

Convolutional Neural Networks

Microsoft London ML OpenHack

Tim Scarfe, Ph.D

Principal Software Engineer @ Microsoft
Commercial Software Engineering (CSE)
@ecsquendor
youtube.com/machinelearningatmicrosoft

MACHINE LEARNING @ MICROSOFT





Commercial Software Engineering

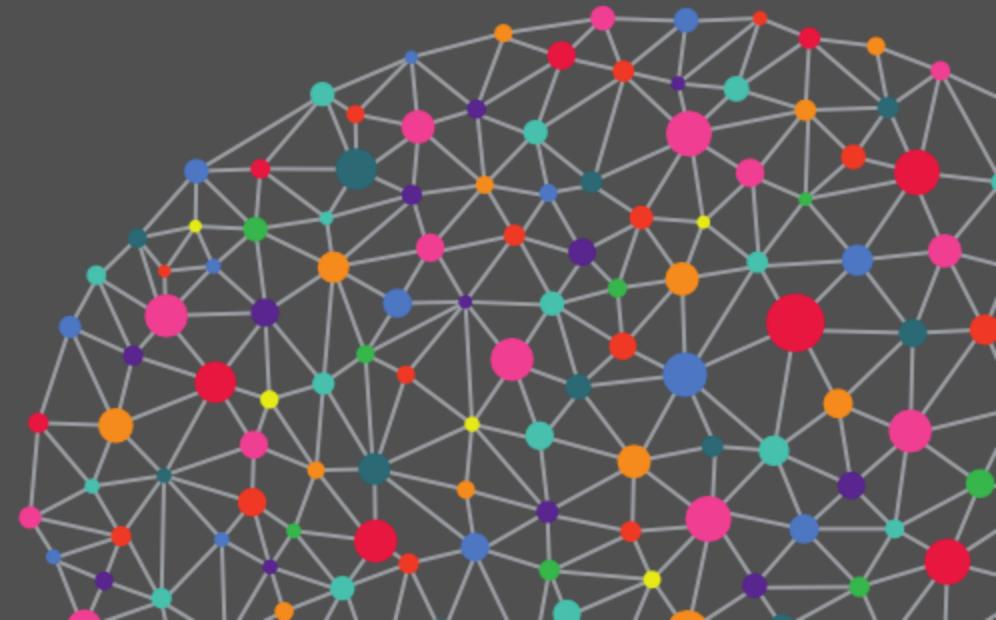


Confused.^{com}
On a mission to save you money

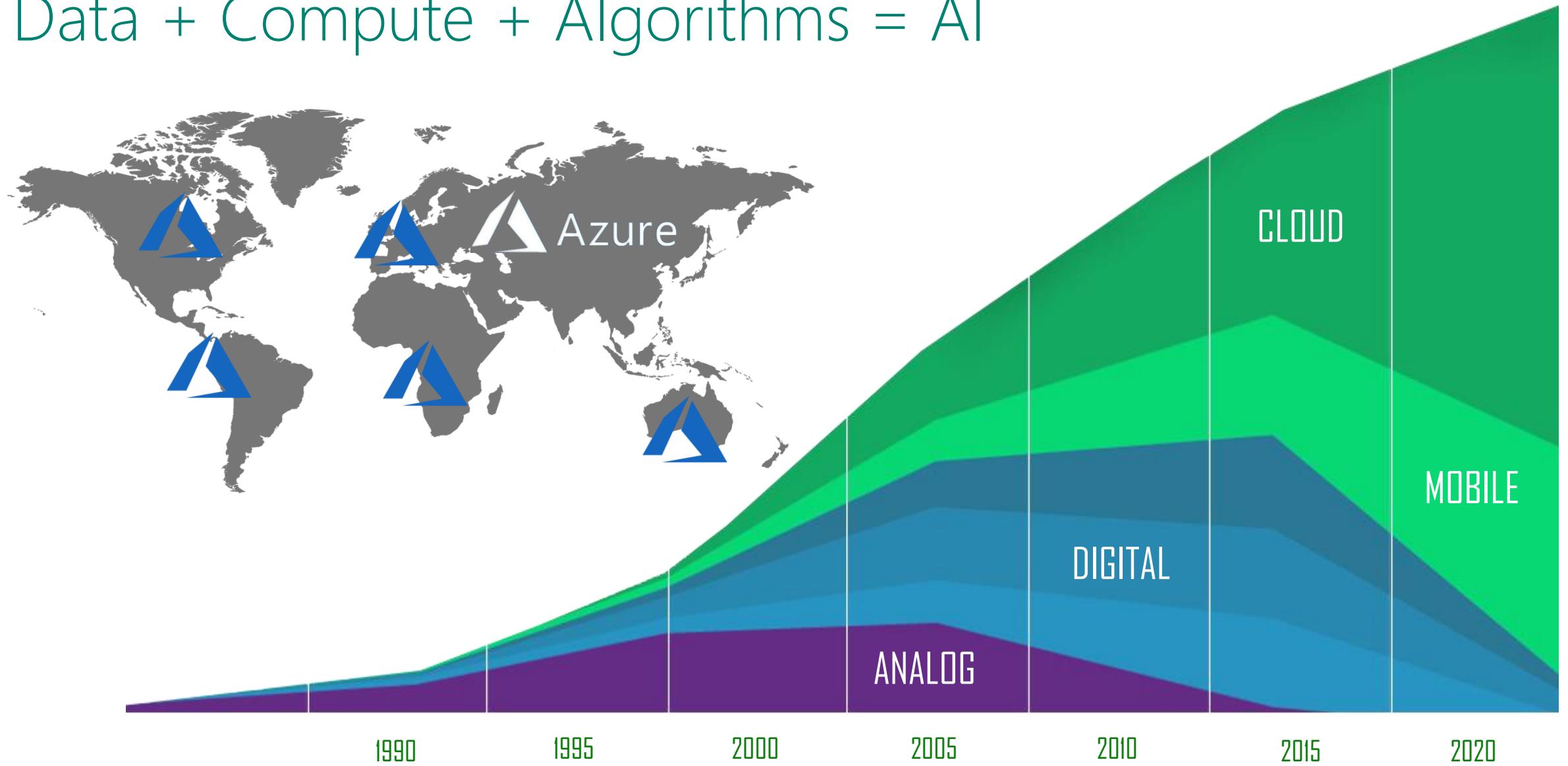
TALK OUTLINE



- High level concepts of deep learning
- Advanced topics
- Old days of vision
- How does convolution work?
- MNIST example in Keras code
- Data Augmentation on Cats+Dogs dataset
- Transfer Learning
- Visualising CNNs



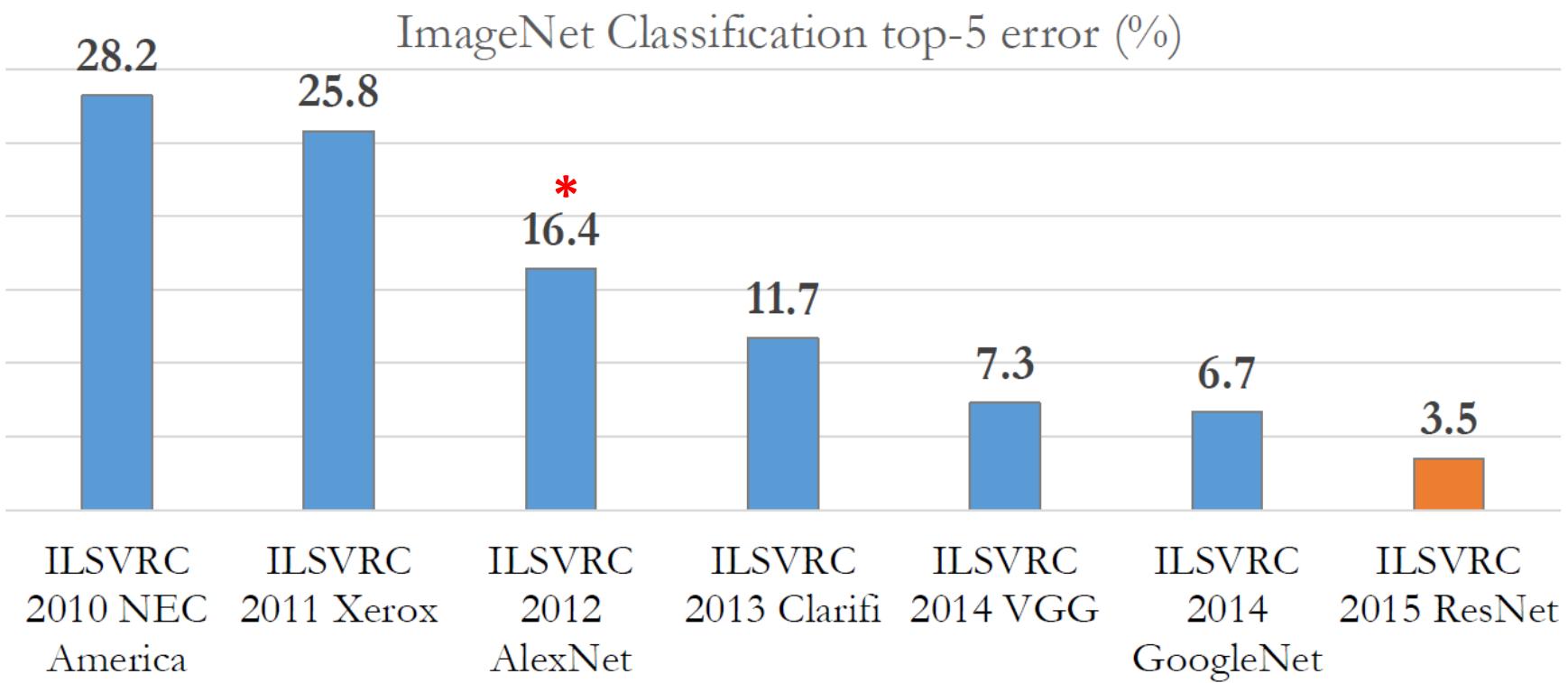
Data + Compute + Algorithms = AI



ImageNet

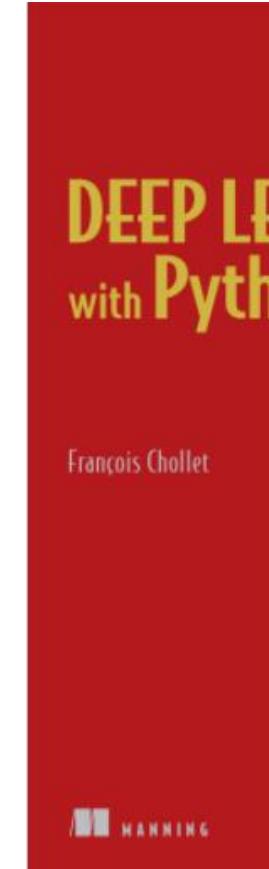


Alex Krizhevsky



- Alex Krizhevsky wins the ImageNet competition in 2012
- AlexNet
- Heralds a new era of AI
- Same story in language processing, speech recognition etc

Francois Chollet. The man. The legend. Please buy his book!



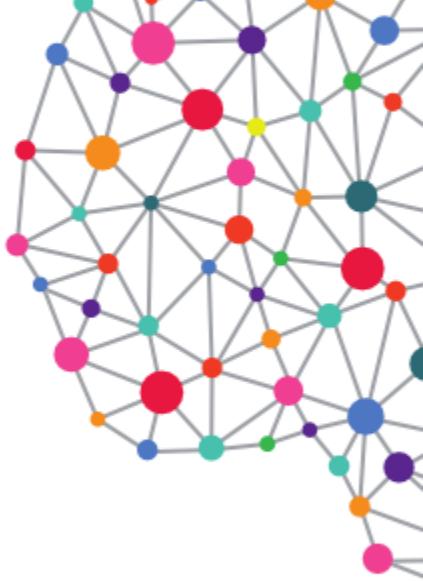
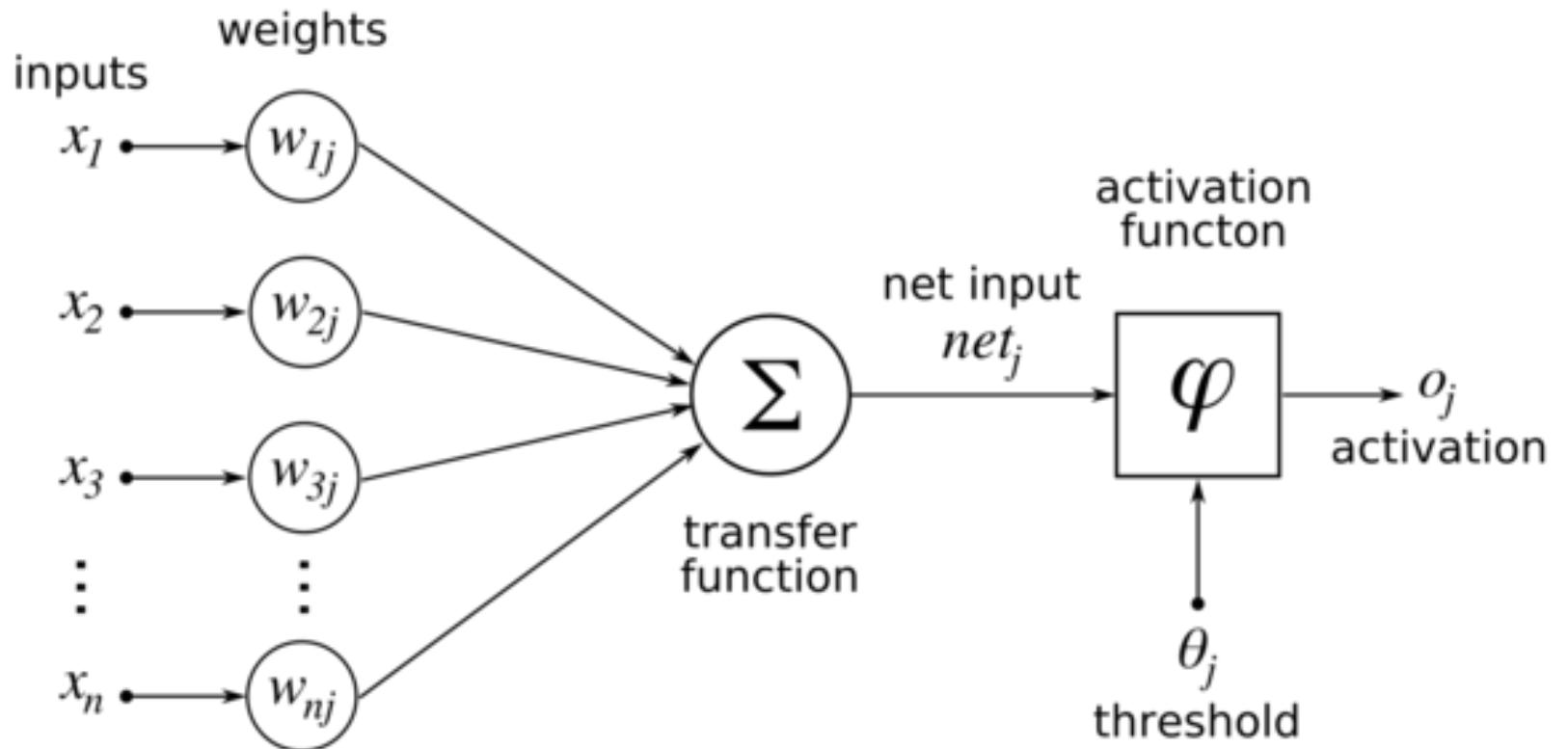
<https://github.com/fchollet/deep-learning-with-python-notebooks/>

<https://www.manning.com/books/deep-learning-with-python>

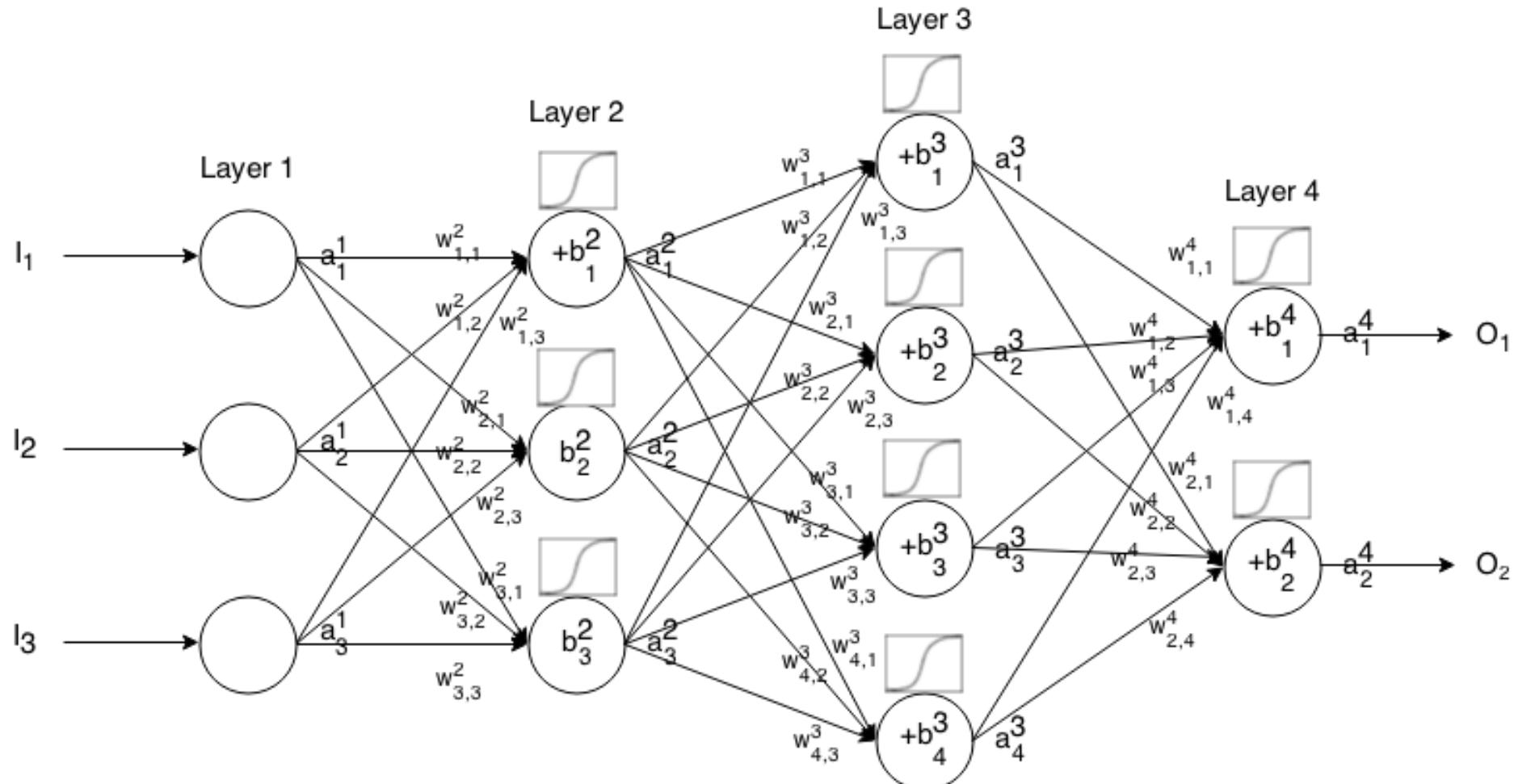
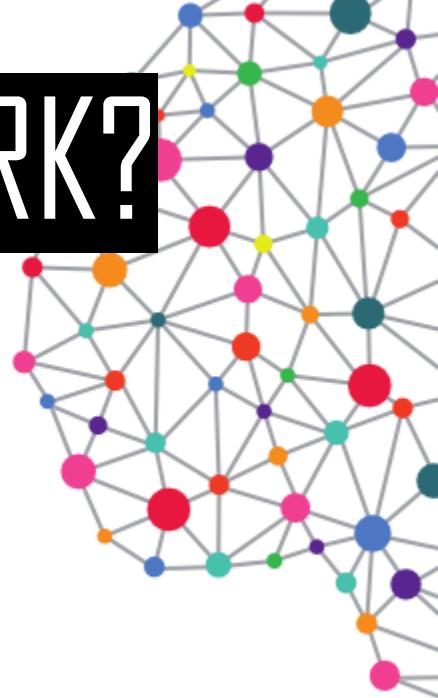
I have marked his material with © Francois Chollet

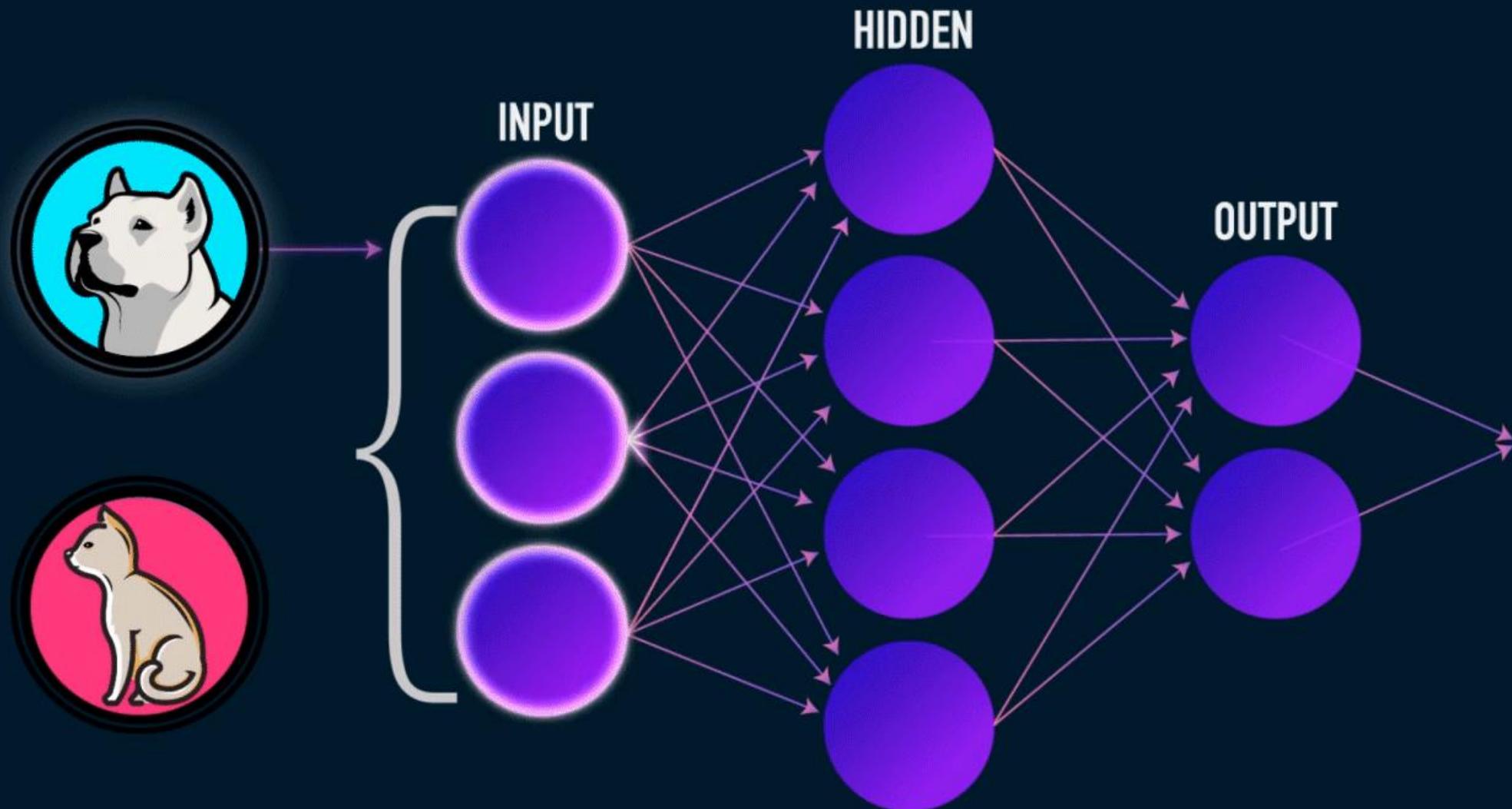


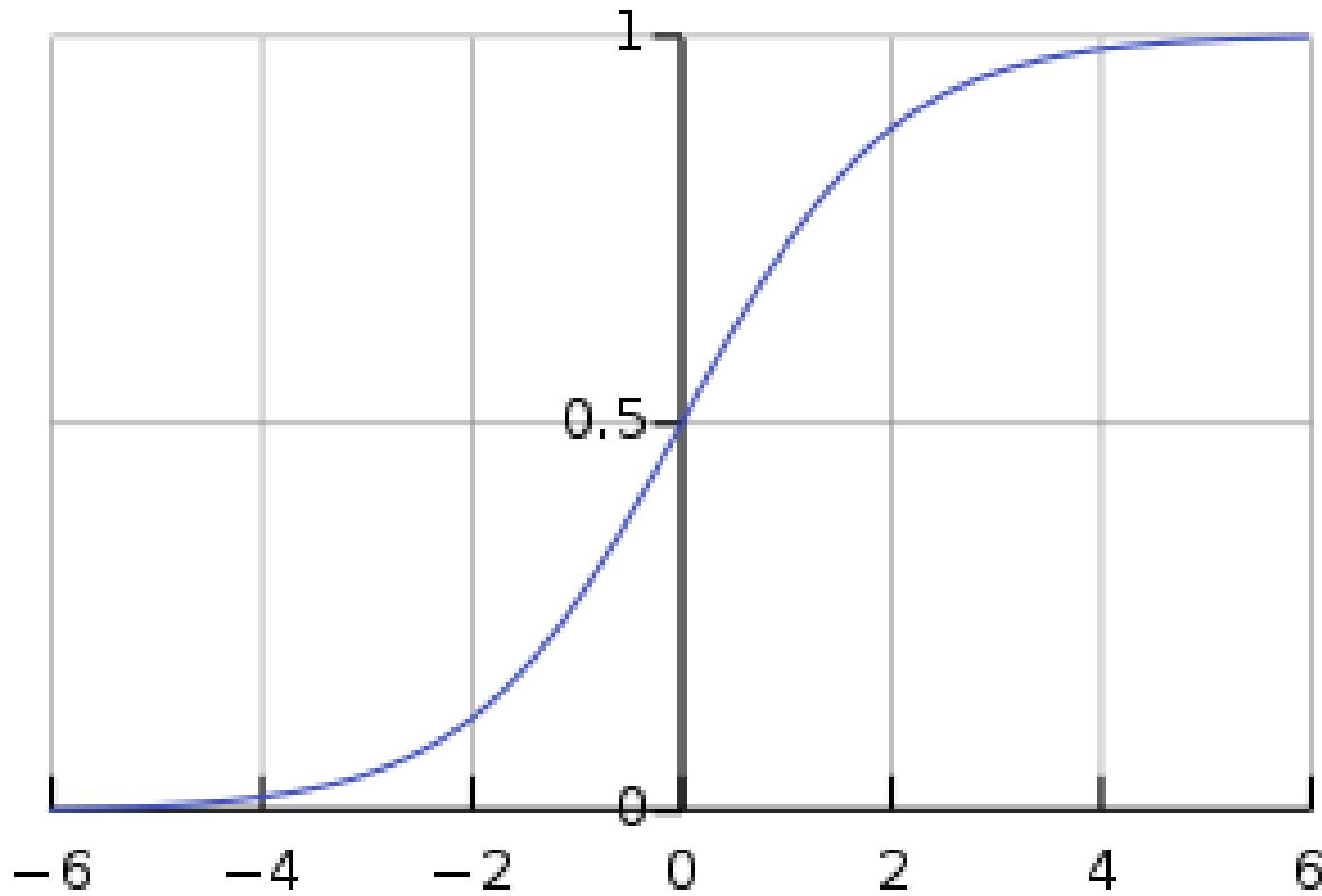
WHAT IS A NEURAL NETWORK?



