Università degli
Studi di Milano-Bicocca

2D CNNs + Semantic Segmentation

Prof. Flavio Piccoli - Dr. Mirko Paolo Barbato

Image formation

Reflected light enters in the camera and hits the sensor

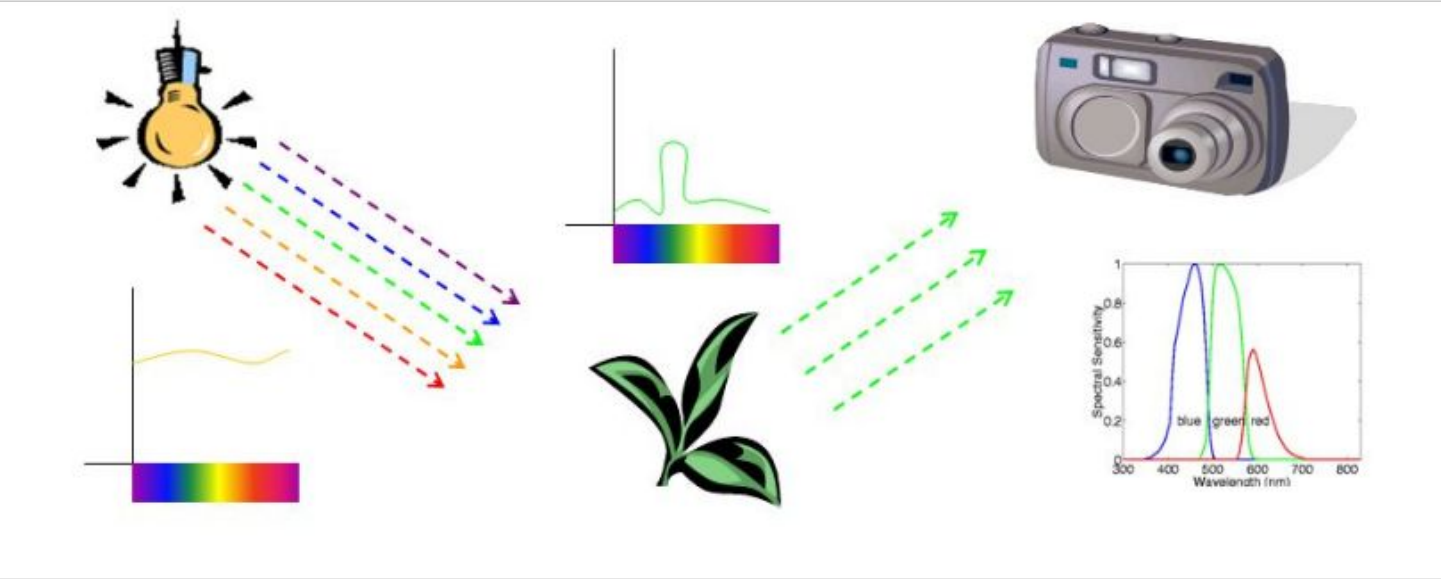
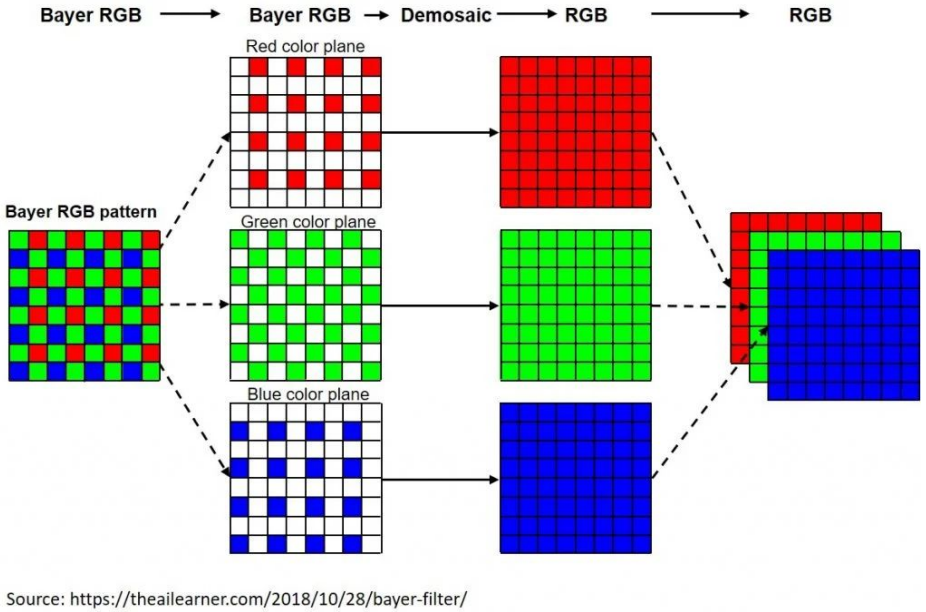


Image courtesy by Merav Mazouz &Matan Kolath

Image formation

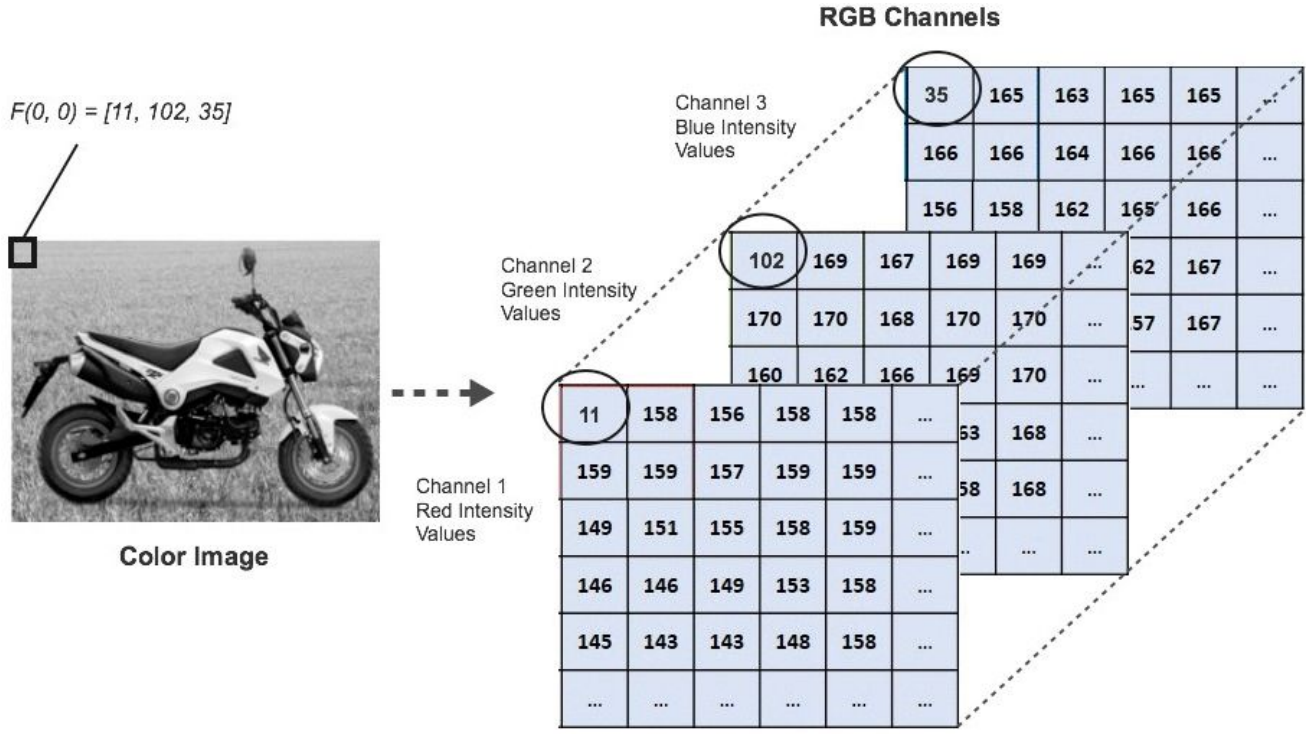
- Entering light is acquired by a set of sensors
- Each sensor acquires a single color following the **Bayer pattern**
- RGB planes are created



Source: https://theailearner.com/2018/10/28/bayer-filter/

Image formation

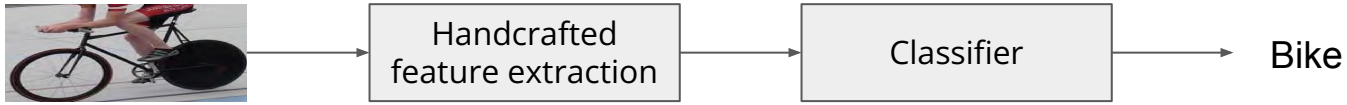
- Every color channel is a bidimensional matrix



Traditional computer vision VS deep learning

Traditional computer vision

- In the first place, hand-crafter features must be extracted from the input image
- Then, a classifier is user to perform classification
- It's really hard to find the most representative features for the domain under analysis



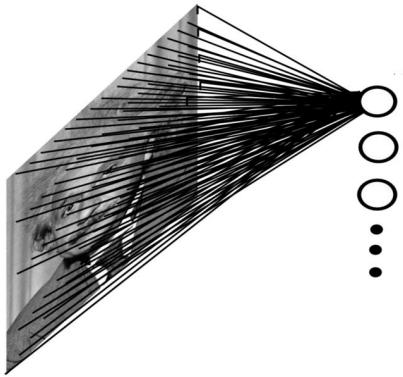
Deep Learning

- Neural Networks learn automatically the most suitable features for the task under analysis
- It is about learning multiple levels (a hierarchy) of representations and abstractions

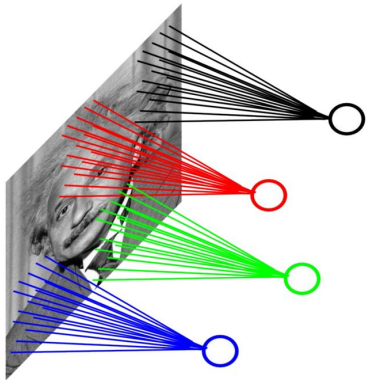
Convolutional Neural Networks for images

- Also in the case of images we can exploit local relationship among pixels
- CNNs are still NNs but with local connectivity

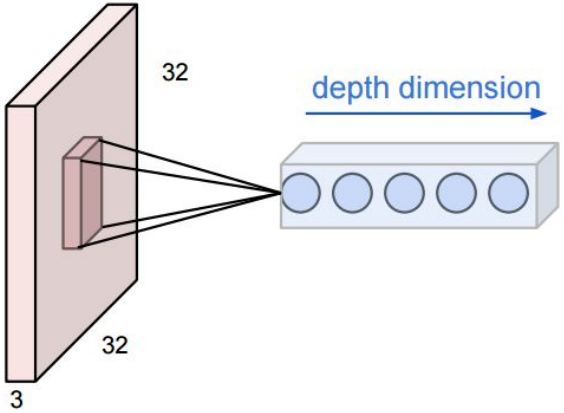
Neural Network

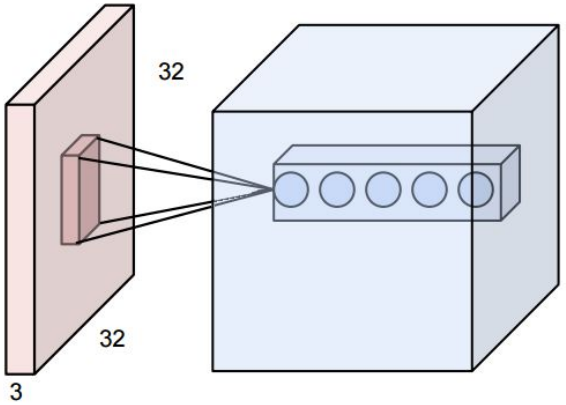


Convolutional Neural Network



Difference Between NNs And CNNs





With NNs:

- the output is a vector (1D + batch dim. = 2D)
- every output unit is given by 3072 (32*32*3+1) neurons

With CNNs:

- the output is a cube (3D + batch dim. = 4D)
- every output channel is given by kernel_size*kernel_size+1 neurons (e.g. 3*3+1=9)
- multiple neurons all looking at the same region of the input volume, stacked along depth

Image credit: L. Fei-Fei

Sliding window approach

- Convolutional filters slide across all the subportions of the image
  - parallel computation

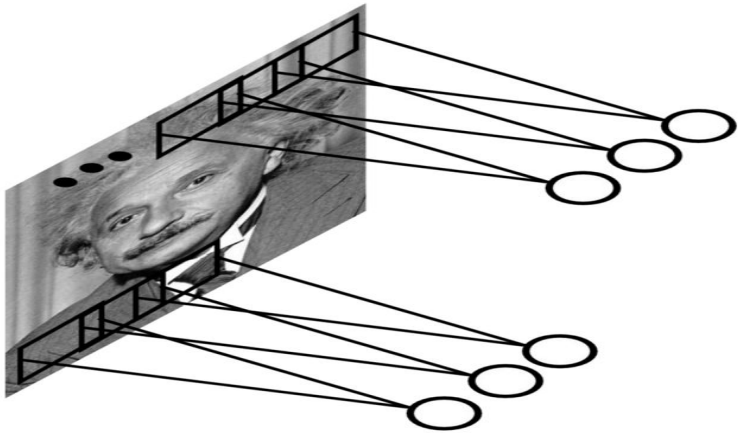
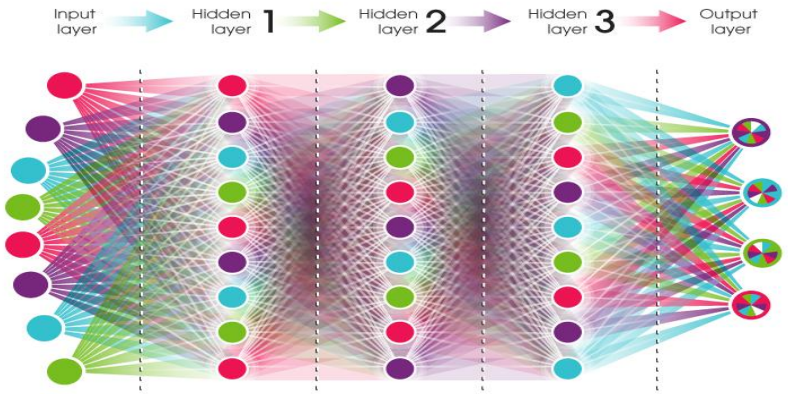
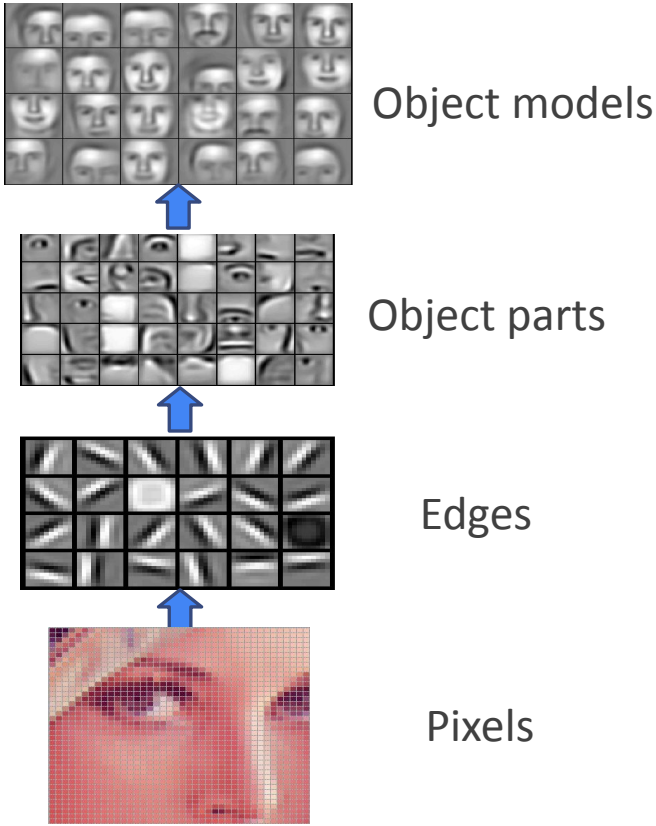


Image credit: L. Fei-Fei

Feature hierarchy

- hierarchy of representations and abstractions is given by using multiple levels (layers)
- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable non-linear feature transform
- Can share the lower-level representations for multiple tasks





Hierarchy of the features learned

Image Recognition

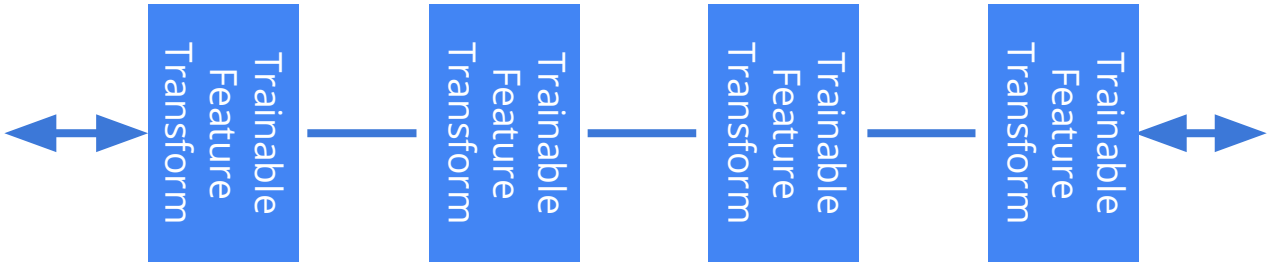
Pixel → Edge → Texton → Motif → Part → Object

Speech

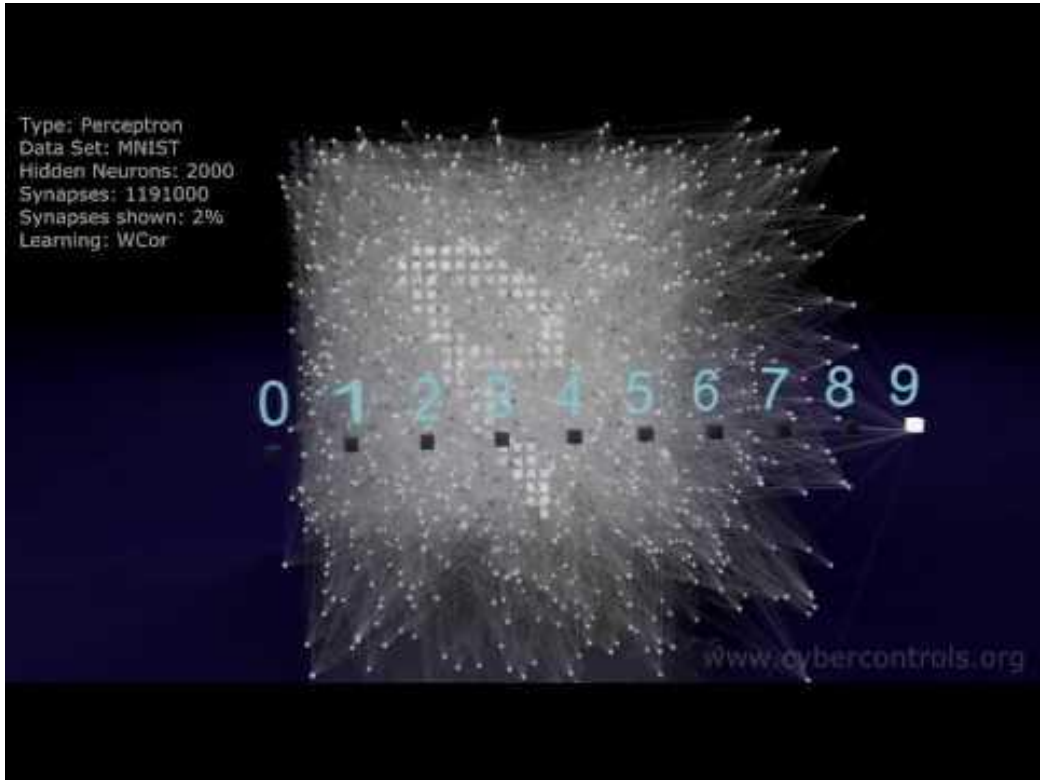
Sample → Spectral band → Sound → ... → Phonema → Word

Text

Character → Word → ... → Clause → Sentence → Story



Video


Type: Perceptron
Data Set: MNIST
Hidden Neurons: 2000
Synapses: 1191000
Synapses shown: 2%
Learning: WCor
0 1 2 3 4 5 6 7 8 9
www.cybercontrols.org

Demo online

http://scs.ryerson.ca/~aharley/vis/conv/

General architecture

- Supervised or not supervised
- Classification and regression
- Based on Gradient Descend
  - Backpropagation

Layer 1

Layer 2

Layer 3

. . .

