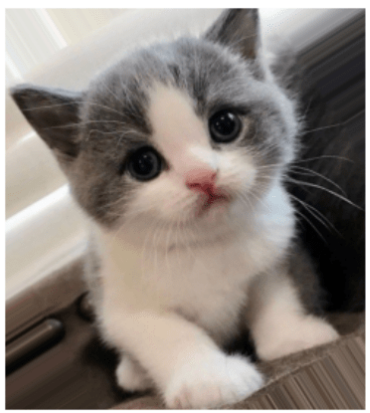
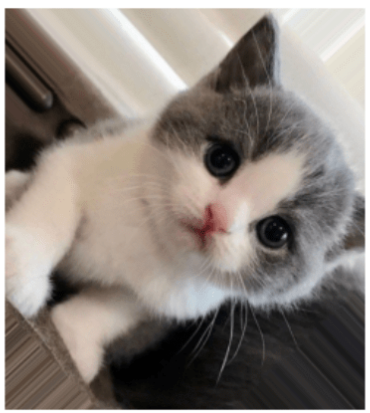
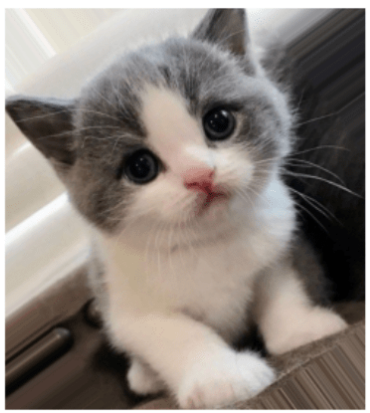
Plan

Need of data in real world

- Modern CNNs are made of **millions** of parameters
- It requires a huge amount of data to learn them
- For example, ImageNet has 1.2M images for 1k classes and is being replaced by ImageNet21k that has 1B images
- Manually annotating such amount of images is **costly** in terms of money and time
- Two techniques to deal with lacking of data
  - **Data augmentation**
  - **Transfer Learning**

Data Augmentation

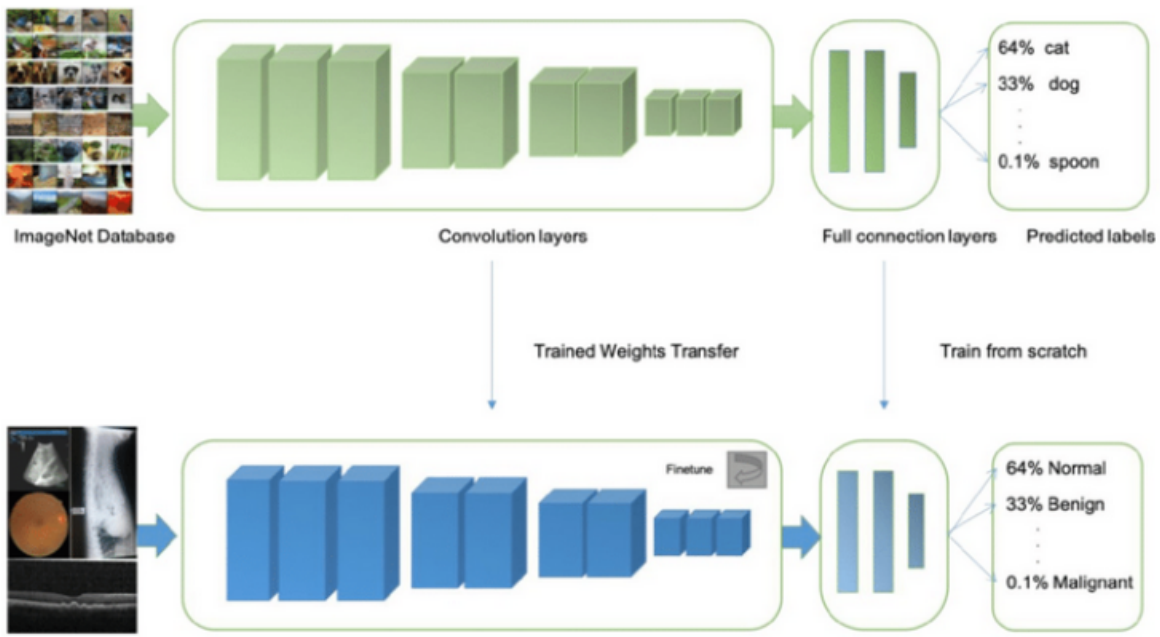
- **Data Augmentation**: increase the **diversity** of data by transforming the source data such as images, using synonyms in text or noising sound
- The goal is to not change the class of the object no matter the transformation
- It removes the cost to label and collect new samples thanks to simple tricks
- Can make the model **invariant** to some transformations

