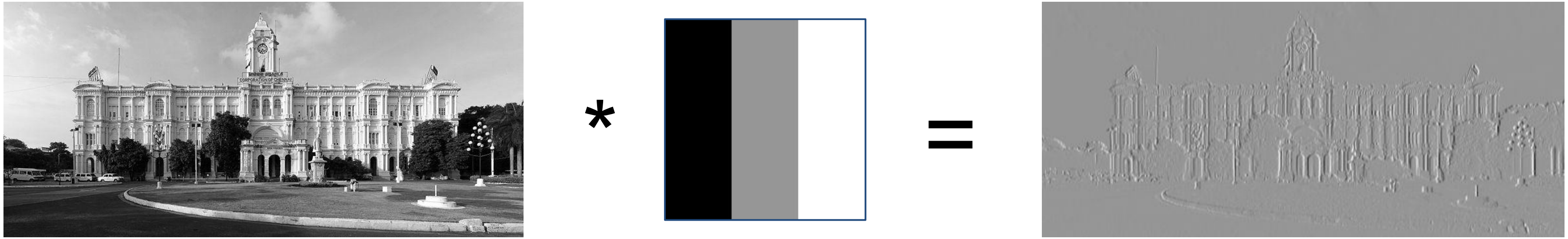
Physalia Courses

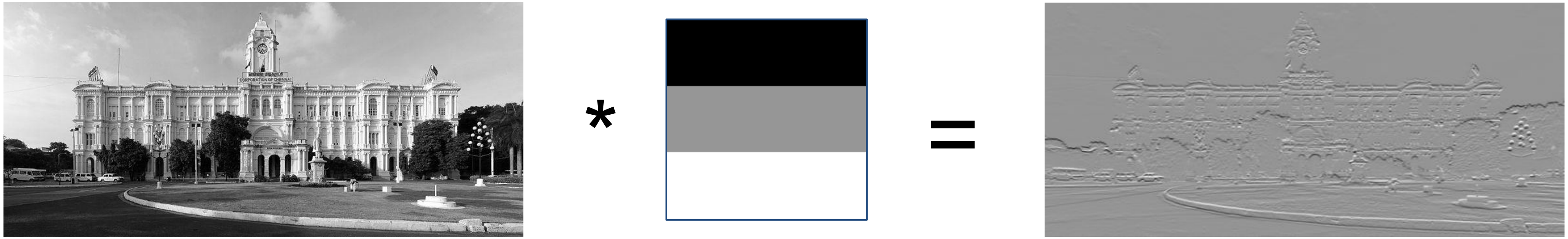
The building blocks of deep learning models/2

A not so light overview

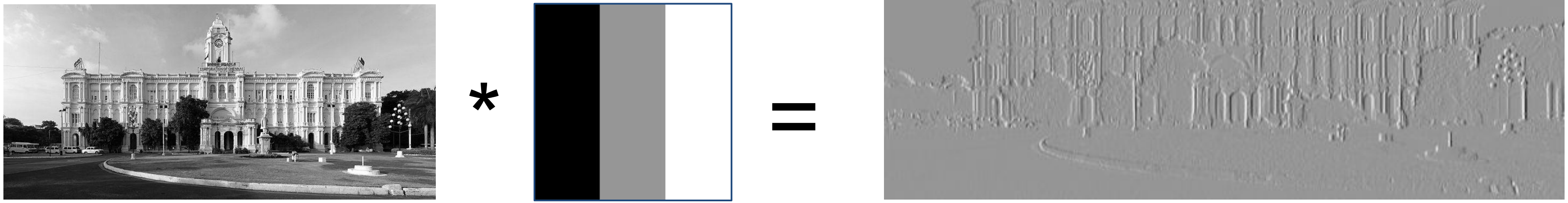
Filippo Biscarini
Senior Scientist
CNR, Milan (Italy)

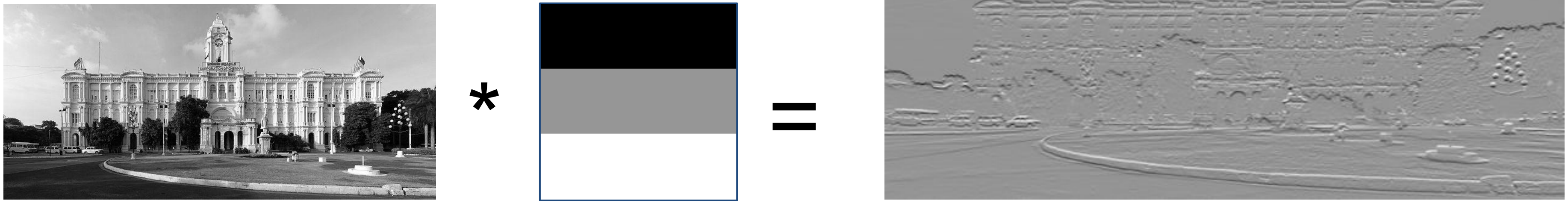
Nelson Nazzicari
Senior Scientist
CREA, Lodi (Italy)