Layer n

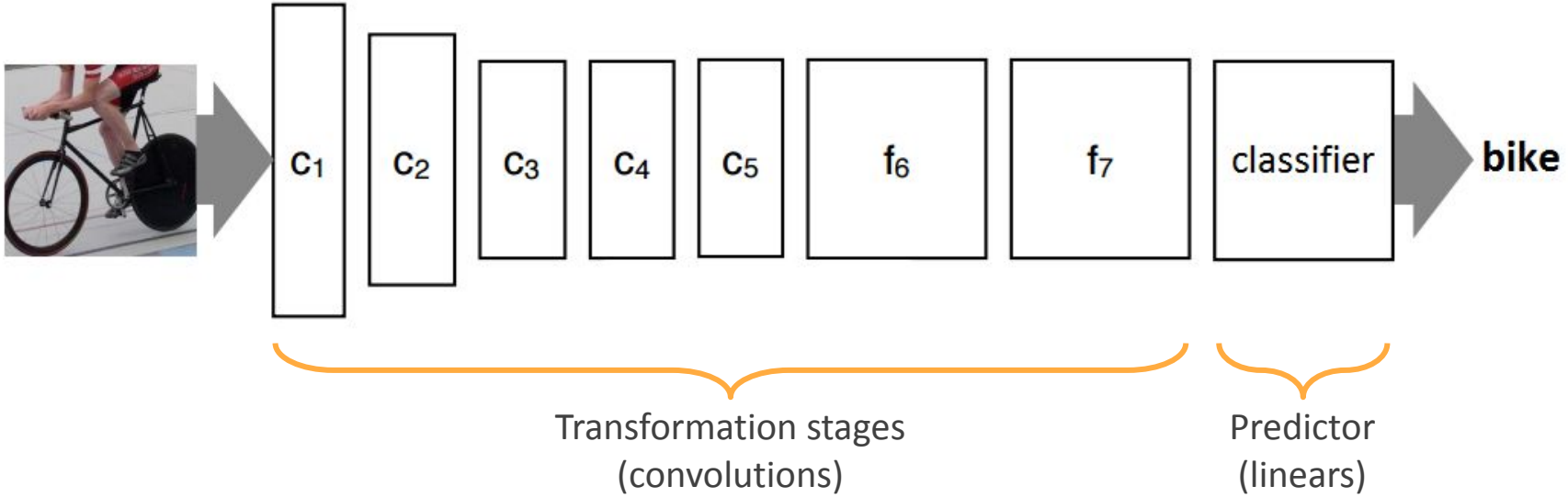
OUTPUT

Loss

Each layer is an operation. Can be:
- A pointwise operator
- A local operator

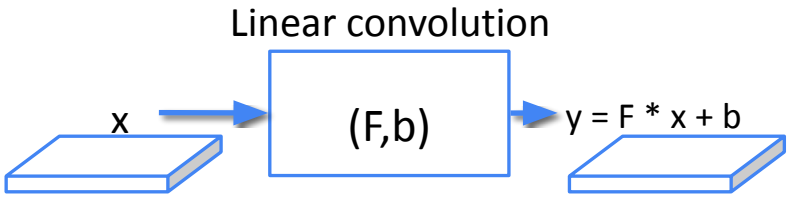
Used only in training time

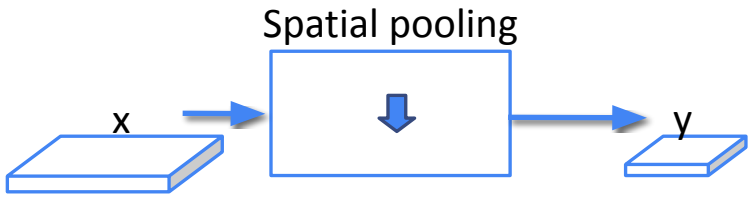
CNN architecture for classification

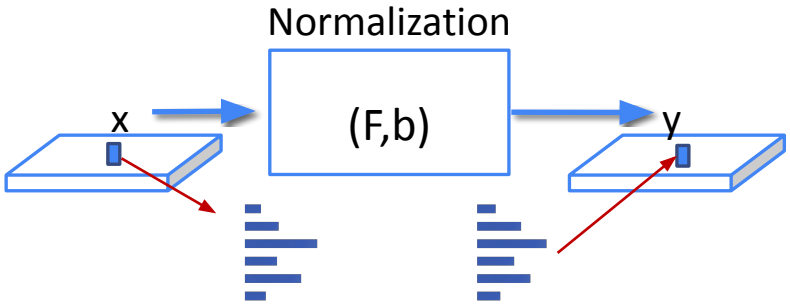


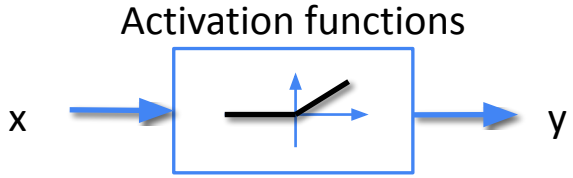
CNN Layers

Types of layers









What is convolution?

It is powerful local operator used in image processing. Can be used to remove noise:

Key Idea:

Spike noise can be attenuated by averaging values with its neighbours

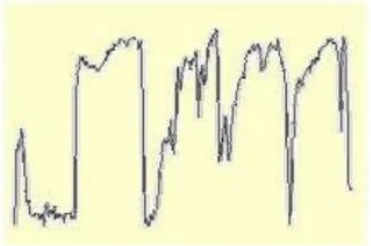


Image f

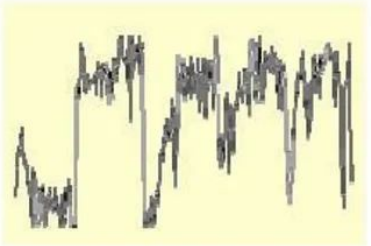
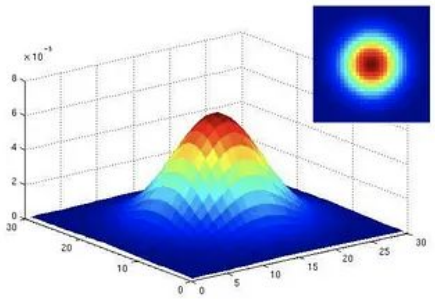


Image f
With Gaussian noise

Noise removal

Gaussian filtering helps to decrease noise at the prices of increased blurriness

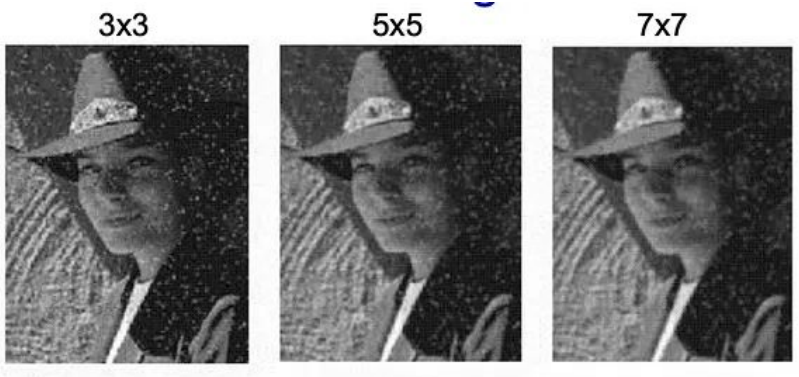
Gaussian filter



$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

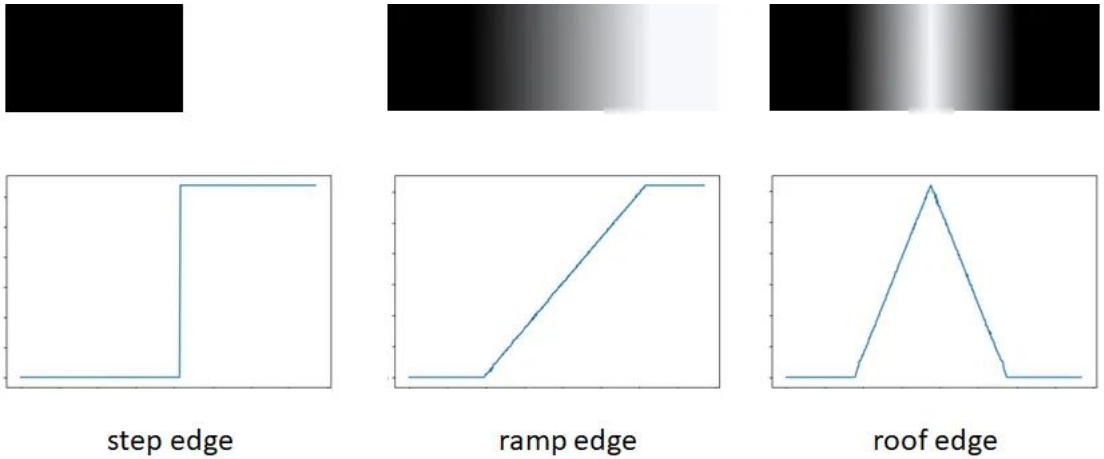
Gaussian



Edge detection

Convolution can also be used to highlight edges

Types of edges



Edge detection

Edges can be found by computing the derivative of the image

$$f'(x_i) = \lim_{\epsilon \to 0} \frac{f(x_i + \epsilon) - f(x_i)}{\epsilon}$$

		$\epsilon = 1$ (x →)	$\epsilon = 1$ (y ↓)
(a) Forward	Finite diff.	$\frac{\partial I[x_m, y_n]}{\partial x} \approx \frac{I[x_m + \epsilon, y_n] - I[x_m, y_n]}{\epsilon}$	$\frac{\partial I[x_m, y_n]}{\partial y} \approx \frac{I[x_m, y_n + \epsilon] - I[x_m, y_n]}{\epsilon}$
	Kernel	0 -1 1	0 / -1 / 1
(b) Backward	Finite diff.	$\frac{\partial I[x_m, y_n]}{\partial x} \approx \frac{I[x_m, y_n] - I[x_m - \epsilon, y_n]}{\epsilon}$	$\frac{\partial I[x_m, y_n]}{\partial y} \approx \frac{I[x_m, y_n] - I[x_m, y_n - \epsilon]}{\epsilon}$
	Kernel	-1 1 0	-1 / 1 / 0
(c) Central	Finite diff.	$\frac{\partial I[x_m, y_n]}{\partial x} \approx \frac{I[x_m + \epsilon, y_n] - I[x_m - \epsilon, y_n]}{\epsilon}$	$\frac{\partial I[x_m, y_n]}{\partial y} \approx \frac{I[x_m, y_n + \epsilon] - I[x_m, y_n - \epsilon]}{\epsilon}$
	Kernel	-1 0 1	-1 / 0 / 1

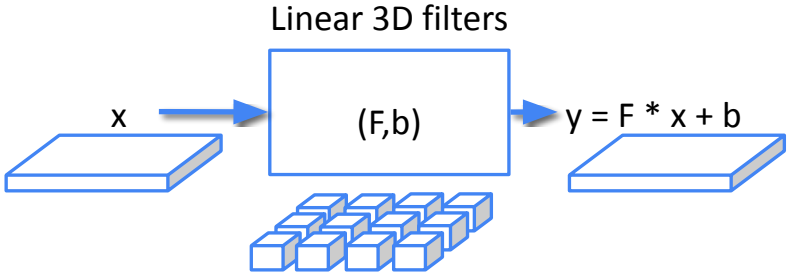
Edge detection with Laplacian Filter

It is an approximation of second order derivative that defines zeros crossing. For Example 3x3 laplacian is :

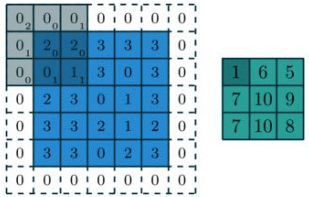
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

With CNNs we will find the filters automatically

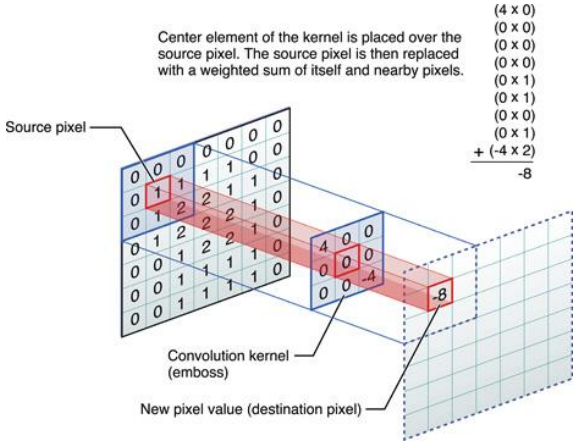
Linear convolution



- Linear
- Local
- Translation invariant
- Filter bank to form a richer representation of the data

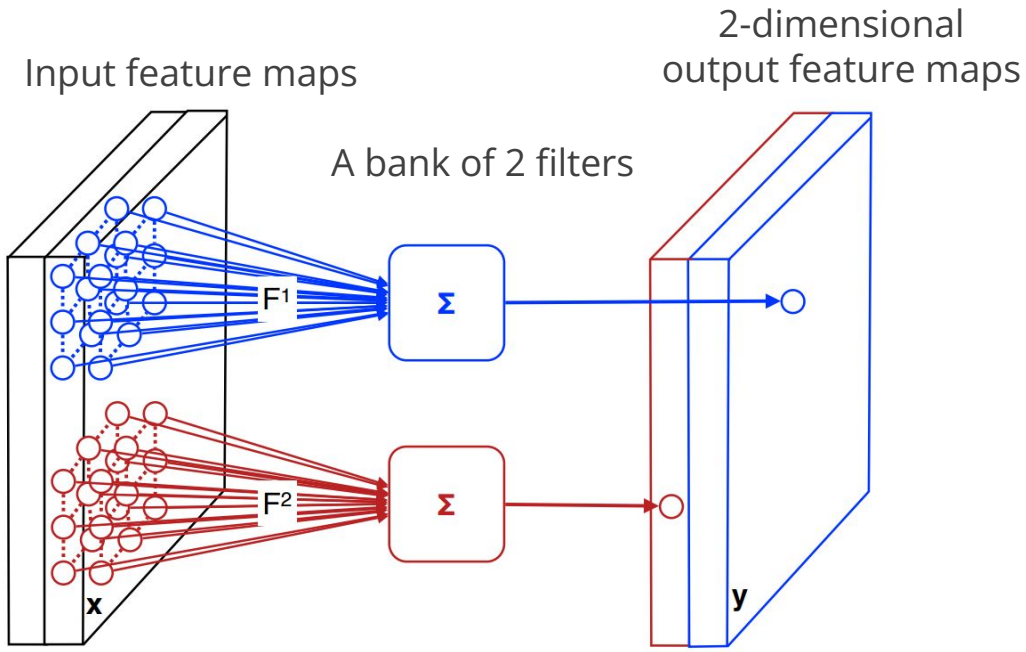


1	6	5
7	10	9
7	10	8



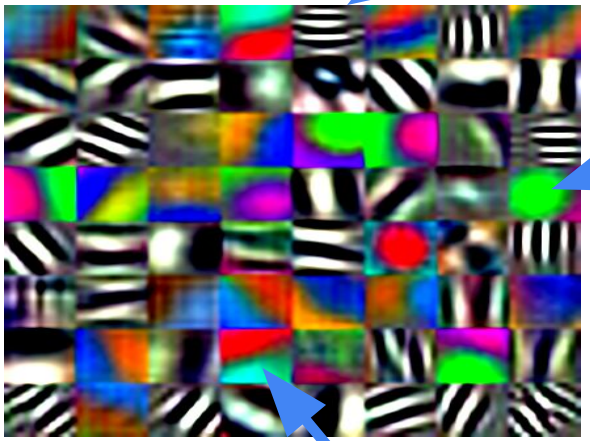
$$y_{ijq} = y_q + \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \sum_{k=1}^{K} x_{u+i,v+j,k} \, F_{u,v,k,q}$$

Linear convolution



Learned filters of the first conv layer

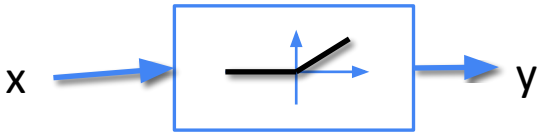
Filters learned during VGG net training



Did the network overfit?

Hard to interpret filters of layers different from the first one, because they do not work directly on image but on features maps

Activation functions



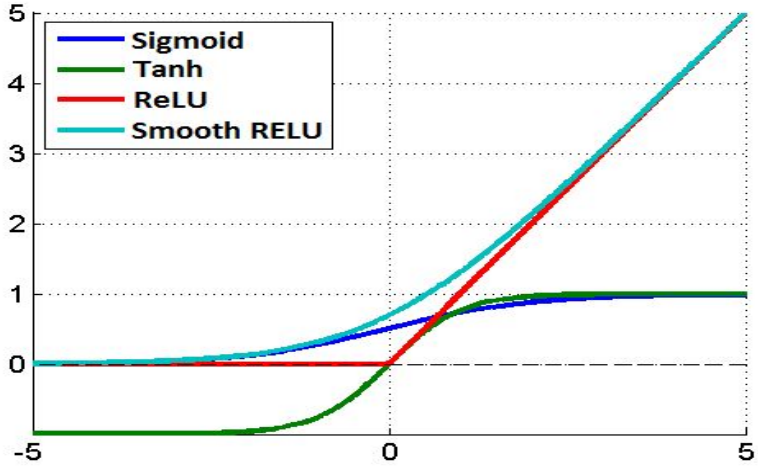
- Scalar non-linearity

$$y = \frac{1}{1 + e^{-x}}$$ Sigmoid

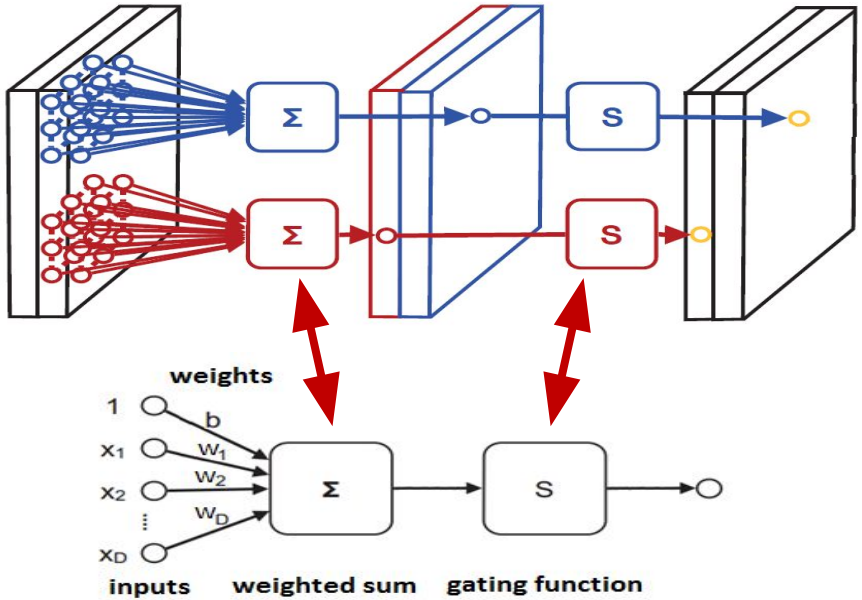
$$y = \tanh x$$ Hyperbolic tangent

$$y = \max\{0, x\}$$ Rectified Linear Unit (ReLU)

$$y = \log\ (1 + e^{x})$$ Smooth ReLU



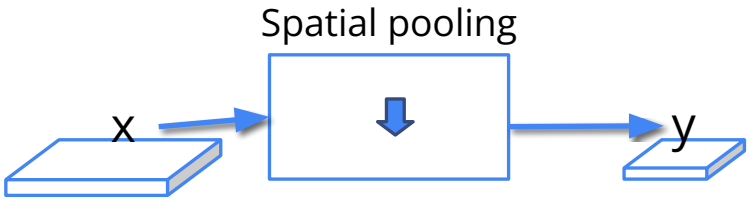
Activation functions



Filters are followed by non-linear operators (e.g. gating function)

Image credit: A. Vedaldi

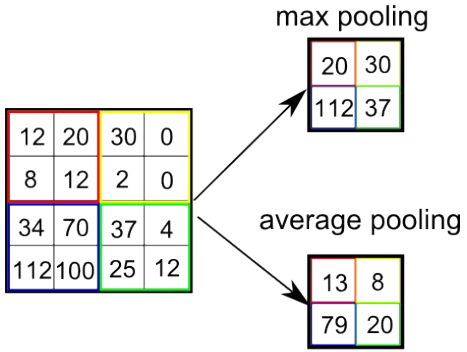
Spatial pooling



- Pooling and sub-sampling
- Encodes translation invariance
- Pooling computes the average/max of the features in a neighbourhood
- It is applied channel-by-channel

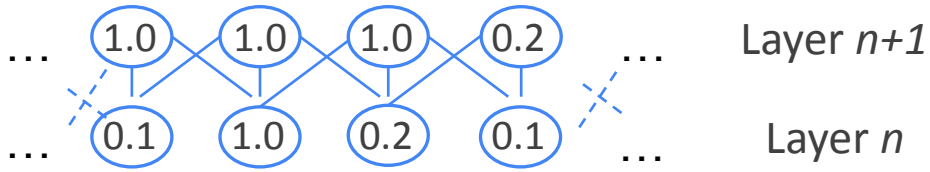
$$y_{ijk} = \max_{pq \in \Omega_{ij}} x_{pqk}$$ Max pooling

$$y_{ijk} = \operatorname{avg}_{pq \in \Omega_{ij}} x_{pqk}$$ Average pooling

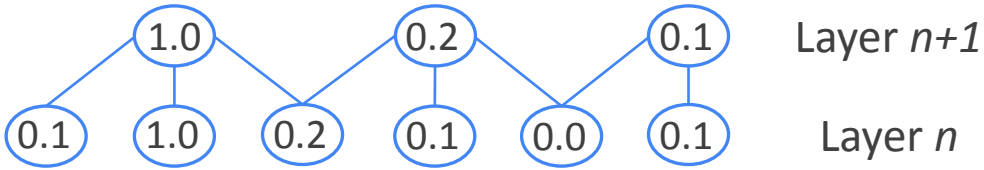


Spatial pooling

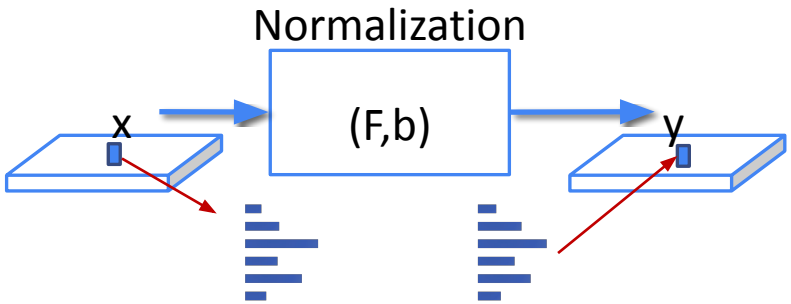
- Aggregate to **achieve translation invariance** (gain robustness to the exact spatial location of features)



- Subsampling to **reduce spatial scale** and computation



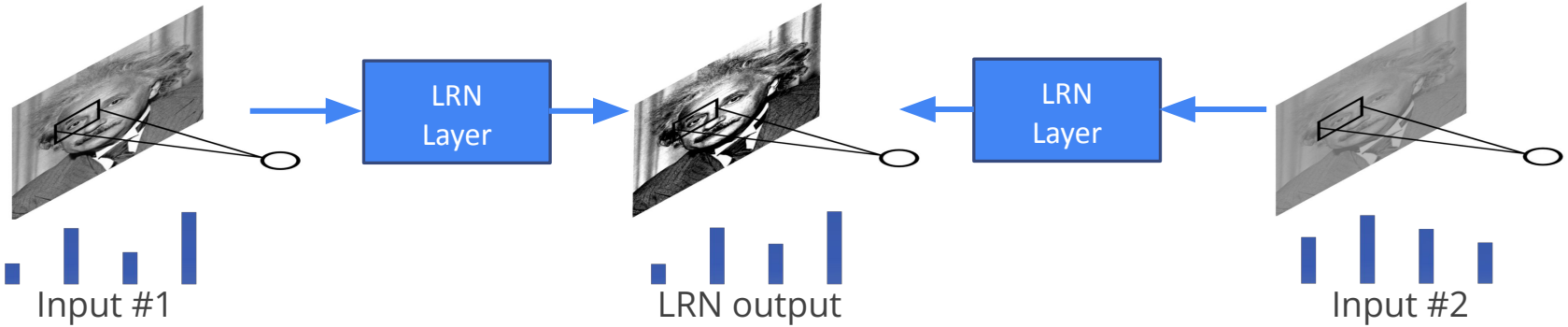
Local response normalization (LRN)



Contrast normalization

Effects:

- Improves invariance
- Improves optimization
- Improves sparsity

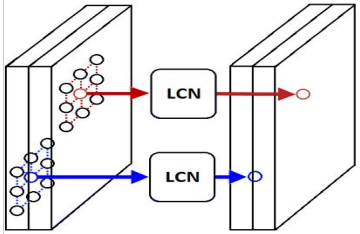


We want the same response

Local response normalization (LRN)

WITHIN CHANNEL

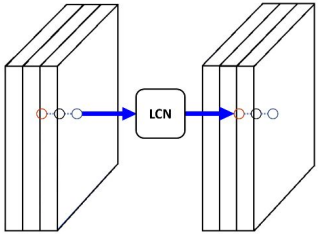
- Operates independently on different feature channels
- Rescales each input feature basing on a local neighborhood



$$y_{ijk} = x_{ijk}\left(\kappa + \alpha \sum_{(u,v)\in\mathcal{N}(i,j)} x_{uvk}^2\right)^{-\beta}$$

ACROSS CHANNELS

- Operates independently at each spatial location and groups of channels
- Normalizes groups G(k) of feature channels
- Groups are usually defined in a sliding window manner



$$y_{ijk} = x_{ijk}\left(\kappa + \alpha \sum_{q\in G(k)} x_{ijq}^2\right)^{-\beta}$$

Batch normalization

A layer with weights to be learned, is easier (or sometime it doen't work elsewise) if input data has:

- mean = 0
- stddev = 1

Running mean

$$bnorm(x) = \frac{x - mean(x)}{stddev(x)}$$

Running standard deviation

Batch normalization layer is used many times inside a network

Batch normalization is the only layer that introduces a dependency among the samples in the same batch.

