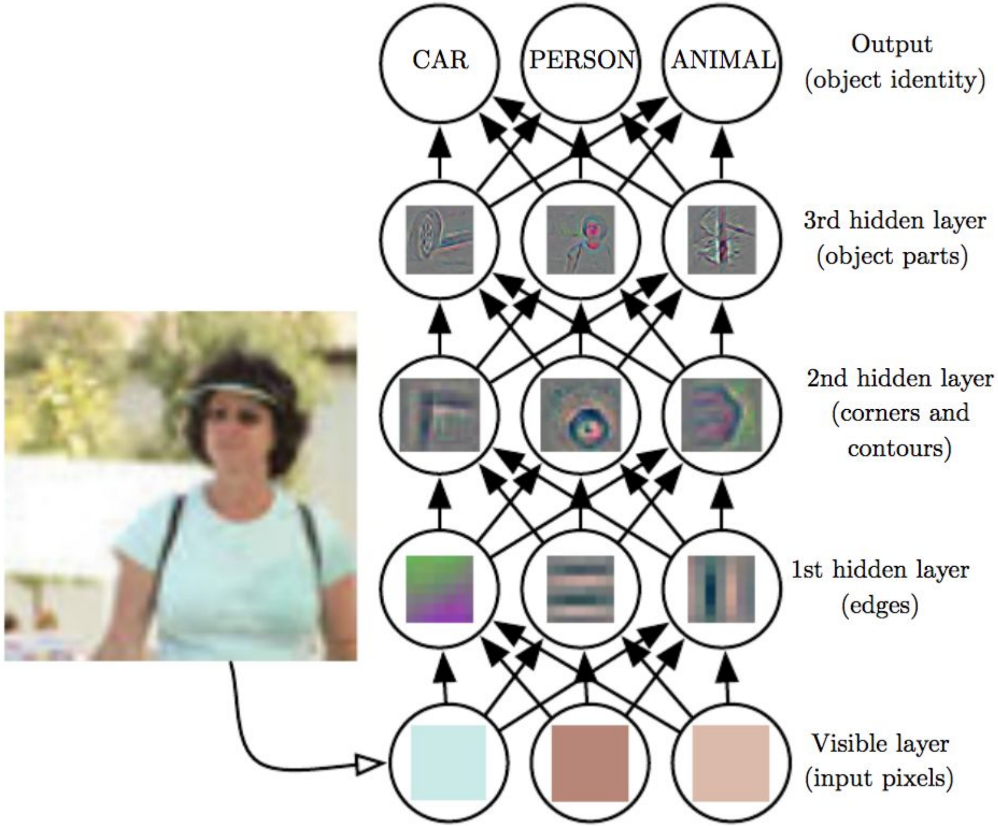
ConvNets

convolutional neural nets

Marcus Frean marcus@ecs.vuw.ac.nz

Feature learning



some well-known image data sets

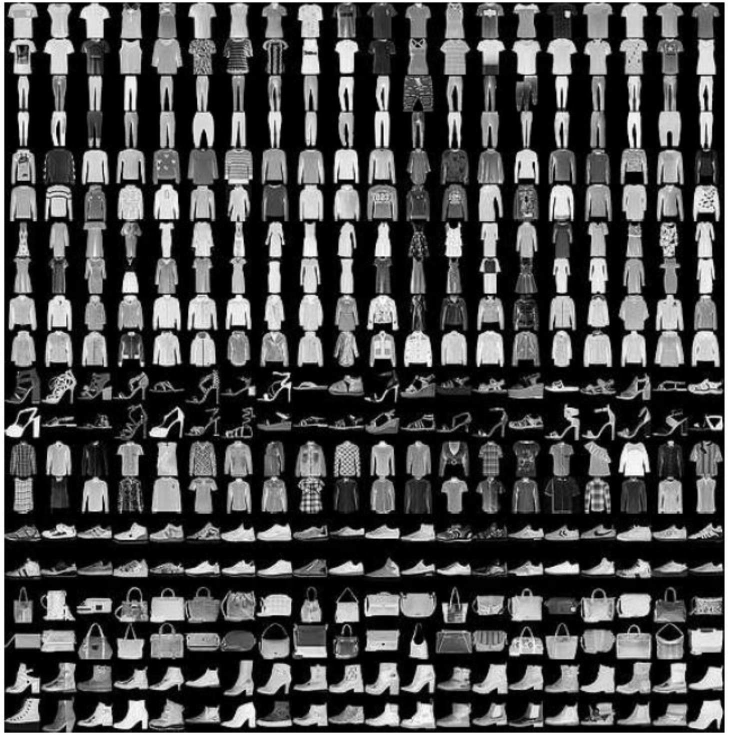
MNIST

- 10 classes
- 60,000 training images and 10,000 tes
  images, grayscale, of size 28 × 28
- ~60 MB
- Web: http://yann.lecun.com/exdb/mni

FashionMNIST

- same specs as MNIST
- Web: https://github.com/zalandoresearch/fashion-mnist

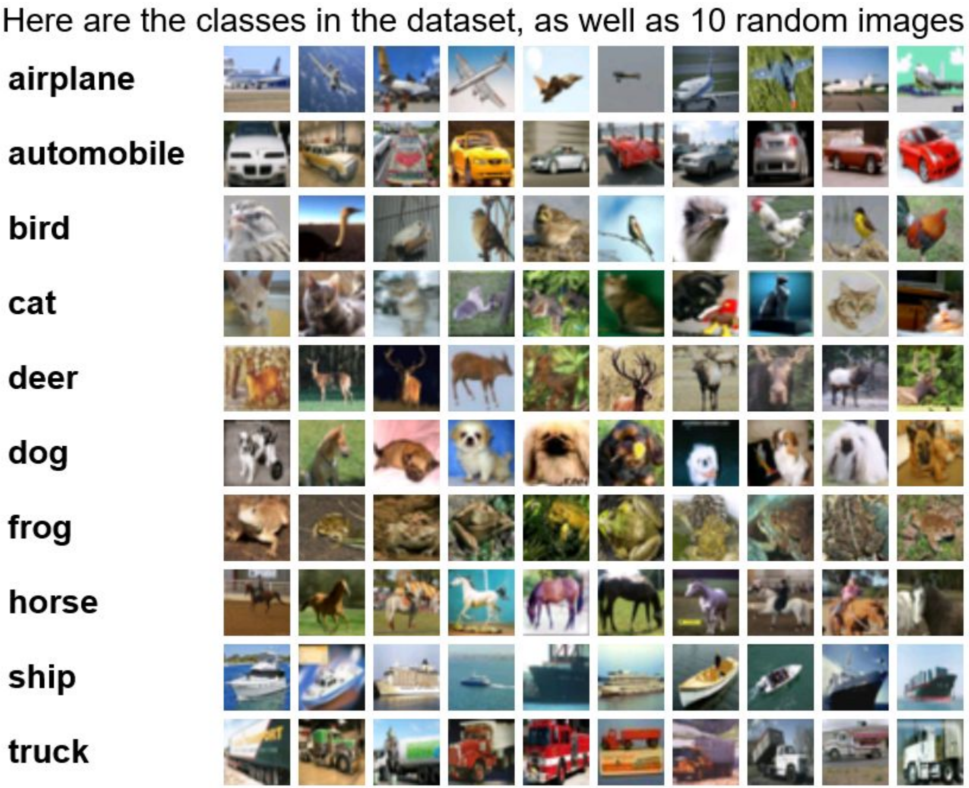
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	



some well-known image data sets

CIFAR10 (CIFAR100)