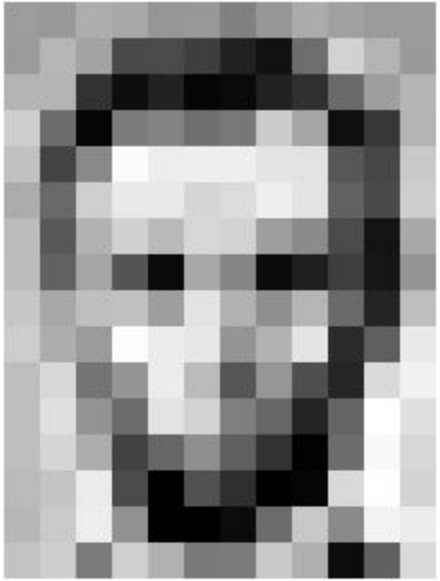


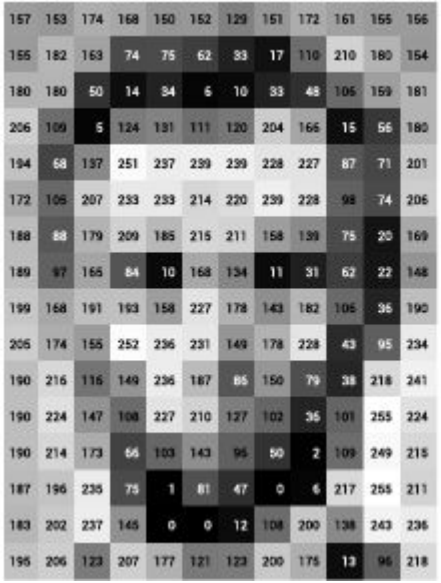
Convolution: an example



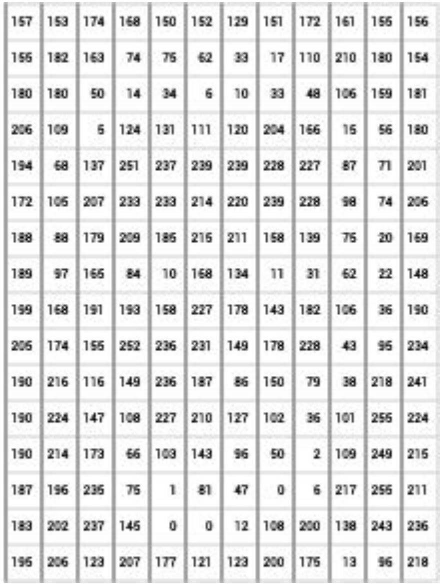


Sidetrack: how a computer sees an image



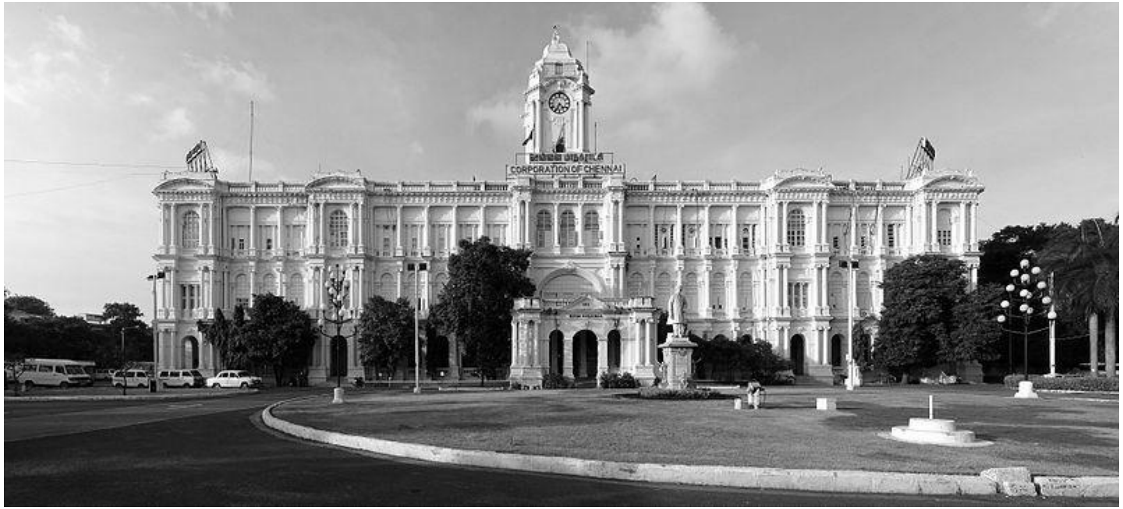


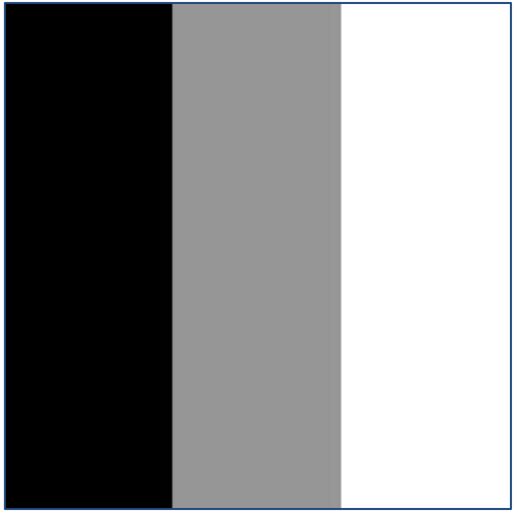
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	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



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

I lied





Back to convolution: numbers

1	9	8	0	4	3
2	6	10	5	7	6
5	0	4	6	4	5
1	2	5	9	5	0
0	4	4	9	5	6
1	2	4	12	4	3

*

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

=

Back to convolution: numbers

1	9	8	0	4	3
2	6	10	5	7	6
5	0	4	6	4	5
1	2	5	9	5	0
0	4	4	9	5	6
1	2	4	12	4	3

*

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

=

14	-4	-7	3
...	...	...	...
...	...	...	...
...	...	...	...

1 x -1 + 2 x -1 + 5 x -1 + 9 x 0 + 6 x 0 + 0 x 0 + 8 x 1 + 10 x 1 + 4 x 1 = 14

9 x -1 + 6 x -1 + 0 x -1 + 8 x 0 + 10 x 0 + 4 x 0 + 0 x 1 + 5 x 1 + 6 x 1 = -4

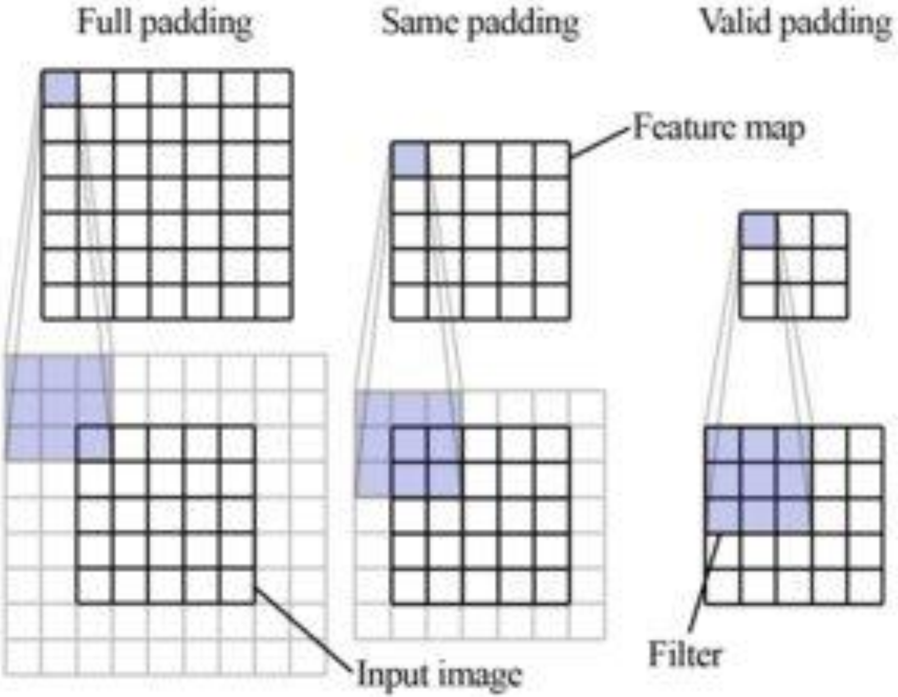
...

