DL/Vision/CNNs

Microsoft ML OpenHack Series

Tim Scarfe, Ph.D

Principal Software Engineer @ Microsoft
Commercial Software Engineering (CSE)
@ecsquendor

Azure

Microsoft

Home
Trending
Subscriptions

Library
History
Watch Later
Purchases 51
Liked videos

SUBSCRIPTIONS
CloudTechTV ...
Jordan Knight
3Blue1Brown
Robert Miles
ciruselvirus
Michael Lanzetta
The Artificial In...
Show 11 more

MORE FROM YOUTUBE
YouTube Movies

Machine Learning at Microsoft

755 subscribers

CUSTOMISE CHANNEL | YOUTUBE STUDIO (BETA)

HOME | VIDEOS | PLAYLISTS | CHANNELS | DISCUSSION | ABOUT

Uploads ▶ PLAY ALL

A Joe Rogan style round table meditation on comput...

81 views • 3 weeks ago

Paper Reviews Call 004 -- (Mikolov)...

153 views • 3 weeks ago

Paper Reviews Call 006 - GloVe

179 views • 3 weeks ago

Paper Reviews Call 005 - Compact Prediction Trees

60 views • 3 weeks ago

Paper Reviews Call 003 -- Learning fine-grained image...

188 views • 1 month ago

FEATURED CHANNELS

Siraj Raval
SUBSCRIBED

Microsoft
SUBSCRIBE

Brandon Rohrer
SUBSCRIBED

Microsoft Research
SUBSCRIBE

POPULAR CHANNELS ON YOUTUBE

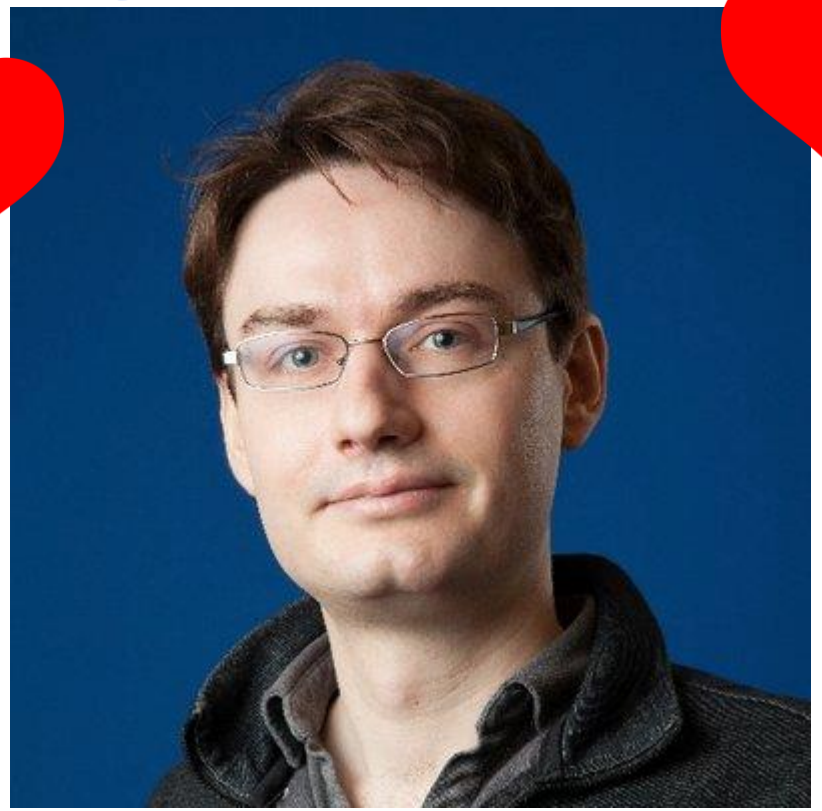
A PRACTICAL GUIDE TO DEEP LEARNING

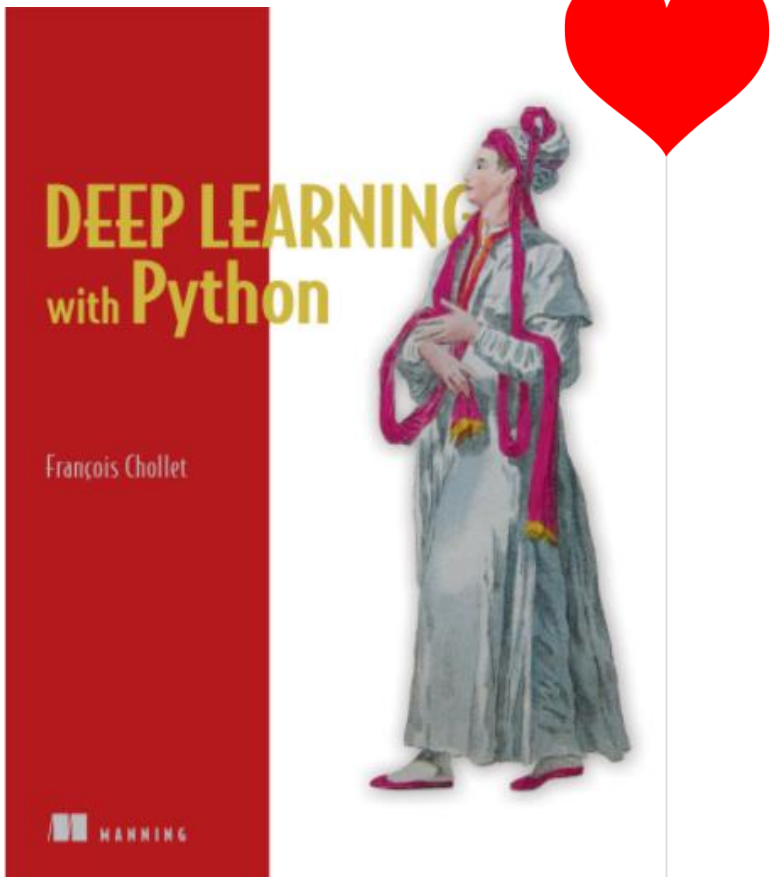
Tess Ferrandez - Microsoft - @TessFerrandez

Tess Ferrandez

Principal Software Engineer @ Microsoft

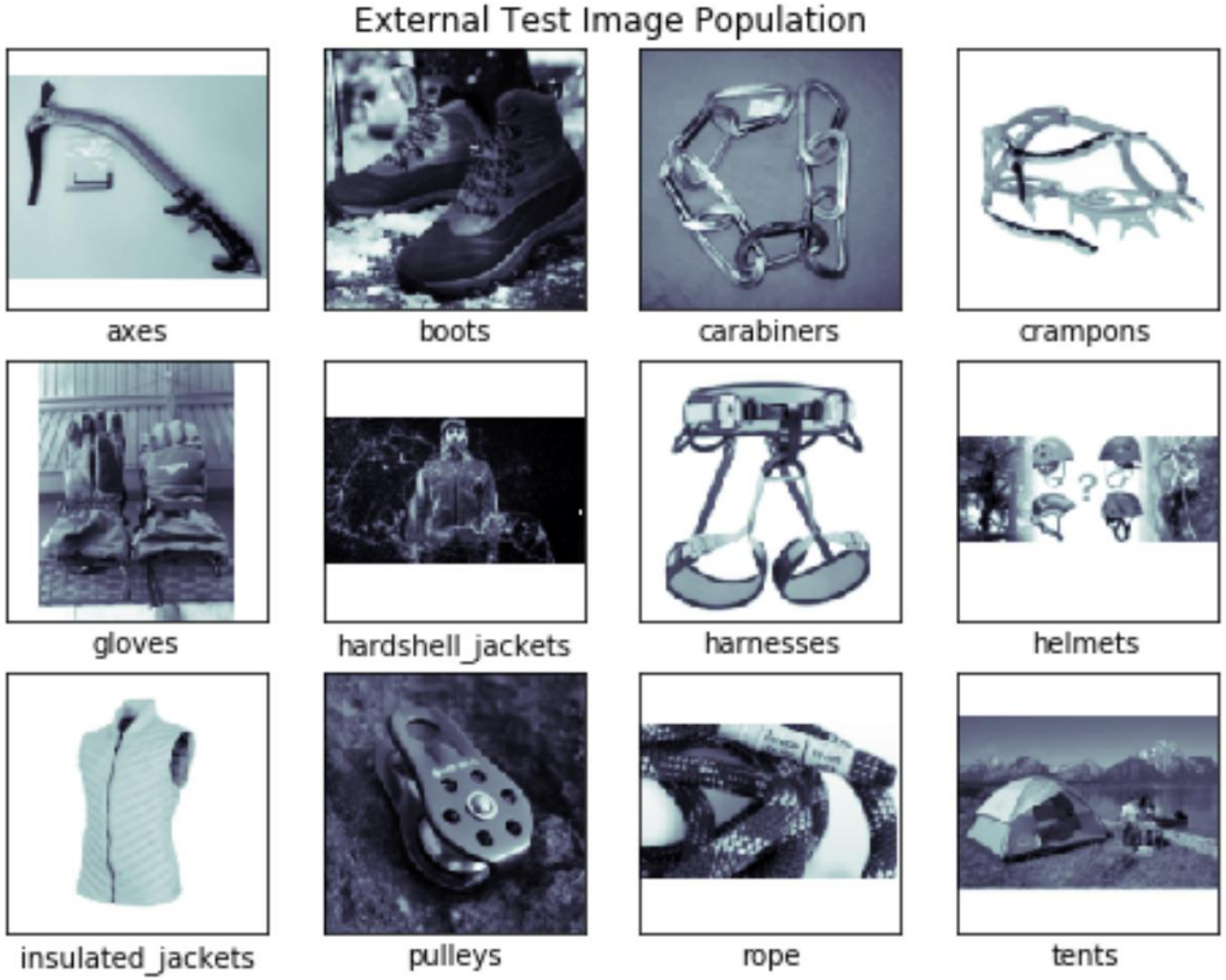
Francois Chollet. The man. The legend.
Please buy his book!





