SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Convolution

PROF. D. HERREMANS

DORIENHERREMANS.COM

50.038 Computational Data Science

How to represent images

- We've dealt with:
  - Numbers
  - Time series
  - Text
  - What about images?

Computer vision tasks

Image Classification

Image Classification + Localization

Object Detection

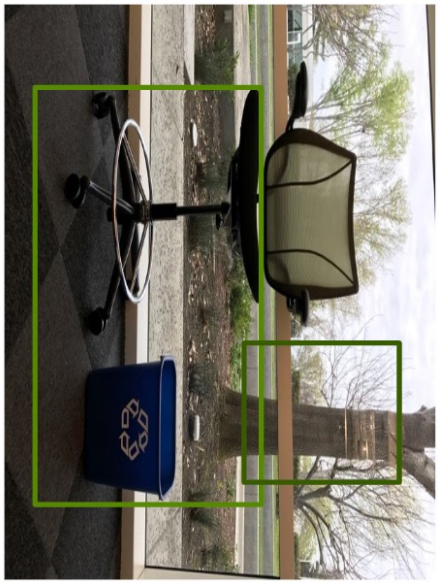


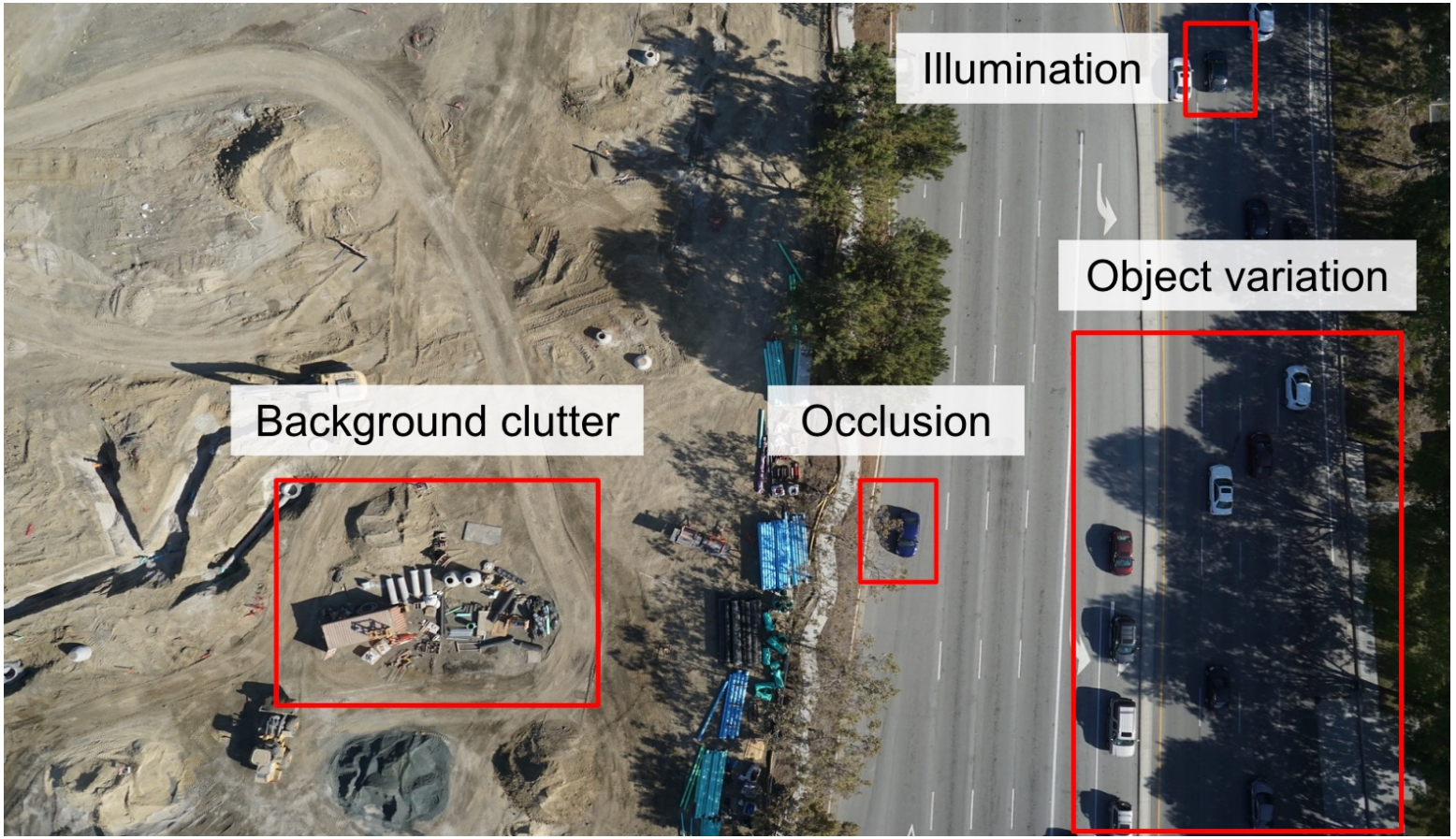
Image Segmentation

(inspired by a slide found in cs231n lecture from Stanford University)

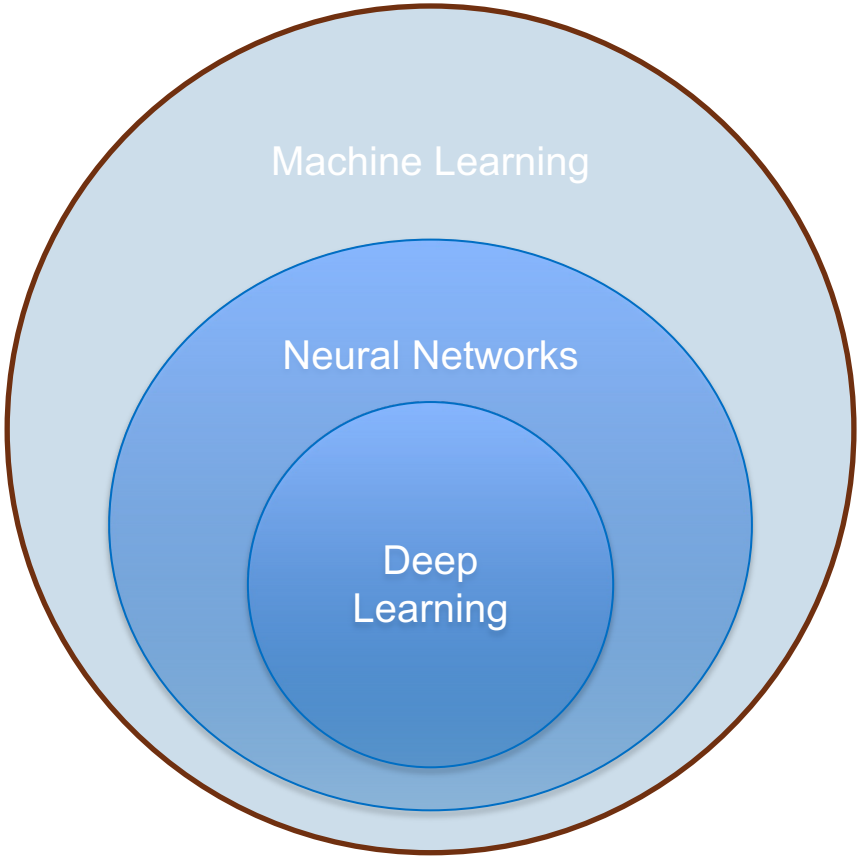
Image classification

- Very useful in data science
- Classifying images into 1 or more categories
- Training data must have images labeled with their class
  - Can be hard to find / produce training dataset
  - Possible sources: captcha's, Mechanical Turk, etc.

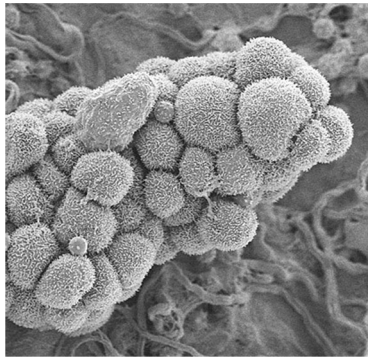
Challenges in images

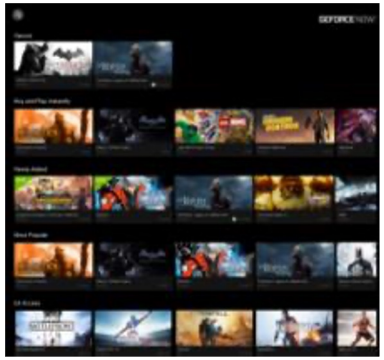


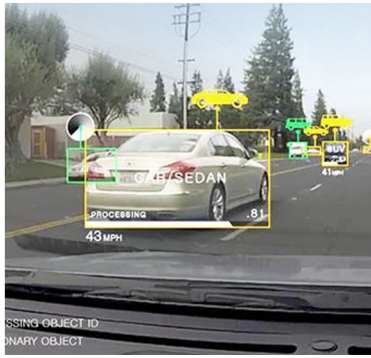
Deep learning



Deep learning with CNNs everywhere







INTERNET & CLOUD

Image Classification
Speech Recognition
Language Translation
Language Processing
Sentiment Analysis
Recommendation

MEDICINE & BIOLOGY

Cancer Cell Detection
Diabetic Grading
Drug Discovery

MEDIA & ENTERTAINMENT

Video Captioning
Video Search
Real Time Translation

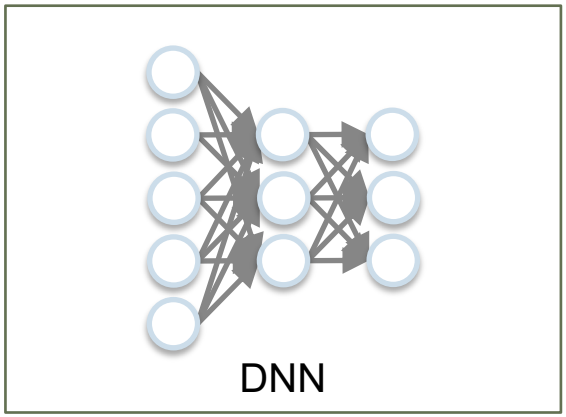
SECURITY & DEFENSE

Face Detection
Video Surveillance
Satellite Imagery

AUTONOMOUS MACHINES

Pedestrian Detection
Lane Tracking
Recognize Traffic Sign

The big bang in machine learning

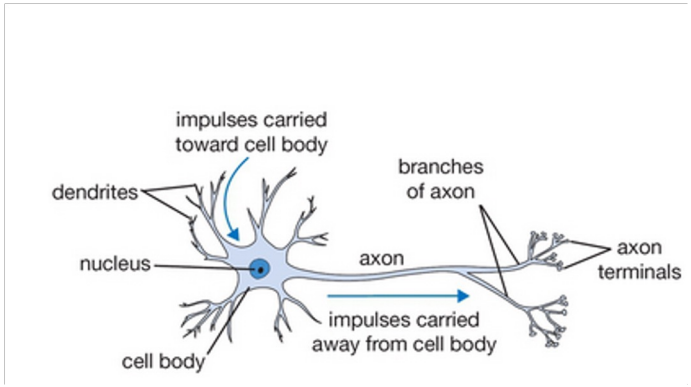


> *Google's AI engine also reflects how the world of computer hardware is changing. (It) depends on machines equipped with GPUs… And it depends on these chips more than the larger tech universe realizes.*

WIRED

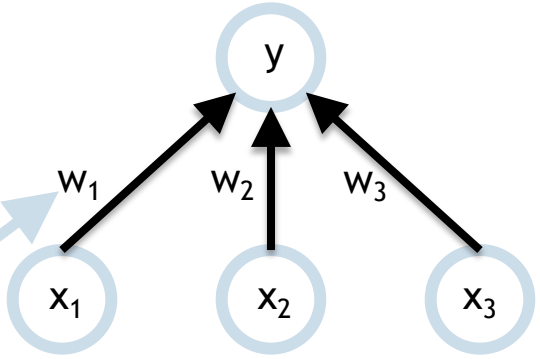
Artificial neurons

Biological neuron



From Stanford cs231n lecture notes

Artificial neuron

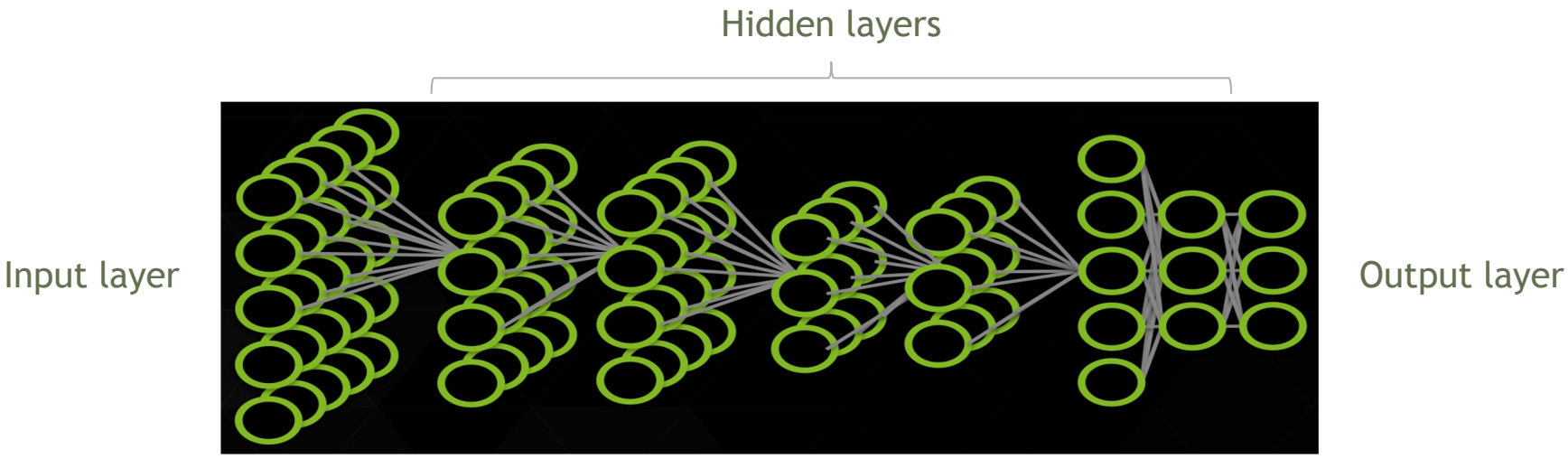


Weights (W_n)
= parameters

$y=F(w_1x_1+w_2x_2+w_3x_3)$

Artificial neural network

A collection of simple, trainable mathematical units that collectively learn complex functions



Given sufficient training data an artificial neural network can approximate very complex functions mapping raw data to output decisions

Yann LeCun

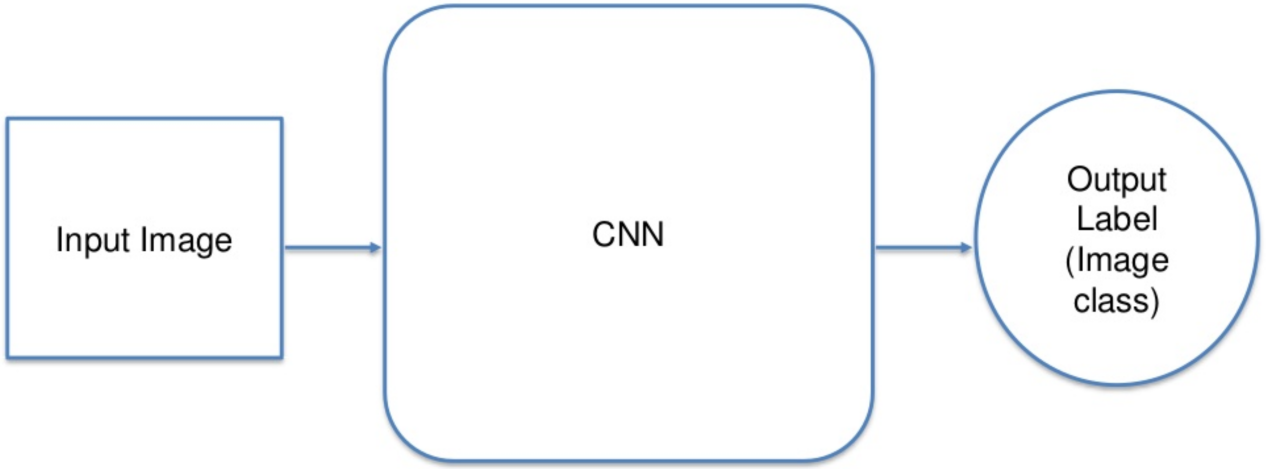
- Founding father of convolutional neural networks
- Chief AI scientist Facebook

Convolutional neural networks

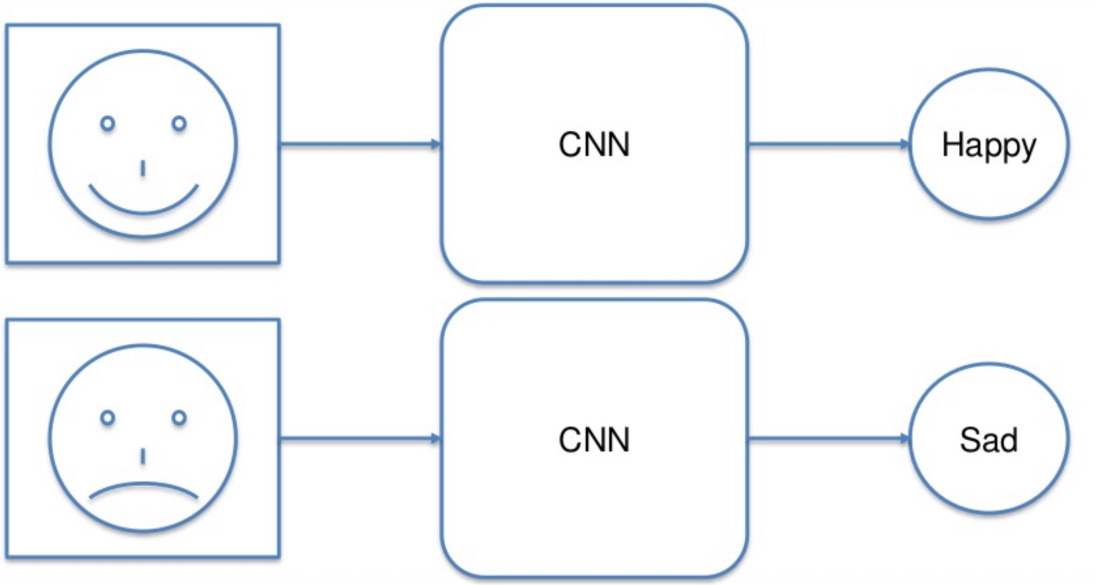
- LeCun, 1989
- "…are a specialized kind of neural network for processing data that has a known **grid-like topology**. Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image data, which can be thought of as a 2-D grid of pixels. Convolutional networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network **employs a mathematical operation called convolution**. Convolution is a specialized kind of linear operation." (Goodfellow et al., 2016)

Convolutional Neural Networks (CNNs)

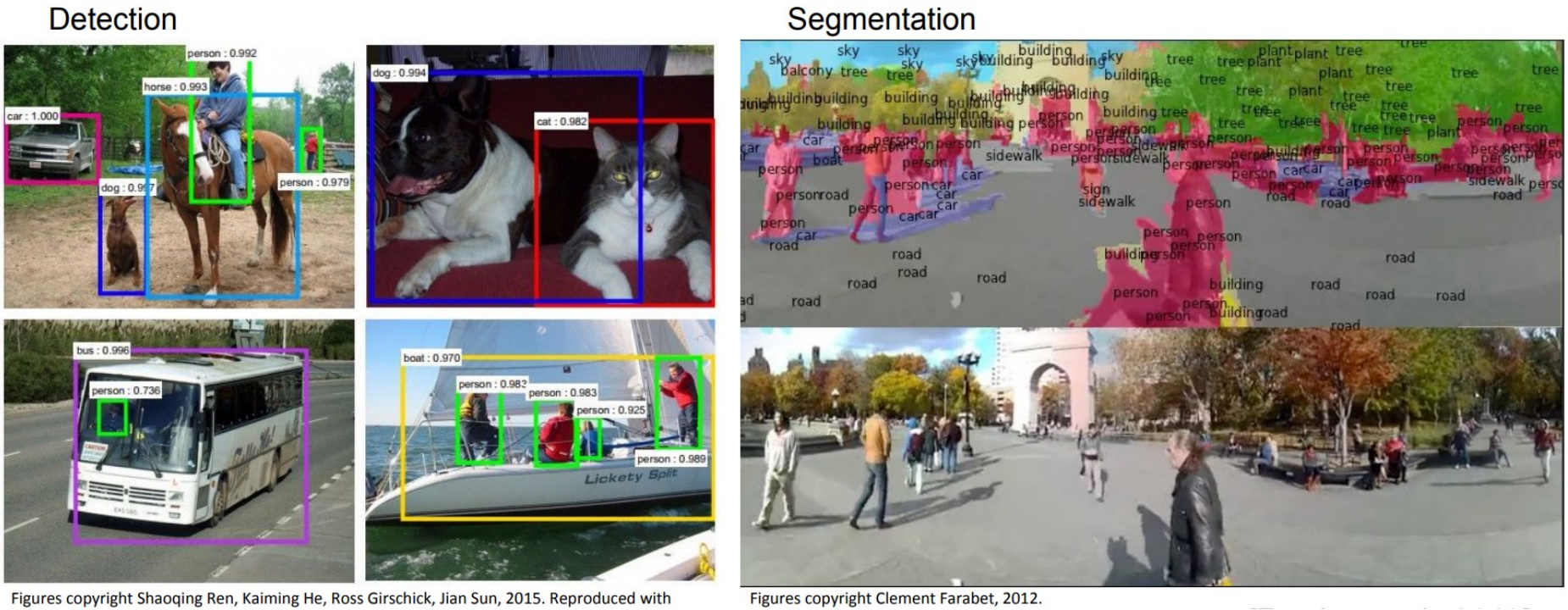
- Neural networks that use convolution in place of general matrix multiplication in at least one of their layers.



CNNs



CNNs are everywhere



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with

Figures copyright Clement Farabet, 2012.

CNNs are everywhere

Photo by Lane McIntosh. Copyright CS231n 2017.

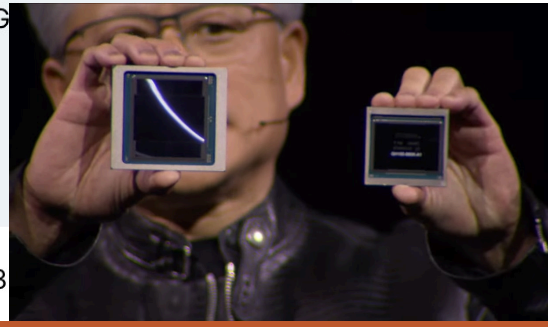
self-driving cars

This image by GBPublic_PR is licensed under CC-BY 2.0

NVIDIA Orin

- The NVIDIA DRIVE Orin SoC (system-on-a-chip) delivers 254 TOPS (trillion operations per second)
- The central computer for intelligent vehicles
- Lets developers build, scale, and leverage one development investment across an entire fleet, from Level 2+ systems all the way to Level 5 fully autonomous vehicles.
- Orin OS

NVIDIA TESLA GRAPHICS CARD	NVIDIA B200	NVIDIA H200 (SXM5)	NVIDIA H100 (SMX5)	NVIDIA H100 (PCIE)	NVIDIA A100 (SXM4)	NVIDIA A100 (PCIE4)	TESLA V100S (PCIE)	TESLA V100 (SXM2)	TESLA P100 (SXM2)	TESLA P100 (PCI-EXPRE
GPU	B200	H200 (Hopper)	H100 (Hopper)	H100 (Hopper)	A100 (Ampere)	A100 (Ampere)	GV100 (Volta)	GV100 (Volta)	GP100 (Pascal)	GP100 (Pascal
Process Node	4nm	4nm	4nm	4nm	7nm	7nm	12nm	12nm	16nm	16nm
Transistors	208 Billion	80 Billion	80 Billion	80 Billion	54.2 Billion	54.2 Billion	21.1 Billion	21.1 Billion	15.3 Billion	15.3 Bil
GPU Die Size	TBD	814mm2	814mm2	814mm2	826mm2	826mm2	815mm2	815mm2	610 mm2	610 mn
Memory Size	Up To 192 GB HBM3 @ 8.0 Gbps	Up To 141 GB HBM3e @ 6.5 Gbps	Up To 80 GB HBM3 @ 5.2 Gbps	Up To 94 GB HBM2e @ 5.1 Gbps	Up To 40 GB HBM2 @ 1.6 TB/s Up To 80 GB HBM2 @ 1.6 TB/s	Up To 40 GB HBM2 @ 1.6 TB/s Up To 80 GB HBM2 @ 2.0 TB/s	16 GB HBM2 @ 1134 GB/s	16 GB HBM2 @ 900 G	16 GB HBM2 @ 732	16 GB HBM2 (732 GB
TDP	700W	700W	700W	350W	400W	250W	250W	3		



CNNs are everywhere

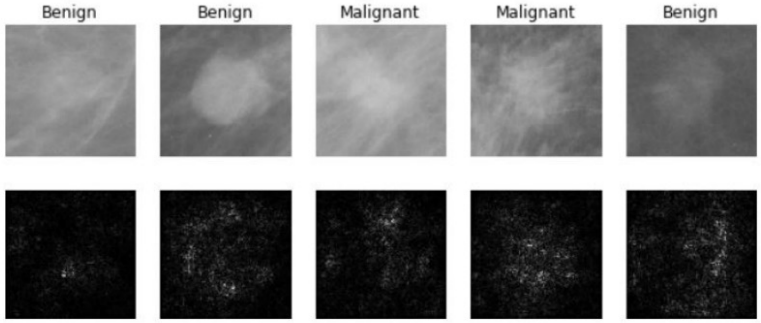
Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

CNNs are everywhere



[Levy et al. 2016]

Figure copyright Levy et al. 2016.
Reproduced with permission.