Convolution: operations (1/3)

Given a filter and a slice (subset) of the original image of the same size:

1. Do a cell-by-cell multiplication
2. Sum over all cells
3.
4. Move to next slice
5.

Convolution: padding

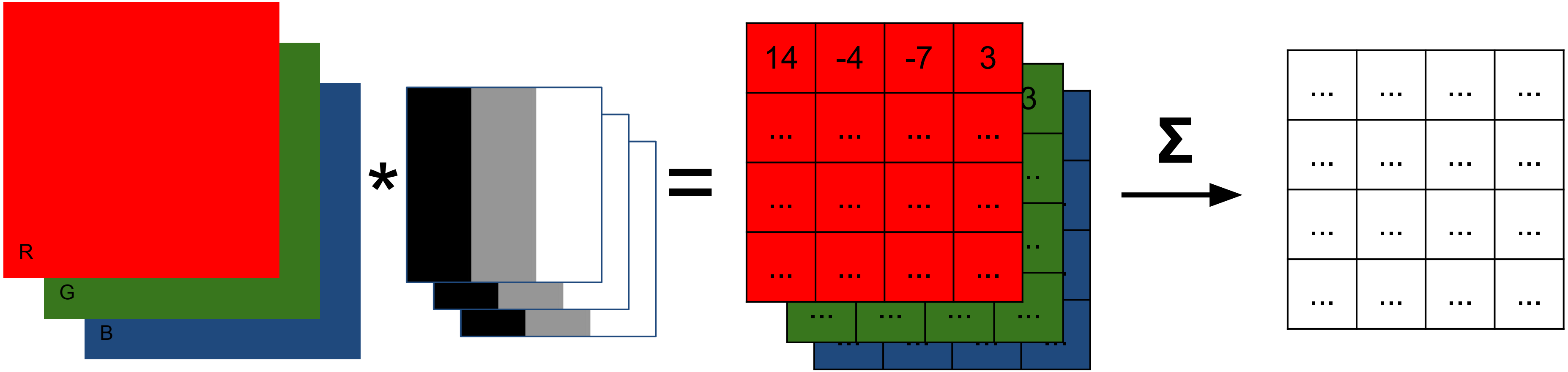


Convolution: operations (2/3)

Given a filter and a slice (subset) of the original image of the same size:

1. Do a cell-by-cell multiplication
2. Sum over all cells
3.
4. Move to next slice
5. Do zero-padding (if requested)

Convolution: multichannels



14	-4	-7	3
…	…	…	…
…	…	…	…
…	…	…	…

…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

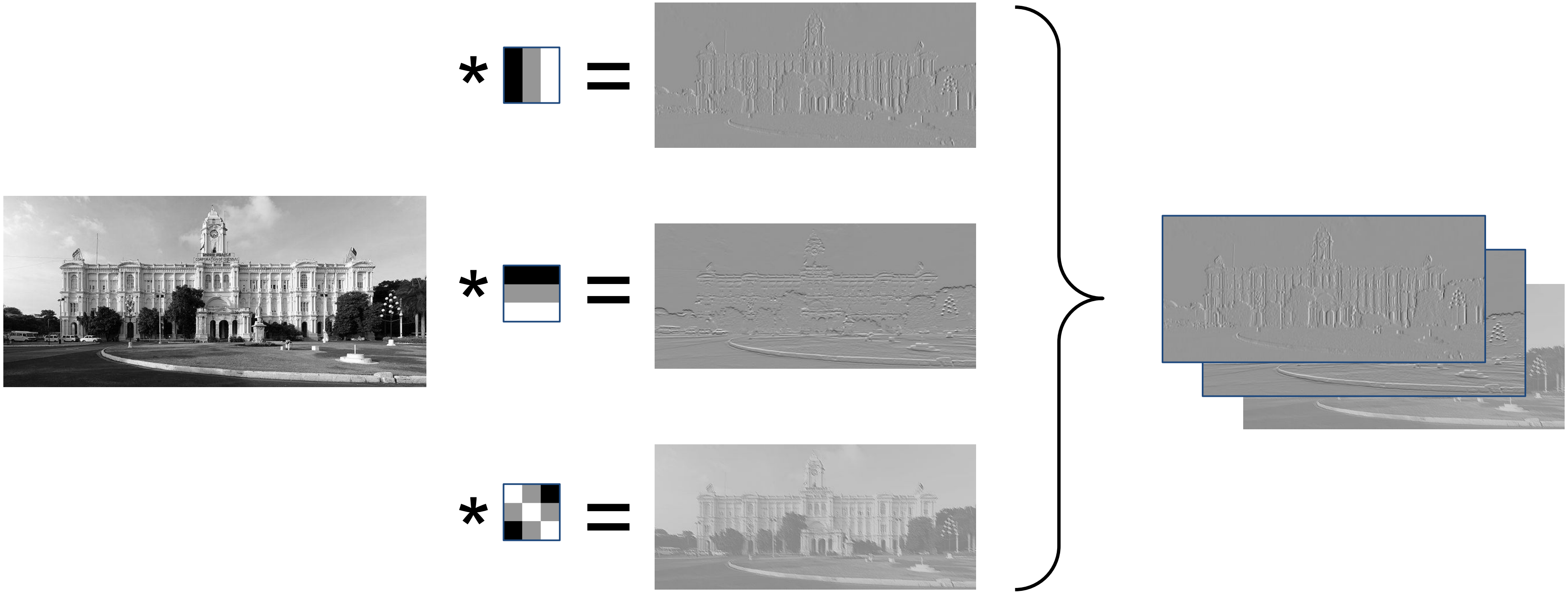
Convolution: operations (3/3)

Given a filter and a slice (subset) of the original image of the same size:

1. Do a cell-by-cell multiplication
2. Sum over all cells
3. Sum over channels
4. Move to next slice
5. Do zero-padding (if requested)

Convolution: parallel filters (units)

Convolution: parallel filters (units)



Convolution: hyperparameters

- Number of units
- Size of the filter (f, usually odd)
- Stride (s, usually 1)
- Padding (p=0 → “valid”, p=(f-1)/2 → “same”)
- Activation function (usually RELU)

Note: the numbers inside the filters are parameters, not hyperparameters. The whole point is to learn them.