Data Augmentation examples for images

Transfer learning

- Sometimes there is no possibility to learn from scratch (random initialisation) because of **lack of enough data**
  - 10k images with data augmentation can go up to maybe around 200k images $<<$ 1.2M images
- In this case you can make use of an already **pretrained** backbone from a source task for the target task

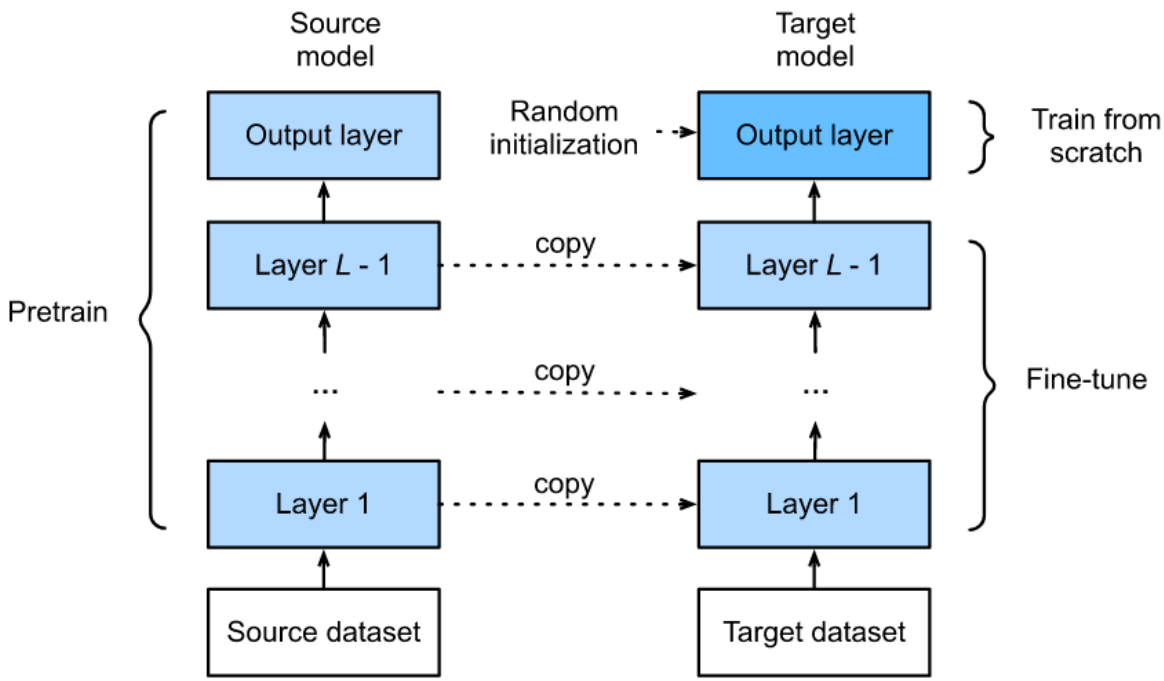


Transfer learning

- If the target task looks like the source
  - If the size is comparable: transfer learning directly works
  - If the size is inferior: risk to overfit to the target task after few epochs
- For small problems the transfer learning is accomplished by learning a linear classifier with last layers of a pretrained backone
- If target task is very different from the source
  - If small problem: classifier linear from lower level layers
  - If large problem: fine tuning
- In all cases, starting from initialized weights is **better than nothing**