WHAT IS A (DENSE) DEEP NEURAL NETWORK?

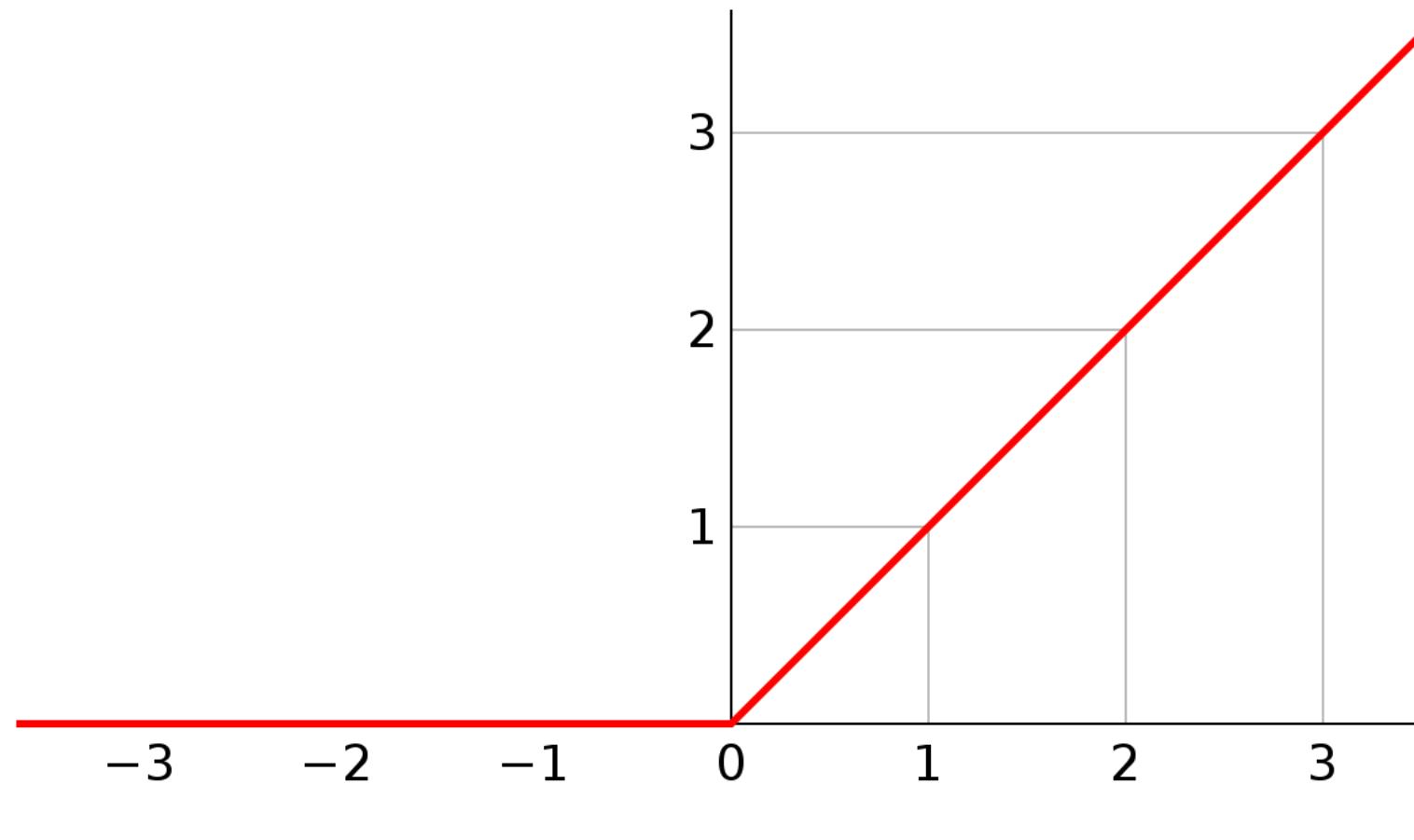




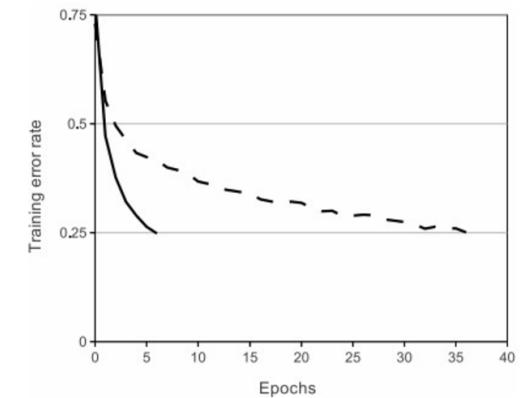


$$\frac{e^x}{e^x + 1}$$

SIGMOID SQUASHING FUNCTION



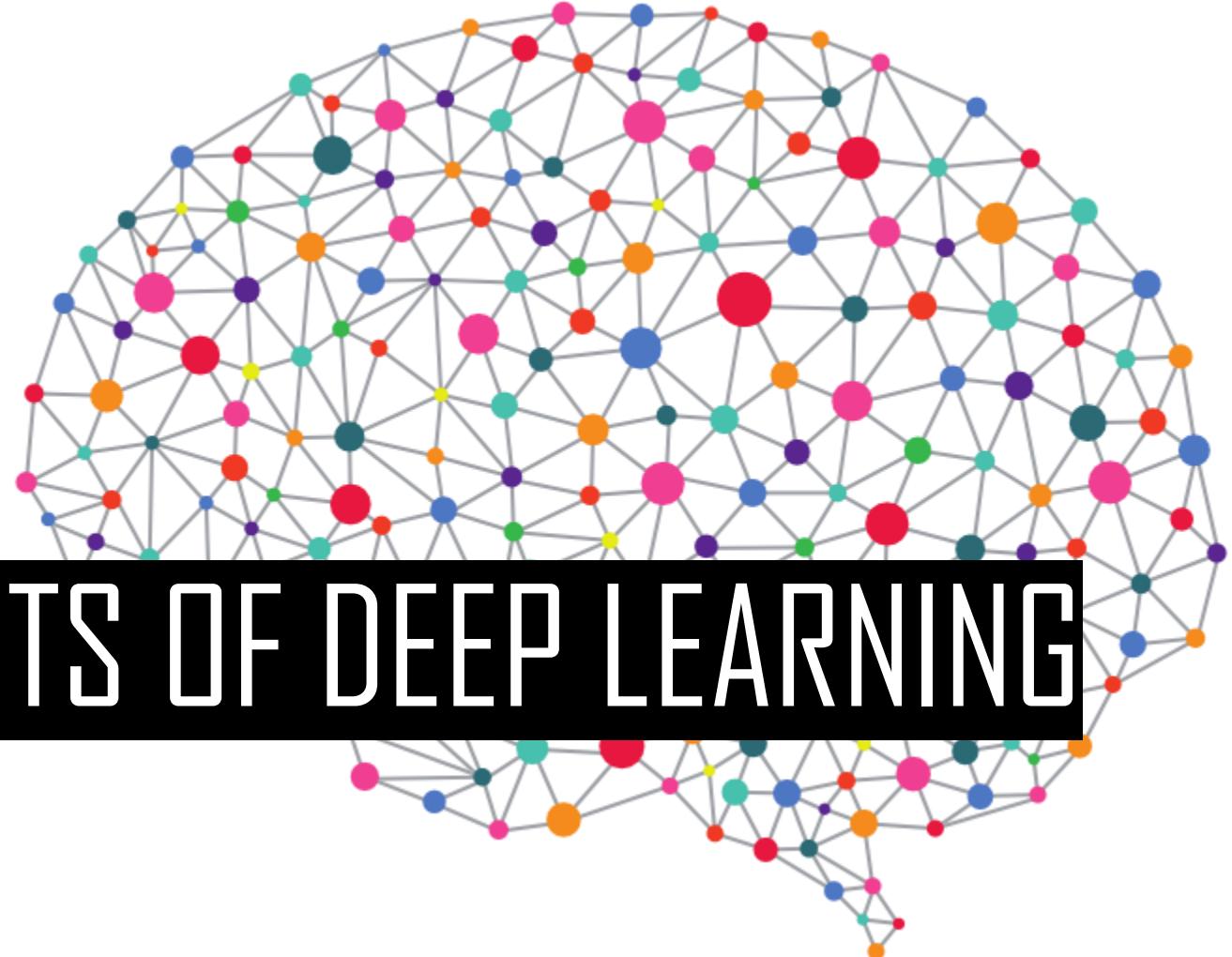
$$f(x) = x^+ = \max(0, x)$$



RELU SQUASHING FUNCTION

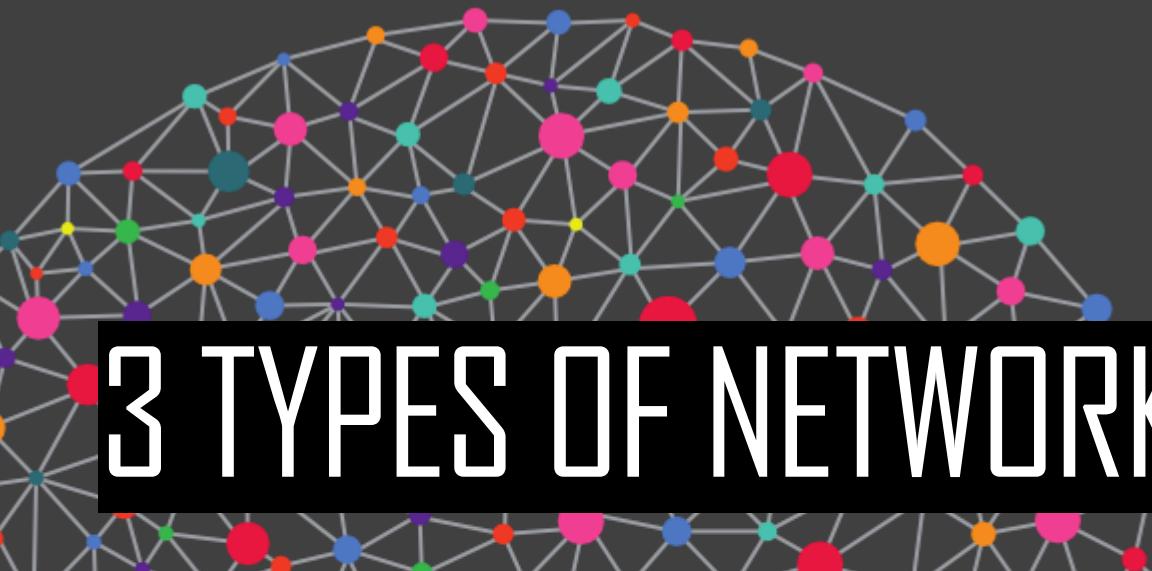


DISTILLED CONCEPTS OF DEEP LEARNING





1. DENSELY CONNECTED NETWORKS
2. CONVOLUTIONAL NEURAL NETWORKS (MODEL SPACE)
3. RECURRENT NEURAL NETWORKS (MODEL TIME)



3 TYPES OF NETWORK ARCHITECTURE



Traditional ML

Manual Features



Trainable Classifier



Deep Learning

Representations are hierarchical and trained automatically

Low Level Features



Mid Level Features

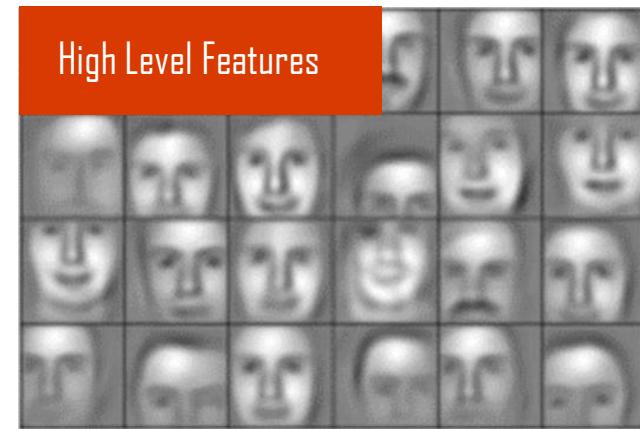
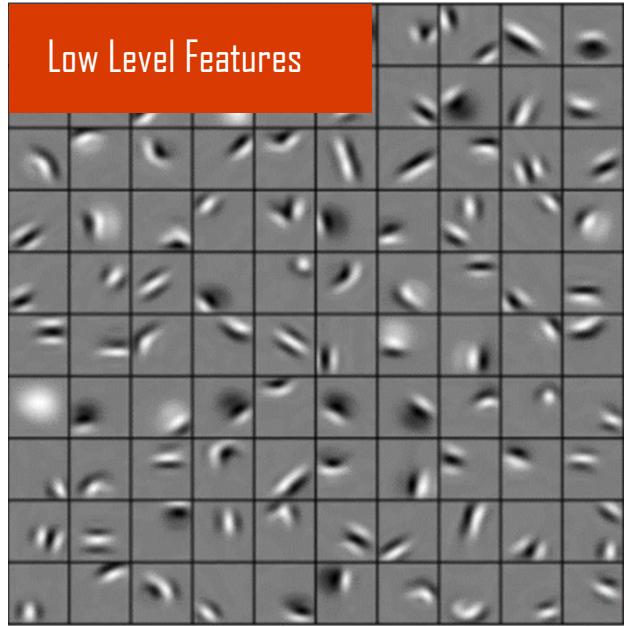


High Level Features



Trainable Classifier

Deep Learning = learning a representation



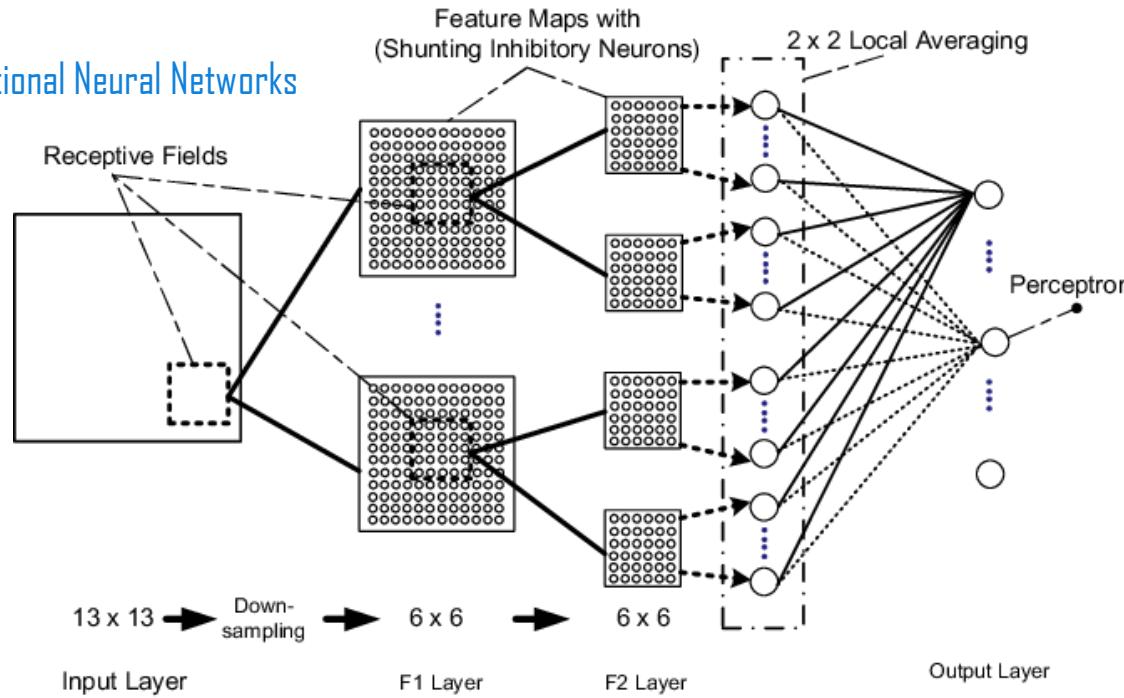
Representations are learned automatically



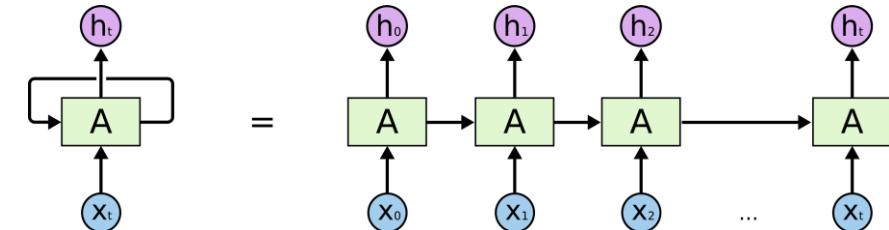
Unlike other algorithms, NNs can natively encode useful and obvious relationships in the data domain

- Local spatial dependencies (vision) i.e. CNNs
- Time dependencies (language, speech) i.e. RNNs

Convolutional Neural Networks



Recurrent Neural Networks



NATIVE DATA-DOMAIN FEATURES

Universal functional approximators

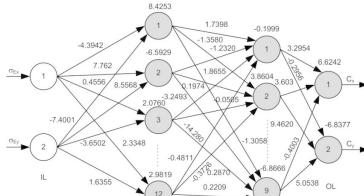
SEQUENCES

SPEECH

LANGUAGE

IMAGES

[...]



SEQUENCES

SPEECH

LANGUAGE

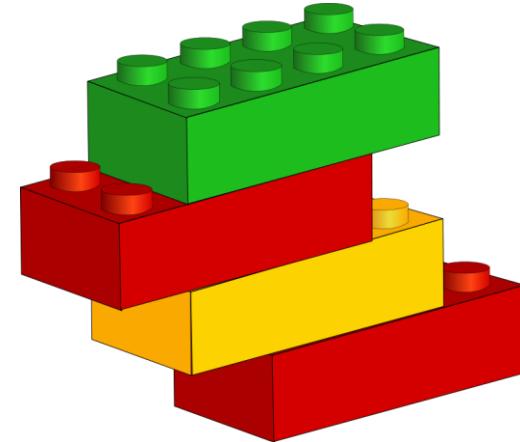
IMAGES

[...]

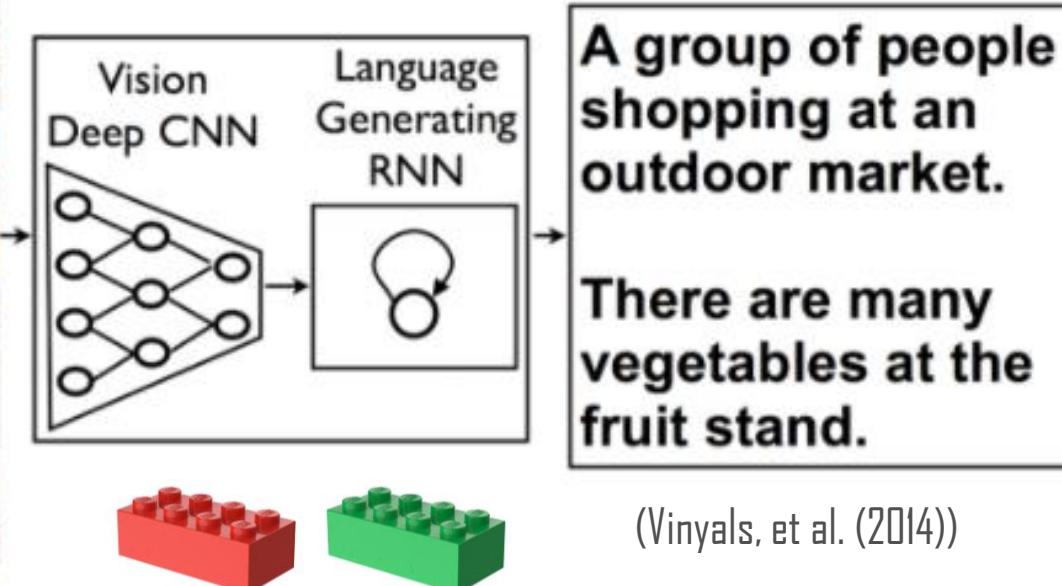
- Map between domains
- Temporal & spatial modelling
- Generative models possible!
- Multiple inputs and outputs!

Composability

- Machine learning is becoming a form of software development
- Machine learning models are like software
- There is a dichotomy between the software engineering process and the data science process



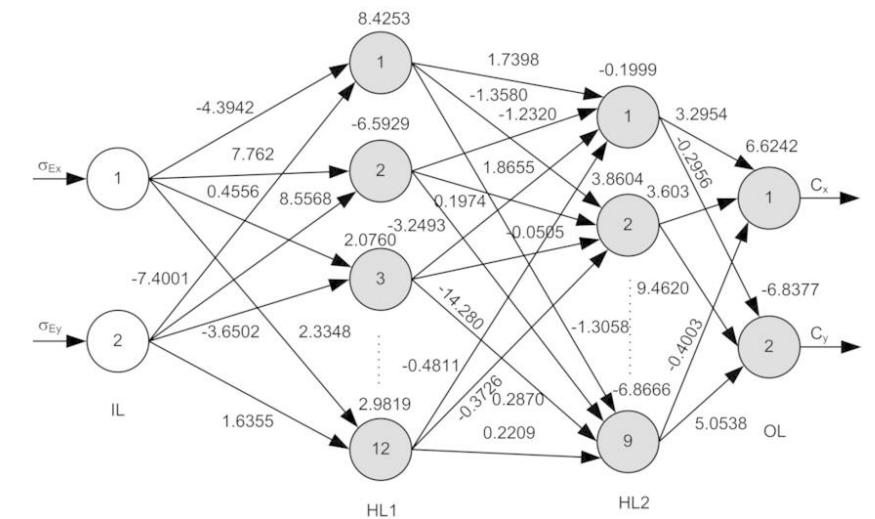
Building predictive architectures like lego blocks



(Vinyals, et al. (2014))

We call this “Software 2.0”

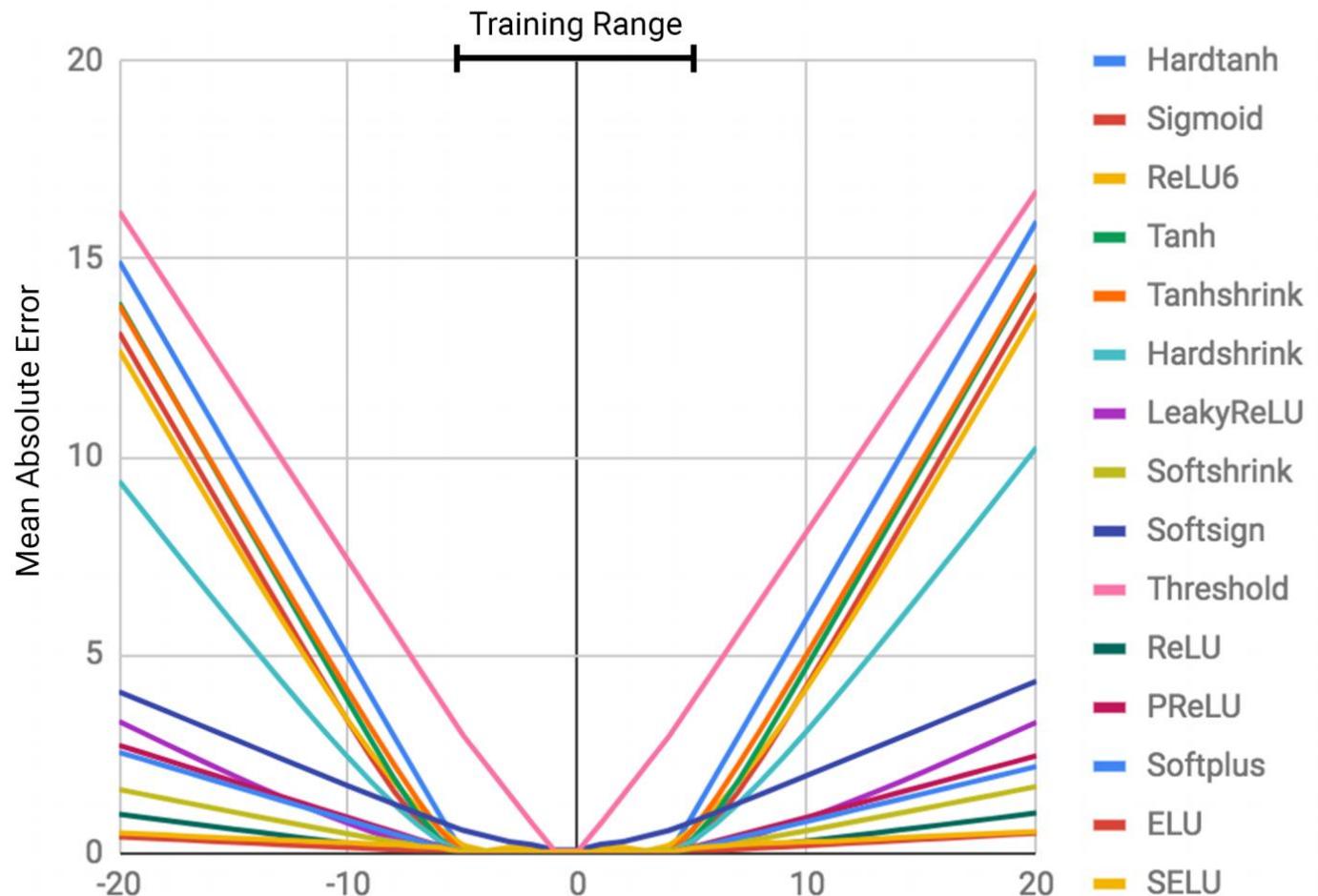
- Software 2.0 is a confection of data generated code and handwritten code
- Some features are **impossible** to write with traditional code i.e. speech synthesis, machine translation, computer vision use cases, game AI
- Computationally homogeneous
- Simple to bake into silicon.
- It is highly portable.
- Composable
- Better than you



Deep learning is not magic; interpolation vs extrapolation

ML algorithms perform very, very badly on data out of range. They don't generalise as well as you think. Trask et al 2018 (DeepMind)

$$f(x)=x$$



When to use deep learning vs classical machine learning

- Large volumes of data
- Unstructured data i.e. images, video, sound
- Modelling temporal or spatial dependencies
- Novel prediction architectures
- Composability, reusability, transferability of models
- Consistency of approach is more conducive to software engineering

When not to use deep learning

- When interpretability is important
- When the data is very small deep learning behaves like classical machine learning i.e. no representation learning
- When you need statistical guarantees

Machine learning models are becoming ubiquitous in software

Software is comprised of some things you can write with code, and some parts which must be generated from data i.e. ML



More companies are becoming data companies

- Companies are becoming data driven
- Diversifying, creating data products & services