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

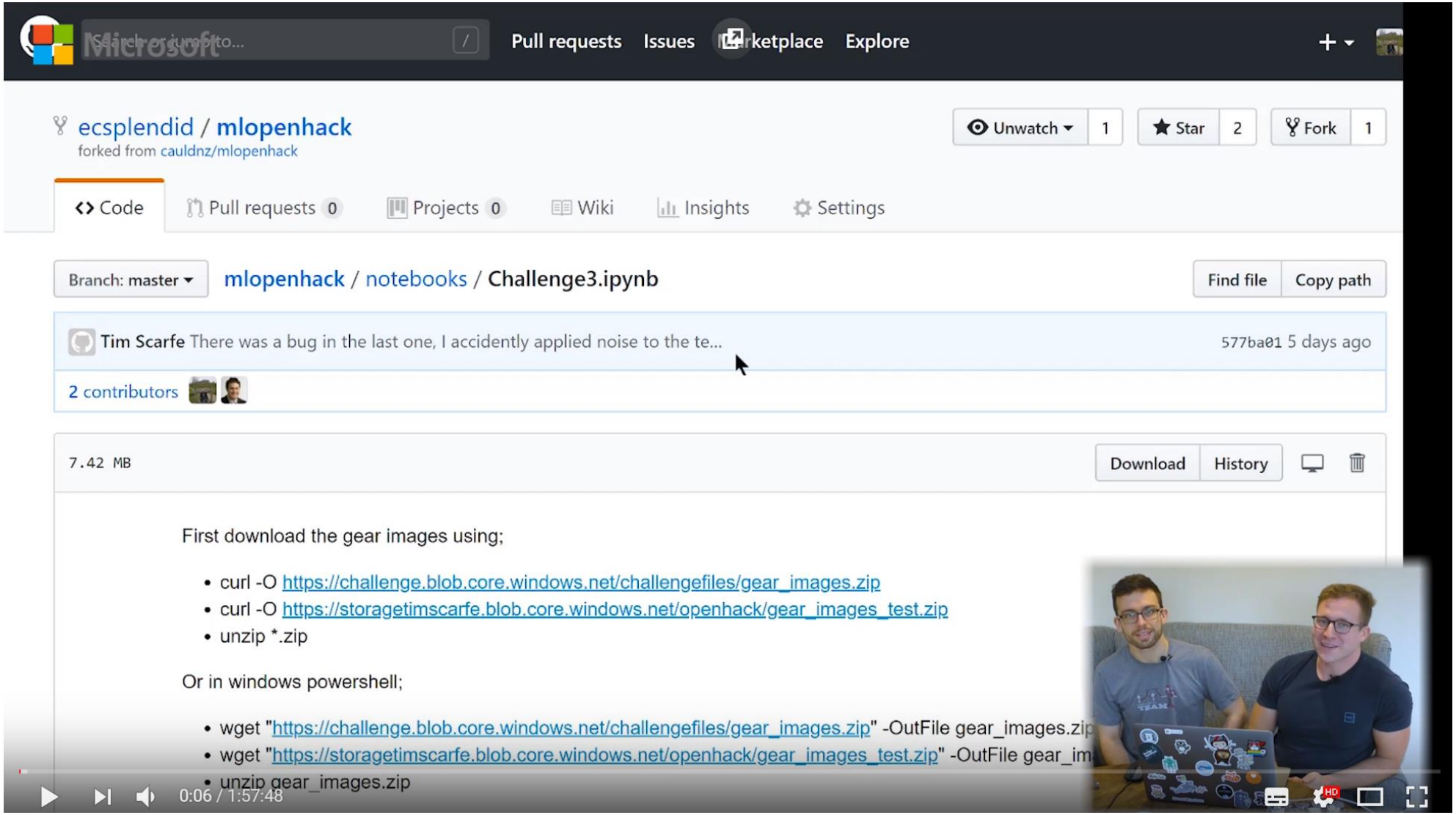
Microsoft



https://storagetimscarfe.blob.core.windows.net/openhack/gear_images_test.zip

Experiment	F1 Validation	F1 Test/External	Difference
Histogram 20 bins Trees	0.55	0.11	0.44
Raw Pixels trees	0.66	0.15	0.51
SVM HOG	0.29		
Pixels->PCA + SVM	0.9	0.18	0.72
CNN raw pixels	0.91	0.23	0.68
Raw Pixels + Noise + Trees	0.34		
Pixels+Noise->PCA + SVM	0.68	0.03	0.65
CNN raw pixels + noise	0.94	0.03	0.91
Noise + Augmented 10k trees	0.15		
Noise + Augmented 20k PCA SVM	0.49	0.04	0.45
Noise + Augmented 20k HOG PCA SVM	0.48	0.1	0.38
Augmented 20K CNN	0.83	0.29	0.54
Augmented 20K HOG + PCA	0.61	0.25	0.36
Augmented 20K + PCA + SVM	0.8	0.21	0.59
Noise + Augmented 20K CNN	0.80	0.12	0.68
Transfer learning - Train just on external test set		**0.39**	
Transfer learning Augmented 20K CNN	0.89	0.50	0.39

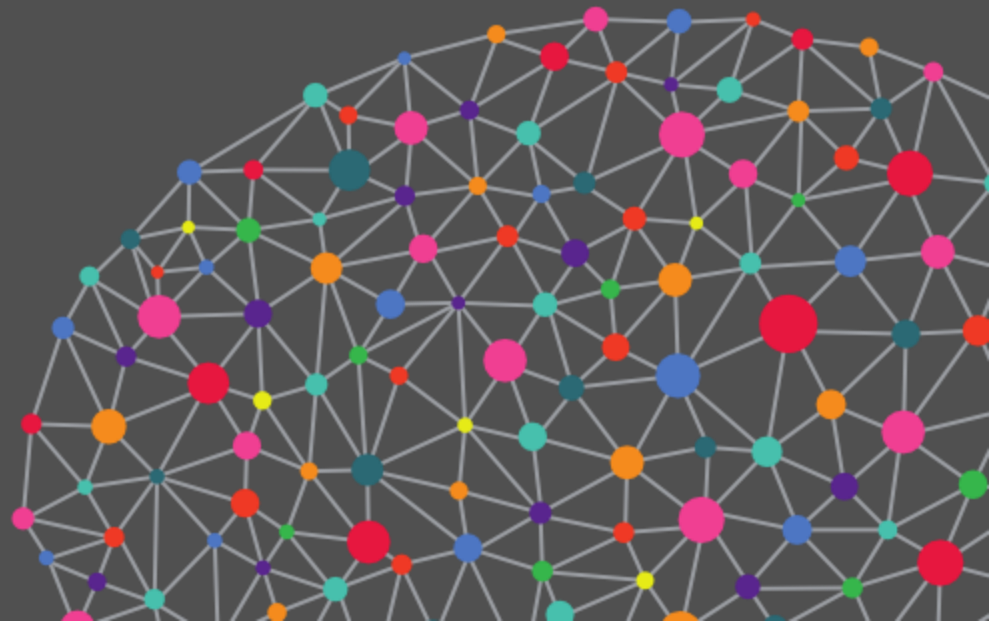
https://github.com/ecsplendid/mlopenhack/blob/master/notebooks/Challenge3.ipynb