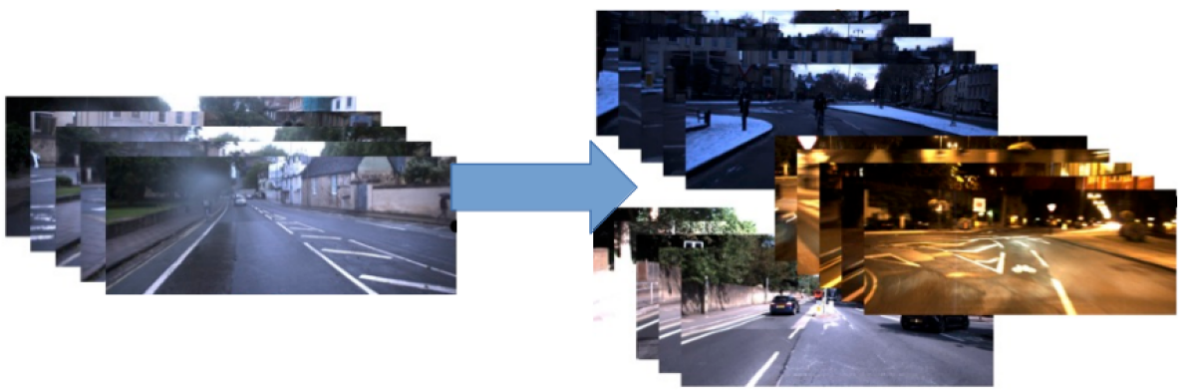
Fine tuning

- Fine tuning consists in training several layers of a pretrained backbone



Domain adaptation

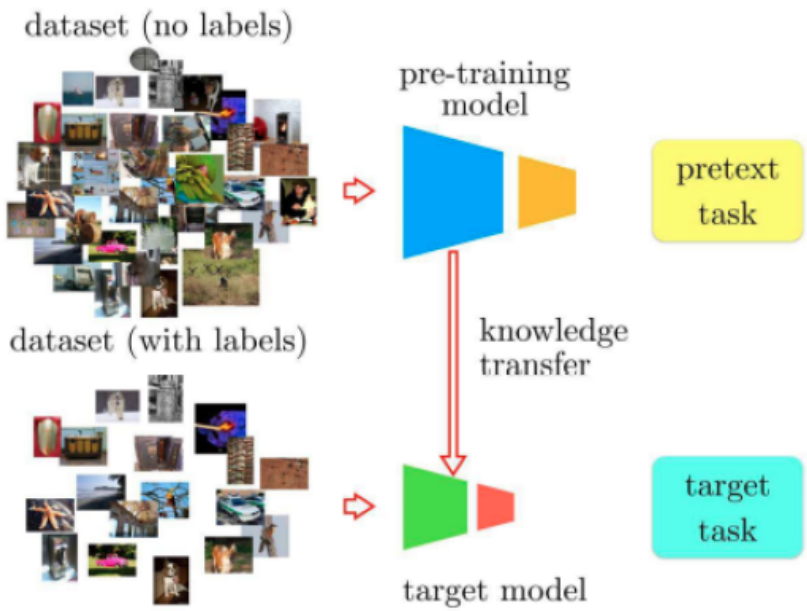
- Sometimes the target task and the source task share the same classes
- The target data has a different distribution than the source (birds from Europe or Africa)
- This is an **entire field in Machine Learning** to properly adapt our features to the new domain and have the best possible results



Plan

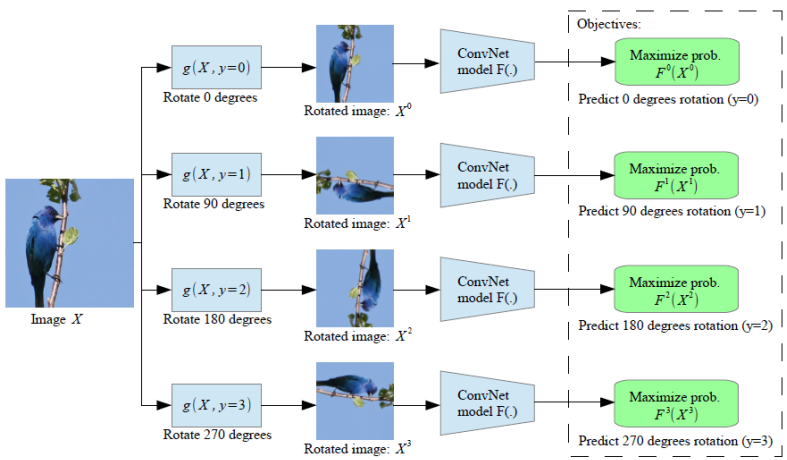
What is Self-supervised learning (SSL) ?