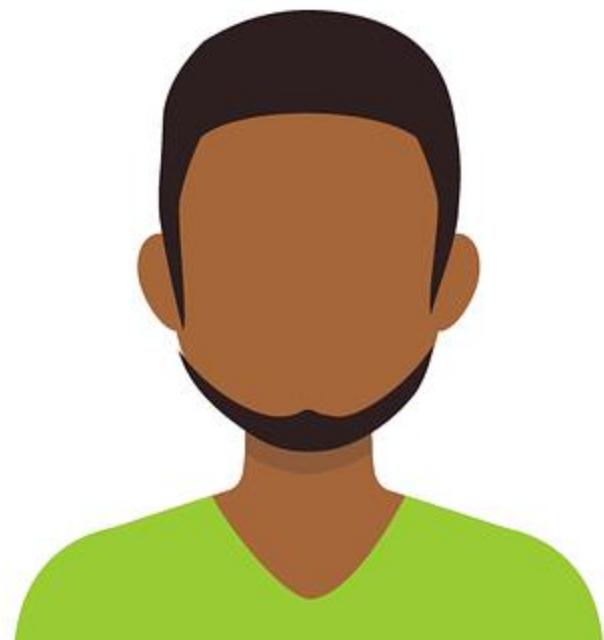


Culture clash in software engineering

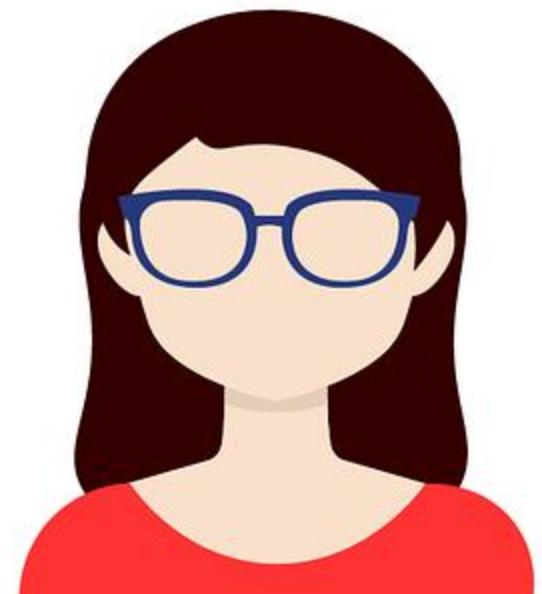
- Data scientists now ship directly
- DevOps in Data Science is *much* harder



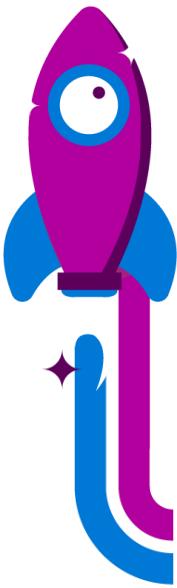
Data Scientist



Software Engineer



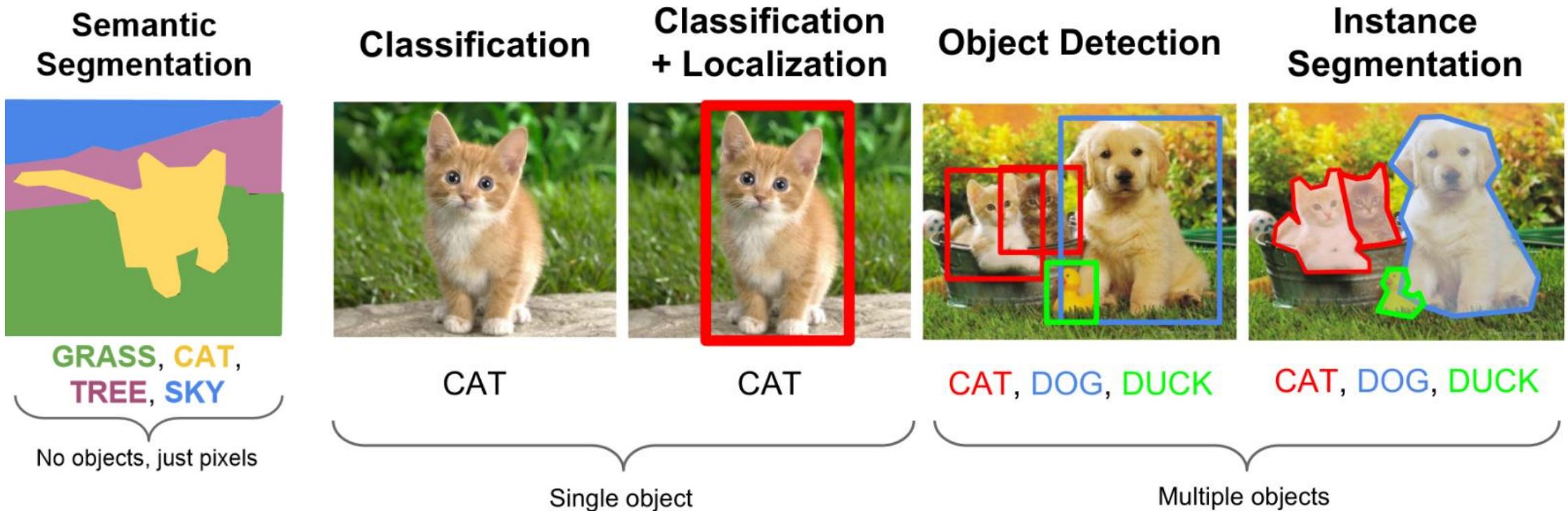
Operations



What can you do with computer vision?



DISCRIMINATIVE ARCHITECTURES



GENERATIVE ARCHITECTURES



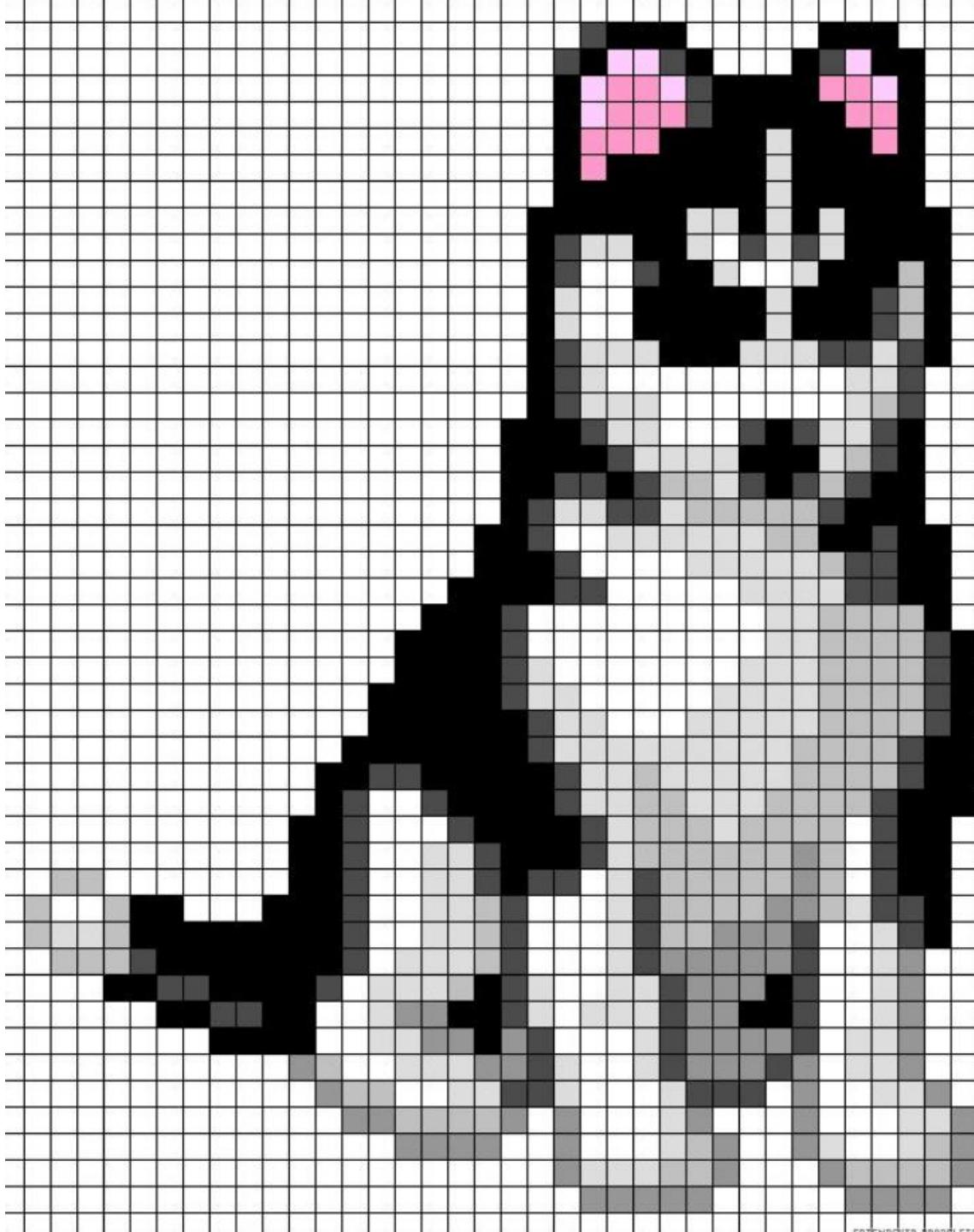


How did we used to do computer vision?



PIXEL-WISE LEARNING

- Not “translation” invariant
- Not scale invariant
- Not rotational invariant
- Affine transformations?



In [32]:

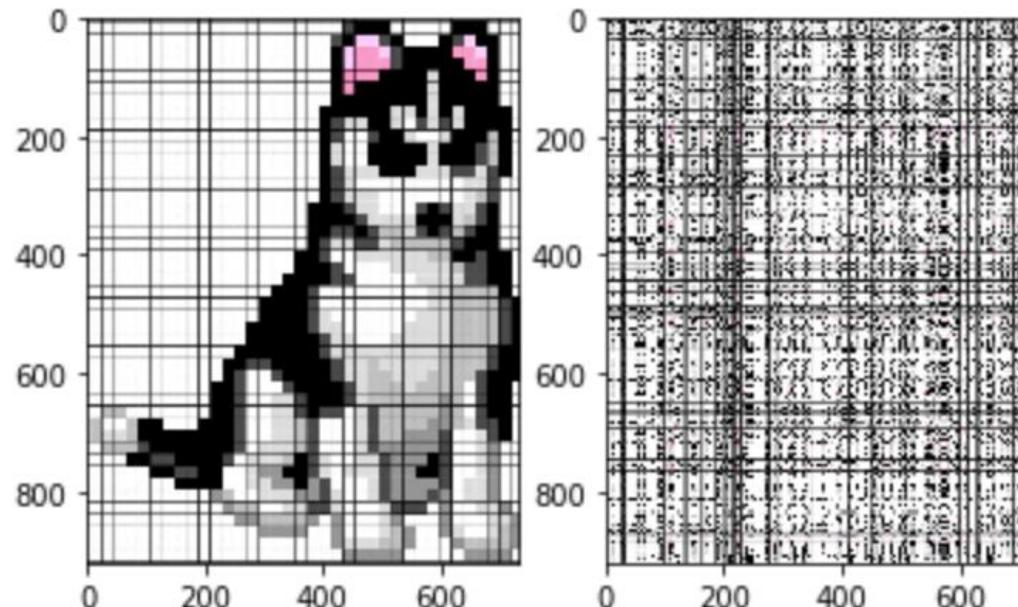
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img=mpimg.imread('dog.png')

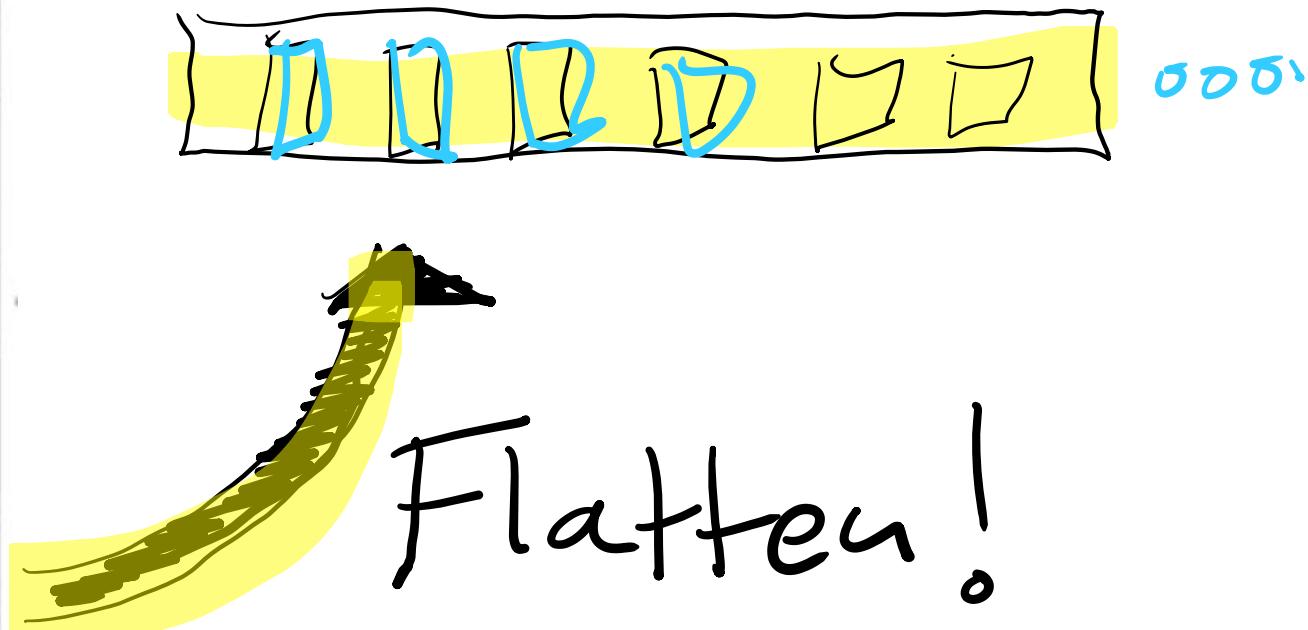
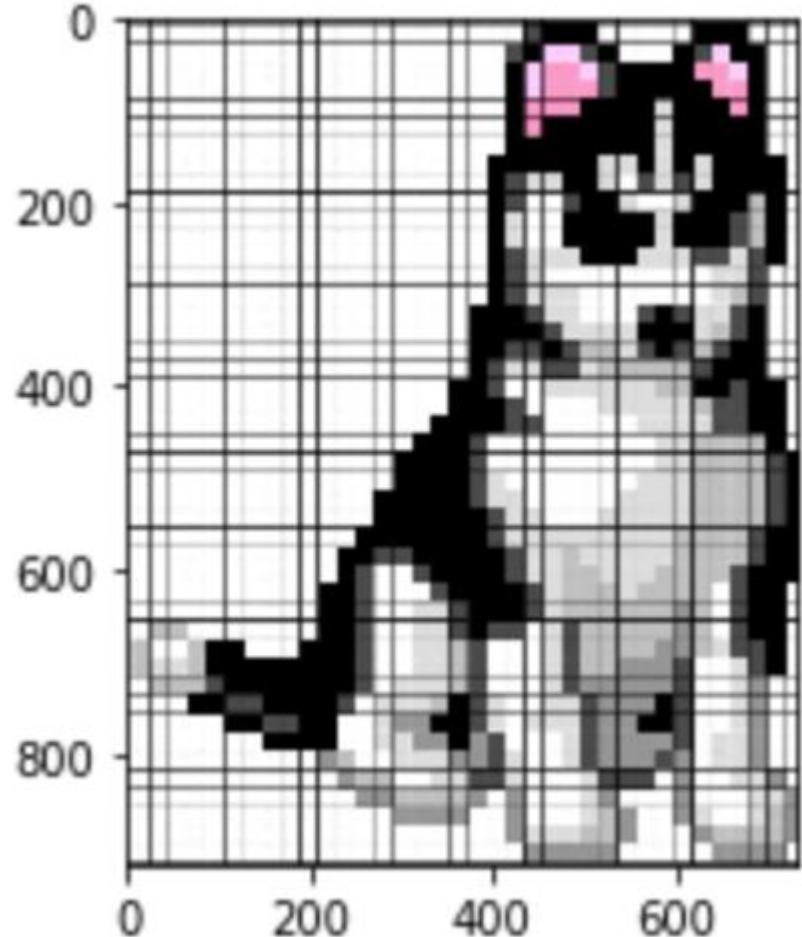
plt.subplot(121)
imgplot = plt.imshow(img)

