SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN
Established in collaboration with MIT

Convolution – media processing

PROF. D. HERREMANS

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

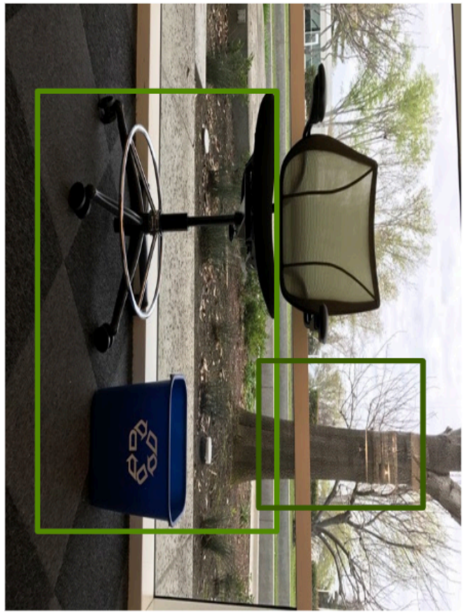


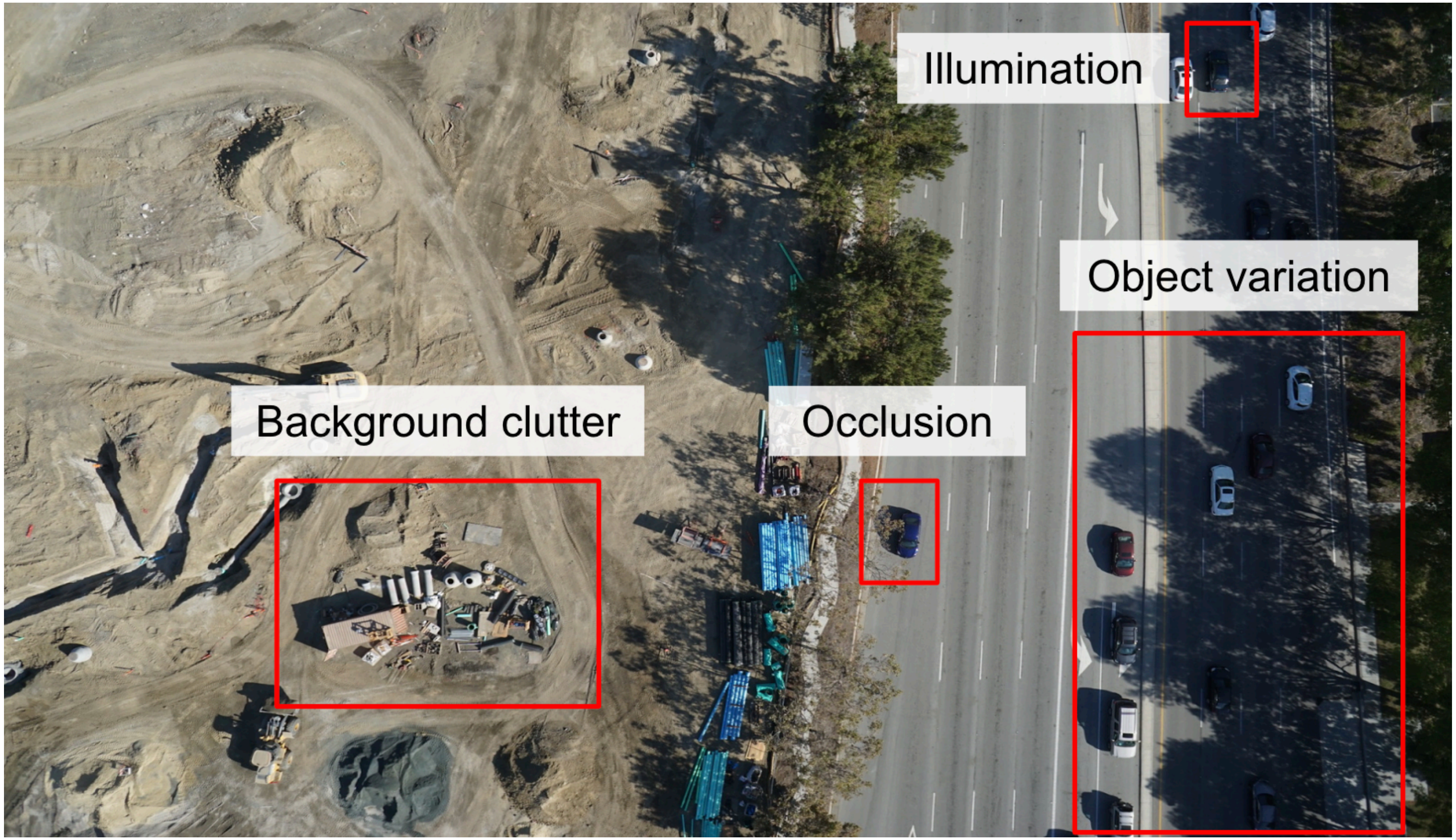
Image Segmentation

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

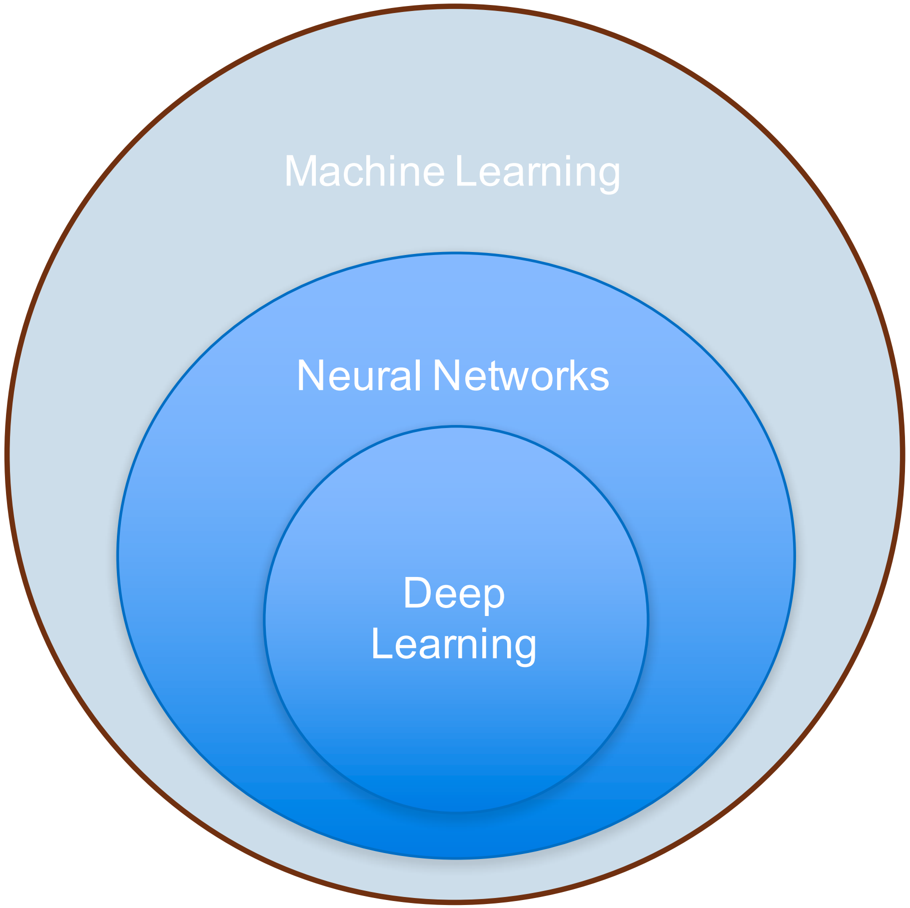
Image classification

- 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.

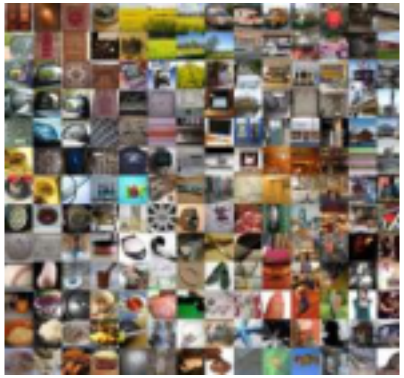
Challenges in images

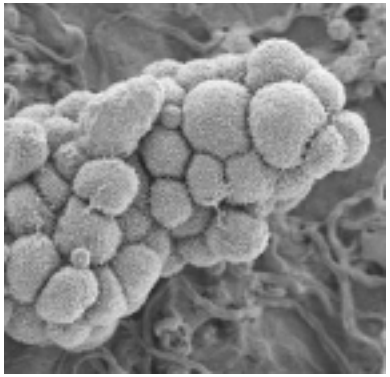


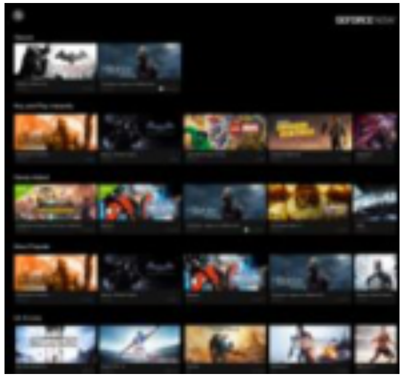
Deep learning



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

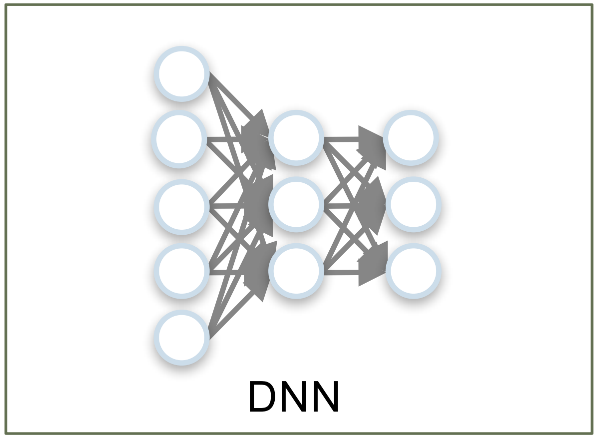
SECURITY & DEFENSE

Face Detection
Video Surveillance
Satellite Imagery

AUTONOMOUS MACHINES

Pedestrian Detection
Lane Tracking
Recognize Traffic Sign

The big bang in machine learning

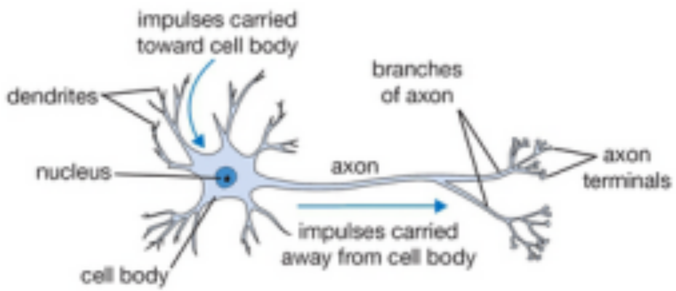


“Google’s AI engine also reflects how the world of computer hardware is changing. (It) depends on machines equipped with GPUs… And it depends on these chips more than the larger tech universe realizes.”

WIRED

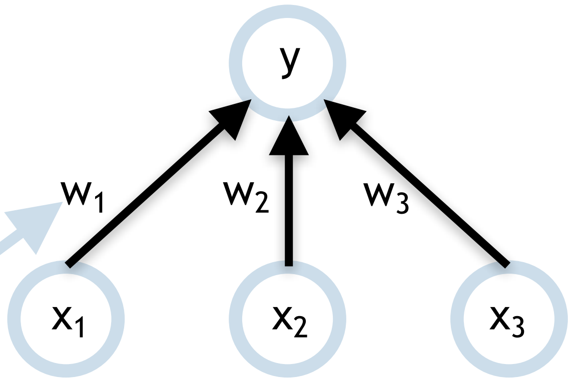
Artificial neurons

Biological neuron



From Stanford cs231n lecture notes

Artificial neuron

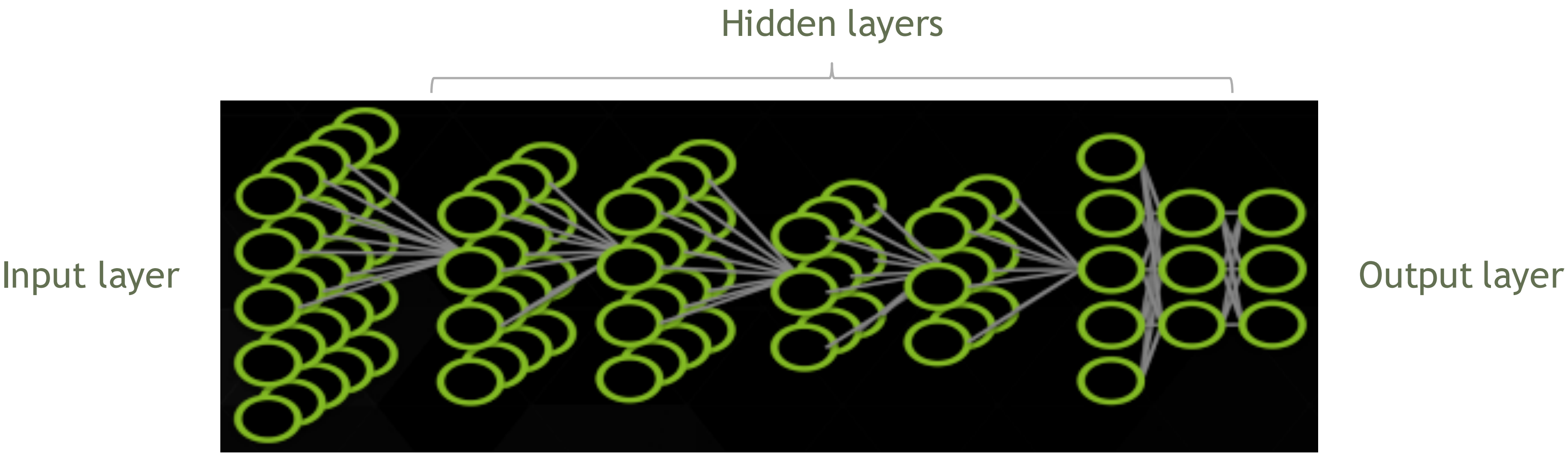


Weights (W_n)
= parameters

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

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

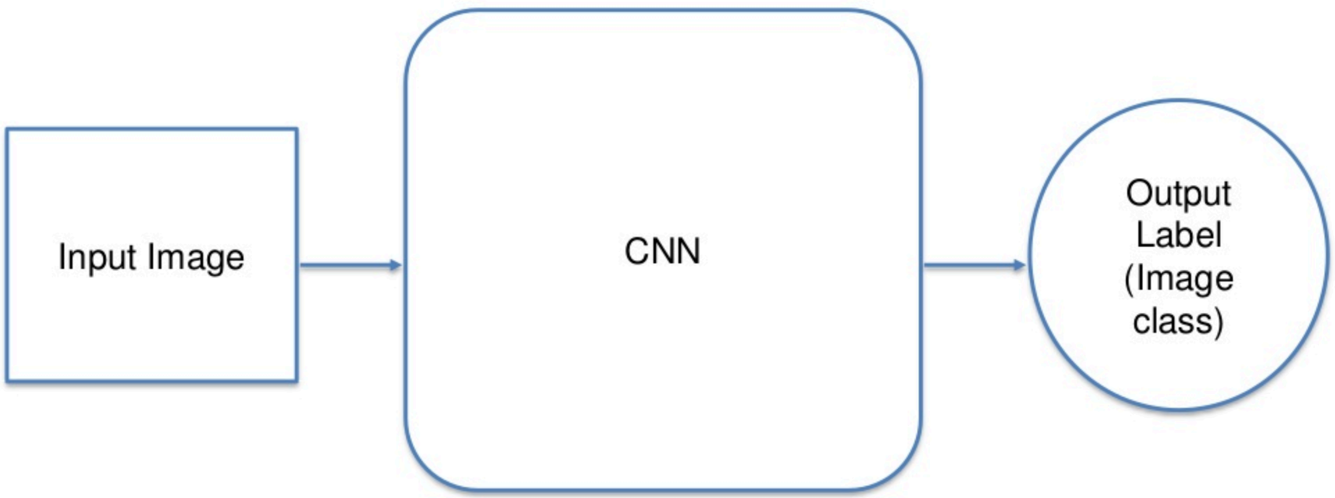
- Founding father of convolutional neural networks
- Chief AI scientist Facebook

Convolutional neural networks

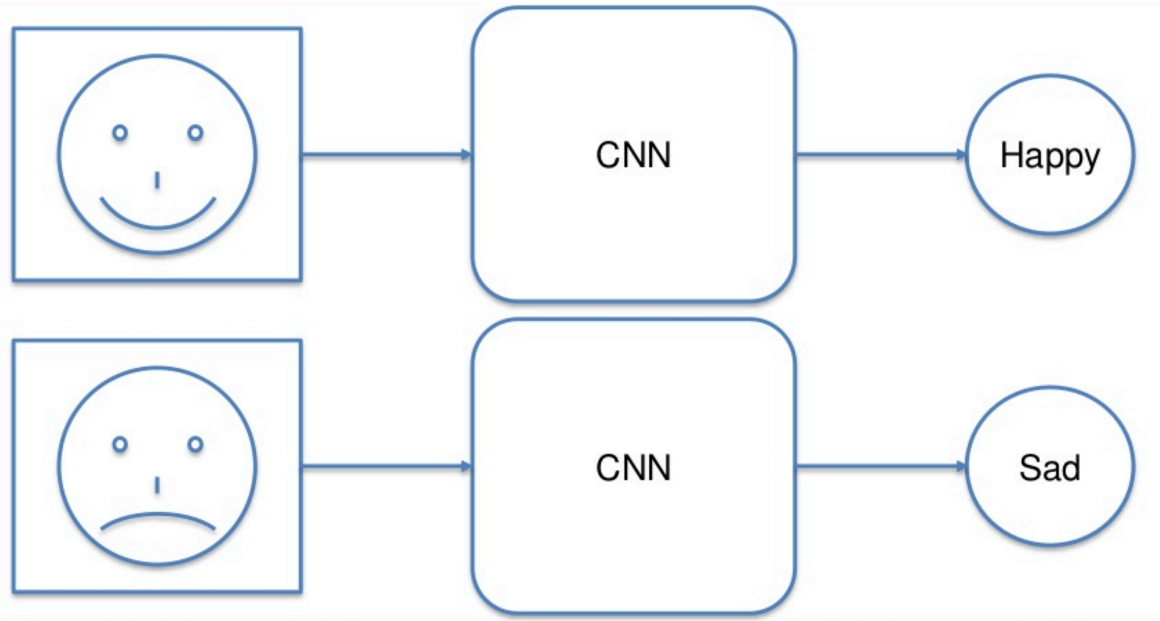
- LeCun, 1989
- "…are a specialized kind of neural network for processing data that has a known **grid-like topology**. Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image data, which can be thought of as a 2-D grid of pixels. Convolutional 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.