Softmax

Example of a classification network

CNN

feature vector with unconstrained numbers

SOFTMAX

Probability that image depicts a cat

Probability that image depicts another class

probability vector (sums to 1)

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

Data normalization and regularization

Data normalization

- **Local mean subtraction:** subtract the mean from the original data

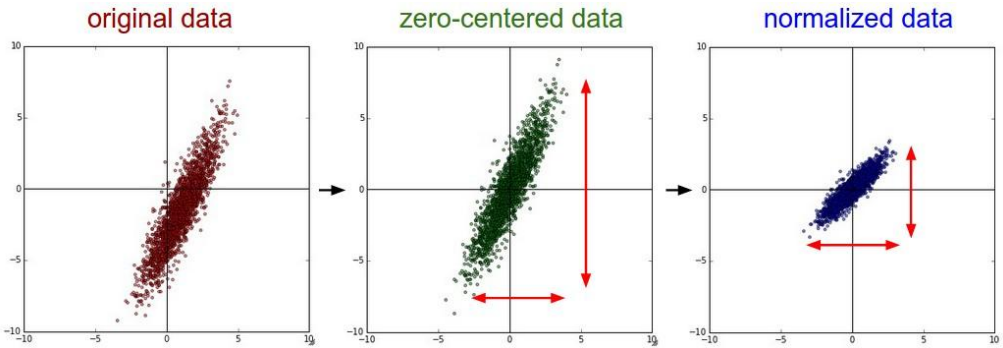
$$x = x - \mu$$

- **Normalization:** scale original data to a specific range
  - **min-max** normalization if the range is known a-priori

$$x = \frac{x - min(X)}{max(X) - min(X)}$$

  - **standardization** if it is not known or the variable is not constrained

$$x = \frac{x - \mu}{\sigma}$$

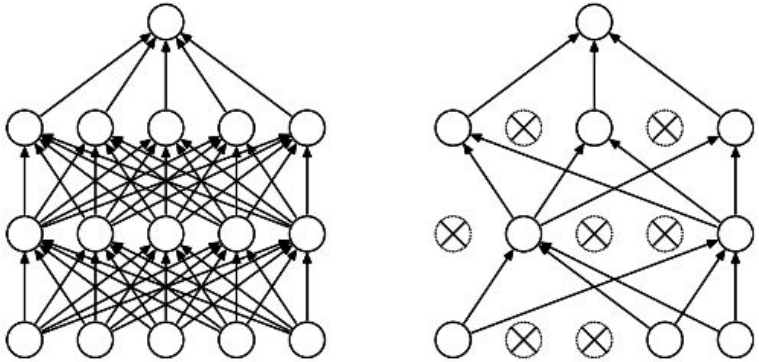


Images credit: L. Fei-Fei

Dropout (regularization)

- Used only in training time to strengthen the learning
- Randomly switches off connections

x_{ijk} → Dropout → $y_{ijk} = \frac{1}{1-p} \begin{cases} x_{ijk} & \text{with prob. } p \\ 0 & \text{otherwise} \end{cases}$



(a) Standard Neural Net (b) After applying dropout.

torch.nn.Dropout(p=**0.5**, inplace=False)

Data augmentation (regularization)

- Augment the training set by “jittering” samples
- Label preserving image transformations
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion
  - …

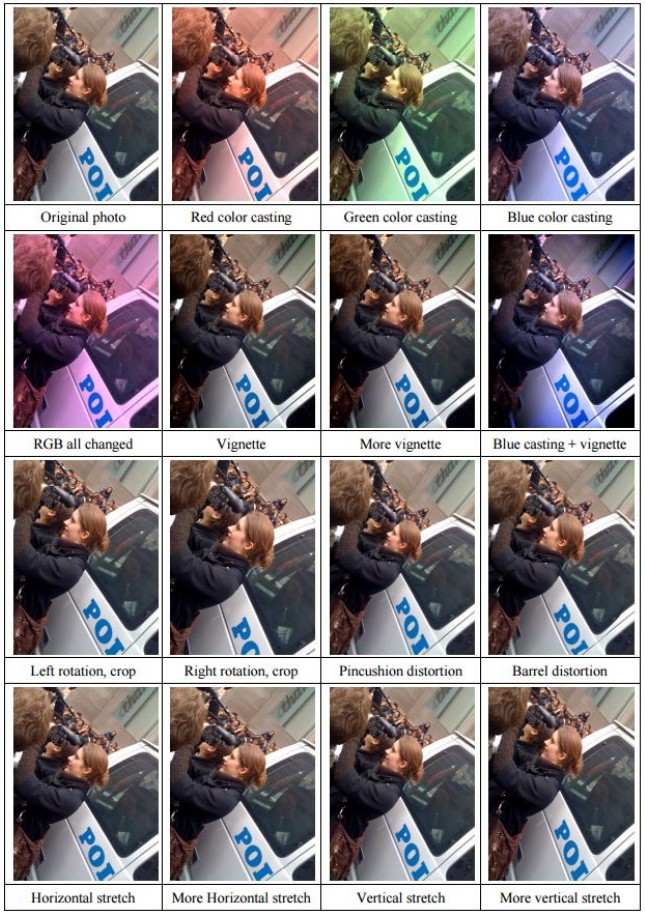


Image credits: R. Wu, S. Yan, Y. Shan, Q. Dang and G. Sun, Deep image: Scaling up image recognition, *arXiv preprint arXiv:1501.02876, 2015*.

Albumentation

- It is a python library for data augmentation
- Easy integration with Pytorch and Keras
- Supporting all kind of ground-truths
  - semantic layouts
  - masks
  - bounding-boxes
  - ...

Example

```
# define transform
transform = A.Compose([
    A.RandomRotate90(),
    A.Transpose(),
    A.ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.50, rotate_limit=45, p=.75),
    A.Blur(blur_limit=3),
    A.OpticalDistortion(),
    A.GridDistortion(),
    A.HueSaturationValue(),
])

# apply transform
augmented_image = transform(image=image)['image']
```

Simultaneous augmentation of multiple targets (e.g. masks, bbs, pts)

- It is possible to specify additional targets and their types with:

```
# define transform
transform = A.Compose(
    [
        A.HorizontalFlip(p=0.5),
        A.ShiftScaleRotate(p=0.5),
        A.RandomBrightnessContrast(p=0.2),
        A.RGBShift(p=0.2),
    ],
    additional_targets={'image0': 'image', 'image1': 'image'}
)

# apply transform
transformed = transform(image=image, image0=image0, image1=image1)

# get augmented sample
image0_new = transformed['image0']
image1_new = transformed['image1']
```

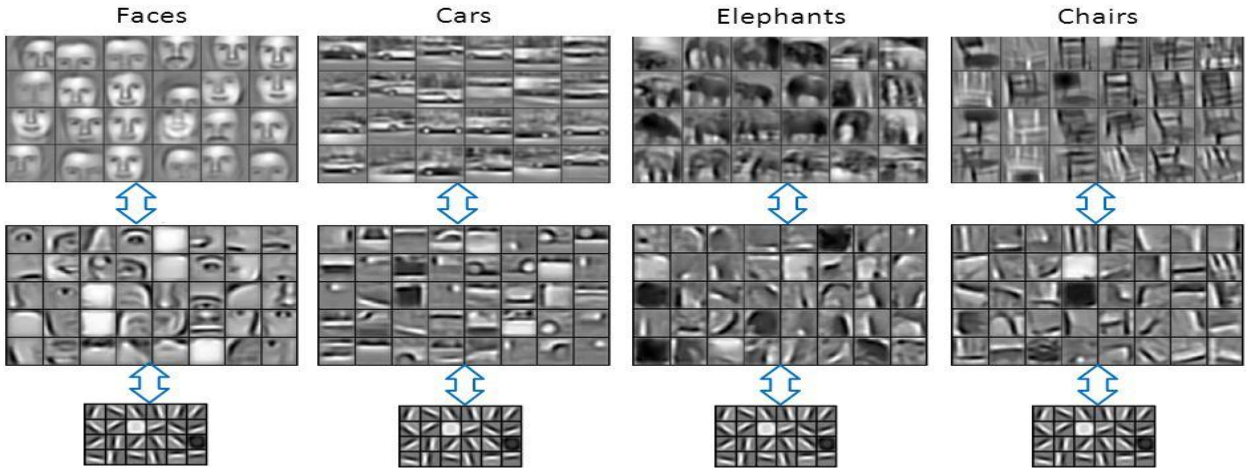
- Supported types: images, bounding boxes, masks, points, etc..

Transfer learning

Transfer Learning

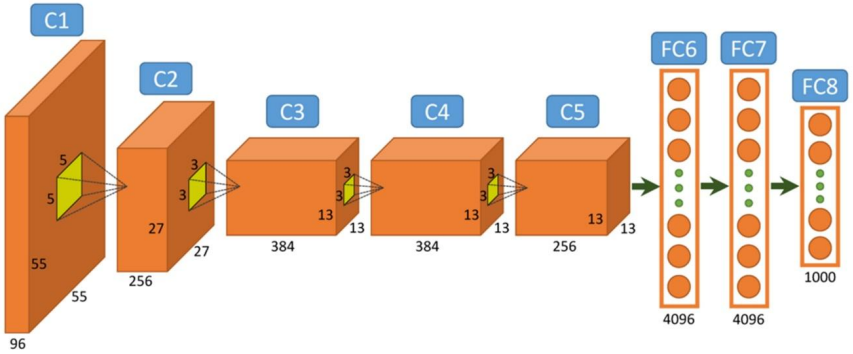
Why does it work?

- Motivated by the observation that the earlier features of a CNN contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks
- Later layers of the CNN becomes progressively more specific to the details of the classes contained in the original dataset.

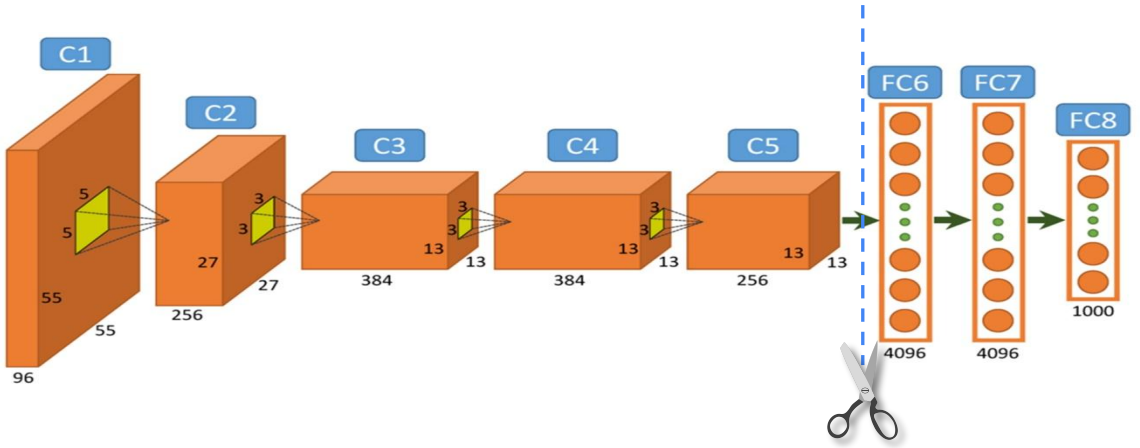


Transfer Learning

Task 1 (e.g. ImageNet, 1000 categories)

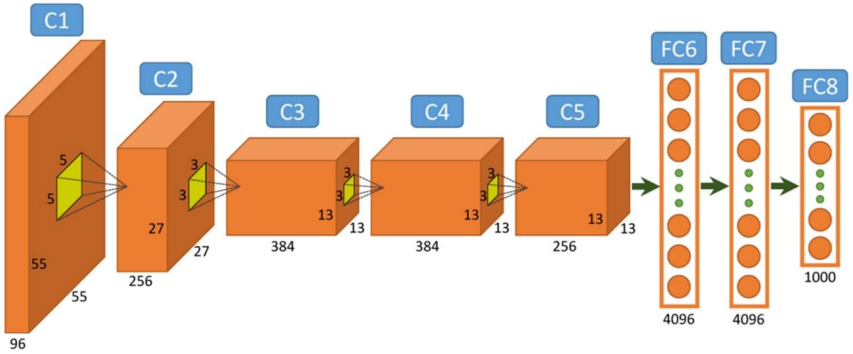


Task 2 (10 categories)

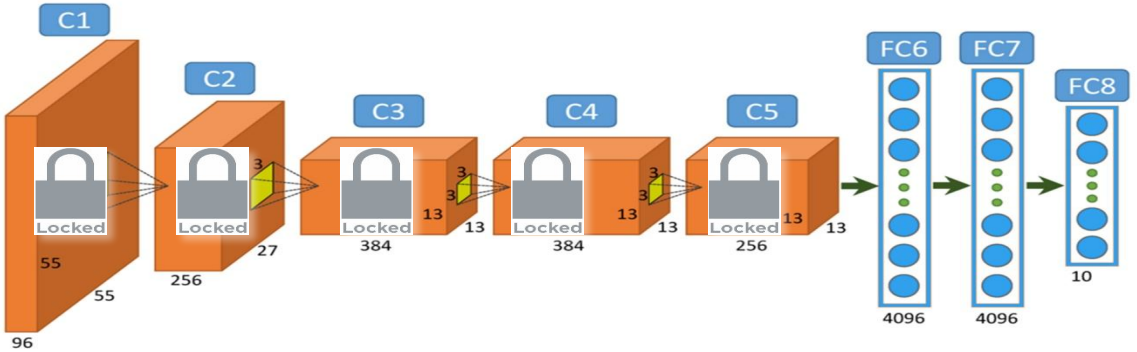


Transfer Learning

Task 1 (e.g. ImageNet, 1000 categories)



Task 2 (10 categories)



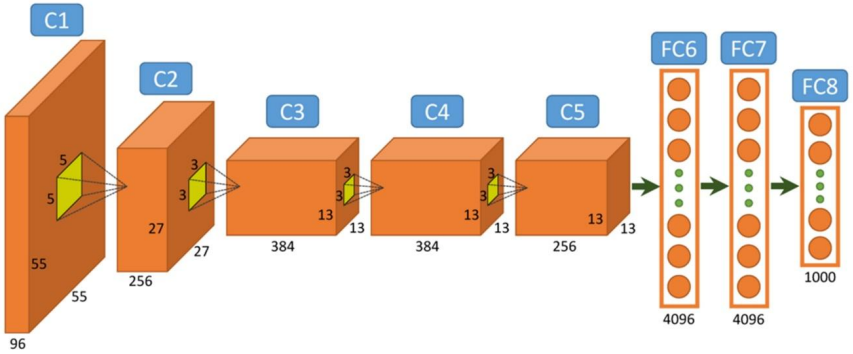
Transfer Learning with PyTorch

In PyTorch this can be done very easily

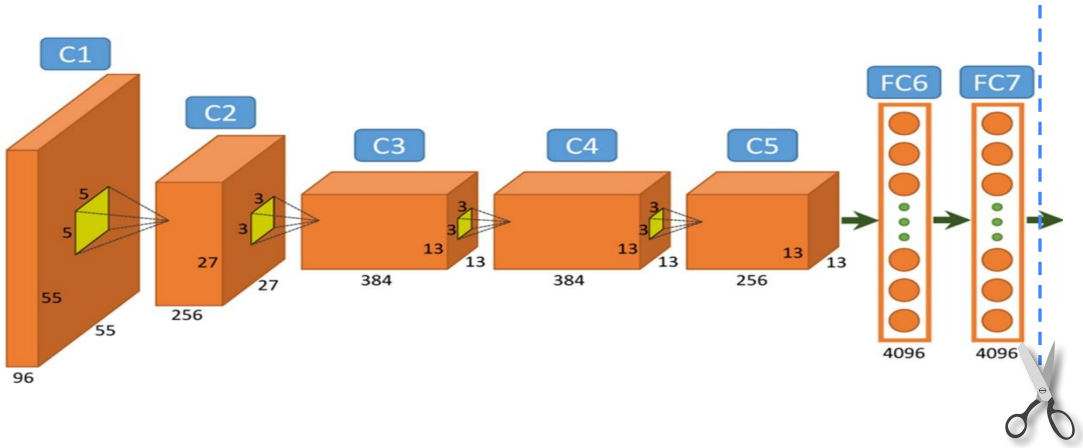
```
import torch
import torch.nn as nn
from torchvision.models import resnet18, ResNet18_Weights


# define new number of classes
n_classes = 10

# get pretrained resnet18
model = resnet18(weights=ResNet18_Weights.DEFAULT)

# replace last layer
model.fc = nn.Linear(in_features=512, out_features=n_classes)
```

CNN as feature extractor

Task 1 (e.g. ImageNet, 1000 categories)



Task 2 (10 categories)



CNN as feature extractor in PyTorch - strategy 1

Replace the following layers with nn.Identity() which does not perform any operation to its input (output = input)

```
import torch
import torch.nn as nn
from torchvision.models import resnet18, ResNet18_Weights


# define new number of classes
n_classes = 10

# get pretrained resnet18
model = resnet18(weights=ResNet18_Weights.DEFAULT)

# remove last layer
model.fc = nn.Identity()
```

CNN as feature extractor in PyTorch - strategy 2

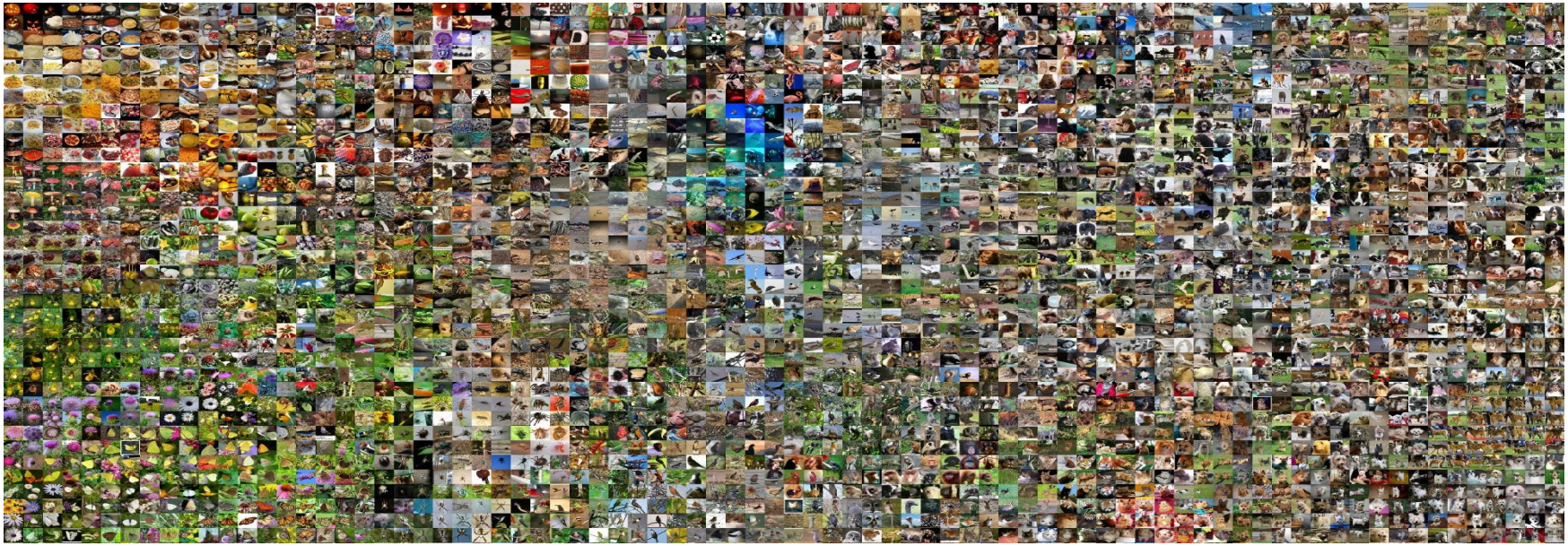
Use forward hook to intercept the activation

```
import torch
import torch.nn as nn
from torchvision.models import resnet18, ResNet18_Weights


# create variable that will contain the activation
cur_activation = None

# define hook function
def hook(model, input, output):
        global cur_activation
        cur_activation = output.detach()

# get pretrained resnet18
model = resnet18(weights=ResNet18_Weights.DEFAULT)

# register forward hook
model.avgpool.register_forward_hook(hook)


# perform forward pass
model(torch.rand(1,3,255,255))

# use hooked activation
print(cur_activation.shape)
```

Famous networks

IMAGENET

- 14,197,122 images
- 21,841 synsets



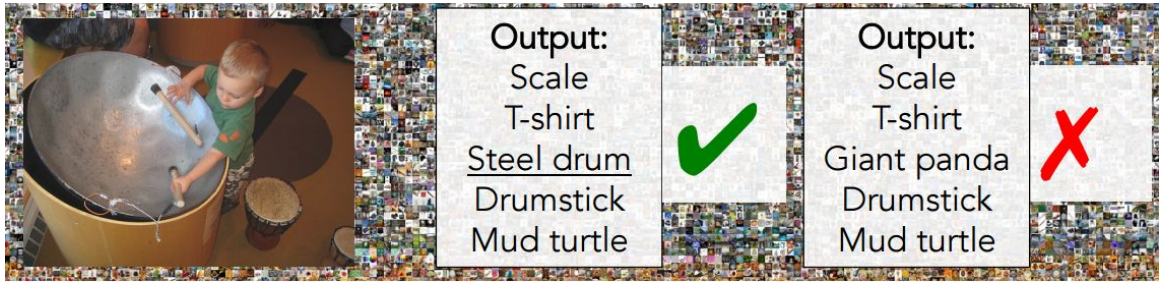
J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, **ImageNet: A Large-Scale Hierarchical Image Database**, *IEEE Computer Vision and Pattern Recognition (CVPR), 2009*.