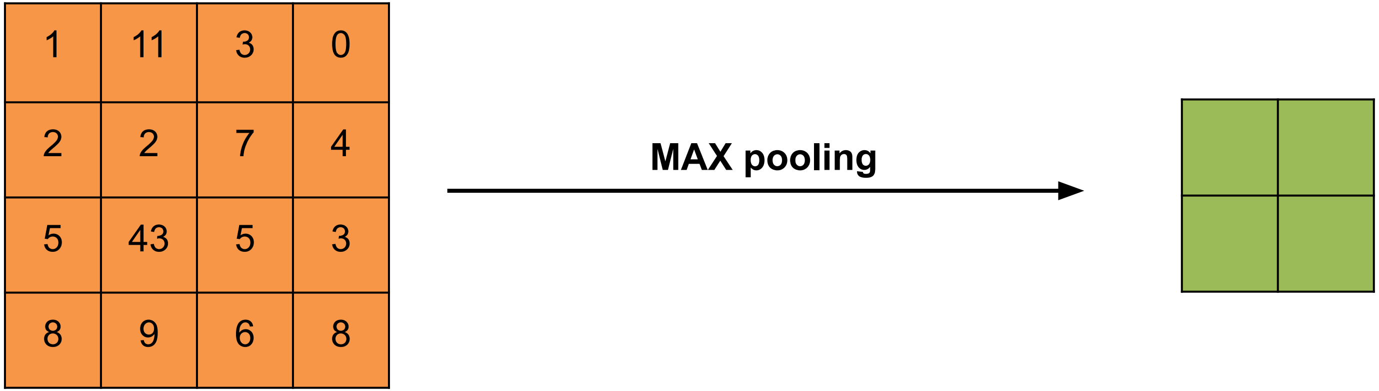
Convolution: in keras

```
from keras.layers import Conv2D

<declare the model, somehow>



model.add(Conv2D(filters=32, kernel_size=(3, 3), strides=1, padding="same", activation="relu",
      input_shape=(200, 200, 3)))
```

Let’s do it!

[notebook]

Pooling: a working example



1	11	3	0
2	2	7	4
5	43	5	3
8	9	6	8

MAX pooling

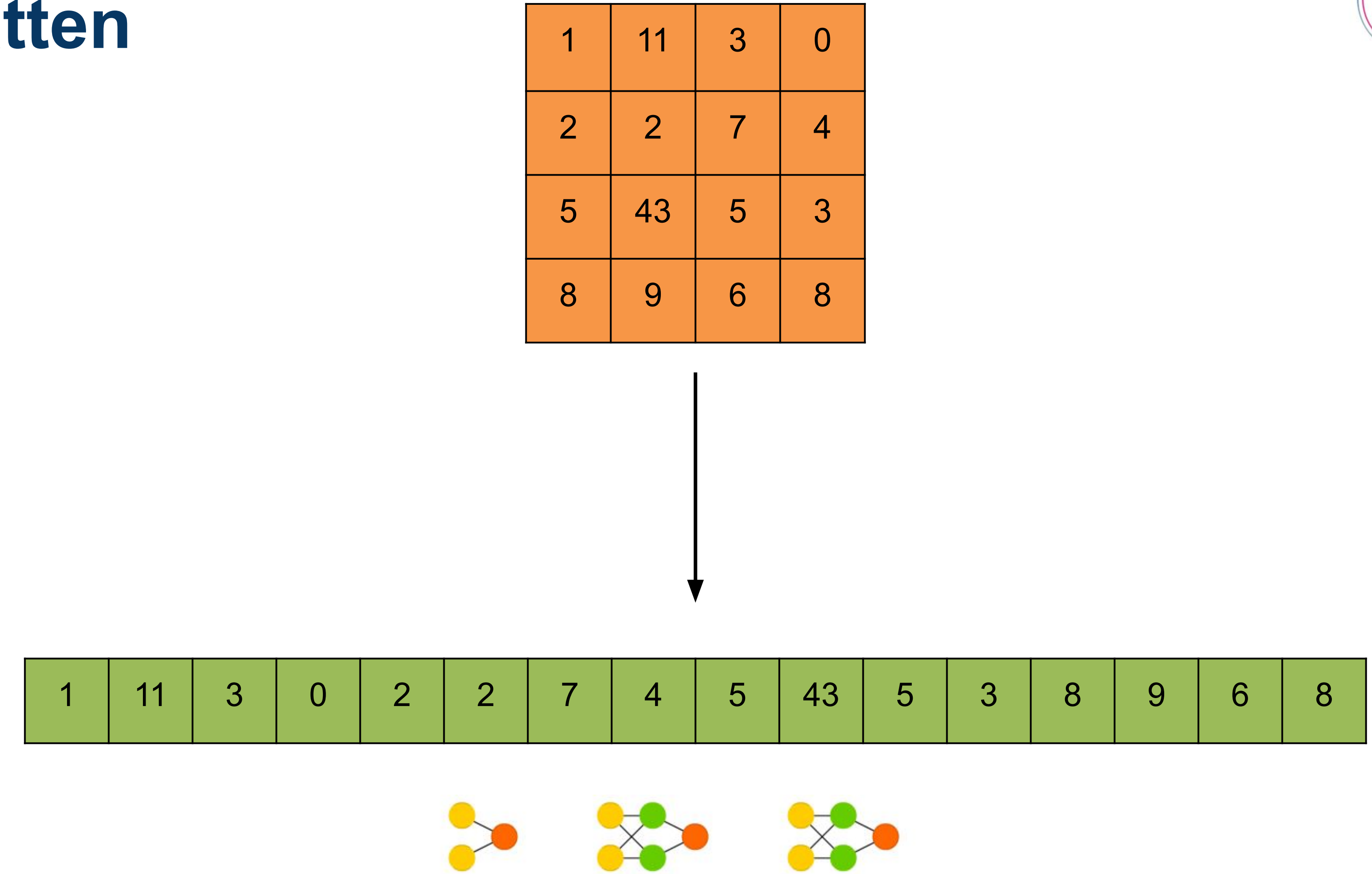
Pooling: hyperparameters

- Size of the filter (f, usually even)
- Stride (s, usually s=f)
- Padding (usually p=0 “valid”)
- Function (usually MAX, can be AVG)
- No activation function
- No parameters to be learned

Pooling: in keras

```
from keras.layers import MaxPooling2D

<declare the model, somehow>



model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
```