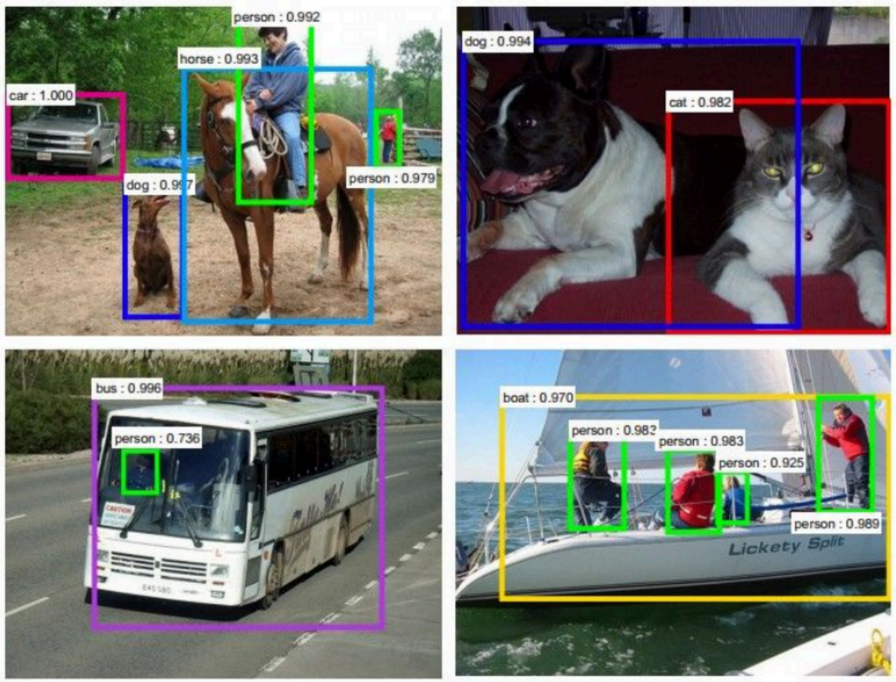


CNNs



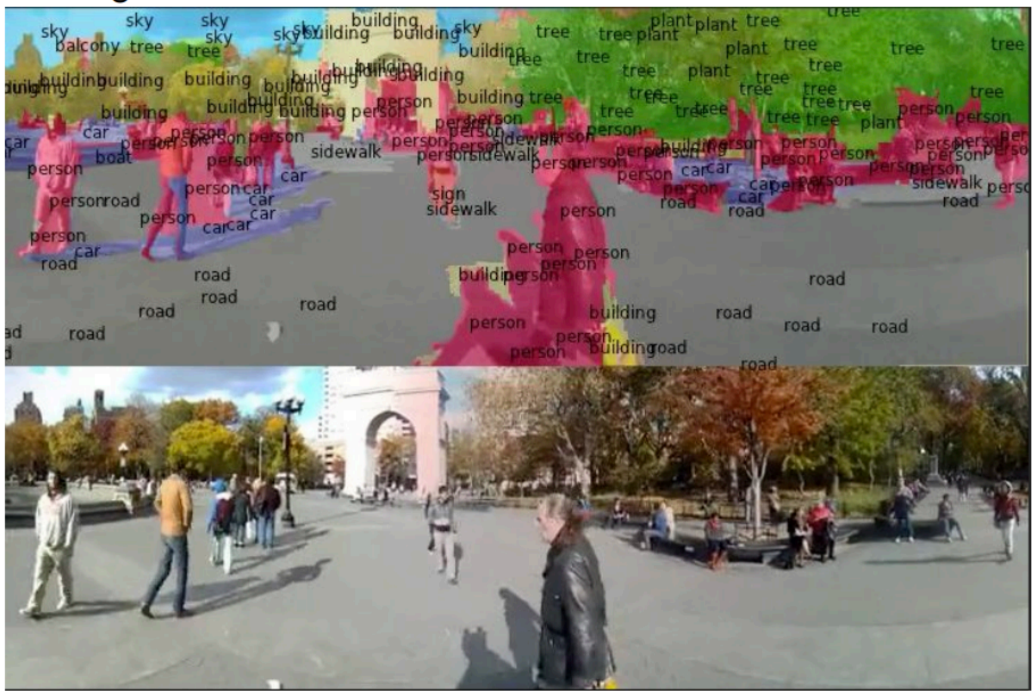
CNNs are everywhere

Detection



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

Segmentation



Figures copyright Clement Farabet, 2012.

CNNs are everywhere

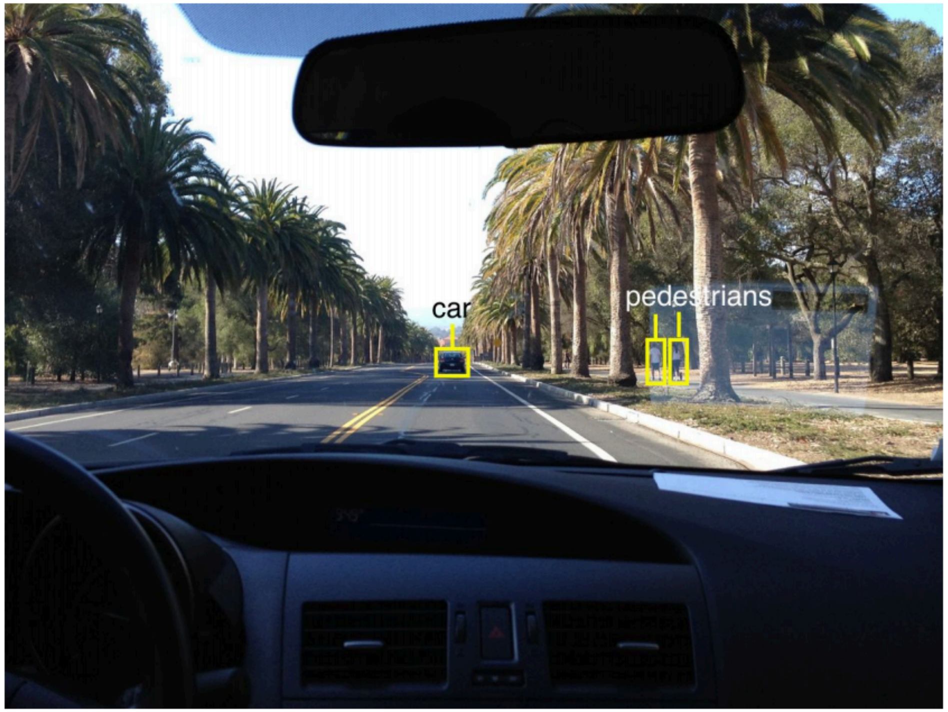
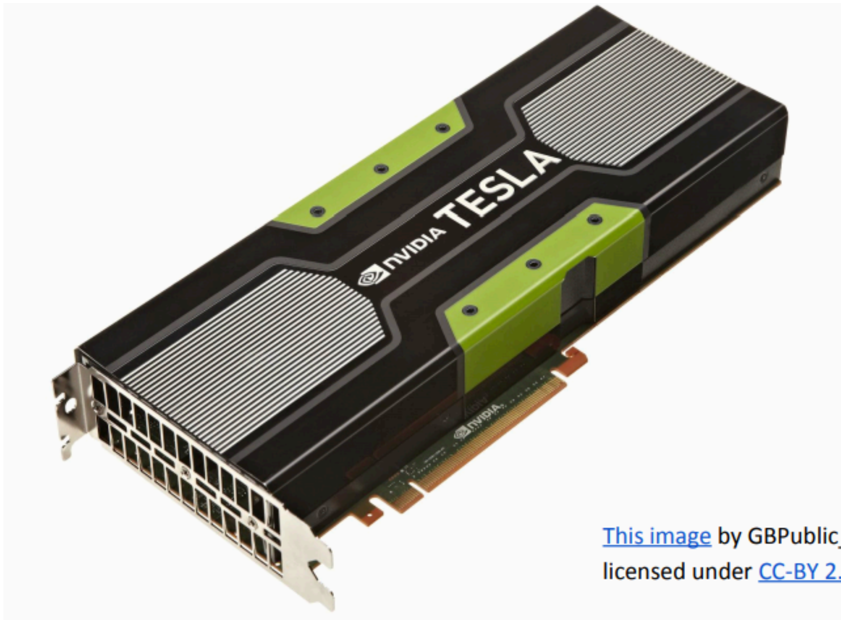


Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars



CNNs are everywhere

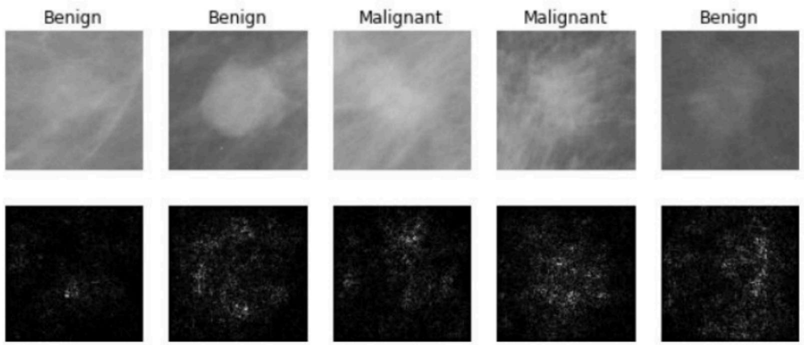
Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[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]

[Dieleman et al. 2014]

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

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

Image Captioning

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

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

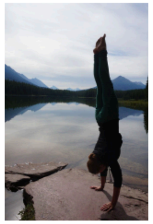
Minor errors

A man in a baseball uniform throwing a ball

A cat sitting on a suitcase on the floor

Somewhat related

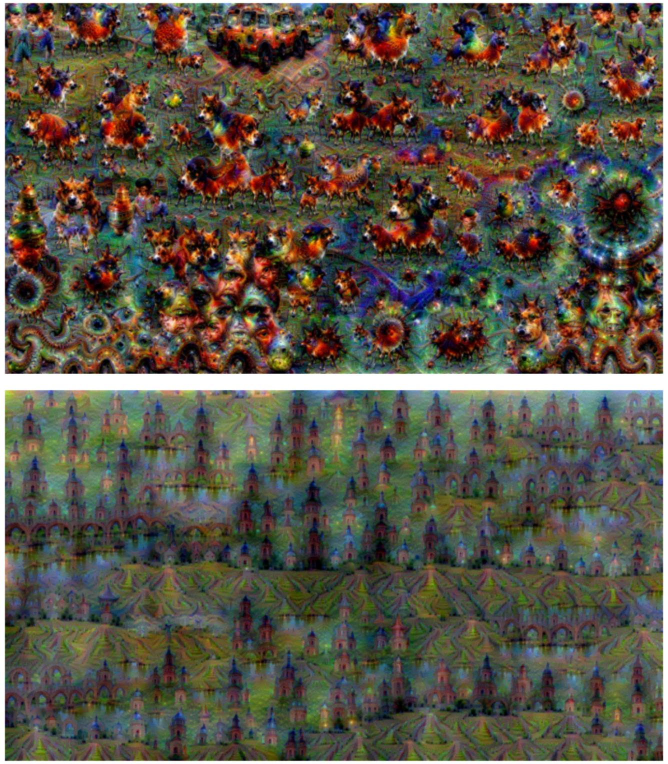
A woman is holding a cat in her hand



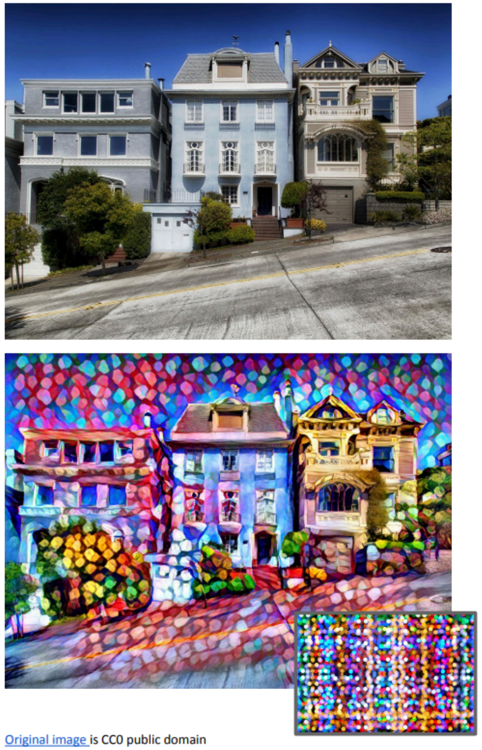
A woman standing on a beach holding a surfboard

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.

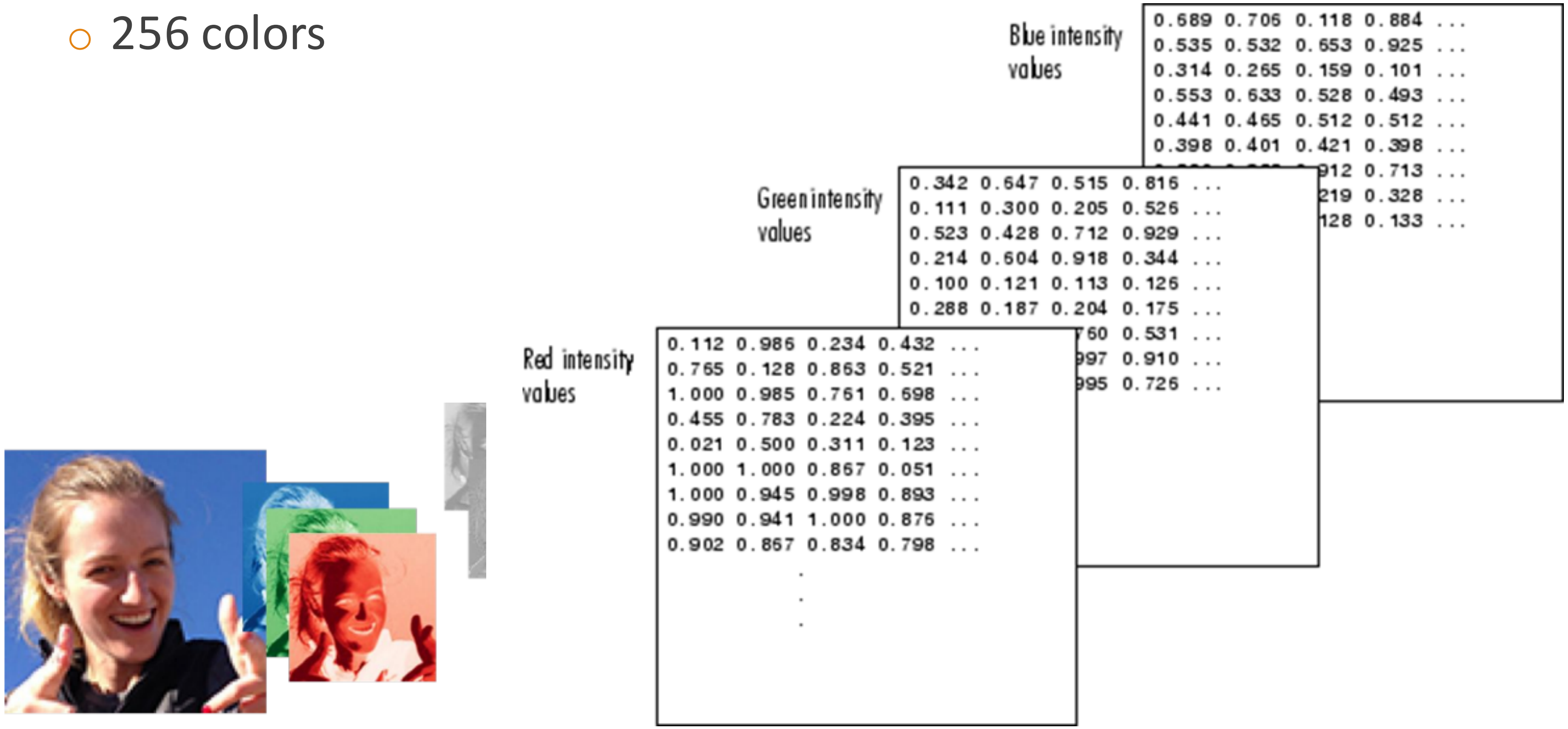


Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;

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?