https://www.youtube.com/watch?v=ZNC-ZNHcQ88

TALK OUTLINE

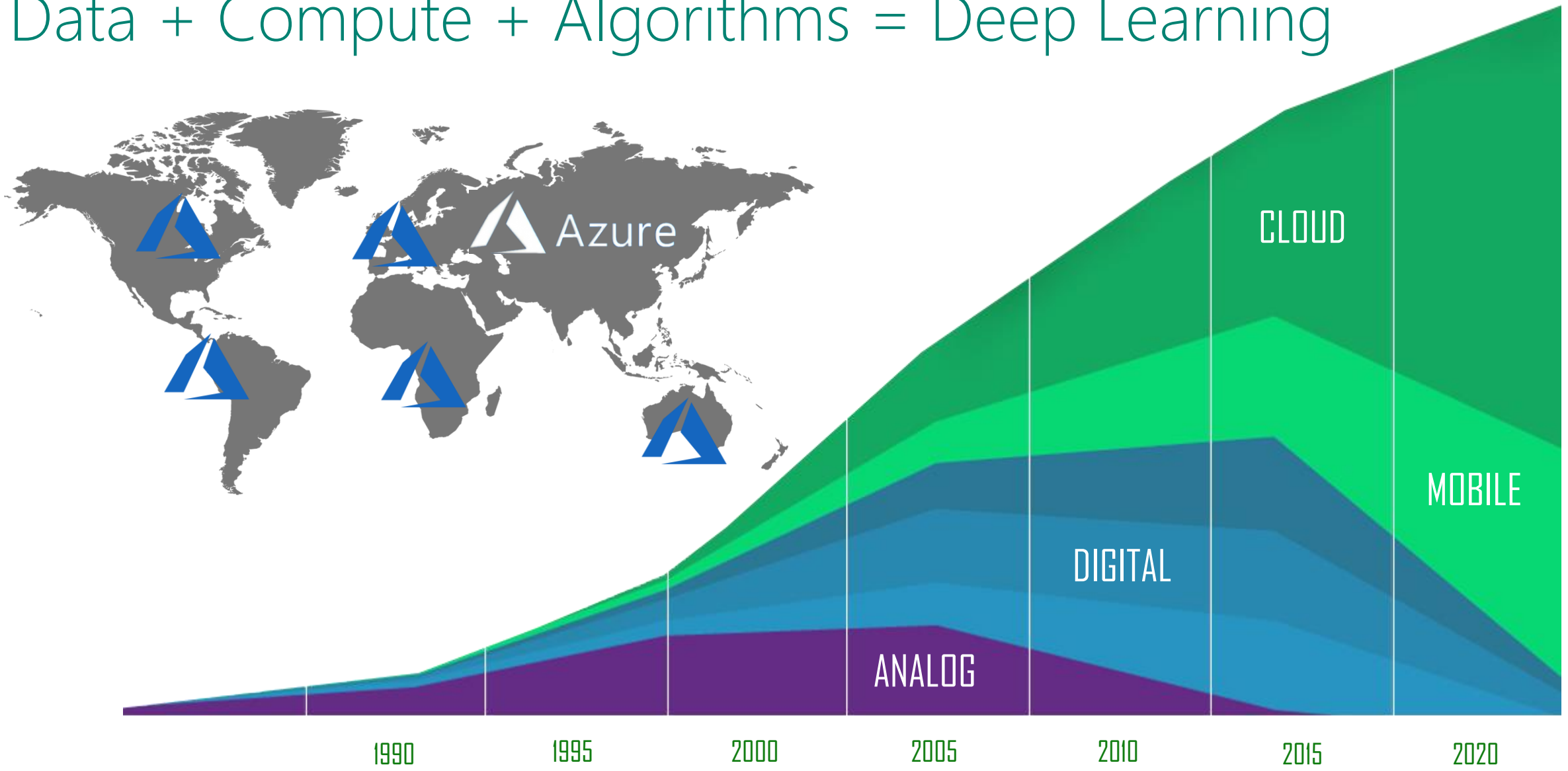
- High level concepts of deep learning
- Advanced topics
- Old days of vision
- How does convolution work?
- MNIST example in Keras code
- Data Augmentation on Cats+Dogs dataset
- Transfer Learning
- Visualising CNNs



Microsoft

Microsoft

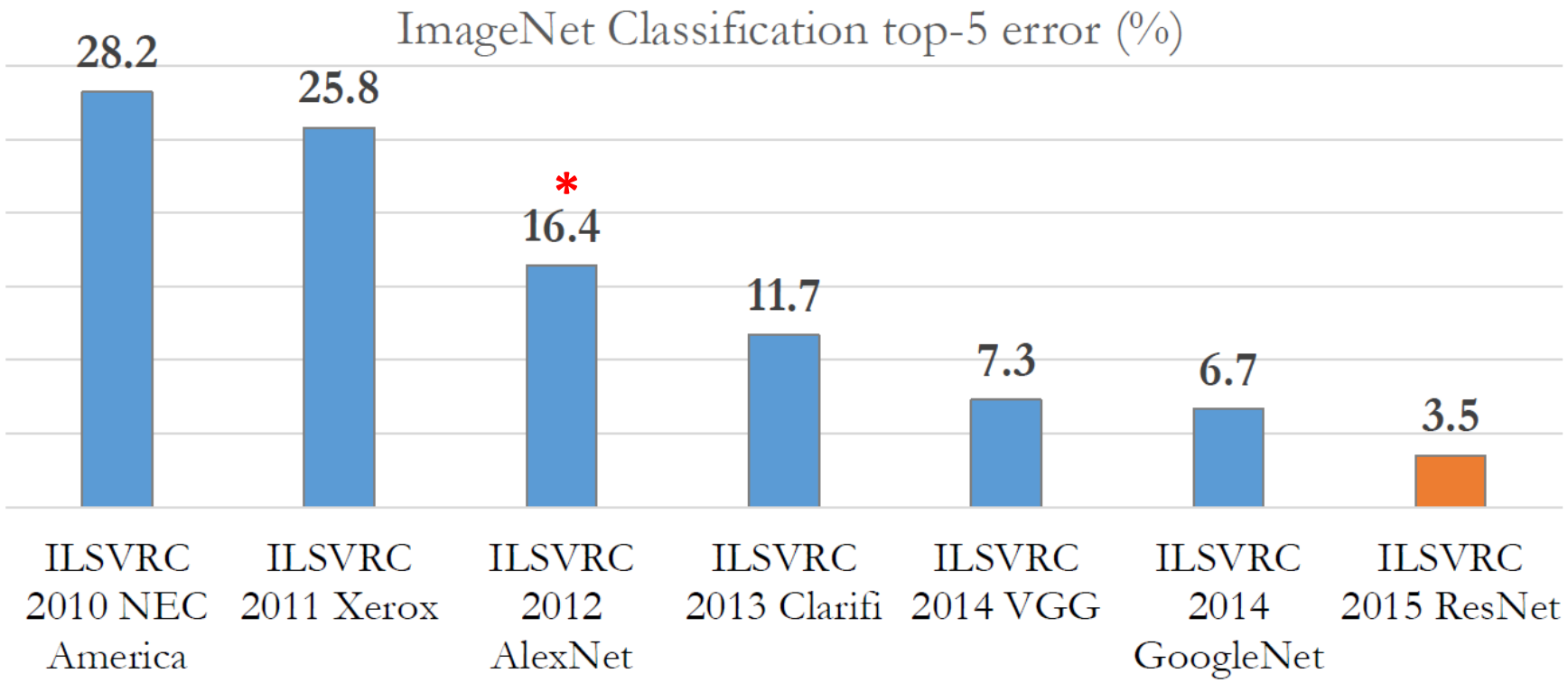
Data + Compute + Algorithms = Deep Learning



Microsoft

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

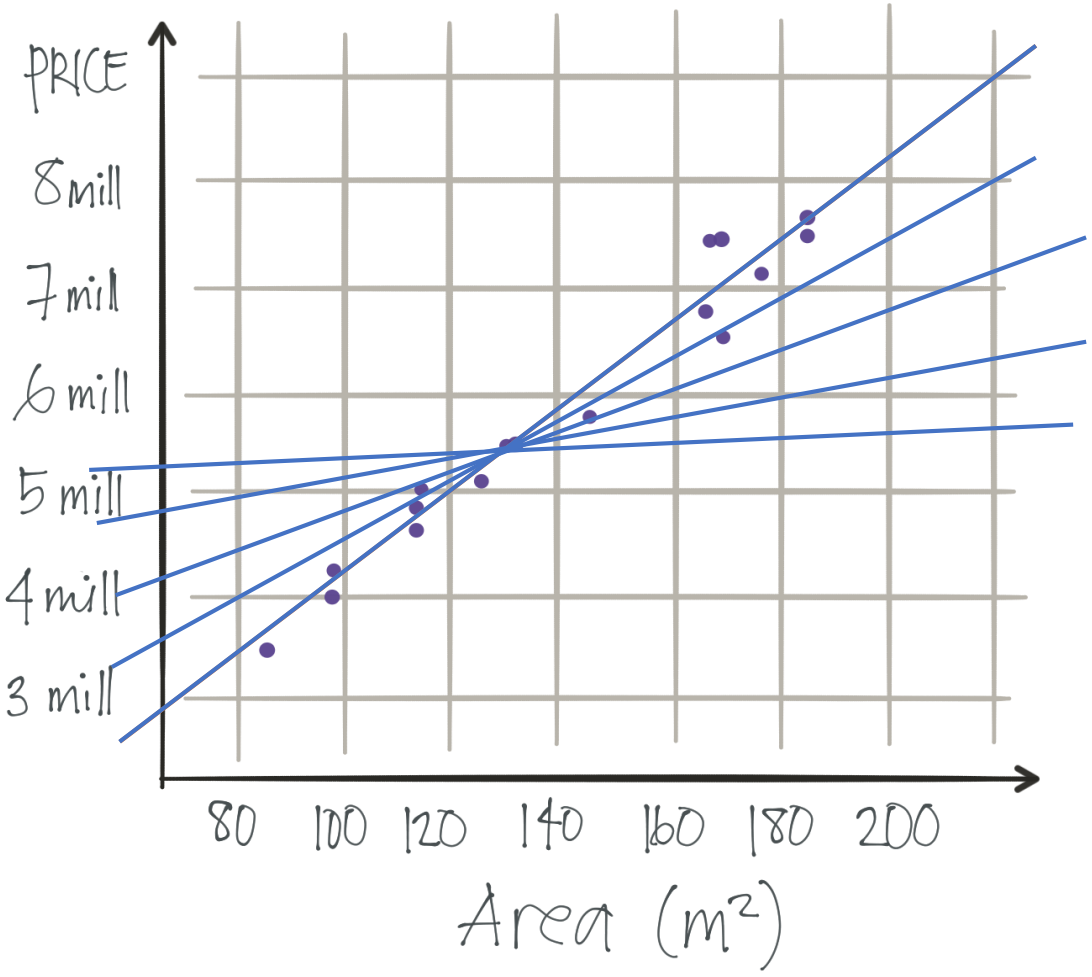
AREA (m^2)	TYPE	ZIP·CODE	PRICE (kkr)
134	HOUSE	90210	5 495
115	TOWNHOME	11436	4700
167	HOUSE	90210	7500
185	HOUSE	11436	7775
84	HOUSE	79021	3500
98	TOWNHOME	11436	4000