[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

[Sermanet et al. 2011]
[Ciresan et al.]

Photos by Lane McIntosh.
Copyright CS231n 2017.

CNNs are everywhere

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

Mnih and Hinton, 2010

CNNs are everywhere

No errors

A white teddy bear sitting in the grass

Minor errors

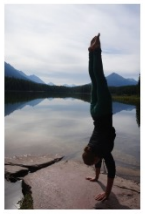
A man in a baseball uniform throwing a ball

Somewhat related

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor



A woman standing on a beach holding a surfboard

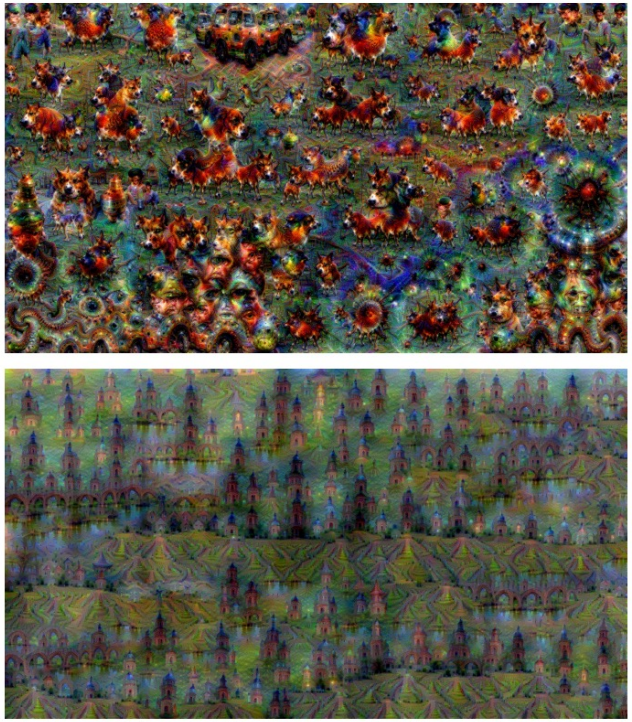
Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

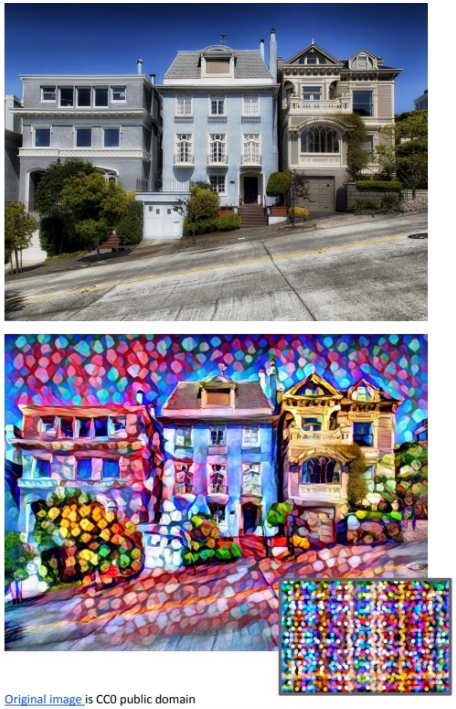
All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

Captions generated by Justin Johnson using Neuraltalk2

CNNs are everywhere



Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.



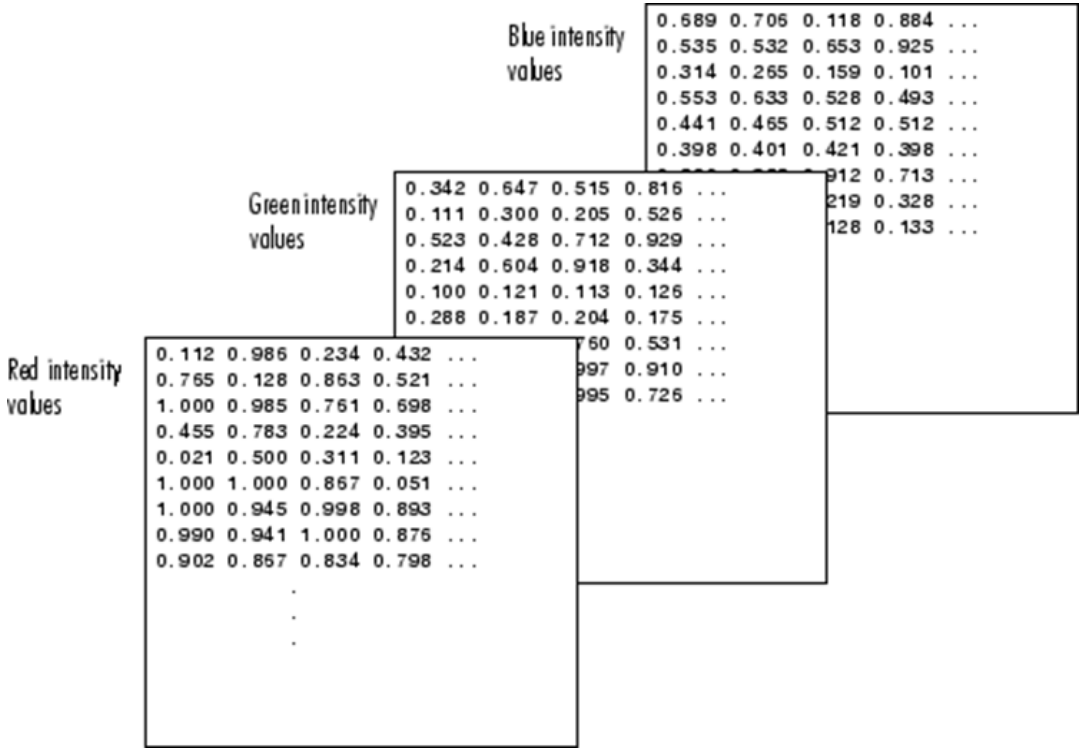
Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

What does an image look like?

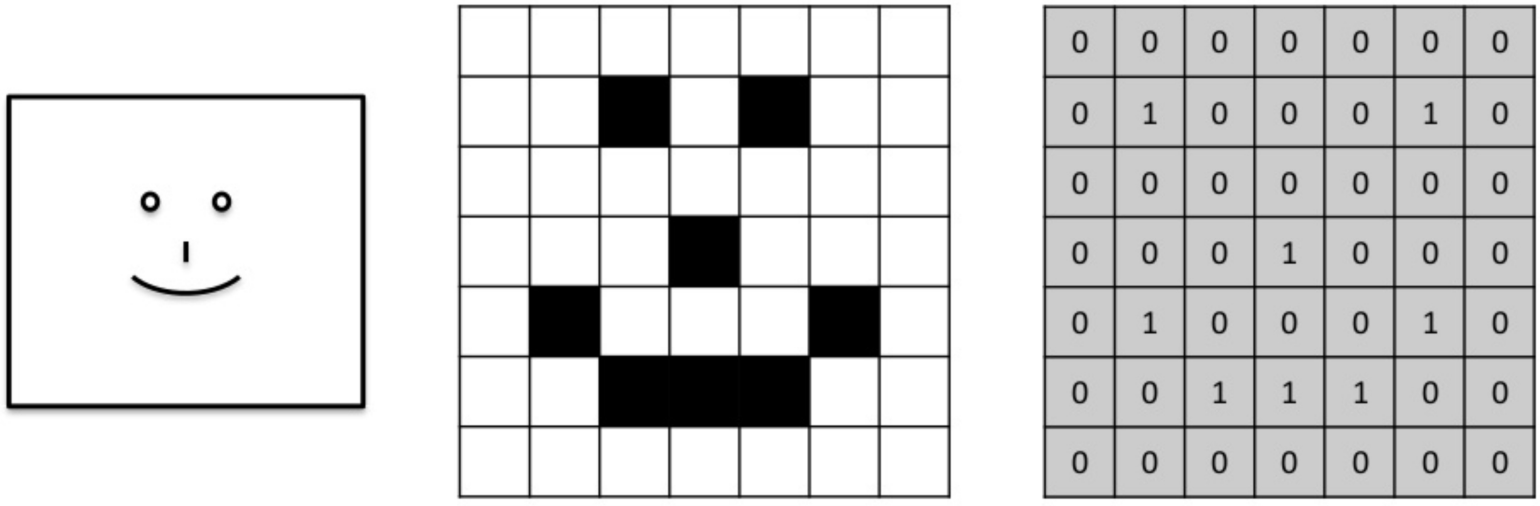
- Matrix with 3 layers (RGB)
- 256 colors

Original image RGB channels



https://blog.datawow.io/interns-explain-cnn-8a669d053f8b

Simple black and white image



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

2D matrix
(no grayscale)

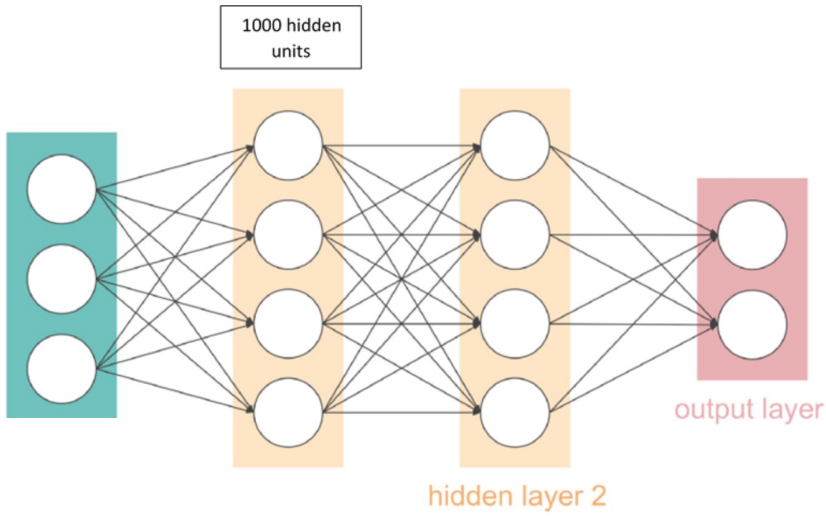
Before convolution

- Original values of a **24-bit color** images (True Color):
  - 8-bit per color: 0 – 255.
  - Total: 256 * 256 * 256 = 16,777,216 colors
- Value for red, green, and blue.
- Preprocessing color values: **normalized** between 0 and 1
  -> will increase performance

How does convolution work?

Problems with NN and images

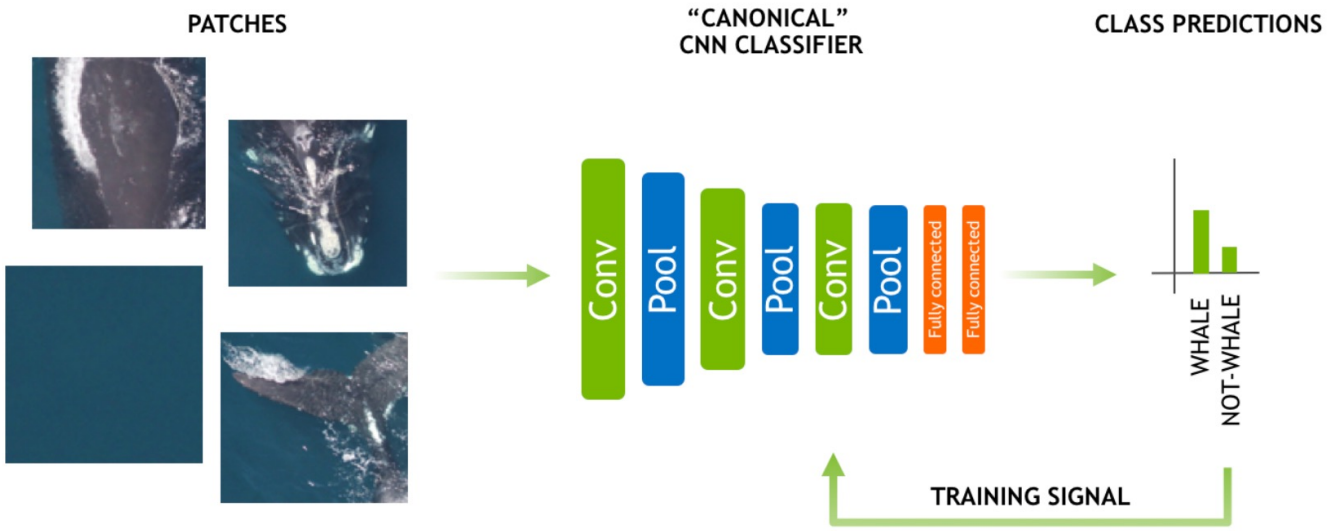
- A color image with size 300 x 300 would have 300 x 300 x 3 input values which is equal to 270,000 inputs. If, for example, we have 1,000 hidden units in our first hidden layer, there would be approximately *270 million parameters* or weights for us to train which is infeasible.
  - High chance of overfitting and highly complex network



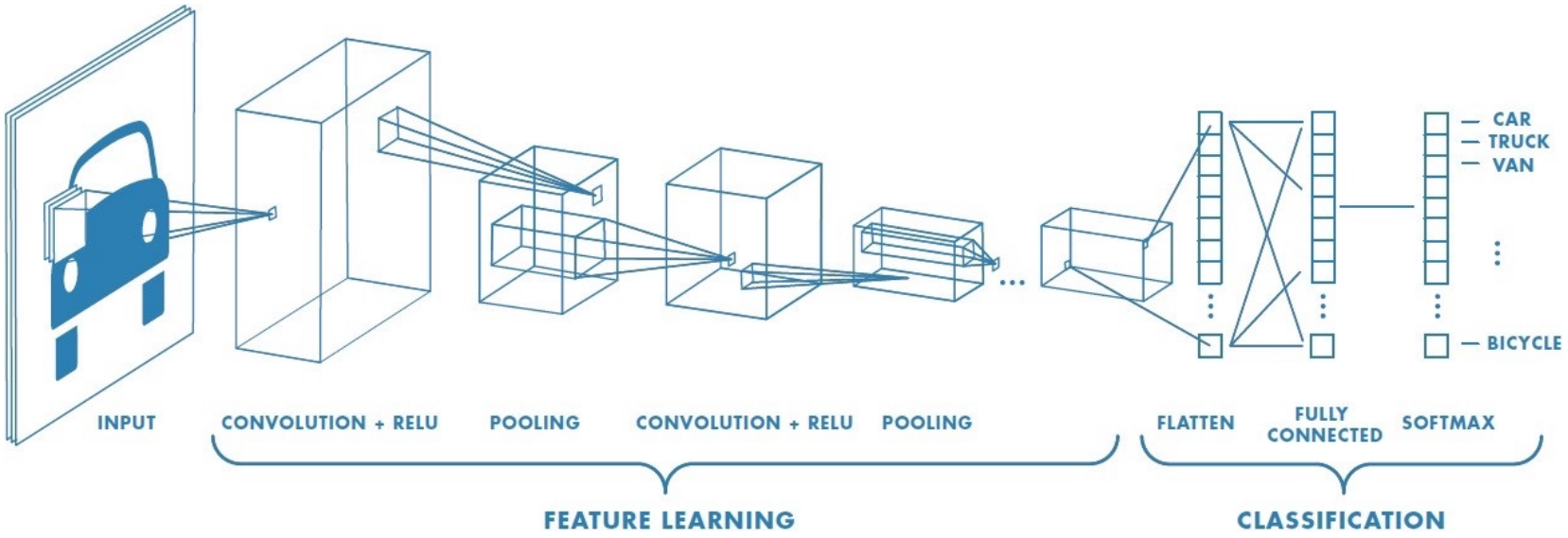
https://blog.datawow.io/interns-explain-cnn-8a669d053f8b

Solution: convolution

- Reduces the number of parameters we need to learn.
- Preserves locality. We don't have to flatten the image matrix into a vector, thus the relative positions of the image pixels are preserved.



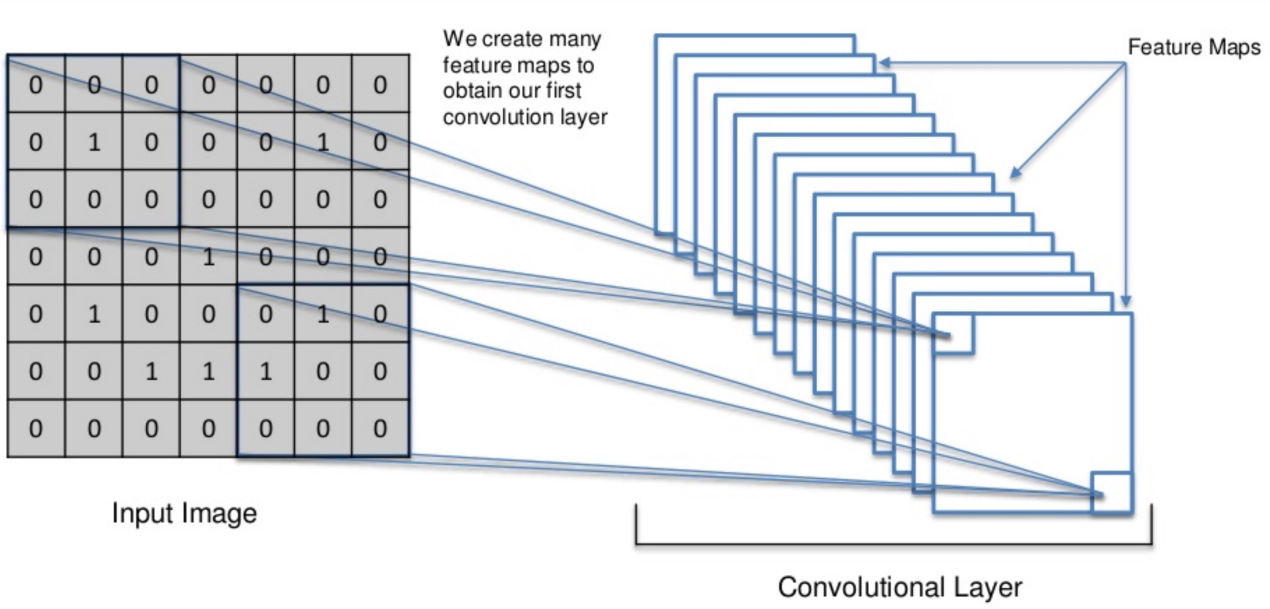
What goes on in a typical CNN?



https://blog.datawow.io/interns-explain-cnn-8a669d053f8b

Convolutional layer

- Many feature maps are created, using filters (also called kernels).
- Kernels (or filters) are learned to best fit the task at hand.



Convolution

- For a 2-D image *H* and a 2-D kernel *F*:
- Convolution Operator: *G = H * F*

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

-> continuous version would be: integral of two functions after one is reversed and shifted. In CNNs, we often do not flip the kernel, hence *we actually calculate* ***cross-correlation*** *instead of convolution*:

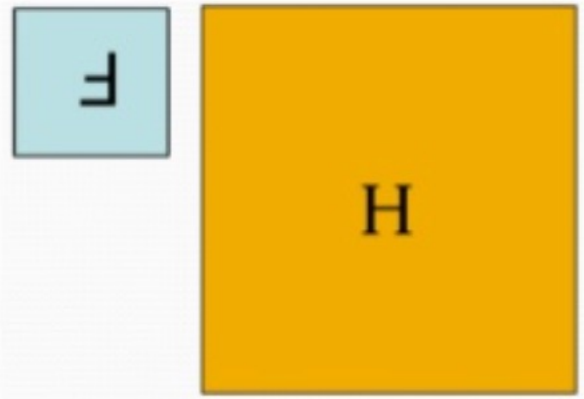
- Correlation Operator: $G = H \otimes F$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v]F[i+u, j+v]$$

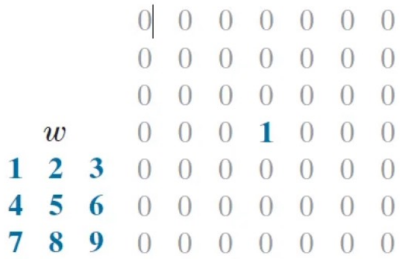
- *u* and *v* represent the position in the filter (given size k), *i* and *j* the positions in the resulting activation map

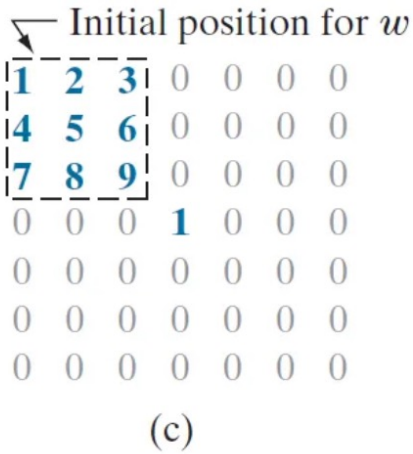
Convolution

- Convolution is equivalent to flipping the filter in both dimensions (bottom – top and right – left).
- For symmetric kernels, both result in the same output
- Many machine learning libraries implement cross-correlation, but call it convolution! We will also do so.
- If you flip the kernel in a neural network or not is just a computational operation, the kernel contents is learned anyway, so we may as well do the cheaper operation cross-correlation

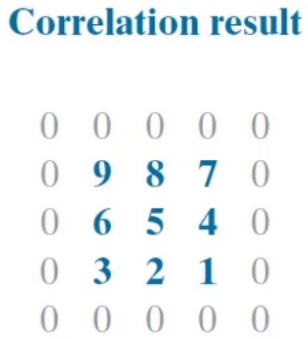


Convolution vs cross correlation

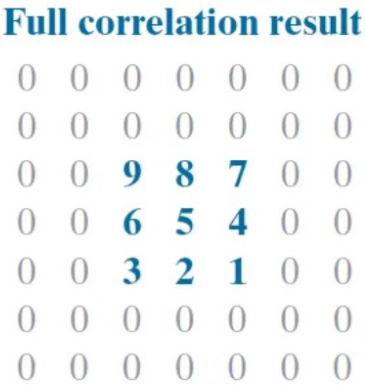




(c)

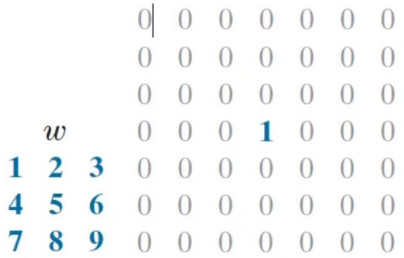


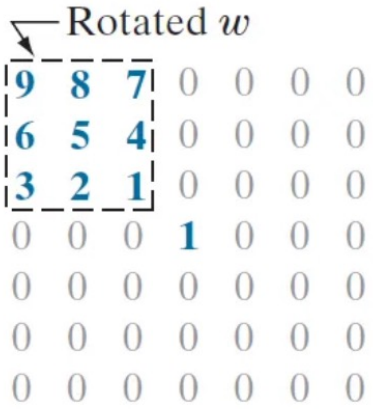
(d)



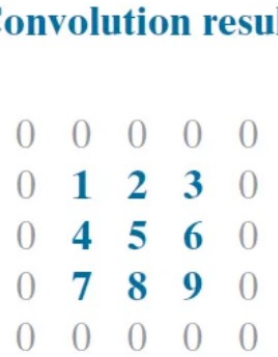
(e)

Convolution vs cross correlation

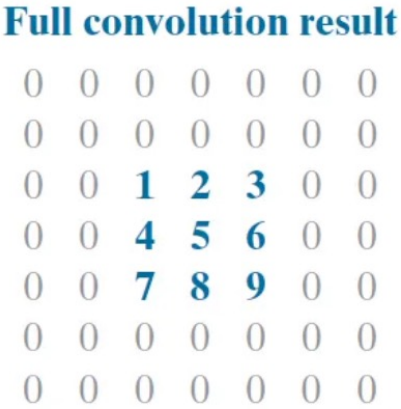




(f)