Imagenet - Large Scale Visual Recognition Challenge (ILSVRC)

The Image Classification Challenge:

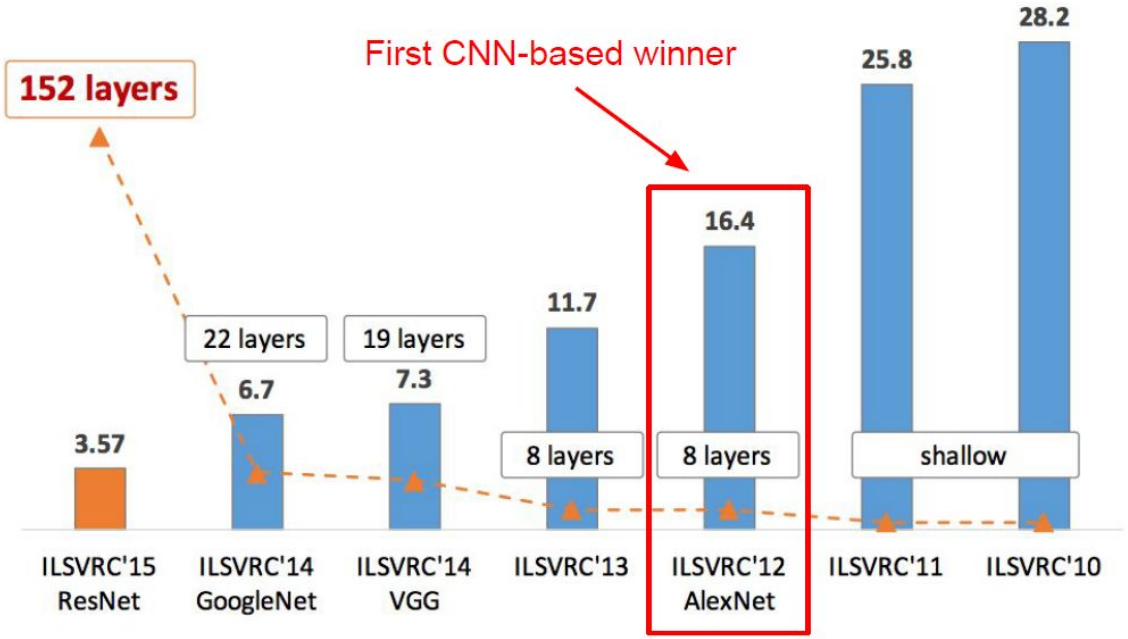
1,000 object classes

1,431,167 images



Imagenet Large Scale Visual Recognition Challenge (ILSVRC)

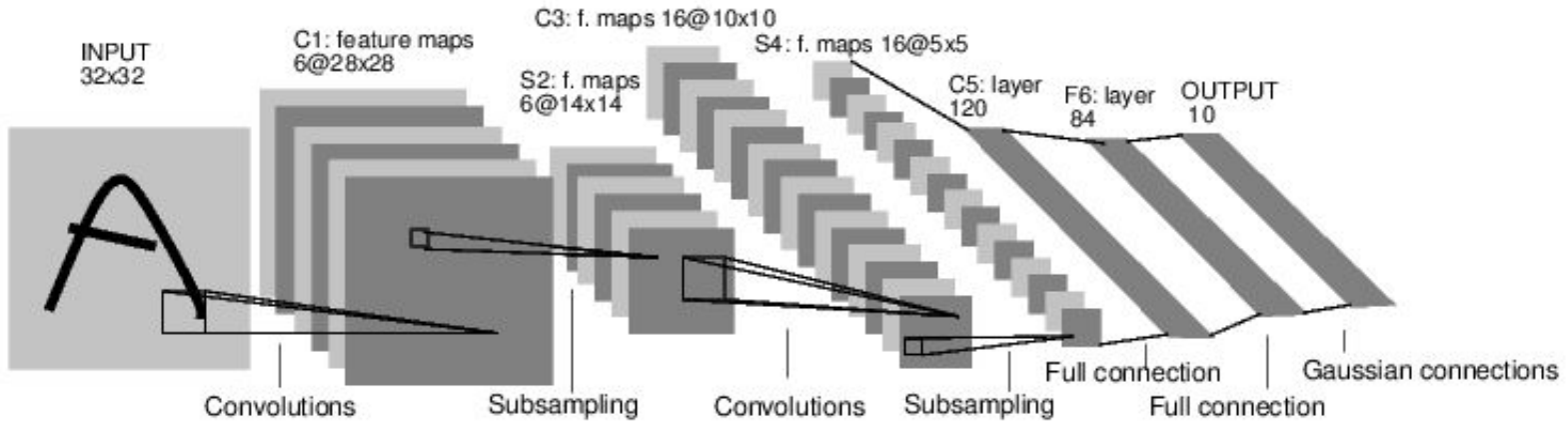
ImageNet classification error



Imagenet Large Scale Vi hallenge (ILSVRC)

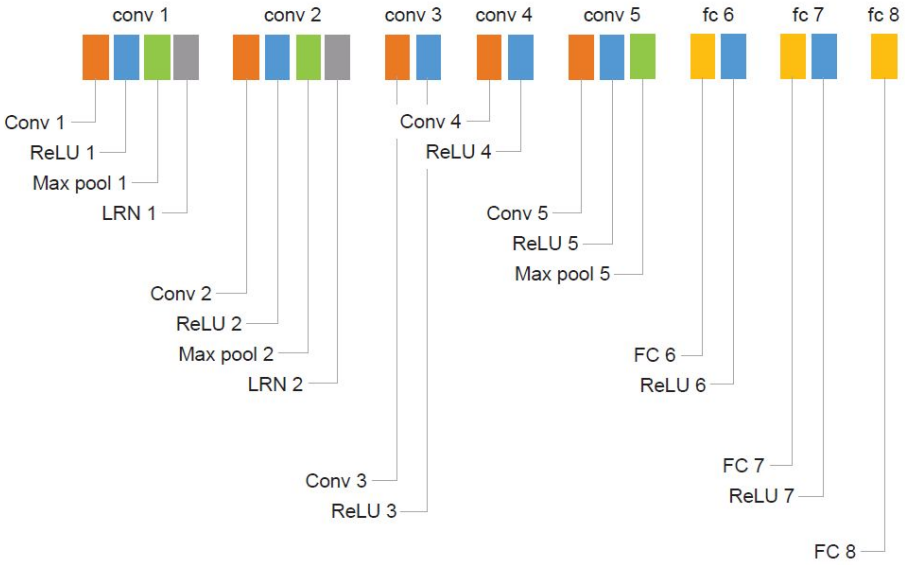
ImageNet classifi

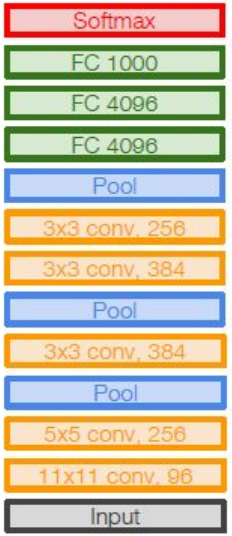
LeNet (LeNet-5)



LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

AlexNet

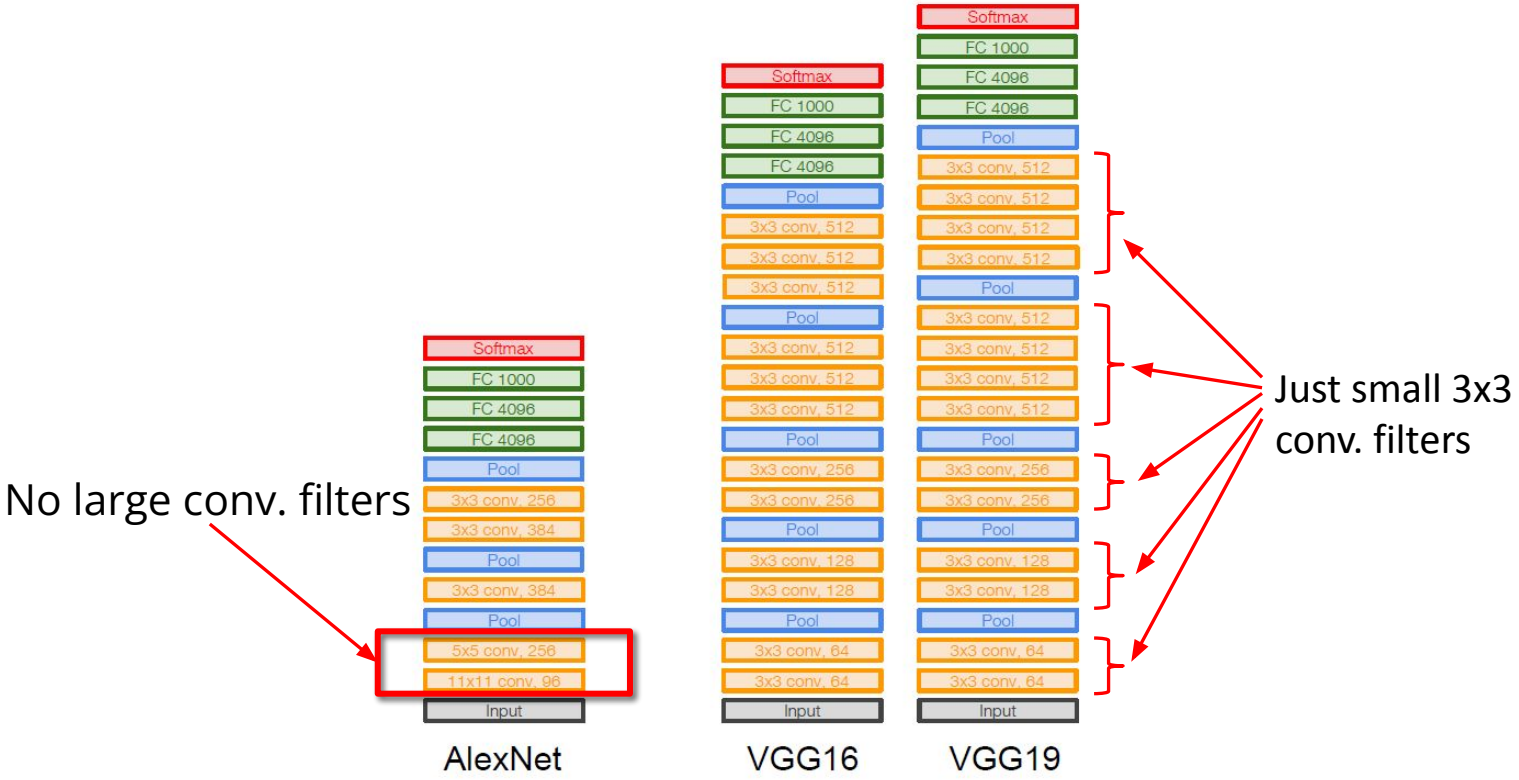




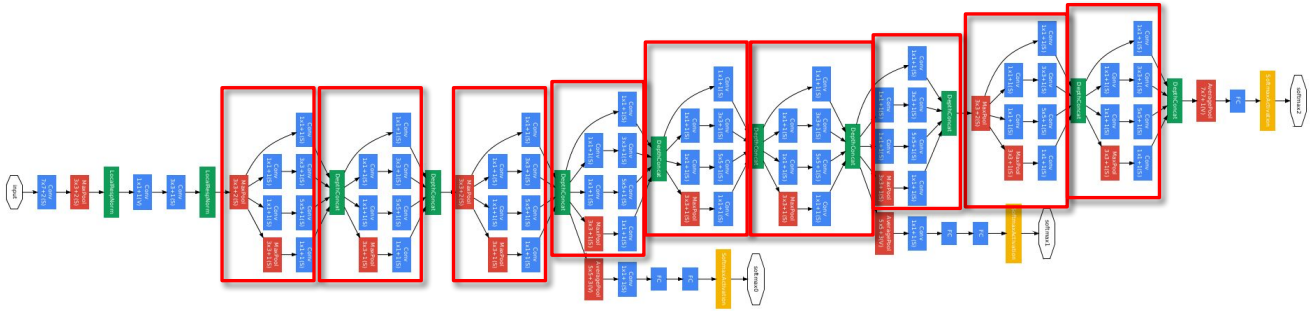
AlexNet

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

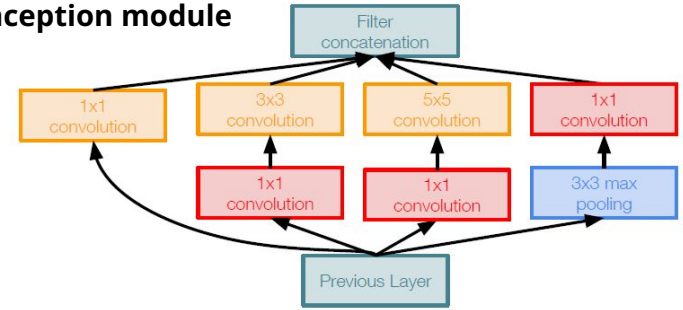
VGG


No large conv. filters
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 256
3x3 conv, 384
Pool
3x3 conv, 384
Pool
5x5 conv, 256
11x11 conv, 96
Input
AlexNet
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
VGG16
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
VGG19
Just small 3x3 conv. filters

GoogLeNet (Inception)



Inception module



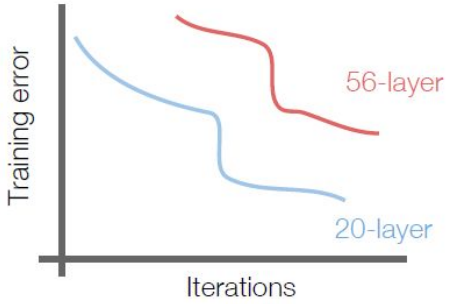
- "Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other
- Use 1x1 conv layer as bottleneck to reduce feature depth

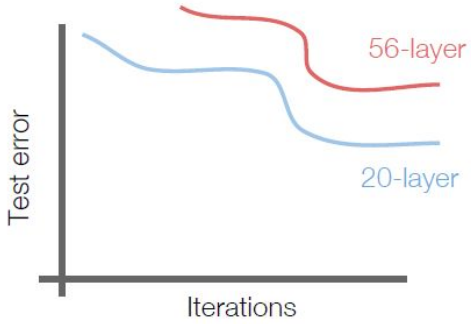
Famous Architecture

So far the deeper the better

ResNet

- So far the deeper the better
- Can we continue on adding layer and go deeper?

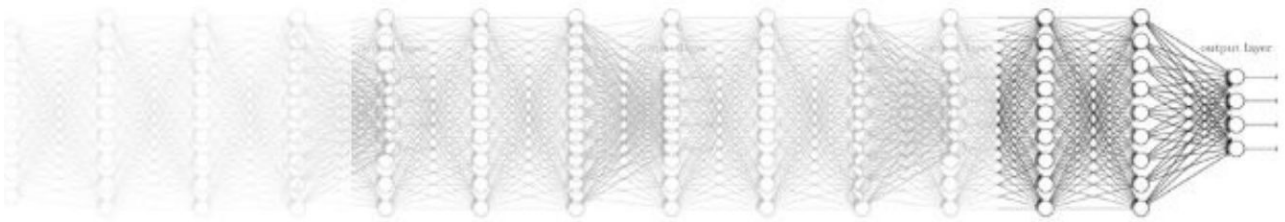




He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

ResNet

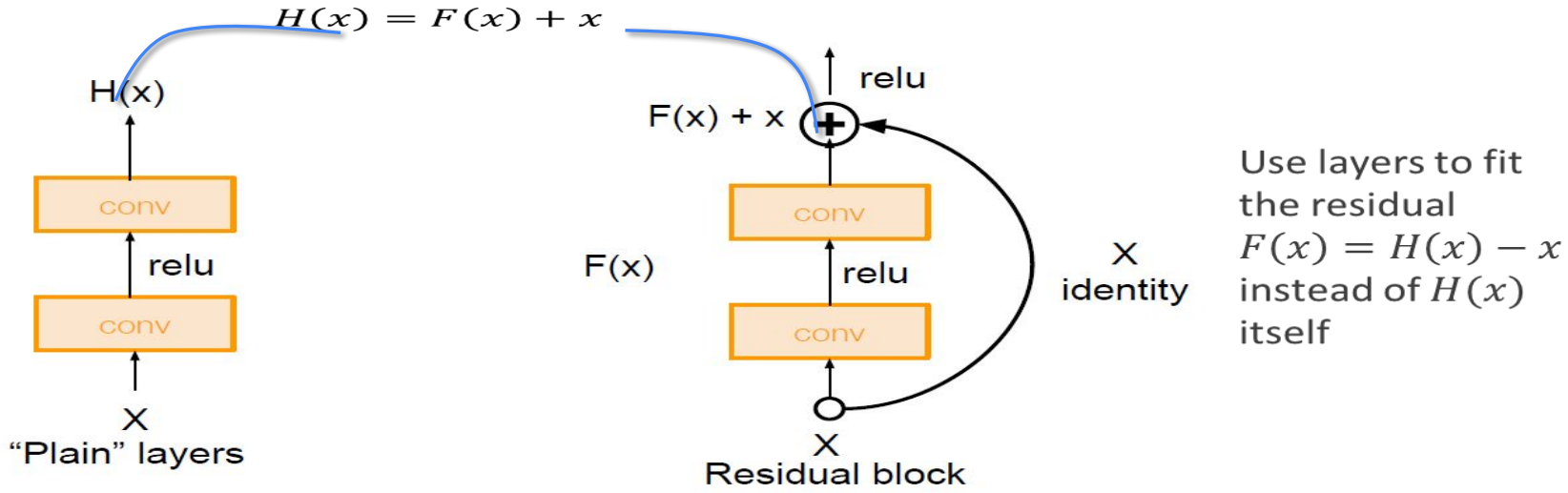
- So far the deeper the better
- Can we continue on adding layer and go deeper?
- Unfortunately no
- Hypothesis: the problem is an optimization problem, deeper models are harder to optimize (e.g. **vanishing gradients**)



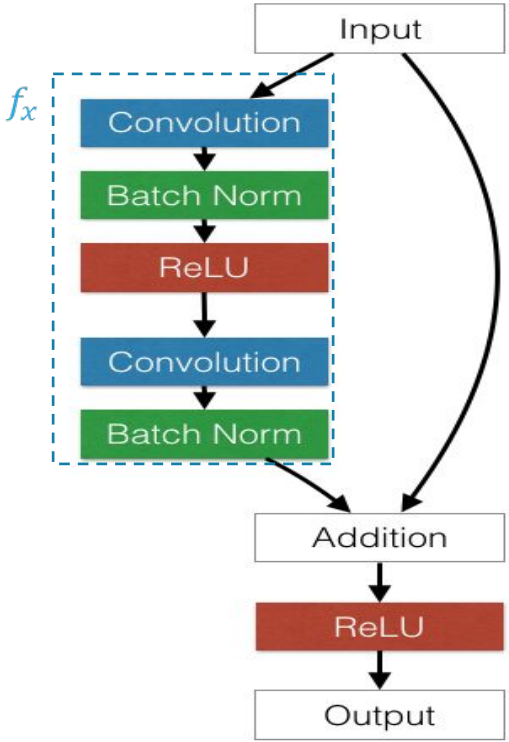
0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5 = 0.0000038

ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



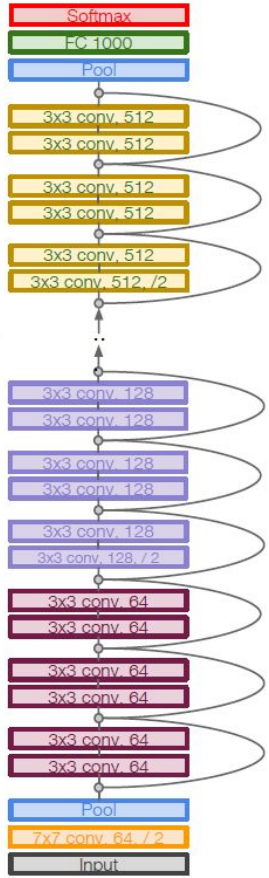
ResNet



Identity function

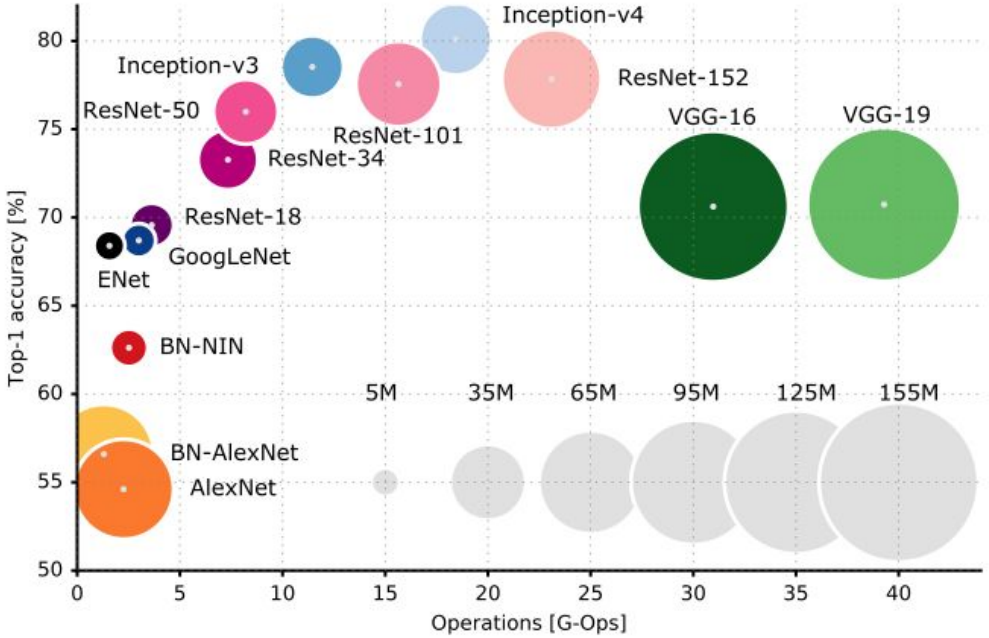
$$output = f(x) + x$$

Shortcut helps solving the vanishing gradients



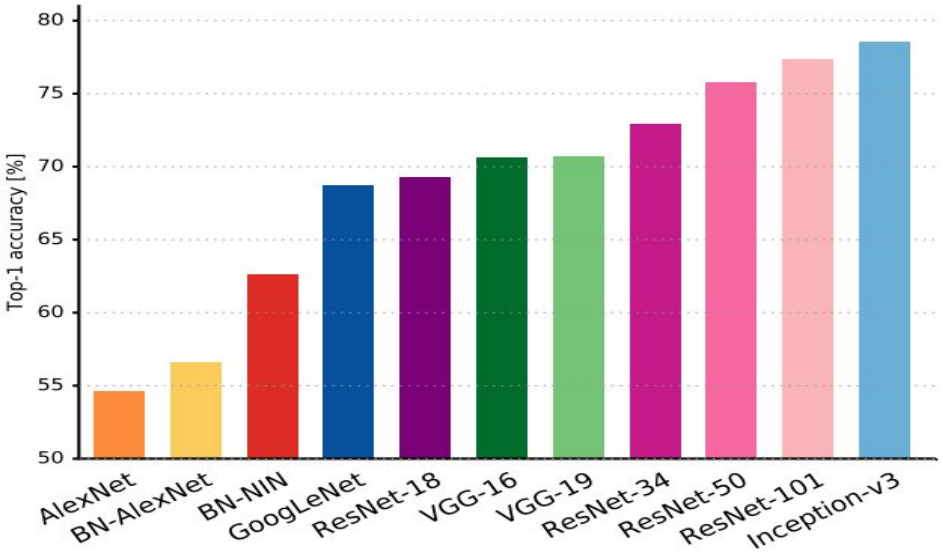
Famous Networks

- **IMAGENET CLASSIFICATION ERROR**



Famous Networks

●FAMOUS ARCHITECTURES



Method	Top-5 error (%)
Human-expert [1]	**5.1**
Inception-v3 [2]	3.5

[1] O. Russakovsky, et al., **Imagenet large scale visual recognition challenge**, *International Journal of Computer Vision* 115.3 (2015): 211-252.
[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, **Rethinking the inception architecture for computer vision**, *arXiv preprint arXiv:1512.00567,* (2015).

Useful Links

https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Solvers
https://cs.stanford.edu/people/karpathy/convnetjs/demo/trainers.html

Applications

Image to Image

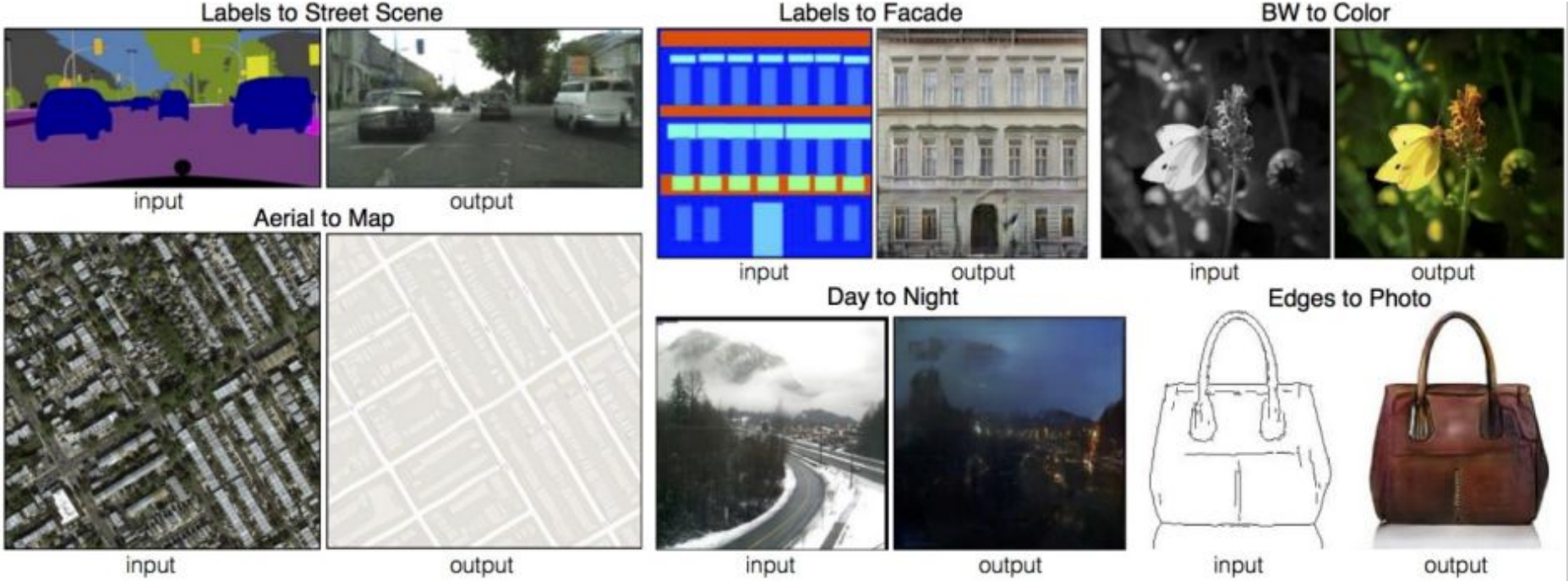
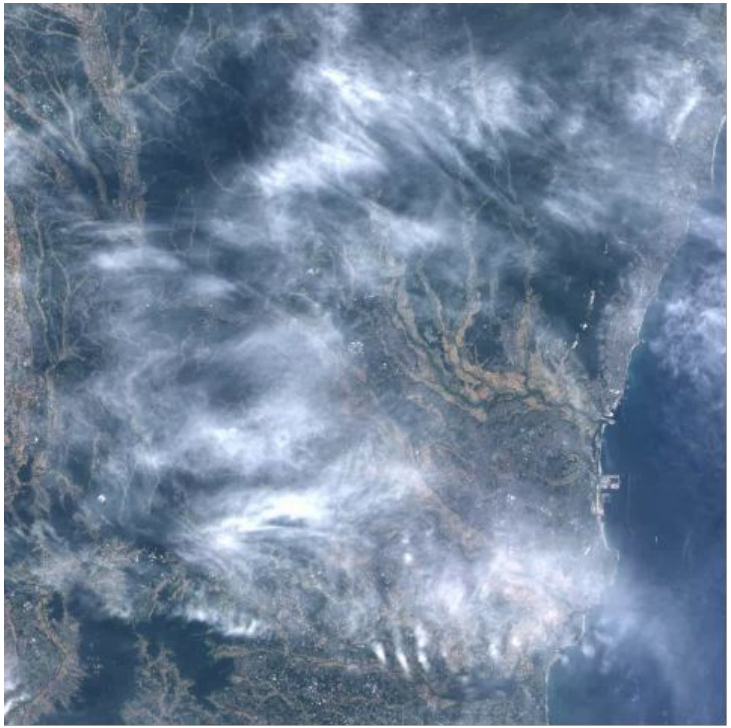
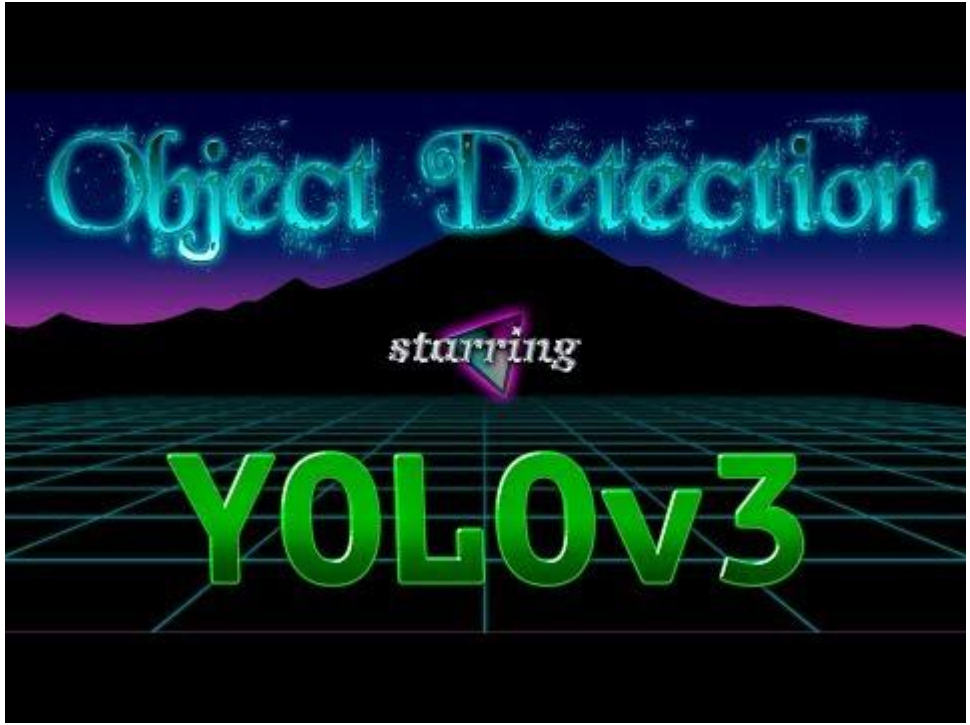


Image-to-image translation with conditional adversarial networks
Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. - CVPR 2017

Cloud Removal



Object detection



Semantic Segmentation

Pose estimation

Image manipulation

Image manipulation

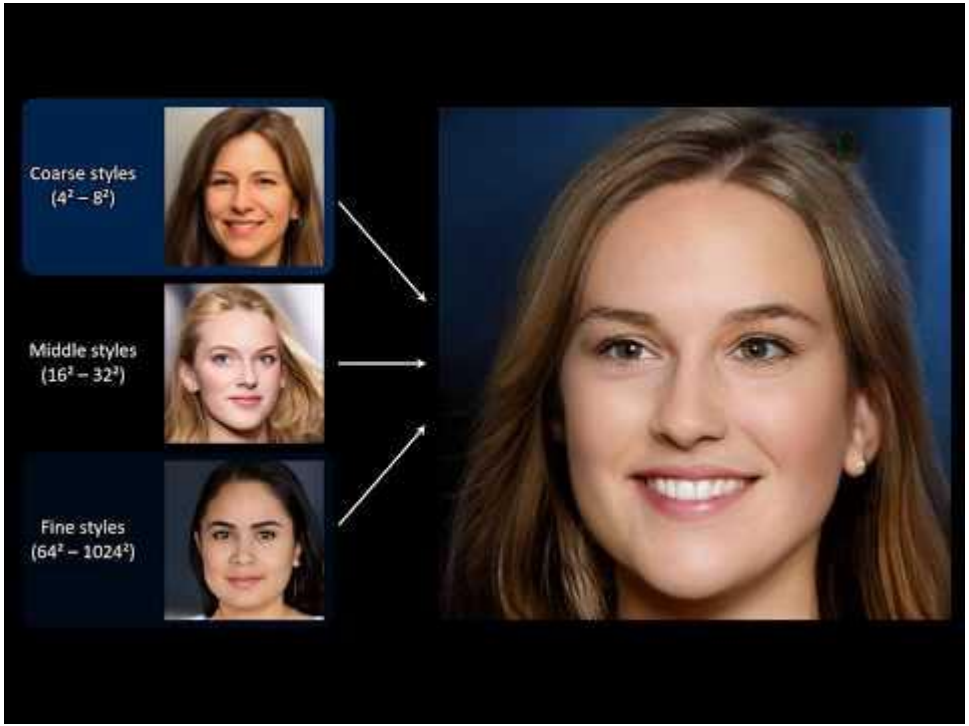


Image manipulation

Semantic Segmentation

Applications

Scene understanding

- autonomous driving
- object localization
- ….

Semantic layout

- It is the ground truth of a semantic segmentation task
- Has the same size of the input
- for every pixel, indicates the class of belonging

Image credits: https://www.v7labs.com/blog/semantic-segmentation-guide

Output

- The output is a cube with n channels
- each channel represents a class

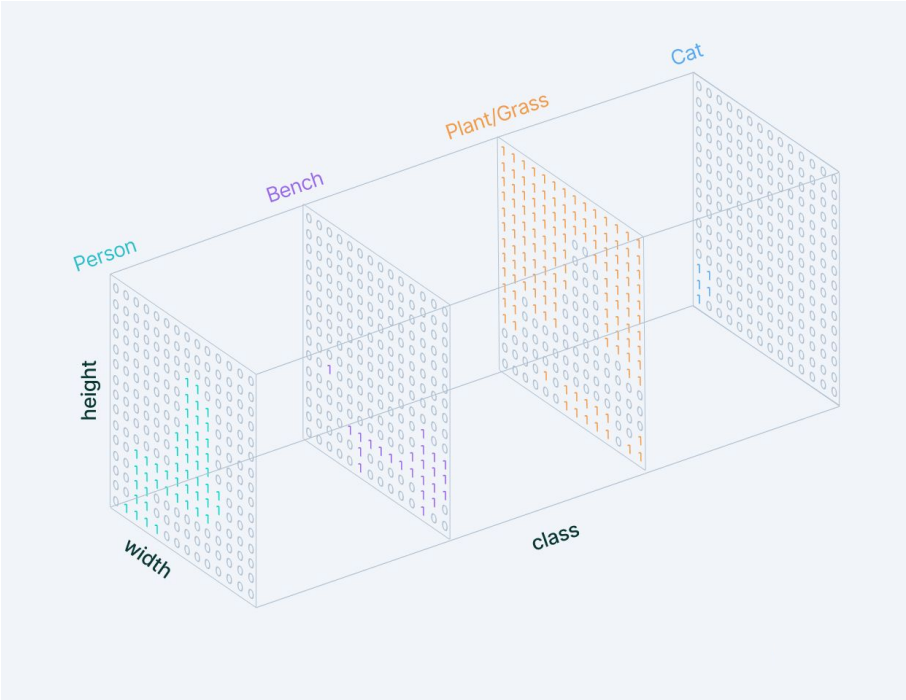
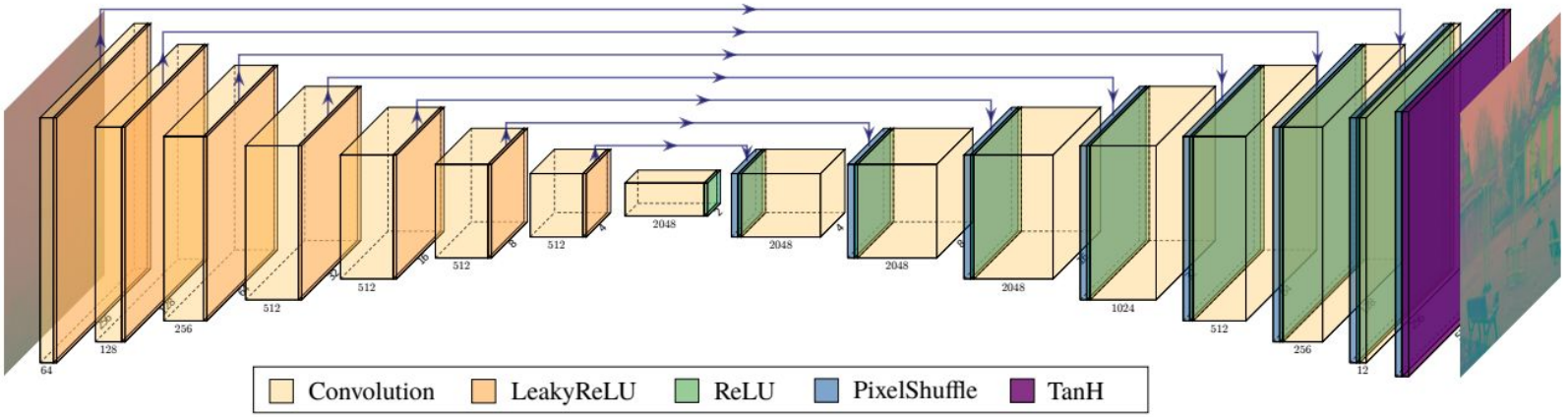


Image credits: https://www.v7labs.com/blog/semantic-segmentation-guide

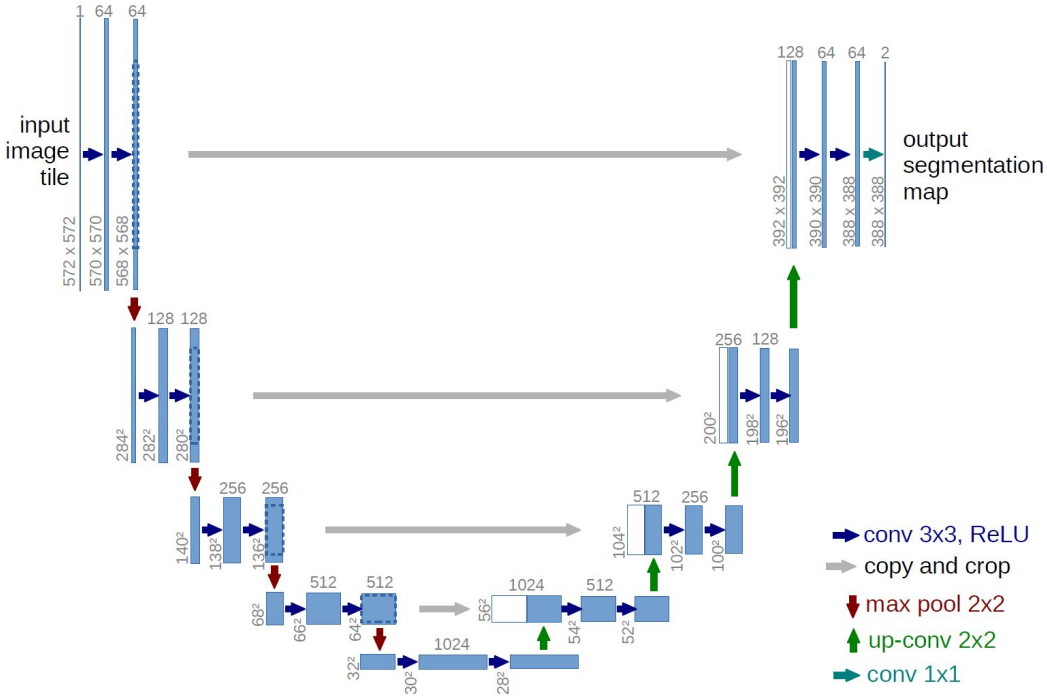
UNET

- The typical shape of an image-to-image CNN is the hourglass shape
- It is composed by an encoder, which projects the input image in a compressed feature space and
- a decoder, which reconstruct the image



Why it is called UNET?