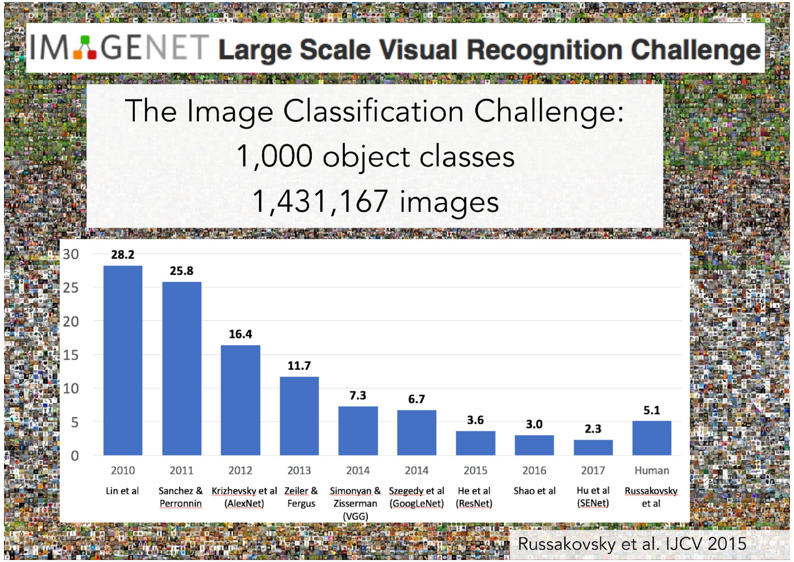
- 10 classes (100 classes)
- 60,000 training images and 10,000 test images, grayscale, of size 32 × 32
- ~160 MB
- https://www.cs.toronto.edu/~kriz/cifar.html



some well-known image data sets

ImageNet

- about 1000 instances of each of about 100,000 labels (mostly nouns)
- basis of the "Large Scale Visual Recognition Challenge":

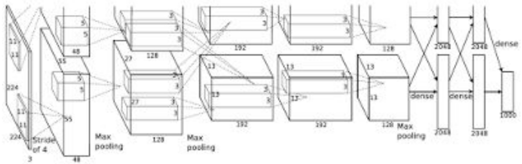


examples: https://nikhilweee.me/shis/html/index-7.html

- Web: https://image-net.org/

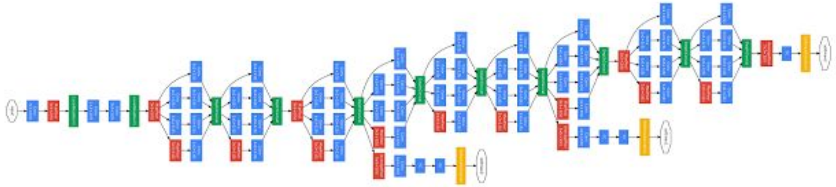
some milestones on the ImageNet Challenge

pre-2012: SVM + carefully hand-crafted features: 26% (top5) error rate.



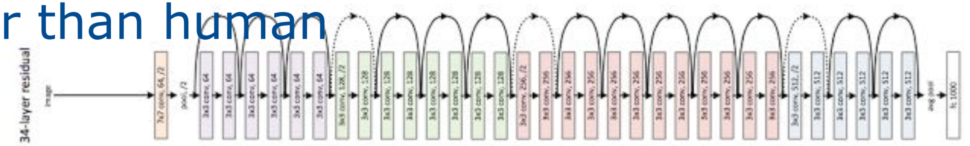
2012: AlexNet (early ConvNet): 16%

2013: a variant of AlexNet: 11.7%



2014: GoogLeNet: 6.6%

2015: deep ResNet: 4.5% ; better than human



2021: nets that use “attention”: under 2%

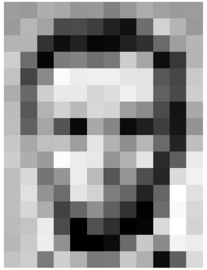
https://paperswithcode.com/sota/image-classification-on-imagenet?metric=Top%205%20Accuracy


WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
~~I'LL NEED A RESEARCH TEAM AND FIVE YEARS.~~
IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Not anymore

RGB images

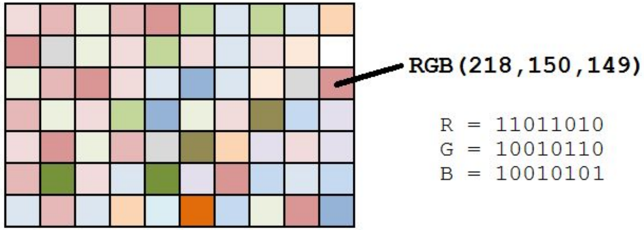
Pixel value: 8-bit integer, values from 0 to 255.



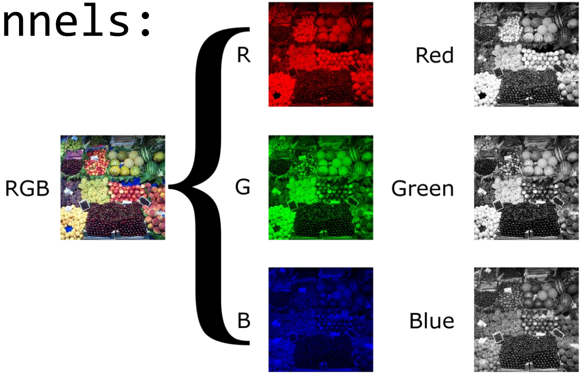
Input Image

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Pixel Representation



RGB three channels:

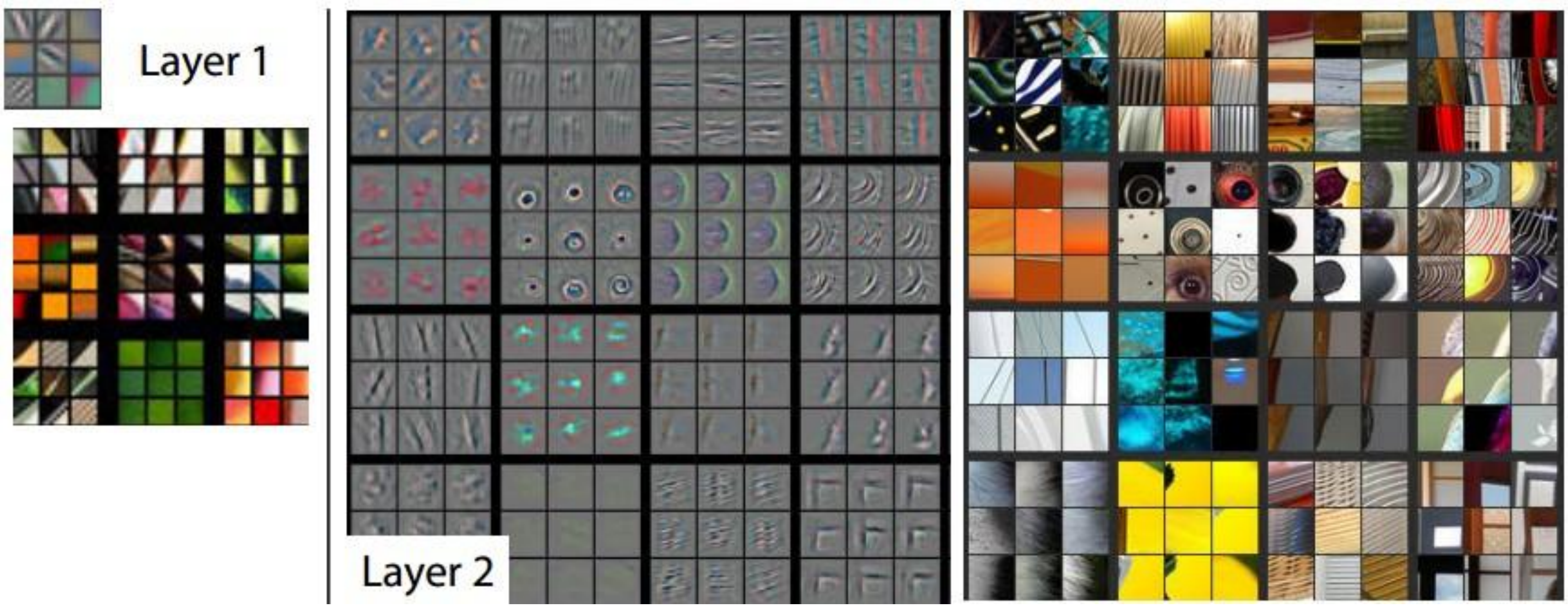


you could say the input pattern has a "[depth](#)" of 3

what does the CNN see?