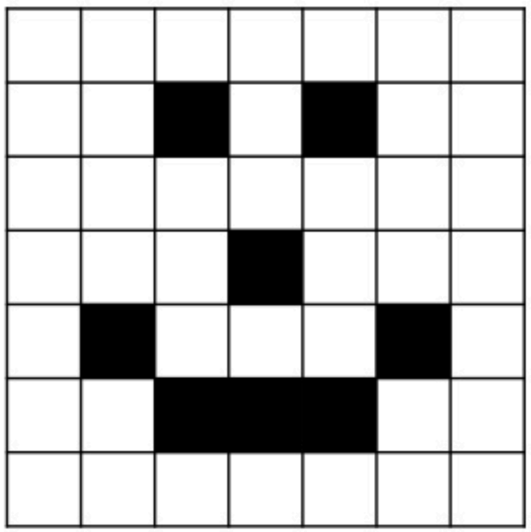
- Matrix with 3 layers (RGB)
- 256 colors



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

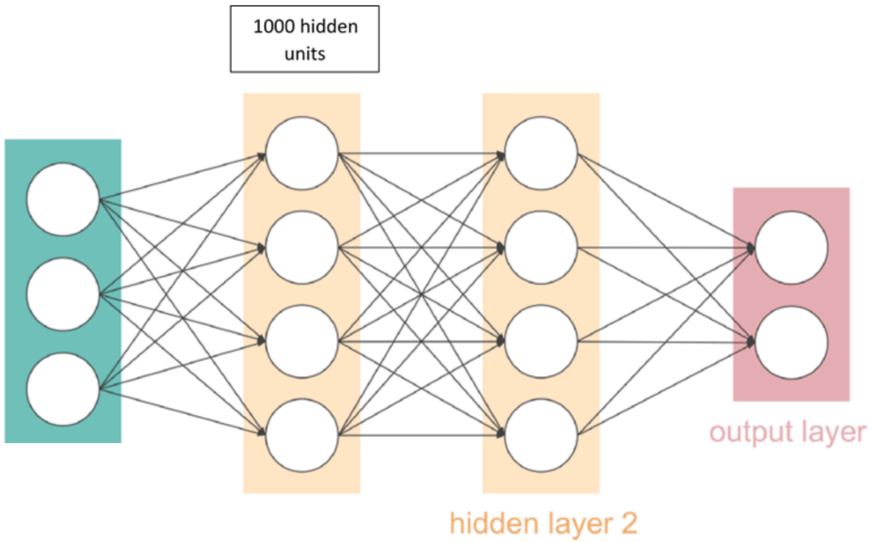
- Original values of a **24-bit color** images (True Color):
  - 8-bit per color: 0 – 255.
  - Total: 256 * 256 * 256 = 16,777,216 colors
- Value for red, green, and blue.
- Preprocessing color values: **normalized** between 0 and 1
  -> will increase performance

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.

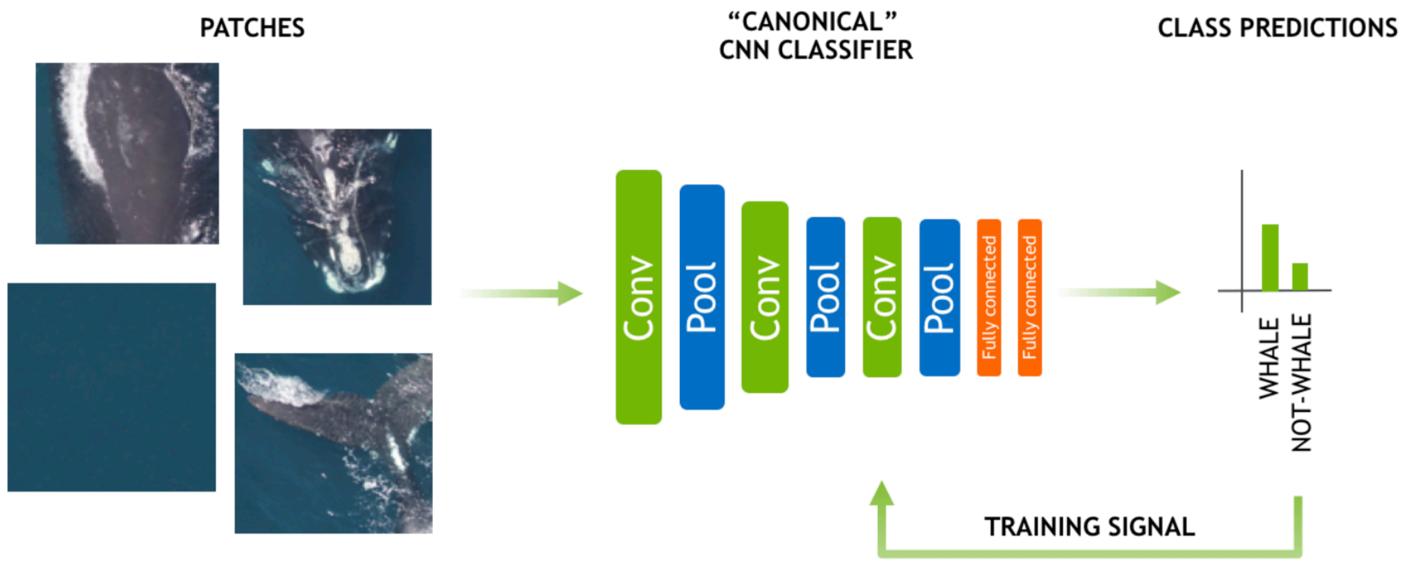
  - High chance of overfitting and highly complex network



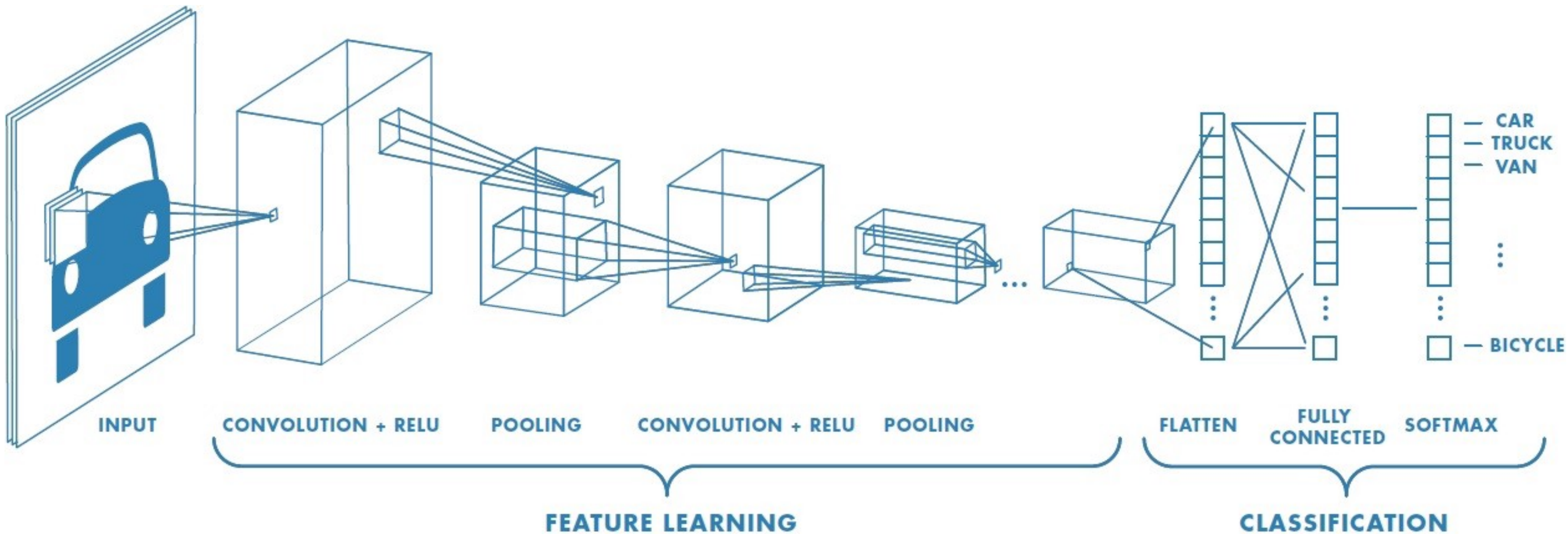
https://blog.datawow.io/interns-explain-cnn-8a669d053f8b

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.



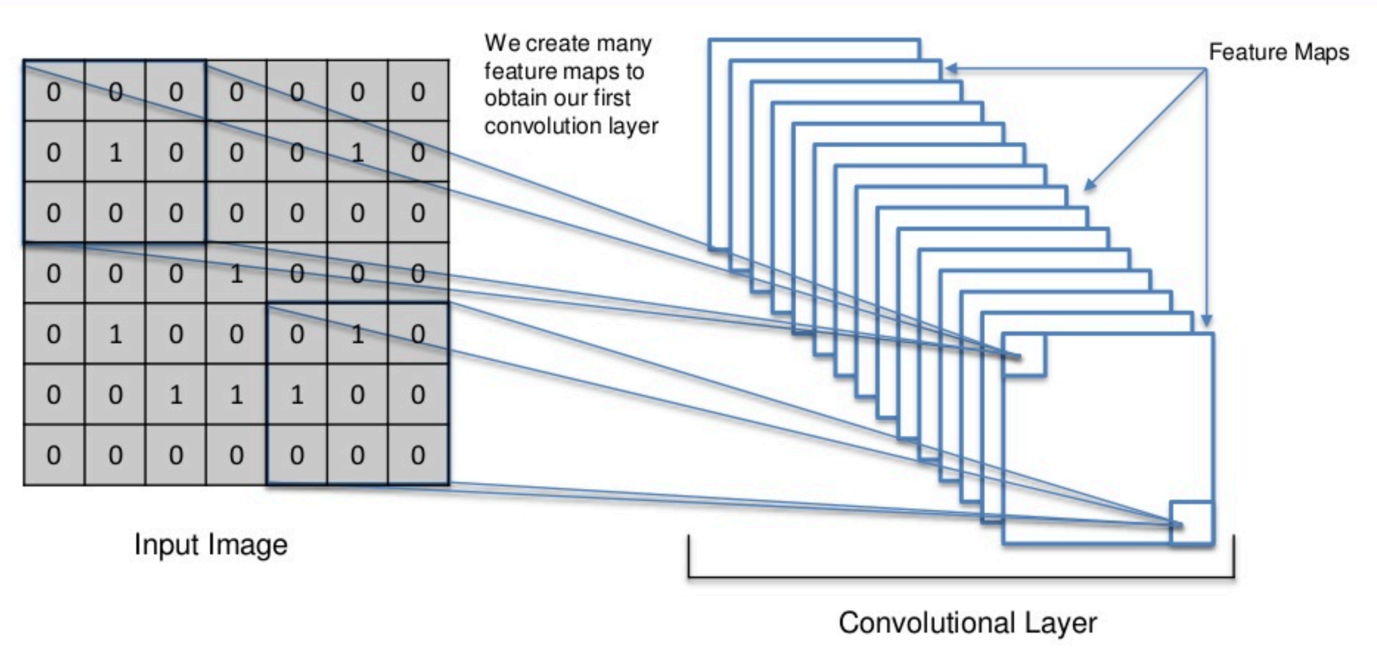
What goes on in a typical CNN?



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

Convolutional layer

- Many feature maps are created, using filters (also called kernels).
- Kernels 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 are the filter dimentions, i and j the positions in the resulting activation map

Convolution

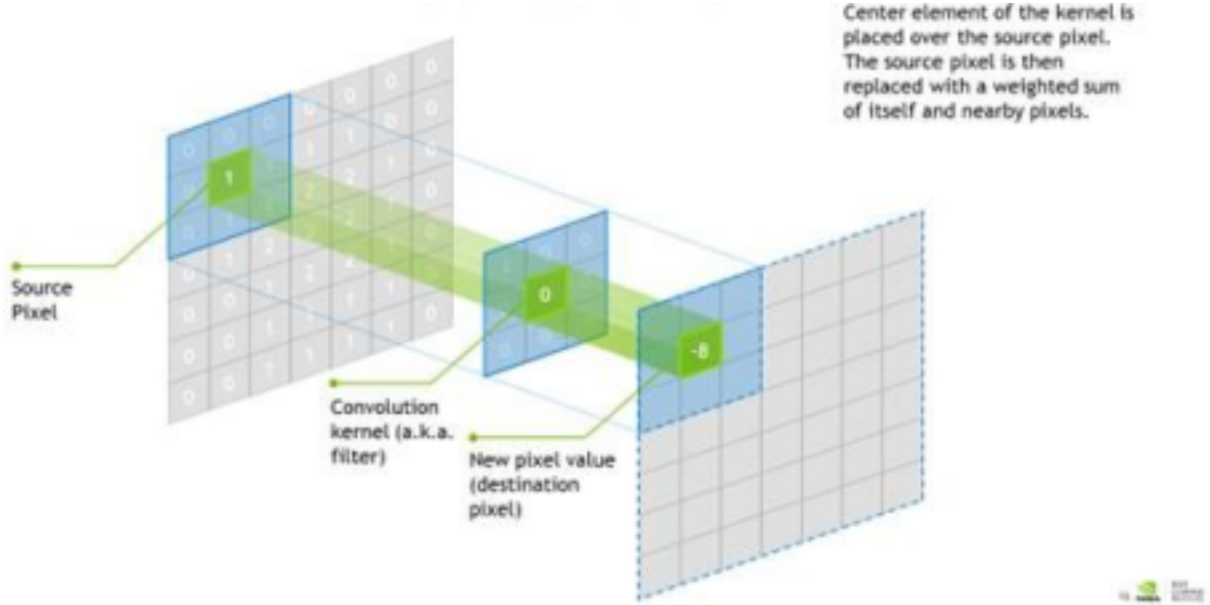


Image source

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

=

Convolution

- Example: edge detection filter/kernel

$$
\underbrace{\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix}}_{6 \text{ x } 6} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\substack{3 \text{ x } 3 \\ \text{filter}}} = \begin{bmatrix} \, & \, & \, & \, & \, & \, \\ \, & \, & \, & \, & \, & \, \\ \, & \, & \, & \, & \, & \, \\ \, & \, & \, & \, & \, & \, \\ \, & \, & \, & \, & \, & \, \\ \, & \, & \, & \, & \, & \, \end{bmatrix}
$$

Convolution

- Example: edge detection filter/kernel

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

$$\begin{array}{|c|c|c|c|c|c|}
\hline
3 & 0 & 1 & 2 & 7 & 4 \\ \hline
1 & 5 & 8 & 9 & 3 & 1 \\ \hline
2 & 7 & 2 & 5 & 1 & 3 \\ \hline
0 & 1 & 3 & 1 & 7 & 8 \\ \hline
4 & 2 & 1 & 6 & 2 & 8 \\ \hline
2 & 4 & 5 & 2 & 3 & 9 \\ \hline
\end{array}
\ast
\begin{array}{|c|c|c|}
\hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
\end{array}
=
\begin{array}{|c|c|c|c|}
\hline
\boxed{-5} & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
\end{array}$$