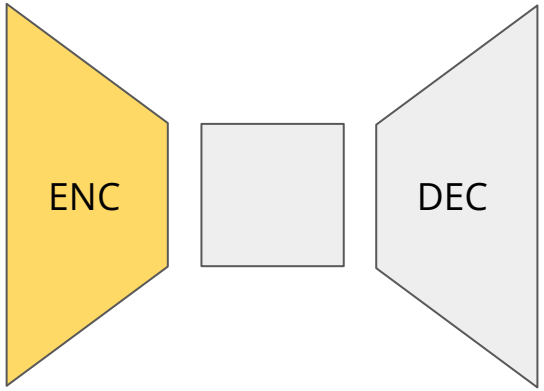
- Rearranging graphically the layers, the network has the shape of a U



Pretrained network in hourglass-shaped CNN

Encoder can be a pretrained network on a classification task, e.g.:

- resnet18
- resnet50
- VGG

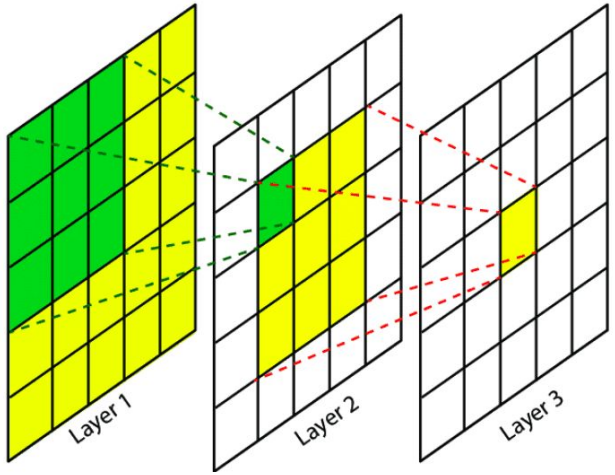


can be a pretrained network on a classification task, e.g.:

- resnet18

Receptive field or field of view

- It is the region in the input space that a particular CNN's feature is affected by
- It is the part of input that after an operation becomes a feature



- A feature in layer 2 is a local operation applied on 9 elements of the input
- A feature in layer 3 is the result of a cascade of operations on 5x5 elements of the input

Receptive field or field of view

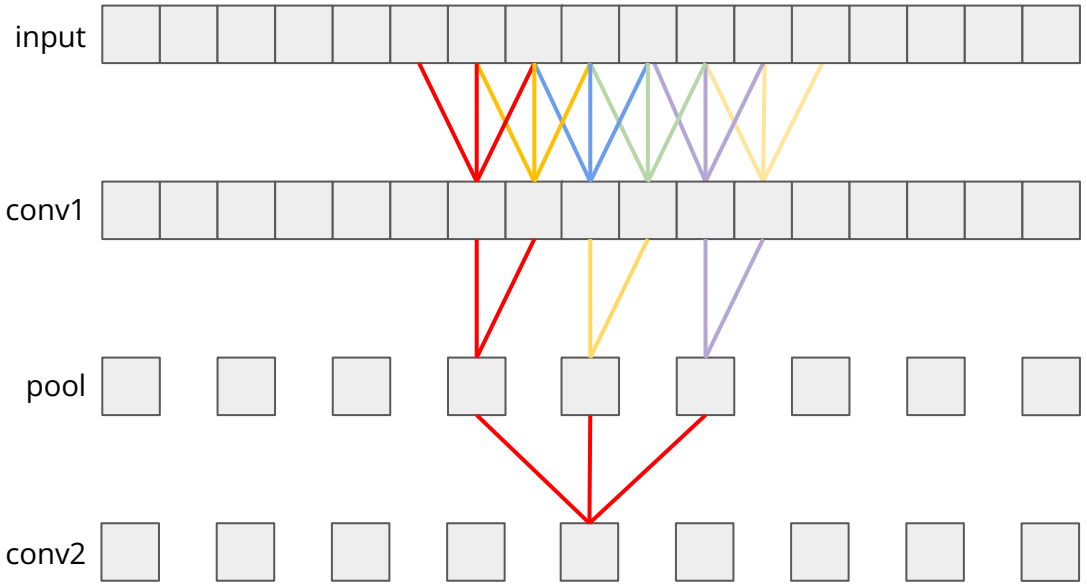
Example: calculate the receptive field of this network

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        return x
```

Receptive field or field of view

Example: calculate the receptive field of this network

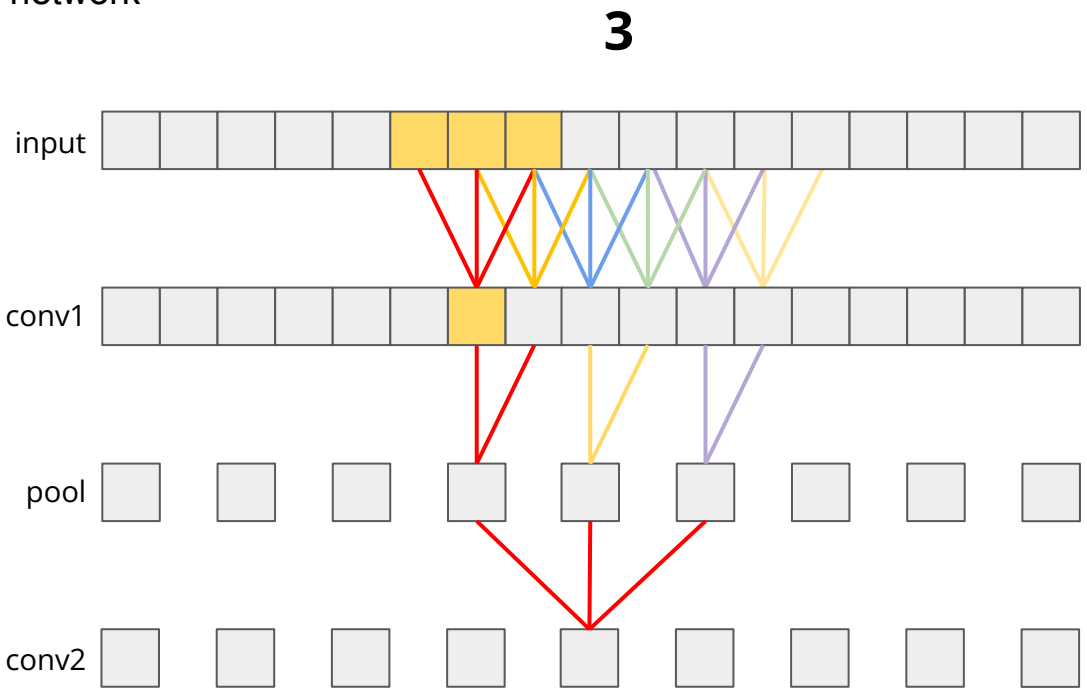
```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        return x
```



Receptive field of conv1

Example: calculate the receptive field of this network

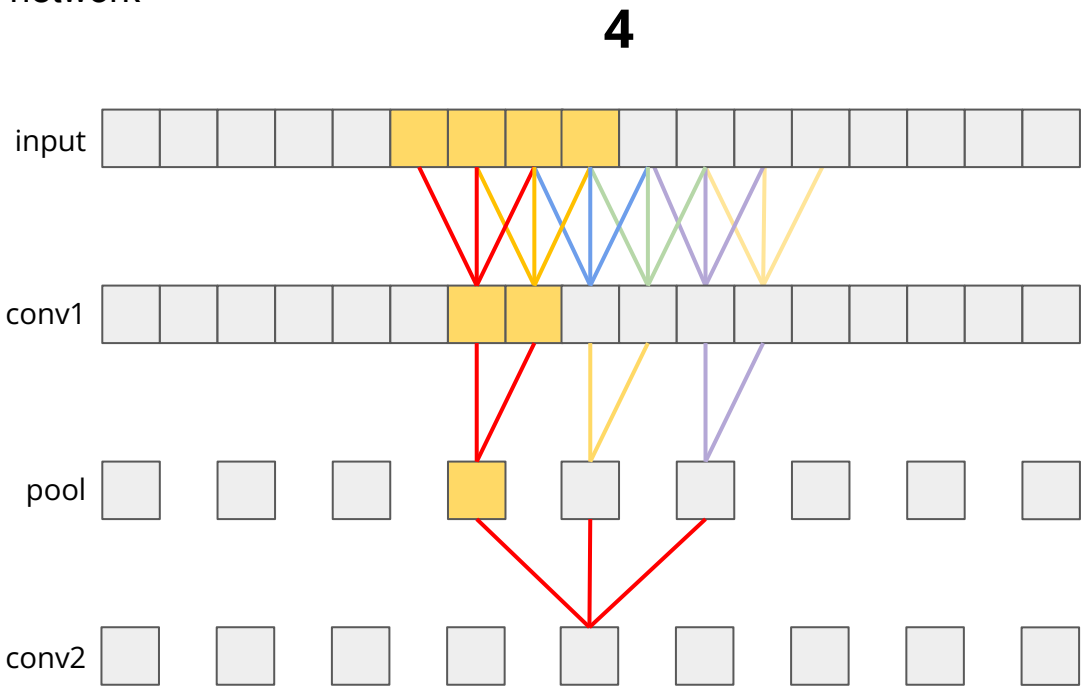
```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        return x
```



Receptive field of pool

Example: calculate the receptive field of this network

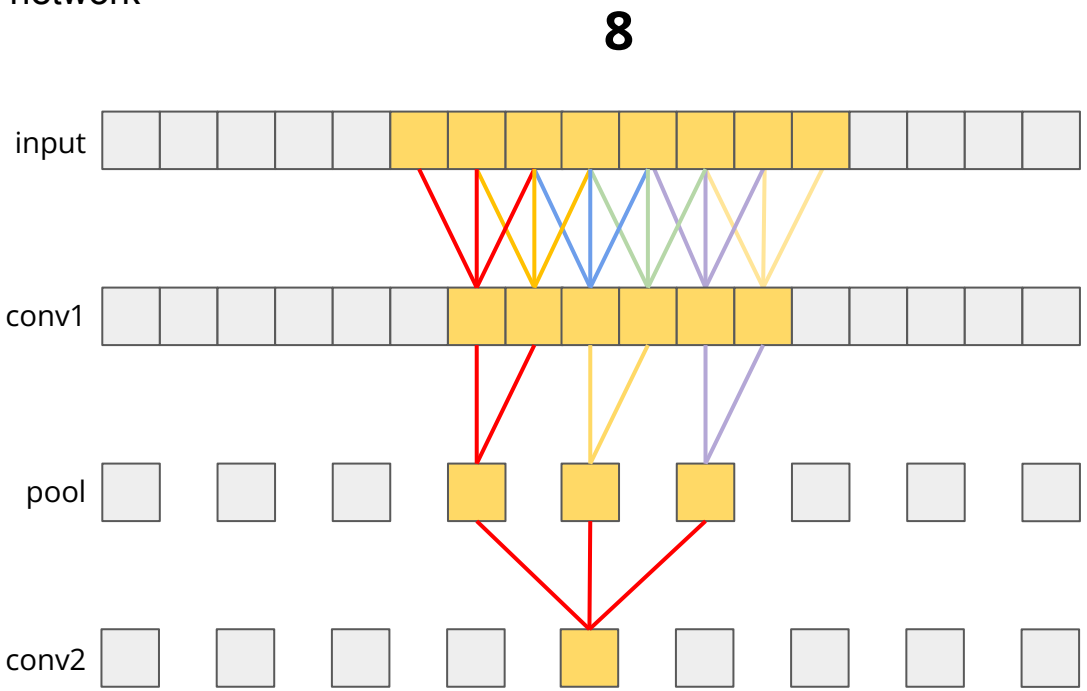
```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        return x
```



Receptive field of the network

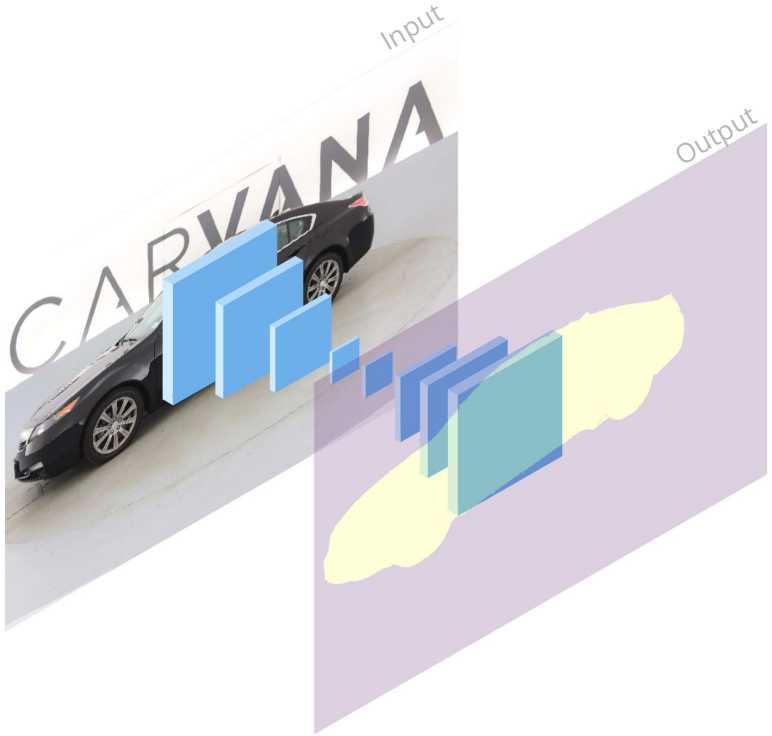
Example: calculate the receptive field of this network

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        return x
```



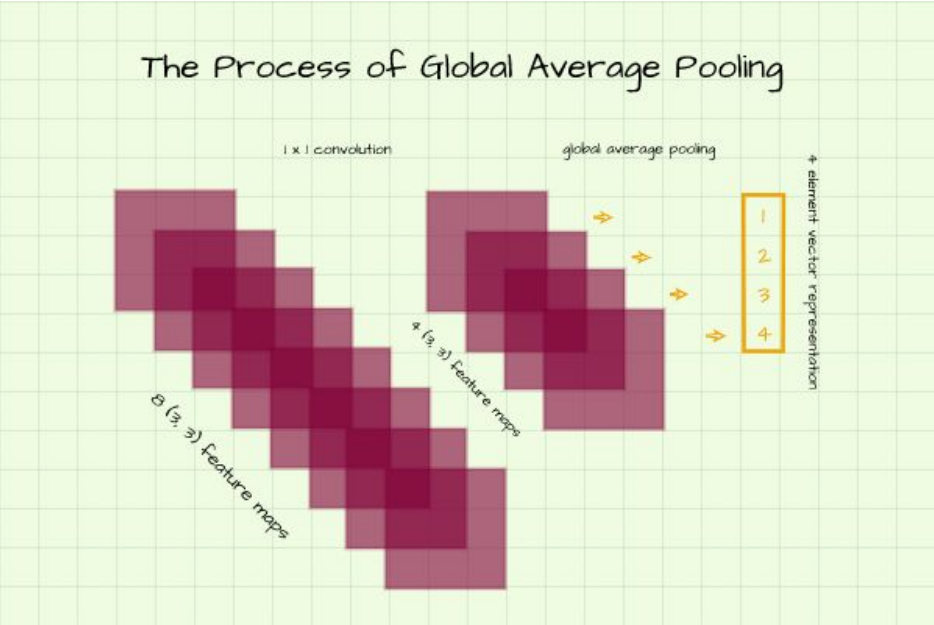
Fully Convolutional Networks (FCN)

- Are networks composed of layers that can be applied to any input size higher than the receptive field
- e.g.: networks composed only of conv+relu+bn layers



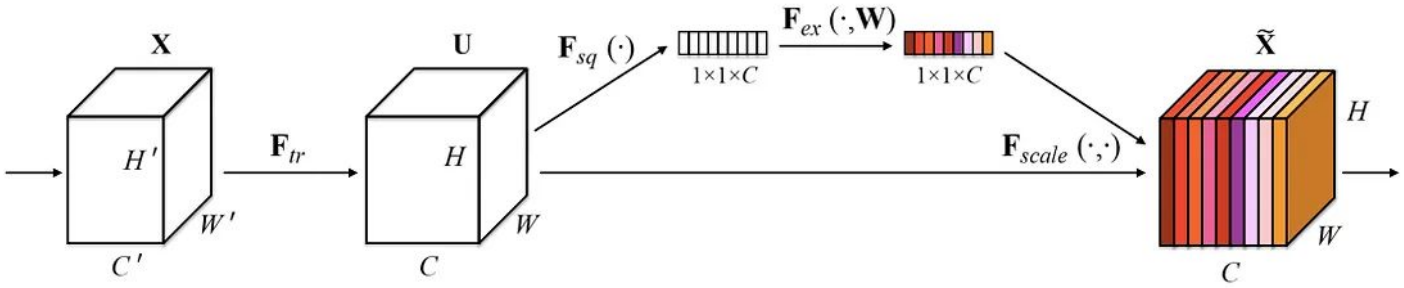
Global Average Pooling

- Useful layer to reduce an arbitrary spatial dimension in a vector
- n channels → n outputs
- performs global average, channelwise



Squeeze and Excitation

To significantly increase the receptive field, it is possible to use this technique in the middle of UNet



The idea is to weight every channel by a weight computed on all activations. To do so:

1. Global average pooling
2. Linear projection to a vector of reduced size + ReLU
3. Linear projection back to number of channels
4. Weight each channel by the corresponding weight computed with previous steps

Pytorch library for semantic segmentation with pretrained nets

Do not re-invent the wheel…there are many ready solutions

https://github.com/qubvel/segmentation_models.pytorch

```
import segmentation_models_pytorch as smp

model = smp.Unet(
    encoder_name="resnet34",        # choose encoder, e.g. mobilenet_v2 or efficientnet-b7
    encoder_weights="imagenet",     # use `imagenet` pre-trained weights for encoder initialization
    in_channels=3,                  # model input channels (1 for gray-scale images, 3 for RGB, etc.)
    classes=10,                     # model output channels (number of classes in your dataset)
    decoder_attention_type='sqse'   # squeeze and excitation or none
)
```

Upsampling strategies

How can we upsample the feature maps? Two strategies:

- **handcrafted**
  - the feature map is upsampled by traditional methods and then refined

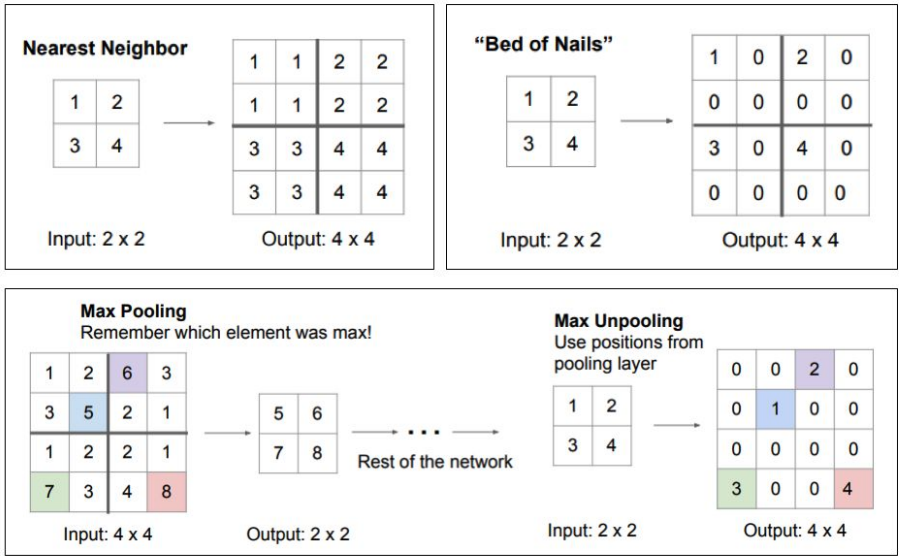


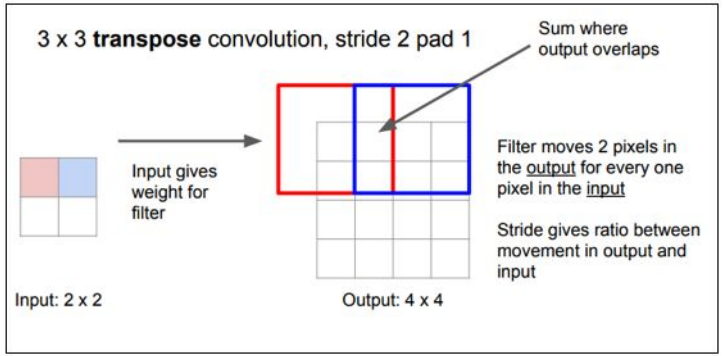
Image credits: https://www.jeremyjordan.me/semantic-segmentation/

Upsampling strategies

How can we upsample the feature maps? Two strategies:

- **learned**
  - the upsampling is learned during training

Deconvolution (Transposed Convolution)