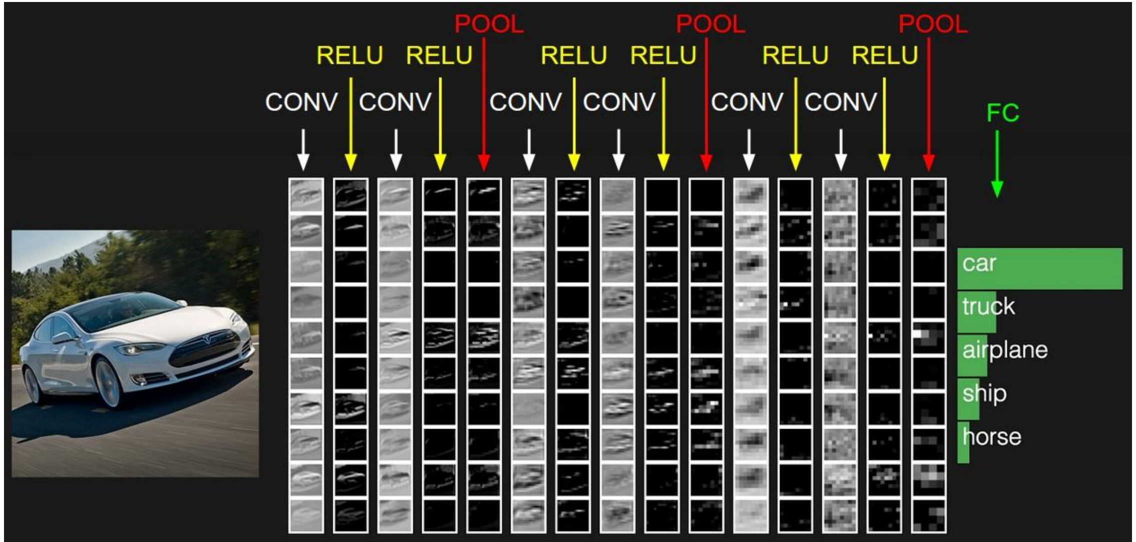
we can interpret filter parameters as “images” themselves

- 1st layer learns to look for lines
- 2nd layer learns to look for textures…
- nth layer learns to look for dogs, cars, people, …



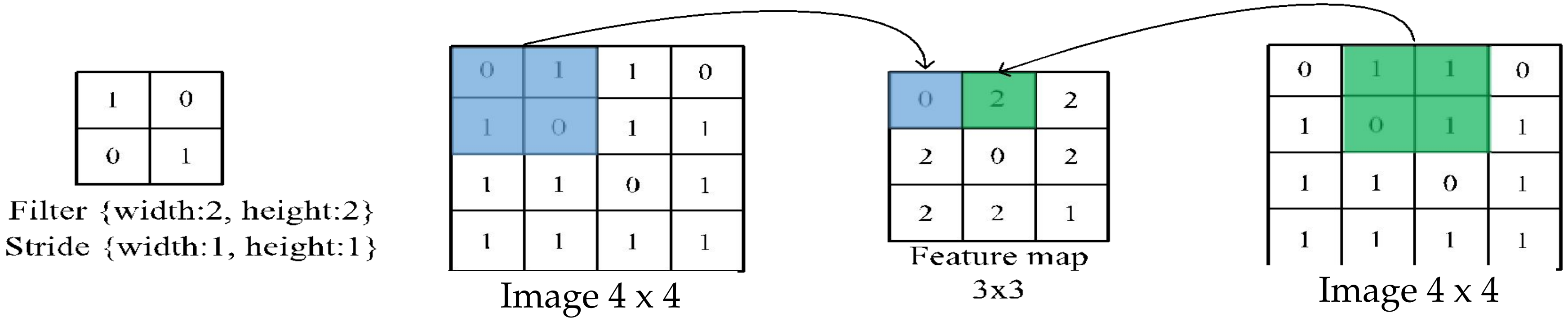
overall architecture

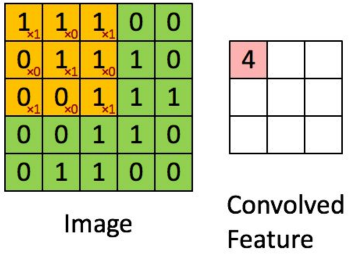
- Overall architecture:
  - Convolutional Layers are followed by
    - Non-linear activation function (e.g. ReLU)
    - Pooling layer (down sampling)
    - Stack many such layers to learn more complex filters
  - At the end: fully connected (`dense’) network for classification



Convolutional layer

- Convolutional Layers
  - Slide low dimensional “filter” across the image
  - Same filter is used to create *entire feature map*





filters

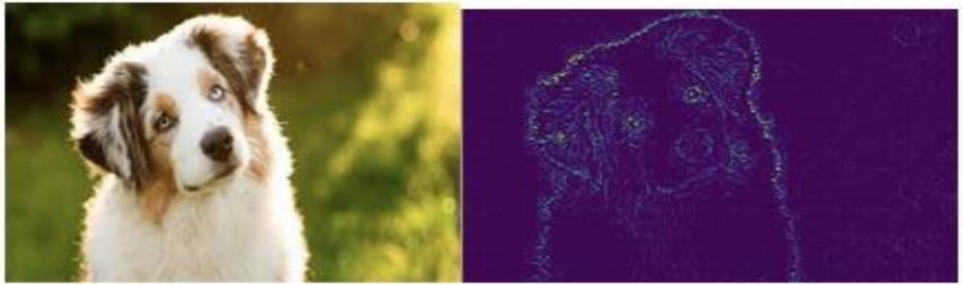
Filters have been used on images for years

0	-1	0
-1	5	-1
0	-1	0

Filters, like sobel filters, can perform *edge detection* and other operations

-1	-2	-1
0	0	0
1	2	1

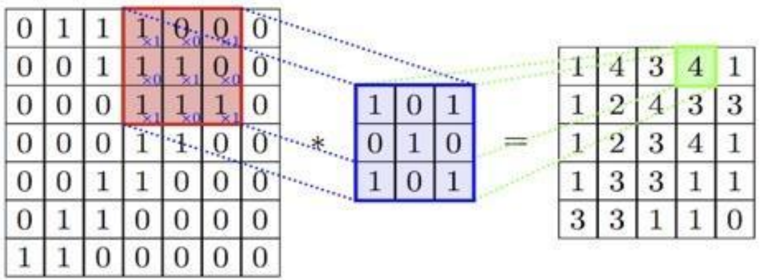
Horizontal

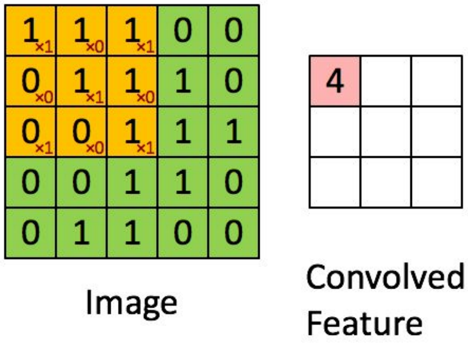


https://www.saama.com/blog/different-kinds-convolutional-filters/

filters are "convolved" with the image

https://www.saama.com/blog/different-kinds-convolutional-filters/





A convolution operation is an element-wise matrix multiplication operation.

Two ways to think of this:

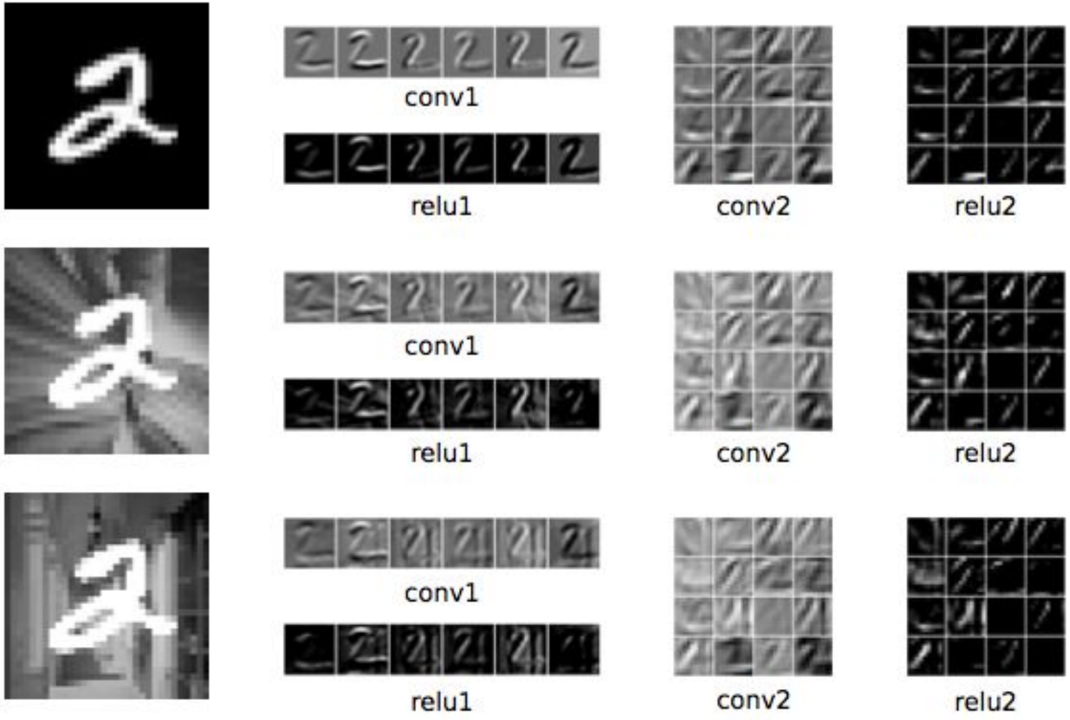
I. the filter slides across the image, filling in the result grid