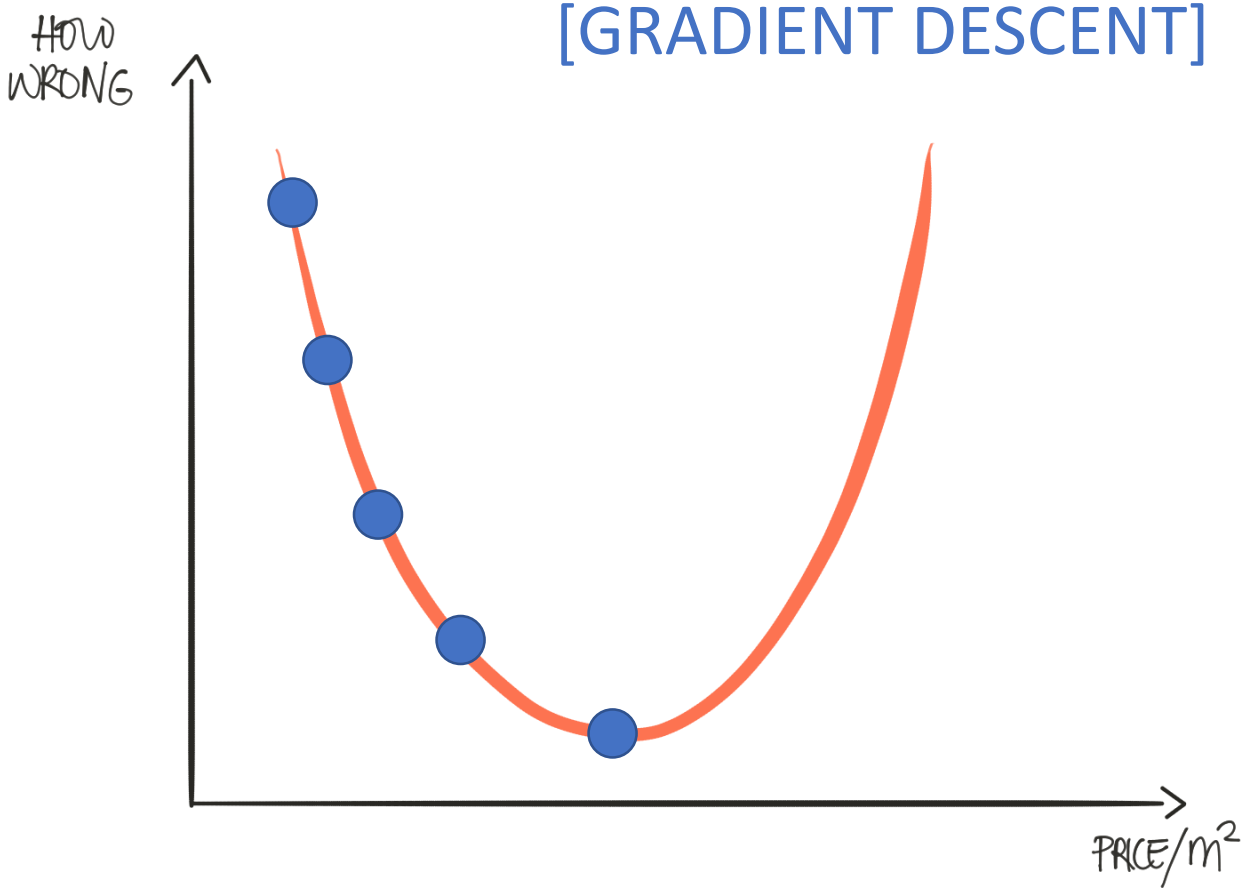
```
int EstimatePrice(...){
    price = 10000 +
            6700 * area_in_sqm +
            20000 * has_pool +
            10000 * new_kitchen +
            5000 * neighborhood_quality;
    return price;
}
```

```
Price = b + w1*area_in_sqm + w2*has_pool + ...
```

[LINEAR REGRESSION]

[GRADIENT DESCENT]



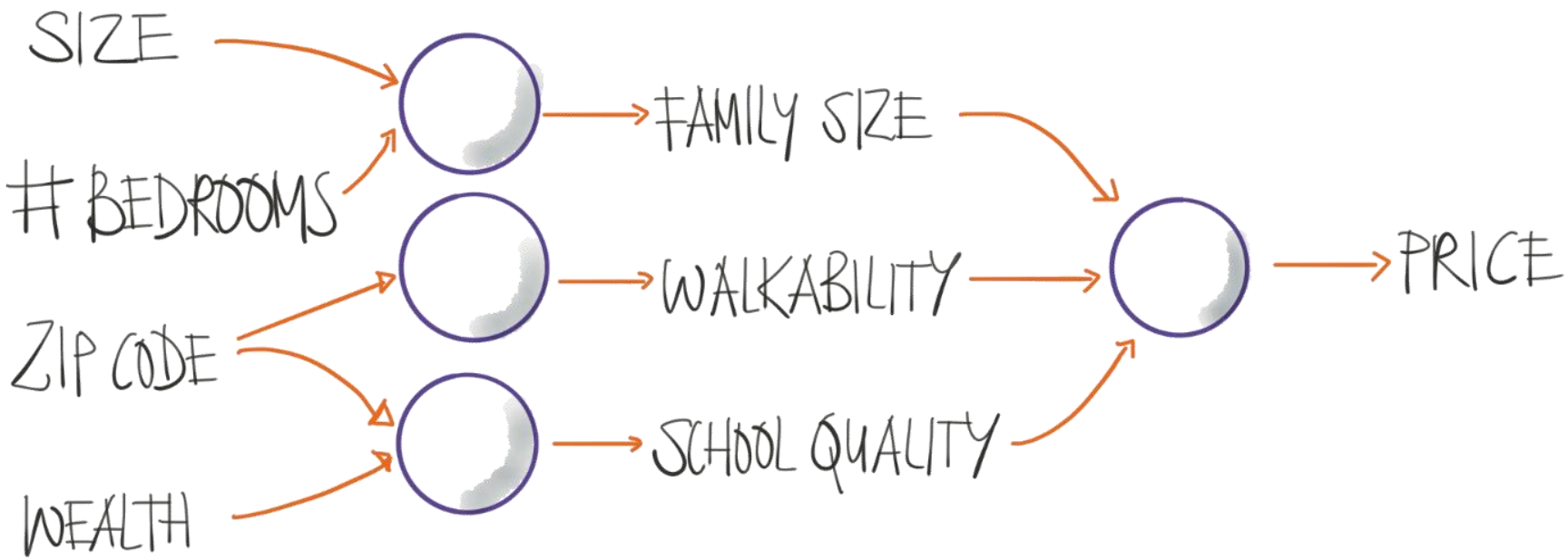



```
Price = b + w1*area_in_sqm
```

```
int EstimatePrice(...){
    price = 10000 +
            6700 * area_in_sqm +
            20000 * has_pool +
            10000 * new_kitchen +
            5000 * neighborhood_quality;
    return price;
}
```