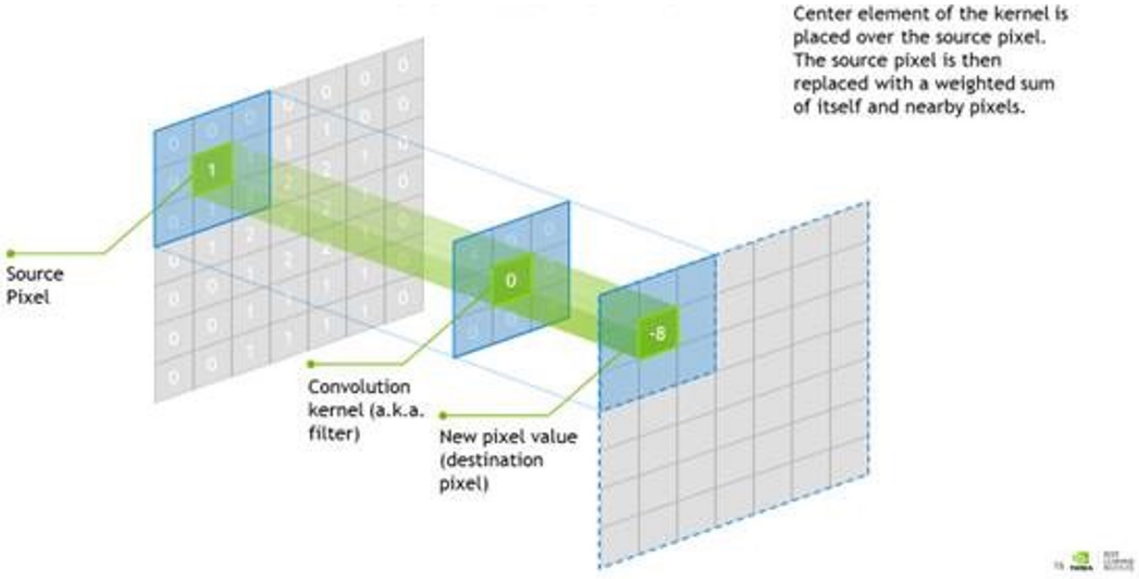


(g)



(h)

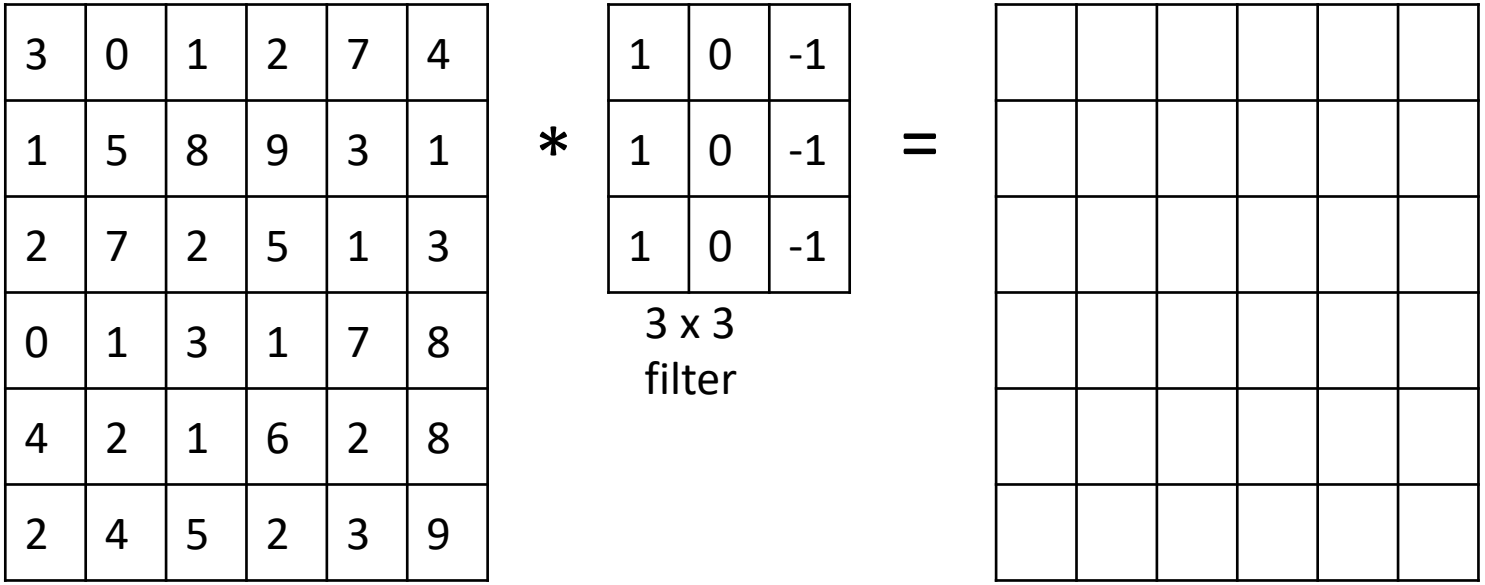
Convolution



[Image source]

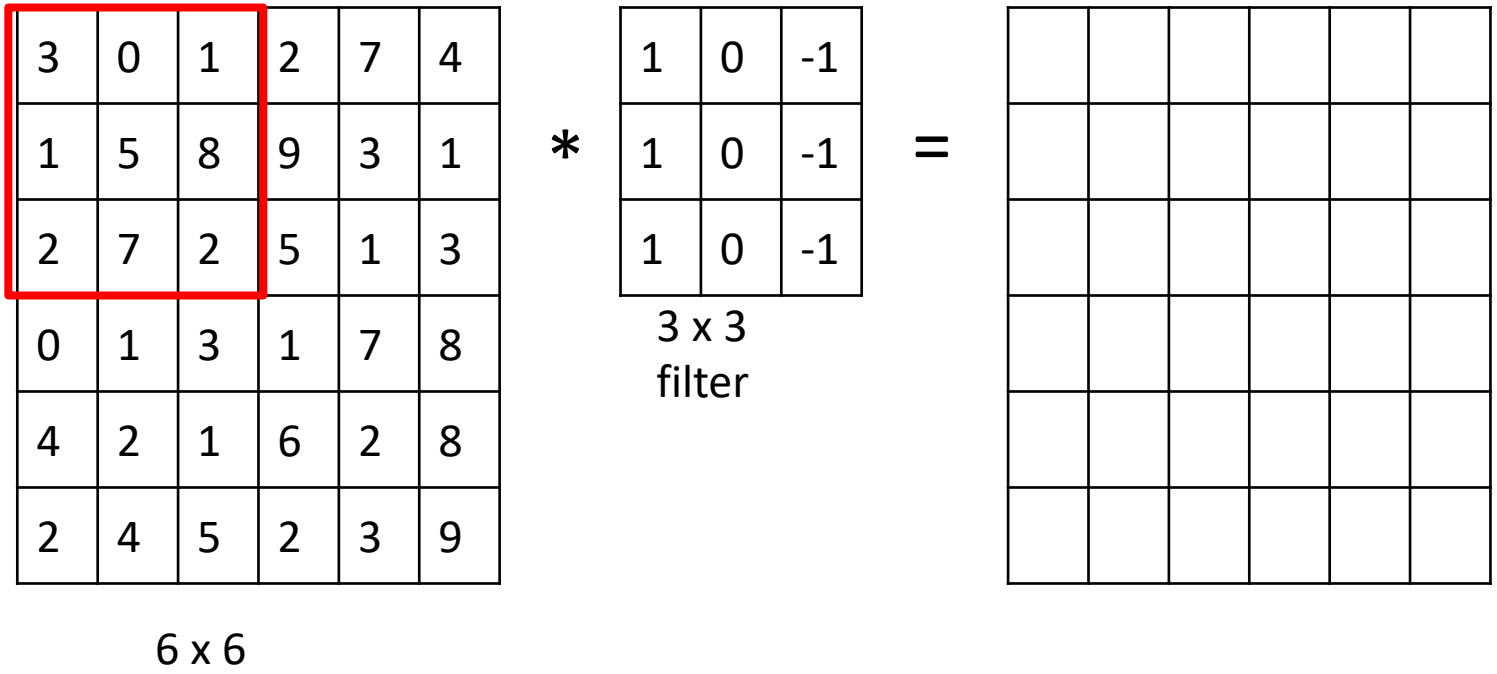
Convolution (as implemented in ML)

- Example: edge detection filter/kernel



Convolution

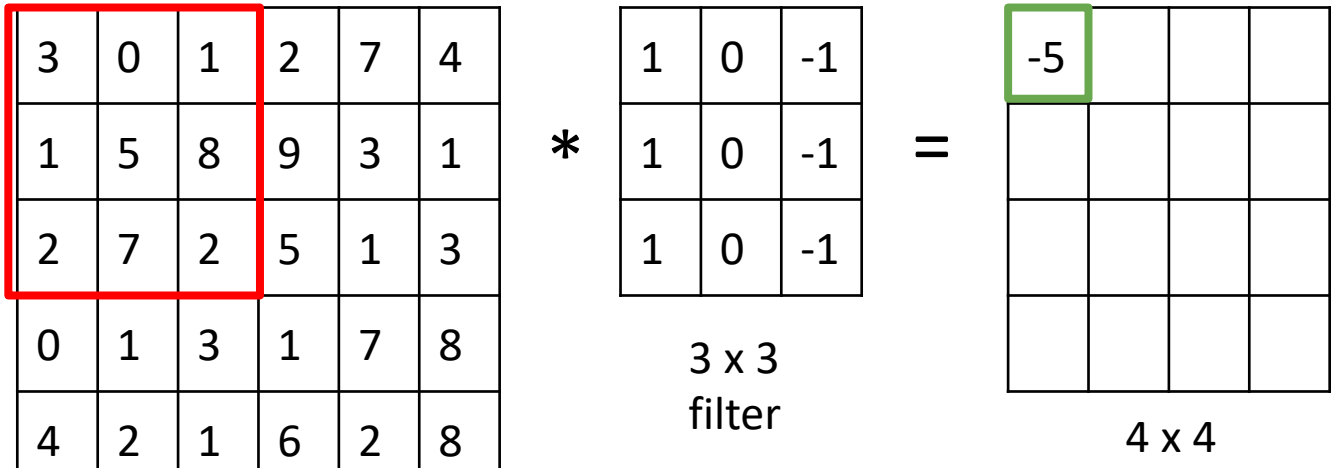
- Example: edge detection filter/kernel



Convolution

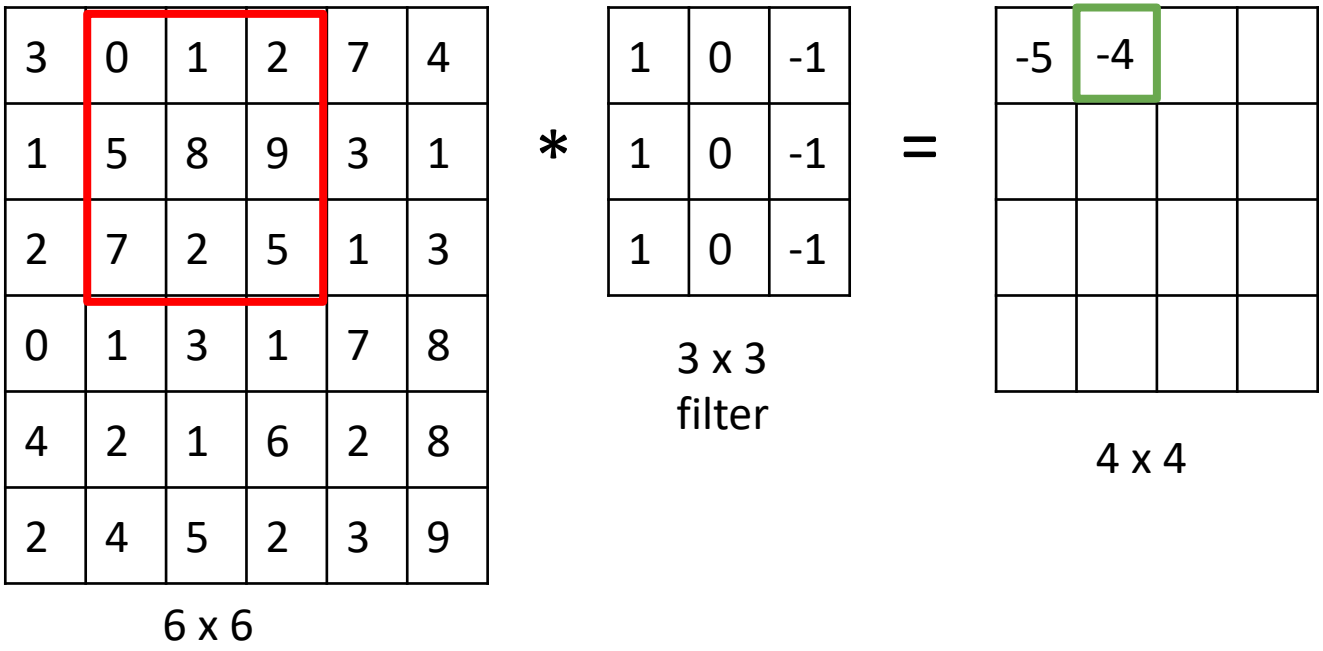
- Example: edge detection filter/kernel

= 3x1 + 1x1 + 2x1 + 0x0 + 5x0 + 7x0 + 1x(-1) + 8x(-1) + 2x(-1)



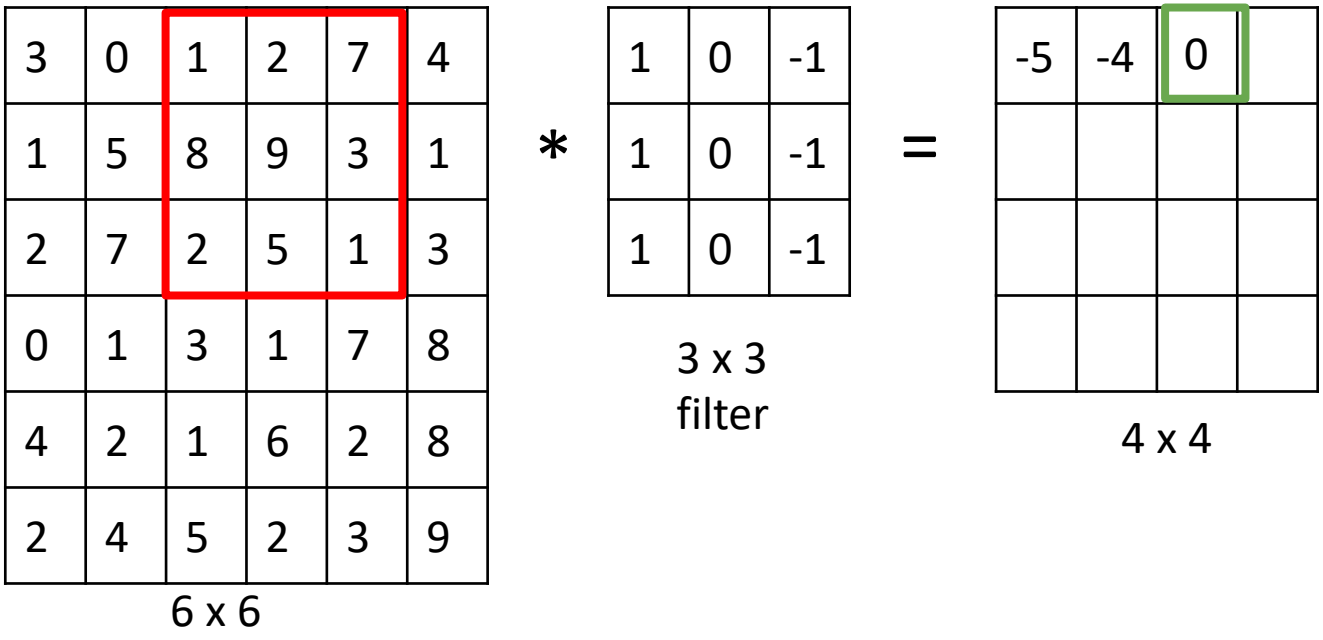
Convolution

- Example: edge detection filter/kernel, skip size.= 1



Convolution

- Example: edge detection filter/kernel



Convolution

- Example: edge detection filter/kernel

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

6 x 6

*

1	0	-1
1	0	-1
1	0	-1

3 x 3
filter

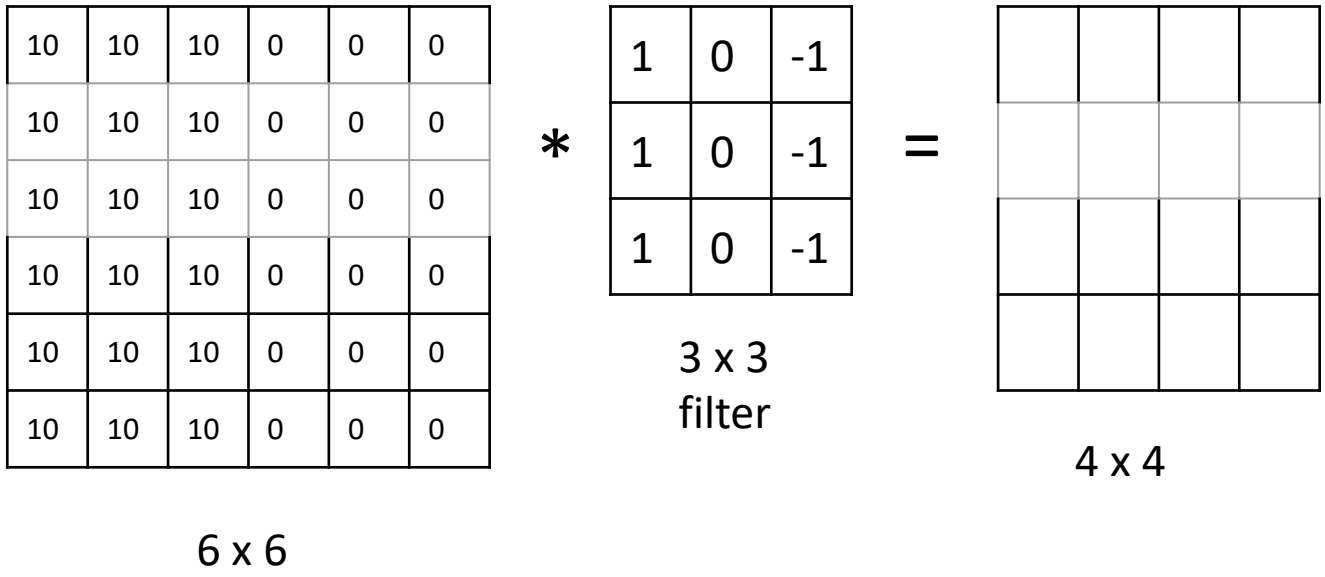
=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

4 x 4

Convolution

- Example: edge detection filter/kernel



Convolution

- Example: edge detection filter/kernel

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

*

1	0	-1
1	0	-1
1	0	-1

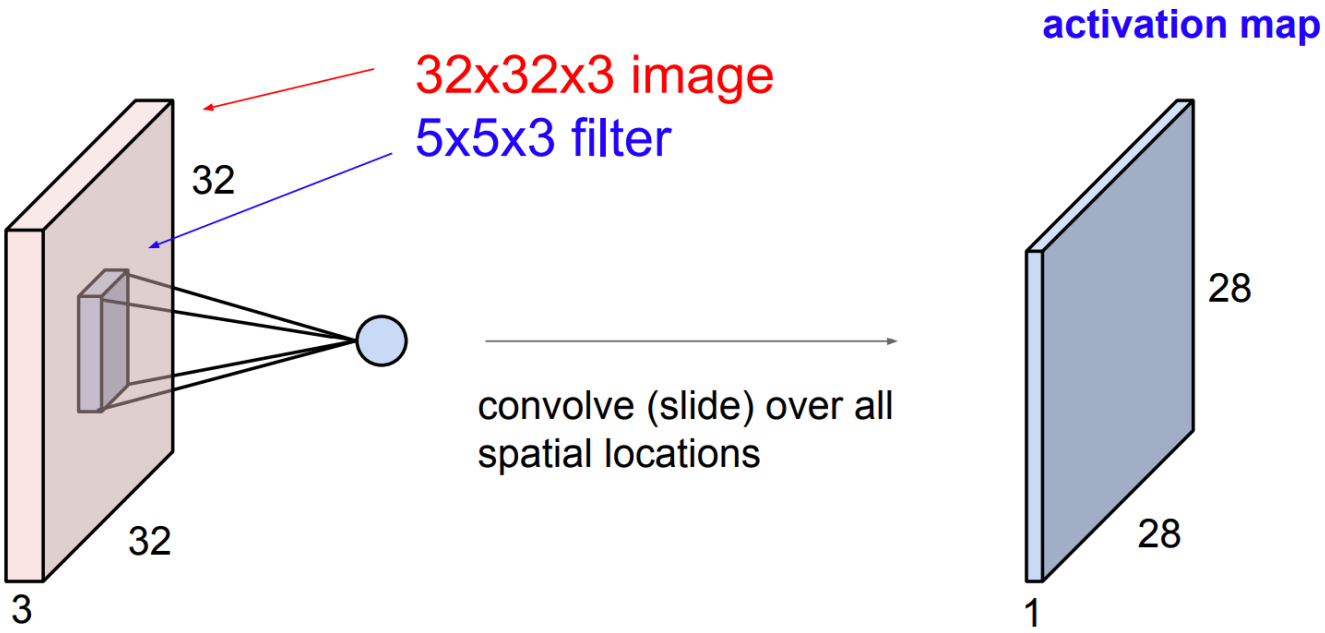
3 x 3 filter

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

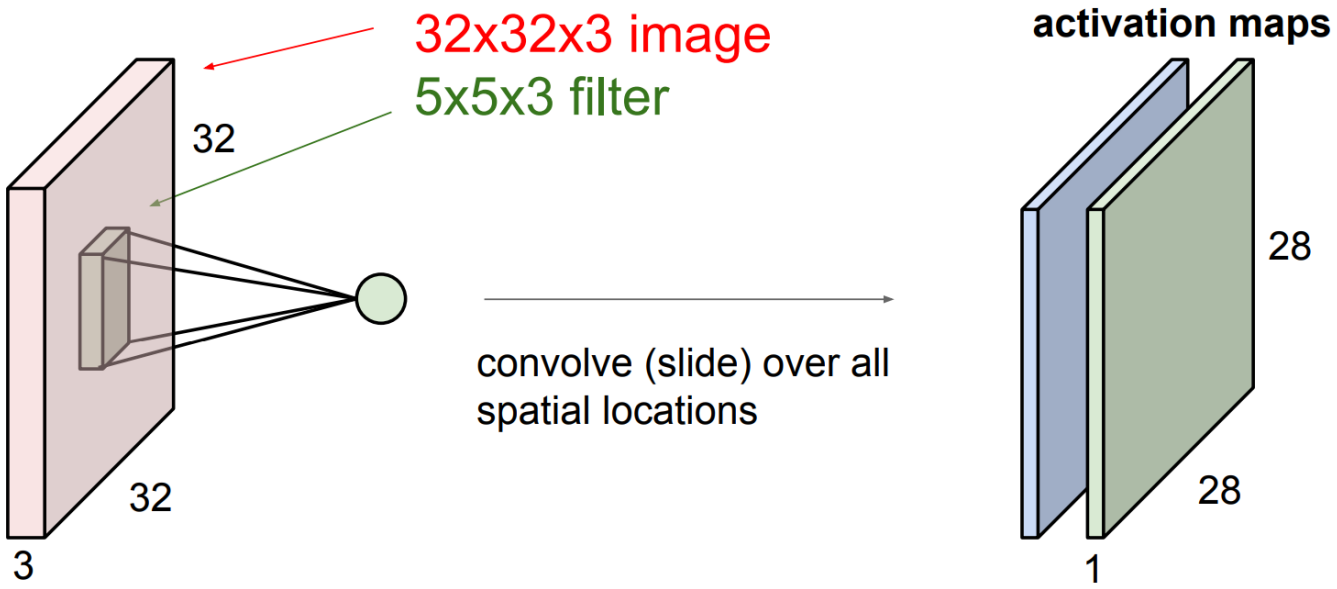
4 x 4

Activation maps



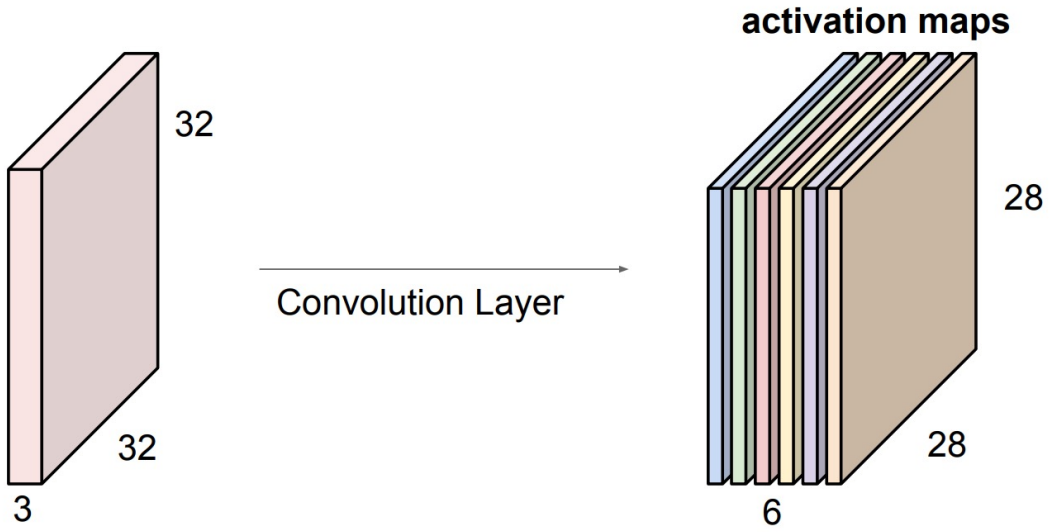
Activation maps

- Each filter creates an activatio map



Activation maps

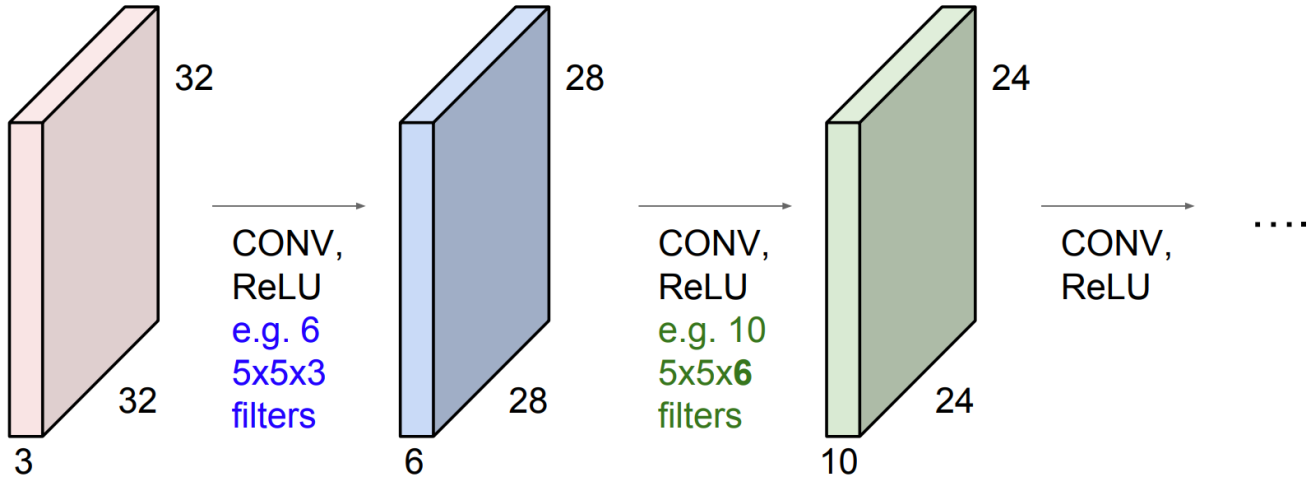
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