II. do them all at once: it's just a big weights matrix, but the weights are *shared*

It's building in a spatially invariant process, while still keeping track of position

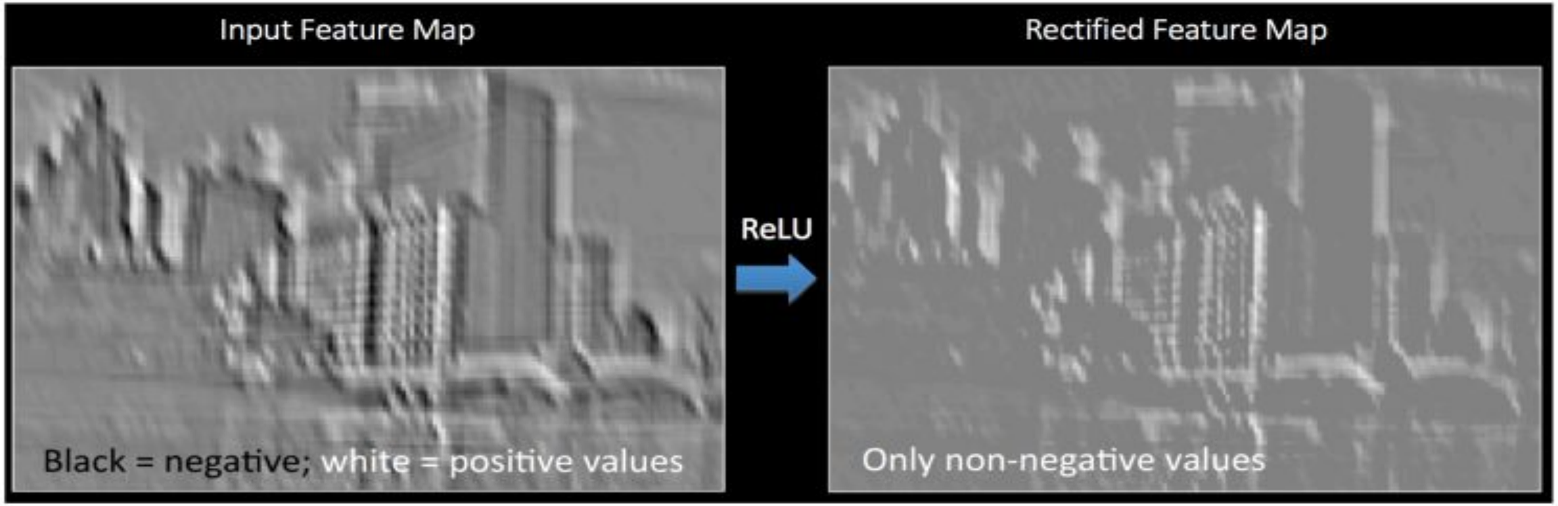
isn't this "just" linear?

Convolutions are linear – so we then do an element-wise non-linearity, such as a ReLU, just like in standard MLPs



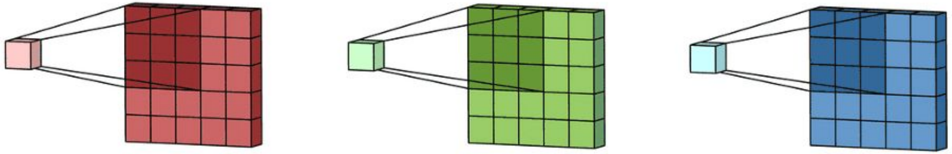
ReLU (or other) layer for non-linearity

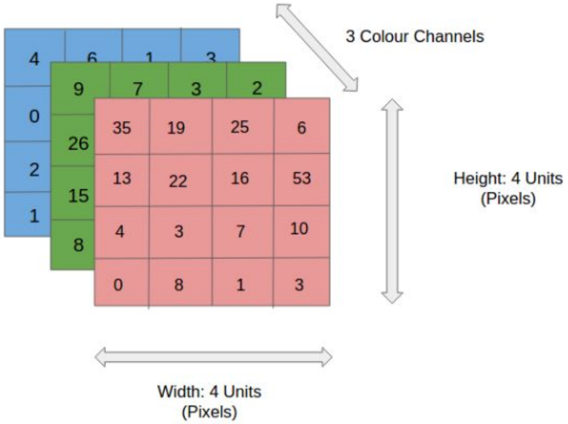
- ReLU stands for Rectified Linear Unit
  - Applicable to any neural net, not just CNNs.
  - Other non linear functions such as **tanh** or **sigmoid** can also be used instead of ReLU, but ReLU has been found to perform better in most situations.

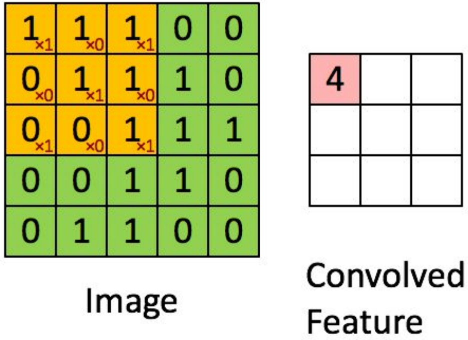


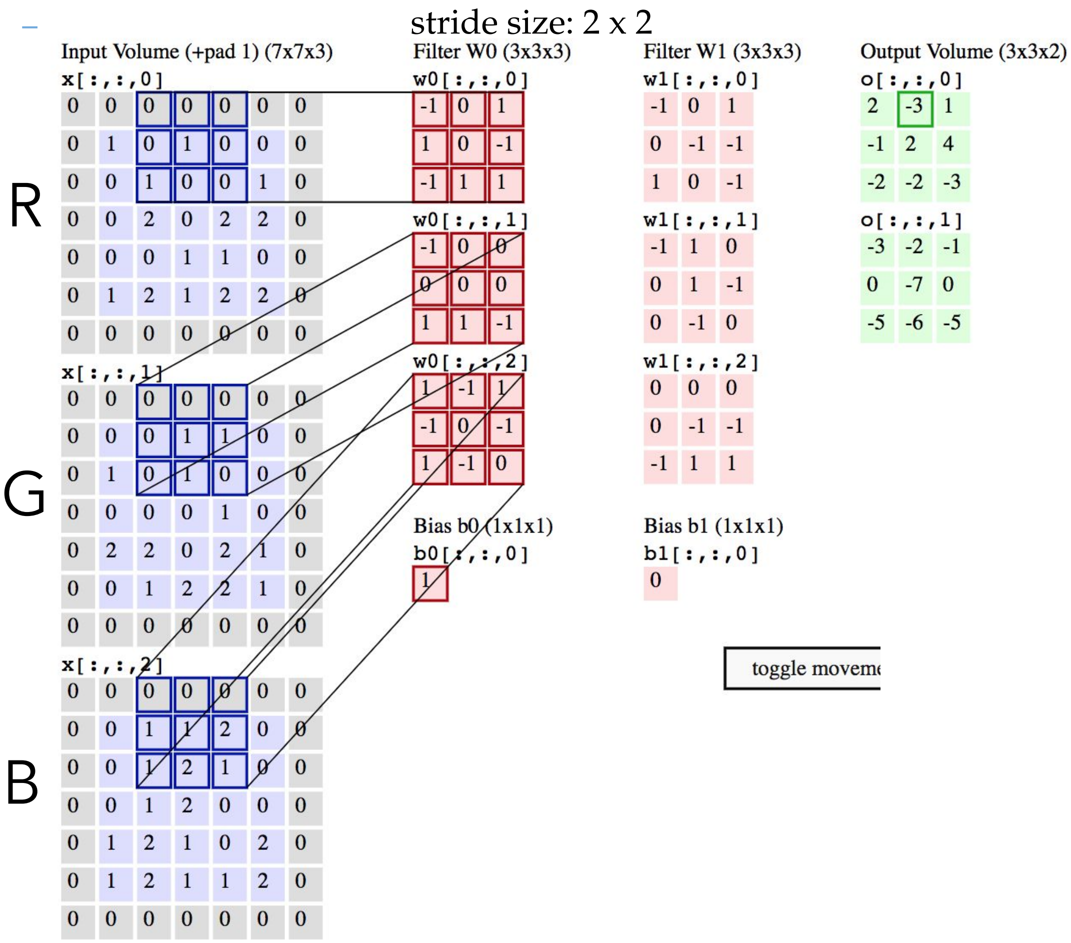
(and don't forget the other generic options- Dropout & Batch-norm)

different filters for different colours?