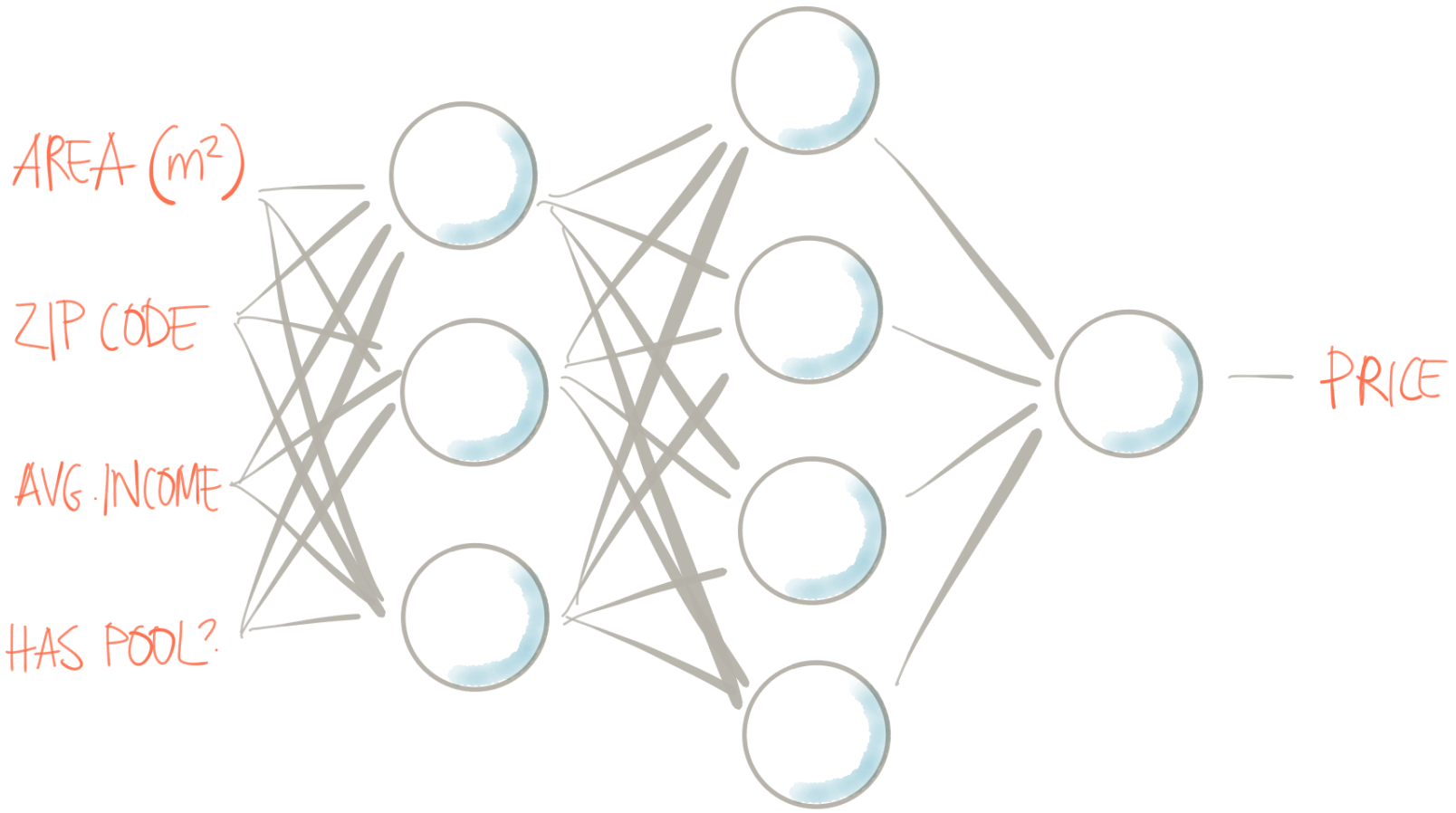
```
Price = b + w1*area_in_sqm + w2*has_pool + ...
```

HOUSE PRICES



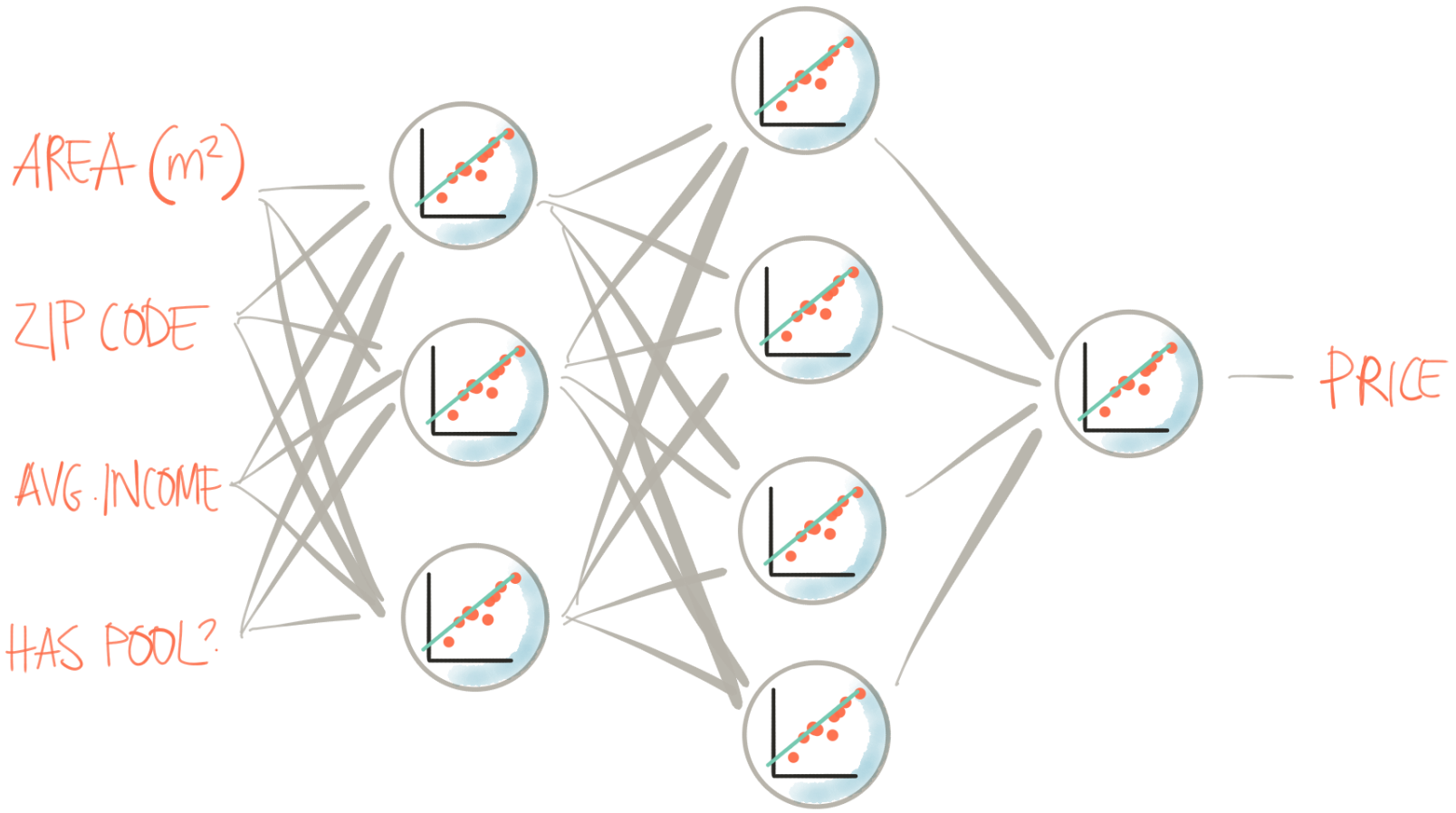
[NEURAL NET]