Flatten



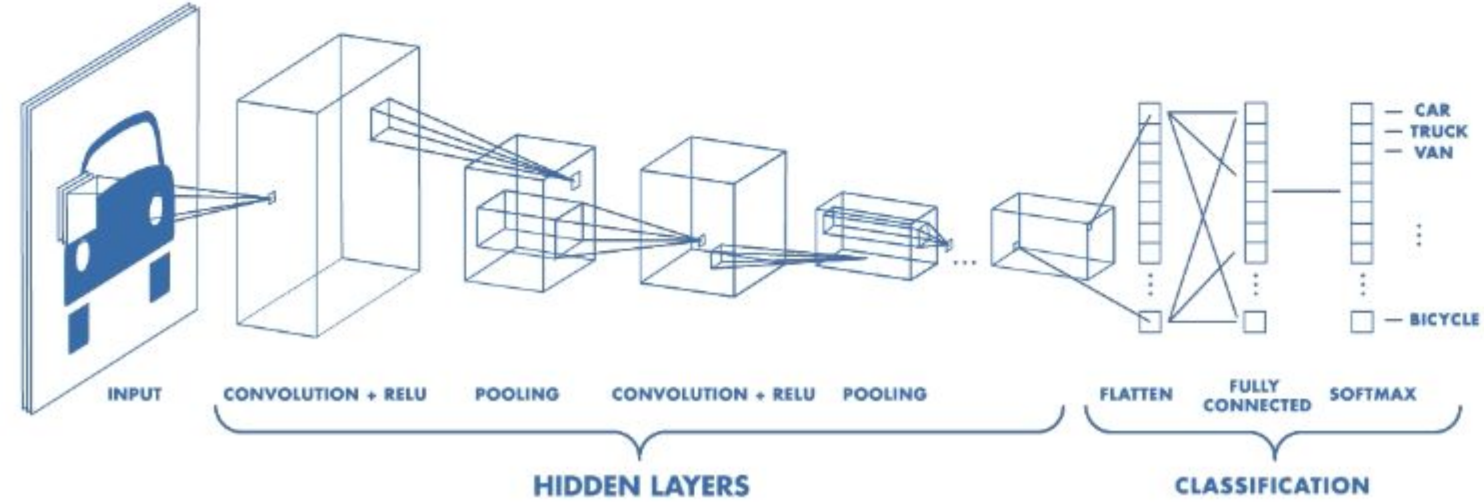
1	11	3	0
2	2	7	4
5	43	5	3
8	9	6	8

1	11	3	0	2	2	7	4	5	43	5	3	8	9	6	8

Flatten: in keras

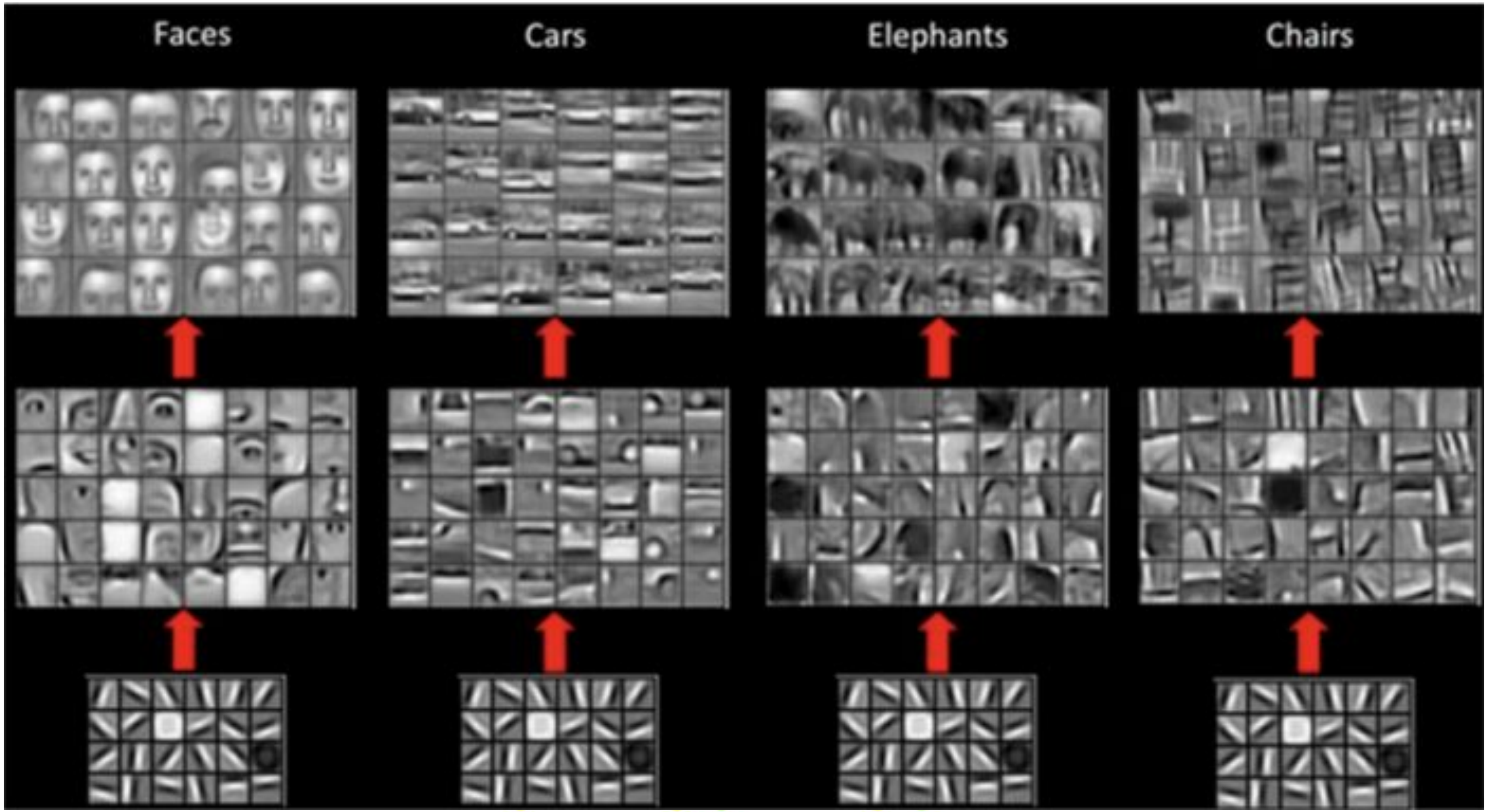
```
from keras.layers import Flatten

<declare the model, somehow>



model.add(Flatten())
```