?

- Elementwise multiplying the highlighted input with the filter (dot product)
- summing it up
- then offsetting the result by the bias

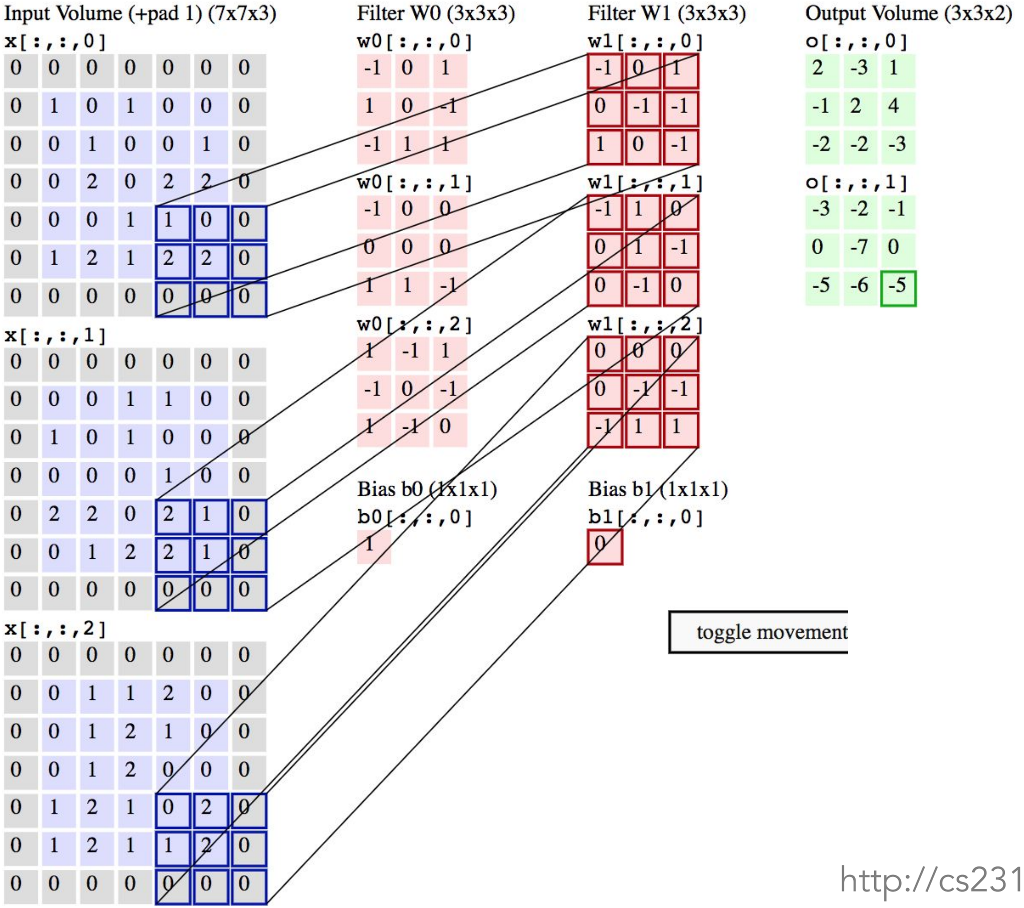
A typical filter on a first layer of a ConvNet

convolution layer

R

G

B

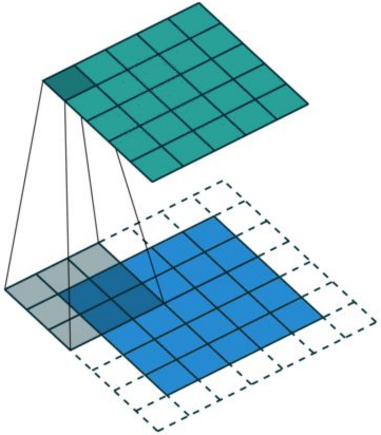


Output?

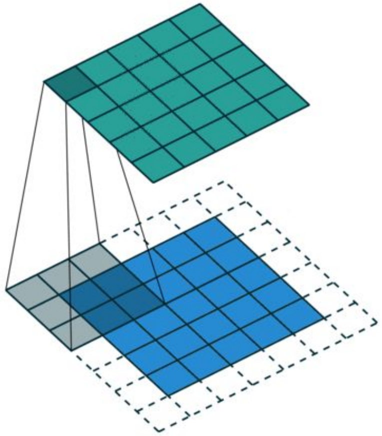
http://cs231n.github.io/convolutional-networks/

padding

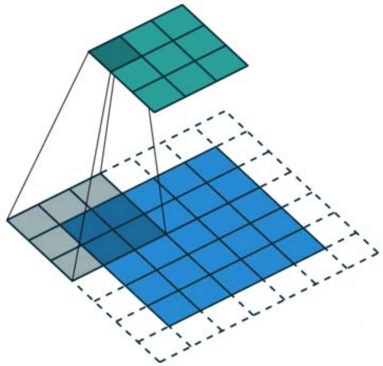
- (Zero-)Padding refers to the process of symmetrically adding zeroes to the input matrix.
  - commonly used modification that allows the size of the input to be adjusted
  - mostly used in designing the CNN layers when the dimensions of the input volume need to be preserved in the output volume.



“stride”



stride=1



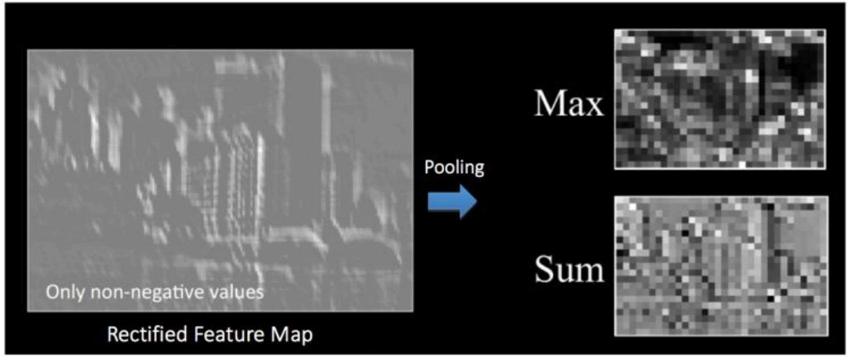
stride=2

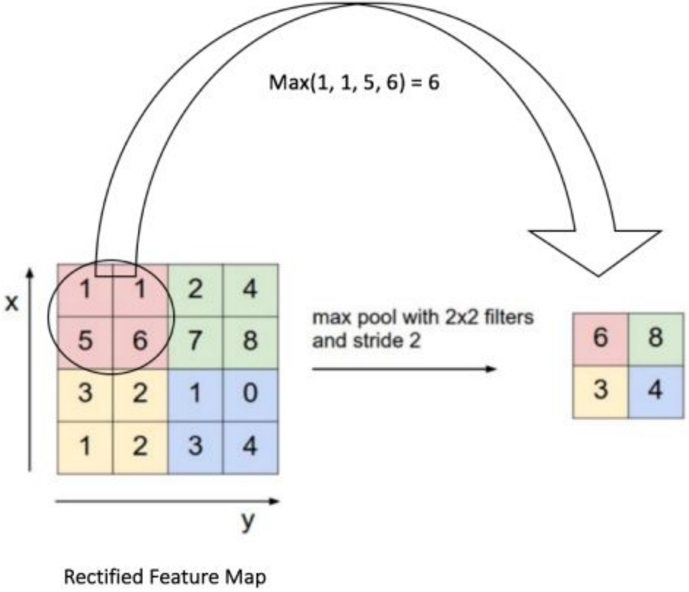
pooling

also called *subsampling* or *downsampling*

- reduces the dimensionality of each feature map but retains the most important information