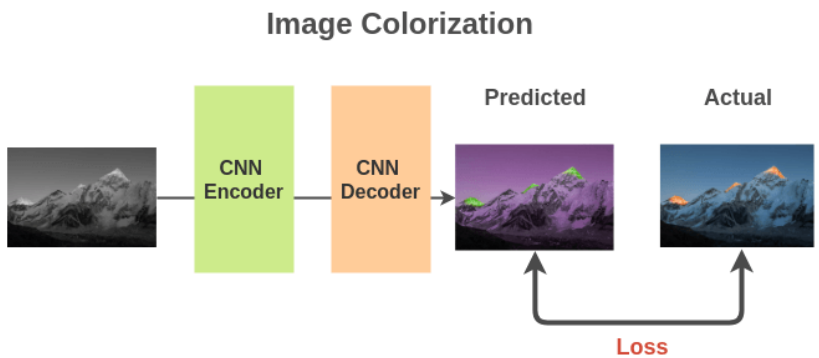
- **Supervised learning**: predict a target (label) associated to an input
- **Self-Supervised Learning**: predict a part of an input from the input
  - Construct a target via a **pretext task** from the input
  - Predict the target
- Free the problem of labeling data
  - Expensive in terms of time, resources and money...
- Improve generalization of neural networks: **better pre-training !**



Pretext tasks for images

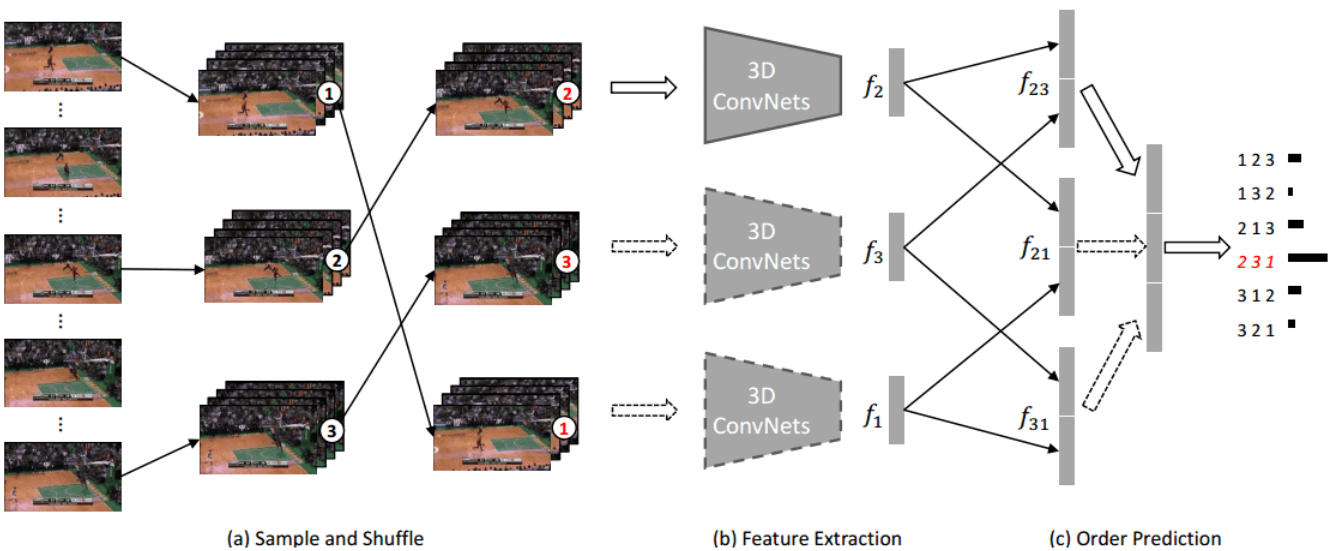
- Various pretext tasks have been firstly proposed
  - rotation, patch localization
  - colorization
  - counting
  - ...