imgshuffle = np.take(img,np.random.permutation(img.shape[1]),axis=1,out=img)
imgshuffle = np.take(imgshuffle,np.random.permutation(img.shape[0]),axis=0,out=img)
plt.subplot(122)
plt.imshow(imgshuffle)
```

Out[32]: <matplotlib.image.AxesImage at 0x25384e0ff98>



WHAT A CLASSICAL ALGORITHM SEES



HISTOGRAMS & FEATURES

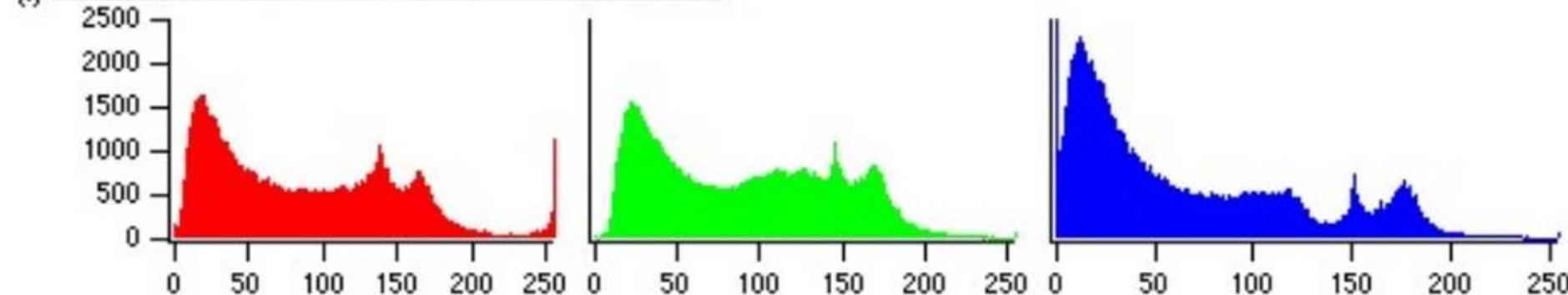
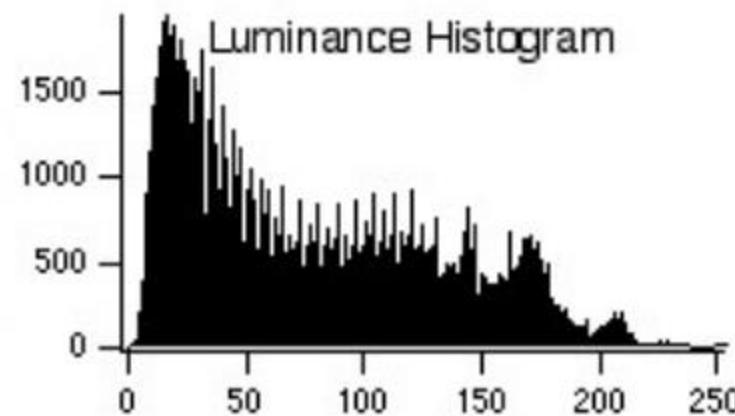
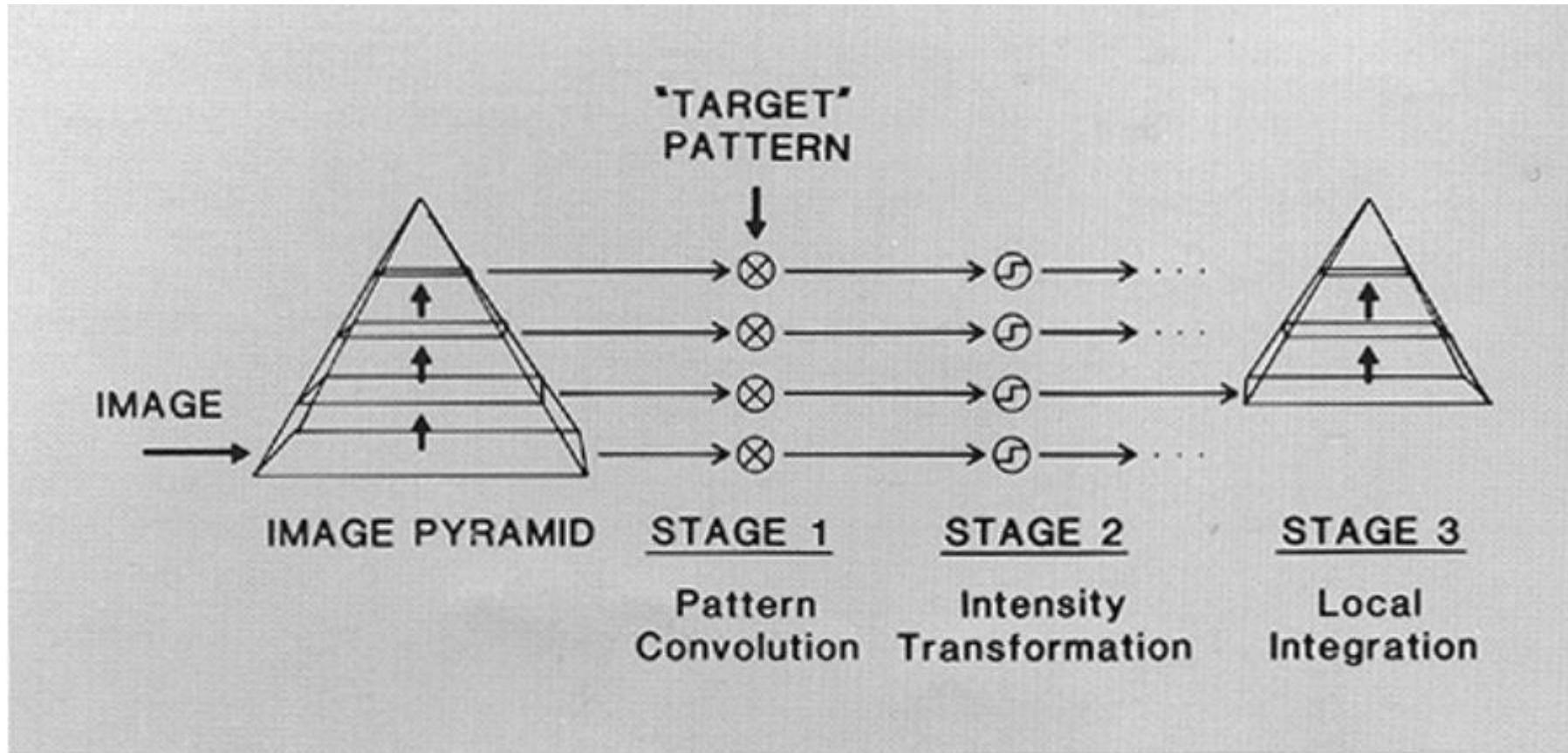
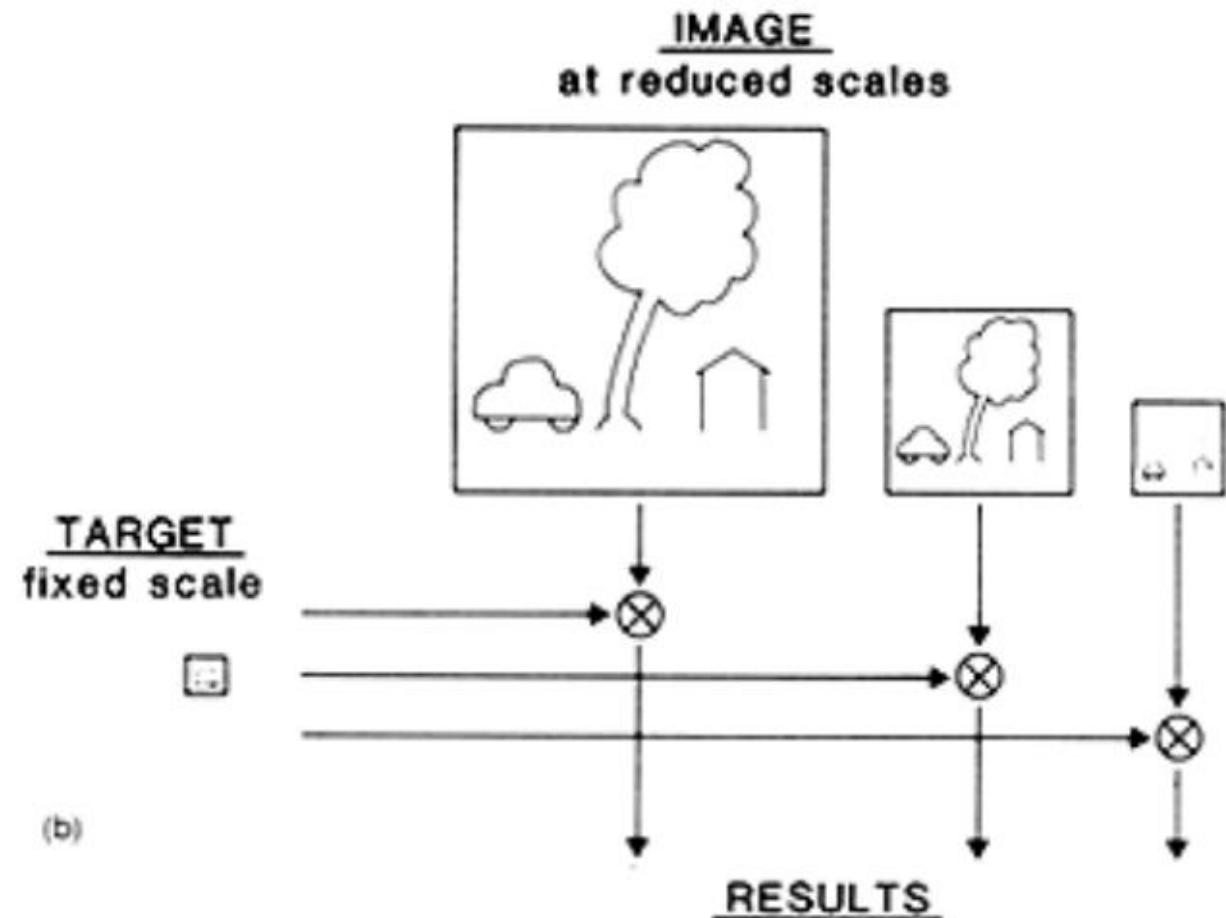
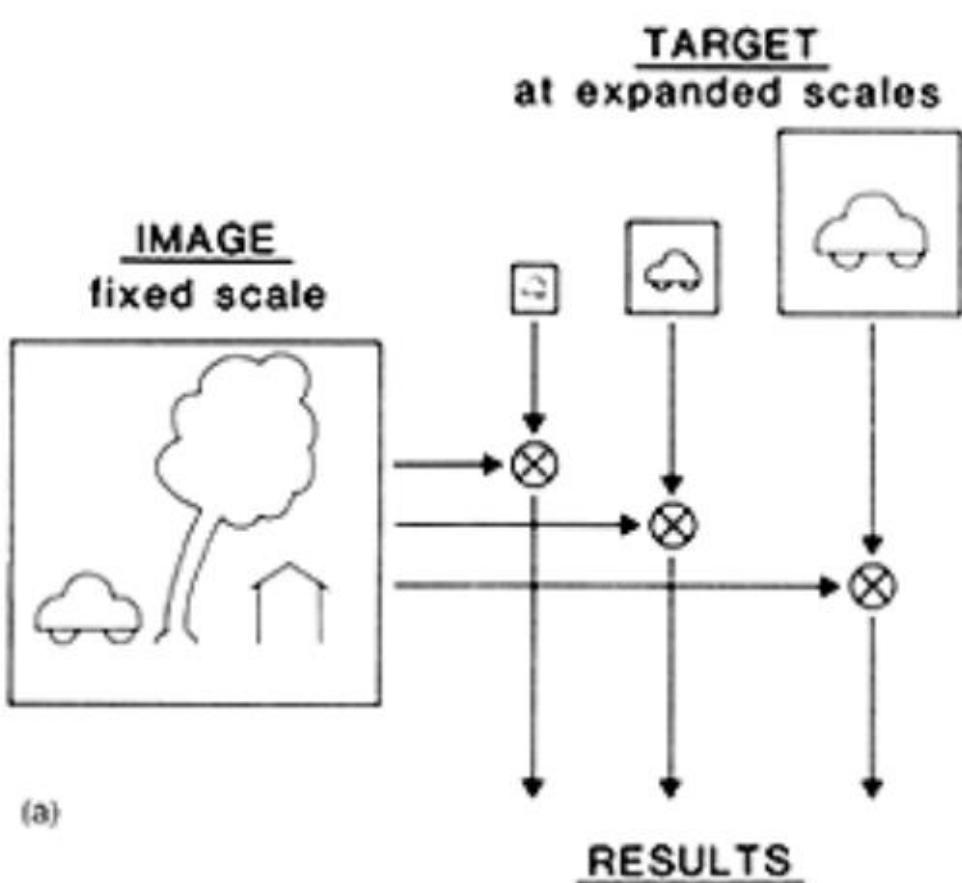


IMAGE PYRAMID (I)



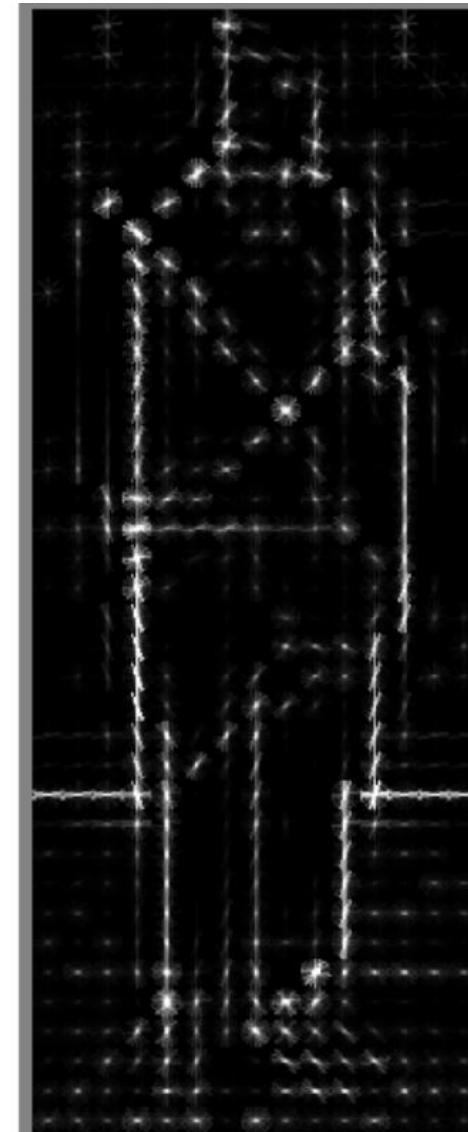
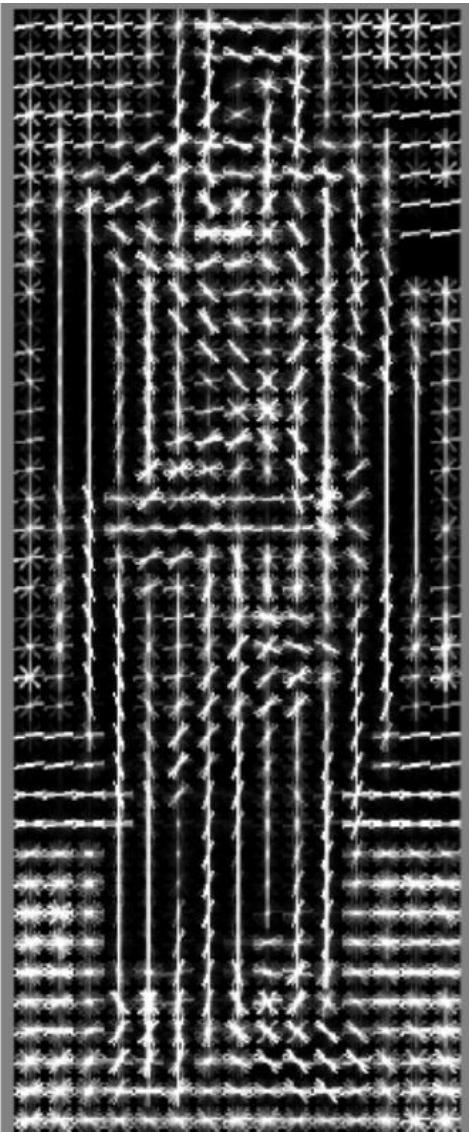
Adelson et al '84

IMAGE PYRAMID (2)



Adelson et al '84

HISTOGRAMS OF GRADIENTS (HOG)



EDGE DETECTION (WITH CONVOLUTION)

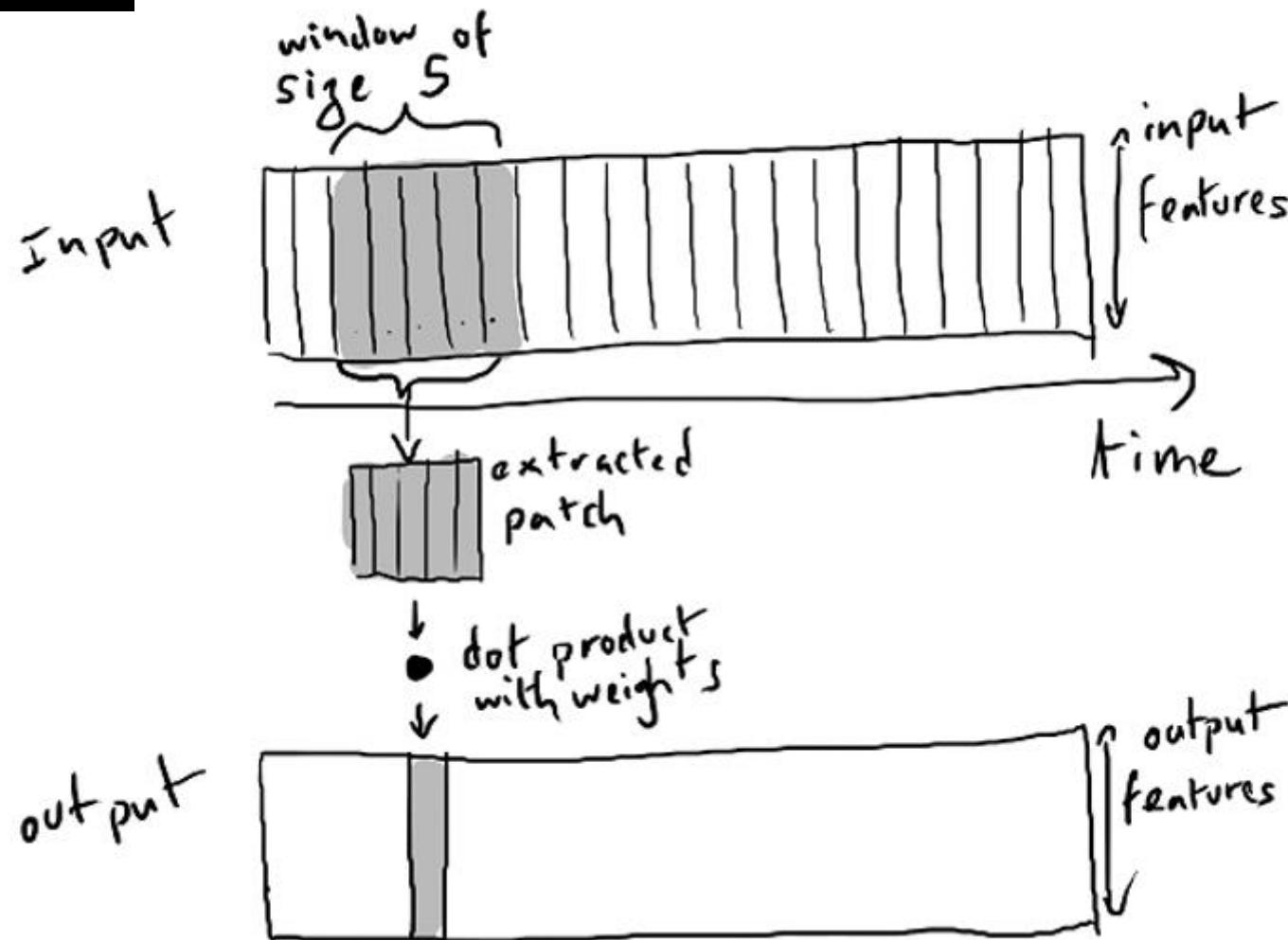




How do convolutional neural networks work?

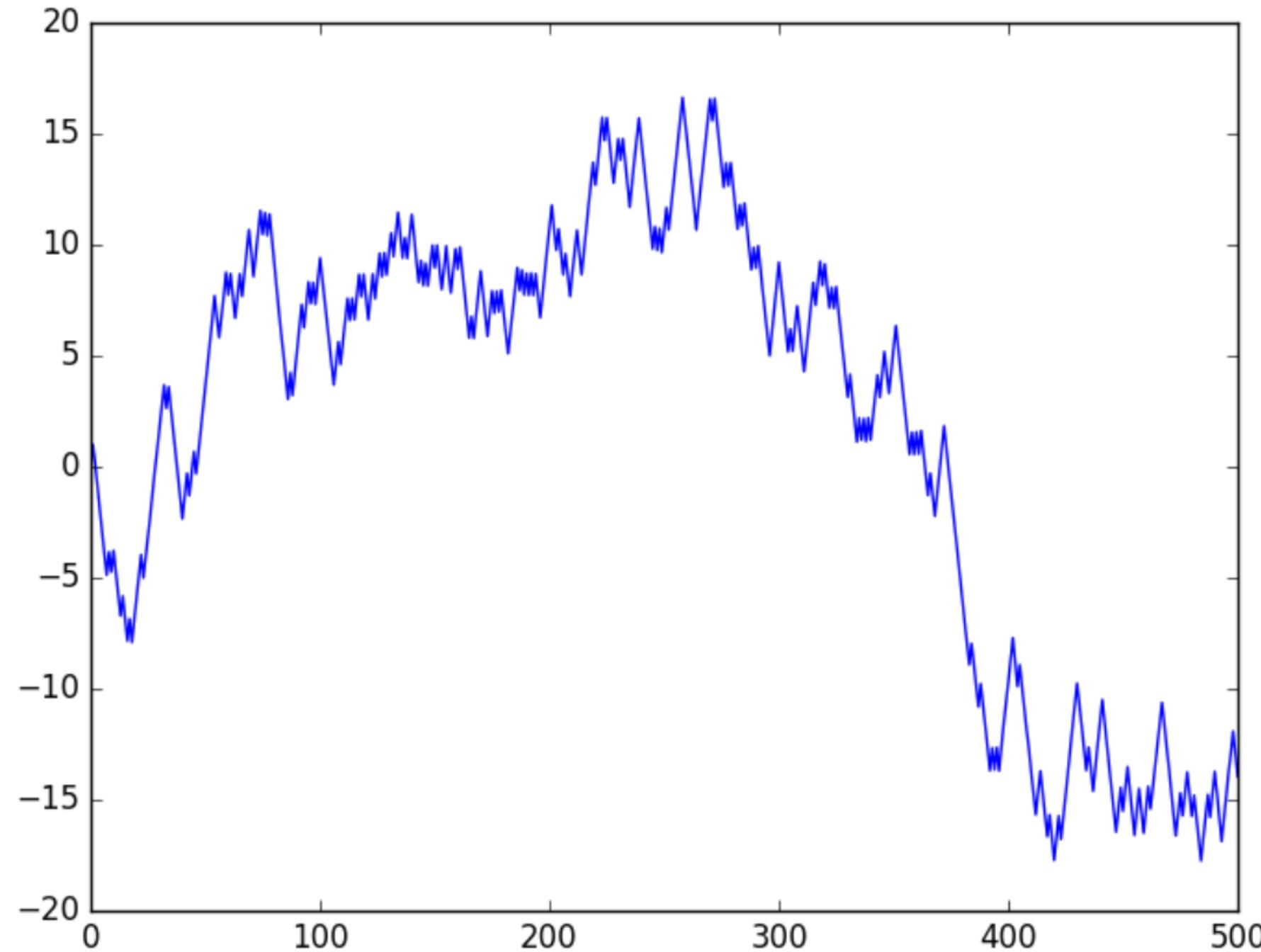


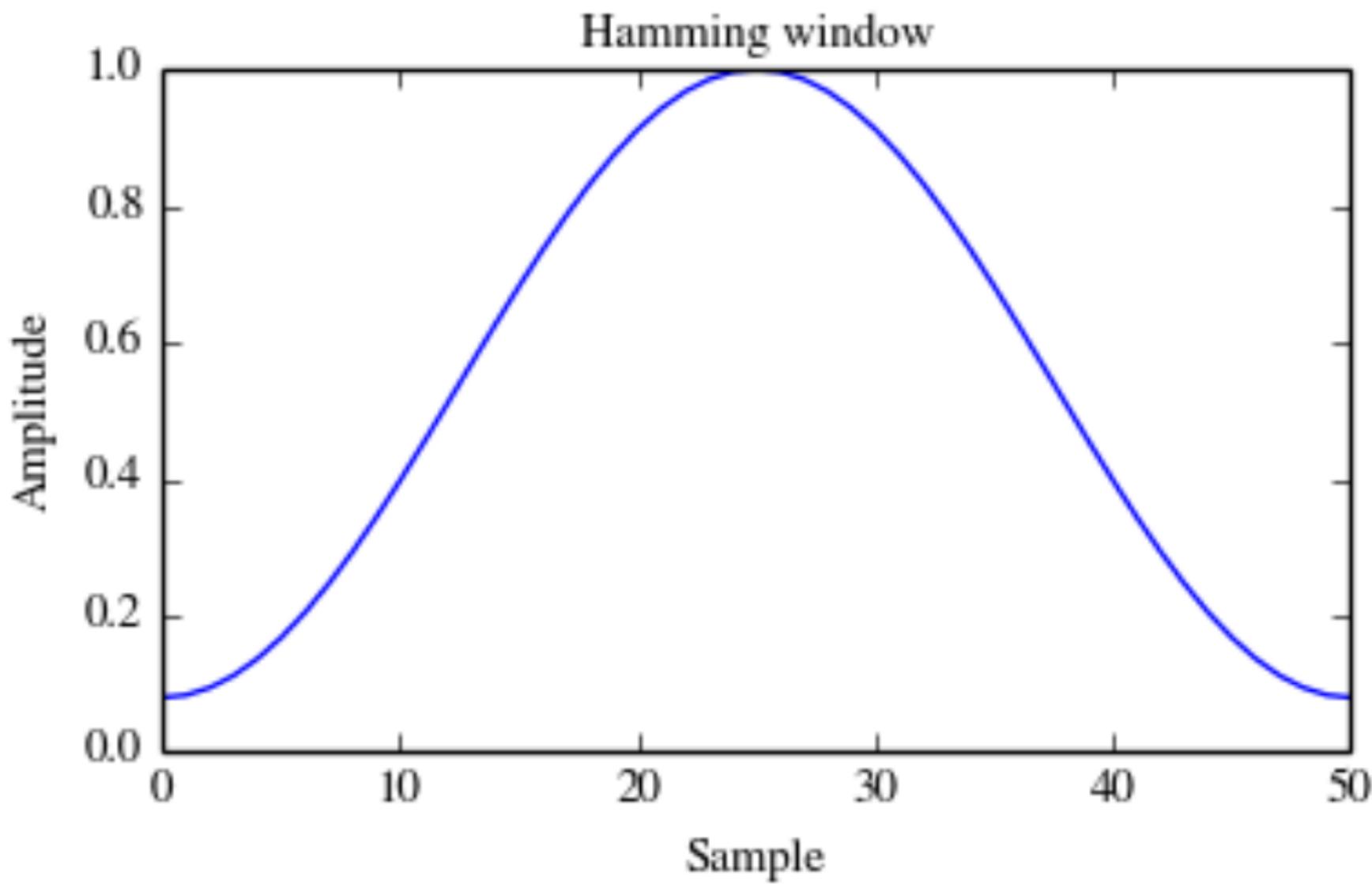
CONVOLUTION



© Francois Chollet

```
1 x = [0]
2
3 for j in range(500):
4     step_x = random.randint(0,1)
5     if step_x == 1:
6         x.append(x[j] + 1 + 0.05*np.random.normal())
7     else:
8         x.append(x[j] - 1 + 0.05*np.random.normal())
```

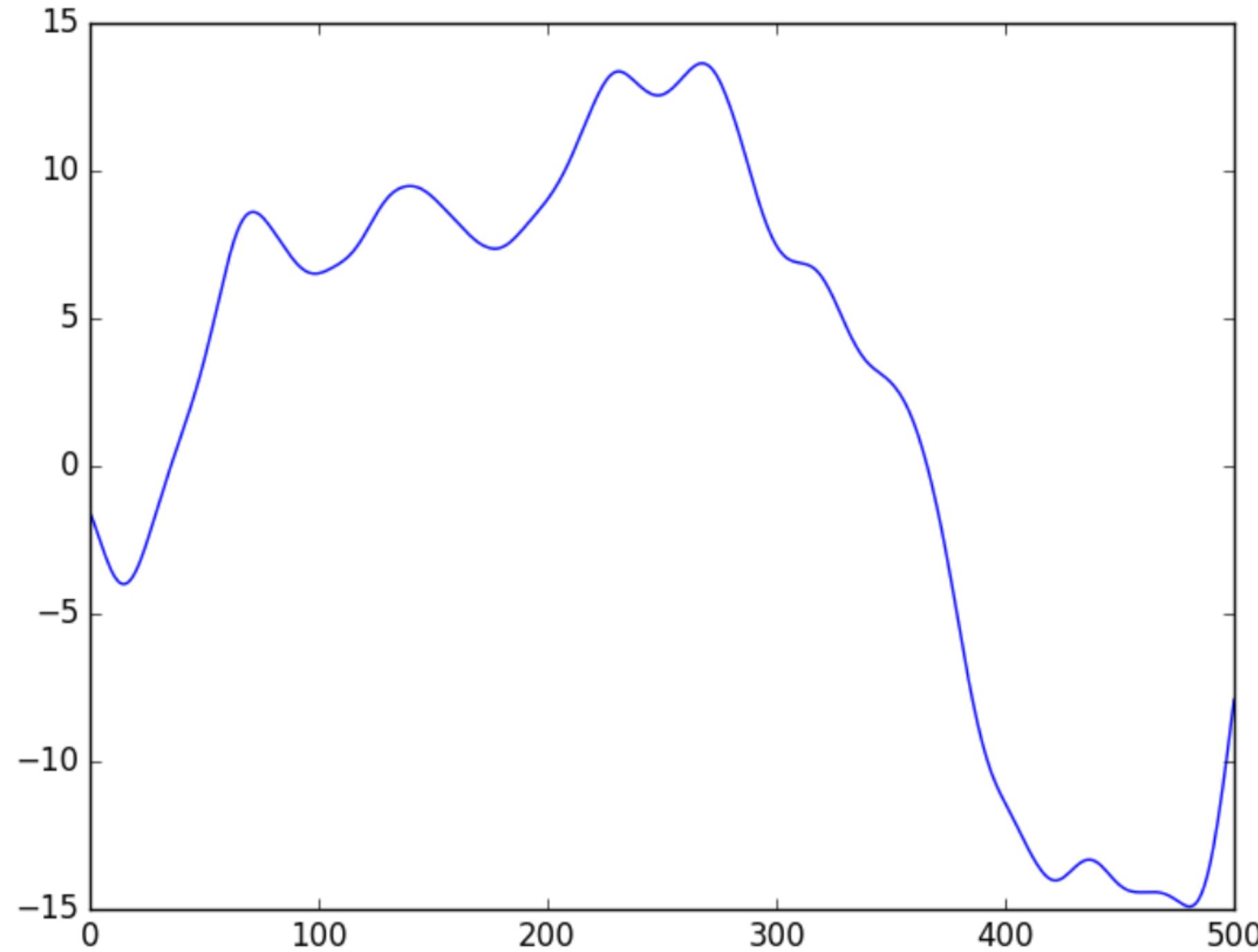


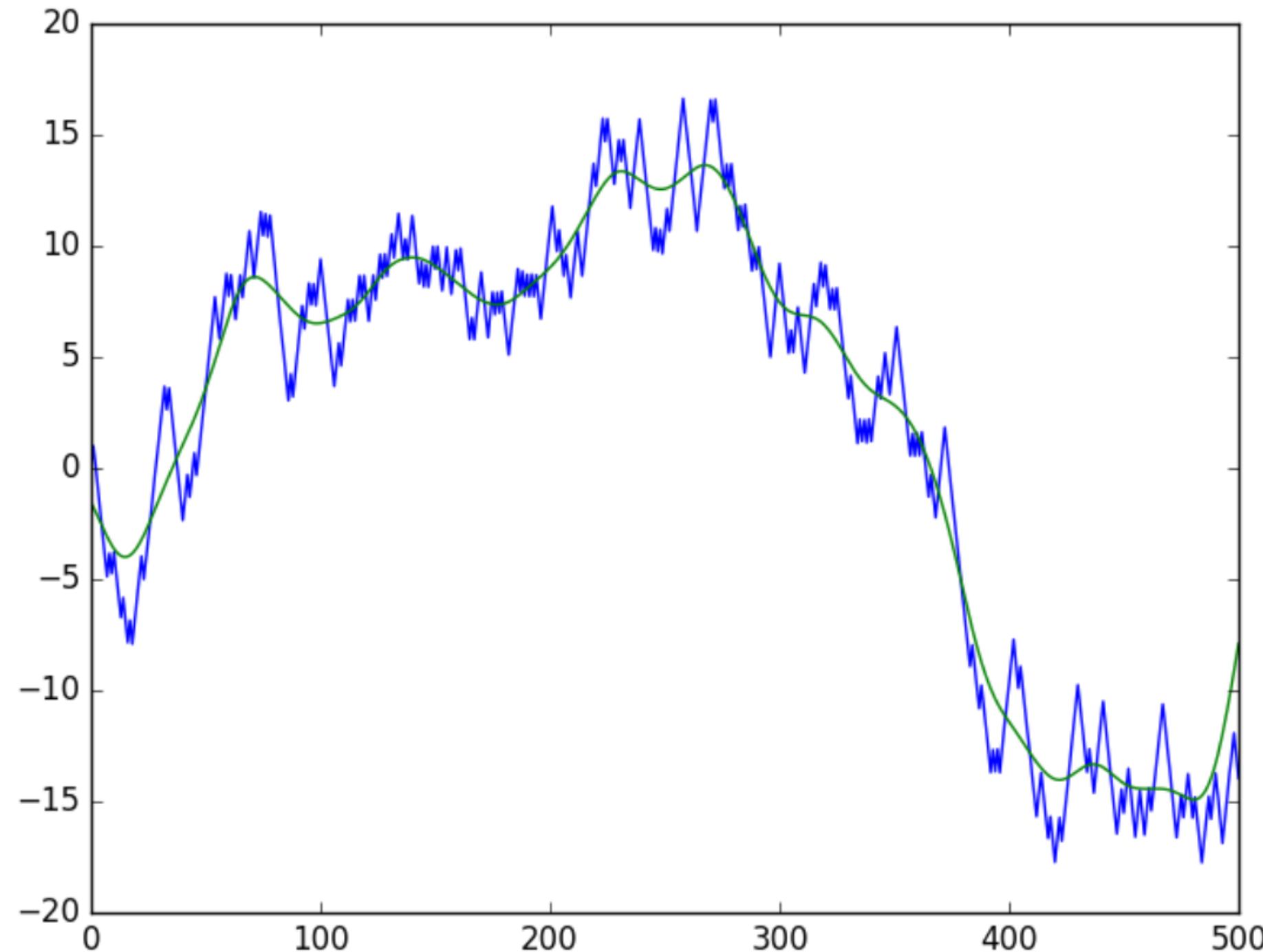


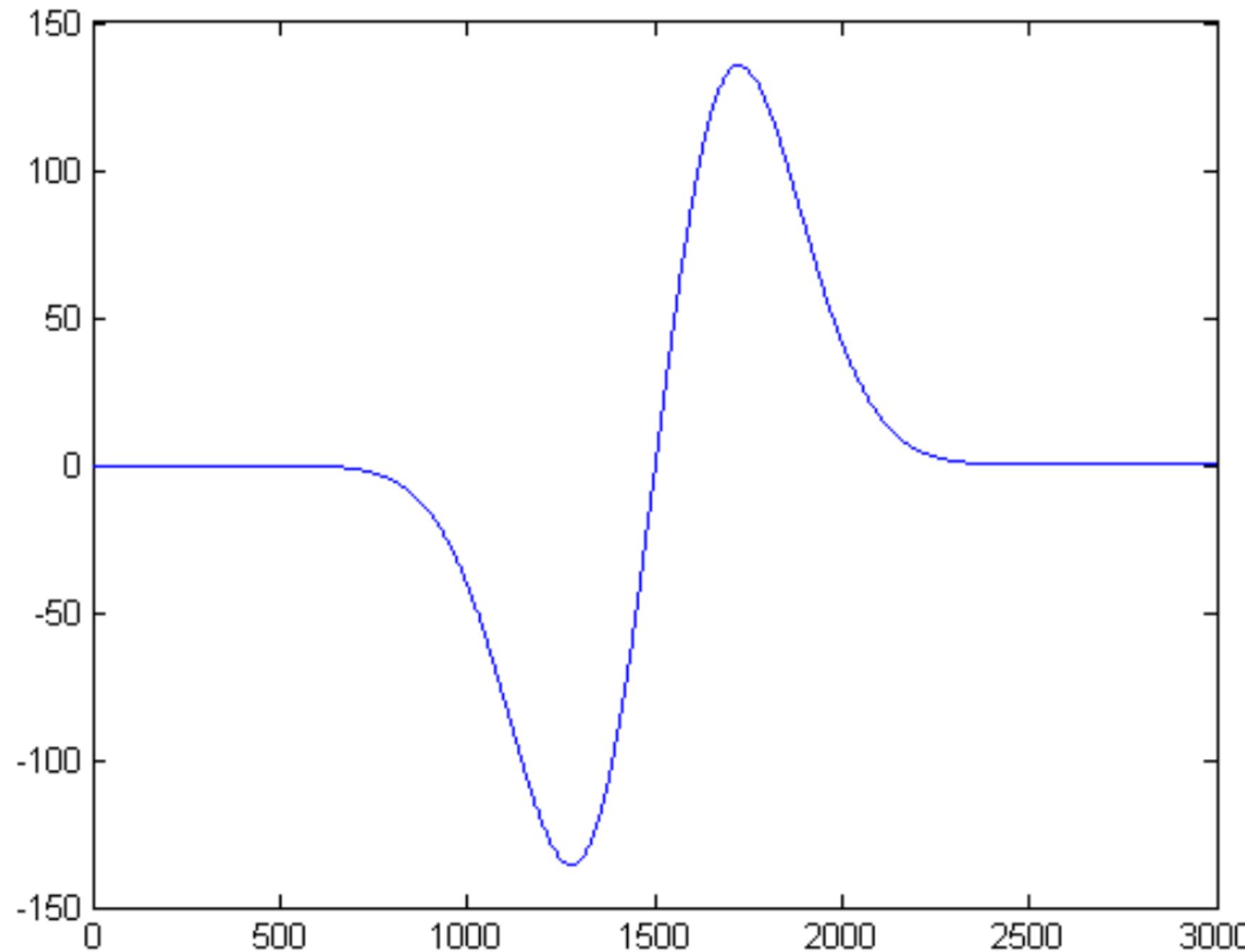
$$w(n) = 0.54 - 0.46 \cos\left(2\pi \frac{n}{N}\right), \quad 0 \leq n \leq N.$$

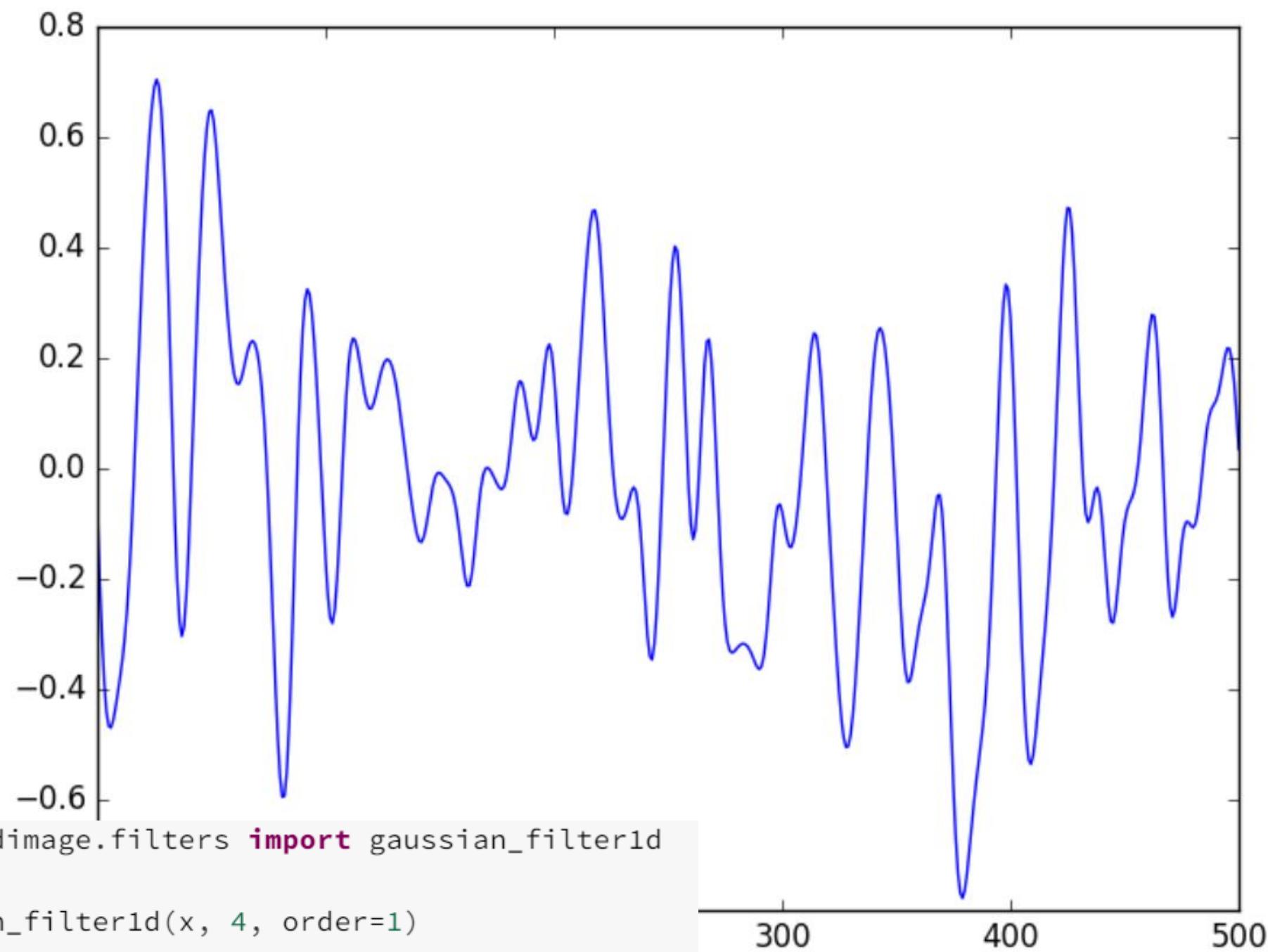
```
from scipy import signal