6 x 6

3 x 3
filter

4 x 4

Convolution

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

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
3 & 0 & 1 & 2 & 7 & 4 \\ \hline
1 & 5 & 8 & 9 & 3 & 1 \\ \hline
2 & 7 & 2 & 5 & 1 & 3 \\ \hline
0 & 1 & 3 & 1 & 7 & 8 \\ \hline
4 & 2 & 1 & 6 & 2 & 8 \\ \hline
2 & 4 & 5 & 2 & 3 & 9 \\ \hline
\end{array}
\;*\;
\begin{array}{|c|c|c|}
\hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
\end{array}
\;=\;
\begin{array}{|c|c|c|c|}
\hline
-5 & \boxed{-4} & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
\end{array}
$$

6 x 6

3 x 3 filter

4 x 4

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	

4 x 4

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

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

=

4 x 4

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

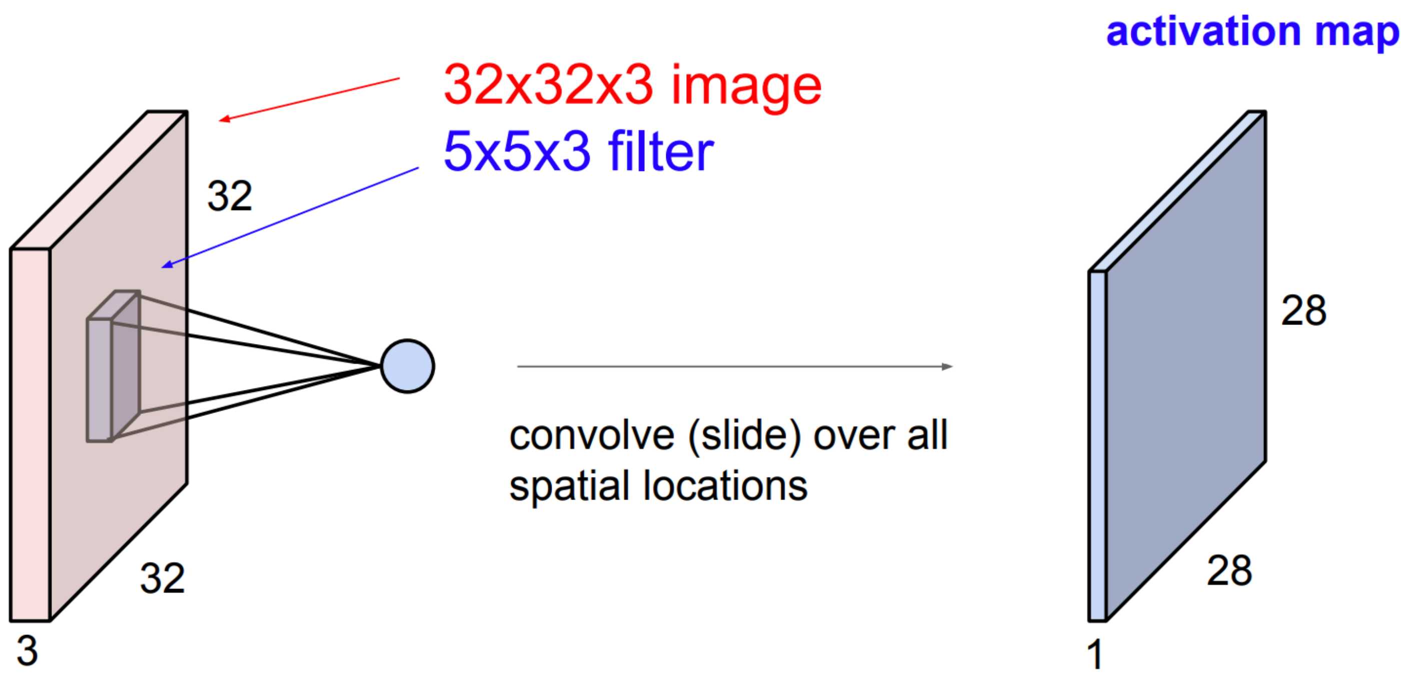
3 x 3
filter

=

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

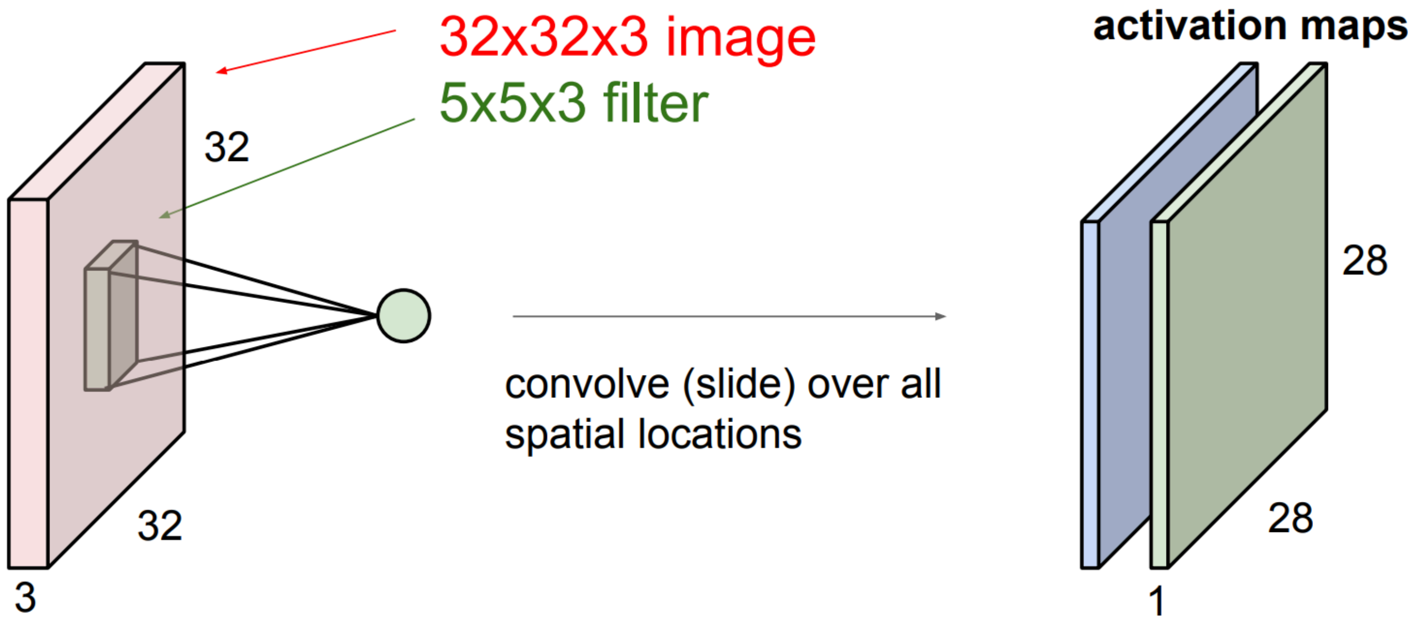
4 x 4

Activation maps



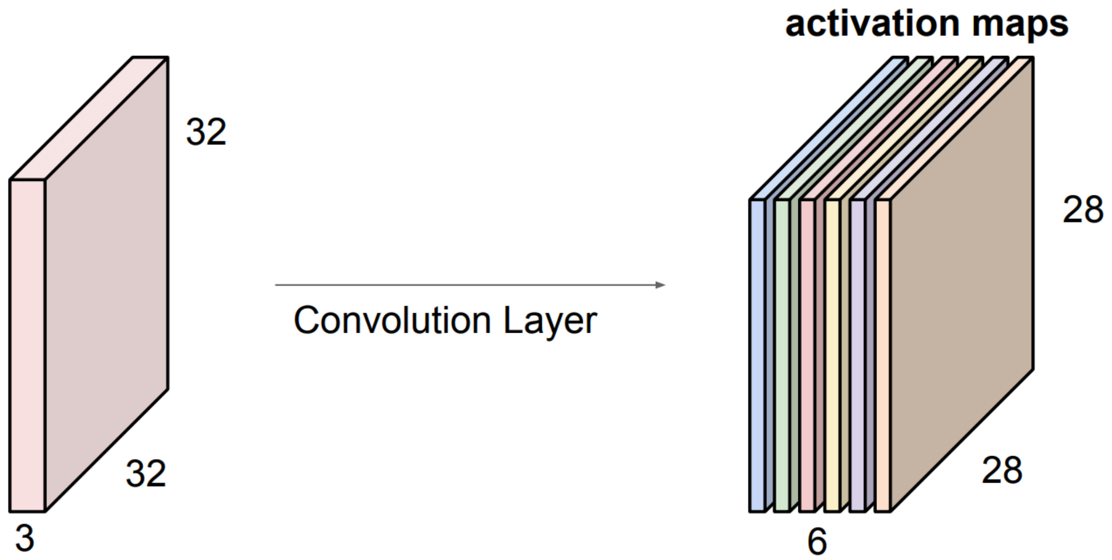
Activation maps

- Each filter creates an activatio map



Activation maps

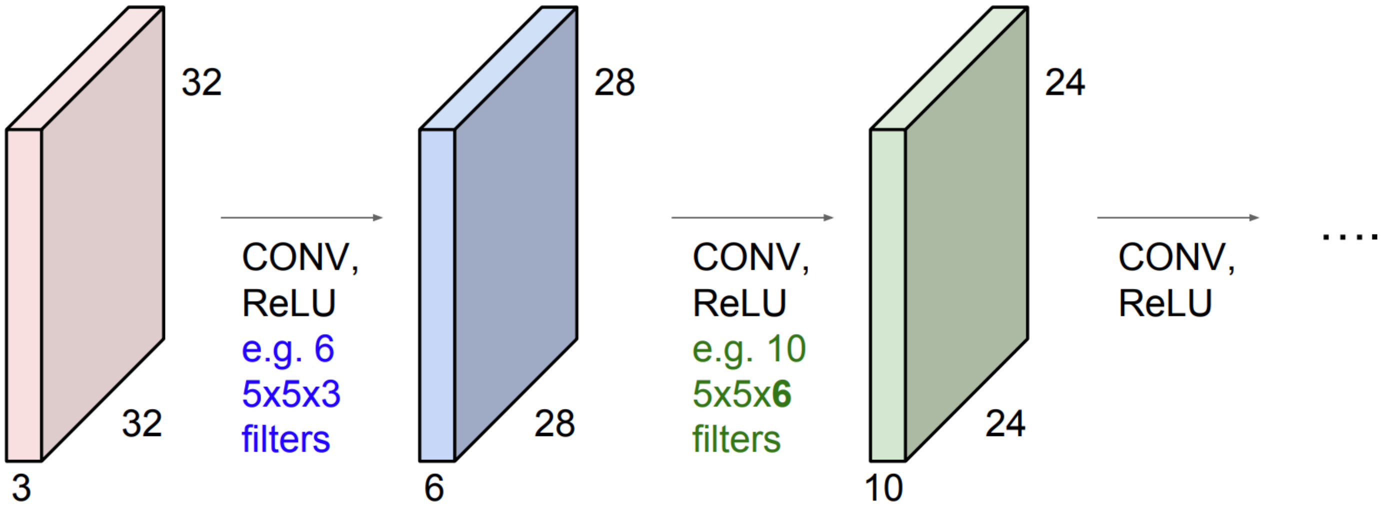
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

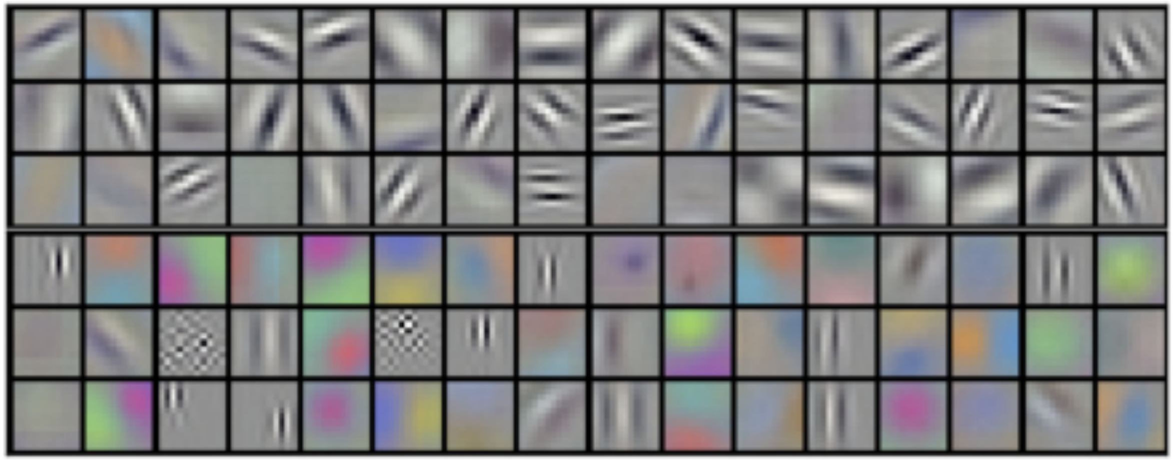
ConvNet

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



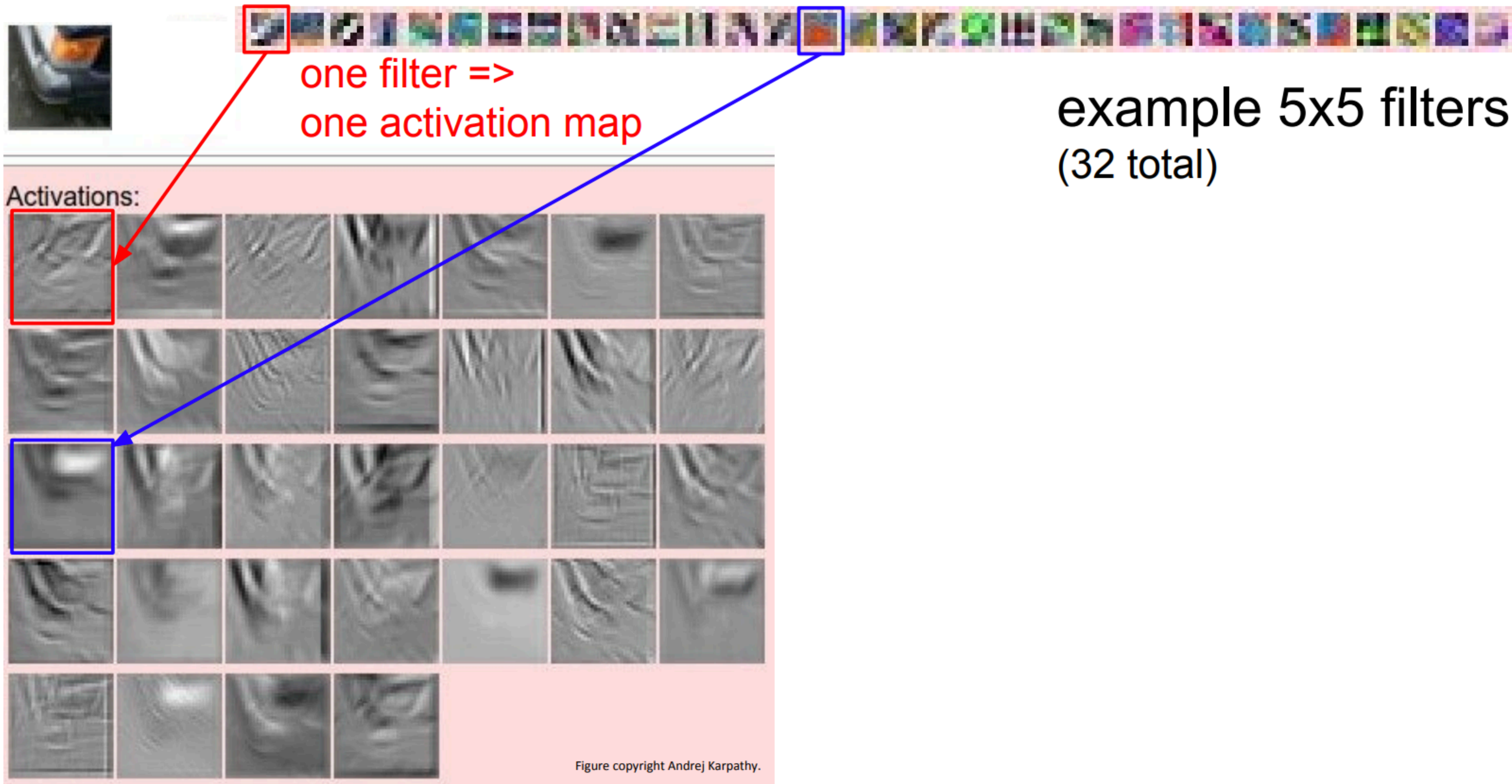
Convolution

- Typical filters that are learned:

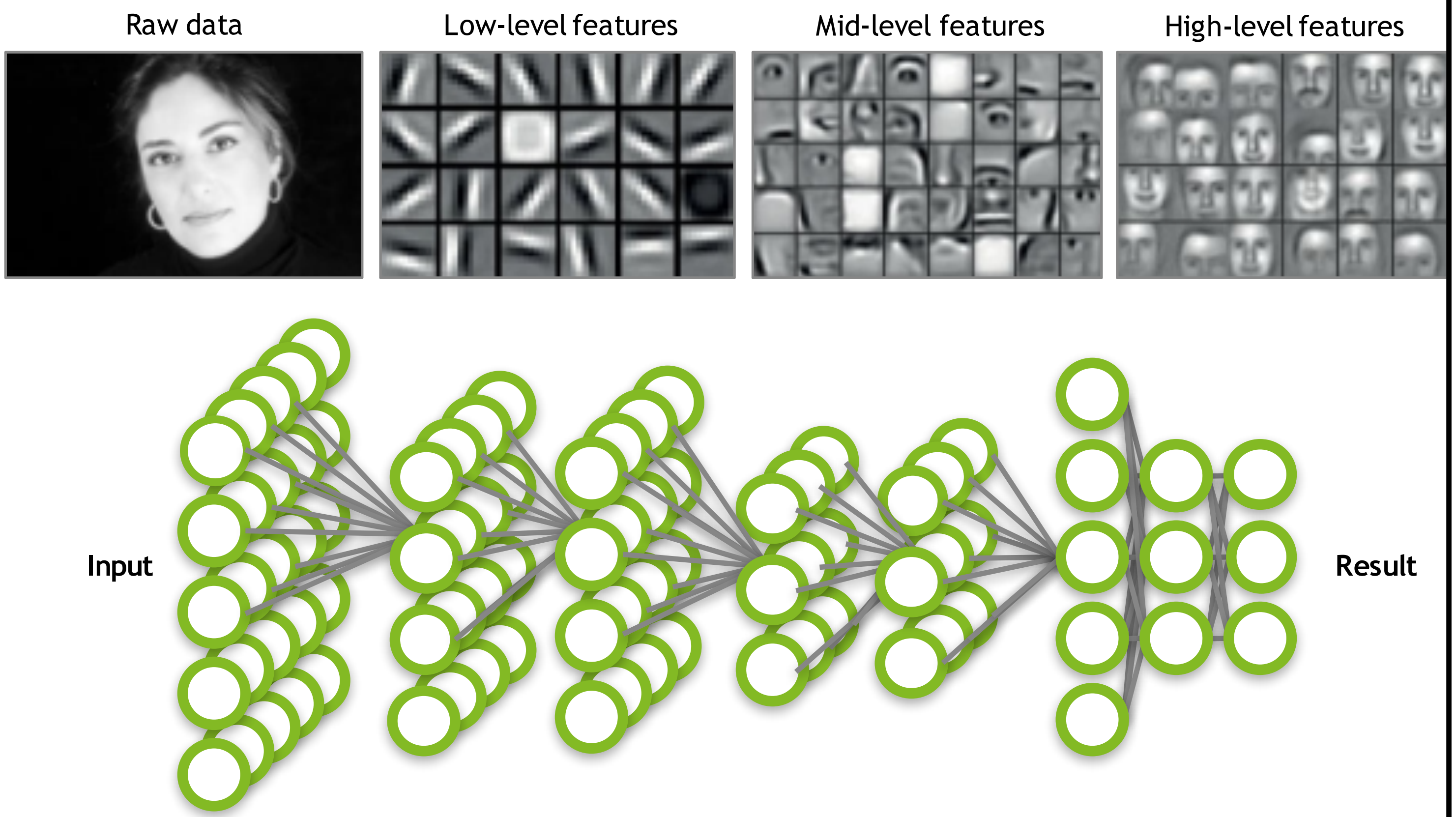


(Krizhevsky et al., 2012)

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


one filter =>
one activation map
example 5x5 filters
(32 total)
Activations:
Figure copyright Andrej Karpathy.

Different levels of features



Application components:

Task objective
e.g. Identify face

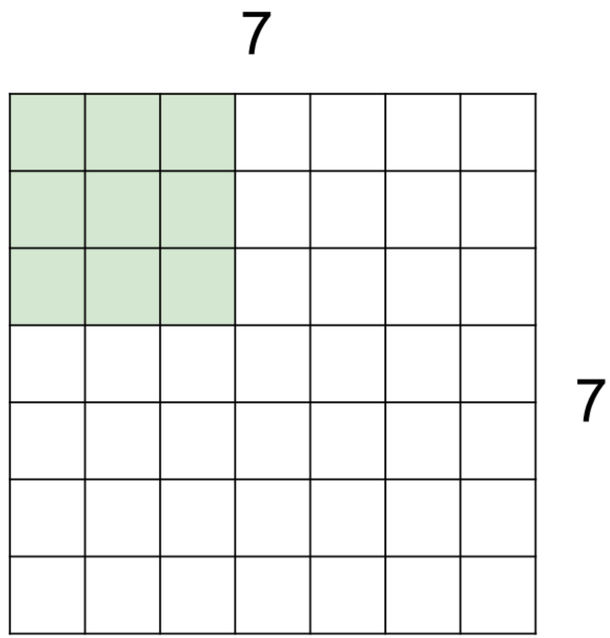
Training data
10-100M images

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

Learning algorithm
~30 Exaflops
1-30 GPU days

A closer look at dimensions

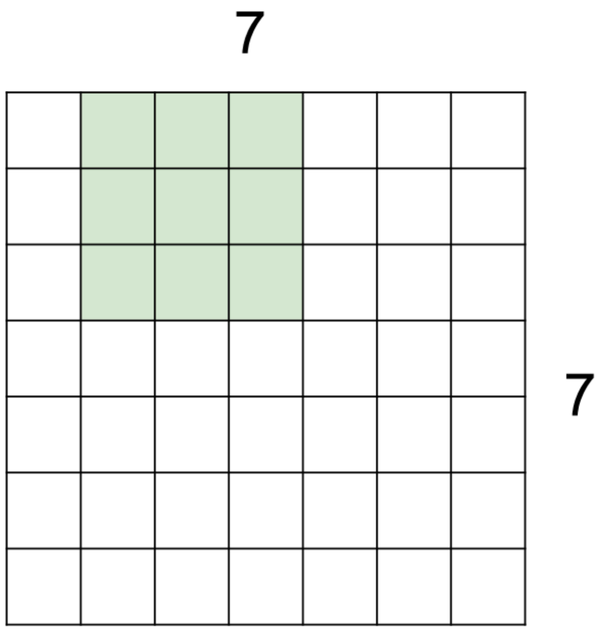
- A 7x7 input with a 3x3 filter:



A closer look at dimensions

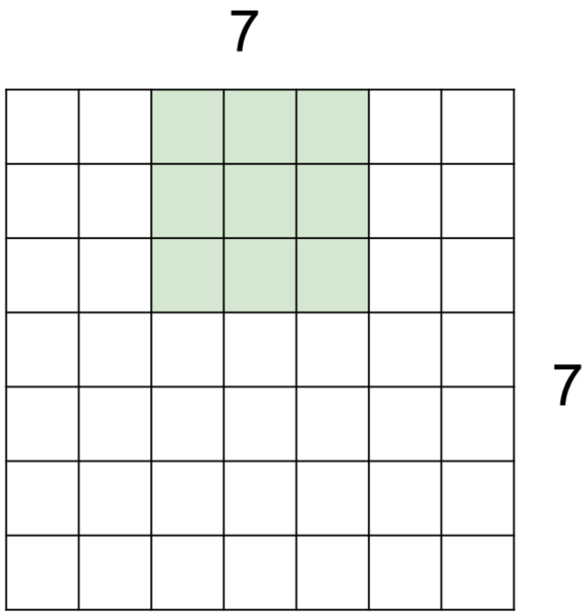
- A 7x7 input with a 3x3 filter:

Stride = 1



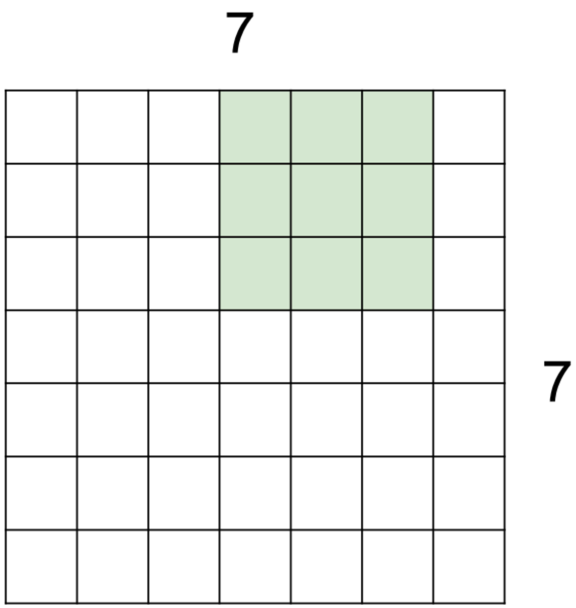
A closer look at dimensions

- A 7x7 input with a 3x3 filter:



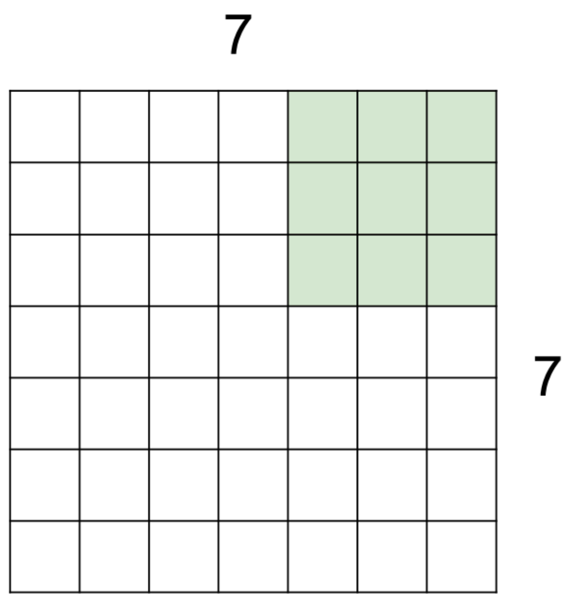
A closer look at dimensions

- A 7x7 input with a 3x3 filter:



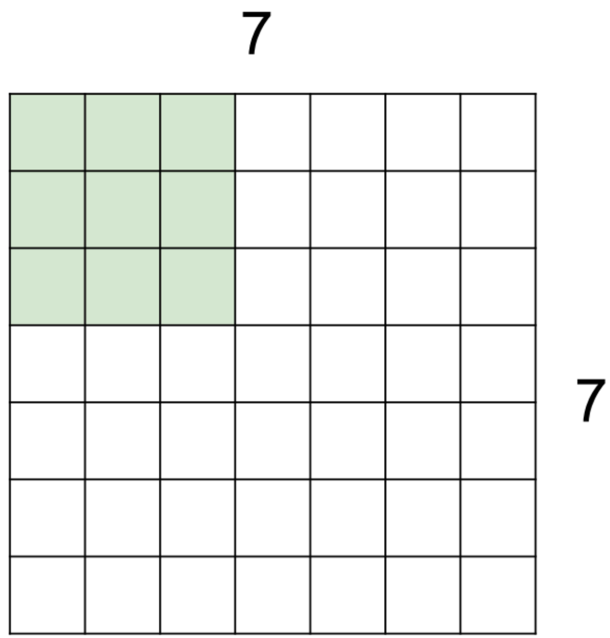
A closer look at dimensions

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



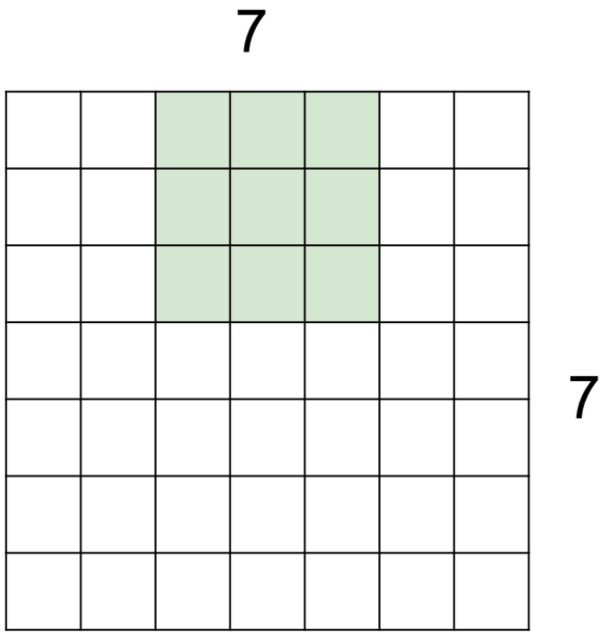
A closer look at dimensions

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



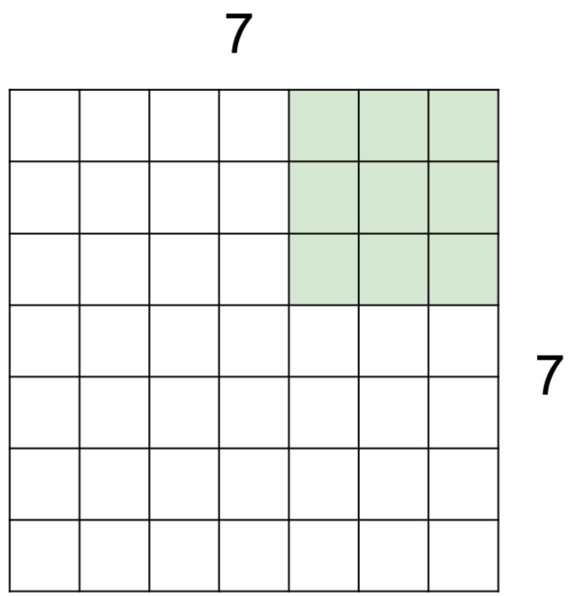
A closer look at dimensions

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



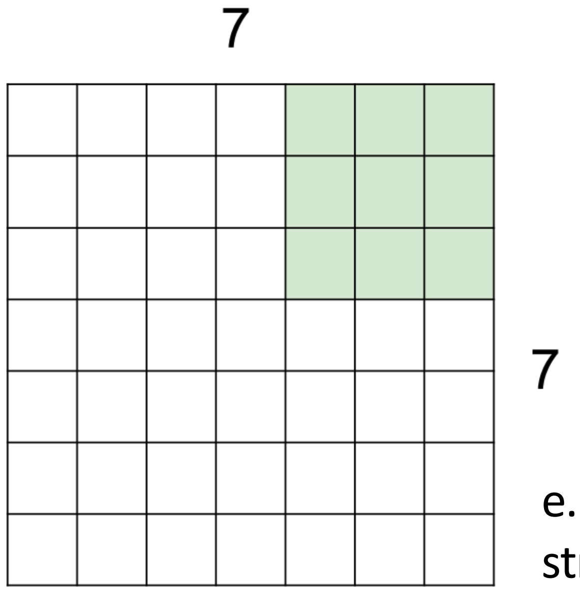
A closer look at dimensions

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



A closer look at dimensions

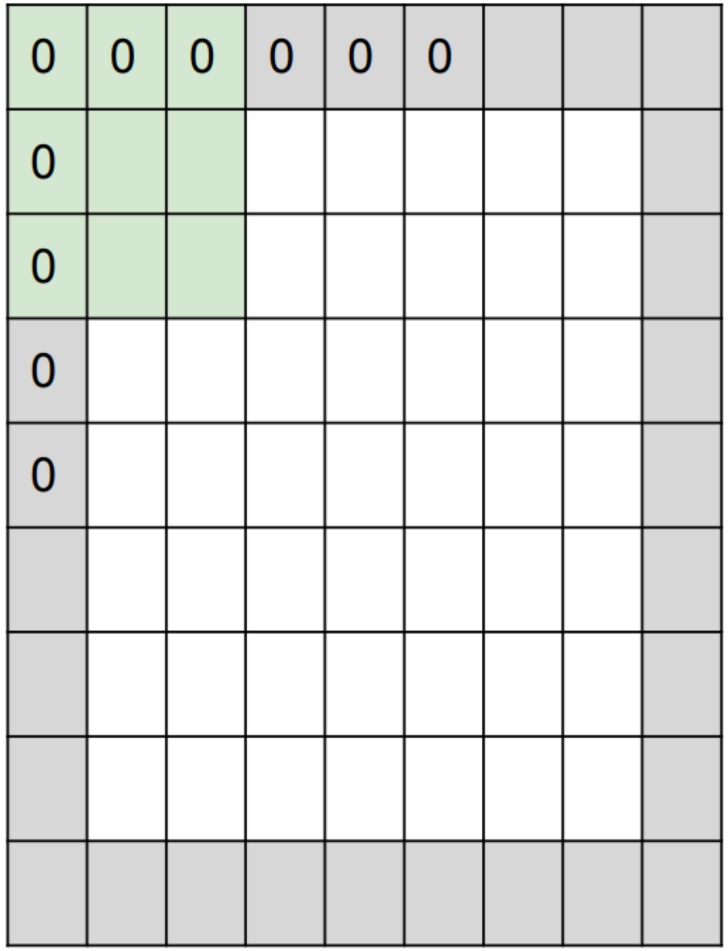
- A 7x7 input **with stride 3??** What is the output size?
- [(N - F) / stride] + 1