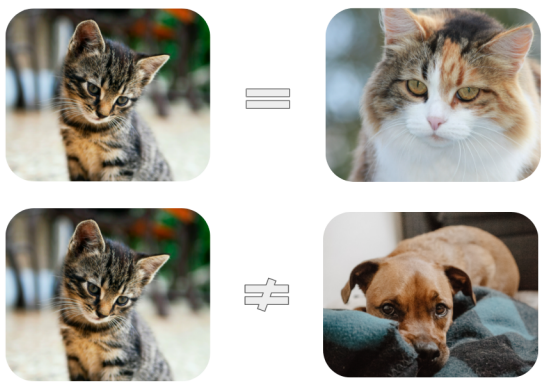
Pretext tasks for videos

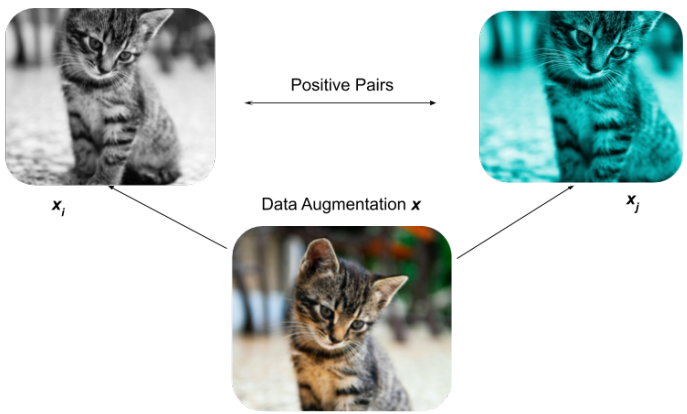
- Videos add the time dimension
- Various pretext tasks have been based on images
  - rotation
  - colorization
  - ...
- Pretext tasks specific to videos
  - mask frames
  - shuffle frames
  - ...



Contrastive Learning

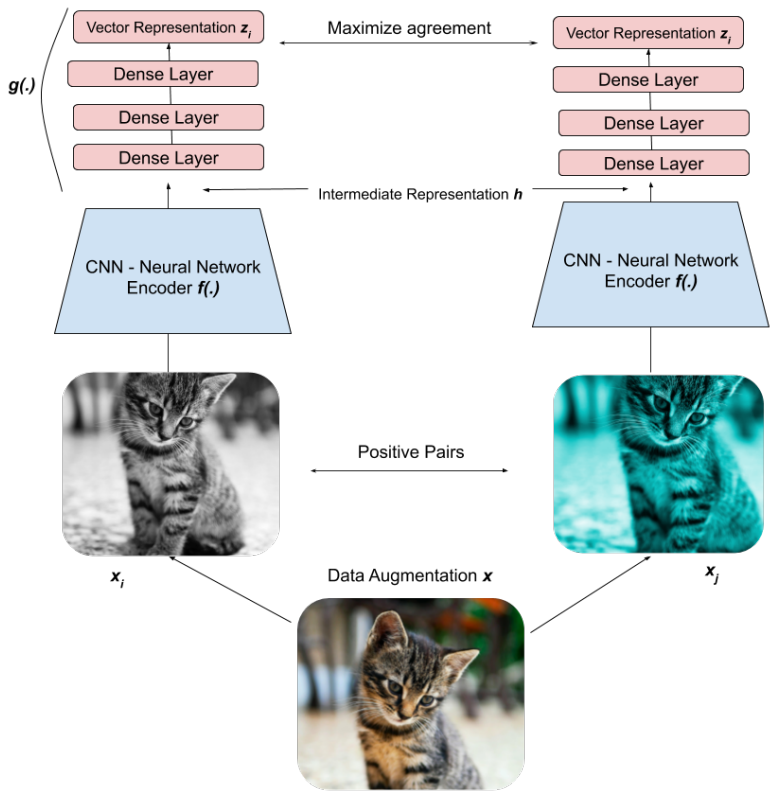
- **Contrastive Learning** is a pretext task for Self-Supervised Learning
- It **aligns** positive pairs of images and **push away** other images
- Problem: what is a positive image ?
- Solution: data augmentation