A typical (?) CNN



Source: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

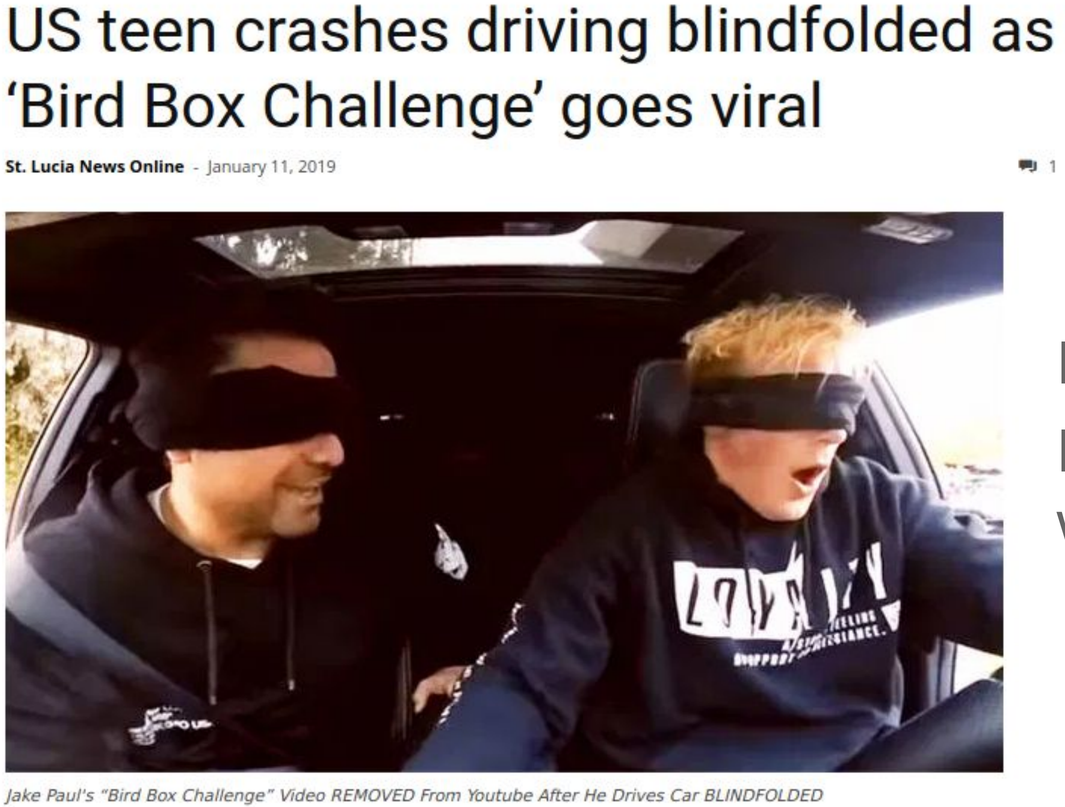
Feature refinement



In keras

```
from keras import models, layers
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = models.Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), strides=1, padding="same", activation="relu",
      input_shape=(200, 200, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(units=5, activation='softmax'))
```

Regularization: overfitting



US teen crashes driving blindfolded as 'Bird Box Challenge' goes viral

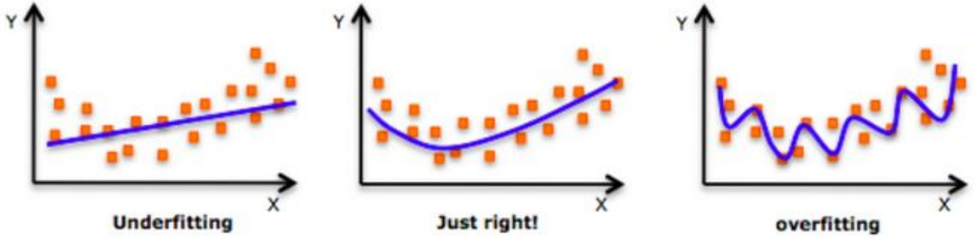
St. Lucia News Online - January 11, 2019 1

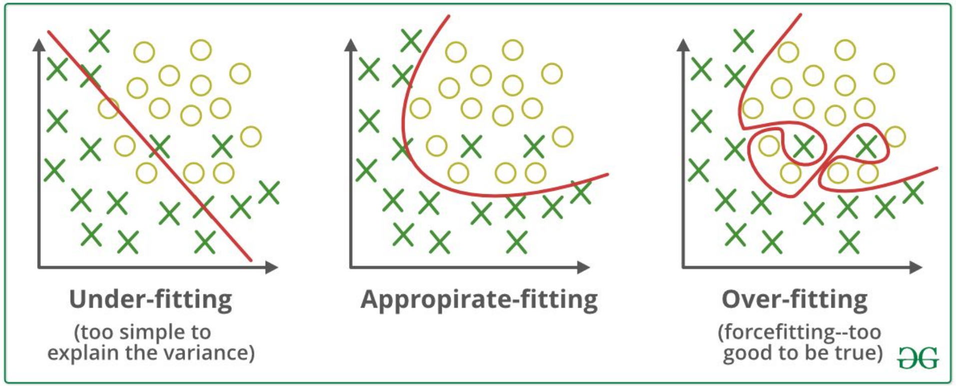
Jake Paul's "Bird Box Challenge" Video REMOVED From Youtube After He Drives Car BLINDFOLDED

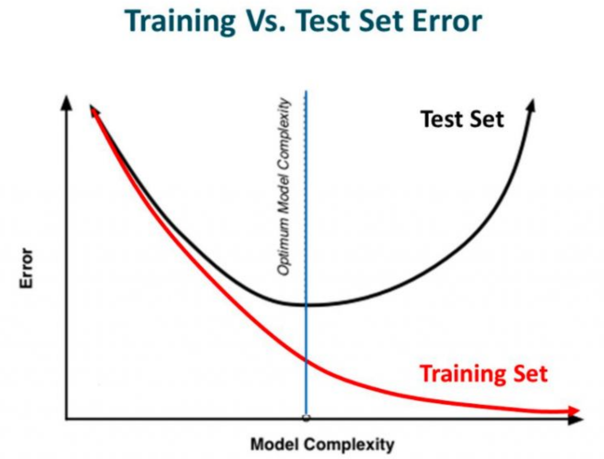
It may work on a known path

It DEFINITELY does not work on a new path

Regularization: overfitting




Under-fitting
(too simple to explain the variance)
Appropirate-fitting
Over-fitting
(forcefitting--too good to be true)
GG



https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/
https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

Regularization: techniques

- L1/L2 regularization
  - Purely algebrical, changes the loss function
  - Penalizes “big” weights
  - With sigmoid forces the net to work with the linear part
- Dropout
  - At each learning iteration some (random) nodes are turned off
  - At test time all nodes are turned on
- Early stopping
  - Stop when error on test set stops shrinking
  - Easy to understand, hidden problems

Regularization: in keras

- L1/L2 regularization

```
from keras import regularizers
model.add(Dense(64, kernel_regularizer=regularizers.l2(0.01)))
```

- Dropout

```
from keras.layers.core import Dropout
model.add(Dense(...))
model.add(Dropout(0.25))
model.add(Dense(...))
```