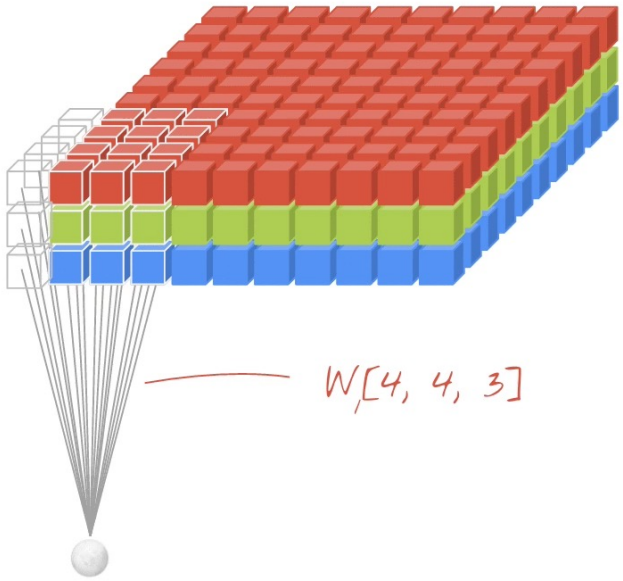
ConvNet

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Dimensionality of filters

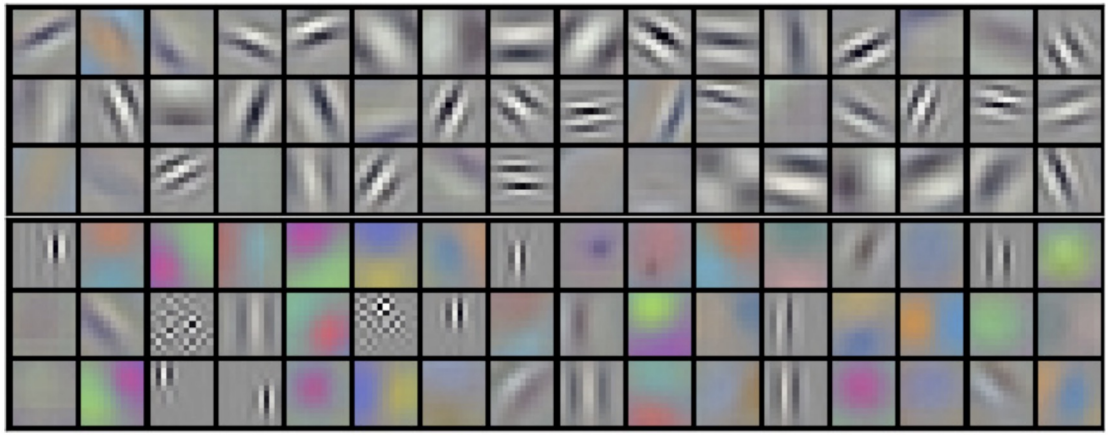
- Color images: filters have a third dimension.
- The resulting feature map is 2-dimensional for each filter.



https://ifding.github.io/images/2018/1/2d-conv-3d-input.gif

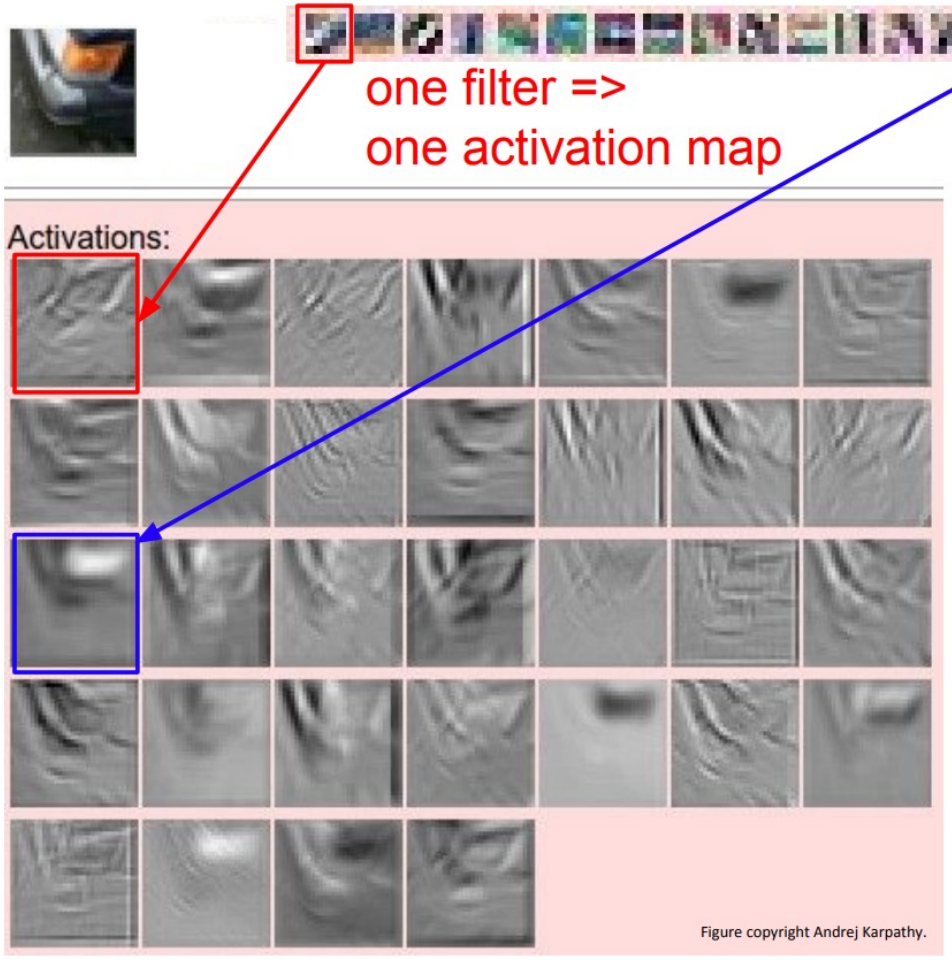
Convolution

- Typical filters that are learned:



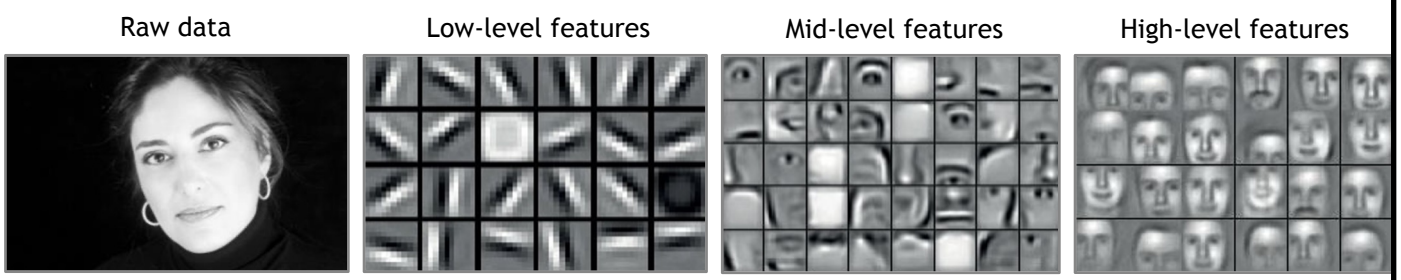
(Krizhevsky et al., 2012)

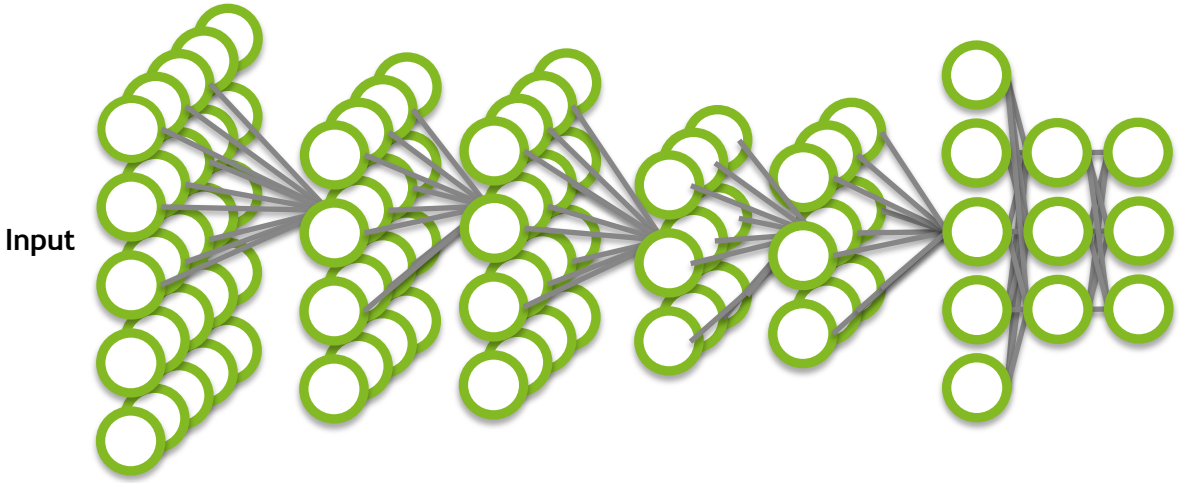
http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo



example 5x5 filters
(32 total)

Different levels of features





Application components:

Task objective
e.g. Identify face

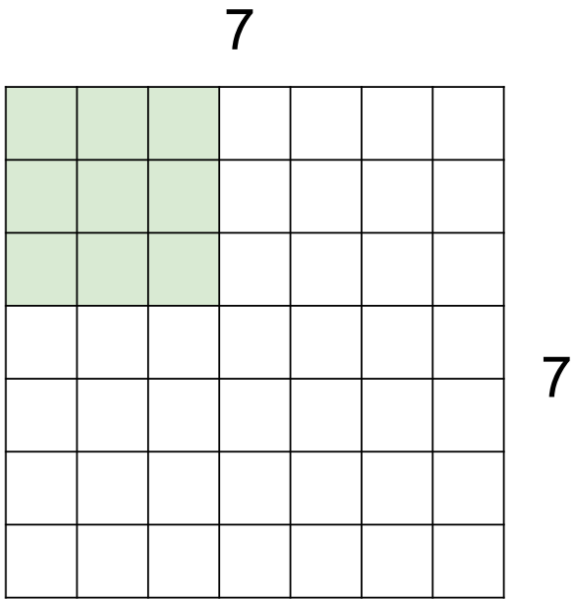
Training data
10-100M images

Network architecture
~10s-100s of layers
1B parameters

Learning algorithm
~30 Exaflops
1-30 GPU days

A closer look at dimensions

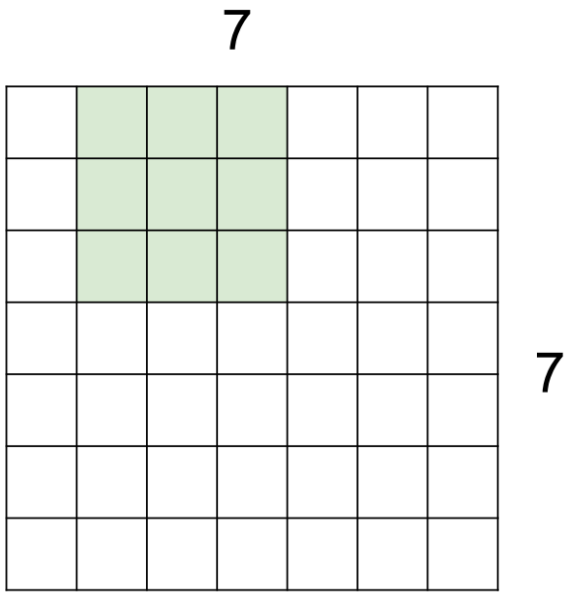
- A 7x7 input with a 3x3 filter:



A closer look at dimensions

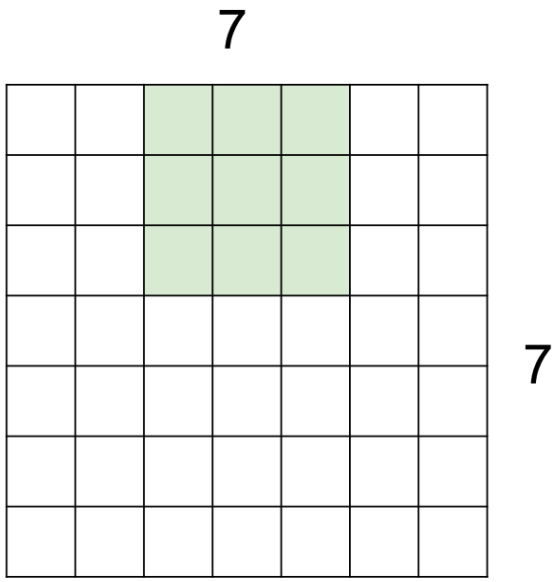
- A 7x7 input with a 3x3 filter:

Stride = 1



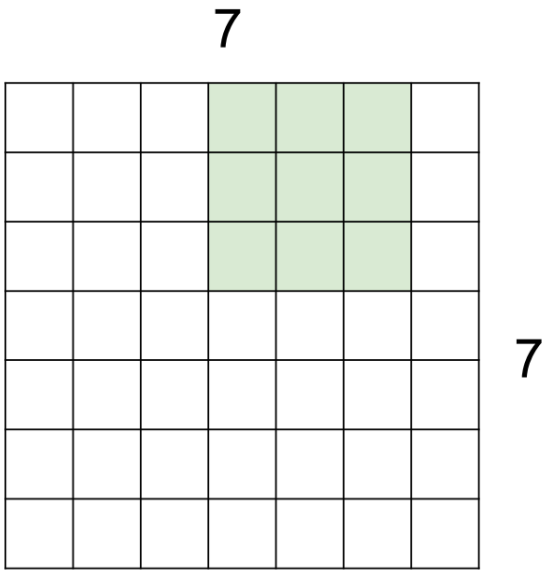
A closer look at dimensions

- A 7x7 input with a 3x3 filter:



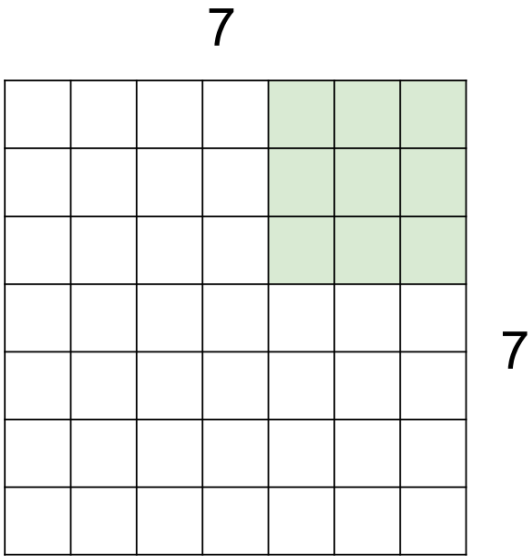
A closer look at dimensions

- A 7x7 input with a 3x3 filter:



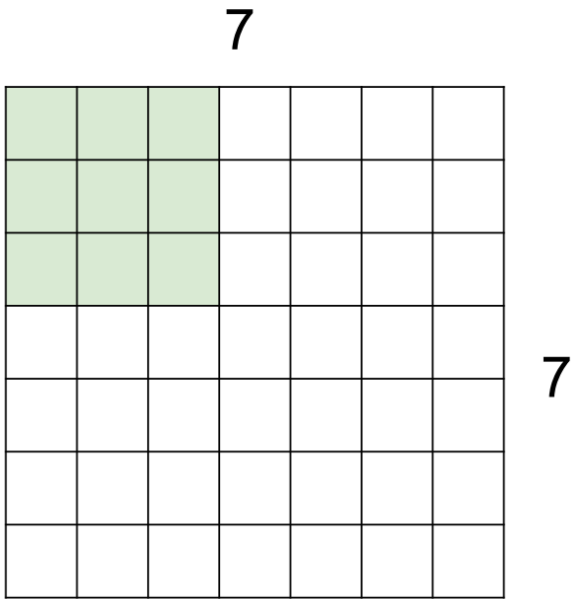
A closer look at dimensions

- A 7x7 input with a 3x3 filter: **5x5 output**



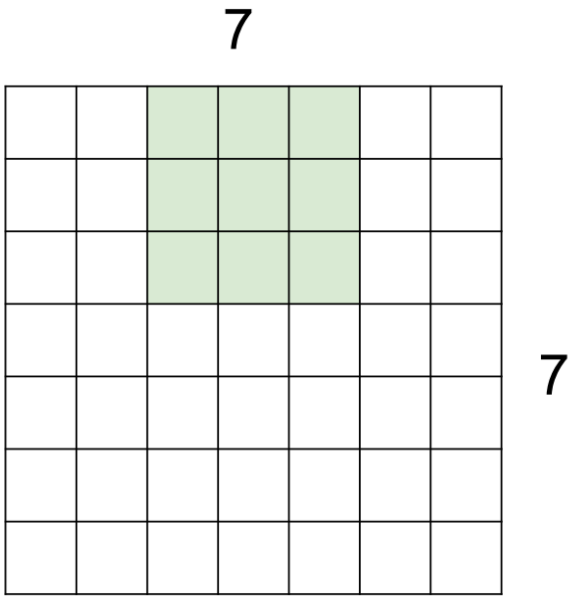
A closer look at dimensions

- A 7x7 input **with stride 2** and a 3x3 filter:



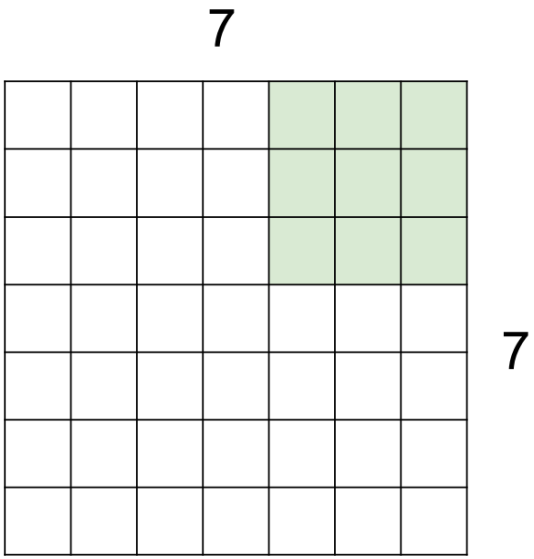
A closer look at dimensions

- A 7x7 input **with stride 2** and a 3x3 filter:



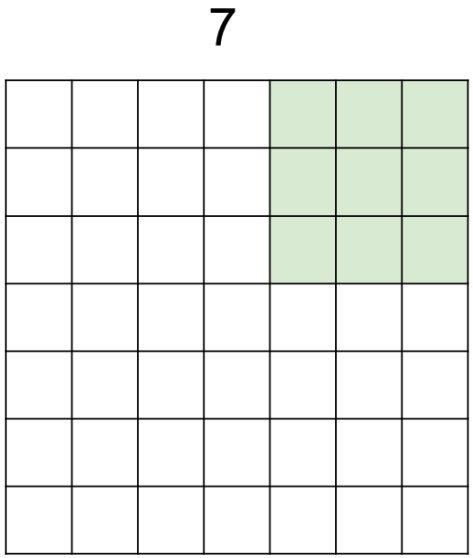
A closer look at dimensions

- A 7x7 input **with stride 2** and a 3x3 filter: **3x3 output**



A closer look at dimensions

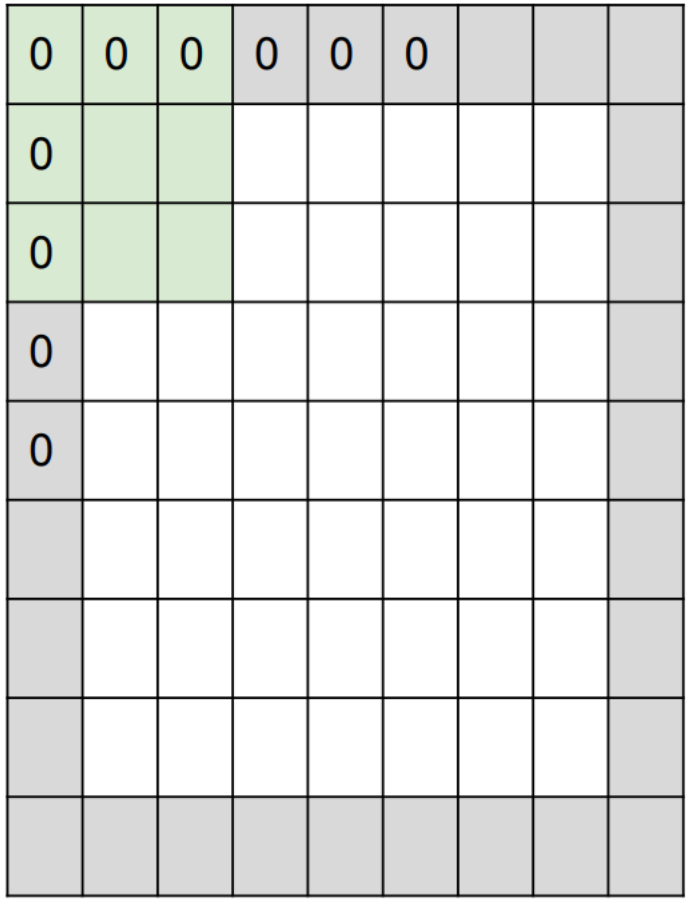
- A 7x7 input **with stride 3??** What is the output size?
- [(N - F) / stride] + 1



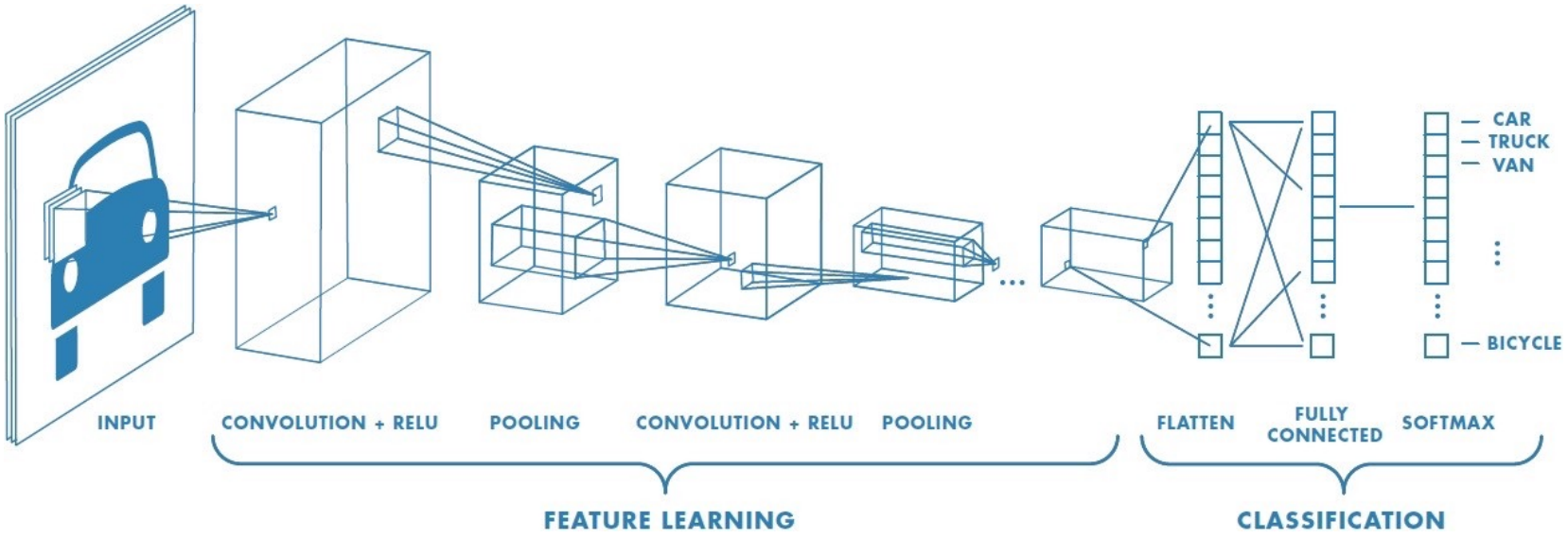
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