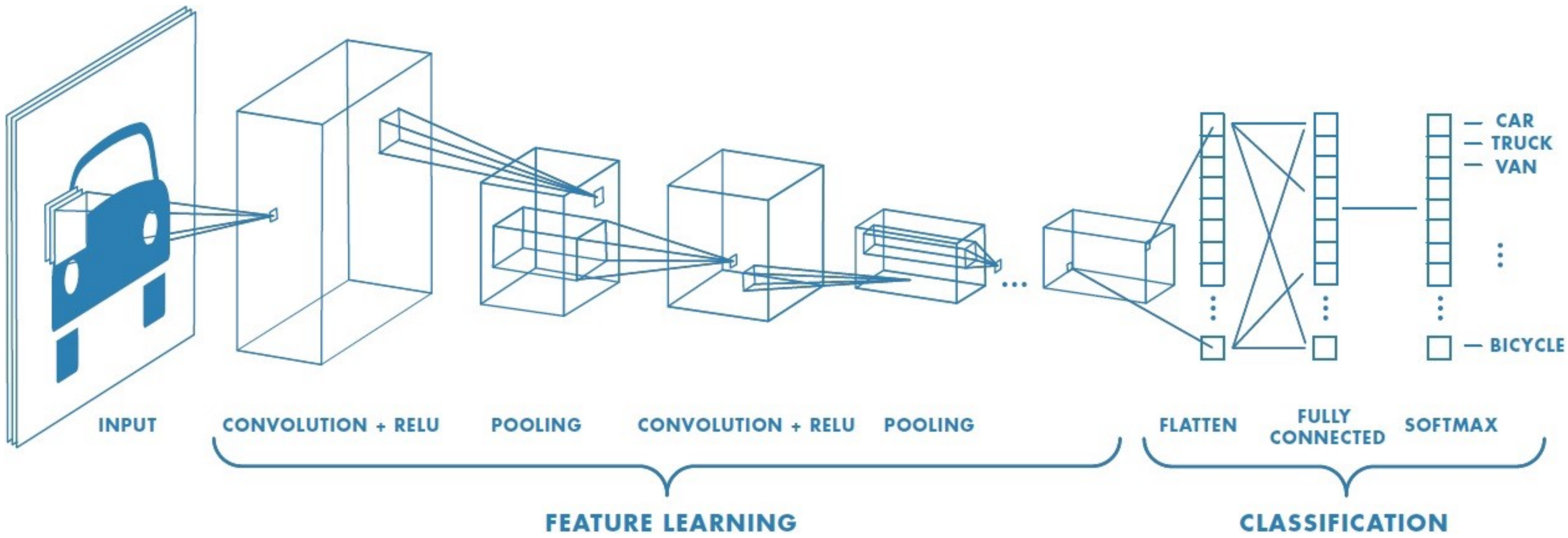
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

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)



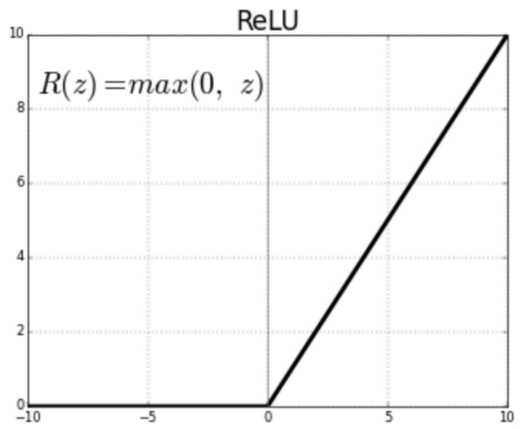
After convolution



Non-linear activation

- Tanh, sigmoid, or ReLu activation function
- Most popular (often performs best): **ReLu** $f(x) = \max(0, x)$

  => **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.

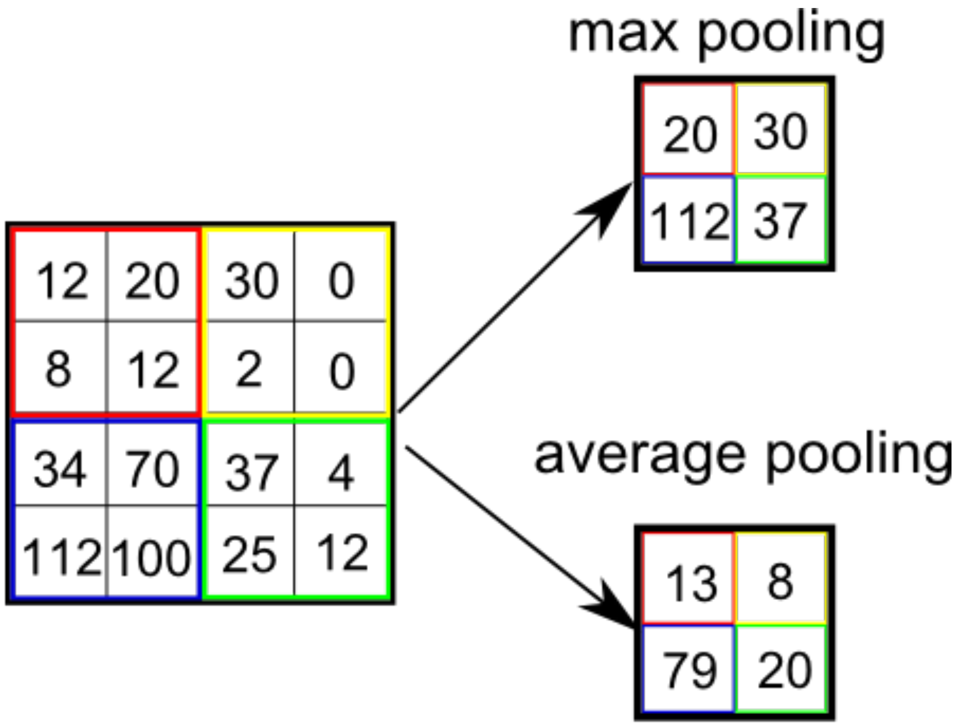


Pooling

- 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

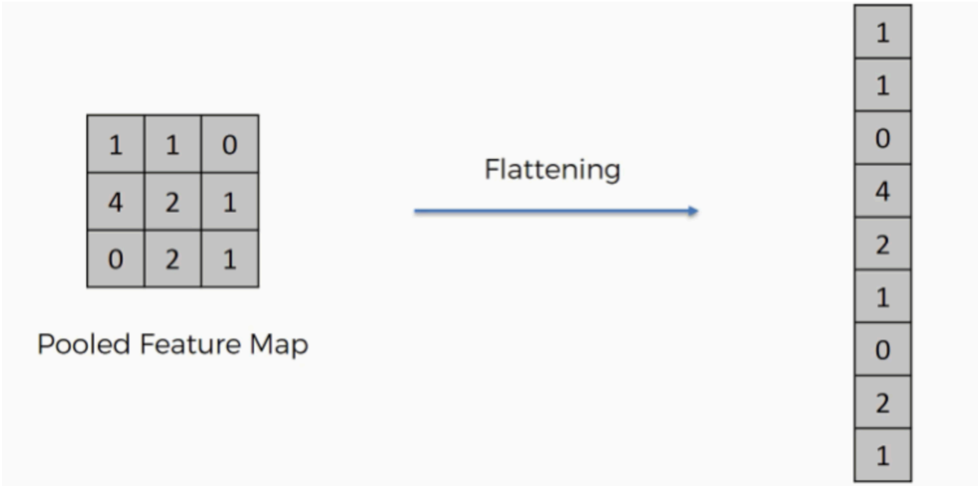
- Hyperparameters:
  - Stride size
  - Pooling window size



- 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

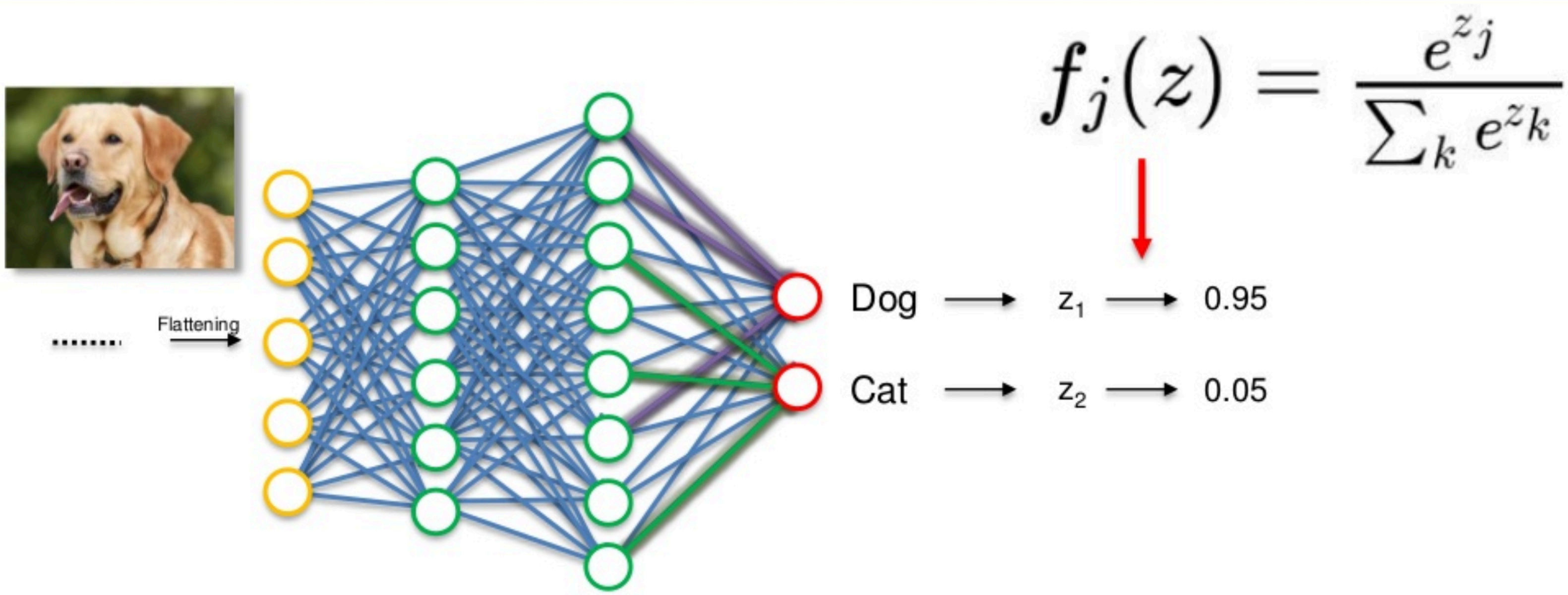


Fully connected layer

- 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-valued. (could be mean squared error: mse)

Softmax



Keras for CNNs

- Making sure the images are fed with a fixed image width & height. Generators are used to feed the images in batches from a directory (rescaled if needed)

```
train_generator = datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='binary')
```

Keras for CNNs

- Sequential model means the layers are stacked

```
model = Sequential()
```

- For each convolutional layer:

```
model.add(Convolution2D(32, (3, 3), input_shape=(img_width, img_height,3)))
ze (3,3)
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

32 kernels of size 3 x 3 are used, input size is width x height x 3 (color images), pooling size is 2 x 2, and

- The last layers are fully connected, dense layers:

```
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

Keras for CNNs

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

```
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

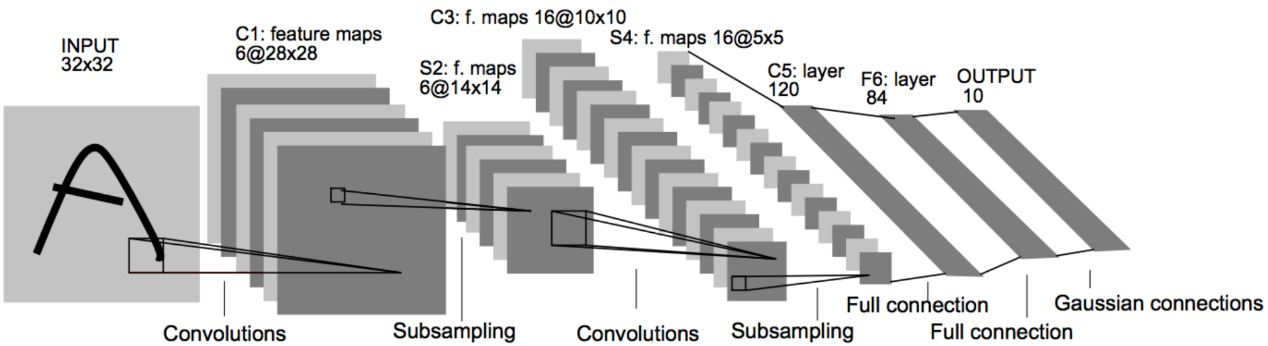
- And model training:

```
model.fit_generator(
        train_generator,
        steps_per_epoch=train_samples // batch_size,
        epochs=epochs,
        callbacks=[history],  # save the history so that we can plot it later
        validation_data=validation_generator,
        validation_steps=validation_samples// batch_size,)
```

Famous prebuilt networks

LeNet-5

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

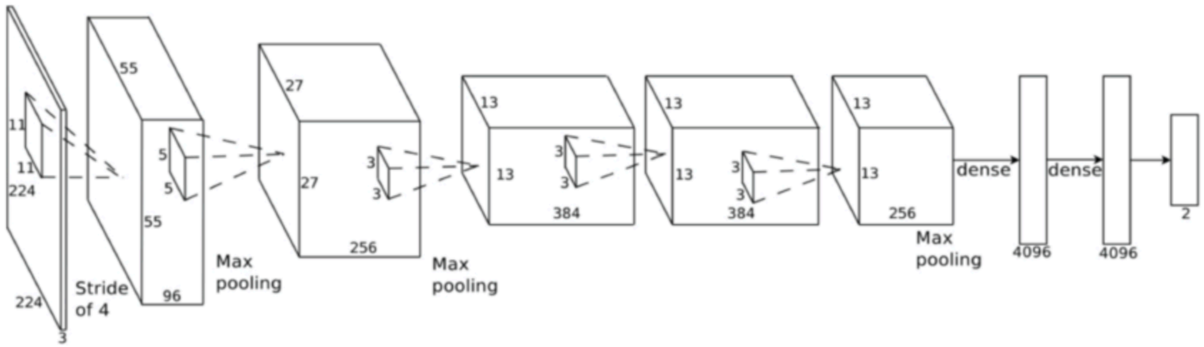


- Convolutional layers use a subset of the previous layer's channels for each filter to reduce computation and force a break of symmetry in the network.
- Subsampling is Average Pooling with learnable weights per feature map.
- Parameters: 60,000

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)



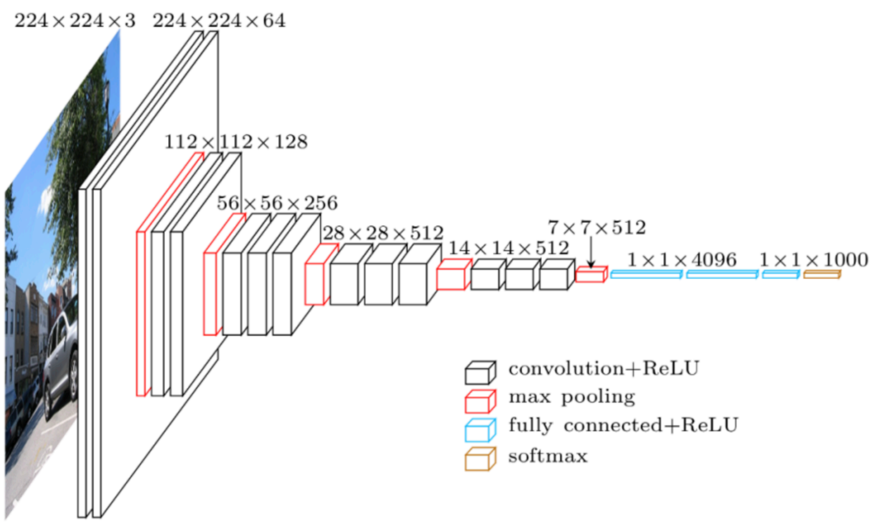
- The general architecture is quite similar to LeNet-5, although this model is considerably **larger**. The success of this model convinced a lot of the computer vision community to take a serious look at deep learning for computer vision tasks.

- 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
- Wxtremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end
- Parameters: 138 million

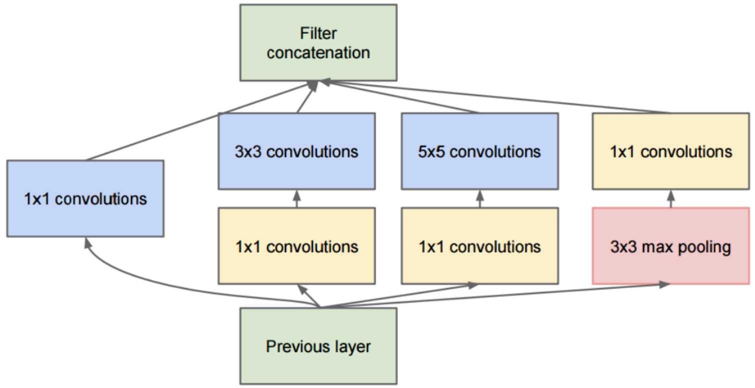


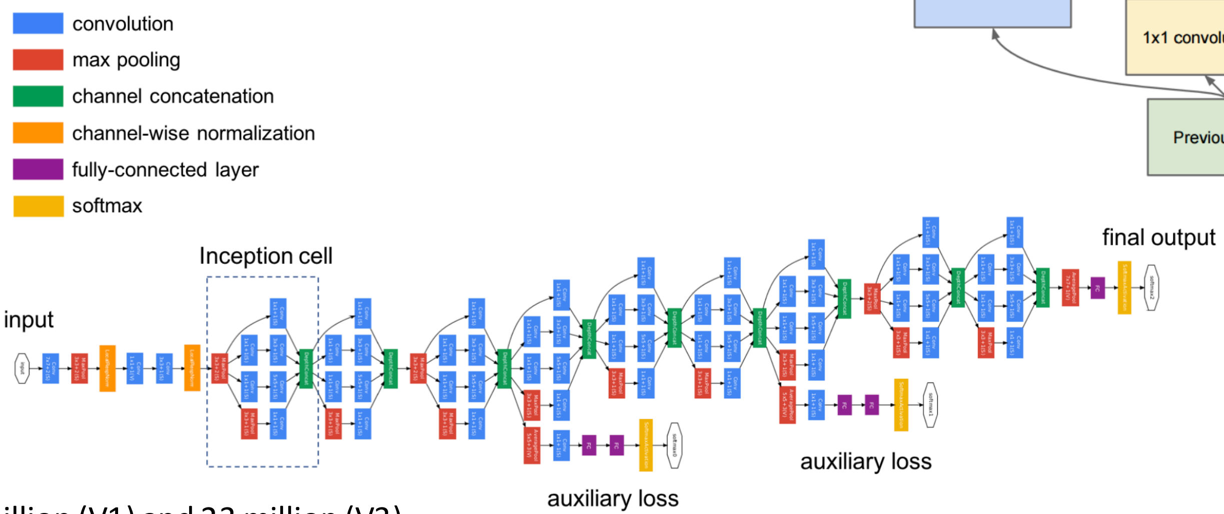