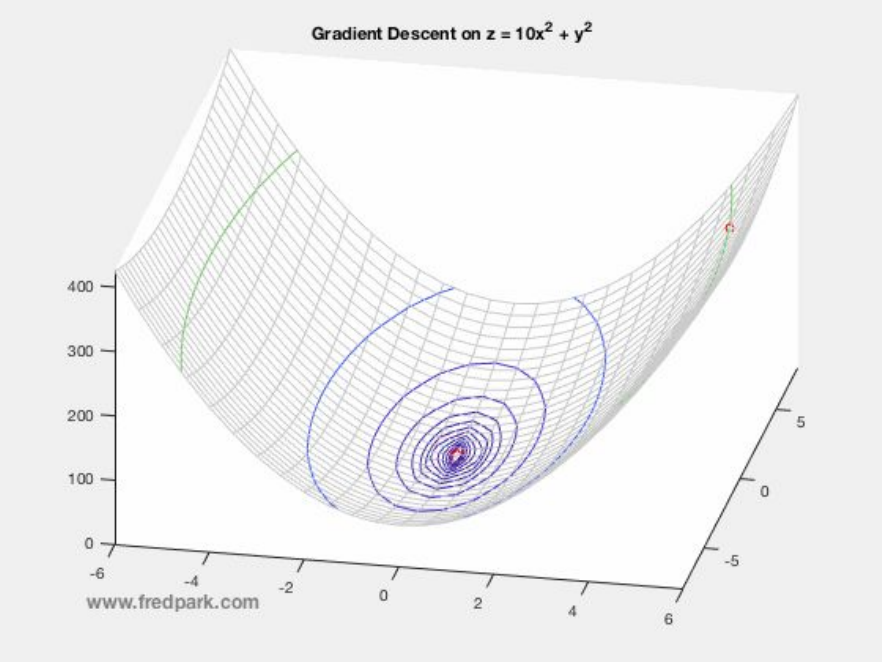
- Early stopping

```
from keras.callbacks import EarlyStopping
model.compile(..., callbacks = [EarlyStopping(monitor='val_acc', patience=3)])
```

Optimizers: a hard truth

Nobody uses Gradient Descent.

Can you guess why?



Sidetrack: exponential average

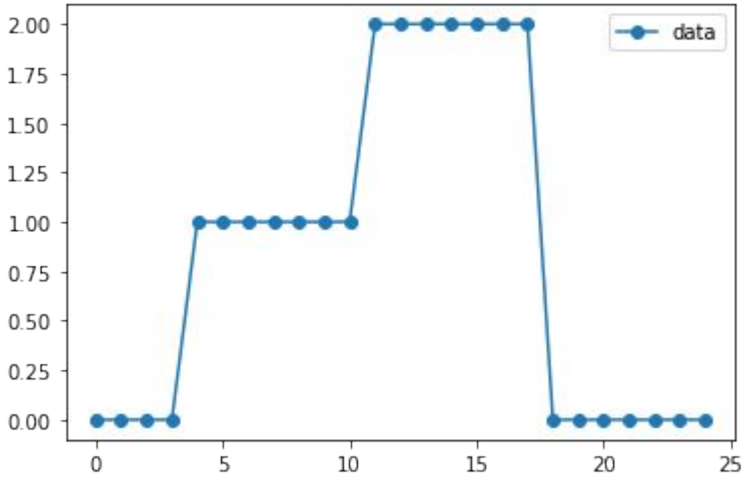
Also called “Exponentially weighted moving average” (EWMA)

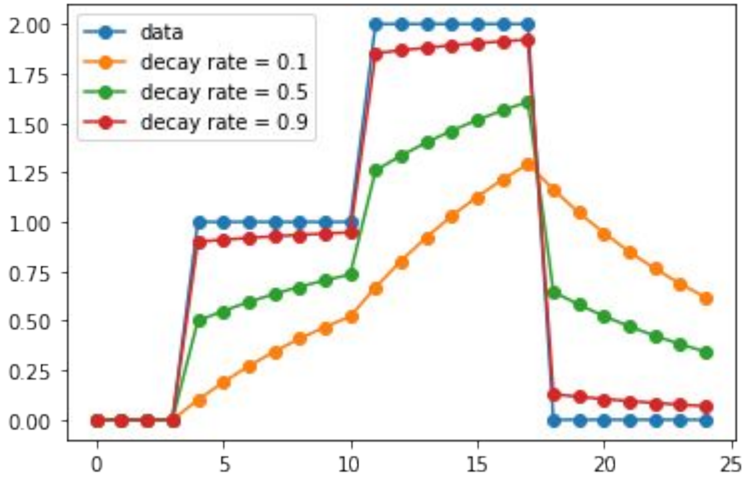
Day	Temperature	Exp average
...	...	10
8	11	
9	8	
10	5	
11	10	
12	8	
13	9	

decay_rate in [0,1]

New value =
Old value * decay_rate
+
New temperature * (1- decay_rate)

Sidetrack: exponential average





Optimizers: gradient descent (reminder)

- TARGET: minimize the cost function J(**Beta**)
- ITERATIVELY:
  - Evaluate the gradient of J(**Beta**)
  - **Delta** = - learning_rate * gradient
  - **Beta** = **Beta** + **Delta**

Optimizers: RMSprop

- TARGET: minimize the cost function J(**Beta**)
- ITERATIVELY:
  - Evaluate the gradient of J(**Beta**)
  - sum_of_gradient_squared =

    previous_sum_of_gradient_squared * decay_rate

    +

    gradient^2 * (1 - decay_rate)
  - **Delta** = - learning_rate * gradient / sqrt(sum_of_gradient_squared)
  - **Beta** = **Beta** + **Delta**

Optimizers: RMSprop

- Intuition:
  - On flat areas (small gradient) delta grows
  - On steep areas (big gradient) delta shrinks
- decay_rate a.k.a. rho
- Keras:
  - model.**compile**(optimizer="rmsprop")
    OR
  - my_opt = tf.keras.optimizers.RMSprop(learning_rate=**0.001**, rho=**0.9**)
    model.**compile**(optimizer = my_opt)