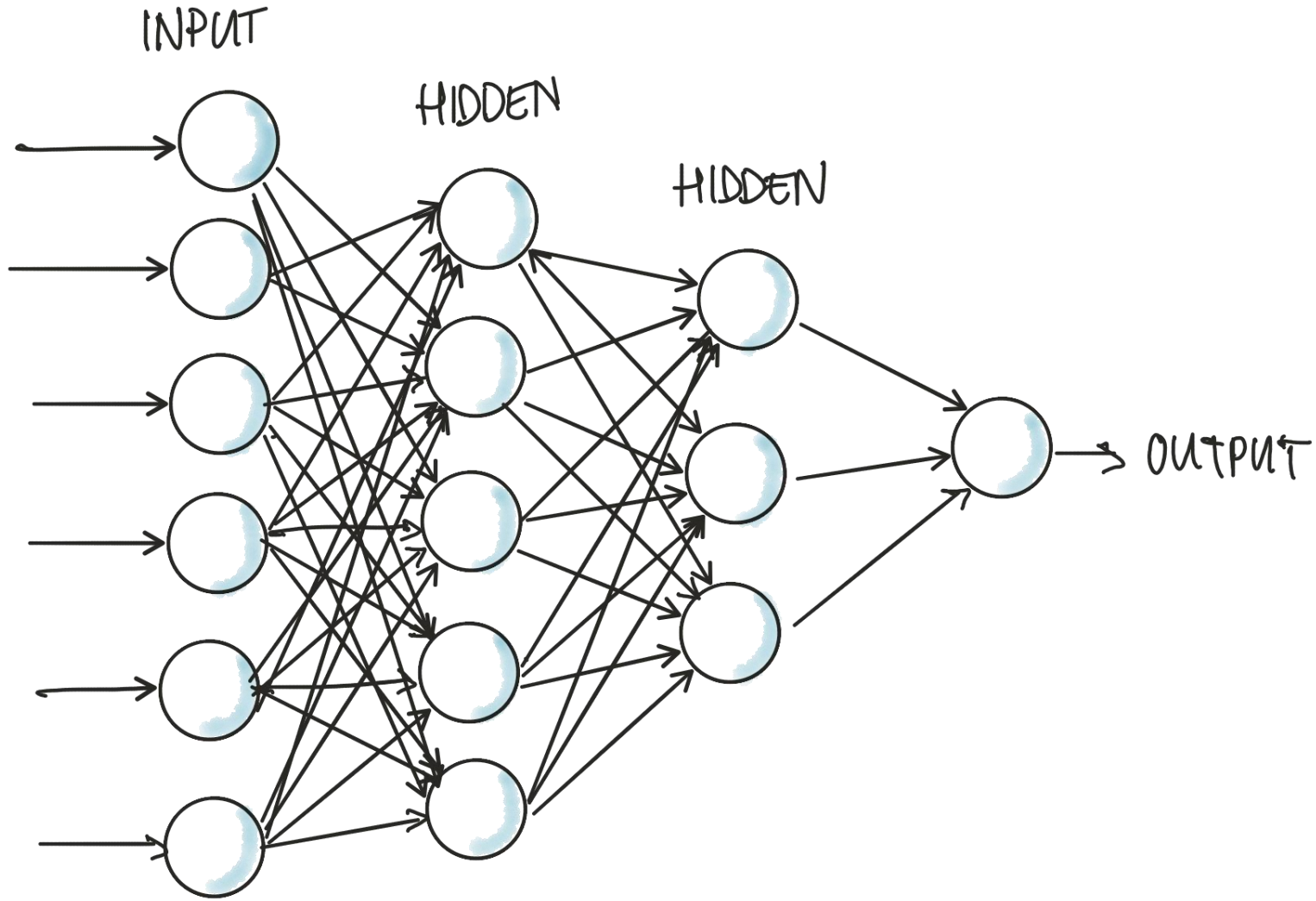
HOUSE PRICES

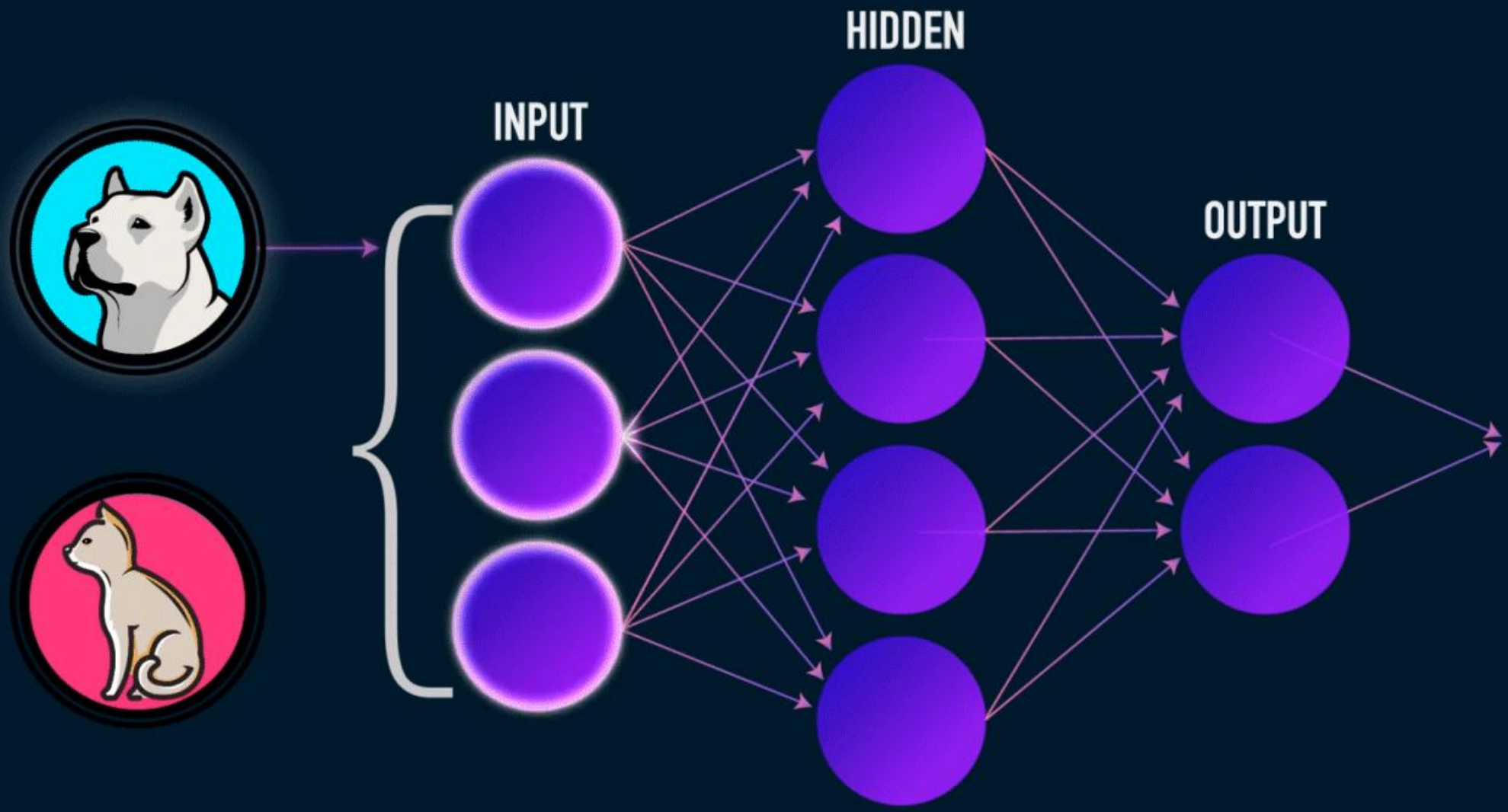


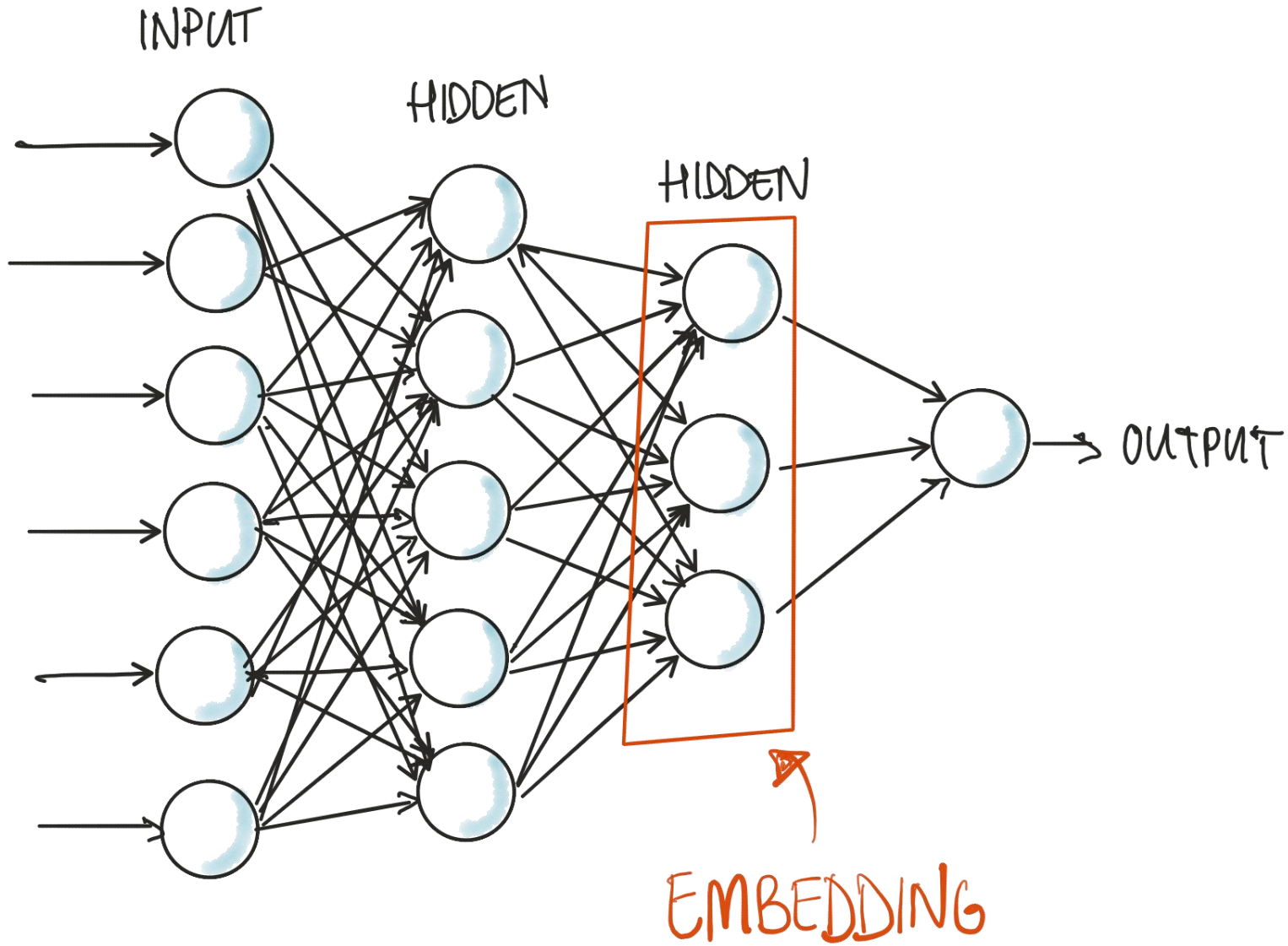
[NEURAL NET]