```
model_vgg = applications.VGG16(include_top=False, weights='imagenet')
```

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.



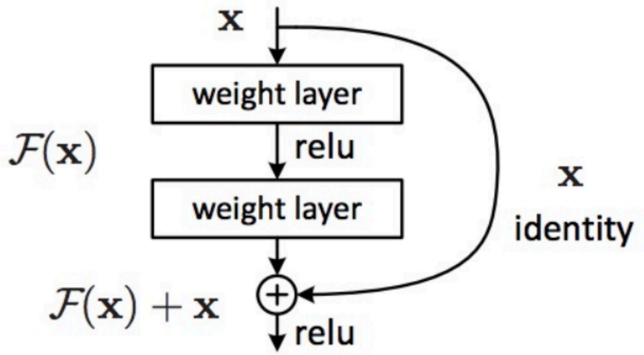


Parameters: 5 million (V1) and 23 million (V3)

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.



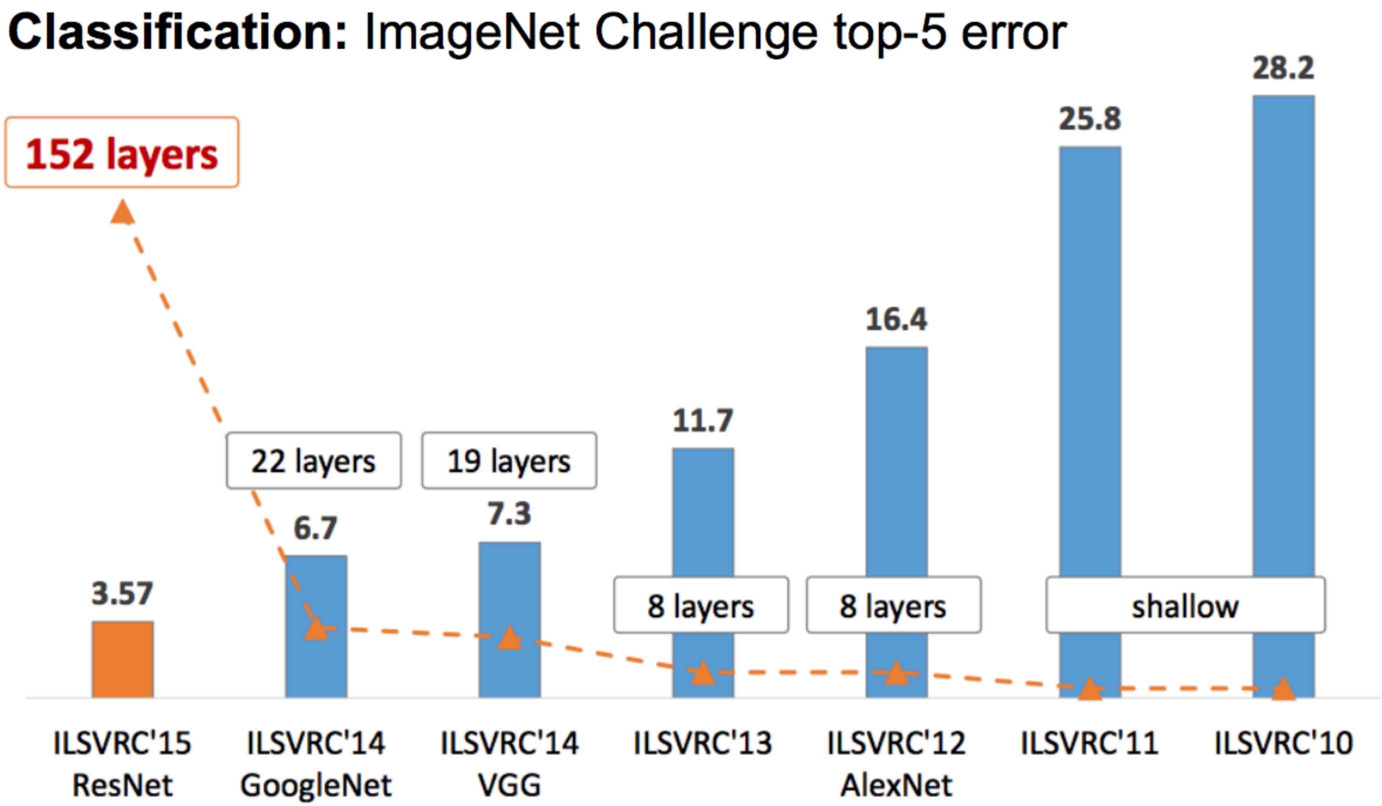
ResNet

- 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

Object Detection: PASCAL VOC mean Average Precision (mAP)

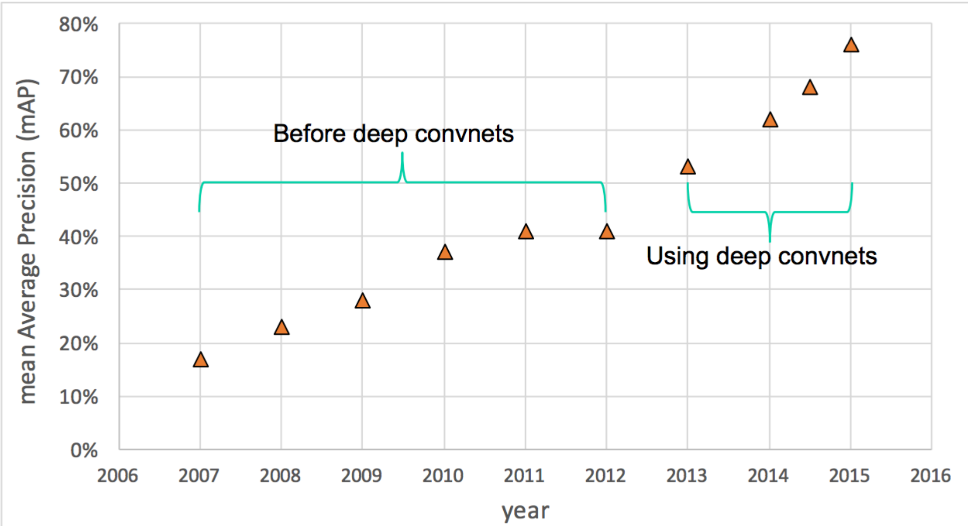
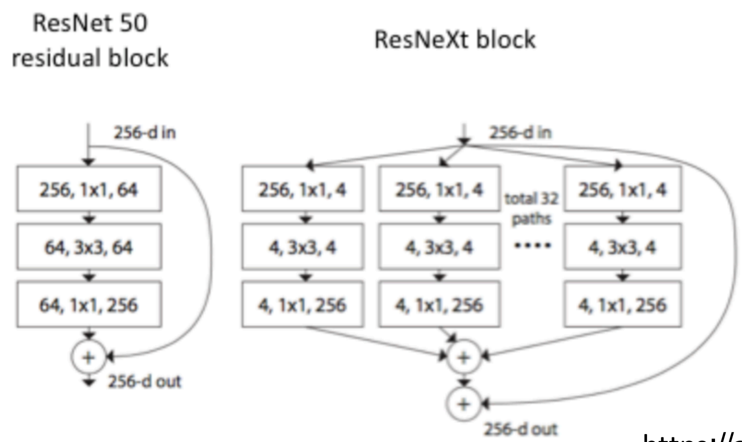


Figure source: Ross Girshick

ResNeXt

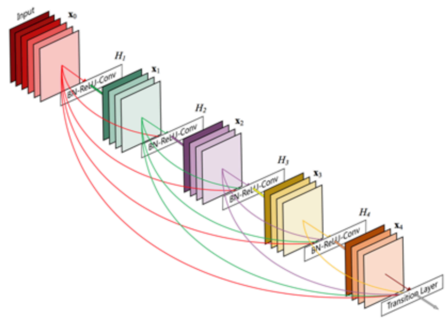
- Extension of ResNet, which replaces the standard residual block with one that leverages a "**split-transform-merge**" used in the Inception models:
  - Instead of performing convolutions over the full input feature map, the block's input is projected into a series of lower (channel) dimensional representations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

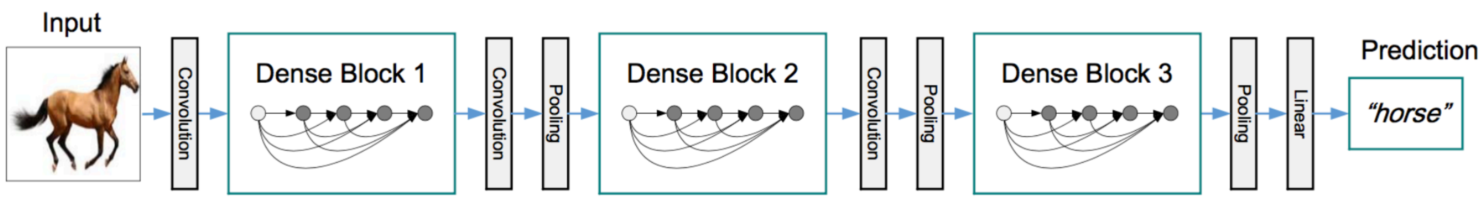


https://arxiv.org/abs/1611.05431

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)





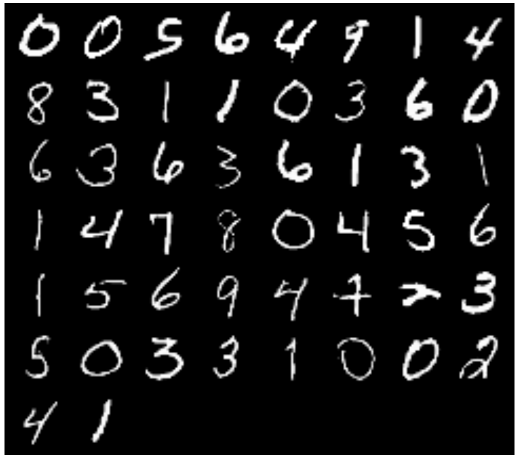
https://arxiv.org/abs/1608.06993

Hello world

MNIST APPLICATION

Handwritten digit recognition

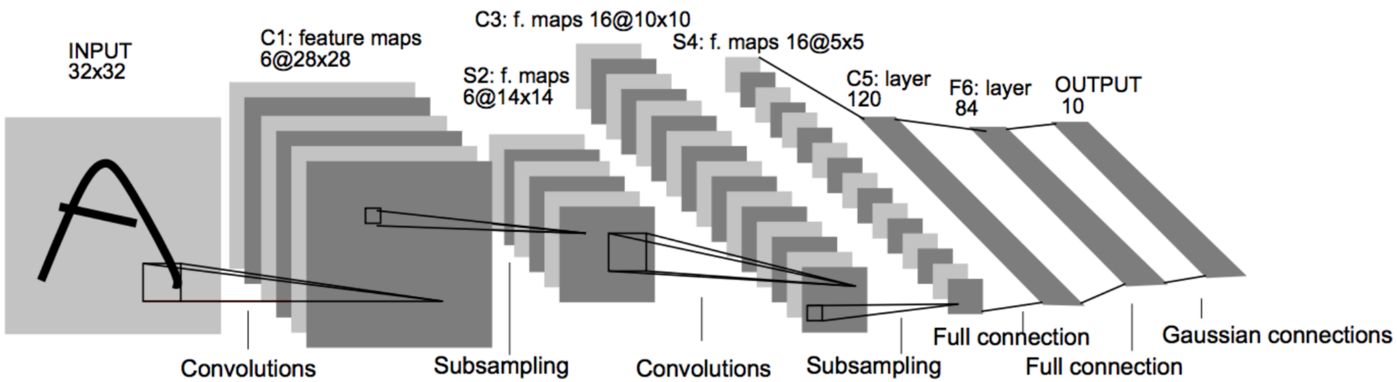
- MNIST data set of handwritten digits from Yann LeCun's website
- All images are 28x28 grayscale
  - Pixel values from 0 to 255
- 60K training examples / 10K test examples
- Input vector of size 784
  - 28 * 28 = 784
- Output value is integer from 0-9



Slides inspired by the NVIDIA SUTD ambassador workshop

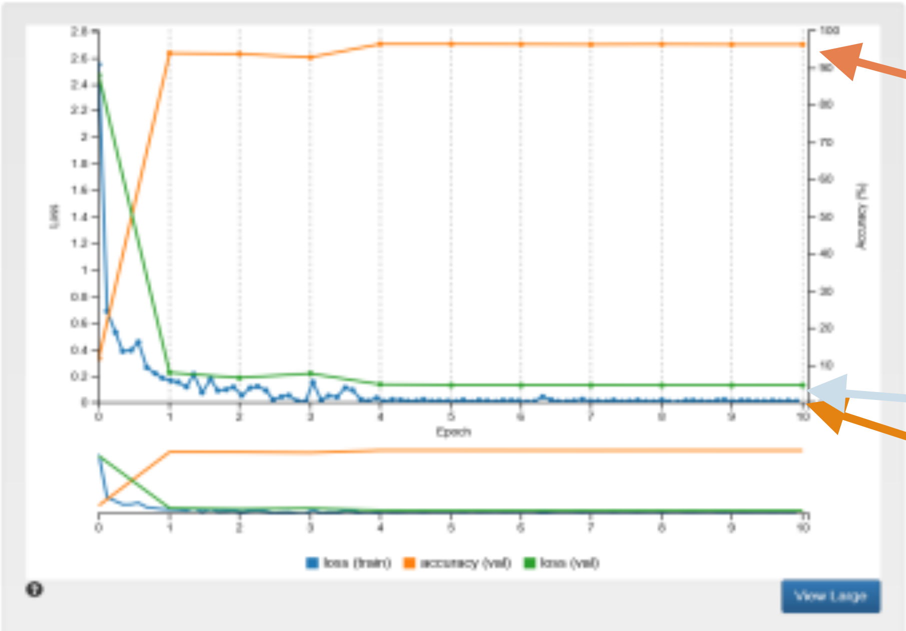
LeNet for MNIST

- 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 function (Training)

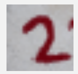
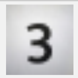
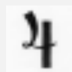
First result

Predictions

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

First result

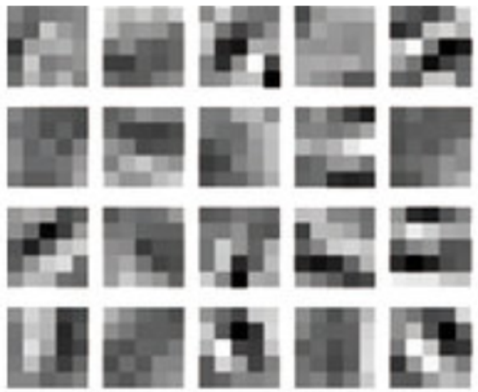
- Training done within minutes
- 96% classification accuracy
- Promising! Let’s train on the complete dataset…

	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

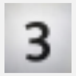
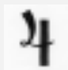
- Now that we have tested our model, let’s run it on the complete dataset, 4x larger
- 60k images for training, 10k test

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

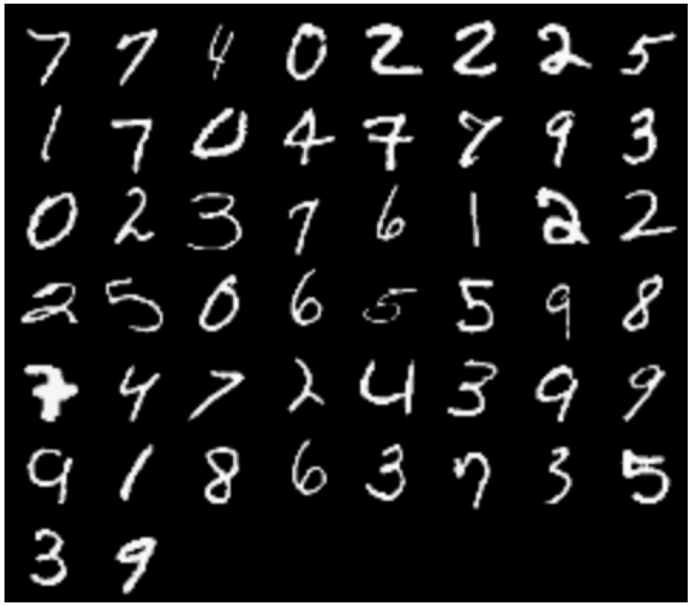
- For our seven test images that the backgrounds are not uniform. In addition, most of the backgrounds are light in color whereas our training data all have black backgrounds.
- We saw that increasing the amount of data did help for classifying the handwritten characters, so what if we include more data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

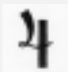
Data augmentation

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



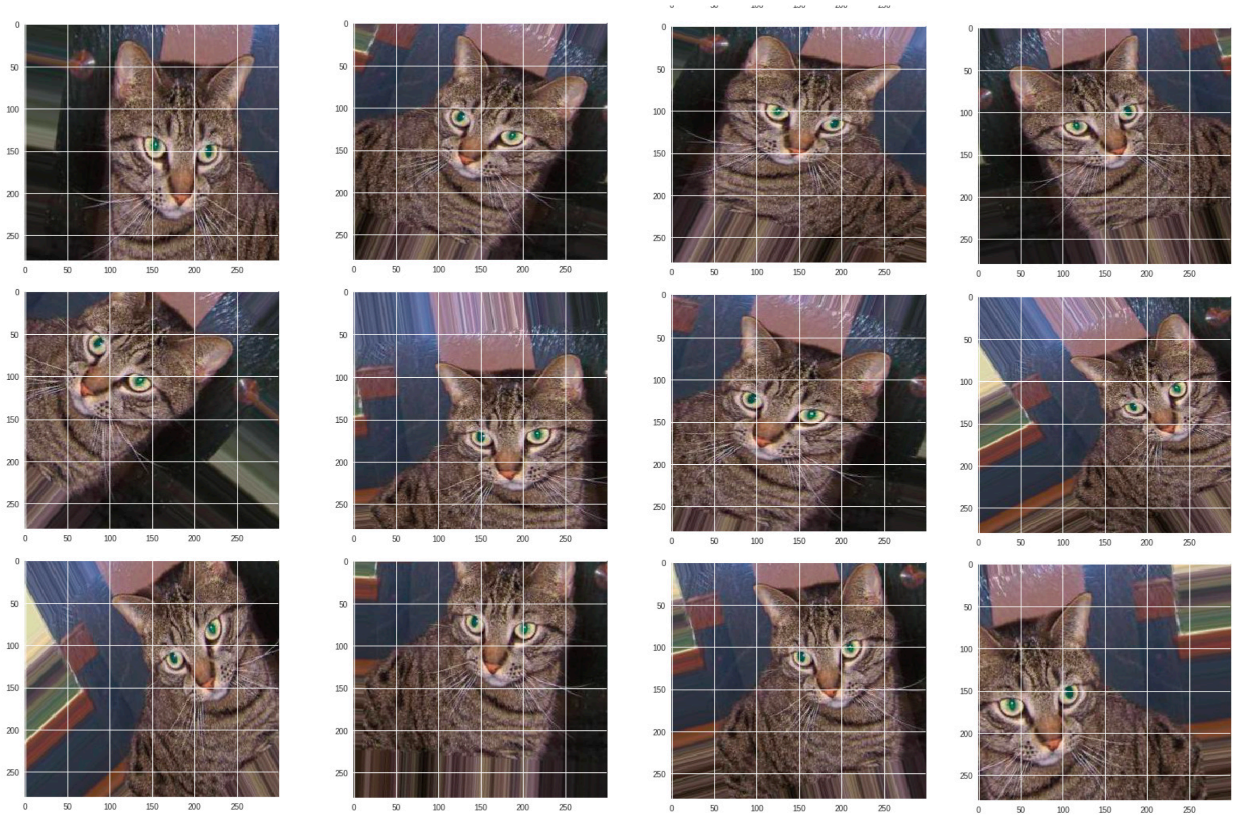
Results with augmentation

- Classification accuracy:

	SMALL DATASET	FULL DATASET	+INVERTED
	1 : 99.90 %	0 : 93.11 %	1 : 90.84 %
	2 : 69.03 %	2 : 87.23 %	2 : 89.44 %
	8 : 71.37 %	8 : 71.60 %	3 : 100.0 %
	8 : 85.07 %	8 : 79.72 %	4 : 100.0 %
	0 : 99.00 %	0 : 95.82 %	7 : 82.84 %
	8 : 99.69 %	8 : 100.0 %	8 : 100.0 %
	8 : 54.75 %	2 : 70.57 %	2 : 96.27 %

Data augmentation

- Other ways of augmenting images include: rotation, shift, rescale, zoom, flip,…