Optimizers: ADAM

- TARGET: minimize the cost function J(**Beta**)
- ITERATIVELY:
  - Evaluate the gradient of J(**Beta**)
  - sum_of_gradient =

    previous_sum_of_gradient * decay_rate1 +
    gradient * (1 - decay_rate1)
  - sum_of_gradient_squared =

    previous_sum_of_gradient_squared * decay_rate2 +
    gradient^2 * (1- decay_rate2)
  - **Delta** = - learning_rate *

    sum_of_gradient / sqrt(sum_of_gradient_squared)
  - **Beta** += **Delta**

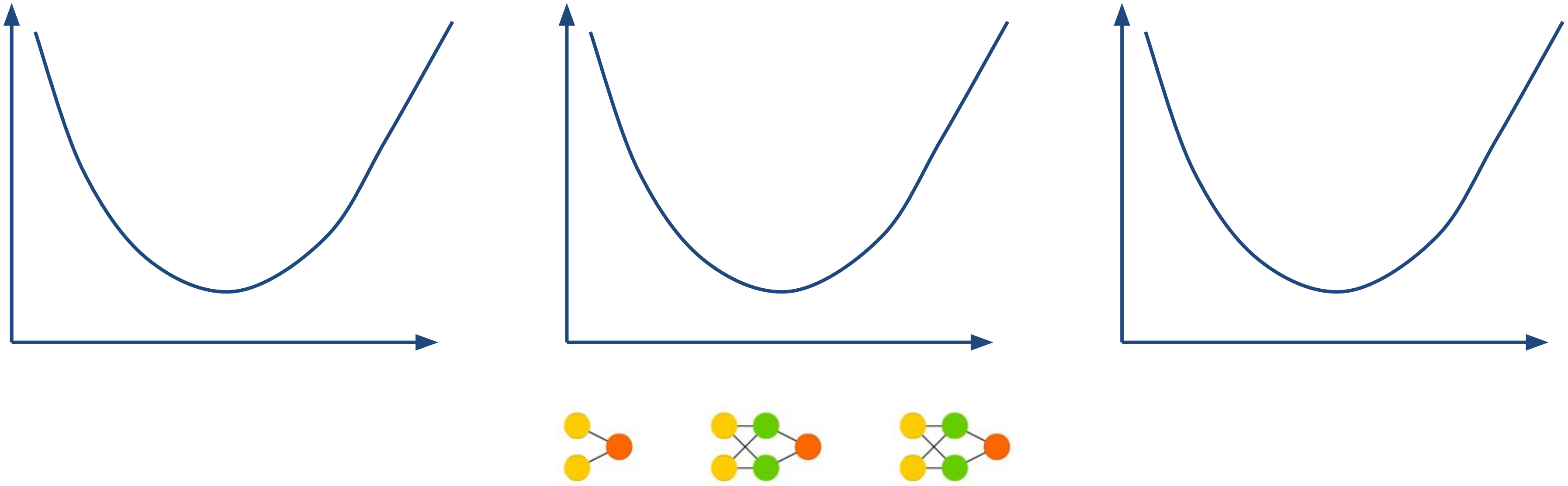
Optimizers: ADAM

- ADAM = RMSprop + Momentum
- decay_rate1 a.k.a. beta_1, decay_rate2 a.k.a. beta_2
- Probably the best optimizer at state of the art
- Keras:
  - model.**compile**(optimizer="Adam")
    OR
  - my_opt = tf.keras.optimizers.adam(learning_rate=**0.001**, beta_1=**0.9**, beta_2=**0.999**)
    model.**compile**(optimizer = my_opt)

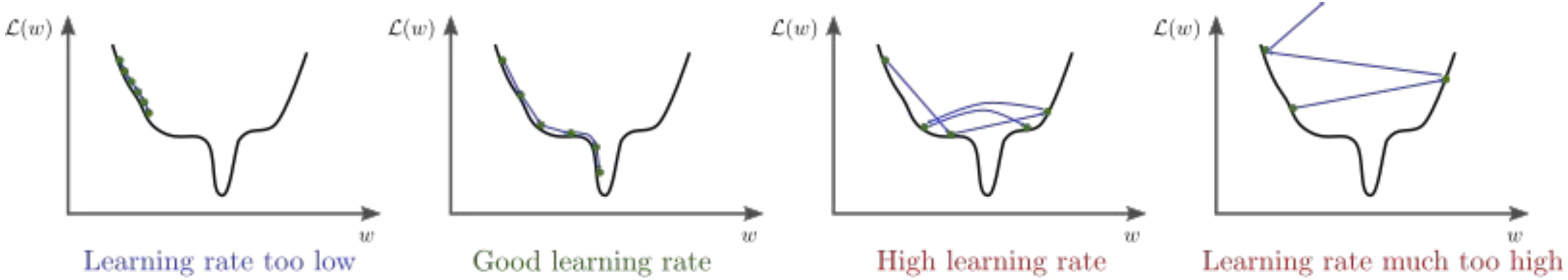
The most important hyperparameter: LR

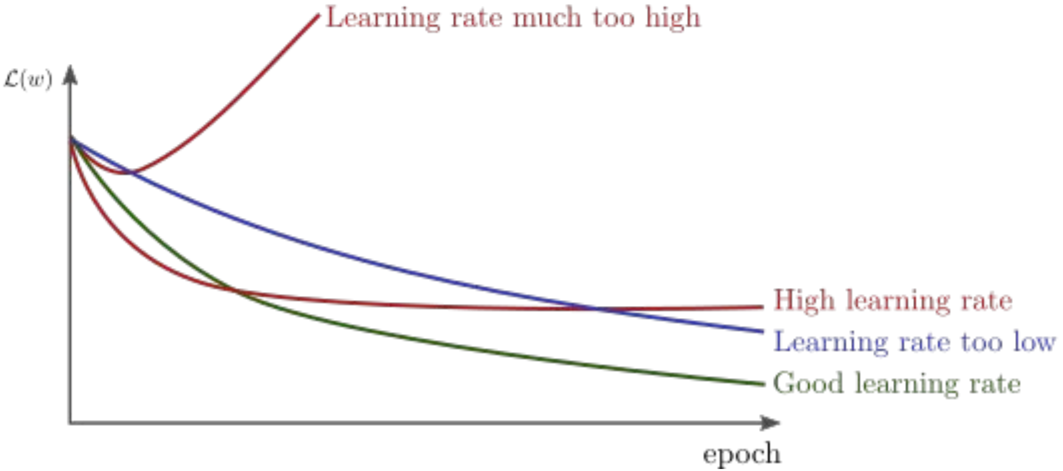
- Learning Rate choice is **by far** the most influential
- Unfortunately there’s no general strategy
  - Depends on the datascape
  - What is good for some dataset is terrible for others
- Fortunately, you get used to it

The most important hyperparameter: LR



The most important hyperparameter: LR





https://www.bdhammel.com/learning-rates/

[REF]

- An intuitive guide to CNN https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/
- Visual comparison of optimizers: https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c
- Regularization techniques: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/
- On effect of learning rates: https://www.bdhammel.com/learning-rates/