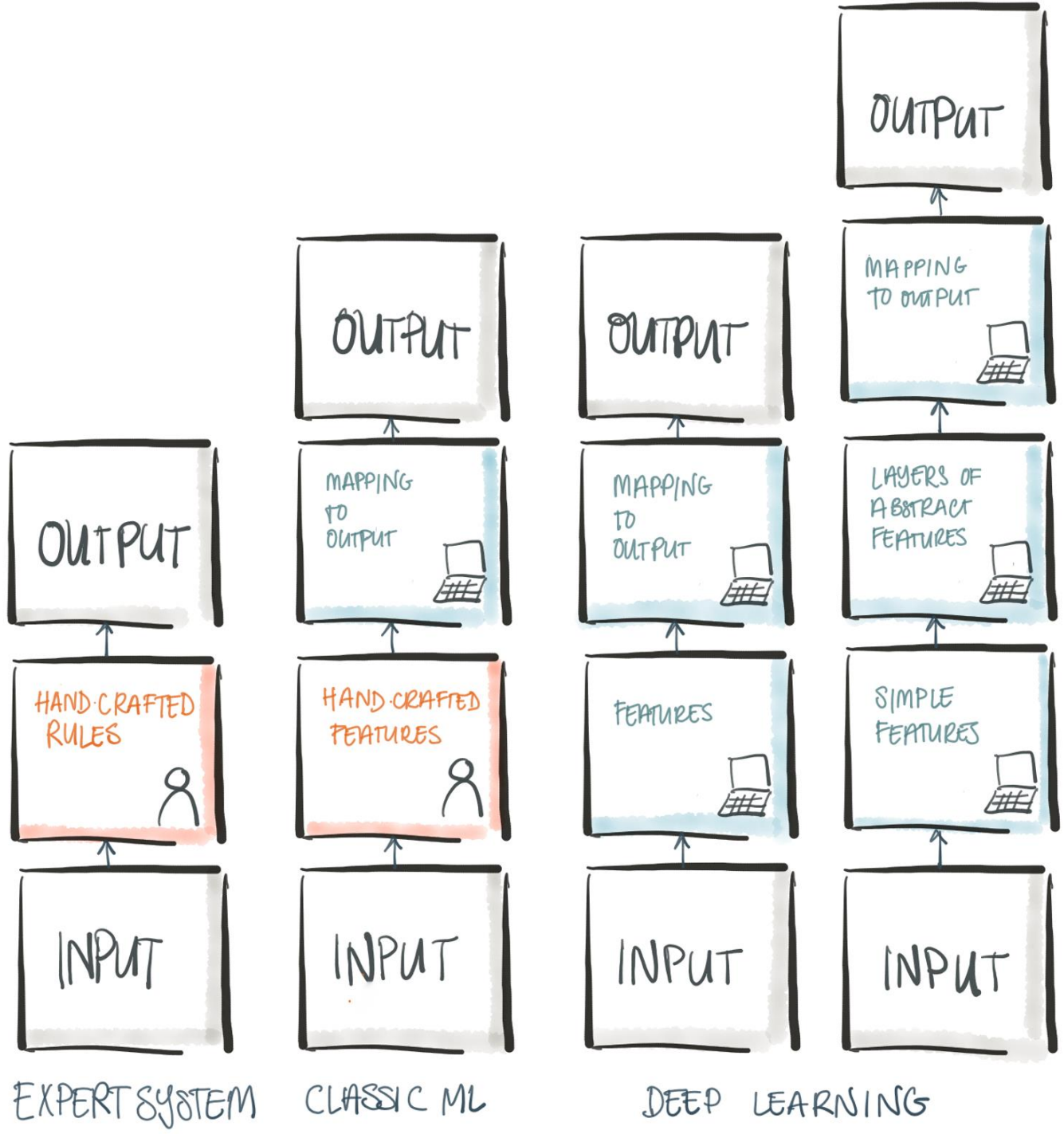
HOUSE PRICES




INPUT
HIDDEN
HIDDEN
OUTPUT

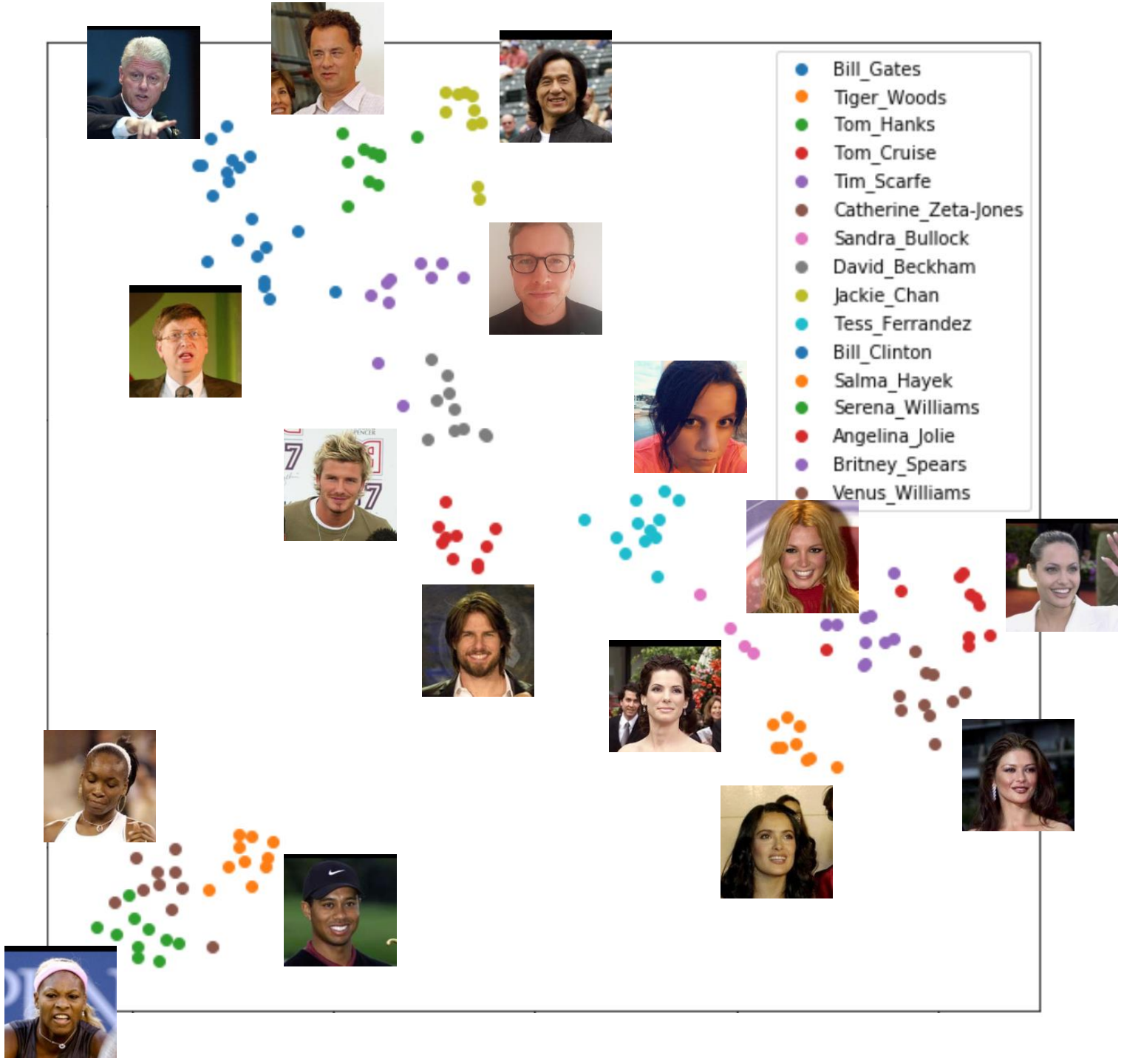

HIDDEN
INPUT
OUTPUT


INPUT
HIDDEN
HIDDEN
OUTPUT
EMBEDDING


OUTPUT
HAND-CRAFTED RULES
INPUT
EXPERT SYSTEM
OUTPUT
MAPPING TO OUTPUT
HAND-CRAFTED FEATURES
INPUT
CLASSIC ML
OUTPUT
MAPPING TO OUTPUT
FEATURES
INPUT
OUTPUT
MAPPING TO OUTPUT
LAYERS OF ABSTRACT FEATURES
SIMPLE FEATURES
INPUT
DEEP LEARNING