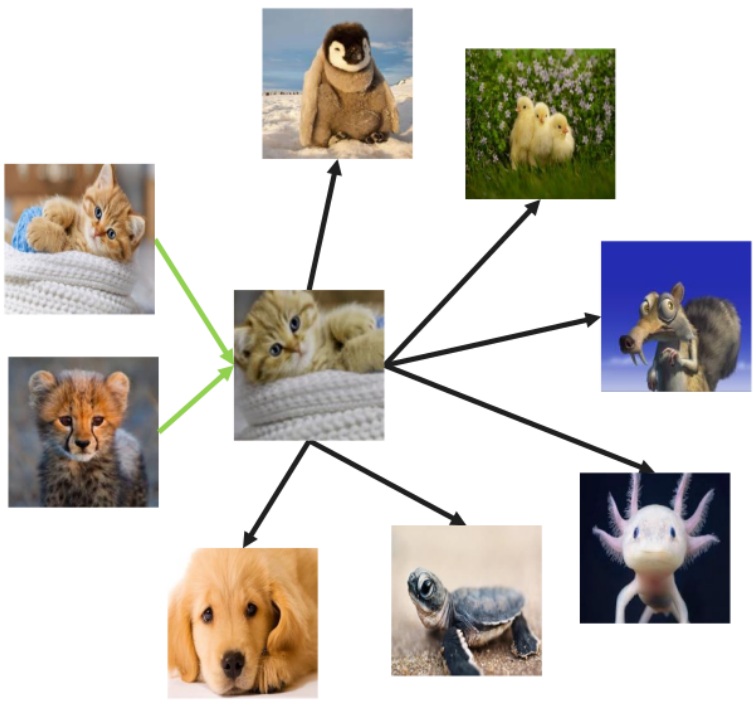
SimCLR, Chen et al. 2020

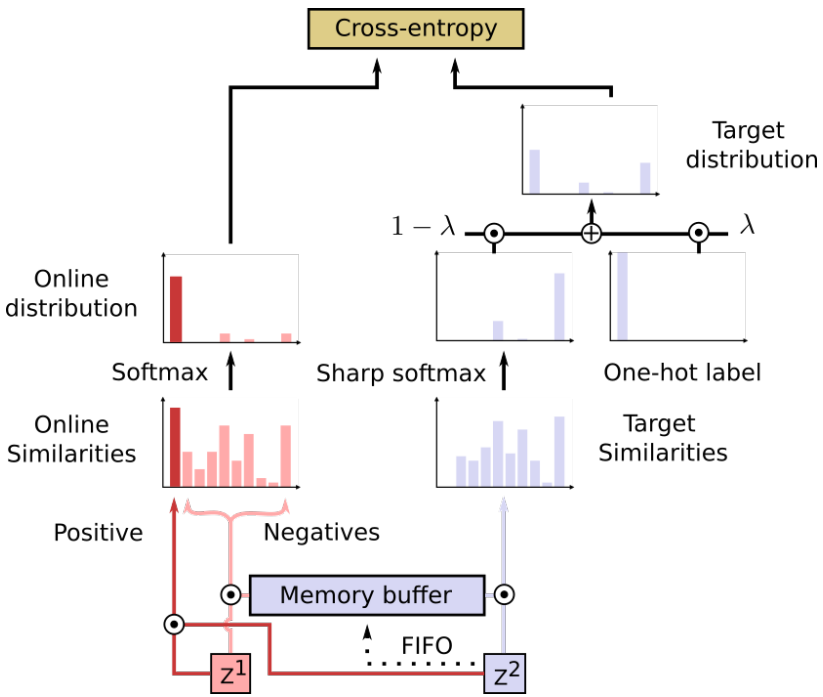
- Define a **siamese** pipeline for contrastive learning
  - Positive pairs from strong data augmentation: color jittering, gaussian blur, grayscale, ...
  - Both pairs go through an encoder and a projector (stacked MLP layers)
  - Contrastive loss applied