win = signal.hann(50)
filtered = signal.convolve(x, win, mode='same') / sum(win)
```

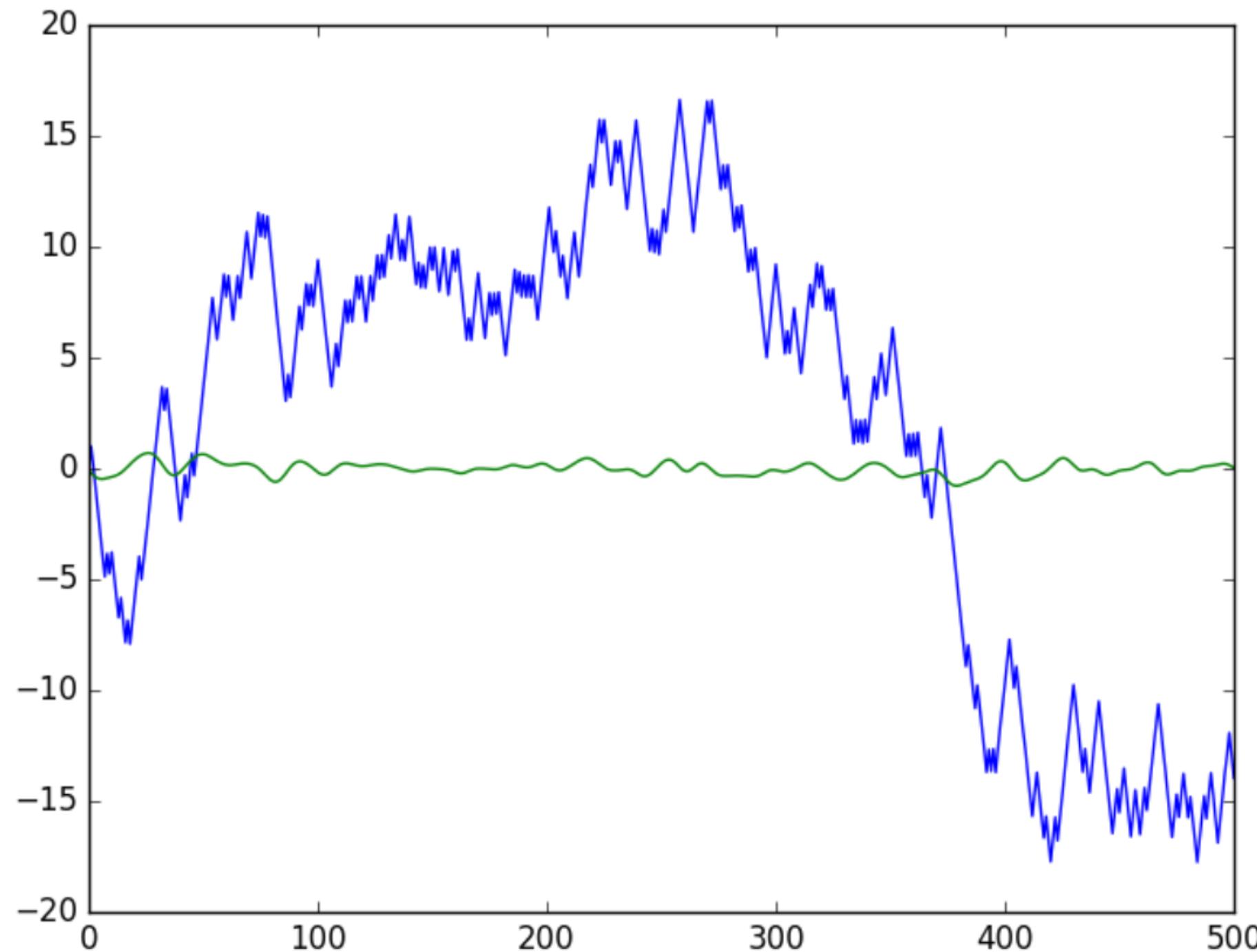