- types: Max, Average (==Sum)
  Max seems to work better



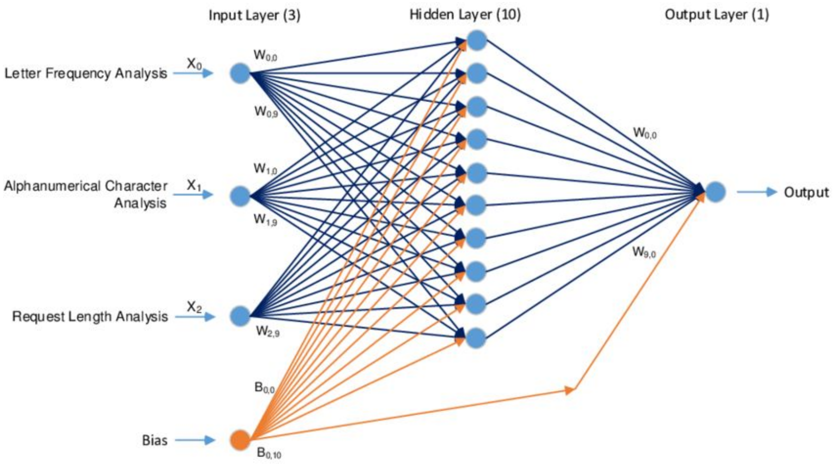


A nice description of all this:
https://cs231n.github.io/convolutional-networks/

fully connected ("dense") layers

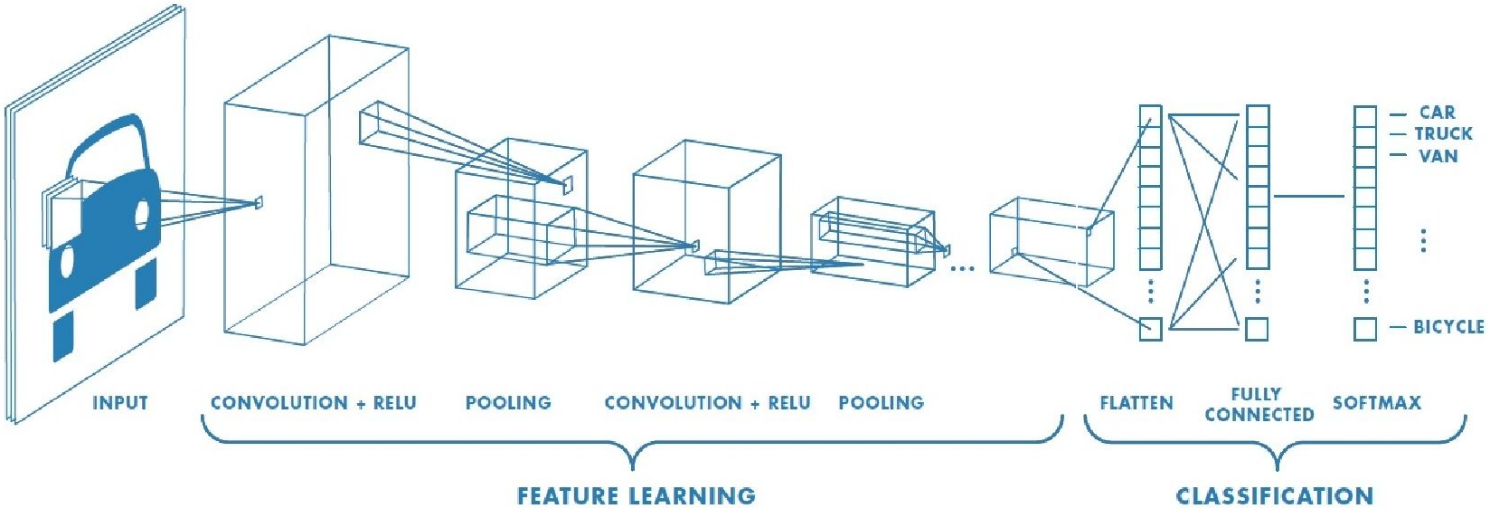
dense layers are just the traditional *Multi Layer Perceptron*

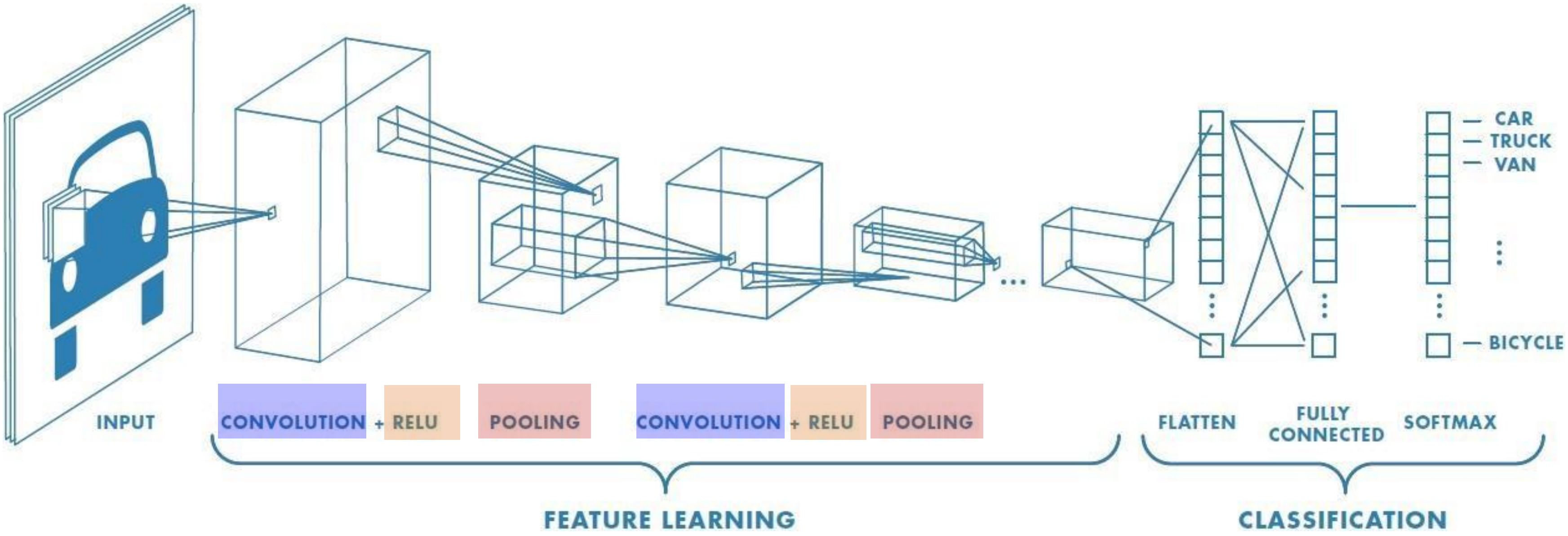
The purpose of the final dense layers (together with softmax) is to use these features for classifying the input image into various classes based on the training dataset



Architecture of a deep CNN

- Convolutional Layers
- Non-linear activation function (e.g. ReLU)
- Pooling layer (down sampling):
- Stack many such layers to learn more complex filters
- fully connected (dense) network for classification