```
datagen = ImageDataGenerator(
          rotation_range=40,
          width_shift_range=0.2,
          height_shift_range=0.2,
          shear_range=0.2,
          zoom_range=0.2,
          horizontal_flip=True,
          fill_mode='nearest')
```

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
- …

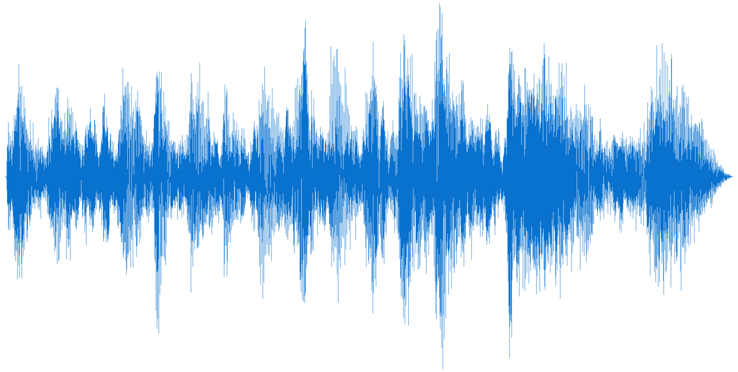
Audio

- 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:

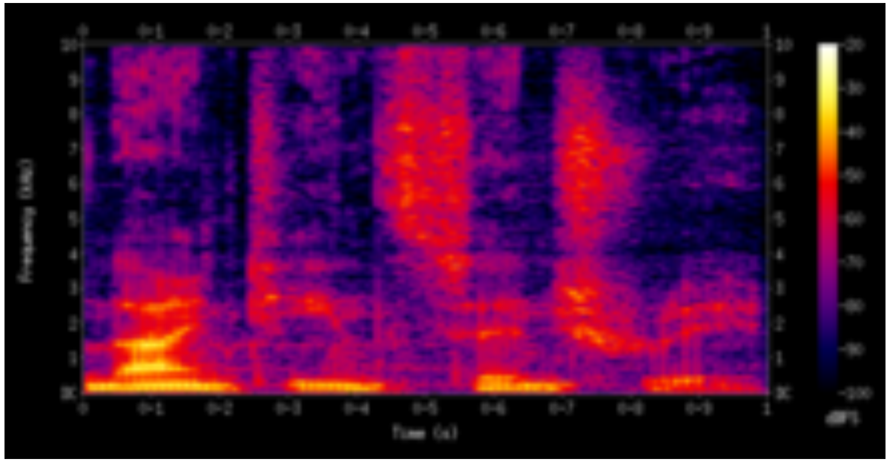


- More informative are spectrograms: a visual representation of the spectrum of frequencies of sound or other signal as they vary with time.

Spectrograms

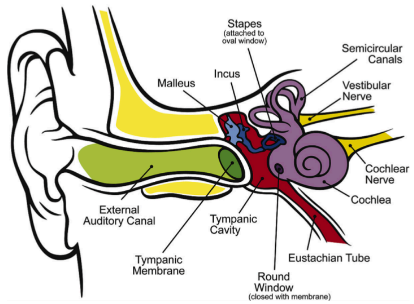
- Typical spectrogram of a few spoken words.
  - Y-axis: frequencies
  - X-axis: time
- Lower frequencies can be seen as more dense (brighter) here, as it's a male voice

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.



MFCCs

- 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.

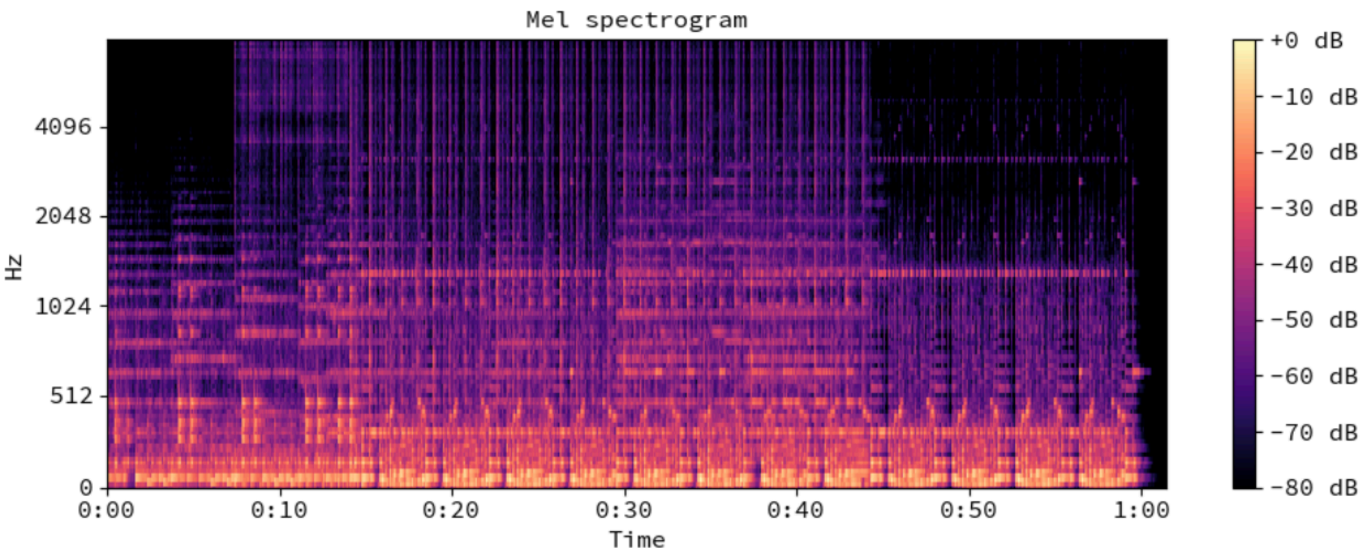


MFCCs

- 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

Spectrogram filters

- Example Mel Spectrogram



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

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

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

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

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

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

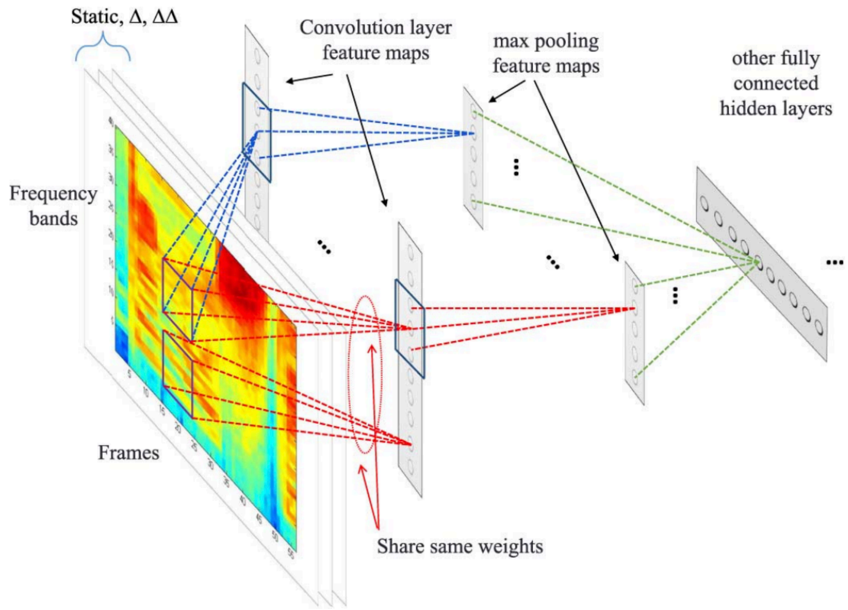
vargs:additional keyword arguments
Mel filter bank parameters. See `librosa.filters.mel` for details.

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

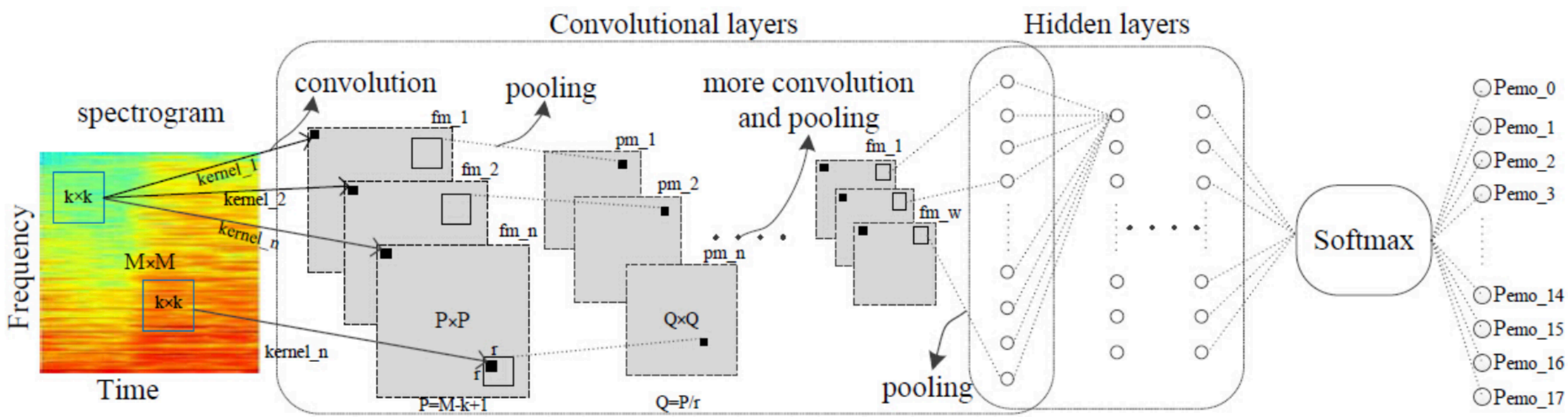
- With and without pre-training (with Restricted Bolzman Machines)
- CNN has:
  - one pair of convolution and pooling
  - two hidden fully connected layers
  - 84 feature maps per section
  - filter size of 8
  - pooling size of 6
  - a stride of 2
- Context window has 11 frames
- Compared to 3-layer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

	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

Music emotion classification

- Liu et al., 2017
- Dataset: CAL500exp
  - 3223 short segments labeled with 18 emotions tags
- 4 convolutional layers with 1 hidden layer



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

Audio signals

- ***Time*** signals
- Next week we will discuss recursive neural networks (RNNs and LSTMs) with a memory effect.

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/