```
from scipy.ndimage.filters import gaussian_filter1d  
  
x2 = gaussian_filter1d(x, 4, order=1)
```

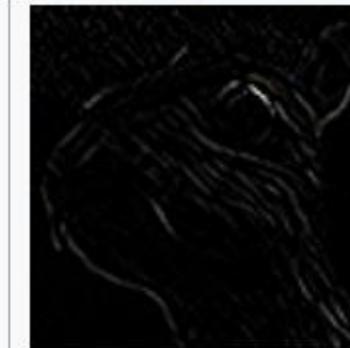


Identity

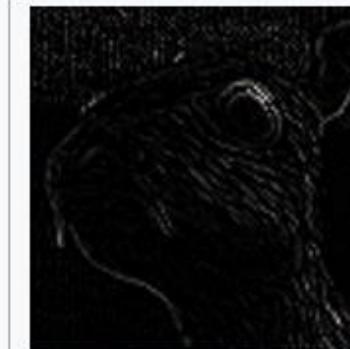
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

**Edge detection**

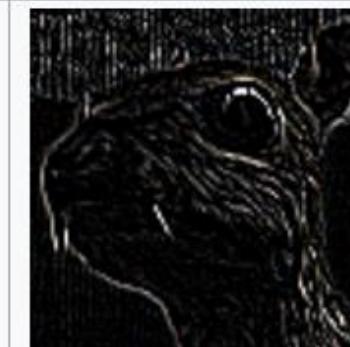
$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

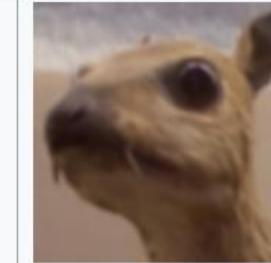
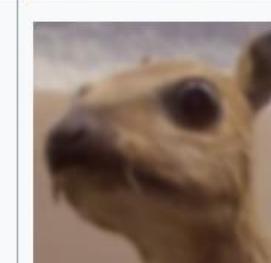
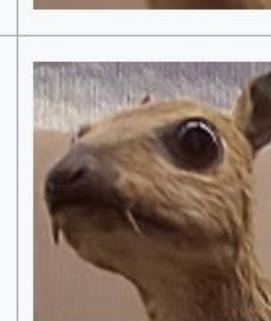


$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

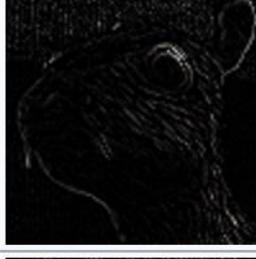
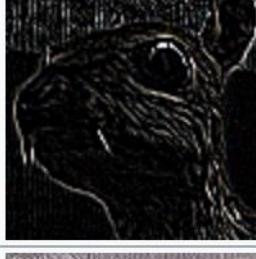


$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3×3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5×5 (approximation)	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5×5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

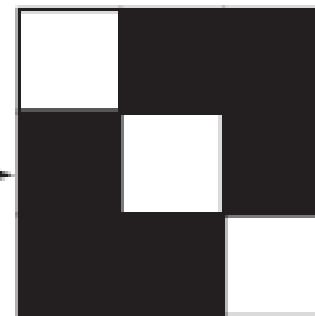
DEMO

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

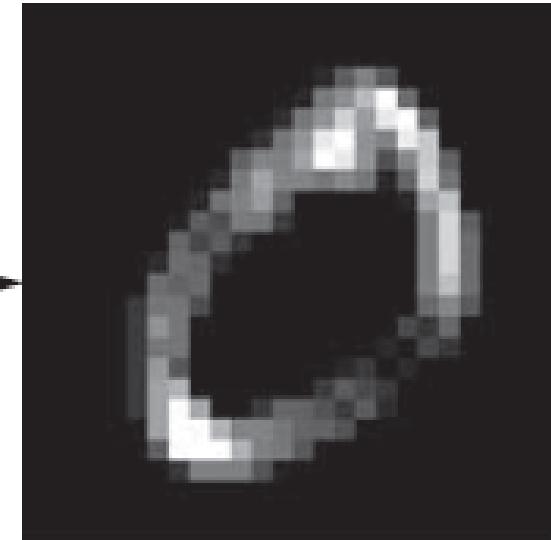
Original input



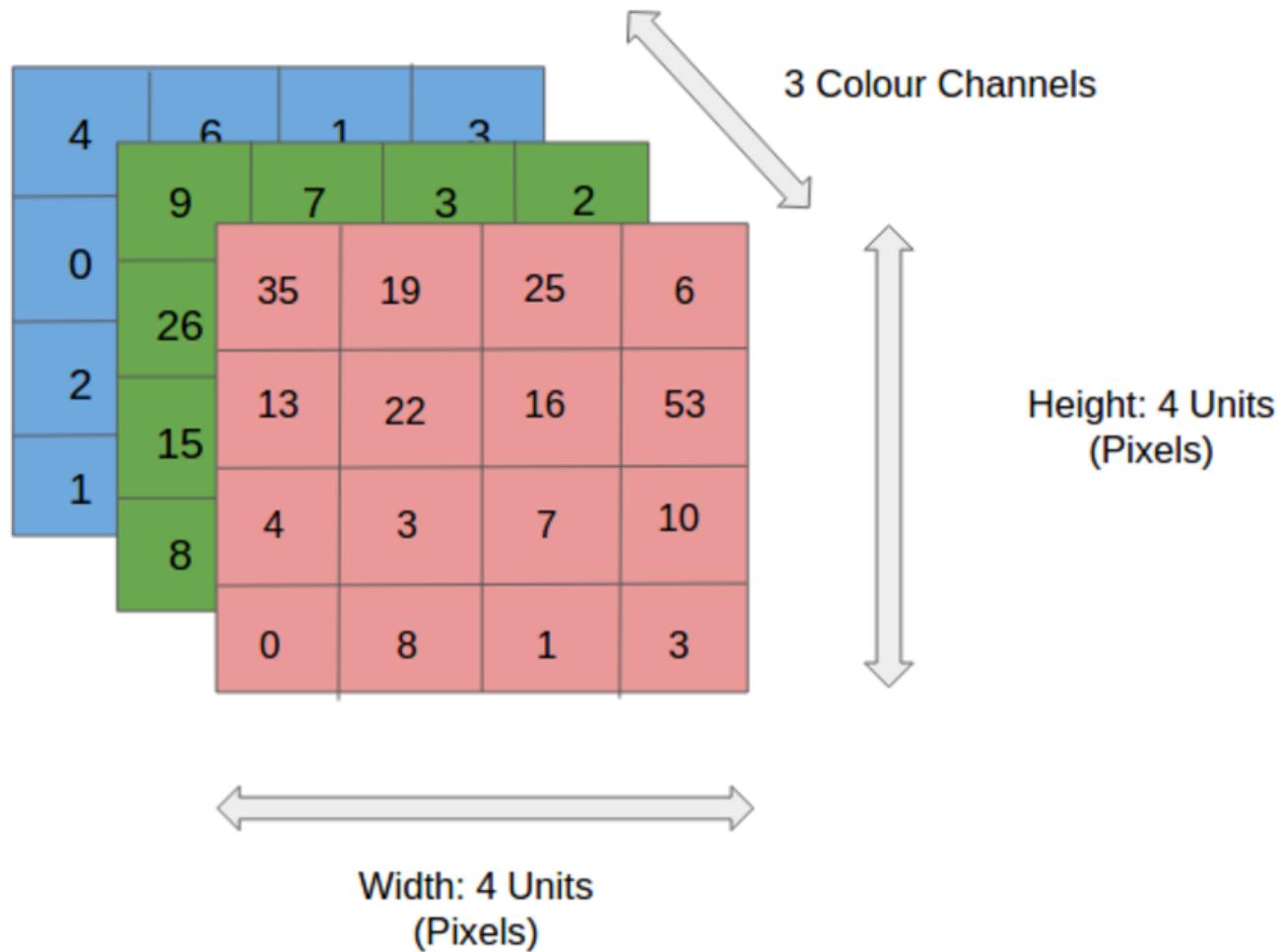
Single filter



Response map,
quantifying the presence
of the filter's pattern at
different locations



PREPARE DATASET OF IMAGES



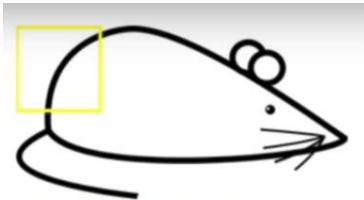
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

CONVOLUTION FILTER



Visualization of the filter on the image



Visualization of the receptive field

$$(50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600$$

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

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

Pixel representation of filter

CONVOLUTION FILTER MATCH



Visualization of the filter on the image

MULTIPLY AND SUMMATION = 0

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

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

Pixel representation of filter

CONVOLUTION FILTER NO MATCH

1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

1	1	1	0	0
0	1	1 x1	1 x0	0 x1
0	0	1 x0	1 x1	1 x0
0	0	1 x1	1 x0	0 x1
0	1	1	0	0

Image

4	3	4
2	4	3

Convolved Feature

1	1	1	0	0
0	1	1	1	0
0	0	1 x1	1 x0	1 x1
0	0	1 x0	1 x1	0 x0
0	1	1 x1	0 x0	0 x1

Image

4	3	4
2	4	3
2	3	4

Convolved Feature



CONVOLUTION

4	6	1	3
0	8	12	9
2	3	16	100
1	46	74	27



35	19	25	6
13	22	16	63
4	3	7	10
9	8	1	3



9	7	3	2
26	37	14	1
15	29	16	0
8	6	54	2



35	19	25	6
13	22	16	63
4	3	7	10
9	8	1	3



(i)

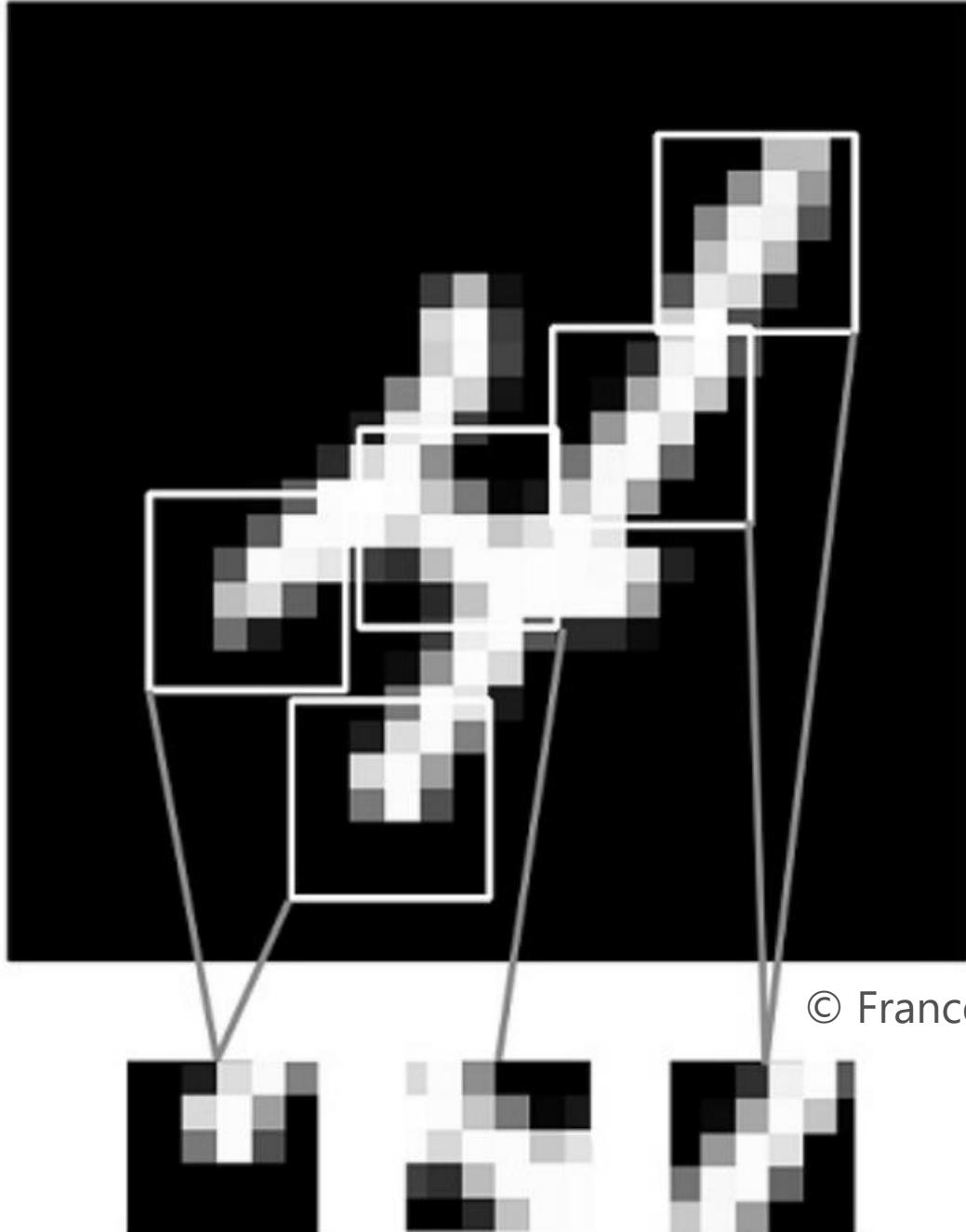
(iii)

(ii)

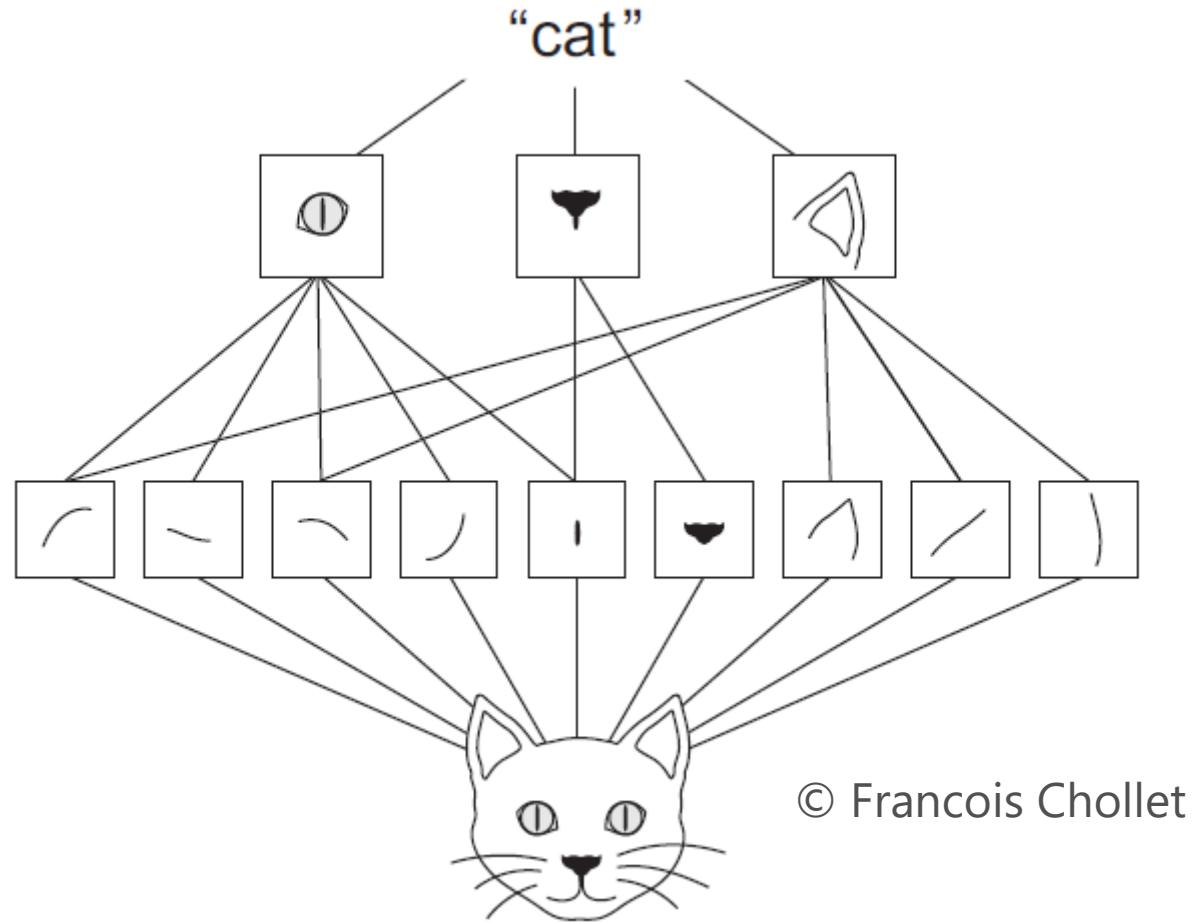
(iv)

POOLING

- Translation Invariant
- Spatial hierarchies of patterns
- Sort of scale invariant
- Not rotationally invariant!



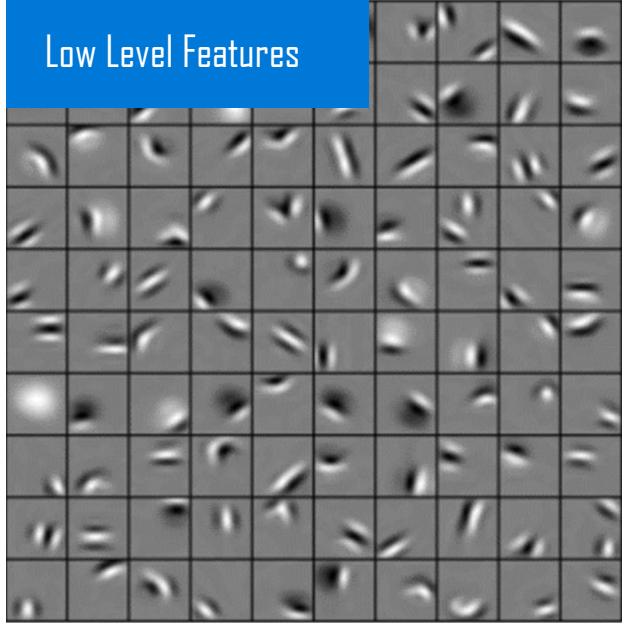
© Francois Chollet



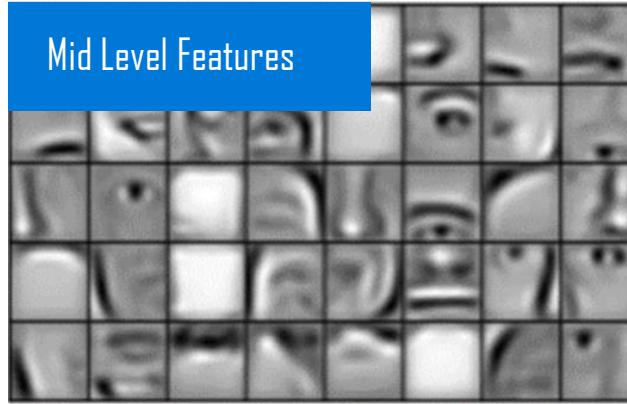
© Francois Chollet

SPATIAL HIERARCHY OF PATTERNS

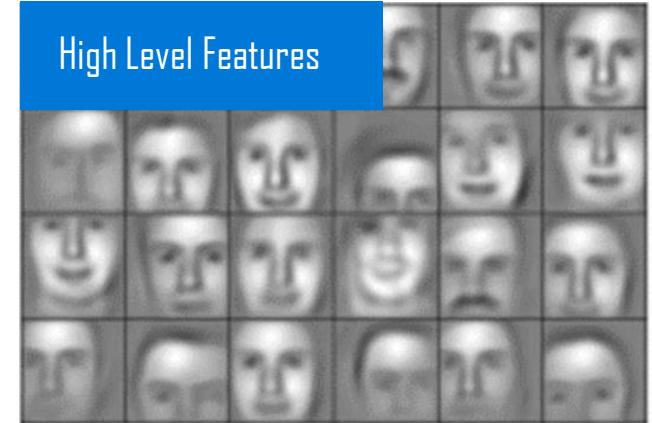
Low Level Features



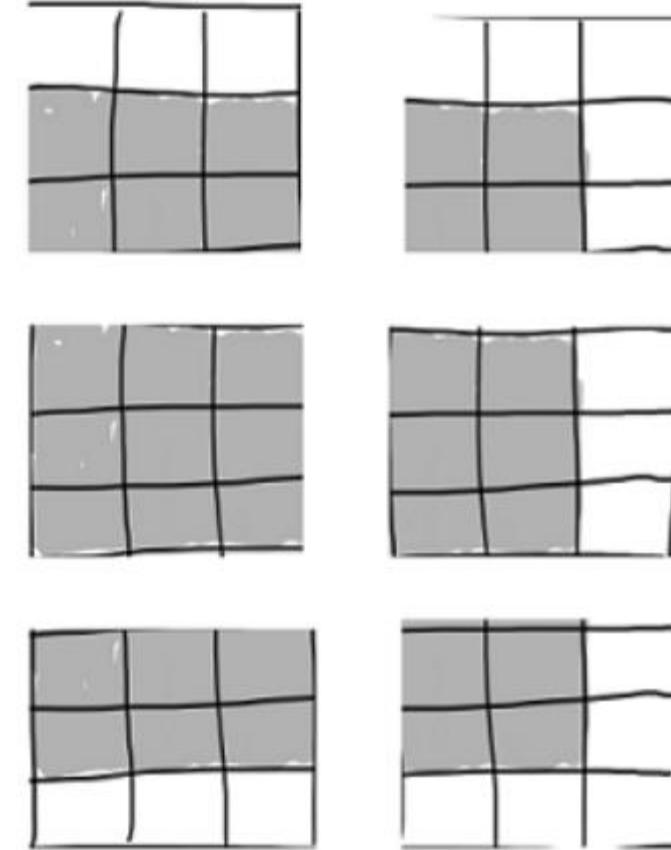
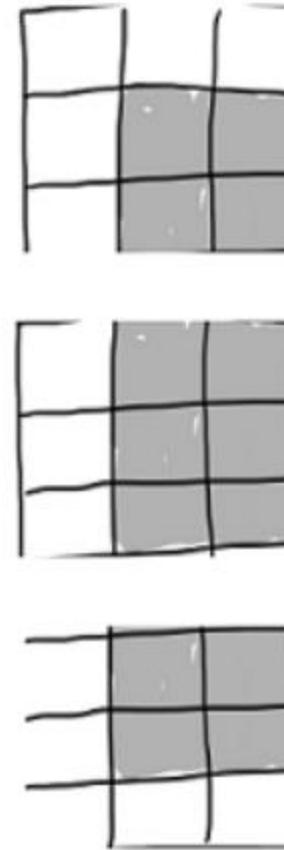
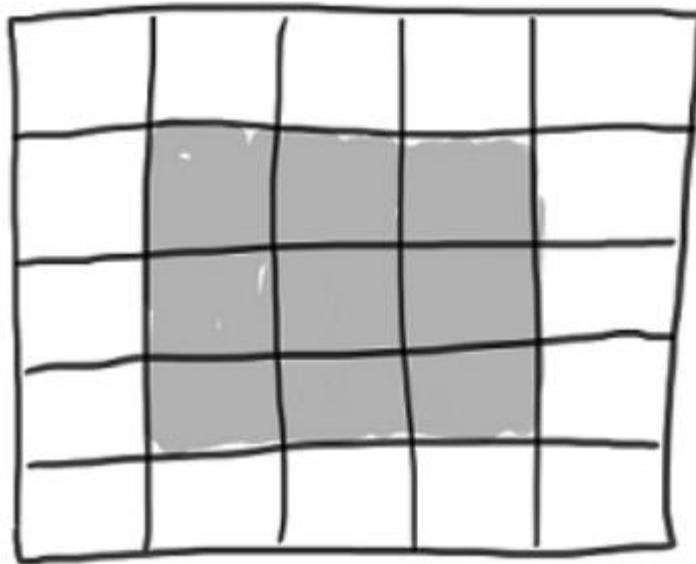
Mid Level Features



High Level Features



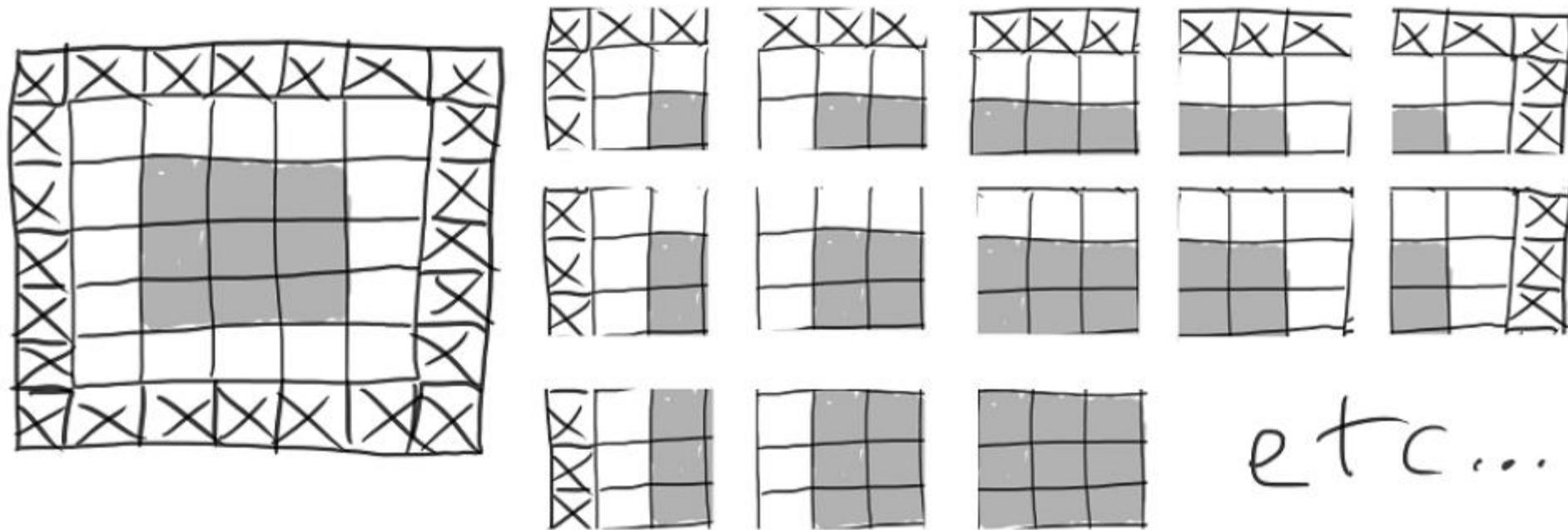
LEARNING REPRESENTATIONS



CONVOLUTION EXAMPLE

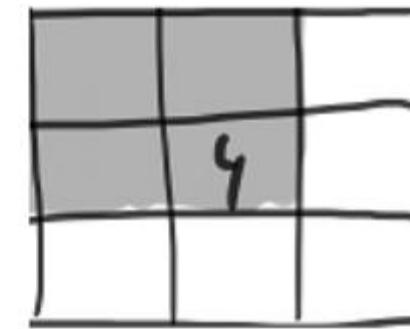
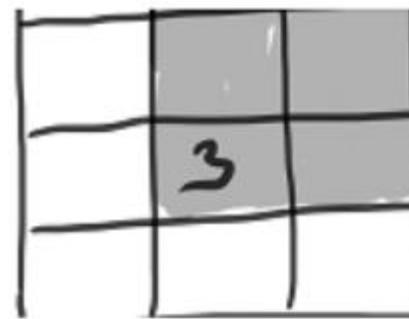
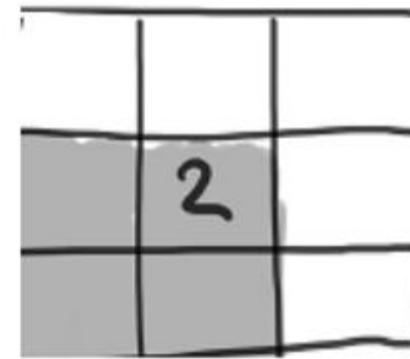
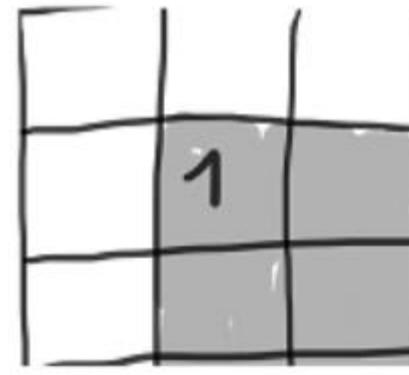
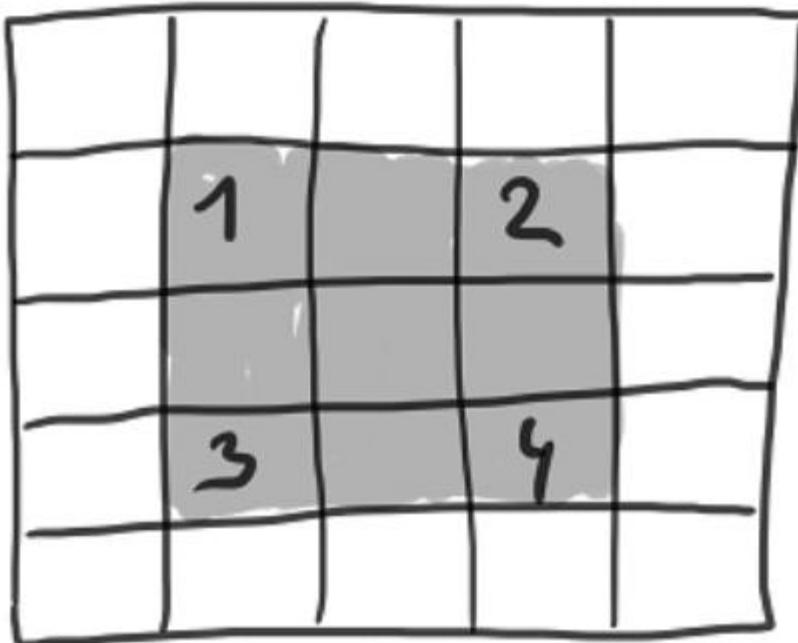
© Francois Chollet

© Francois Chollet



BORDER EFFECTS: 'VALID' VS 'SAME'

© Francois Chollet



STRIDING

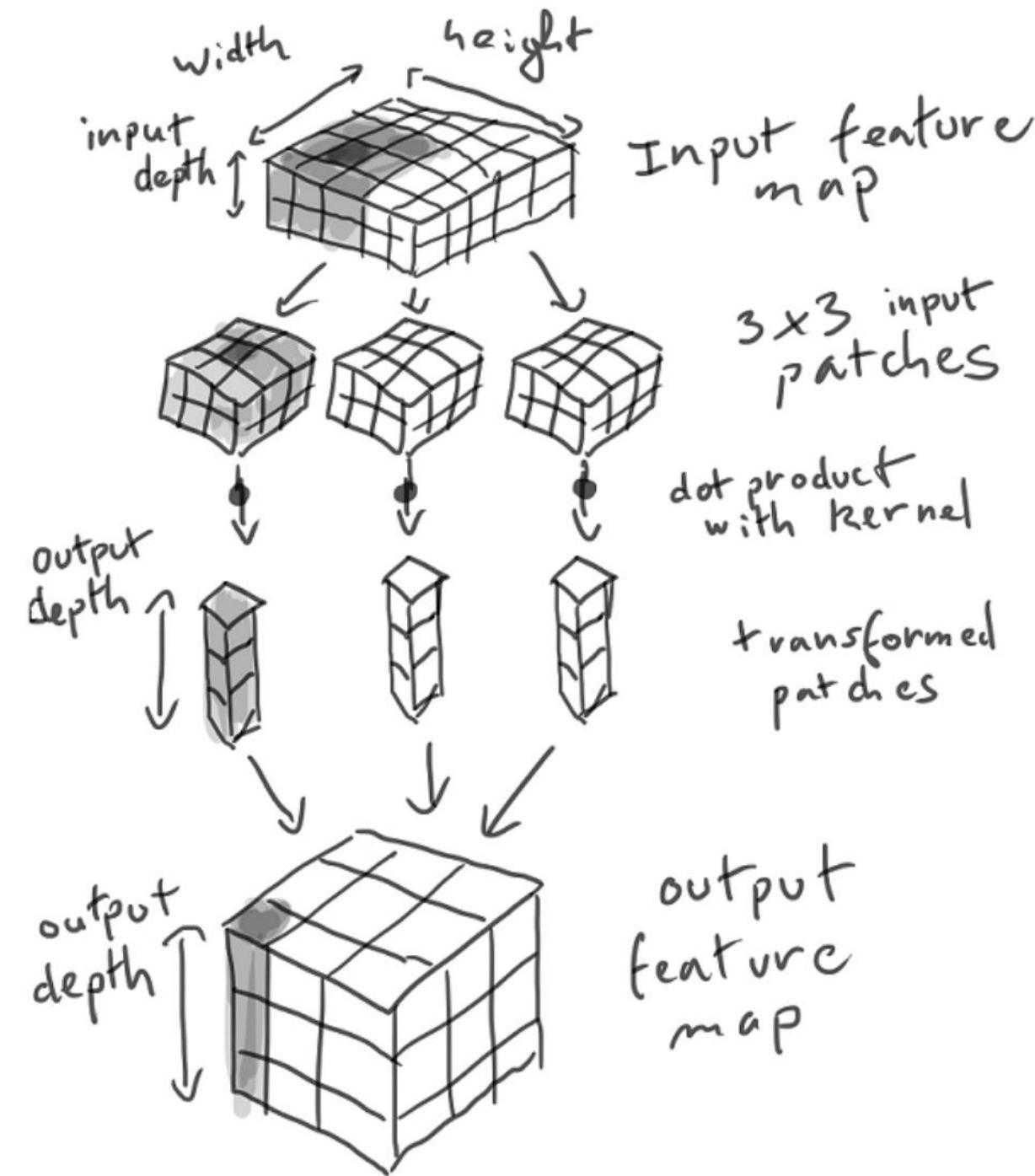
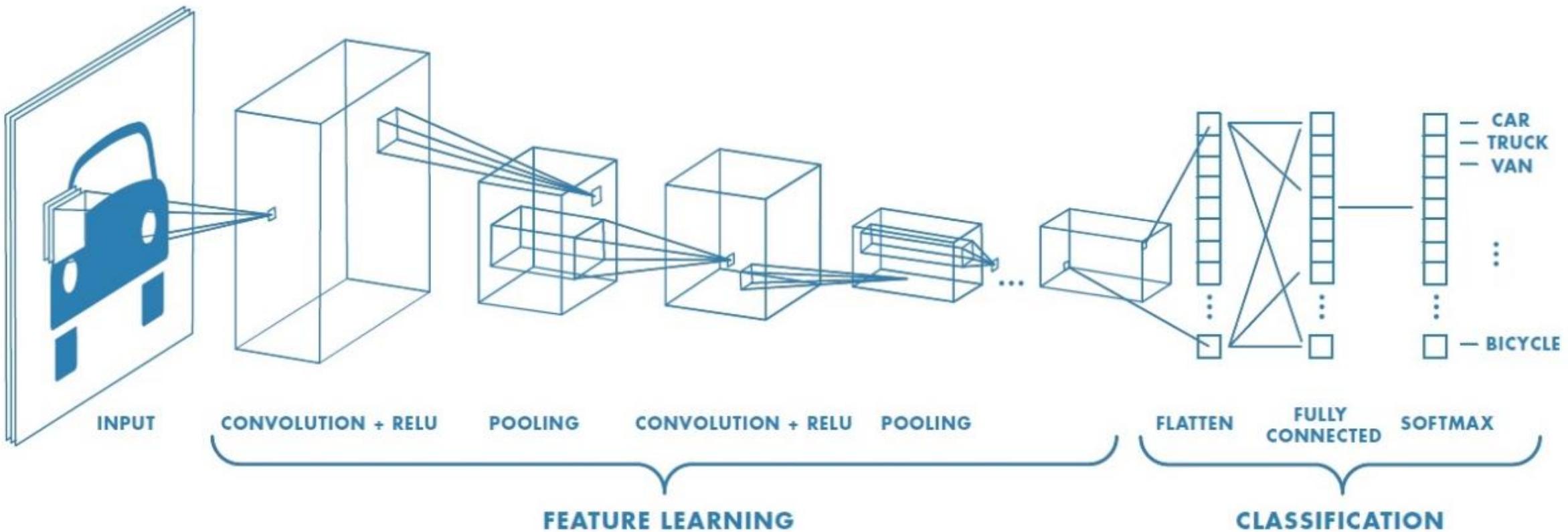
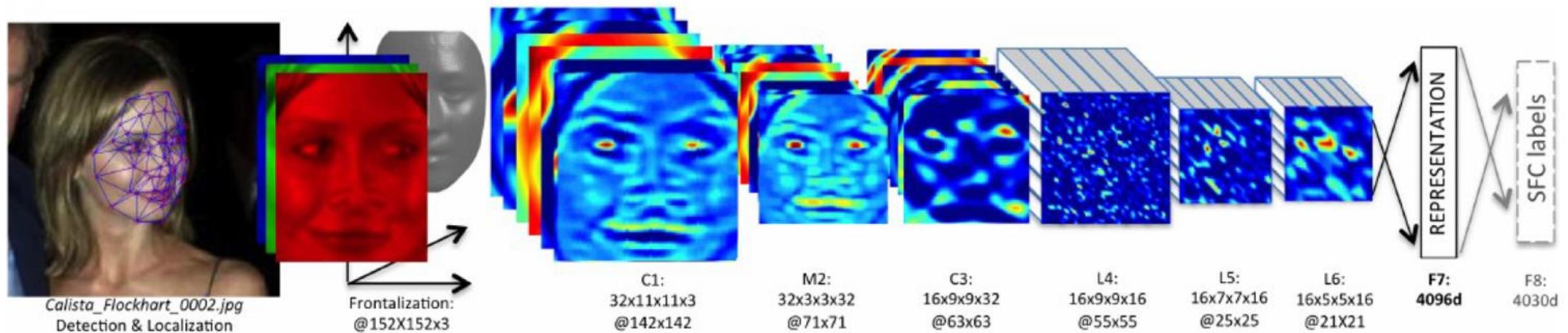


IMAGE CLASSIFICATION EXAMPLE

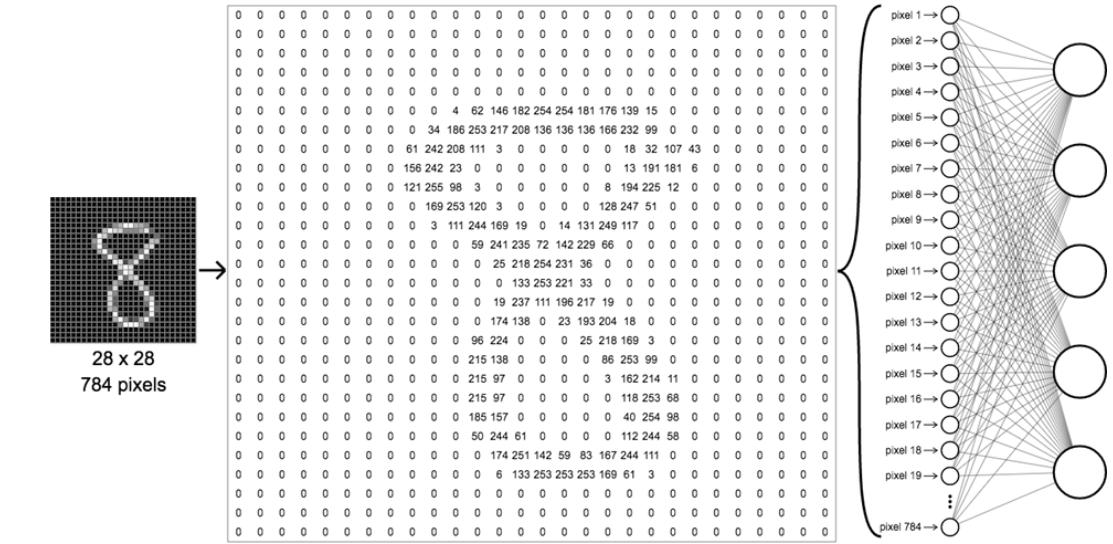
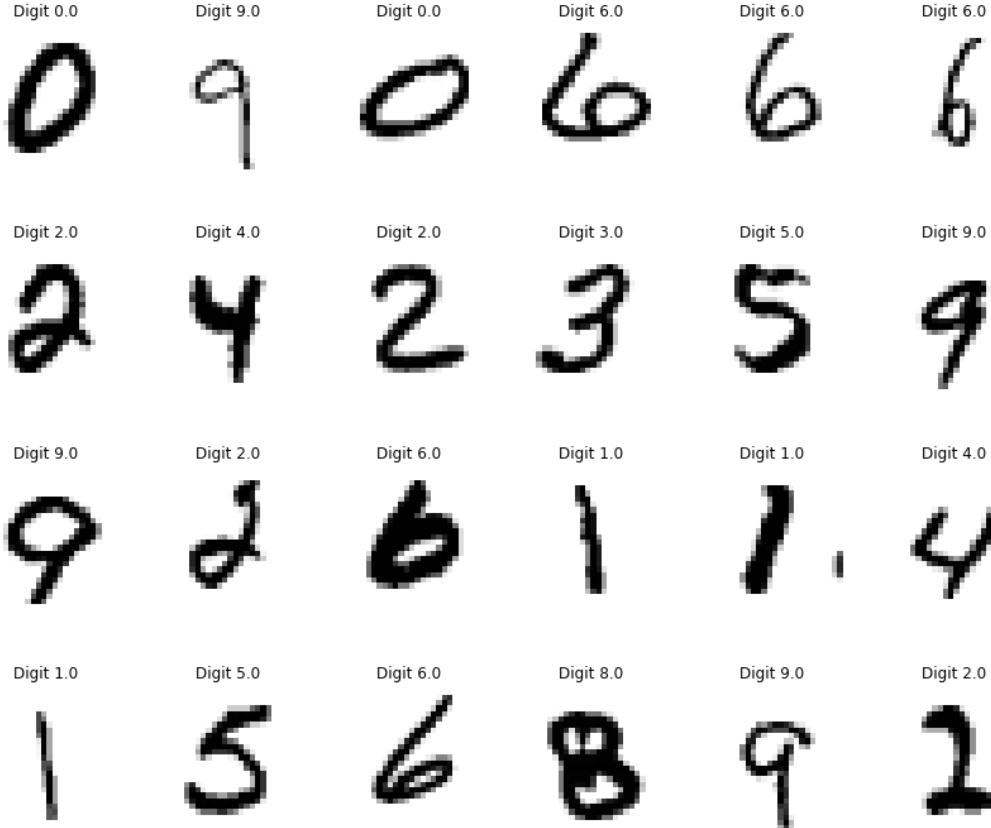


DEEP FACE



- Alignment
- CNN
- Metric Learning

MNIST DIGIT CLASSIFICATION



Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$
 Learn a classifier $f(\mathbf{x})$ such that,

$$f : \mathbf{x} \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
model.summary()
```

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
<hr/>		

Total params: 55,744

Trainable params: 55,744

Non-trainable params: 0

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

```
model.summary()
```

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 64)	36928
dense_2 (Dense)	(None, 10)	650
<hr/>		
Total params: 93,322		
Trainable params: 93,322		
Non-trainable params: 0		

```
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

"one hot encoding"

3 = 0 0 1
2 = 0 1 0

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

Epoch 1/5

60000/60000 [=====] - 8s - loss: 0.1766 - acc: 0.9440

Epoch 2/5

60000/60000 [=====] - 7s - loss: 0.0462 - acc: 0.9855

Epoch 3/5

60000/60000 [=====] - 7s - loss: 0.0322 - acc: 0.9902

Epoch 4/5

60000/60000 [=====] - 7s - loss: 0.0241 - acc: 0.9926

Let's evaluate the model on the test data:

```
In [8]: test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
9536/10000 [=====>..] - ETA: 0s
```

```
In [9]: test_acc
```

```
Out[9]: 0.9912999999999996
```

DOGS AND CATS DATASET



- 2000 Dogs
- 2000 Cats
- We use 2K for training

VISION ON SMALL DATA SETS

- “Deep learning only works on small datasets”
 - Only valid if you want to learn abstract representations as part of the training process
- Data Augmentation helps
- Transfer learning helps even more!

DOGS CATS MODEL

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                      input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.summary()
```

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_1 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_3 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_4 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 512)	3211776
dense_2 (Dense)	(None, 1)	513
<hr/>		
Total params: 3,453,121		
Trainable params: 3,453,121		
Non-trainable params: 0		

```
: from keras import optimizers  
  
model.compile(loss='binary_crossentropy',  
              optimizer=optimizers.RMSprop(lr=1e-4),  
              metrics=['acc'])
```

```
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    # This is the target directory
    train_dir,
    # All images will be resized to 150x150
    target_size=(150, 150),
    batch_size=20,
    # Since we use binary_crossentropy loss, we need binary labels
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary')
```

Found 2000 images belonging to 2 classes.