Padding

- e.g. input 7x7 3x3 filter,
- Applied with stride 1 pad with 1 pixel border => what is the output?
- 7x7 output!
- In general, common to see conv. layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)



After convolution

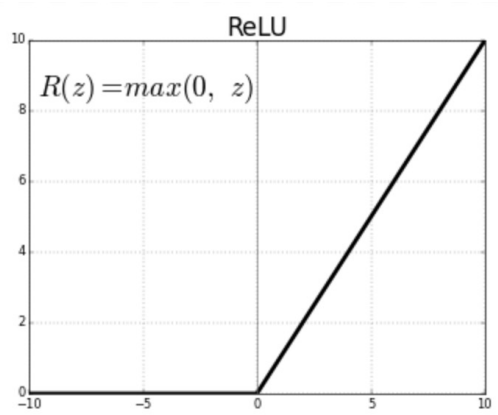


Non-linear activation

- Tanh, sigmoid, ReLu, or Leaky ReLU activation function
- Most popular (often performs best): **ReLu**

  => **to introduce non-linearity** in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear.
- After each conv layer, it is convention to apply a non-linear layer (or activation layer) immediately afterwards.

$$f(x) = \max(0, x)$$

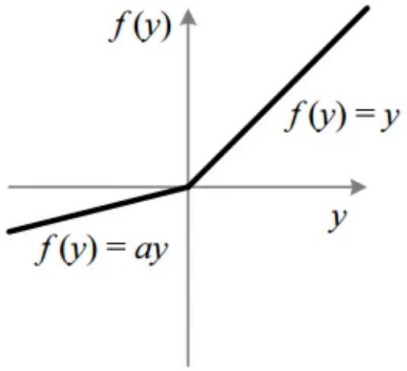


Leaky ReLu

- ReLu: issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly.
  => any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.
- The leak helps to increase the range of the ReLU function. Usually, the value of a is 0.01 or so.
- When a is not 0.01 then it is called Parametric ReLU.

$$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

with parameter α

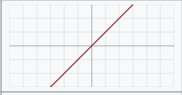
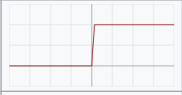
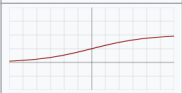
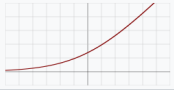


Activation functions

- See full table here: https://en.wikipedia.org/wiki/Activation_function

Table of activation functions [edit]

The following table compares the properties of several activation functions that are functions of one fold x from the previous layer or layers:

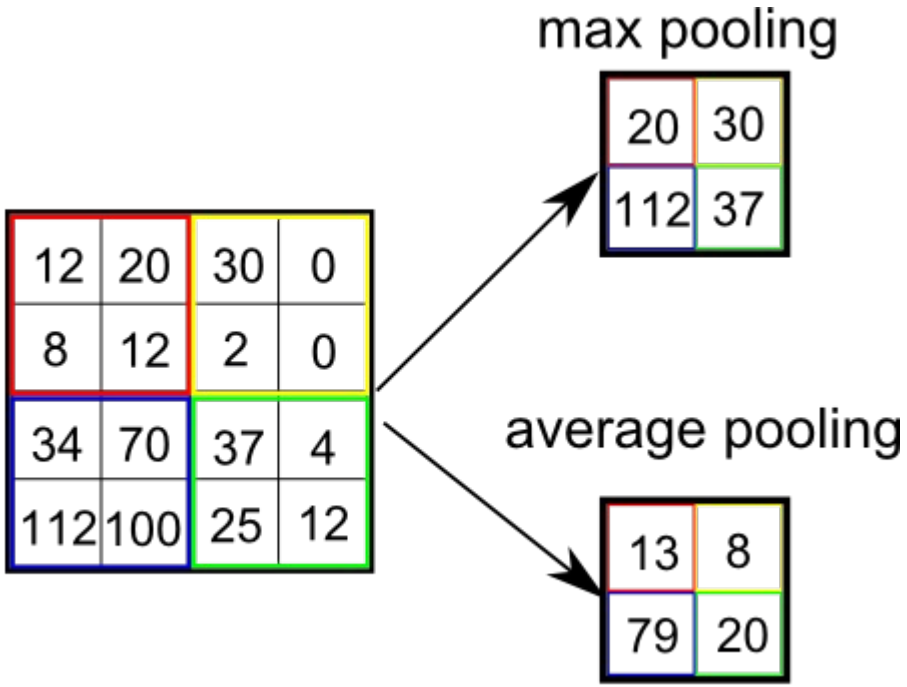
Name	Plot	Function, $g(x)$	Derivative of g, $g'(x)$	Range	Order of continuity
Identity		x	1	$(-\infty, \infty)$	C^∞
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	0	$\{0, 1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$	$g(x)(1-g(x))$	$(0, 1)$	C^∞
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - g(x)^2$	$(-1, 1)$	C^∞
Rectified linear unit (ReLU)[8]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$	C^0
Gaussian Error Linear Unit (GELU)[5]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = x\Phi(x)$	$\Phi(x) + x\phi(x)$	$(-0.17\ldots, \infty)$	C^∞
Softplus[9]		$\ln(1 + e^x)$	$\frac{1}{1+e^{-x}}$	$(0, \infty)$	C^∞

Pooling

- Non-linear down-sampling to simplify the output of the convolutional layer.
- ConvNets often use pooling layers to **reduce the size of the representation**, **speed up** the computation, as well as make some of the detected features a bit more **robust**.
- Types of pooling:
  - Max pooling (popular)
  - Average pooling
- Typical shape: 2x2 or sometimes 4x4
- Too large window: dramatic loss of information
- Non-overlapping windows perform the best

Pooling

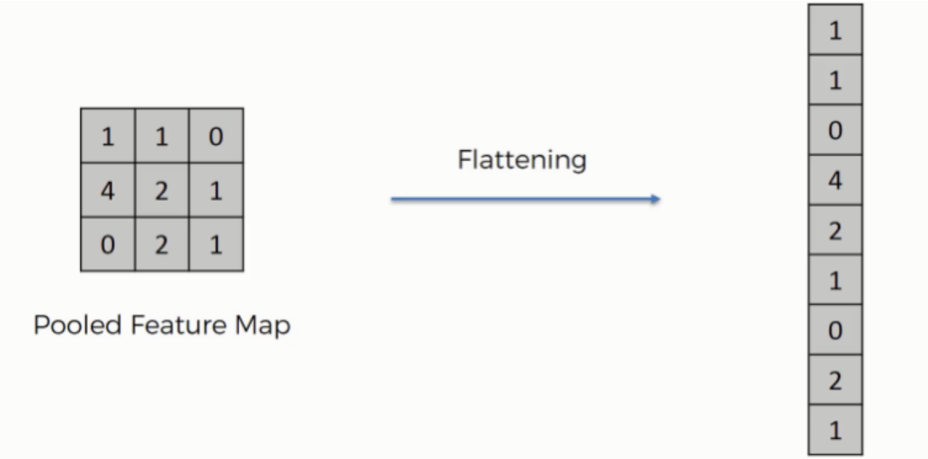
- Hyperparameters:
  - Stride size
  - Pooling window size



- Pooling layers don't learn themselves, they just reduce the size of the problem

Image: https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117

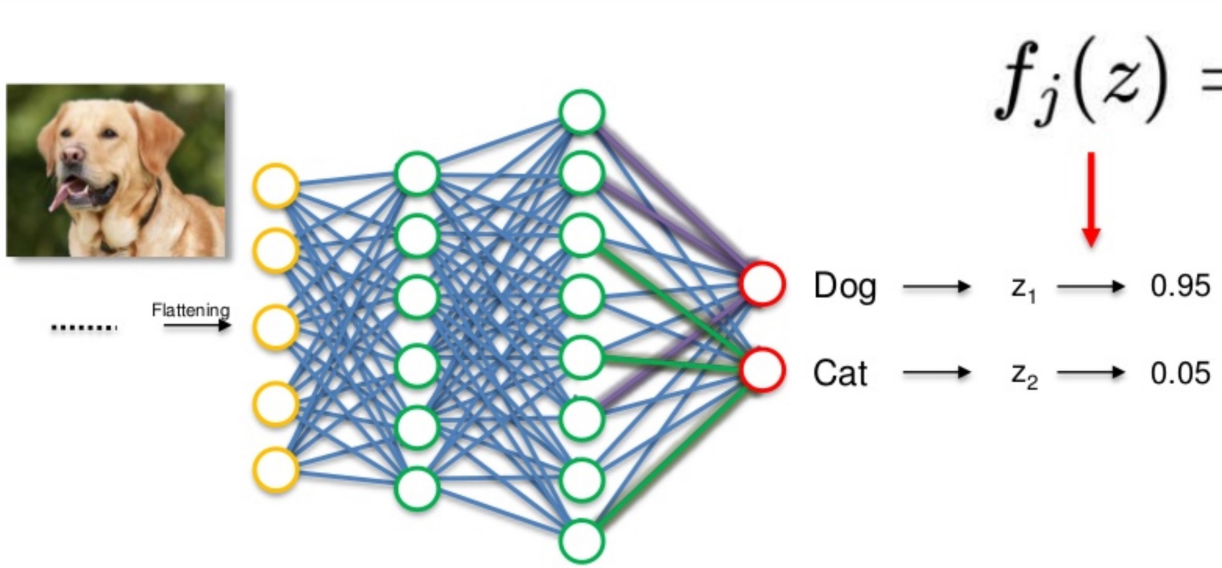
Flatten



Fully connected layer

- Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have **connections to all activations in the previous layer**, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset
- **Training Loss**: how training penalizes the deviation between the predicted and true labels and is normally the final layer. Various loss functions appropriate for different tasks may be used there:
  - **Softmax loss** (a Softmax activation plus a Cross-Entropy loss**)** is used for multiclass classification (it distributes the probability throughout each output node, meaning that the sum of all probabilities is 1)
  - **Sigmoid cross-entropy loss** (Sigmoid activation plus a Cross-Entropy loss**)** is used for binary classification
  - **Euclidean loss** is used for regressing to real-values. (could be mean squared error: mse)

Softmax



PyTorch for CNNs

- Images need to be fed into the network with the same image width & height.
- Dataloaders can be very useful to feed the images efficiently in batches by sampling from a dataset.
- We can create a Dataset from a directory using *ImageFolder* and apply a *transform* operation to it, which do anything from resizing to image augmentation

```
mytransforms = torchvision.transforms.Compose([transforms.Resize((img_height,img_width)),
                                               transforms.ToTensor()])
train_dataset = torchvision.datasets.ImageFolder(train_data_dir, transform=mytransforms)
```

```
batch_size=4
train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,shuffle=True)
```

PyTorch for CNNs

- Sequential model → the layers are stacked.
- Each CNN layer typically includes: a convolution layer (Conv2d), an activation function (ReLU), and a pooling layer (MaxPool2d)
  - The Conv2d arguments are (# input channels, # output channels = how many filters, filter size)

```
import torch.nn as nn
model = nn.Sequential(
    nn.Conv2d(3,32,3),
    nn.ReLU(),
    nn.MaxPool2d(2,2)
)
```

- We can expand our model with more layers like this:

```
layers=list(model.children())
layers.extend([nn.Conv2d(32,64,3),
               nn.ReLU(),
               nn.MaxPool2d(2,2)])
model=nn.Sequential(*layers)
```

PyTorch for CNNs

- The last layers in our model are fully connected layers (FC). In PyTorch we will have to calculate the output size of the last convolutional layer to define the input size of the first FC layer (after the flatten layer):

```
def calcsize(model,channum,imh,imw):
  image=torch.rand(channum,imh,imw)
  image=image[None,:]
  output=model(image)
  return output.shape[1:]

model=nn.Sequential(nn.Conv2d(3, 32, 3),
        nn.ReLU(),
        nn.MaxPool2d(2, 2),
        nn.Conv2d(32, 32, 3),
        nn.ReLU(),
        nn.MaxPool2d(2, 2),
        nn.Conv2d(32, 64, 3),
        nn.ReLU(),
        nn.MaxPool2d(2, 2)
        )
imh=150
imw=150
channum=3

output_size=calcsize(model,channum,imh,imw)
```

After calculating the conv output size, we can initialise the fully connected part of the net like this:

```
layers = list(model.children())
layers.extend([nn.Flatten(),
                nn.Linear(output_size[0]*output_size[1]*output_size[2], 64),
                nn.Dropout(0.5),
                nn.Linear(64,1),
                nn.Sigmoid()
                ])

model=nn.Sequential(*layers).to(device)
```

Sidenote: these code snippets are taken from the lab exercise, so you will get to see this again! ☺

PyTorch for CNNs

- The final activation, tailored to ***classification*** (or regression with *mse*):

```
optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0001)
criterion = torch.nn.BCELoss()
summary(model,(3,img_height,img_width)) # using torchsummary
```

- Torchsummary can give us this useful summary of the model

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 148, 148]             896
              ReLU-2         [-1, 32, 148, 148]               0
         MaxPool2d-3           [-1, 32, 74, 74]               0
            Conv2d-4           [-1, 32, 72, 72]           9,248
              ReLU-5           [-1, 32, 72, 72]               0
         MaxPool2d-6           [-1, 32, 36, 36]               0
            Conv2d-7           [-1, 64, 34, 34]          18,496
              ReLU-8           [-1, 64, 34, 34]               0
         MaxPool2d-9           [-1, 64, 17, 17]               0
           Flatten-10                [-1, 18496]               0
            Linear-11                   [-1, 64]       1,183,808
            Linear-12                    [-1, 1]              65
================================================================
Total params: 1,212,513
Trainable params: 1,212,513
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.26
Forward/backward pass size (MB): 16.29
Params size (MB): 4.63
Estimated Total Size (MB): 21.17
----------------------------------------------------------------
```

PyTorch for CNNs

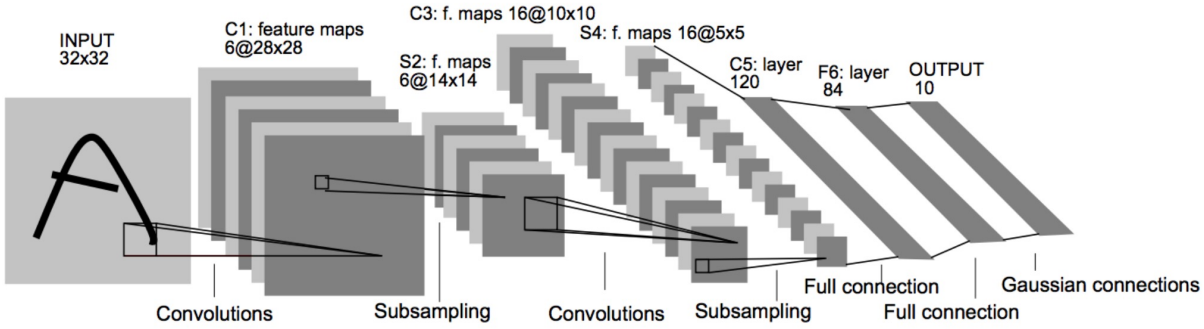
- Storing data during training:
  - here we store the accuracies and losses in each epoch to plot later, full code in lab exercise! :-)

```
ta=[]
va=[]
tl=[]
vl=[]
for epoch in range(15):
  train_acc, val_acc, train_loss, val_loss=train(model,train_dataloader,val_dataloader)
  ta.append(train_acc)
  va.append(val_acc)
  tl.append(train_loss)
  vl.append(val_loss)
```

Famous prebuilt networks

LeNet-5

- Developed in 1998 to identify handwritten digits for zip code recognition in the postal service.

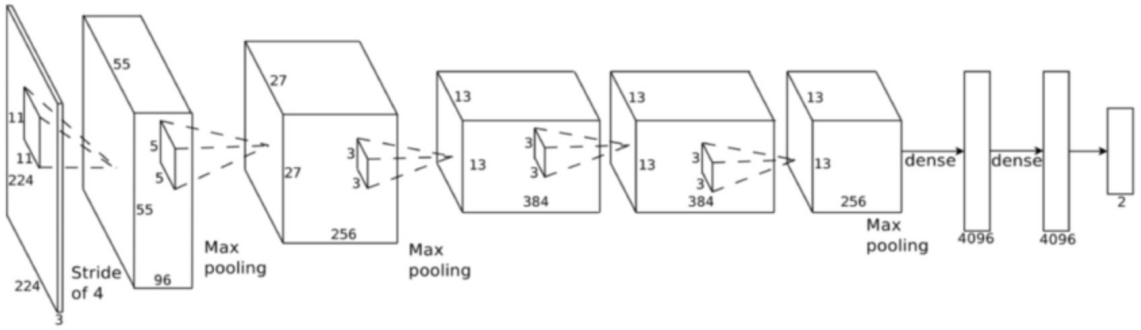


- Convolutional layers use a subset of the previous layer's channels for each filter to reduce computation and force a break of symmetry in the network.
- Subsampling is Average Pooling with learnable weights per feature map.
- Parameters: 60,000

http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

AlexNet

- Alex Krizhevsky et al.: won 2012 ImageNet competition (1.2 million training images. 1000 classes)

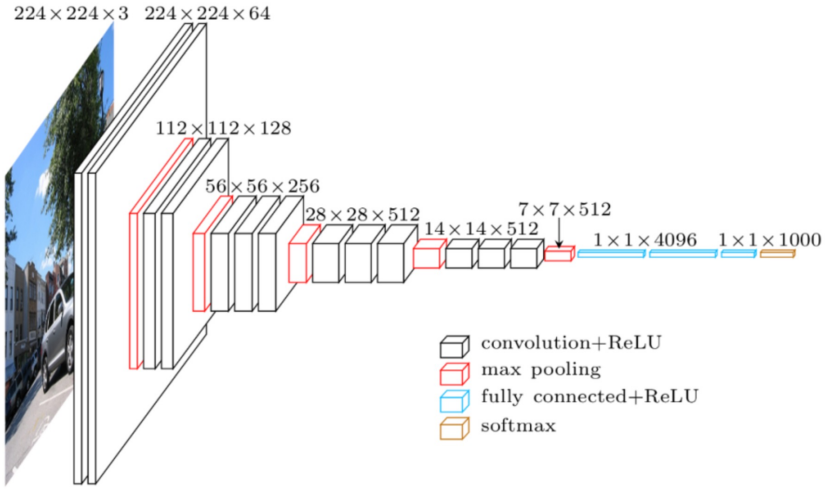


- The general architecture is quite similar to LeNet-5, although this model is considerably **larger**. The success of this model convinced a lot of the computer vision community to take a serious look at deep learning for computer vision tasks.
- Parameters: 60mio (7 hidden layers, 650K units)

https://www.nvidia.cn/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf

VGG-16

- The VGG network, introduced in 2014, offers a deeper yet simpler variant. At the time of its introduction, this model was considered to be very deep.
- 16 Conv layers
- Extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end
- Parameters: 138 million

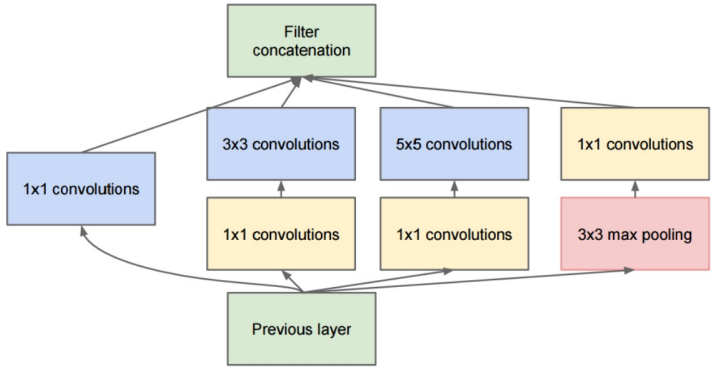


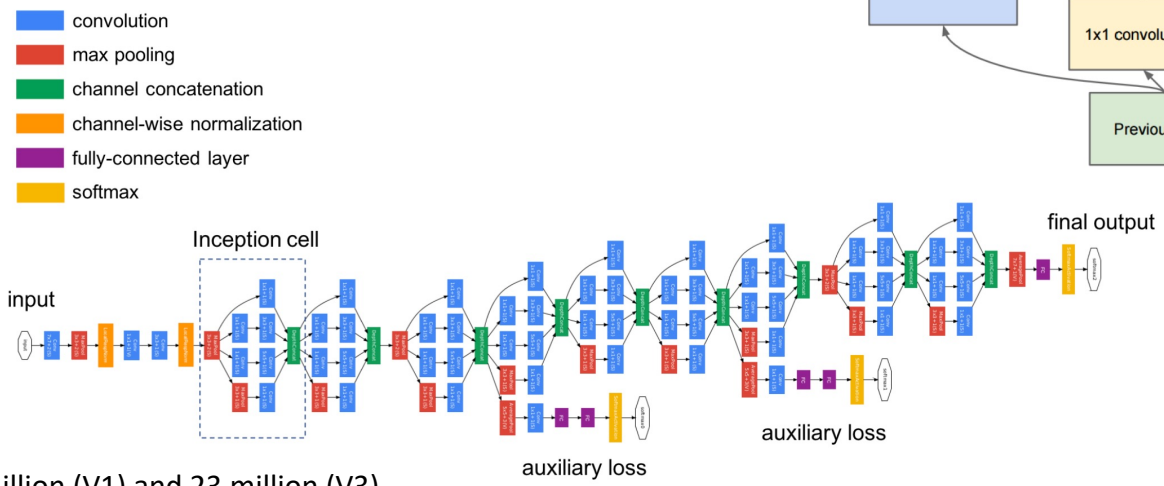