Pixel shuffle

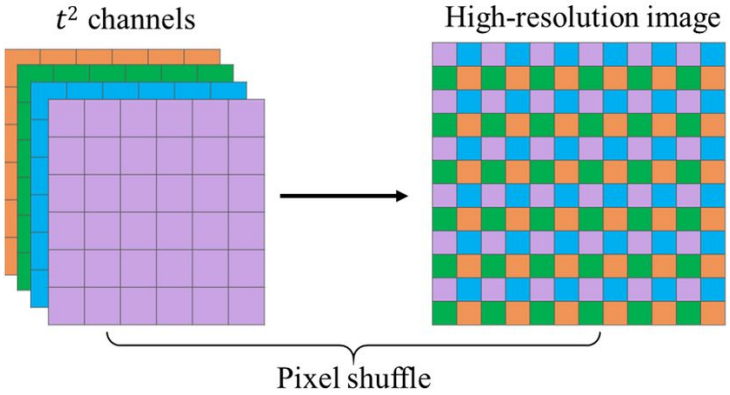


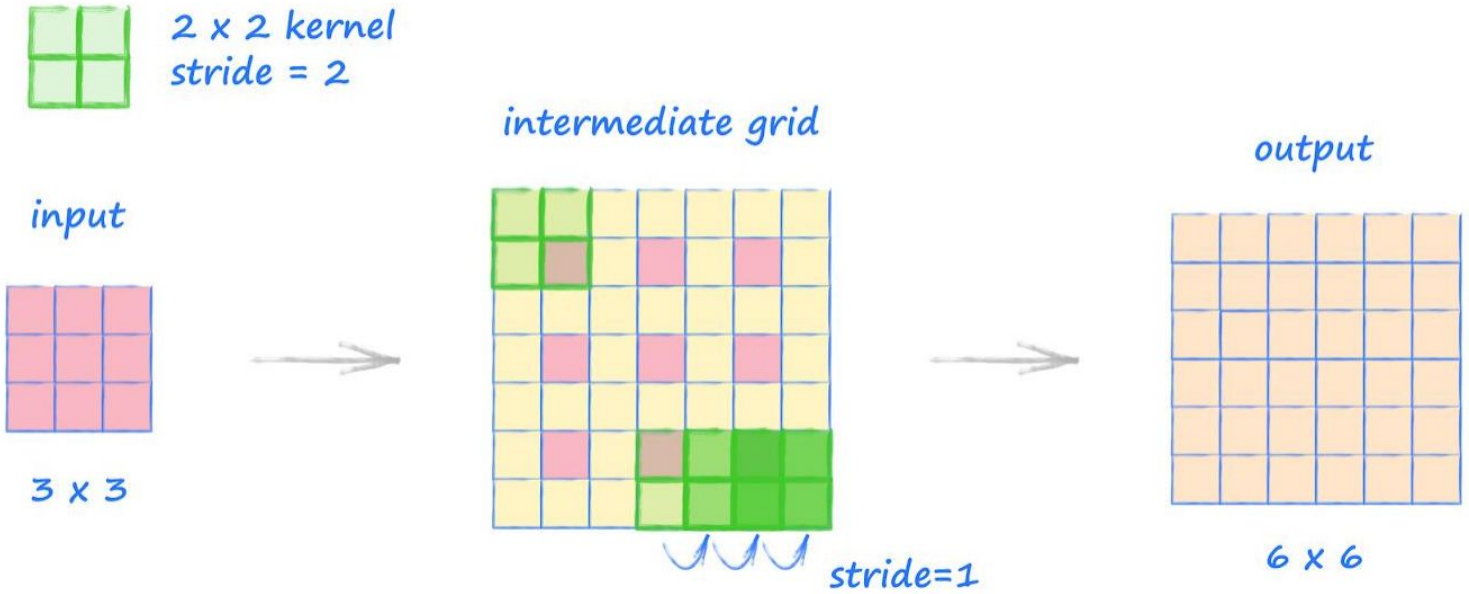
Image credits: https://www.jeremyjordan.me/semantic-segmentation/

Transposed Convolution

Es1: Transpose Convolution With Stride 2, No Padding

```
nn.ConvTranspose2d(in_channels, out_channels, kernel_size=2, stride=2, padding=0)
```

The transpose convolution is commonly used to expand a tensor to a larger tensor. This is the opposite of a normal convolution which is used to reduce a tensor to a smaller tensor



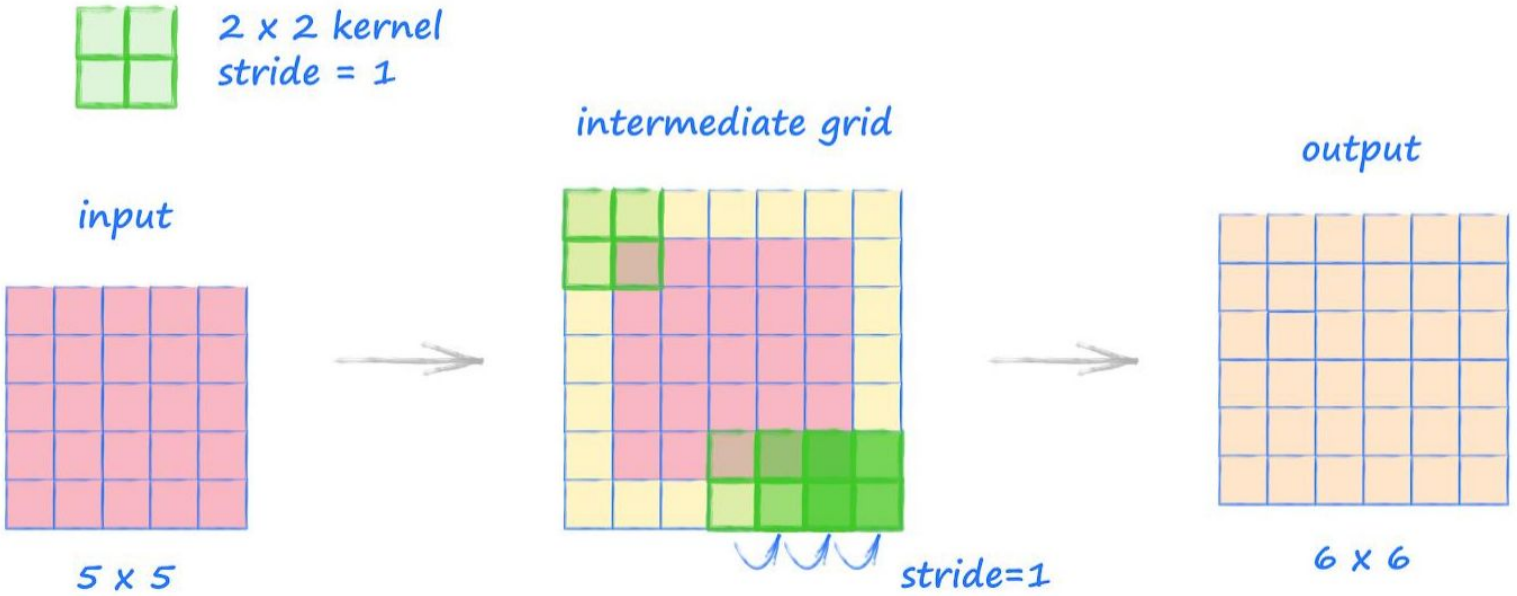
Credits: https://indico.cern.ch/event/996880/contributions/4188468/attachments/2193001/3706891/ChiakiYanagisawa_20210219_Conv2d_and_ConvTransposed2d.pdf

Transposed Convolution

Es2: Transpose Convolution With Stride 1, No Padding

```
nn.ConvTranspose2d(in_channels, out_channels, kernel_size=2, stride=1, padding=0)
```

In the previous example we used a stride of 2 because it is easier to see how it is used in the process. In this example, we use a stride of 1.



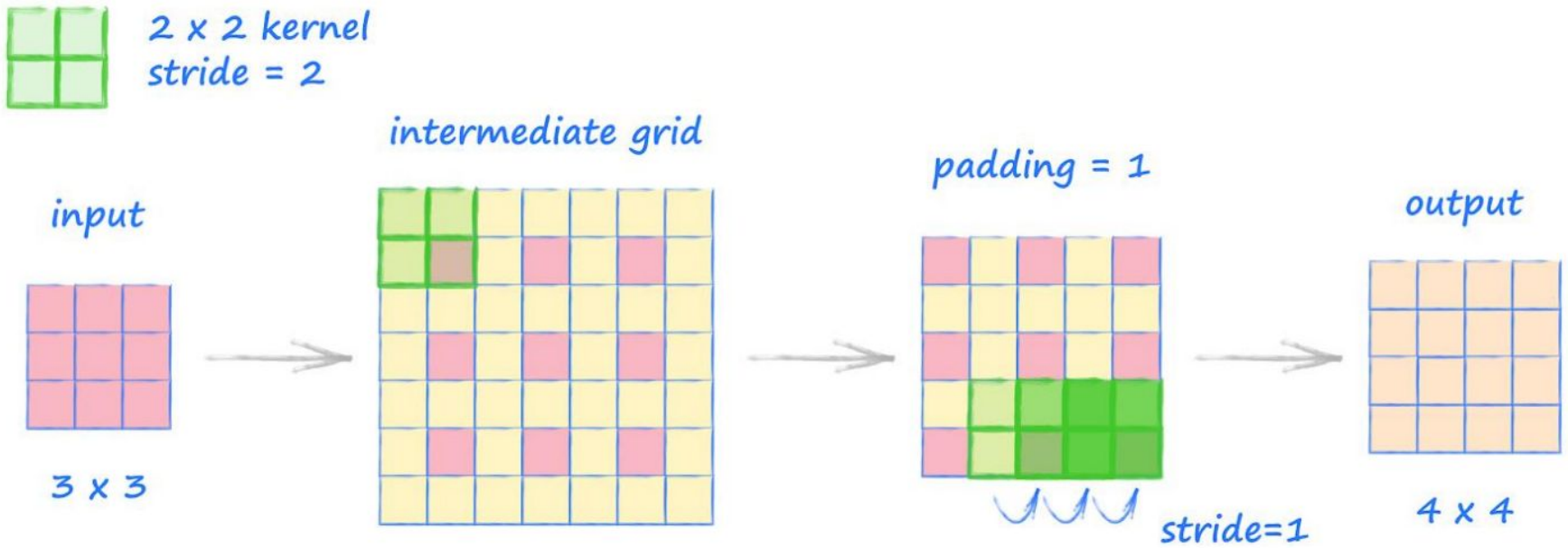
Credits: https://indico.cern.ch/event/996880/contributions/4188468/attachments/2193001/3706891/ChiakiYanagisawa_20210219_Conv2d_and_ConvTransposed2d.pdf

Transposed Convolution

Es3: Transpose Convolution With Stride 2, With Padding 1

```
nn.ConvTranspose2d(in_channels, out_channels, kernel_size=2, stride=2, padding=1)
```

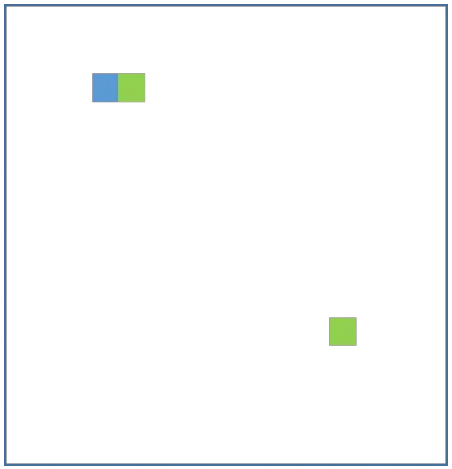
In this transpose convolution example we introduce padding. Unlike the normal convolution where padding is used to expand the image, here it is used to reduce it.



Credits: https://indico.cern.ch/event/996880/contributions/4188468/attachments/2193001/3706891/ChiakiYanagisawa_20210219_Conv2d_and_ConvTransposed2d.pdf

Pixelwise cross entropy loss

- Pixelwise cross entropy often used but has an issue
- Suppose the following situation:
  - for convenience, ground truth and predictions are placed in the same image
  - **blue**: the gt pixels
  - **green**: the predicted pixels



- Both predictions will be punished equally, despite one is very close to the ground truth

Image courtesy: https://medium.com/swlh/understanding-focal-loss-for-pixel-level-classification-in-convolutional-neural-networks-720f19f431b1

Distance-Aware Focal loss

- It is a modified version of the cross entropy loss
- It is distance aware
  - prediction error depends from the distance w.r.t. the ground truth

$$FL(p_t) = -\alpha(1-p_t)^{\gamma} log(p_t)$$

  - p_t is the measurement of prediction accuracy. When perfect, the term $(1-p_t)$ will become 0 and the loss will focus on areas that have not been well trained

- In second step, background pixels within a small radius of the foreground pixel are allowed to be predicted to 1

$$L_{det} = \frac{-1}{N} \sum_{c=1}^{C} \sum_{i=1}^{H} \sum_{j=1}^{W} \begin{cases} (1-p_{cij})^{\alpha} \log(p_{cij}) & \text{if } y_{cij} = 1 \\ (1-y_{cij})^{\beta} (p_{cij})^{\alpha} \log(1-p_{cij}) & \text{otherwise} \end{cases}$$

- Ycij is the distance factor obtained with a gaussian function

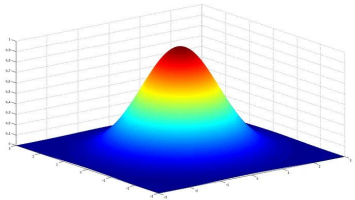
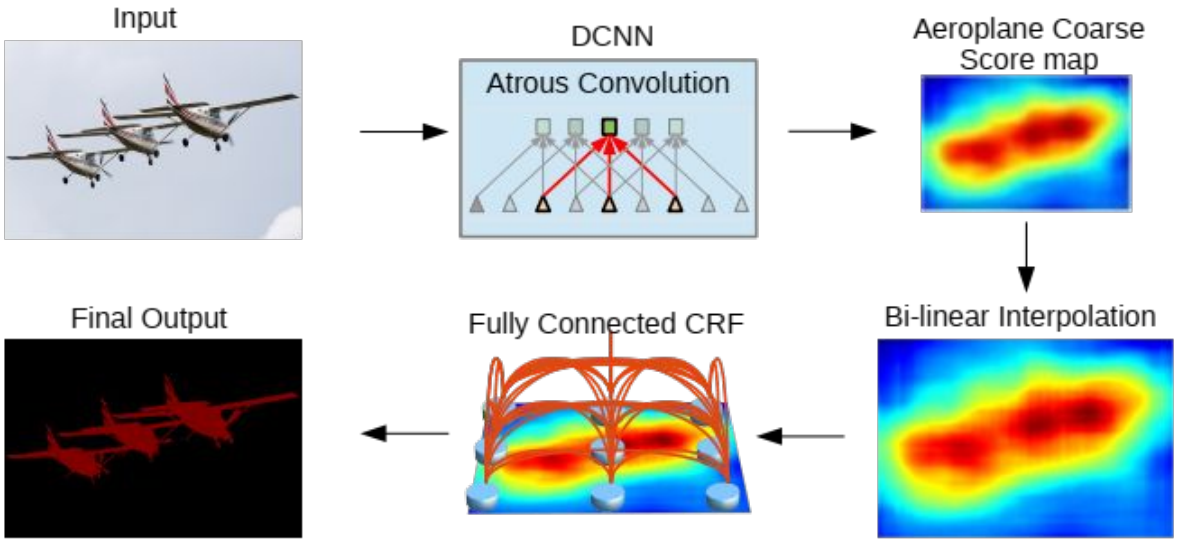


Image courtesy: https://medium.com/swlh/understanding-focal-loss-for-pixel-level-classification-in-convolutional-neural-networks-720f19f431b1

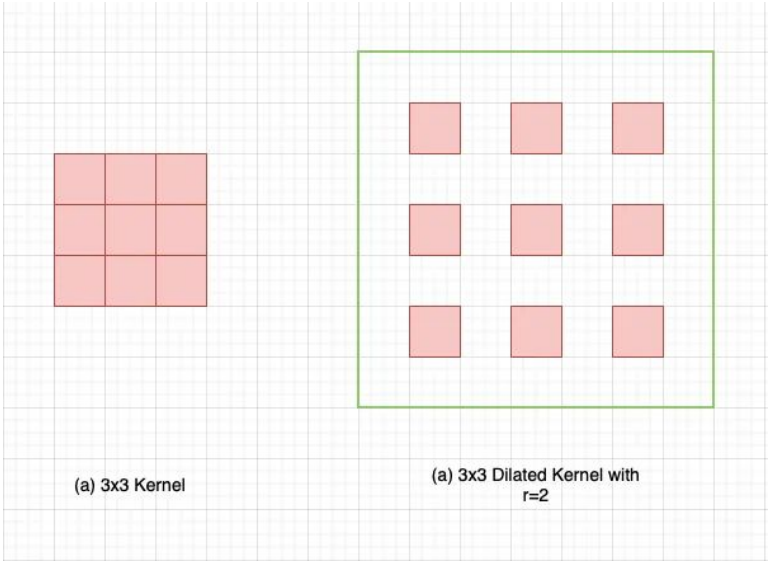
DeepLab v1

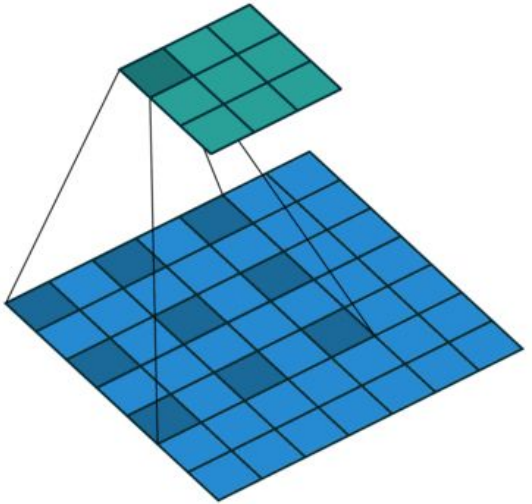
- it is one of the methods composing the state of art for semantic segmentation
- Upsampling is performed with bilinear interpolation followed by a refinement
- Final refinement is performed through a Conditional Random Fields (CRF), trained separately



Atrous convolution (a.k.a. dilated convolution)

- It is a convolution where the elements of the kernel are applied to non-contiguous neighbours of the central pixel
- It simulates an increase of receptive field as it includes a bigger portion of the input

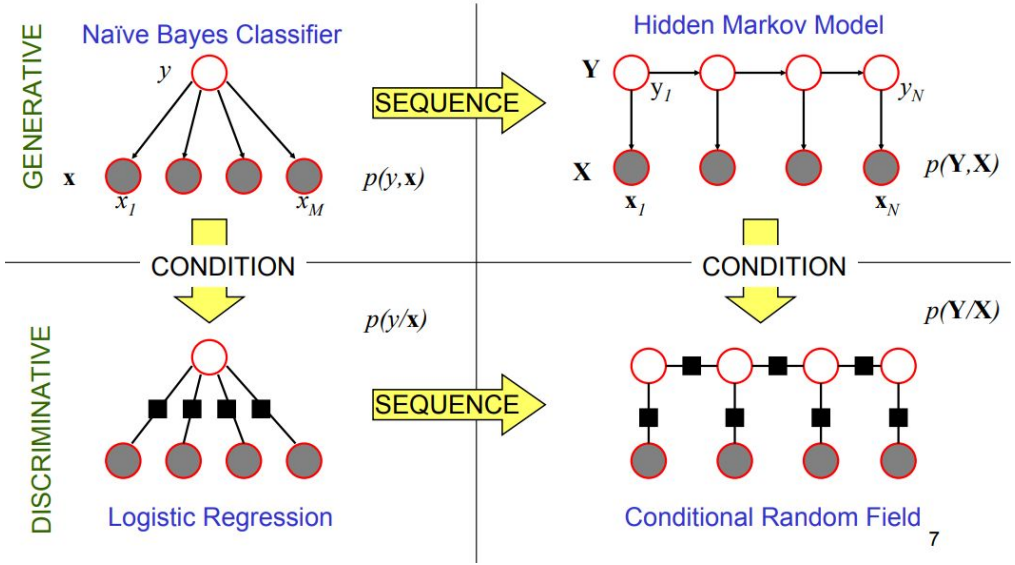




In Pytorch, you can use nn.Conv2D with dilation>1

Conditional Random Fields (CRF)

- Used for classification in sequences
- It is an extension of the Hidden Markov Model



$$p(y,x) = \prod_{t=1}^{T} p(y_t|y_{t-1})p(x_t|y_t)$$

$$p(y,x) = \frac{1}{Z(x)} exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right\}$$

Image credits: https://cedar.buffalo.edu/~srihari/CSE574/Chap13/Ch13.5-ConditionalRandomFields.pdf

Conditional Random Fields (CRF)

Assumptions

- each state depends only on its immediate predecessor
- each observation variable depends only on the current state

Limitations

- Strong independence assumptions among the observations X.
- Introduction of a large number of parameters by modeling the joint probability p(y,x), which requires modeling the distribution p(x)

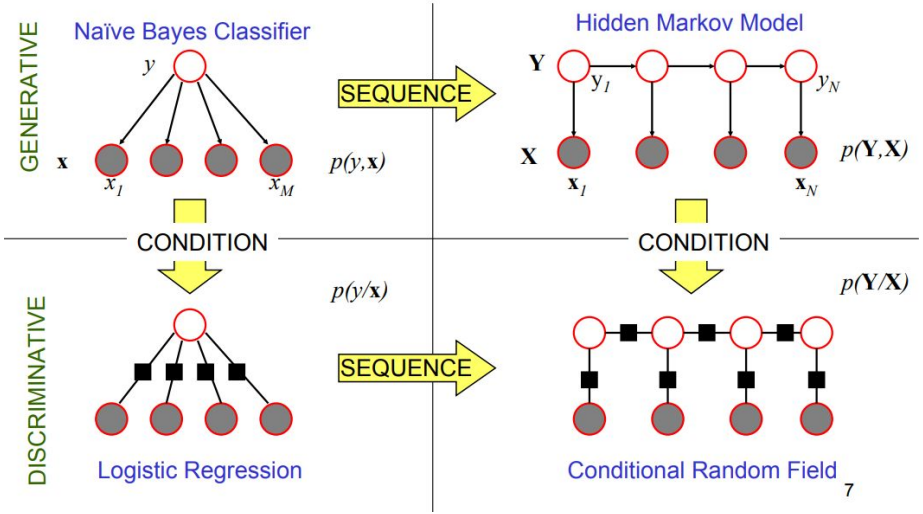


Image credits: https://cedar.buffalo.edu/~srihari/CSE574/Chap13/Ch13.5-ConditionalRandomFields.pdf

CRF for semantic segmentation

- Assumption: every pixel depends only from its neighbours
- CRF training is performed after network training