SCE, Denize et al. 2021

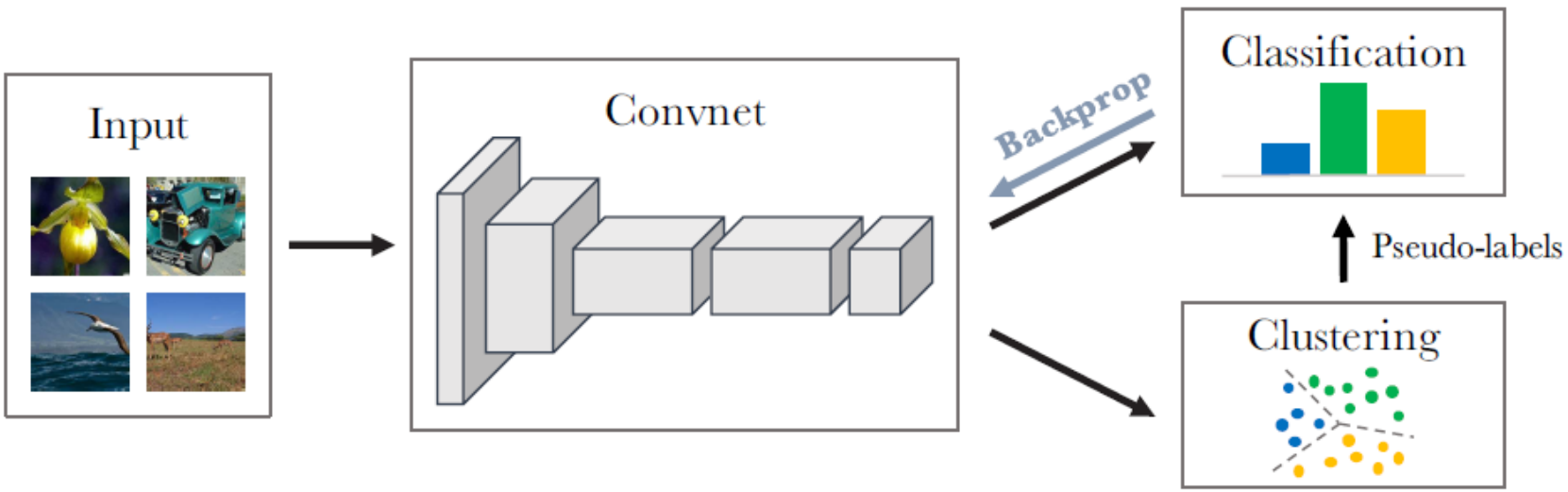
- How to deal with negatives that **share semantic information** with the positive pairs ?
  - two different cats share more information than two different dogs
  - pushing negatives with high semantic similarities makes training unstable
- SCE predicts positive pairs and **estimated relations** among instances





Clustering: Deep Cluster, Caron et al. 2018

- **Clustering** associates several input data to a same **pseudo-label** without supervision
- Deep Cluster applies clustering to train a backbone using several stage
  - Estimate pseudo-label for each instance with a fixed backbone
  - Train the backbone on this label
  - Repeat the operations until convergence



Plan

Conclusion

- Convolutional layers and its properties
  - sparse interactions, parameter sharing, equivariance to translation
- Padding, stride and pooling
- Classification architectures
  - AlexNet, VGG, ResNet
- Detection architectures
  - Two-stage: R-CNN, Fast R-CNN, Faster R-CNN
  - One-stage: YoLo
- Transfer Learning and Data Augmentation
- Self-Supervised Learning
  - Pretext tasks
  - Contrastive Learning
  - Clustering

- Caron, Mathilde et al. (2018). "Deep Clustering for Unsupervised Learning of Visual Features". In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIV*.
- Chen, Ting et al. (2020). "A Simple Framework for Contrastive Learning of Visual Representations". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*.
- Deng, Jia et al. (2009). "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*.
- Denize, Julien et al. (2021). "Similarity Contrastive Estimation for Self-Supervised Soft Contrastive Learning". In: *arXiv*.
- Girshick, Ross B. (2015). "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*.

- Girshick, Ross B. et al. (2014). "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*.
- He, Kaiming et al. (2016). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*.
- He, Kaiming et al. (2017). "Mask R-CNN". In: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*.

- Hinton, Geoffrey E., Simon Osindero, and Yee Whye Teh (2006). "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation*.
- Howard, Andrew G. et al. (2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *arXiv*.
- Huang, Gao et al. (2017). "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). In: *Advances in Neural Information Processing Systems*.
- LeCun, Yann et al. (1998). "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE*.
- Lin, Min, Qiang Chen, and Shuicheng Yan (2013). *Network In Network*.

- Redmon, Joseph et al. (2016). "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*.
- Ren, Shaoqing et al. (2017). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Trans. Pattern Anal. Mach. Intell.*
- Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- Szegedy, Christian et al. (2015). "Going deeper with convolutions". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*.

- Tan, Mingxing and Quoc V. Le (2019). "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*.
- Uijlings, Jasper R. R. et al. (2013). "Selective Search for Object Recognition". In: *Int. J. Comput. Vis.*
- Xie, Saining et al. (2017). "Aggregated Residual Transformations for Deep Neural Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*.