```
model_vgg=torchvision.models.vgg16(pretrained=True)
```

https://arxiv.org/abs/1409.1556

Inception (GoogLeNet)

- Developed in 2014 for ImageNet competition by Google researchers. The model is comprised of a basic unit referred to as an "Inception cell" in which we perform a series of **convolutions at different scales** and subsequently aggregate the results.





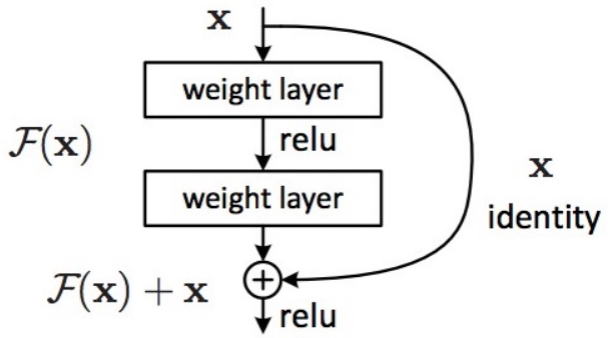
Parameters: 5 million (V1) and 23 million (V3)

https://arxiv.org/abs/1409.4842

ResNet

- Deep residual networks enabled much deeper networks (from 10s to 100s of layers)
- ImageNet competition winner 2015
- More layers = better performance? No, after a while more layers have a negative effect on performance => **degradation problem**:
  - although better parameter initialization techniques and batch normalization allow for deeper networks to converge, they often converge at a higher error rate than their shallower counterparts.
- **Solution: ResNet → residual blocks** in which intermediate layers of a block learn a residual function with reference to the block input.
- You can think of this residual function as a refinement step in which we learn how to adjust the input feature map for higher quality features.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$



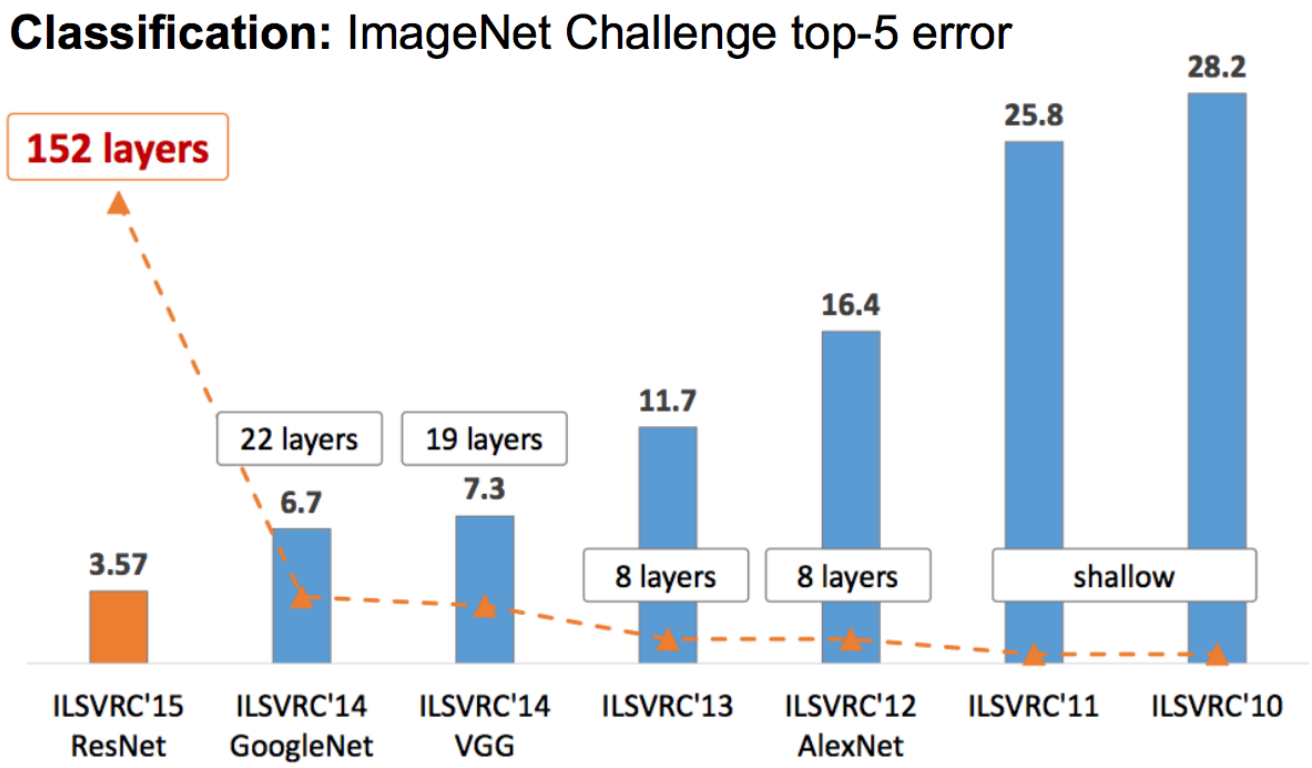
ResNet

- Each colored block of layers = series of convolutions of the same dimension. The feature mapping is periodically downsampled by **strided convolution** accompanied by an **increase in channel depth** to preserve the time complexity per layer. Dotted lines denote residual connections in which we project the input via a 1x1 convolution to match the dimensions of the new block. (no fully connected layers)

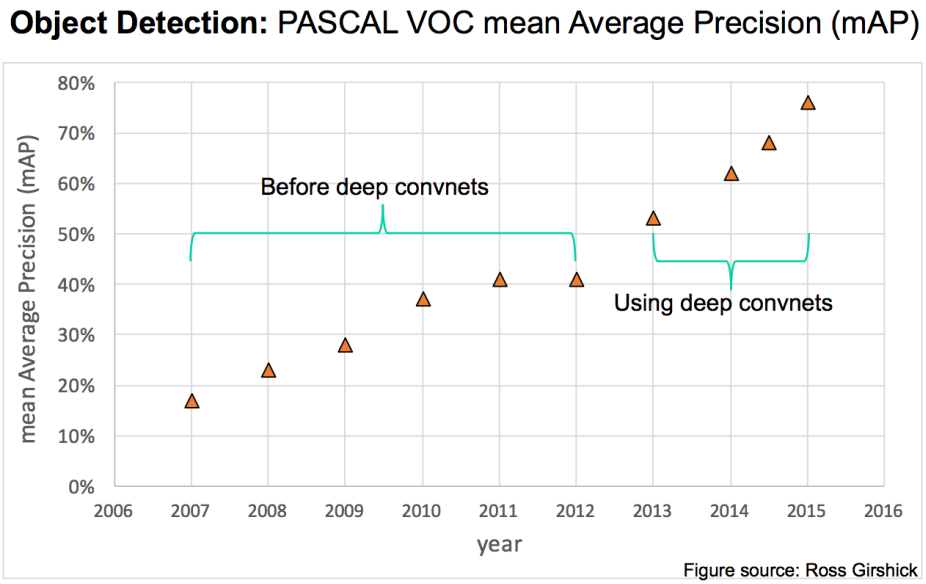
- Parameters: 25 million (ResNet 50)

https://arxiv.org/abs/1512.03385

Depth of the ILSVRC winners

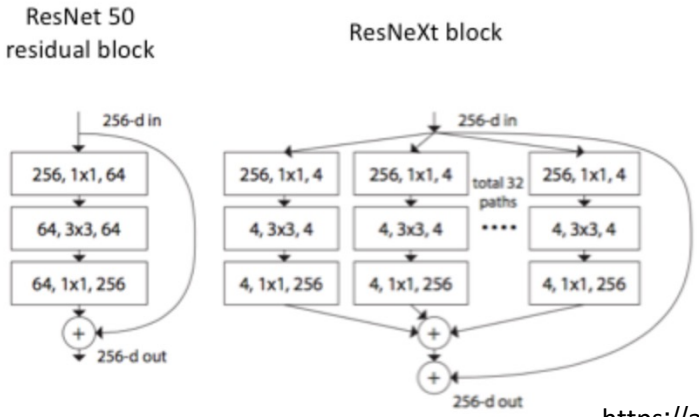


ILSVRC winners



ResNeXt

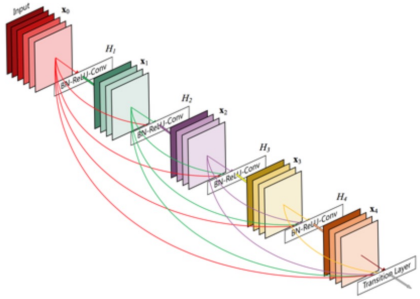
- Extension of ResNet, which replaces the standard residual block with one that leverages a "**split-transform-merge**" used in the Inception models:
  - Instead of performing convolutions over the full input feature map, the block's input is projected into a series of lower (channel) dimensional representations on which we separately apply a few convolutional filters before merging the results.

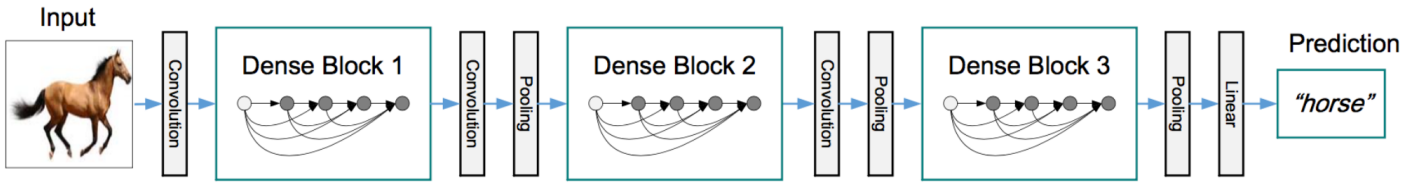


https://arxiv.org/abs/1611.05431

DenseNet

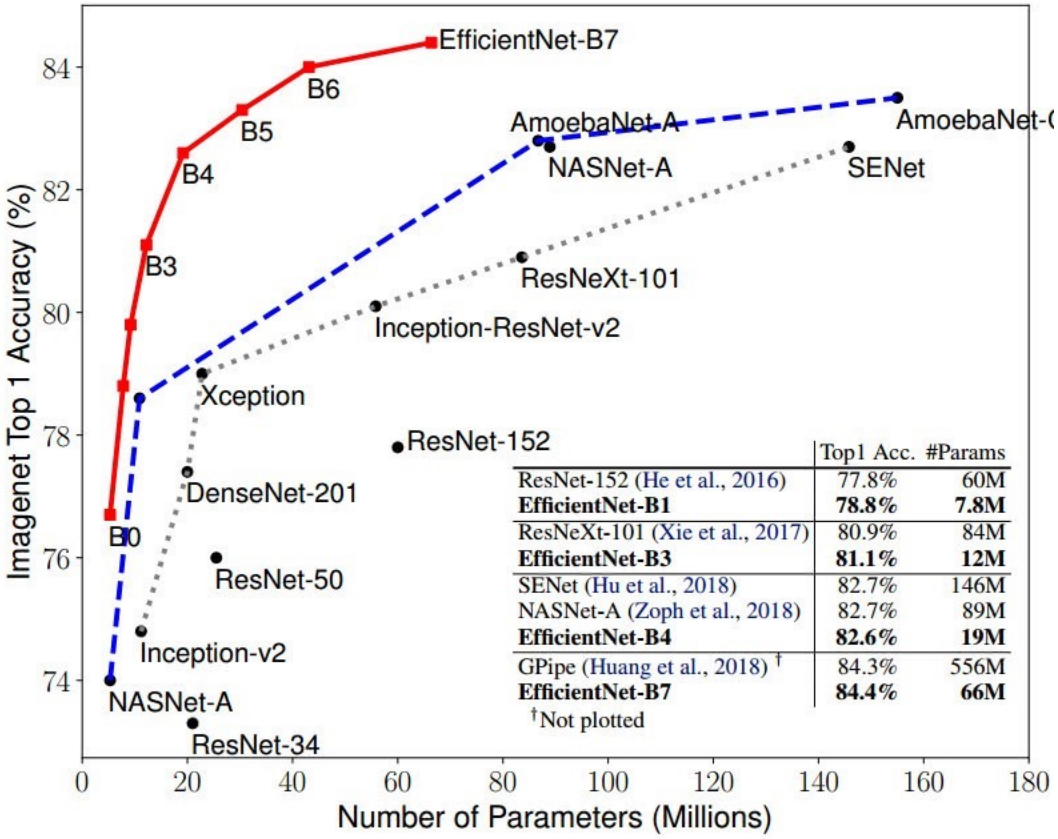
- **DenseNet** – idea: *"it may be useful to reference feature maps from earlier in the network".*
- Thus, each layer's feature map is **concatenated** to the input of every successive layer within a dense block. This allows later layers within the network to directly leverage the features from earlier layers, encouraging feature reuse within the network.
- Even better performance with less complexity, based on ResNet architecture
- Parameters:
  - 0.8 million (DenseNet-100, k=12)
  - 5.3 million (DenseNet-250, k=24)
  - 40 million (DenseNet-190, k=40)





https://arxiv.org/abs/1608.06993

ImageNet competition, anno 2020

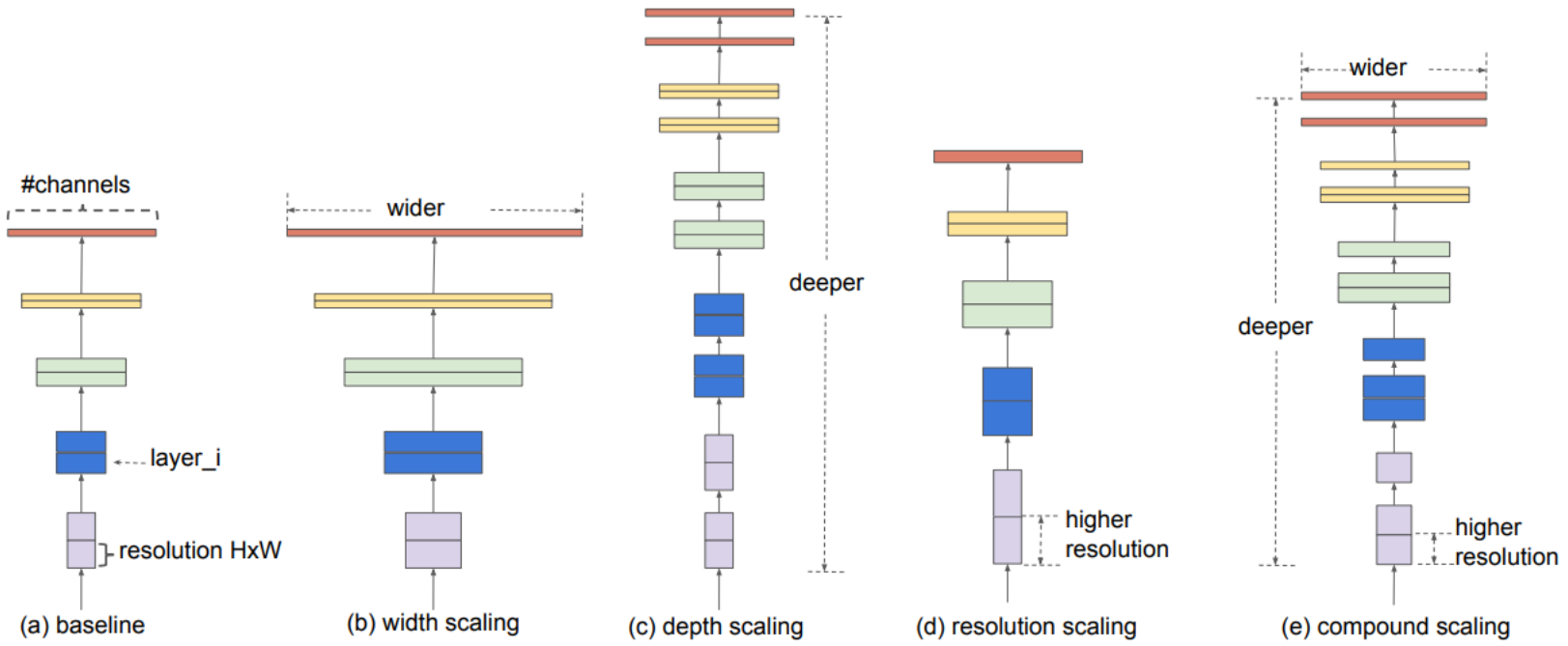


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

EfficientNet

- Step 1: perform a grid search to find the relationship between different scaling dimensions of the baseline network under a fixed resource constraint → appropriate scaling coefficient for each of the dimensions



EffientNet

- Carefully balanced network depth, width, and resolution can lead to better performance
- New scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective compound coefficient → more structured scaling (versus just move from ResNet-18 to ResNet-200)
- Automated machine learning (AutoML)

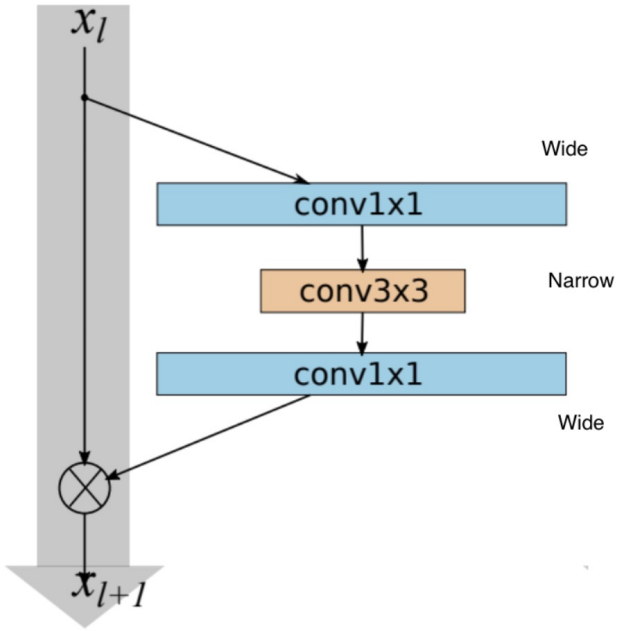
https://arxiv.org/pdf/1905.11946.pdf

MobileNetV2

- A new baseline network by performing a neural architecture search using the AutoML MNAS framework, which optimizes both accuracy and efficiency (FLOPS)
- Mobile **inverted** bottleneck convolution (MBConv), similar to MobileNetV2, but is slightly larger

Normal residual block

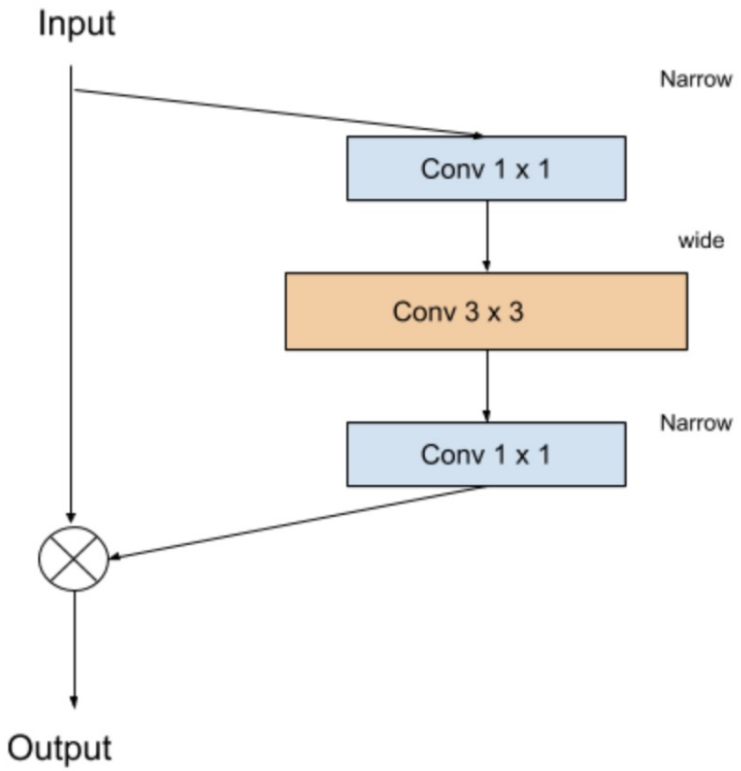
- Wide -> narrow -> wide
- Output is scaled back to the input size so x can be added to it



Residual Block. Image from PapersWithCode: https://paperswithcode.com/method/bottleneck-residual-block for free

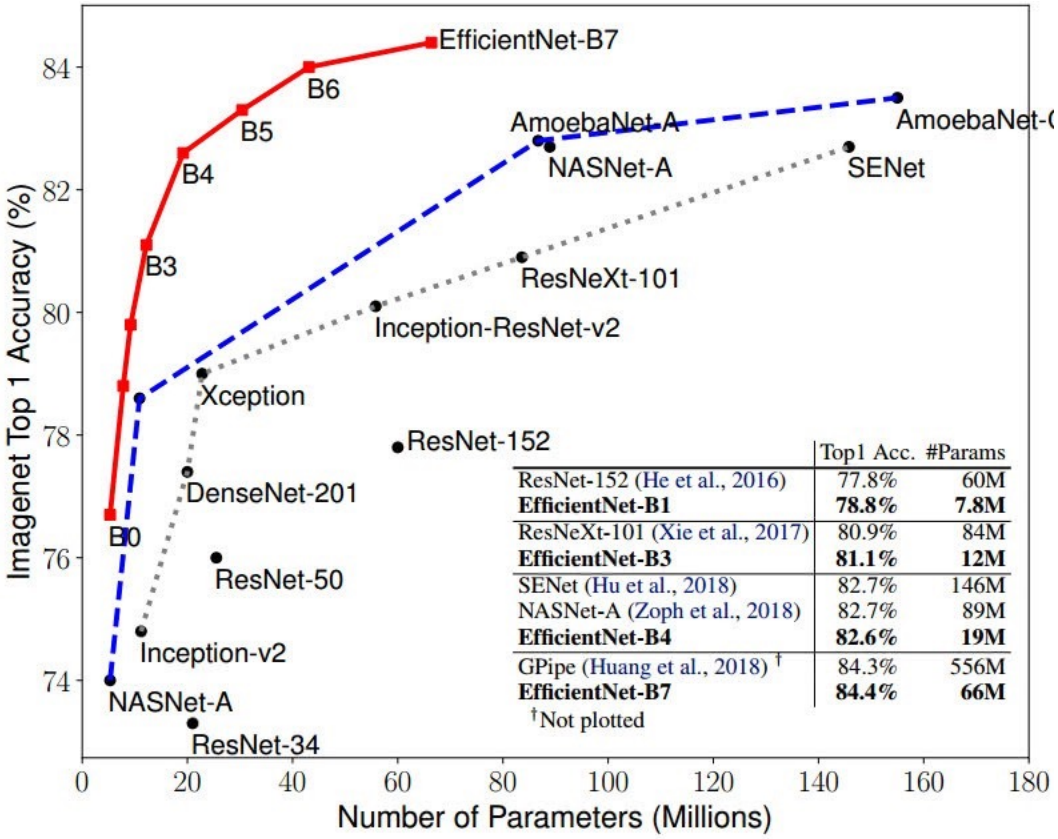
https://towardsdatascience.com/smart-way-to-levitate-convolutional-neural-networks-performance-efficientnet-google-ai-c87a1f67b084

Inverted residual blocks



https://towardsdatascience.com/smart-way-to-levitate-convolutional-neural-networks-performance-efficientnet-google-ai-c87a1f67b084

ImageNet competition, anno 2020



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

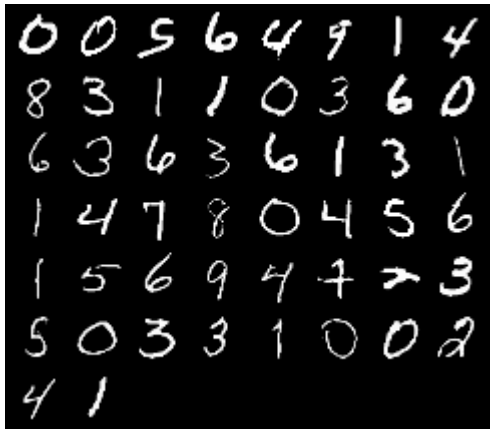
†Not plotted

More info

- In-depth reading on all these variants:
- https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

Handwritten digit recognition

- MNIST data set of handwritten digits from Yann LeCun's website
- All images are 28x28 grayscale
  - Pixel values from 0 to 255
- 60K training examples / 10K test examples
- Input vector of size 784
  - 28 * 28 = 784
- Output value is integer from 0-9



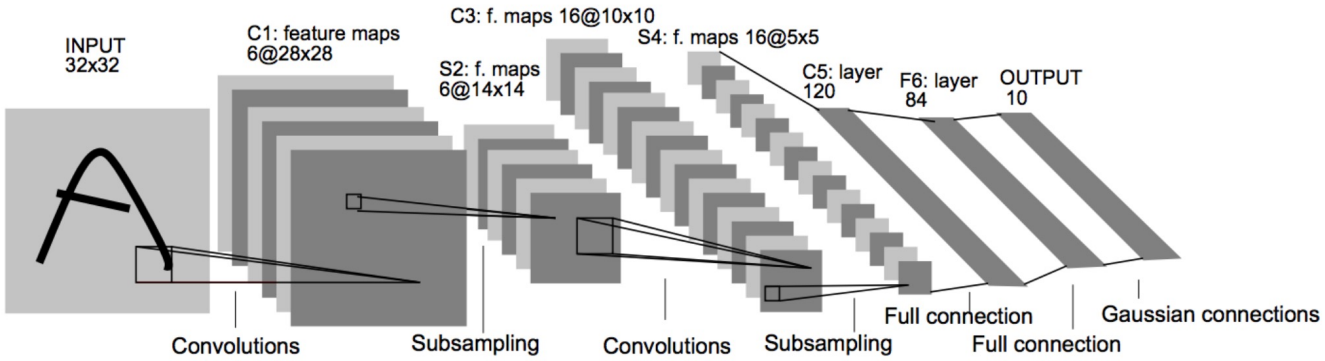
Slides inspired by the NVIDIA SUTD ambassador workshop

Hello world

MNIST APPLICATION

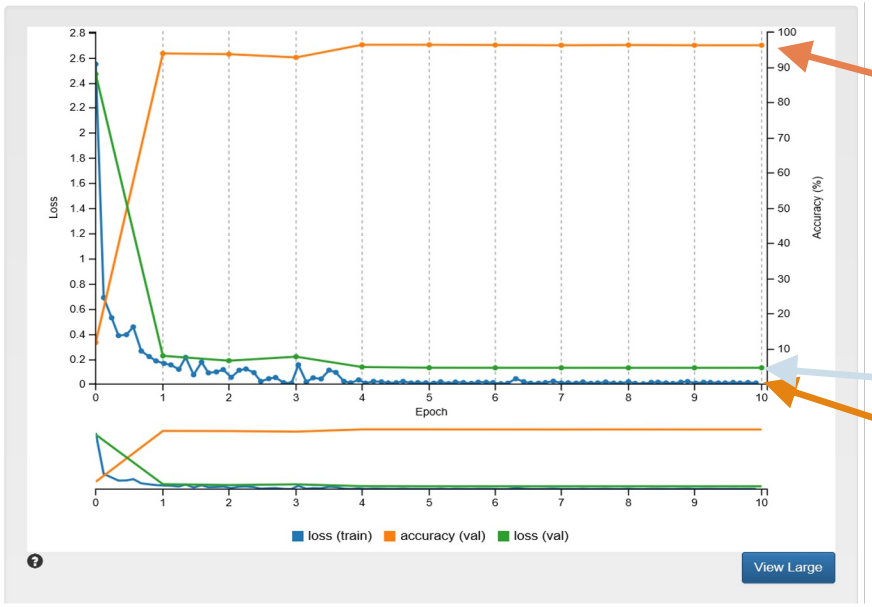
LeNet for MNIST

- LeNet with 10 epochs



Evaluation

- First experiment: trained on reduced small dataset (4x less data). This will allow us to test our architecture and debug.



Accuracy obtained from validation dataset

Loss function (Validation Loss)

Loss function (Training)

First result

Predictions

2	71.24%
0	23.76%
6	2.14%
9	0.65%
8	0.55%

First result

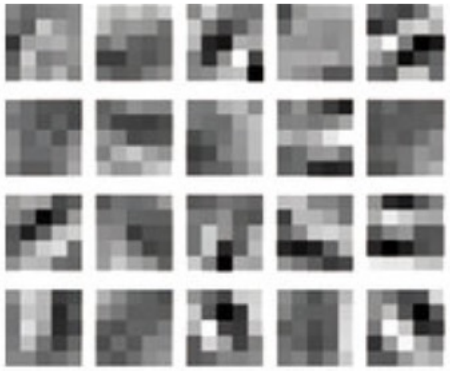
- Training done within minutes
- 96% classification accuracy
- Promising! Let's train on the complete dataset…

	SMALL DATASET
	1 : 99.90 %
	2 : 69.03 %
	8 : 71.37 %
	8 : 85.07 %
	0 : 99.00 %
	8 : 99.69 %
	8 : 54.75 %

Full dataset

- Now that we have tested our model, let's run it on the complete dataset, 4x larger
- 60k images for training, 10k test

What type of filters do we learn?



Results on full dataset

- 10 epochs
- 99% of accuracy achieved
- No improvements in recognizing real-world images
- Better results, but… why those bad results?

	SMALL DATASET	FULL DATASET
	1 : 99.90 %	0 : 93.11 %
	2 : 69.03 %	2 : 87.23 %
	8 : 71.37 %	8 : 71.60 %
	8 : 85.07 %	8 : 79.72 %
	0 : 99.00 %	0 : 95.82 %
	8 : 99.69 %	8 : 100.0 %
	8 : 54.75 %	2 : 70.57 %

Data augmentation

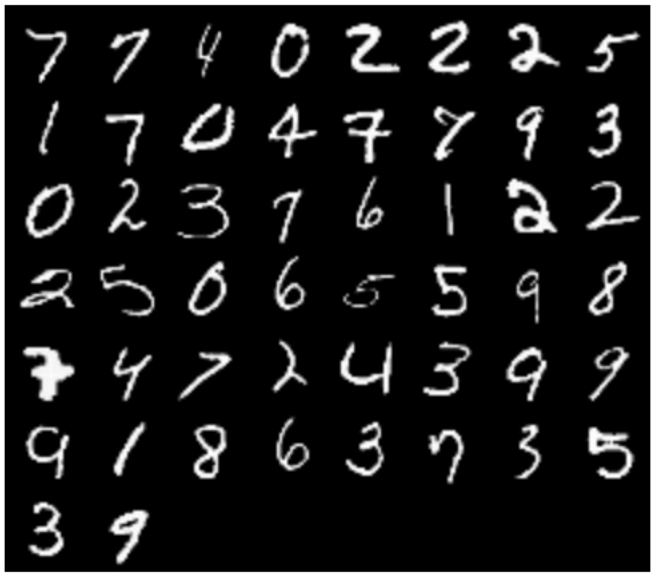
- For our seven test images that the backgrounds are not uniform. In addition, most of the backgrounds are light in color whereas our training data all have black backgrounds.
- We saw that increasing the amount of data did help for classifying the handwritten characters, so what if we include more data that tries to address the contrast differences?

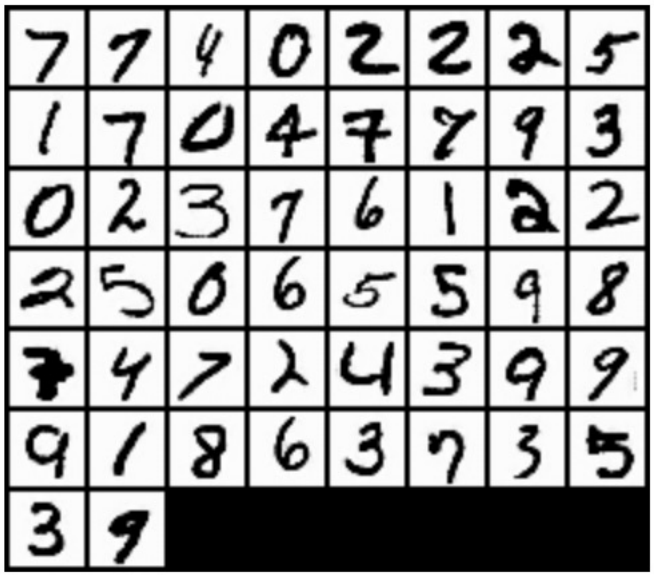
  => invert images

  Let's turn the white pixels to black and vice-versa. Then we will train our network using the original and inverted images and see if classification is improved.

Data augmentation

- Pixel(Inverted) = 255 – Pixel(original)
- White letter with black background. -> Black letter with white background





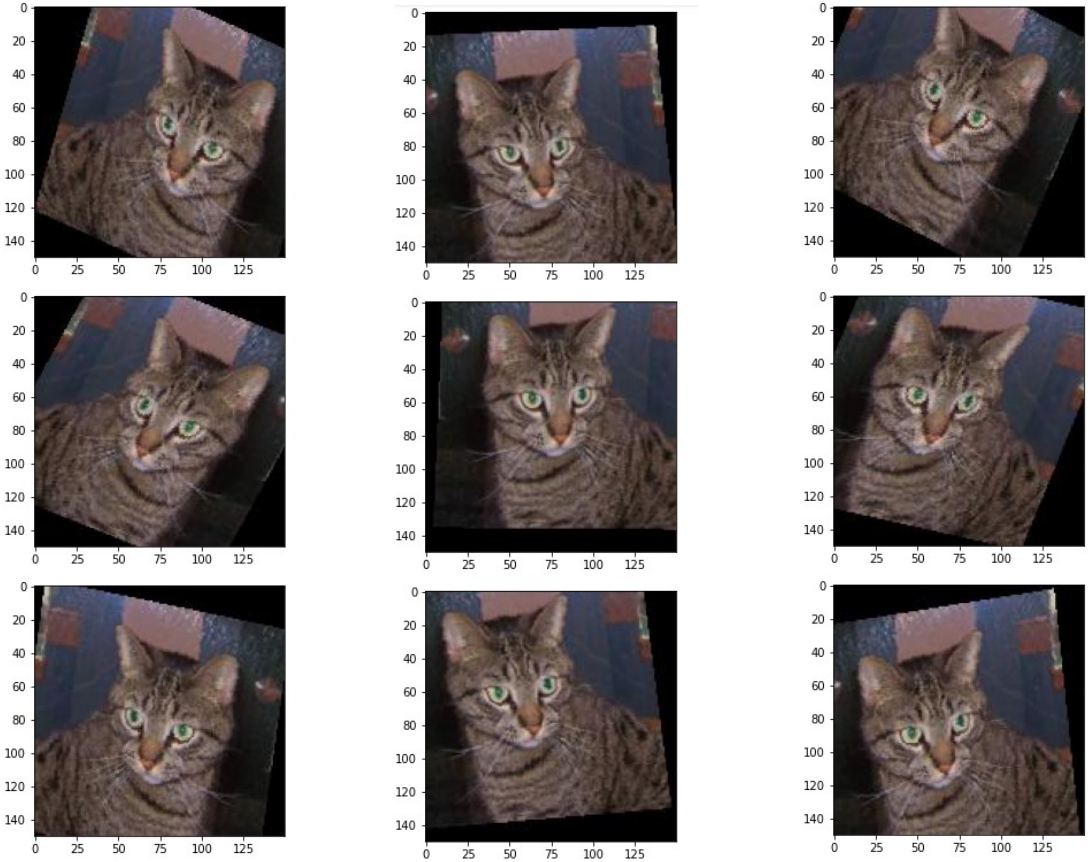
Results with augmentation

- Better!

	FULL DATASET	+INVERTED
	0 : 93.11 %	1 : 90.84 %
	2 : 87.23 %	2 : 89.44 %
	8 : 71.60 %	3 : 100.0 %
	8 : 79.72 %	4 : 100.0 %
	0 : 95.82 %	7 : 82.84 %
	8 : 100.0 %	8 : 100.0 %
	2 : 70.57 %	2 : 96.27 %

Data augmentation

- Other ways of augmenting images include: rotation, shift, resize, zoom,