Found 1000 images belonging to 2 classes.

```
history = model.fit_generator(  
    train_generator,  
    steps_per_epoch=100,  
    epochs=30,  
    validation_data=validation_generator,  
    validation_steps=50)
```

Epoch 1/30

100/100 [=====] - 9s - loss: 0.6898 - acc: 0.5285 - val_loss: 0.6724 - val_acc: 0.5950

Epoch 2/30

100/100 [=====] - 8s - loss: 0.6543 - acc: 0.6340 - val_loss: 0.6565 - val_acc: 0.5950

Epoch 3/30

100/100 [=====] - 8s - loss: 0.6143 - acc: 0.6690 - val_loss: 0.6116 - val_acc: 0.6650

Epoch 4/30

100/100 [=====] - 8s - loss: 0.5626 - acc: 0.7125 - val_loss: 0.5774 - val_acc: 0.6970

It is good practice to always save your models after training:

```
: model.save('cats_and_dogs_small_1.h5')
```

```
: import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

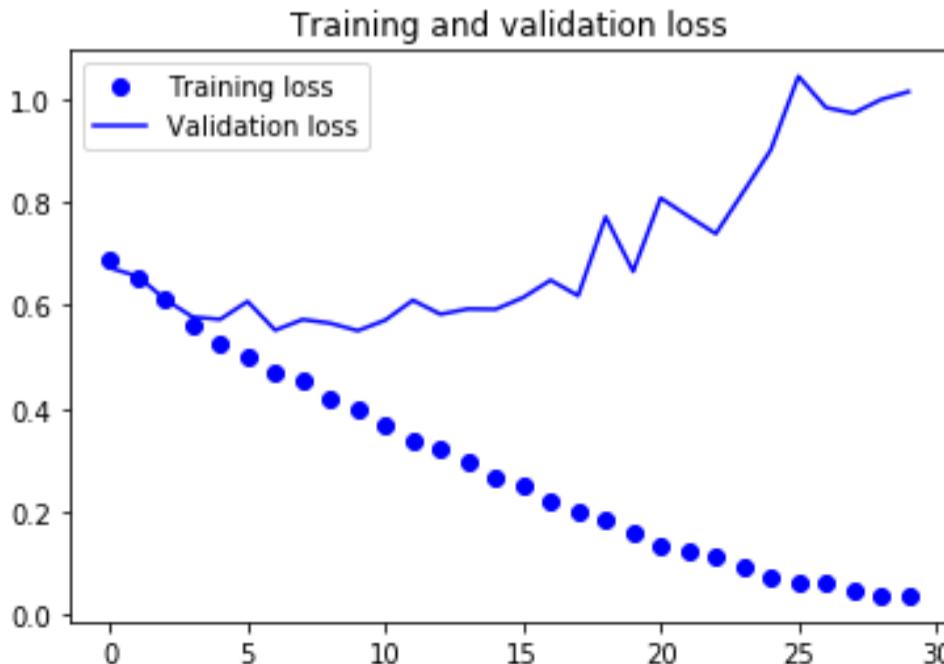
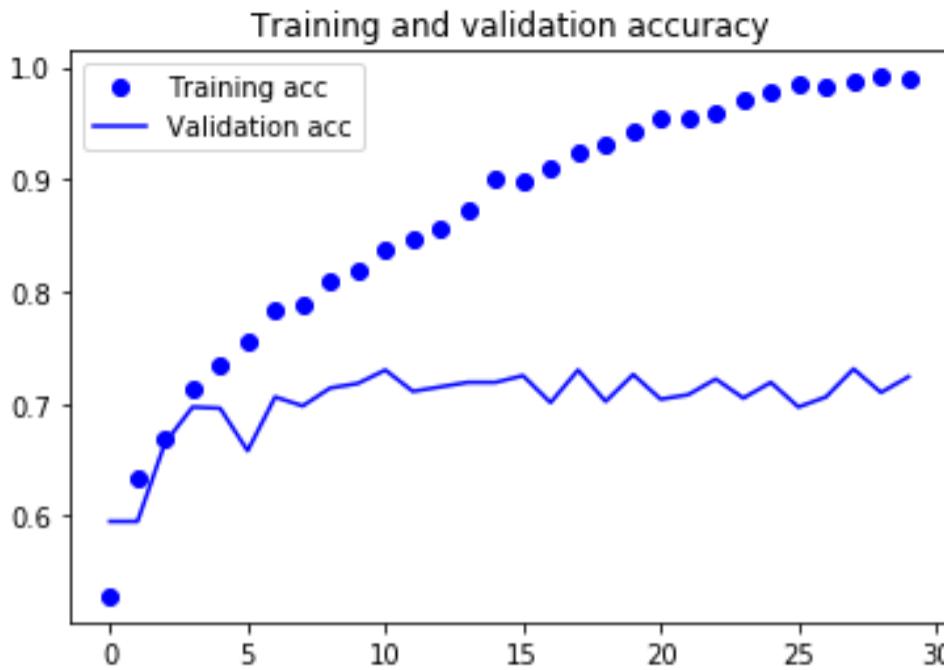
epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

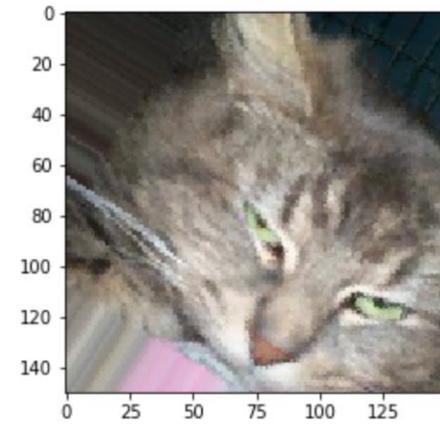
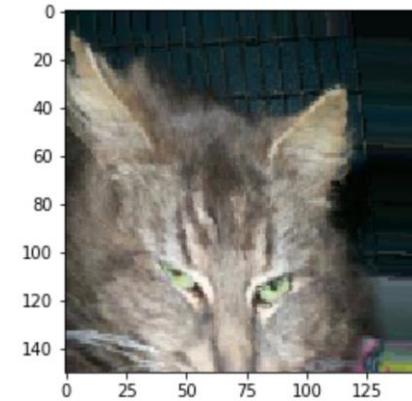
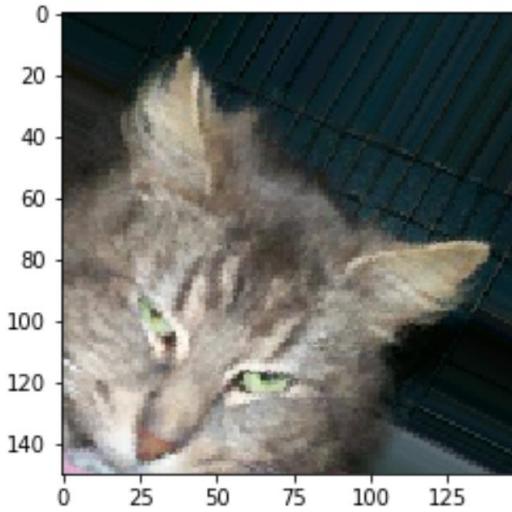
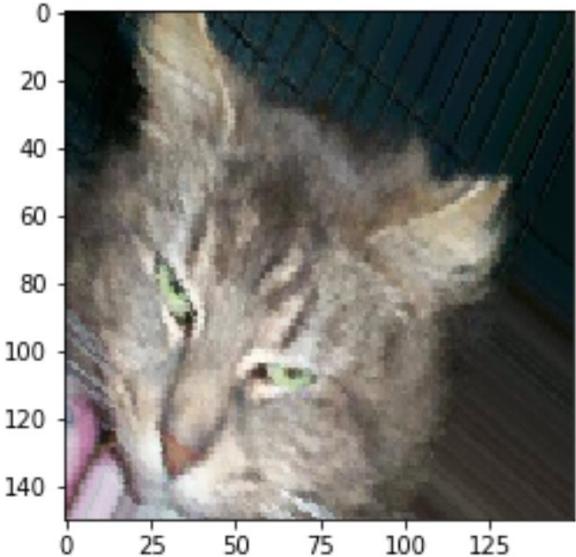


- Val acc of 70%
- Overfitting
- Dropout
- L2 regularisation
- Now... Augmentation

Overfitting is caused by having too few samples to learn from, rendering us unable to train a model able to generalize to new data

```
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')
```

AUGMENTED IMAGES



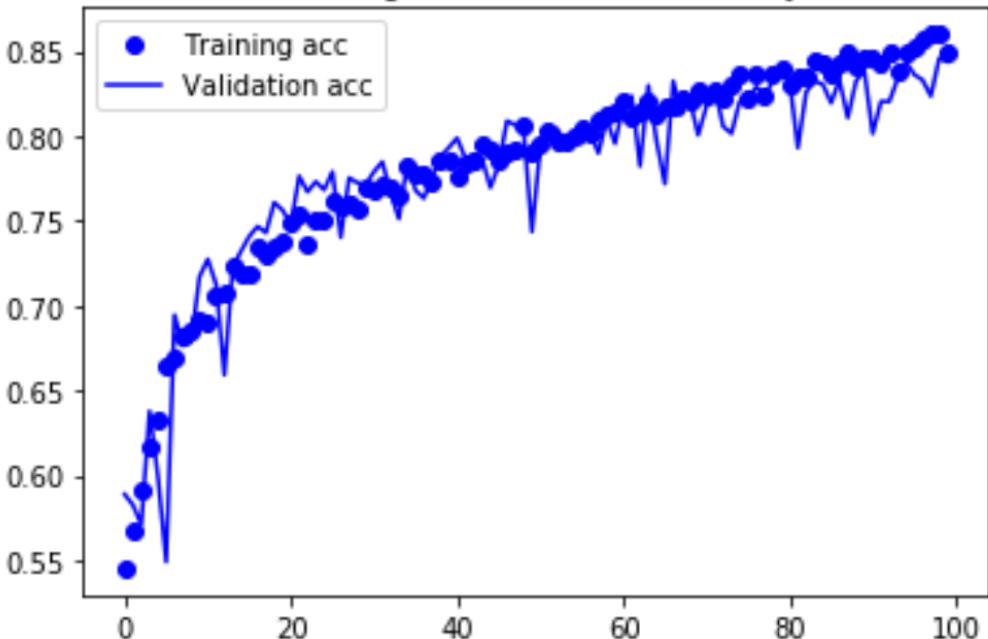
TRAIN AGAIN WITH DROPOUT AND AUGMENTATION

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                      input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

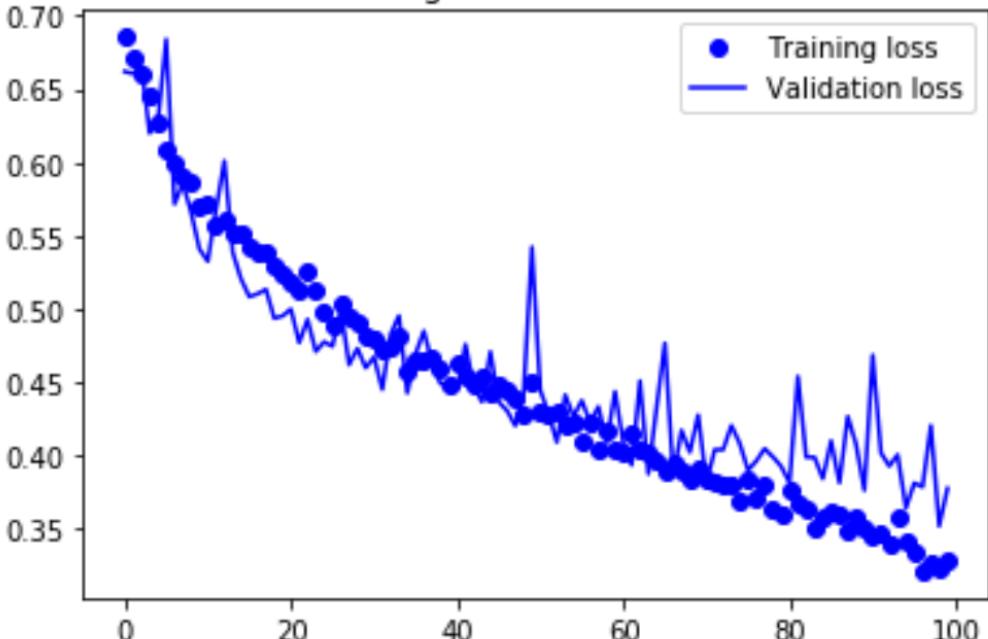
NO OVERFITTING!

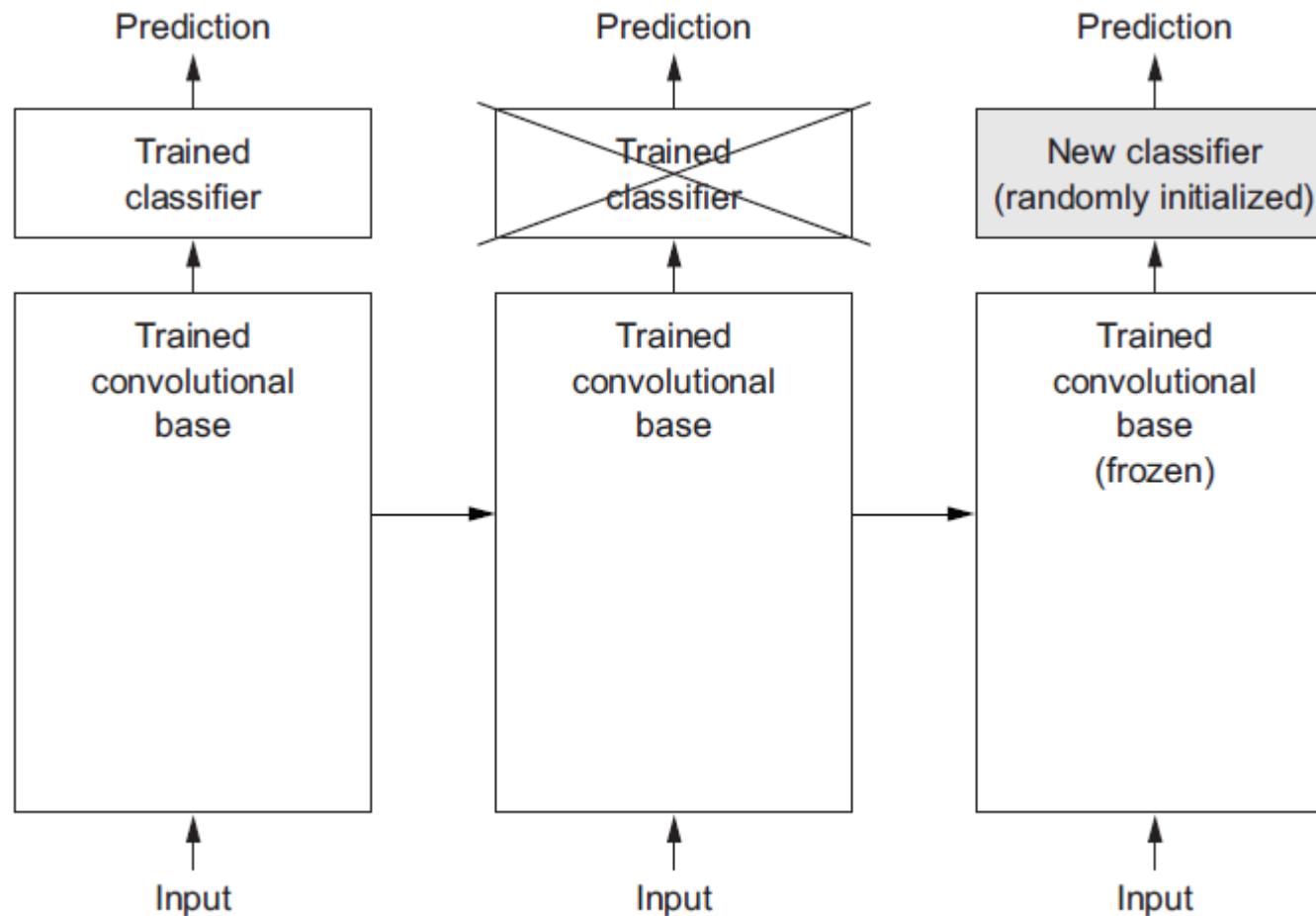
Training and validation accuracy

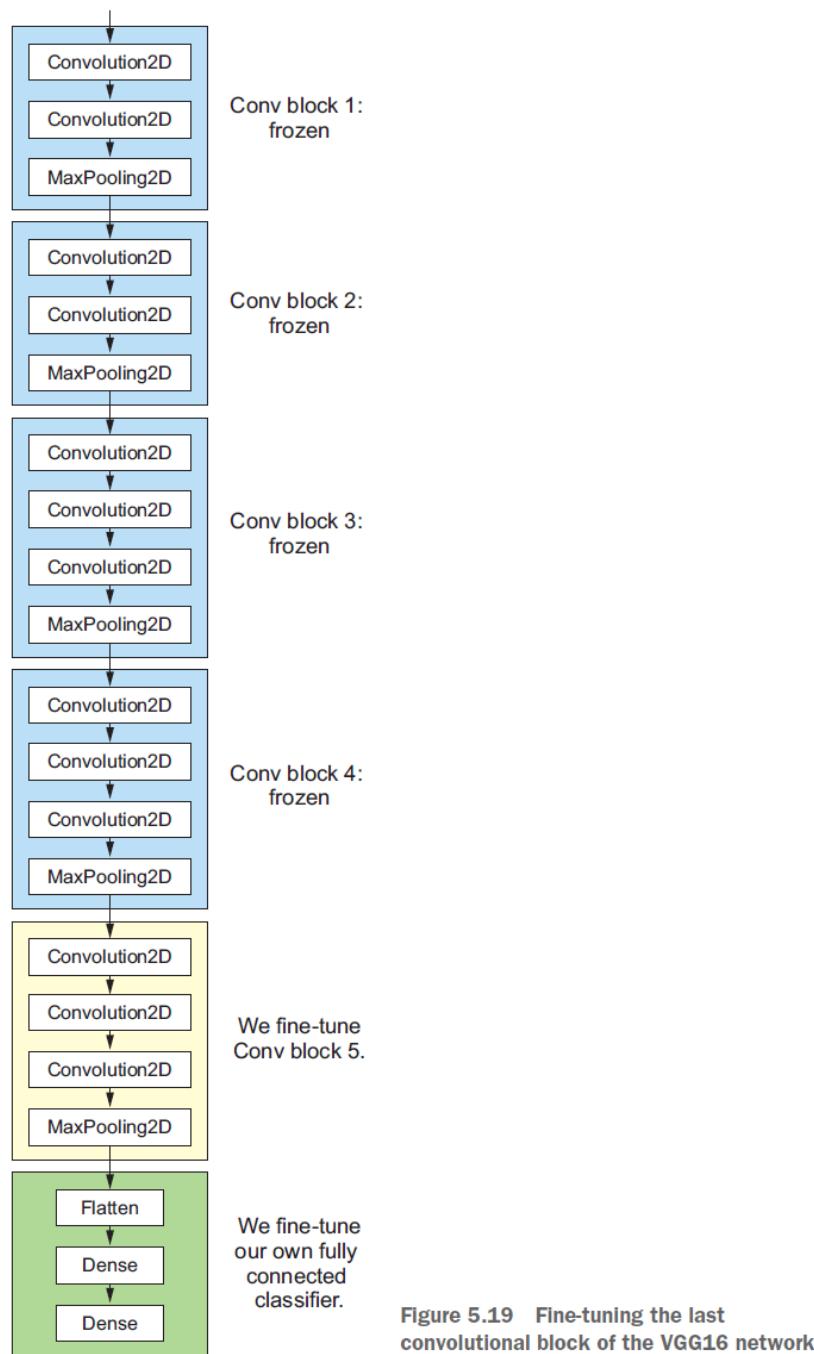


- VAL acc of 82!
- 15% relative

Training and validation loss







- Xception
- InceptionV3
- ResNet50
- VGG16
- VGG19
- MobileNet

Figure 5.19 Fine-tuning the last convolutional block of the VGG16 network

```
from keras.applications import VGG16  
  
conv_base = VGG16(weights='imagenet',  
                  include_top=False,  
                  input_shape=(150, 150, 3))
```

block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

TWO WAYS TO PROCEED

- Precompute embeddings
- Create end-end Architecture (needed with augmentation)

```
from keras import models
from keras import layers
from keras import optimizers

model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])
```

```
history = model.fit(train_features, train_labels,
                     epochs=30,
                     batch_size=20,
                     validation_data=(validation_features, validation_labels))
```

Train on 2000 samples, validate on 1000 samples

Epoch 1/30

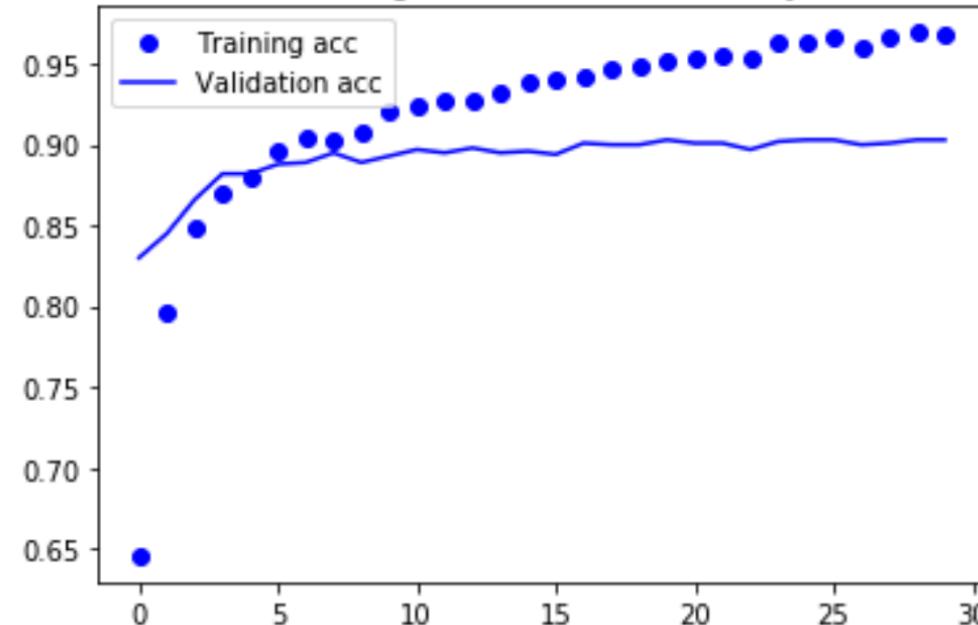
2000/2000 [=====] - 1s - loss: 0.6253 - acc: 0.6455 - val_loss: 0.4!
0.8300

Epoch 2/30

2000/2000 [=====] - 0s - loss: 0.4490 - acc: 0.7965 - val_loss: 0.3!
0.8450

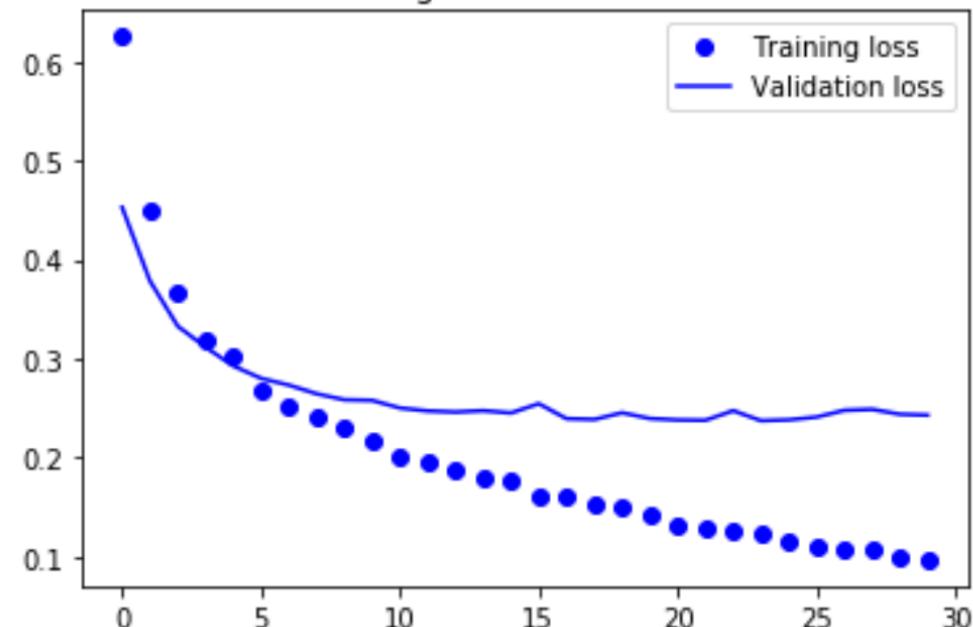
Epoch 3/30

Training and validation accuracy



- Val now ~90%!
- Overfitting a lot

Training and validation loss



ADD THE CONV BASE TO MODEL

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.summary()
```

Layer (type)	Output Shape	Param #
<hr/>		
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten_1 (Flatten)	(None, 8192)	0
dense_3 (Dense)	(None, 256)	2097408
dense_4 (Dense)	(None, 1)	257
<hr/>		

Total params: 16,812,353

Trainable params: 16,812,353

Non-trainable params: 0

```
In [10]: print('This is the number of trainable weights '
      'before freezing the conv base:', len(model.trainable_weights))
```

This is the number of trainable weights before freezing the conv base: 30

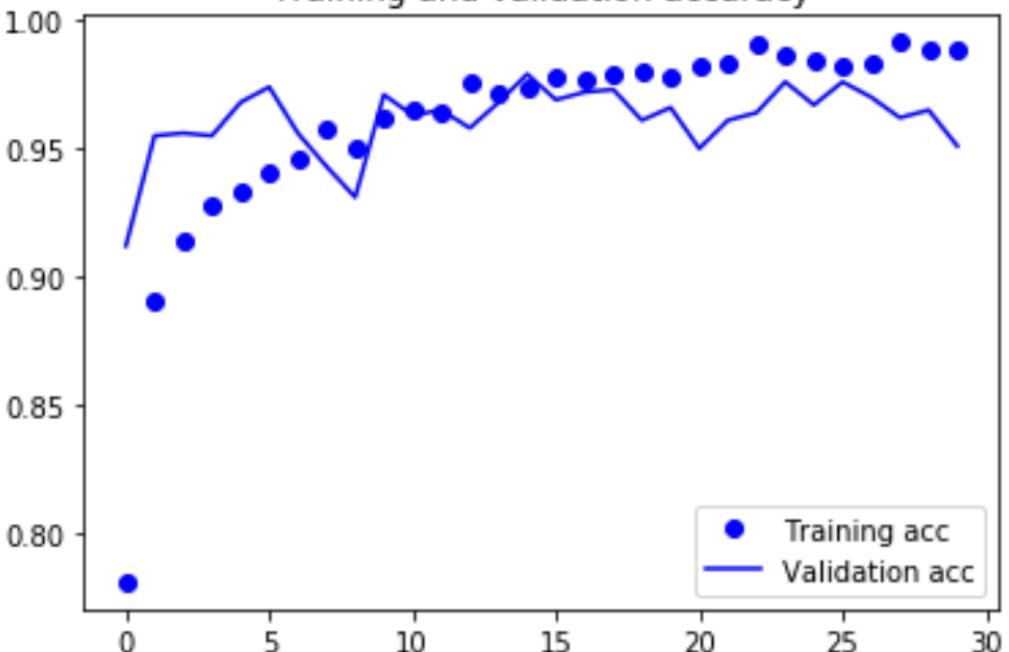
```
In [11]: conv_base.trainable = False
```

```
In [12]: print('This is the number of trainable weights '
      'after freezing the conv base:', len(model.trainable_weights))
```

This is the number of trainable weights after freezing the conv base: 4

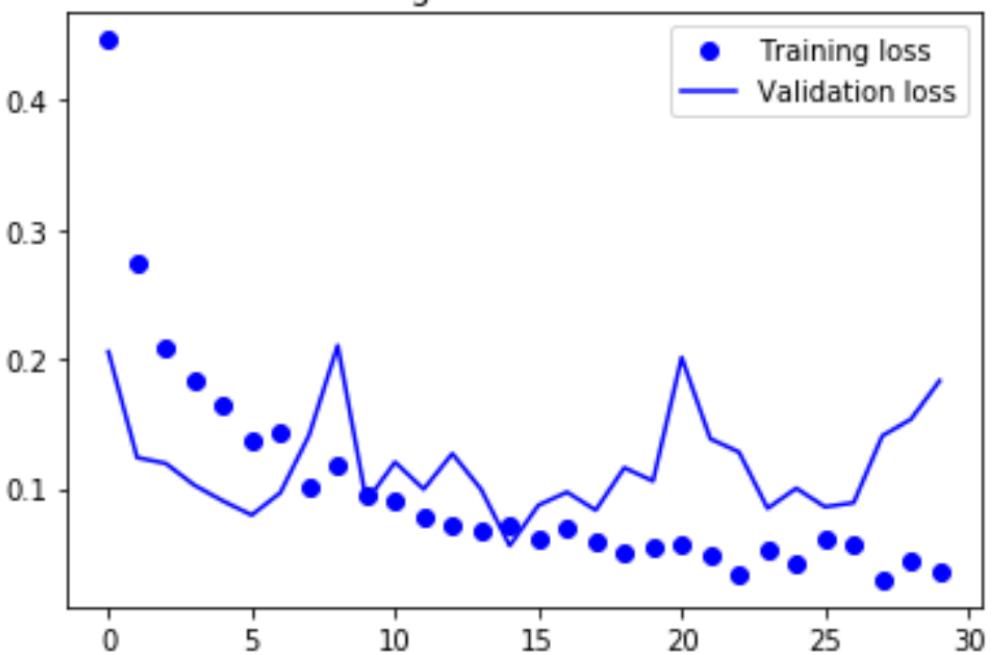
Since the dense layers are randomly initialized, very large weight updates would destroy the representations learned in the convolutional base!