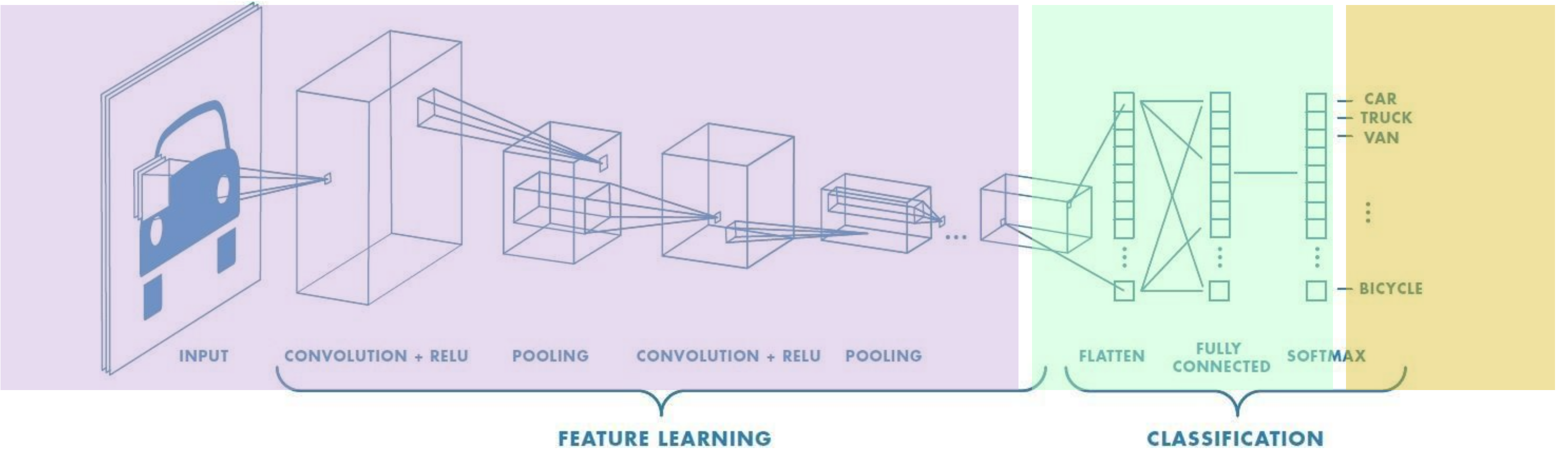


1. informative features, the same anywhere, through **convolution**
2. non-linearity, introduced through the **activation** functions
3. reduce dimensionality while preserving location, through **pooling**

We might go through that cycle of 1,2,3 several times :

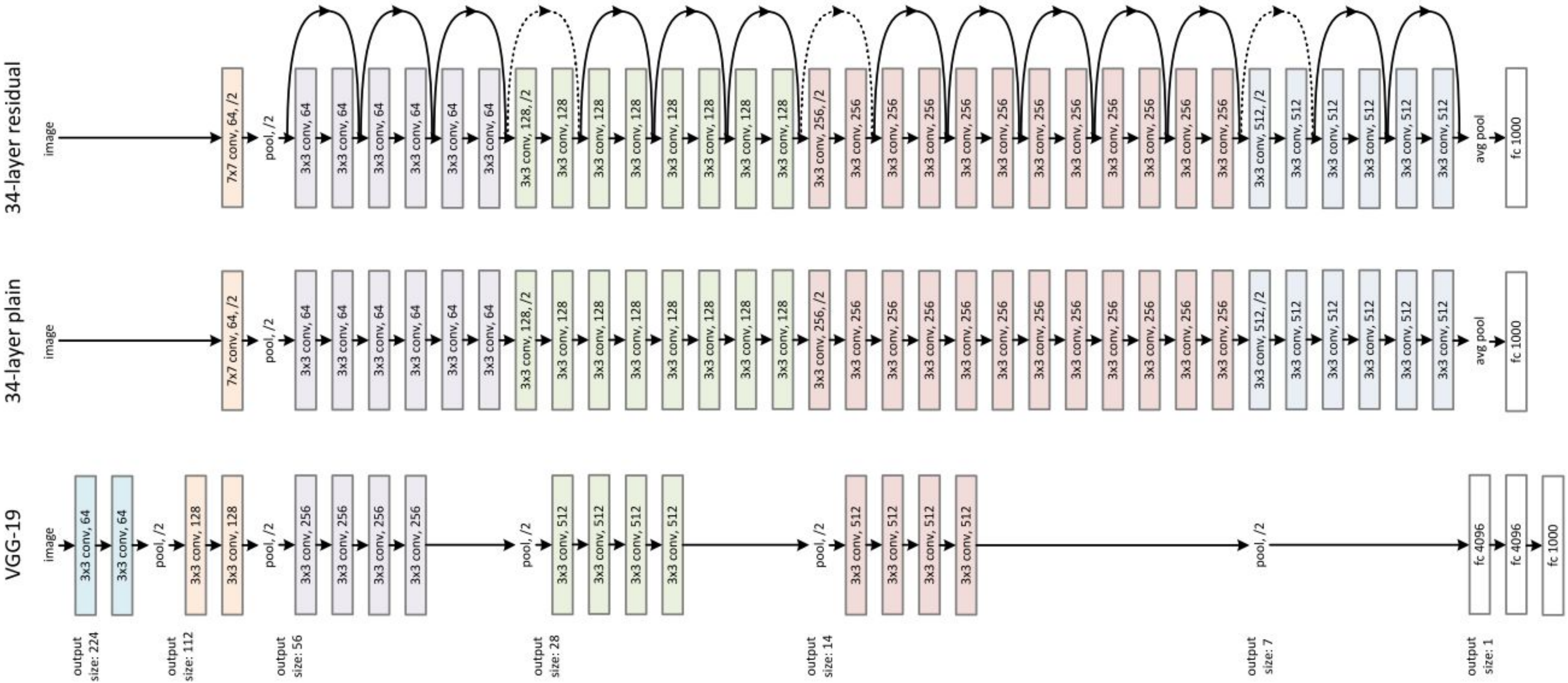
non-linear features in a **hierarchy**, through the network **depth**

suggestion



First	Convolution and pooling layers (1,2,3 on previous slide) output high-level features of input, still in their approximate position
Then	Dense layer(s) use these features for classifying the image overall
Finally	Softmax takes logits → probability of each class, given the image

examples


34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
VGG-19
image
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 256
3x3 conv, 512
fc 4096
fc 1000
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1

data augmentation

Training of large network requires a lot of samples.

Consider augmenting the data set without adding brand new data.
For example, by....

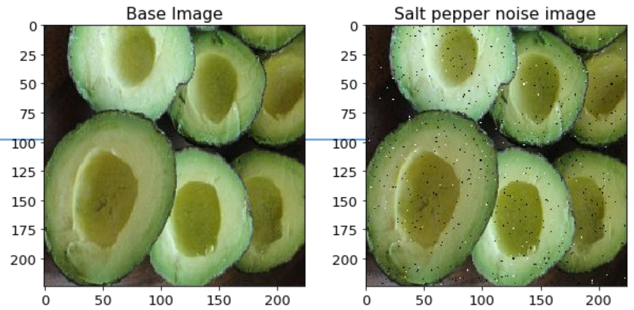
- performing geometric transformations:
  - e.g. translation, rotation, flipping
- changes to color, brightness, contrast
- or even adding some noise:
  - e.g. adding Salt and Pepper noise

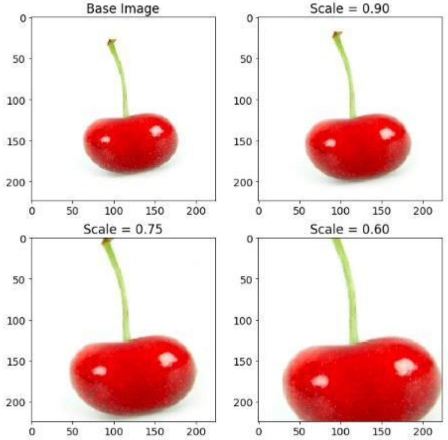
in effect, data augmentation can increase the number of samples in the dataset using the data you already have:

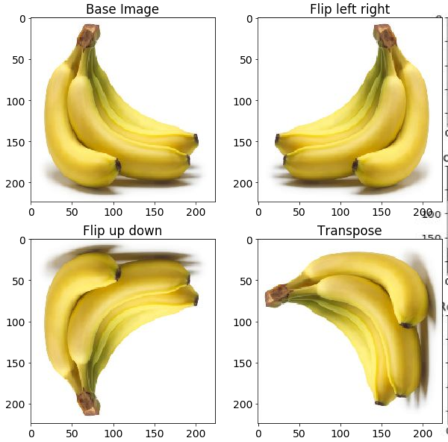
Get creative for your problem!