```
augtransforms=transforms.Compose([
transforms.Resize((150,150)),
transforms.RandomAffine(degrees=(-30,30),
                        translate=(0.1,0.1),
                        shear=(-10,10)),
transforms.RandomHorizontalFlip(),
transforms.ToTensor()])
#transforms.RandomRotation(degrees=(-40,40))])
#transforms.RandomResizedCrop((150,150),
#scale=(0.8,1),
#ratio=(0.9,1.1))
#transforms.RandomCrop(150)
```

Beyond images

Convolution only for images?

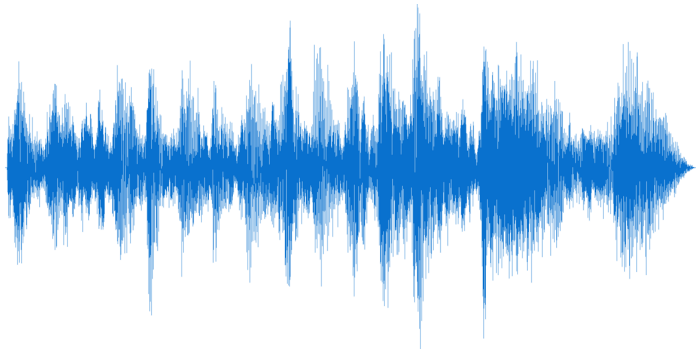
- While image processing has made convolution popular, CNNs have applications in a number of other domains that deal with multi dimensional matrices
- The data does not need to be 2-D, can be applied to 1-D or multi-dimensional data as well
- Video
- Text processing (word vectors)
- Audio tasks (spectrograms)
- …

Audio

- Data analytics is also important for audio:
  - Spoofing detection
  - Music genre classification
  - Spoken word detection
  - Hit prediction
  - Speaker identification
  - Emotion detection from music / voice
  - …
- How do we start on this? Well, audio can be represented as an image…

Audio as an image

- Simple waveforms:

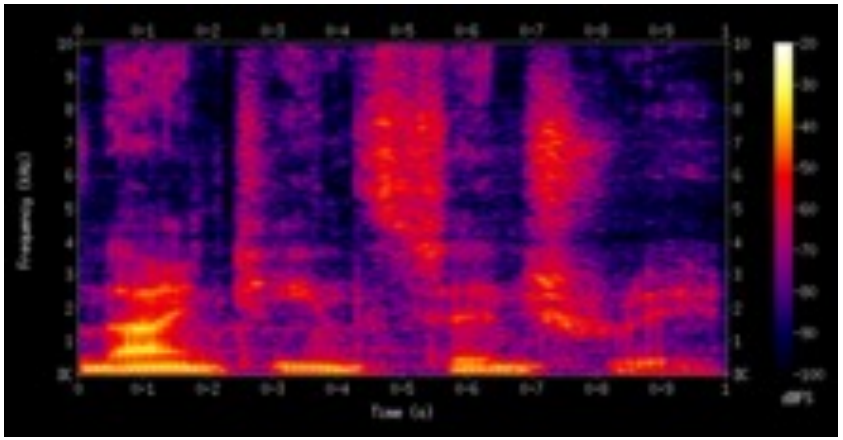


- More informative are spectrograms: a visual representation of the spectrum of frequencies of sound or other signal as they vary with time.

Spectrograms

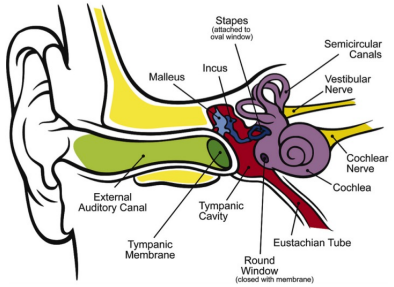
- Typical spectrogram of a few spoken words.
  - Y-axis: frequencies
  - X-axis: time
- Lower frequencies can be seen as more dense (brighter) here, as it's a male voice

The power spectrum of a time signal describes the ***distribution of power into frequency components*** composing that signal. Using ***Fourier analysis***, we can decompose a signal into discrete frequencies (a spectrum of frequencies) over a continuous range.



Mel spectrograms

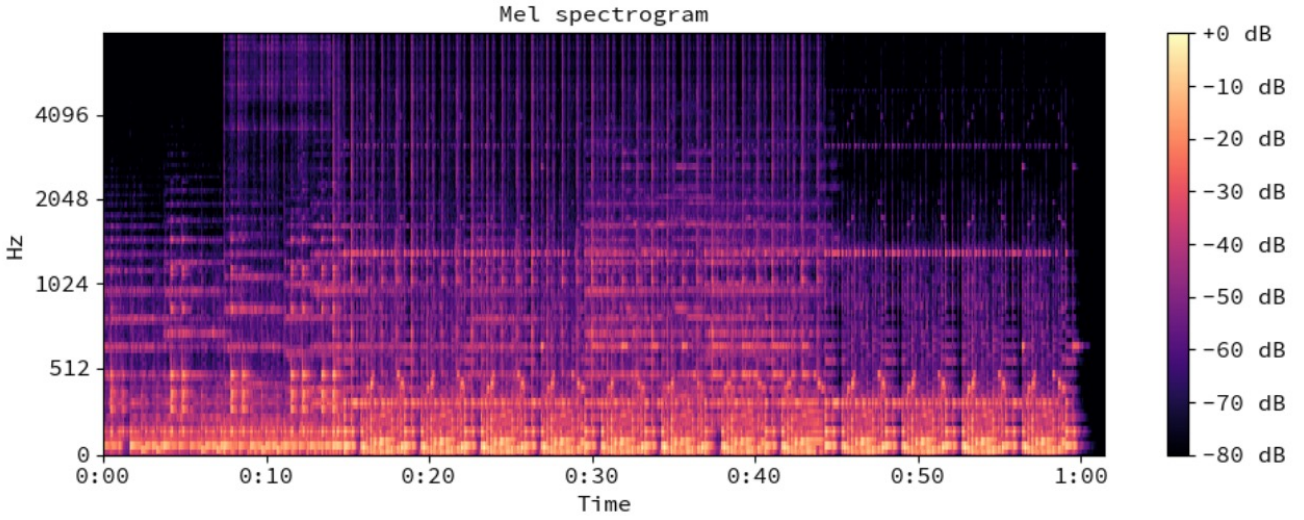
- A effective type of spectrogram in sound processing is **Mel-frequency cepstrum (MFC),** which represents the **short-term** power spectrum of a sound, based on a linear Cosine Transform (similar to Fourier Transform) of a log power spectrum on a **nonlinear Mel scale** of frequency.
- Through historical experiments, researchers have determined that we can distinguish better between tones in the lower frequencies than higher frequencies.
- The Mel filterbank represents **human hearing** more accurately.
- Check out TA Raven's library: nnAudio for fast spectrogram calculation



Mel spectrograms

- Commonly used as features in speech recognition systems, e.g. to automatically recognize numbers spoken into a telephone; and music information retrieval applications such as genre classification, audio similarity measures, etc.
- Not very robust in the presence of additive noise
  -> common to normalize their values

Mel Spectrogram



y:np.ndarray [shape=(n,)] or None
audio time-series

sr:number > 0 [scalar]
sampling rate of y

S:np.ndarray [shape=(d, t)]
spectrogram

n_fft:int > 0 [scalar]
length of the FFT window

hop_length:int > 0 [scalar]
number of samples between successive frames. See `librosa.core.stft`

power:float > 0 [scalar]
Exponent for the magnitude melspectrogram. e.g., 1 for energy, 2 for power, etc.

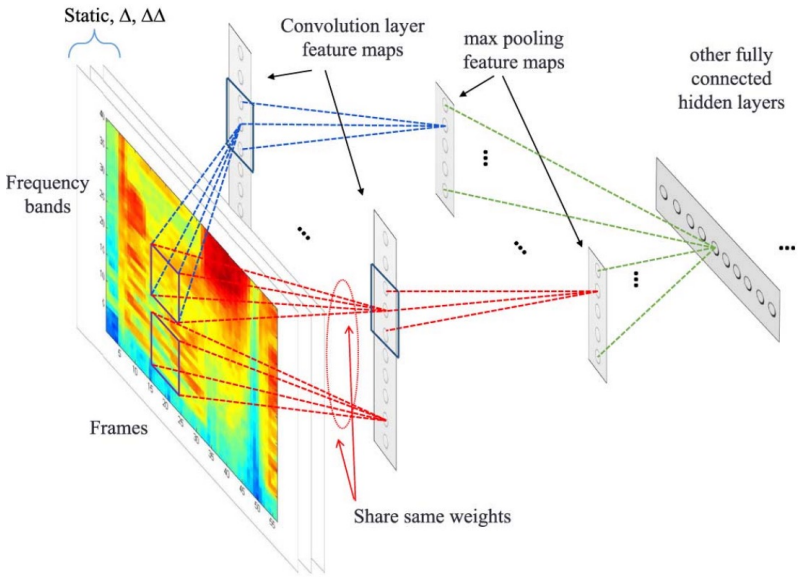
kwargs:additional keyword arguments
Mel filter bank parameters. See `librosa.filters.mel` for details.

S:np.ndarray [shape=(n_mels, t)]
Mel spectrogram

librosa.feature.melspectrogram(*y=None, sr=22050, S=None, n_fft=2048, hop_length=512, power=2.0, **kwargs*) [source]

Speech recognition

- Abdel-Hamid et al., 2014
- Large vocabulary automatic speech recognition task: 18 hours



https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/CNN_ASLPTrans2-14.pdf

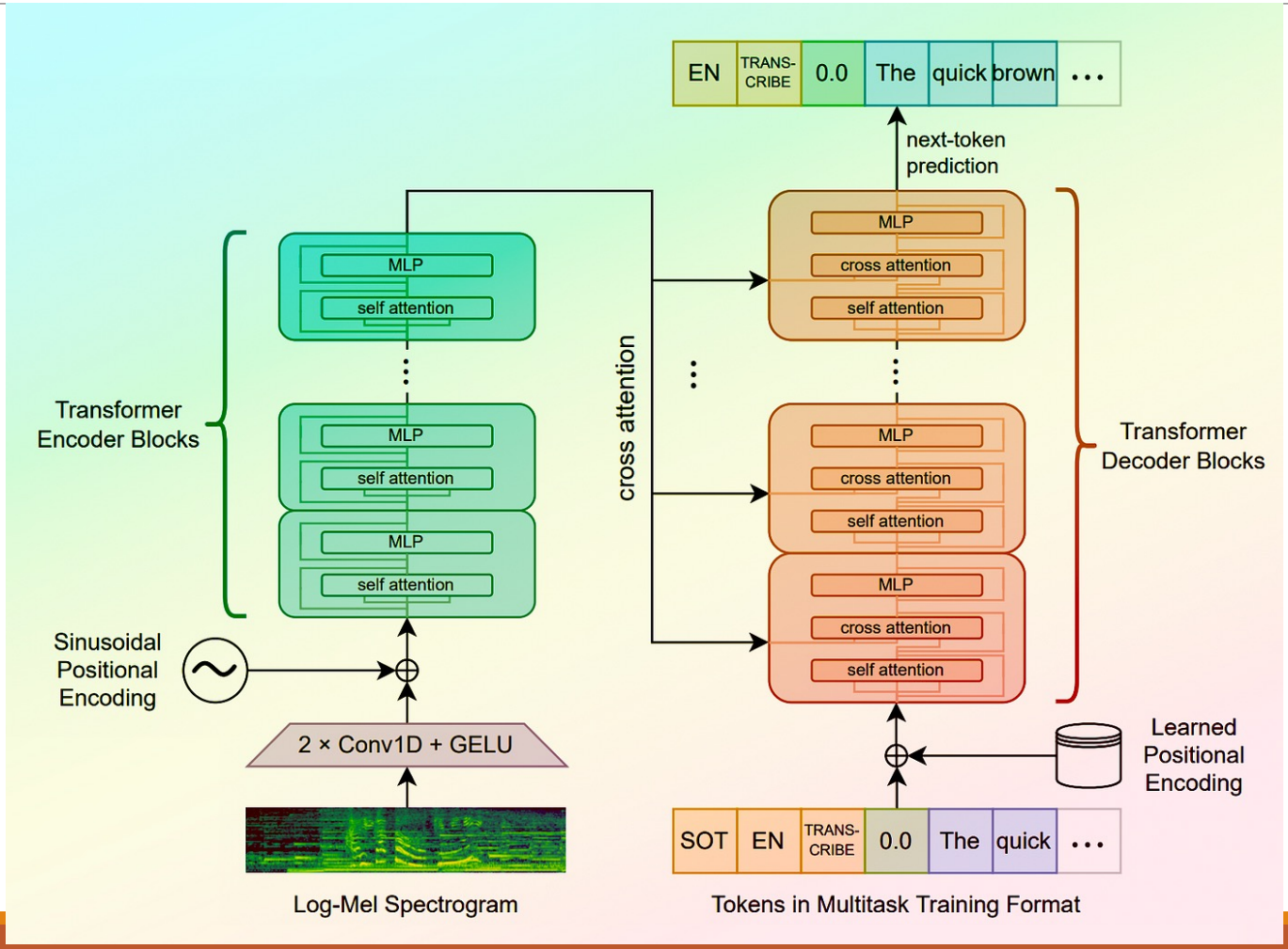
Speech recognition

- With and without pre-training (with Restricted Boltzmann Machines)
- CNN has:
  - one pair of convolution and pooling
  - two hidden fully connected layers
  - 84 feature maps per section
  - filter size of 8
  - pooling size of 6
  - a stride of 2
- Context window has 11 frames
- Compared to 3-layer DNN (size 2000)
- The table represents word error rate (WER)
- Current SOTA: whisper from openAI: transformer based

	No PT	With PT
DNN	37.1%	35.4%
CNN	34.2%	33.4%

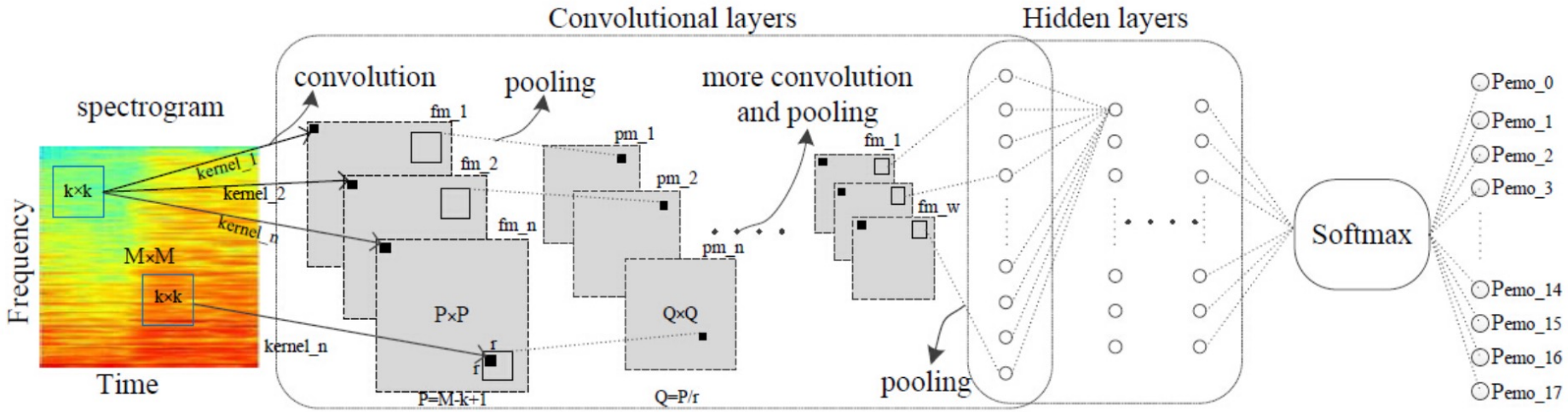
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/CNN_ASLPTrans2-14.pdf

Whisper model



Music emotion classification

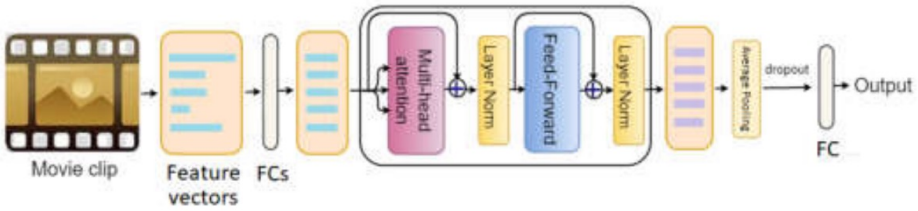
- Liu et al., 2017
- Dataset: CAL500exp
  - 3,223 short segments labeled with 18 emotions tags
- 4 convolutional layers with 1 hidden layer



https://arxiv.org/pdf/1704.05665.pdf

AttendAffectNet - Multimodal emotion classification

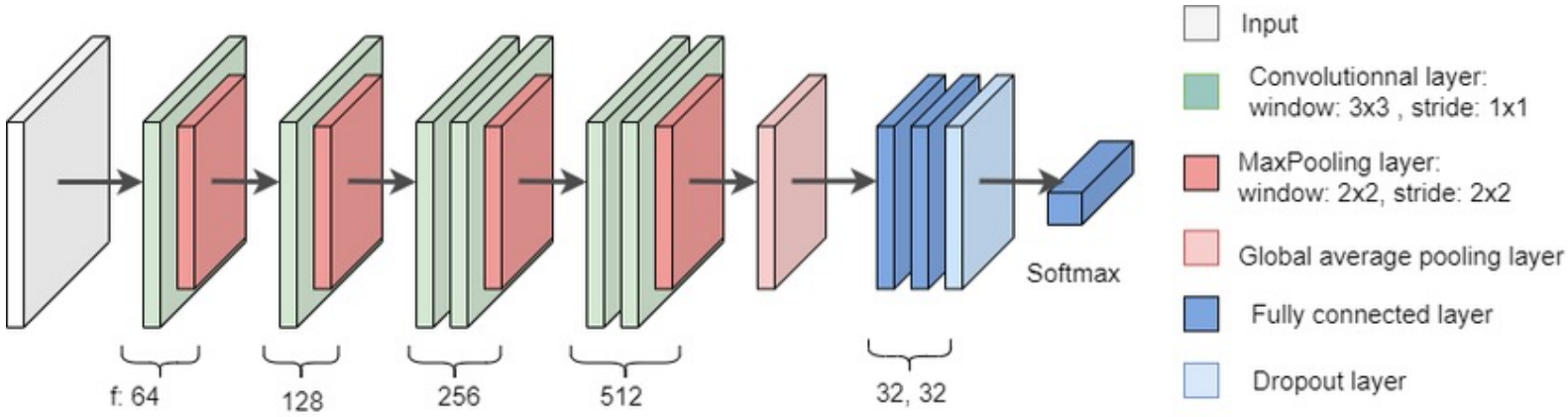
- Dataset: movie clips
- Predicting: valence and arousal
- Input: pretrained latent features:
  - Audio:
    - OpenSmile
    - VGGish (pretrained AudioSet)
  - Video:
    - ResNet-50 (pretrained ImageNet)
    - RGB-stream I3D network (appearance)
    - FlowNetS (motion features)