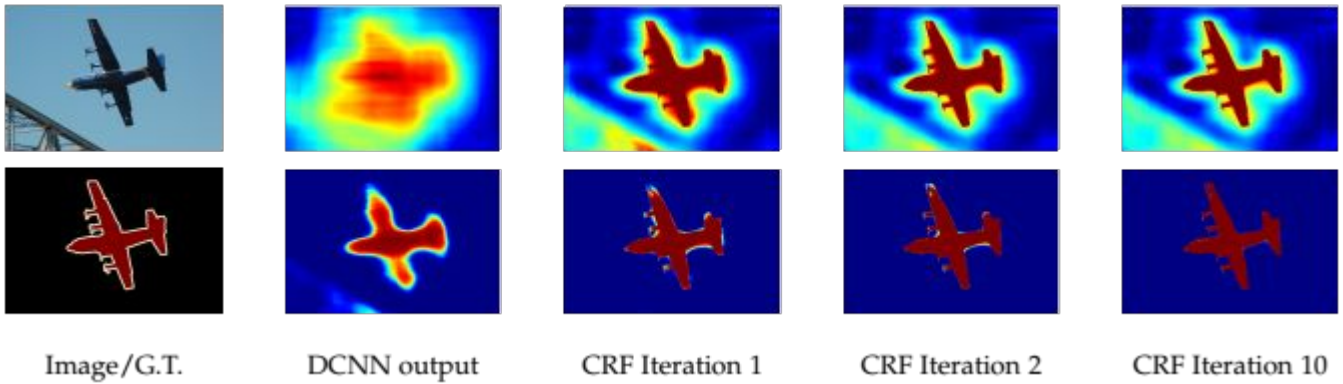
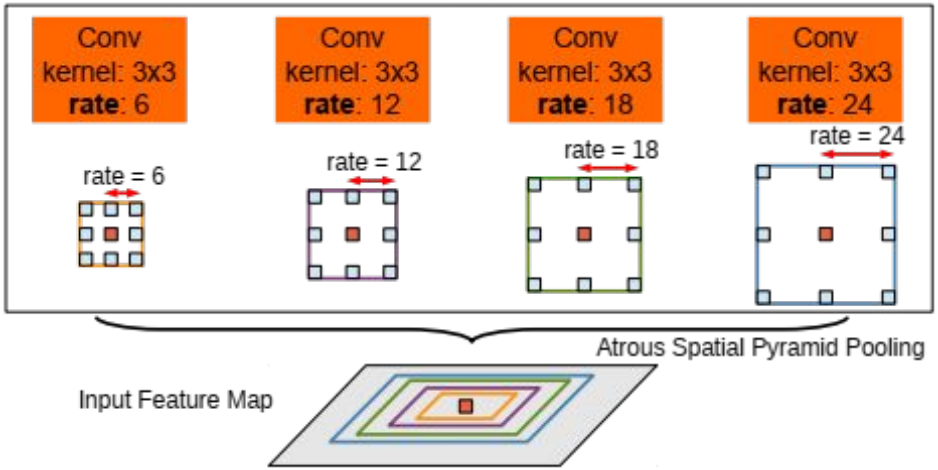


Image credits: https://cedar.buffalo.edu/~srihari/CSE574/Chap13/Ch13.5-ConditionalRandomFields.pdf

DeepLab V2

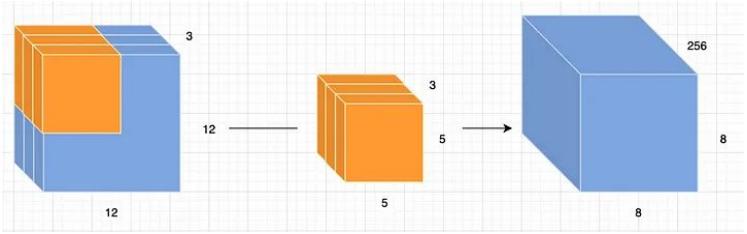
- uses Atrous Spatial Pyramid Pooling (ASPP)
- The idea is to capture details at different levels by using a set of atrous convs with different dilation terms



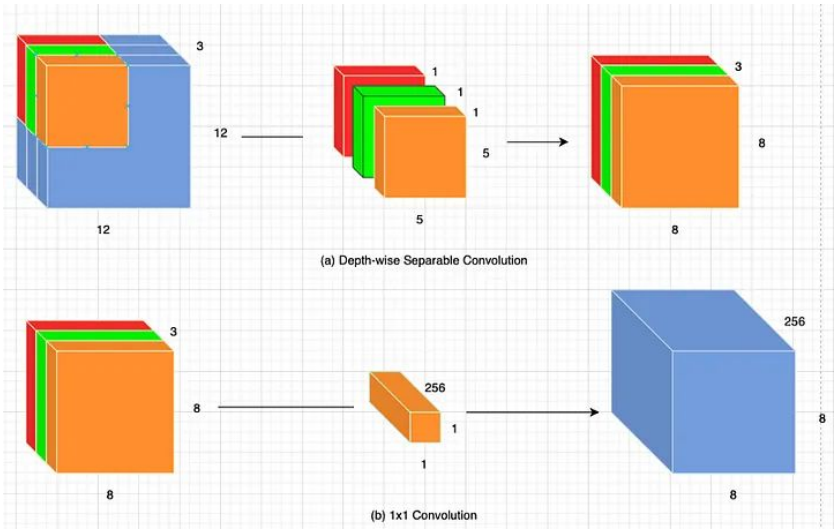
DeepLab V3

- capture sharper object boundaries by gradually recovering the spatial information
- depth-wise separable convolution to increases computational efficiency

Normal convolution



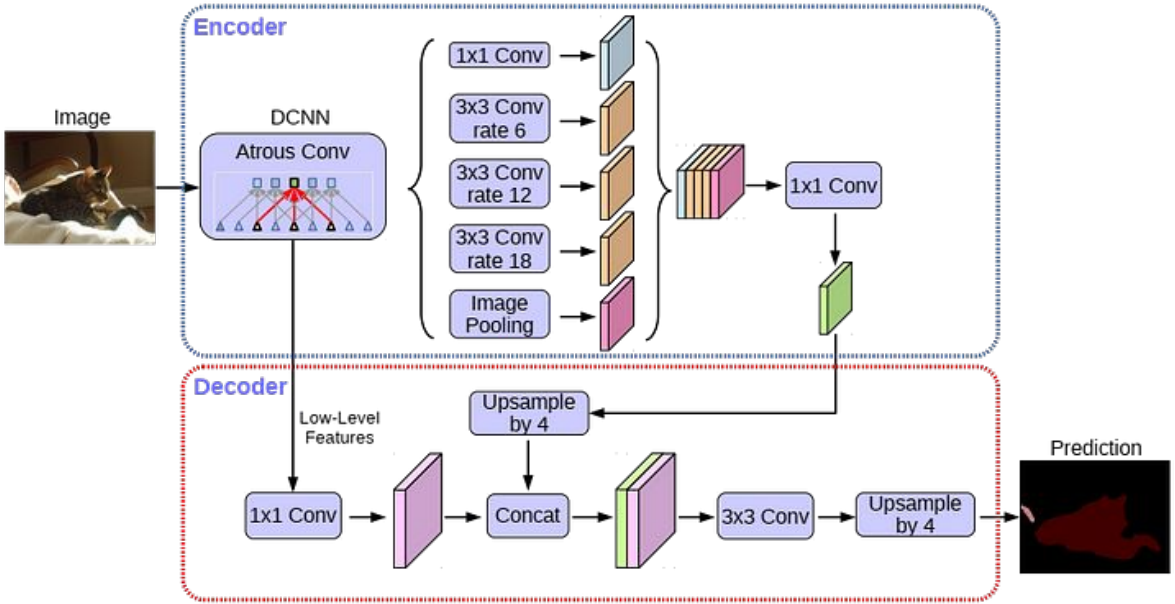
Depth-wise Separable Convolution



- in depth-wise separable conv. each filter is applied independently to each input channel
- dependency among channels is gained through 1x1 convolution
- It is a smart trick to reduce the number of params and increase efficiency

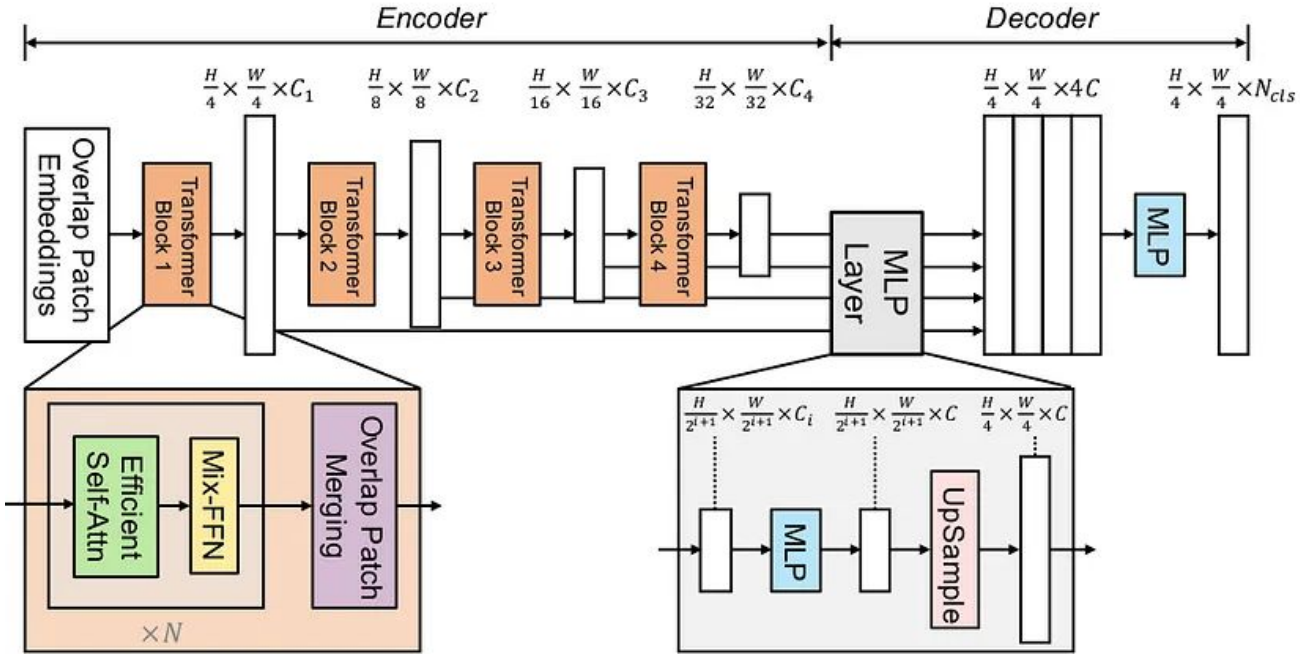
DeepLab V3+

- Encoder is a Aligned Xception instead of ResNet-101
- All max pooling operations are replaced by depth-wise separable convolution
- Concatenation of encoder features is performed by upsampling, concatenating and applying 1x1 conv



Segformer

- It is a neural architecture based on transformers
- transformers are mechanisms to learn attention across features (you will see them extensively in following lessons)



Segformer

- The effective receptive field is bigger than the one of deepLabV3+

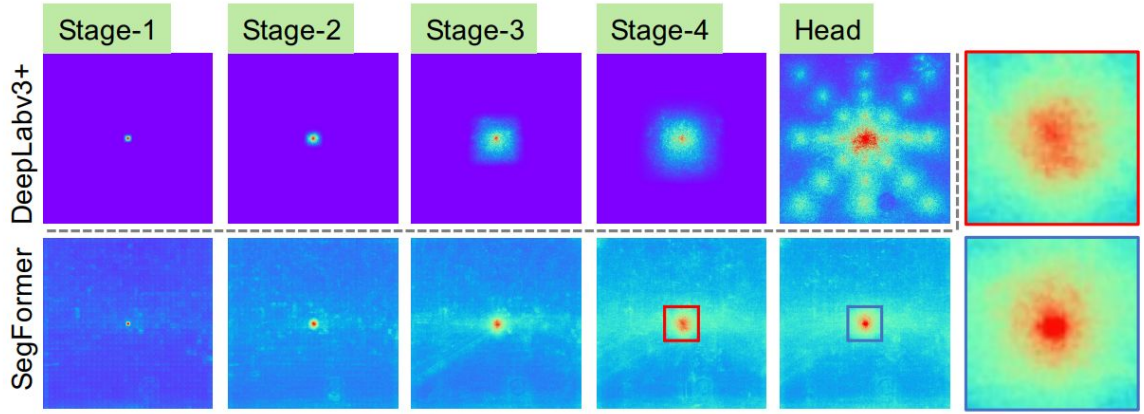
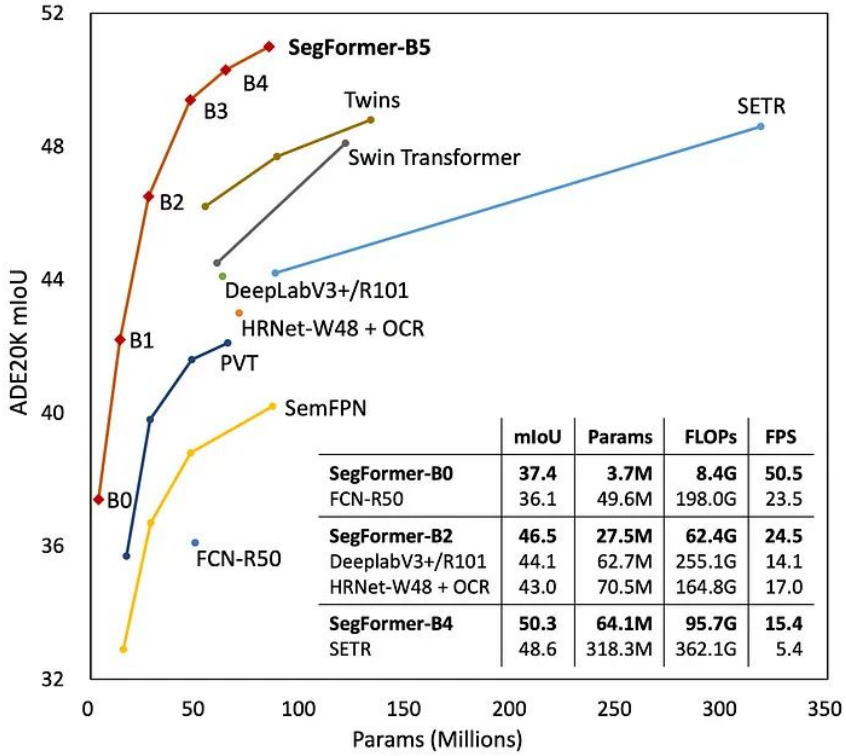


Figure 3: **Effective Receptive Field (ERF) on Cityscapes** (average over 100 images). Top row: Deeplabv3+. Bottom row: SegFormer. ERFs of the four stages and the decoder heads of both architectures are visualized. Best viewed with zoom in.

Comparison

- Transformer-based systems perform much better than other systems with less parameters



	mIoU	Params	FLOPs	FPS
SegFormer-B0	**37.4**	**3.7M**	**8.4G**	**50.5**
FCN-R50	36.1	49.6M	198.0G	23.5
SegFormer-B2	**46.5**	**27.5M**	**62.4G**	**24.5**
DeeplabV3+/R101	44.1	62.7M	255.1G	14.1
HRNet-W48 + OCR	43.0	70.5M	164.8G	17.0
SegFormer-B4	**50.3**	**64.1M**	**95.7G**	**15.4**
SETR	48.6	318.3M	362.1G	5.4

How to speed up the segmentation?

- Two streams, one global at smaller resolution, one local with reduced field of view

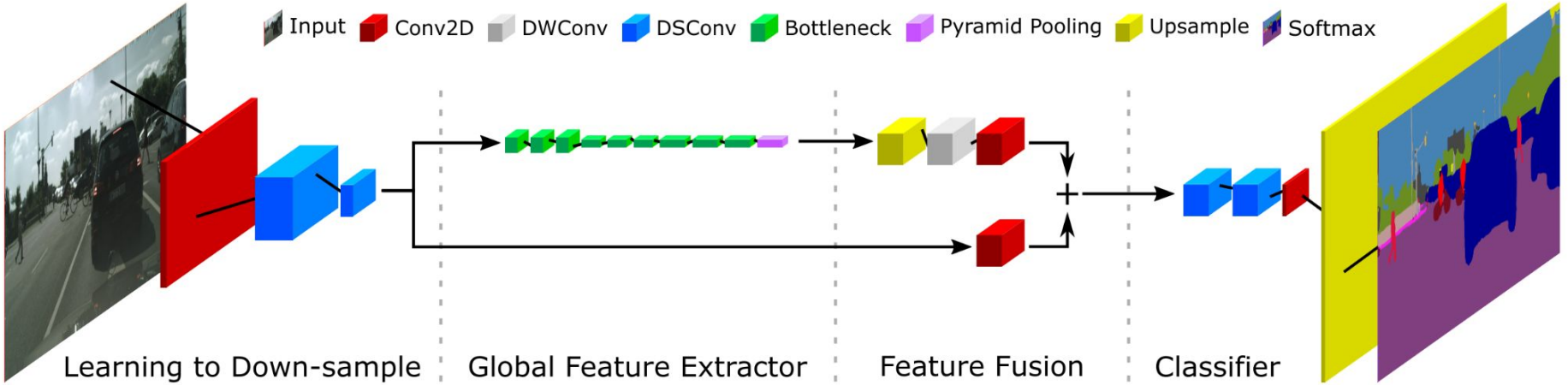


Figure 1. Fast-SCNN shares the computations between two branches (encoder) to build a above real-time semantic segmentation network.

Fast-scnn: Fast semantic segmentation network.
Poudel, R. P., Liwicki, S., & Cipolla, R. (2019).

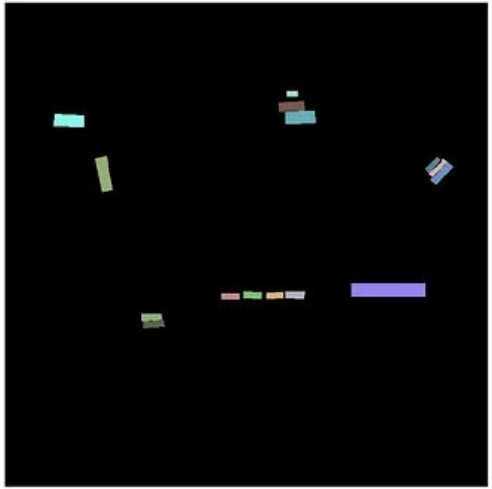
Metrics

Pixel Accuracy

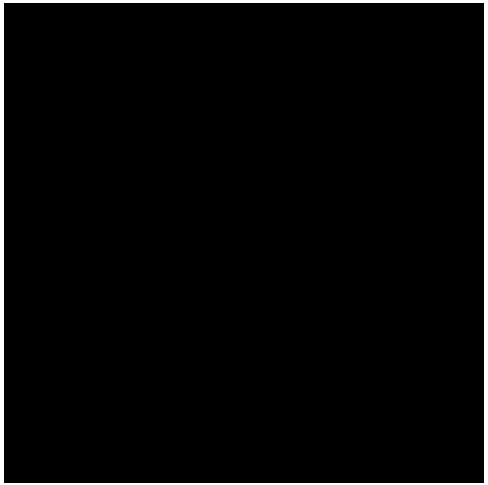
- It is the easiest metric
- Measures the percent of pixels in your image that are classified correctly

Input image

Semantic Layout



Predicted output

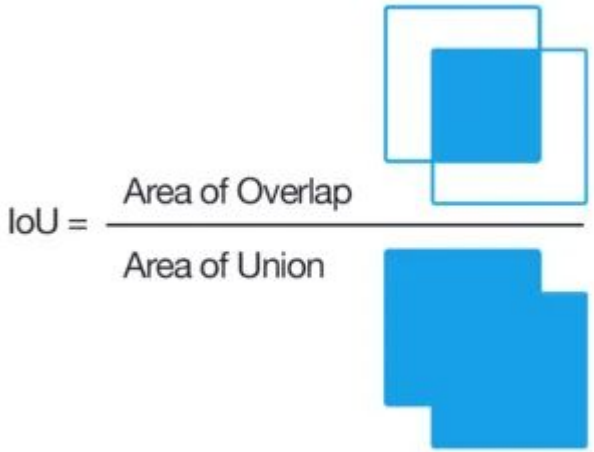


- In this configuration, the accuracy is 95%..
- Not suitable for unbalanced datasets
- Does not keep into account the shapes

https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2

Intersection Over Union (IOU) - a.k.a. Jaccard Index

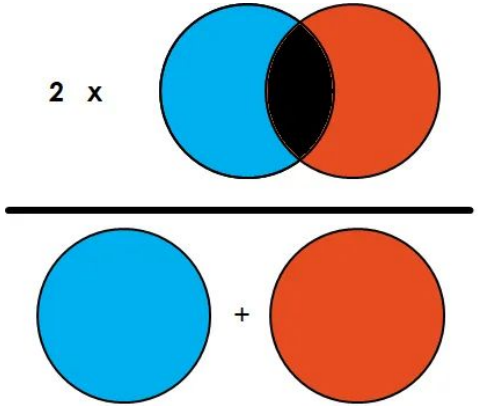
- For each class, measures the coverage of the prediction as:



- Ideally, the maximum is when the overlap among prediction and ground truth is maximum (i.e. are the same)
- What if the prediction is bigger than the gt?
  - in the class under analysis, IOU will be 1
  - in other classes will decrease

Dice coefficient

- similar to IOU
- IOU penalizes too much the errors (similar comparison to L1 vs L2)
- numerically, it is 2*area of overlap divided by the total number of pixels



- IOU measures the worst case performance while
- DICE coefficient measures the average performance

https://stats.stackexchange.com/questions/273537/f1-dice-score-vs-iou/276144#276144

Exercises

Exercise

In this exercise, you will perform the semantic segmentation tasks

You will use a variation of the PASCAL VOC 2012 dataset

- Number of categories: 20 + background
- Number of images: 2913
- train/val/test: 0.5/0.25/0.25

Instructions

- use Albumentations to:
  - perform data augmentation
  - decrease the input size to 64x64

- create dataloader
  - takes as input the transform

- Try to build a CNN for semantic segmentation and then try models from the library Segmentation

Exercise

```
class MyDataloader(torch…Dataset):
        def __init__(self, image_dir, gt_dir, filelist):
                # 1 read the file list
                self.fns = …
                # 2 save attributes
                self.image_dir = image_idr
                self.gt_dir = gt_dir

        def __getitem__(self, idx):
                # get element
                cur_fn = self.fns[idx]
                # concatenate
                img_fn = os.path.join( self.image_dir, cur_fn + '.jpg')
                gt_fn = os.path.join(self.gt_dir, cur_fn + '.png')
                # open images
                …
                # apply transforms
                …
                # return
                return img, gt
```