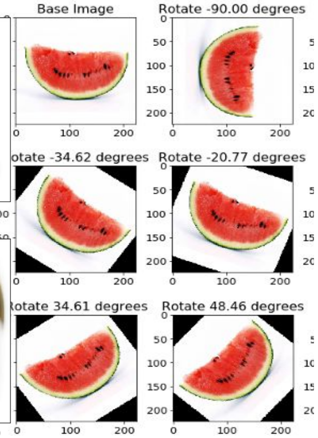
data augmentation

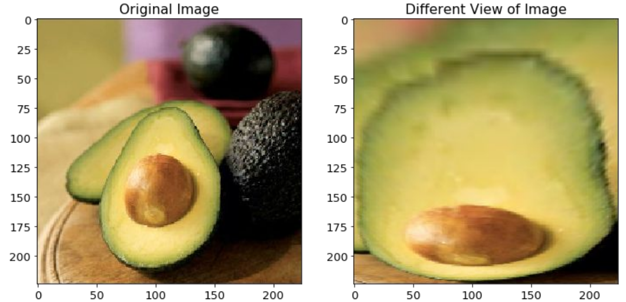
Training of large network requires a lot of samples.
Consider augmenting the data set without adding brand new data. For example…

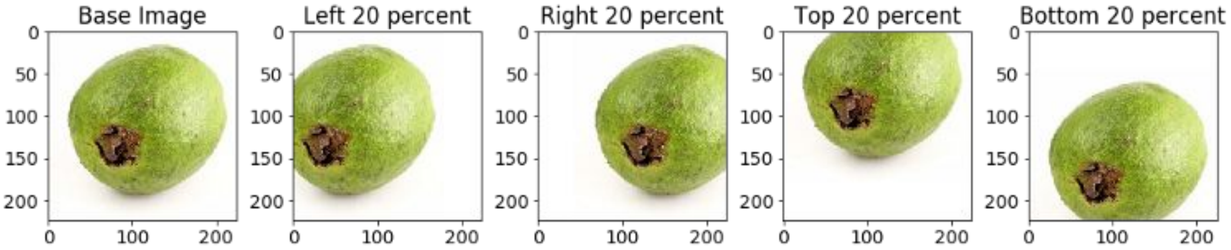












data augmentation

Training of large network requires a lot of samples.

Consider augmenting the data set without adding brand new data.

For example, by....

- performing geometric transformations:
  - translation, rotation (rotation at finer angles), flipping
- changes to color, brightness, contrast:
  - scaling
- or even adding some noise:
  - adding Salt and Pepper noise

in effect, data augmentation can increase the number of samples in the dataset using the data you already have:

Get creative for your problem!

transfer learning

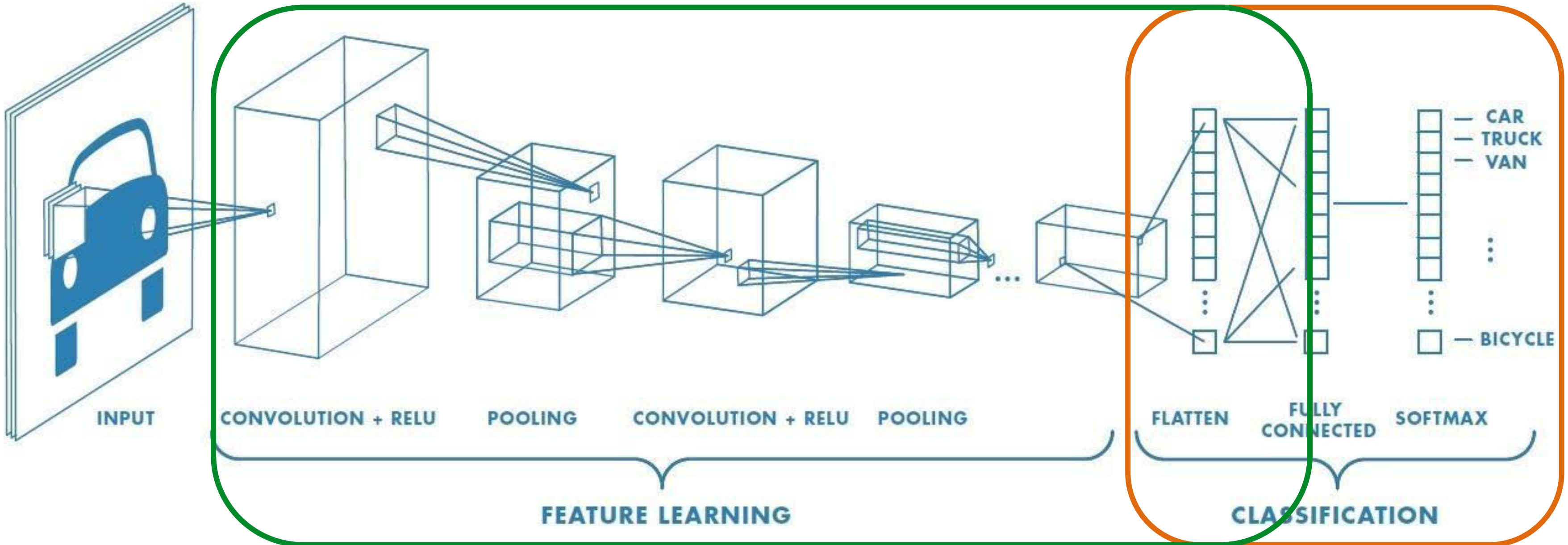
issue: training a CNN can take a lot of (i) time, (ii) data

idea: take a pre-trained CNN and re-purpose it…

Two possible modes: “fine-tuning” vs “feature extraction”

train further, with the new data

replace and train



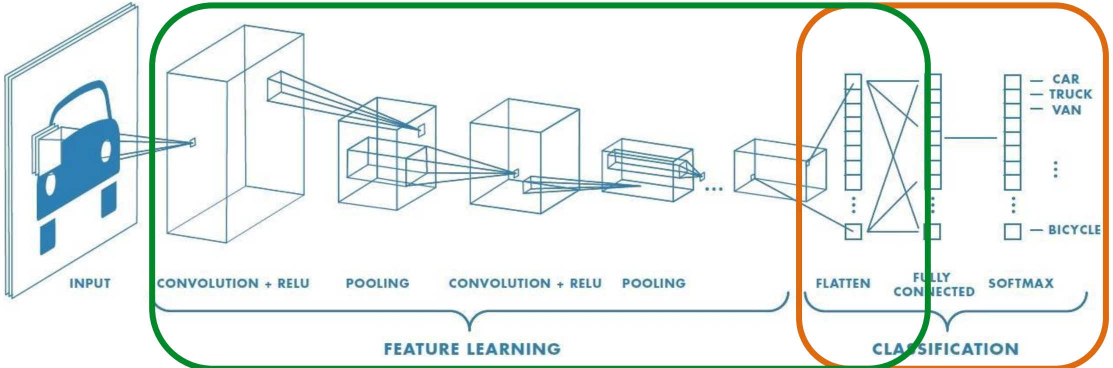
transfer learning

fine-tuning:

- **What**: *Adjust* the weights of a pre-trained network. Typically, you might "freeze" the earlier layers (because early layers capture universal features like edges and textures) and only fine-tune some of the deeper layers.
- **When**: Use fine-tuning when the new dataset is large and not entirely similar to the original dataset.

feature extraction

- **What**: Use the pre-trained network as a *fixed* feature extractor. The output of the network (excluding the final classification layer) is removed. A new classifier (e.g., a single dense layer) is trained on top of that for the new task.
- **When**: Use when the new dataset is small or when it's very similar to the original dataset.



See: torchvision.models

Some examples:

VGG: Developed by the Visual Geometry Group at the University of Oxford, VGG (like VGG16 and VGG19) has deep architectures with up to 19 layers. Simple, uniform architecture.

ResNet: Developed by Microsoft Research, uses "skip connections" or "residual connections" to combat the vanishing gradient problem in very deep networks. Variants include ResNet-50, ResNet-101, and ResNet-152.

Inception (or GoogLeNet): Introduced by Google, the Inception network has a complex structure with "Inception modules". It's known for its efficiency in terms of computation.

MobileNet: Designed for mobile and embedded vision applications, MobileNet is efficient in terms of computation and size. It introduces depthwise separable convolutions which reduce the number of parameters and computations. See teachablemachine.withgoogle.com, which uses a MobileNet.

DenseNet: As opposed to ResNets, DenseNet introduces direct connections from any layer to all subsequent layers. It's very efficient in terms of parameters.

EfficientNet: A newer family of models that scales the width, depth, and resolution of the network based on a set of fixed scaling coefficients.

https://teachablemachine.withgoogle.com/

https://pytorch.org/docs/stable/hub.html

https://assemblyai.com