T. Phuong HThi, BT B, Herremans D., Roig G.. 2021. AttendAffectNet: Self-Attention based Networks for Predicting Affective Responses from Movies. Proceedings of the International Conference on Pattern Recognition (ICPR2020). https://arxiv.org/abs/2010.11188

VGGish

- **VGGish**, by Google (8 layers)
- Trained on AudioSet, 2 million human-labeled 10-second sound clips drawn from YouTube videos



VGGish

- 128-dimensional embeddings of each AudioSet segment produced from a VGG-like audio classification model that was trained on a large YouTube dataset
- Variant of the VGG model, with 11 weight layers. Specifically, here are the changes made:
  - The input size was changed to 96x64 for log Mel spectrogram audio inputs.
  - We drop the last group of convolutional and maxpool layers, so we now have only four groups of convolution/maxpool layers instead of five.
  - Instead of a 1000-wide fully connected layer at the end, we use a 128-wide fully connected layer. This acts as a compact embedding layer.
  - https://github.com/tensorflow/models/tree/master/research/audioset/vggish (source)

AttendAffectNet - Predicted versus actual values

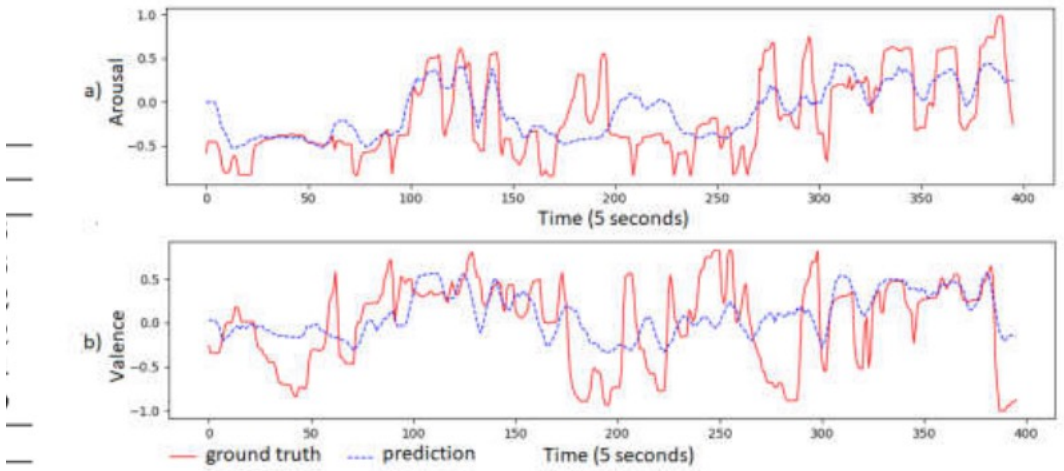


Fig. 5. Arousal (a) and valence (b) values for the "Million Dollar Baby" movie clip.

TABLE I
ACCURACY OF THE PROPOSED MODELS ON THE EXTENDED COGNIMUSE DATASET.

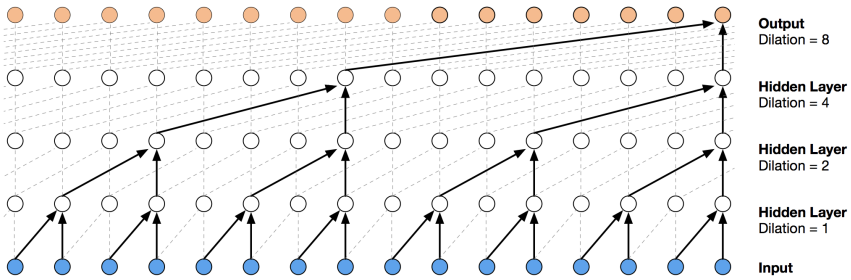
Models	Arousal		Valence	
	MSE	PCC	MSE	PCC
Feature AAN (only video)	0.152	0.518	0.204	0.483
Feature AAN (only audio)	0.125	0.621	0.185	0.543
Feature AAN (video and audio)	**0.124**	**0.630**	**0.178**	**0.572**
Temporal AAN (only video)	0.178	0.457	0.267	0.232
Temporal AAN (only audio)	0.162	0.472	0.247	0.254
Temporal AAN (video and audio)	**0.153**	**0.551**	**0.238**	**0.319**
Sivaprasad et al. [23]	**0.08**	**0.84**	**0.21**	**0.50**

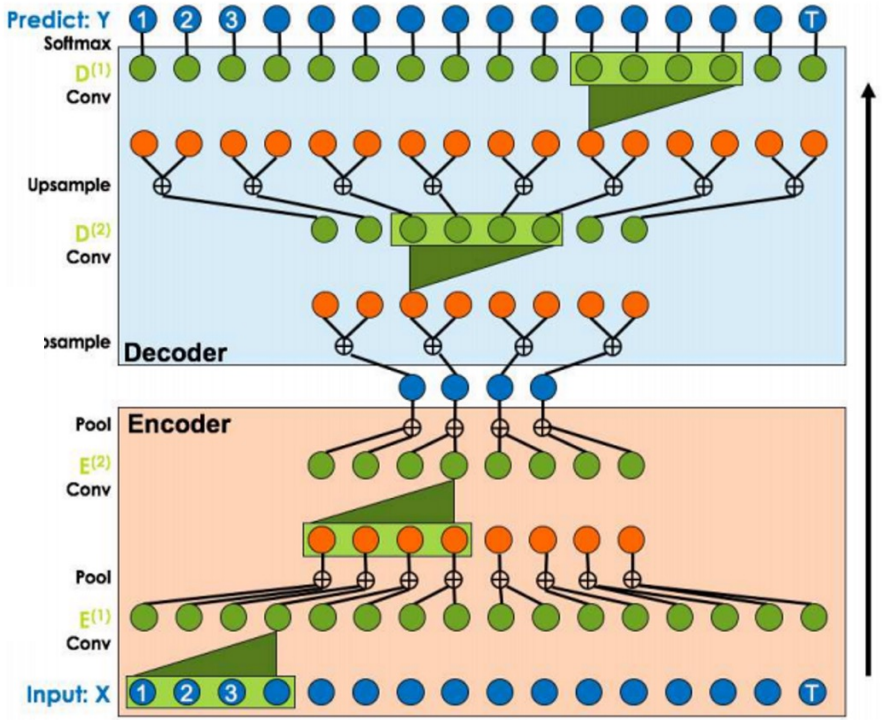
TABLE II
ACCURACY OF THE PROPOSED MODELS IN COMPARISON WITH STATE-OF-THE-ART ON THE GLOBAL EIMT16.

Models	Arousal		Valence	
	MSE	PCC	MSE	PCC
Feature AAN (only video)	0.933	0.350	0.764	0.342
Feature ANN (only audio)	1.111	0.397	0.209	0.327
Feature ANN (video and audio)	**0.742**	**0.503**	**0.185**	**0.467**
Temporal ANN (only video)	1.182	0.151	0.256	0.190
Temporal ANN (only audio)	1.159	0.185	0.225	0.285
Temporal ANN (video and audio)	**0.854**	**0.210**	**0.218**	**0.415**
Liu et al. [56]	1.182	0.212	0.236	0.379
Chen et al. [55]	1.479	0.467	0.201	0.419
Yi et al. [22]	1.173	0.446	0.198	0.399
Yi et al. [41]	**0.542**	**0.522**	**0.193**	**0.468**

Temporal Convolutional NN

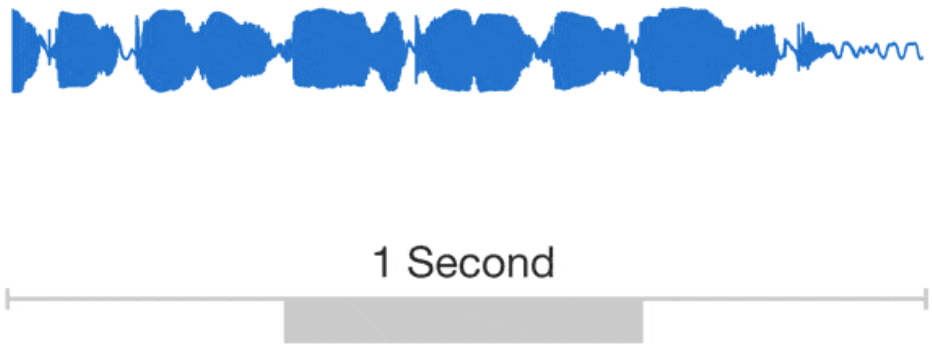
- TCN can take a series of any length and output it as the same length.
- The output at time *t* is only convolved with the elements that occurred before *t*.
- Dilated convolutions.





Wavenet

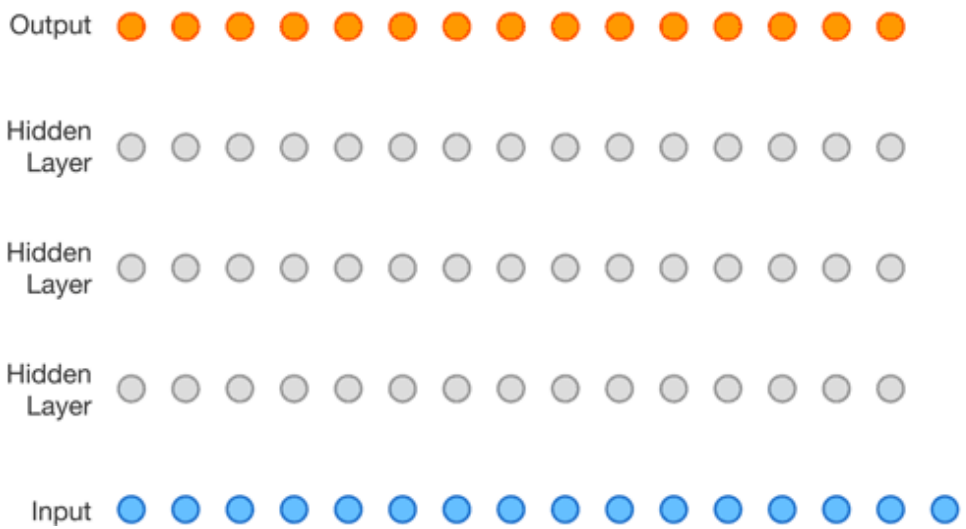
- Modelling raw audio is tricky: typically 16,000 samples per second or more, with important structure at many time-scales



https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

Wavenet

- A fully convolutional neural network, where the convolutional layers have various dilation factors that allow its receptive field to grow exponentially with depth and cover thousands of timesteps.
- Used for generative models as well as classification
  - Gated activations
  - Residual connections



https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

HEAR Benchmark

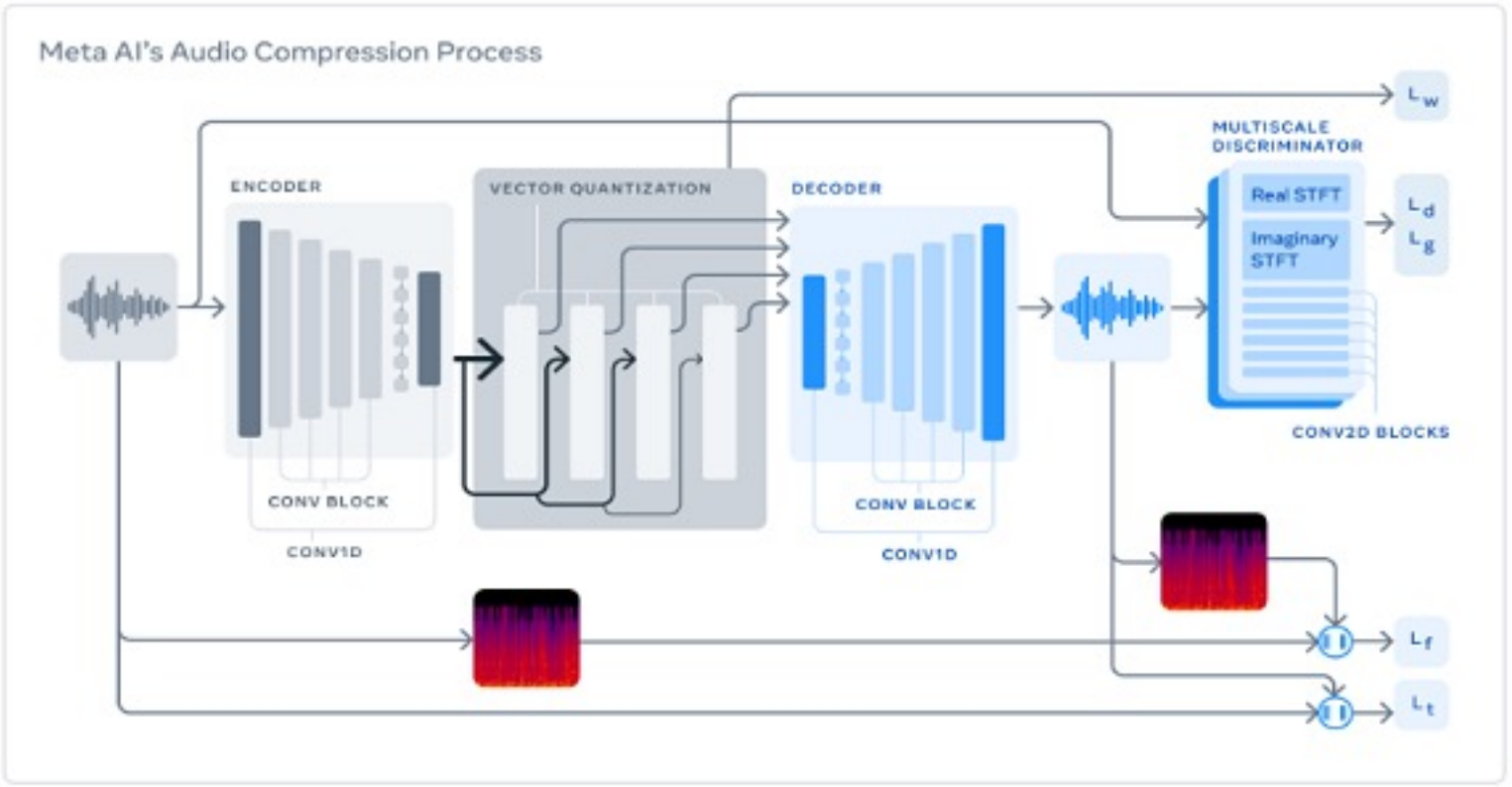
Holistic Evaluation of Audio Representations

- A benchmark of nineteen diverse tasks. These tasks encompass multiple audio domains: speech, environmental sound, and music, with tasks that involve short and long time spans;
- Part of NeurIPS Competitions, 2021

Model	URL	Submission Date	Beehive	Beijing Opera	CREMA-D	DCASE 2016	ESC-50	FSD50K	C
RedRice ced_base		2023-09-26	0.4835	0.9660	0.6910	0.9219	0.9665	0.6548	
GURA Fuse Cat H+w+C		HEAR 2021		0.9660	0.7474	0.8260	0.7335	0.4197	
RedRice ced_small		2023-09-26	0.5170	0.9660	0.6664	0.9163	0.9595	0.6433	
GURA Fuse Cat H+w+C (time)		HEAR 2021		0.9618	0.7427	0.8260	0.6535	0.3742	
RedRice		2023-09-26	0.5017	0.9618	0.6526	0.9066	0.9535	0.6388	

Audio representations

- Recent advances include: ‘Neural audio encoders”
  - Descript Audio Codec (DAC)
  - Encodec (Meta) -> codebooks



Codebooks

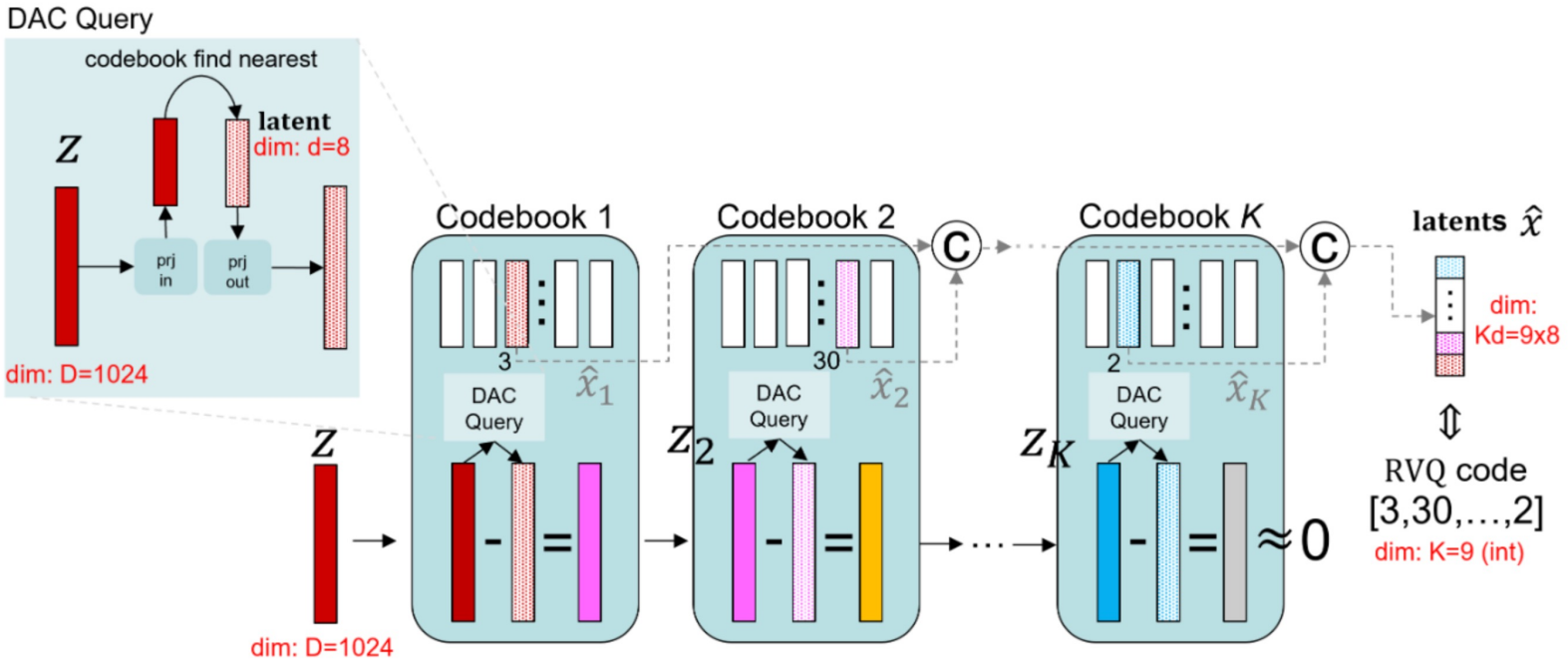
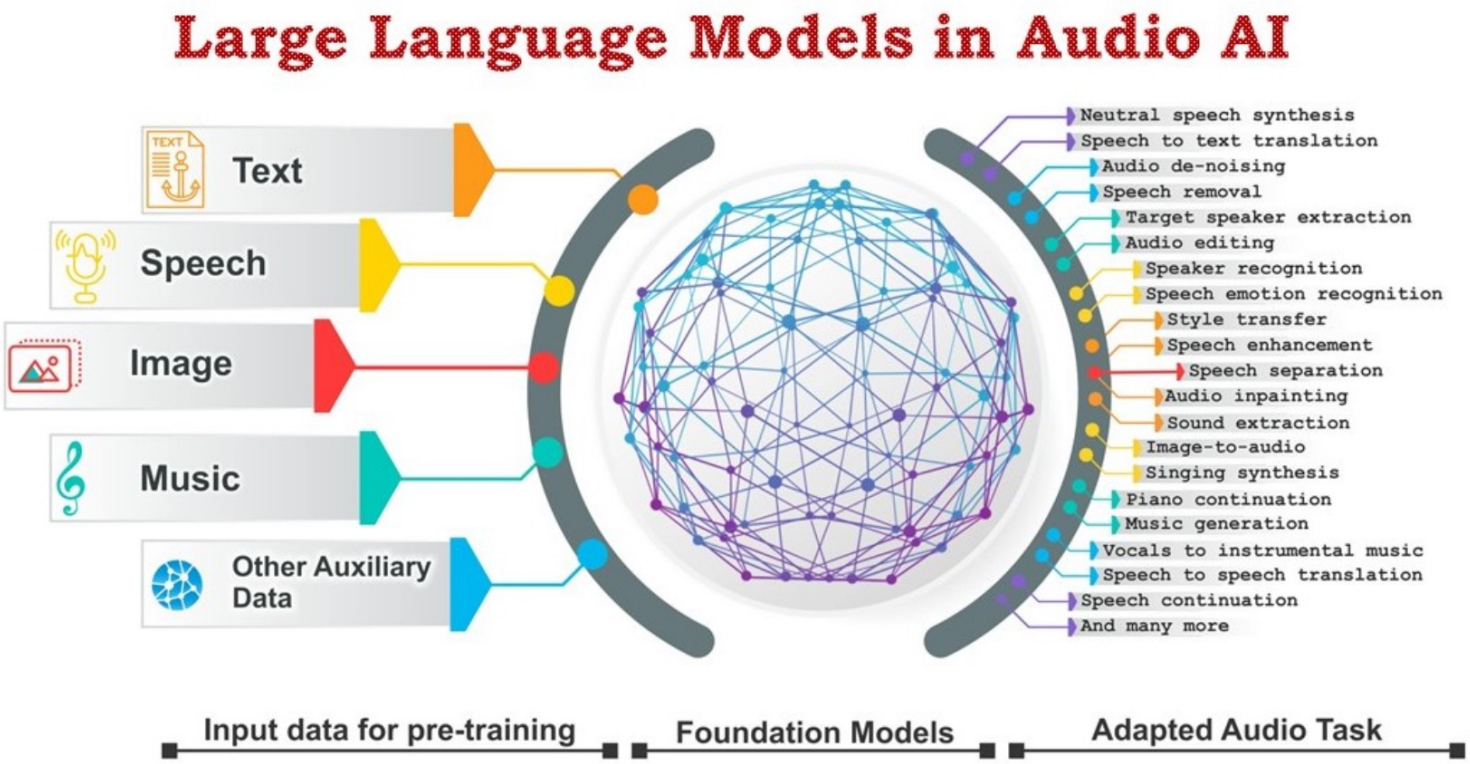


Figure 2. RVQ quantizing process illustrated on DAC, where the "find nearest" operation in EnCodec RVQ is replaced with DAC Query.



This repo supplements our survey: Sparks of Large Audio Models: A Survey and Outlook.

https://github.com/EmulationAI/awesome-large-audio-models

References

- https://www.deeplearningbook.org/contents/convnets.html
- https://github.com/BlackBindy/MNIST-invert-color
- https://www.slideshare.net/GauravMittal68/convolutional-neural-networks-cnn
- http://slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf
- https://www.jeremyjordan.me/convnet-architectures/#resnext
- https://www.superdatascience.com/ppt-the-ultimate-guide-to-convolutional-neural-networks-cnn/