Training and validation accuracy



- Val now ~96%!
- Not overfitting

Training and validation loss



RECAP ON VALIDATION PERFORMANCE

- CATS AND DOGS
 - Basic model 72%
 - Dropout and augmentation 82%
 - Transfer learning 90% (overfitting)
 - Transfer with augmentation: 96% (not overfitting)

TRAINING INTO THE CONV BASE

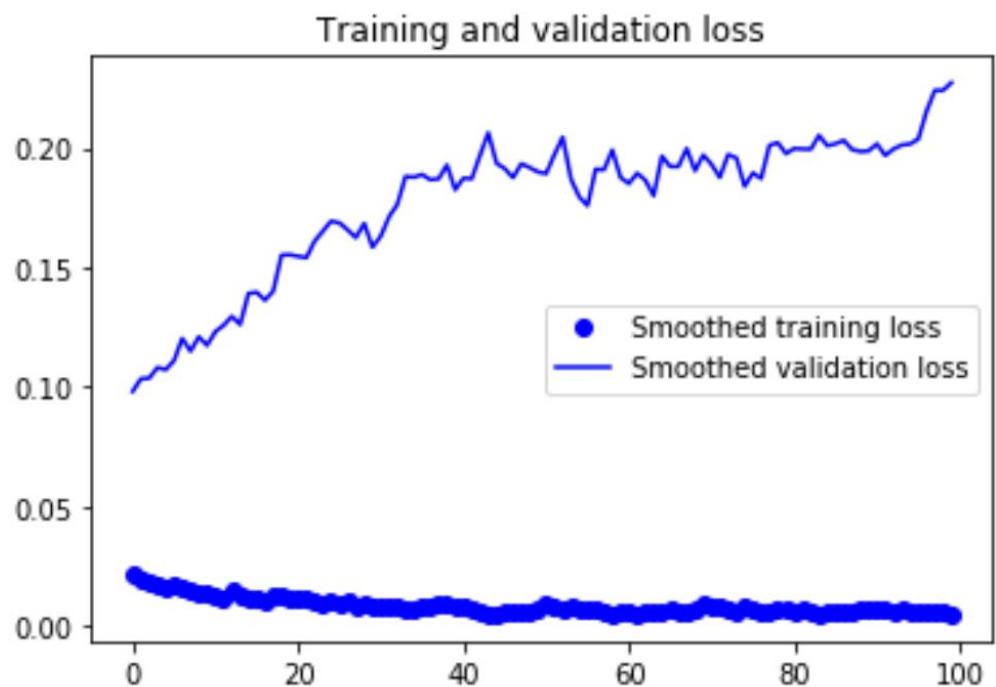
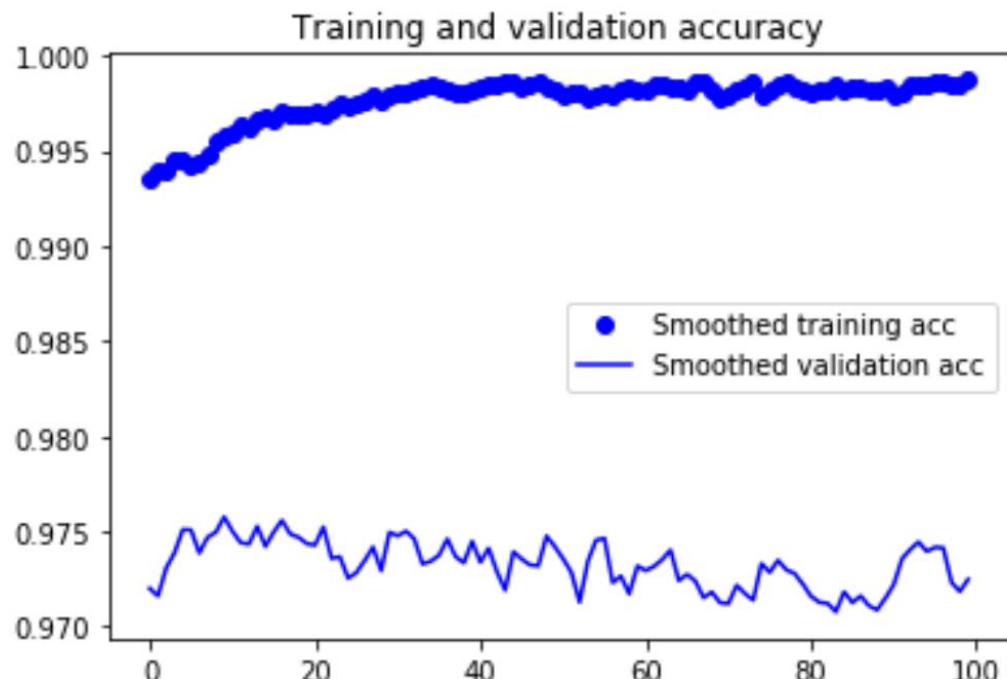
- Add your custom network on top of an already trained base network.
- Freeze the base network.
- Train the part you added.
- Unfreeze some layers in the base network.
- Jointly train both these layers and the part you added.

TRAINING INTO THE CONV BASE (2)

- Why not fine tune all layers?
 - More parameters == more overfitting
- Use a low learning rate == less chance of disaster
- How can loss get worse and accuracy go up?
 - Average of loss vs distribution of loss values might be different i.e. model might still be improving

```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```



- 1% improvement
- 97% would have been a very good score on Kaggle in the day
- Done with only 2000 examples from the 4k!

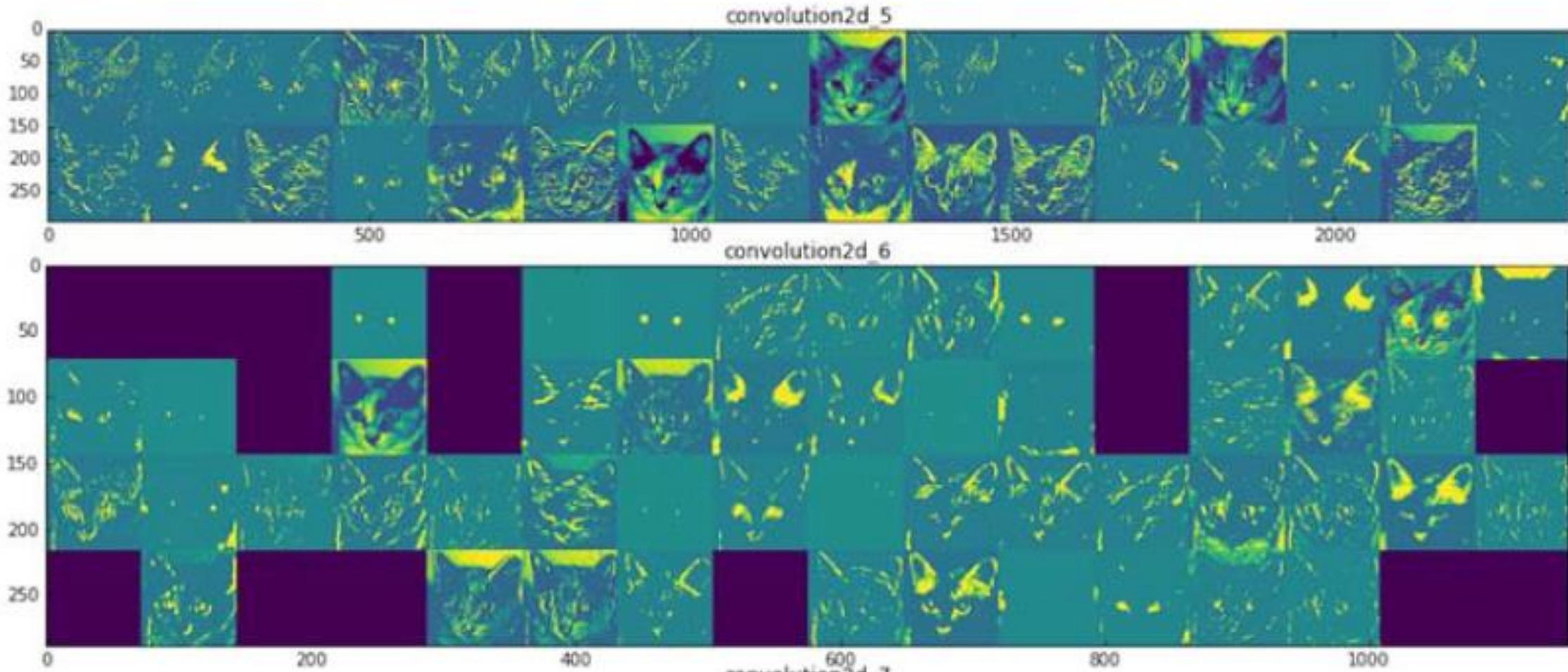
RECAP ON VALIDATION PERFORMANCE

- CATS AND DOGS
 - Basic model 72%
 - Dropout and augmentation 82%
 - Transfer learning 90% (overfitting)
 - Transfer with augmentation: 96% (not overfitting)
 - Training “into” the conv base top layers: 97% aka “fine tuning”

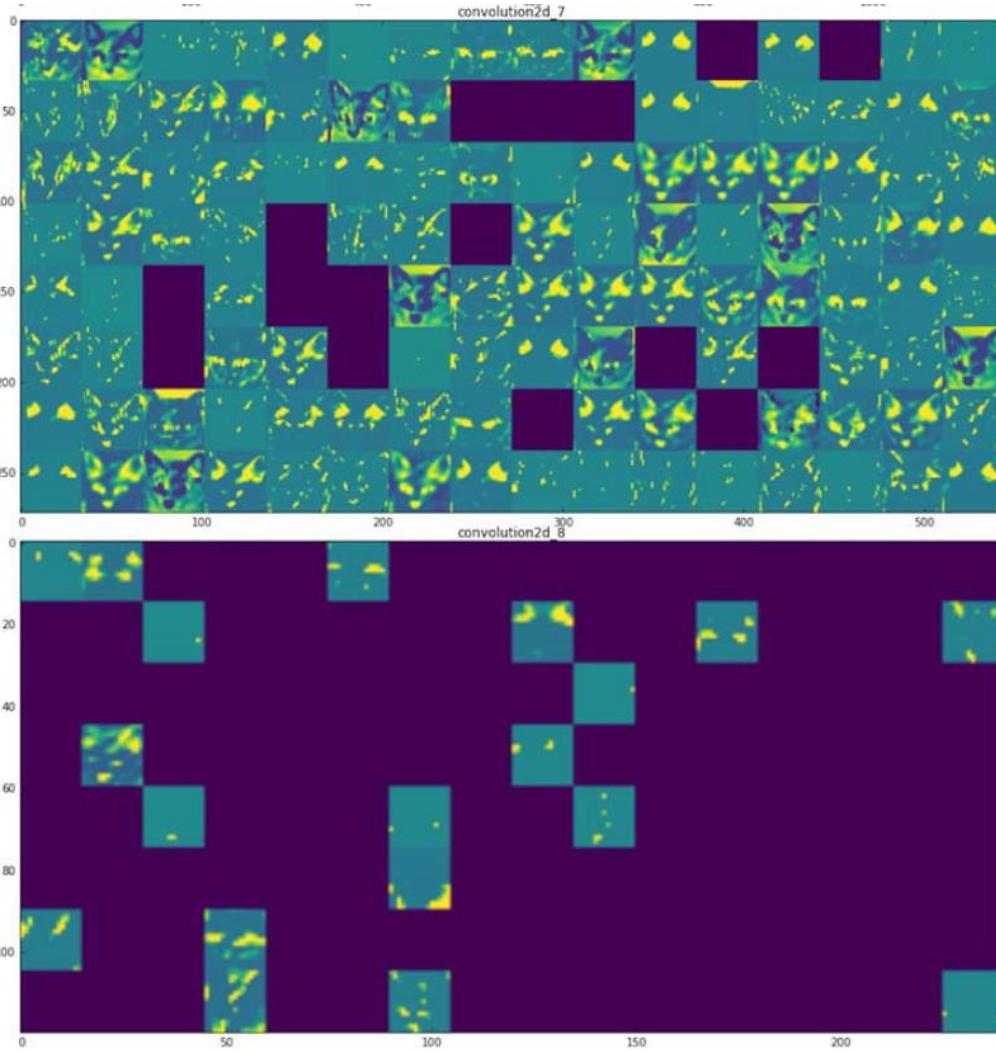
TAKE AWAYS

- Convnets best for vision tasks
- Can train from scratch even with small sets
- Overfitting is a huge issue, augmentation helps
- Reuse existing convnets (transfer)
- Can train into conv base but diminished returns

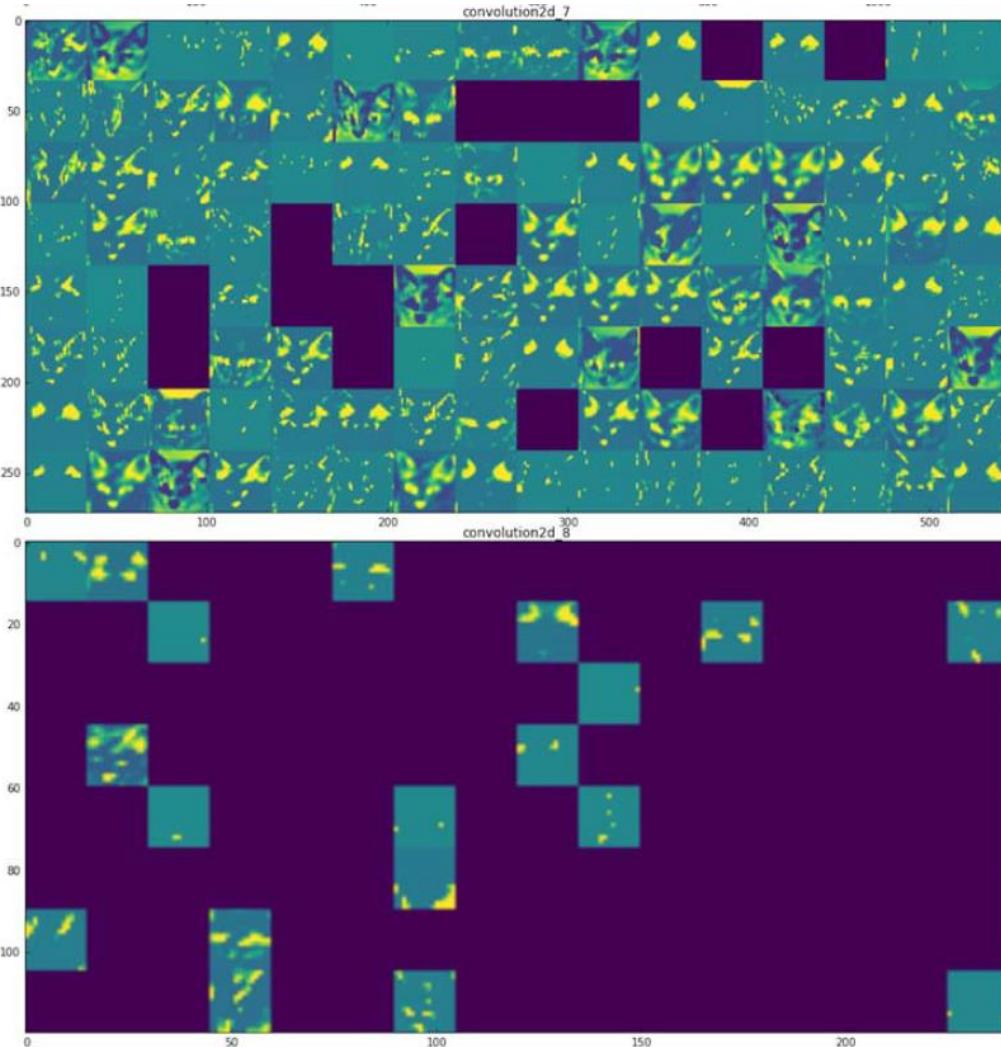
VISUALISING THE FILTERS (1)



VISUALISING THE FILTERS (2)

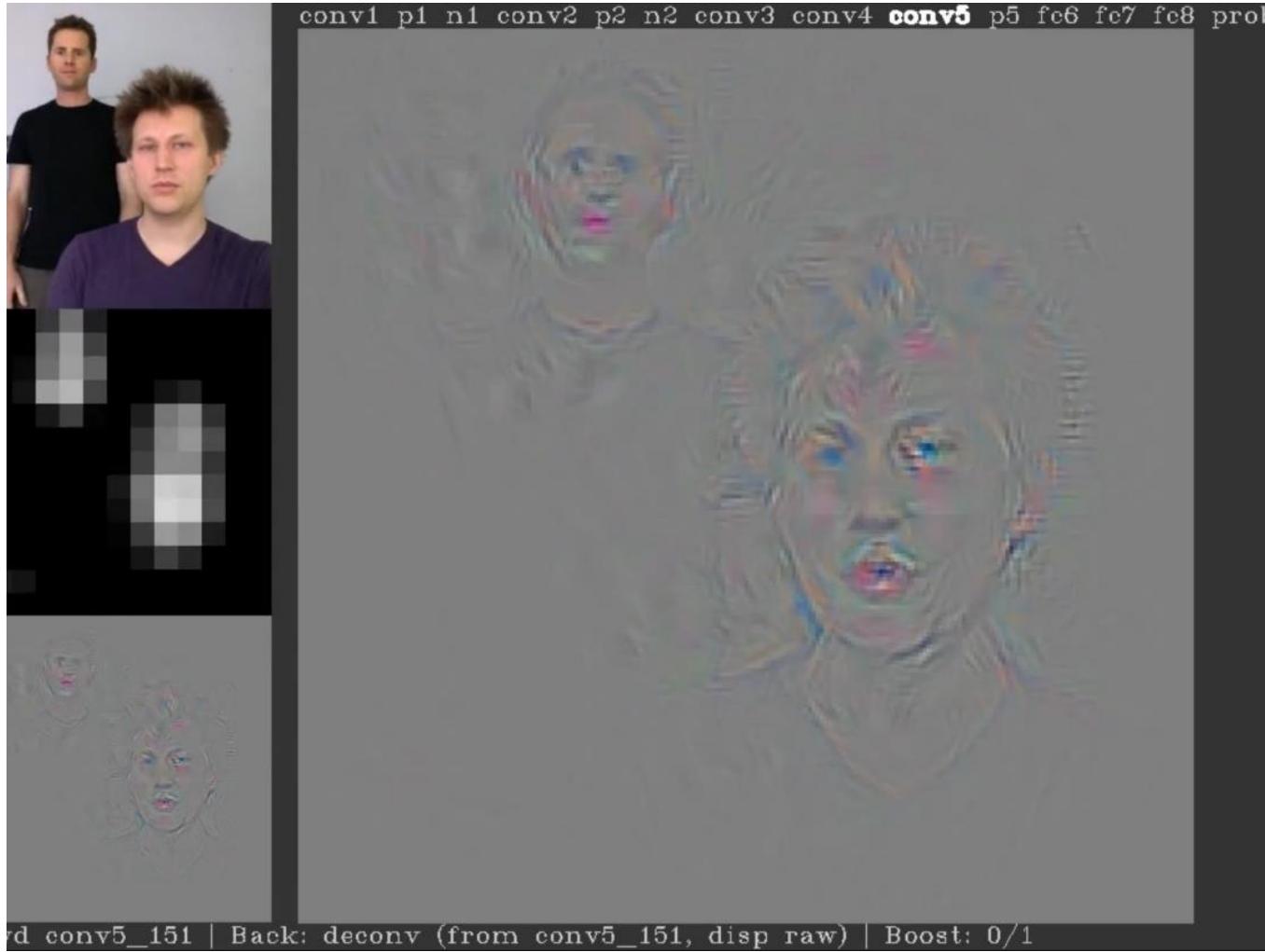


VISUALISING THE FILTERS (3)



- Early layers; content
- Later layers; class
- Sparsity

DEEP VISUALISATION TOOLBOX



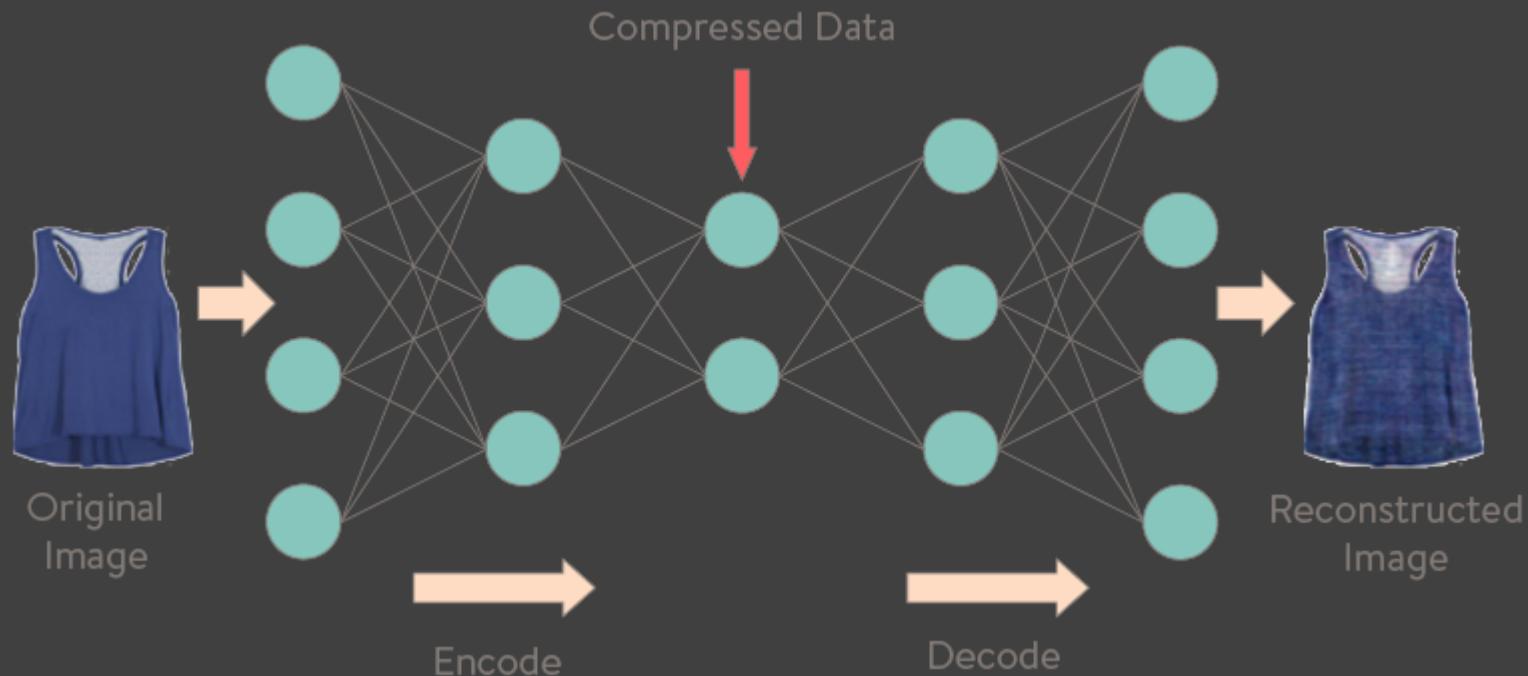
<https://www.youtube.com/watch?v=AgkfIQ4lGaM>

TWO INNOVATIVE DL ARCHITECTURES



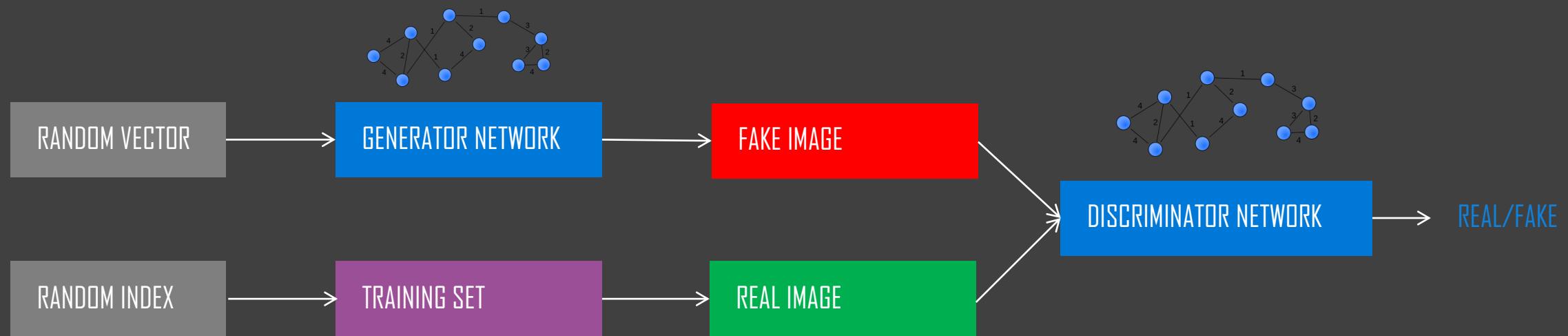


- Autoencoders are an unsupervised method to compress information into an “embedding” similar to unsupervised dimensionality reduction.



AUTOENCODERS

- An example; generative adversarial networks
- Game-theoretical approach to deep learning



Goodfellow et al. NIPS 2014



NOVEL ARCHITECTURES (GANS)



IMAGES GENERATED WITH GANS

Tim Scarfe

@ecsquendor

youtube.com/machinelearningatmicrosoft



THANK YOU!