[FACENET]

T-SNE
Projection of
128D to 2D



Microsoft

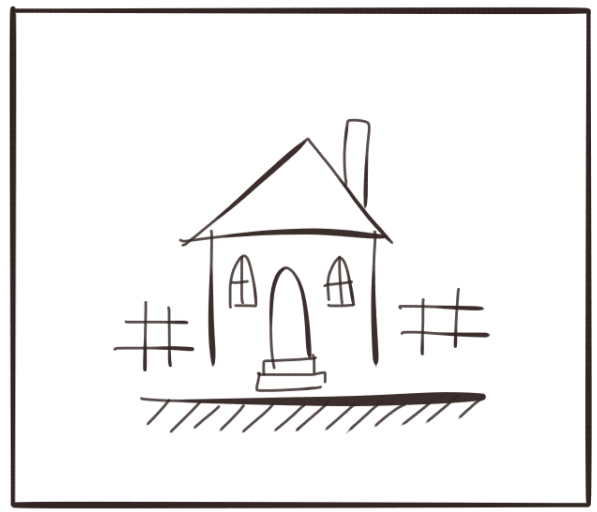
Word2Vec: Word vectors are cool!

https://projector.tensorflow.org

Microsoft

Word2Vec: First names

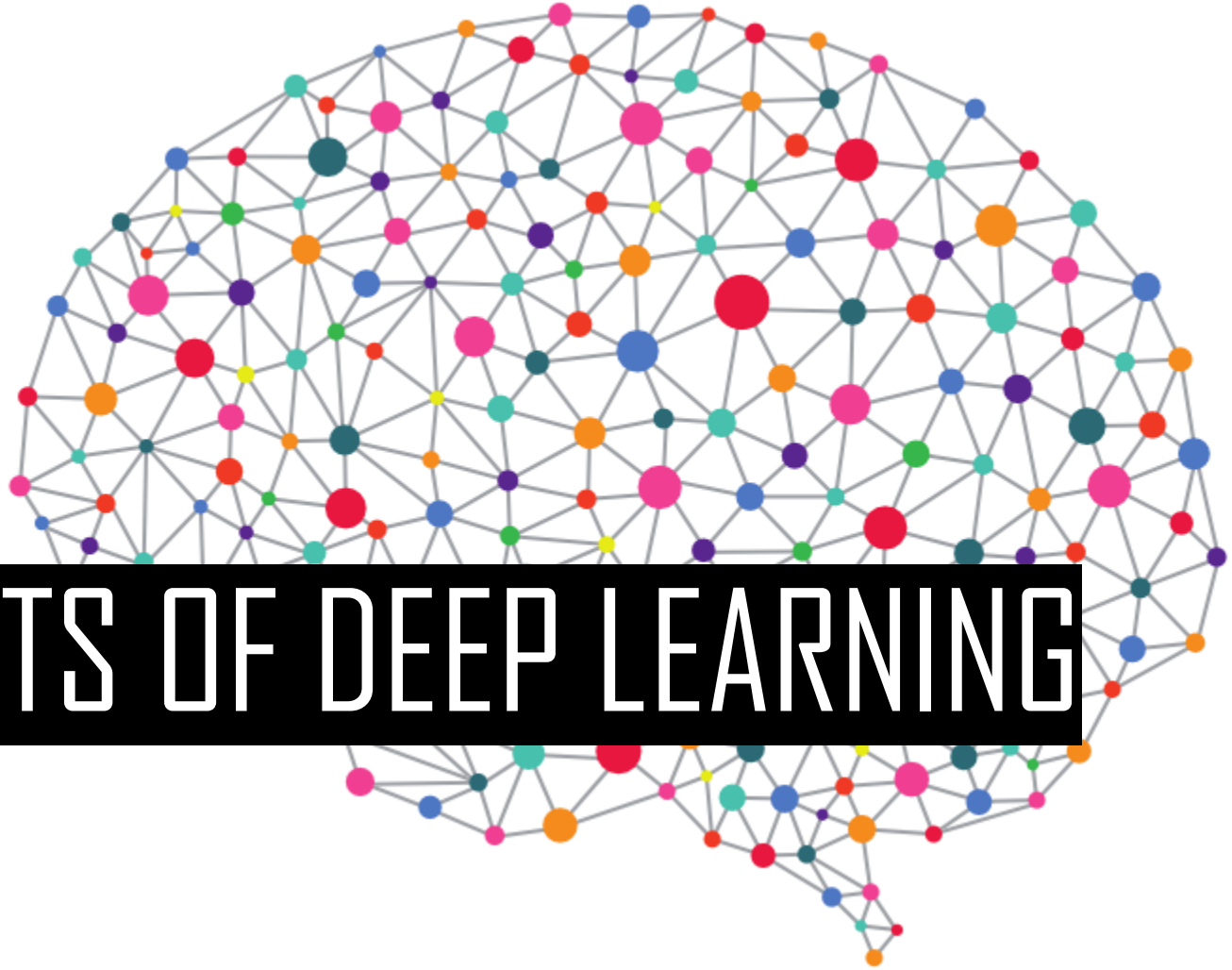
https://projector.tensorflow.org



A HOUSE WITH
A NICE FENCE AND
A BEAUTIFUL
DOOR

A Style-Based Generator Architecture for Generative Adversarial Networks

Tero Karras, Samuli Laine, Timo Aila

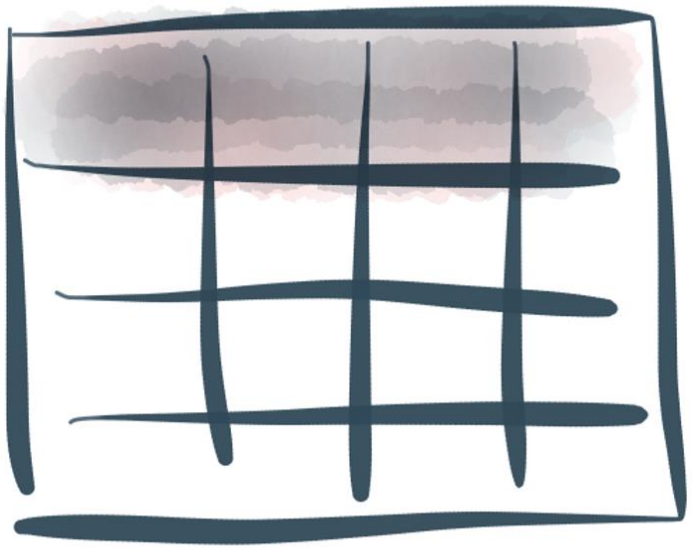


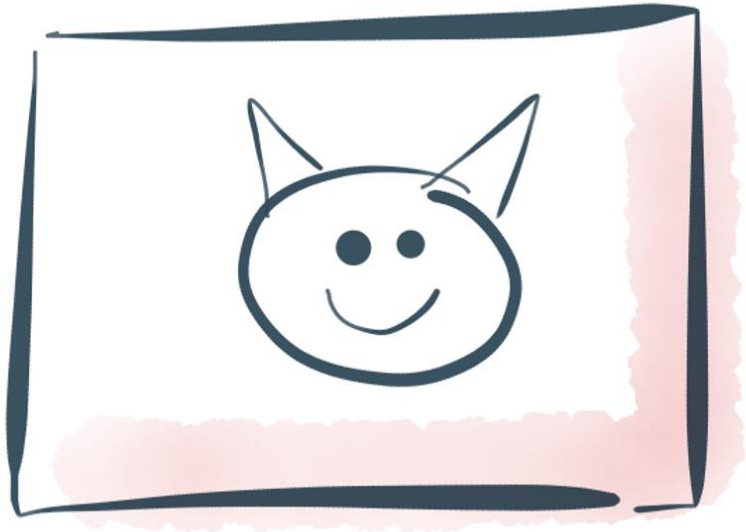
DISTILLED CONCEPTS OF DEEP LEARNING

Microsoft

[SPACE]

[SEQUENCE]





Dense
Neural Network

Convolutional
Neural Network

Recurrent
Neural Network

Microsoft

Traditional ML

Manual Features → Trainable Classifier

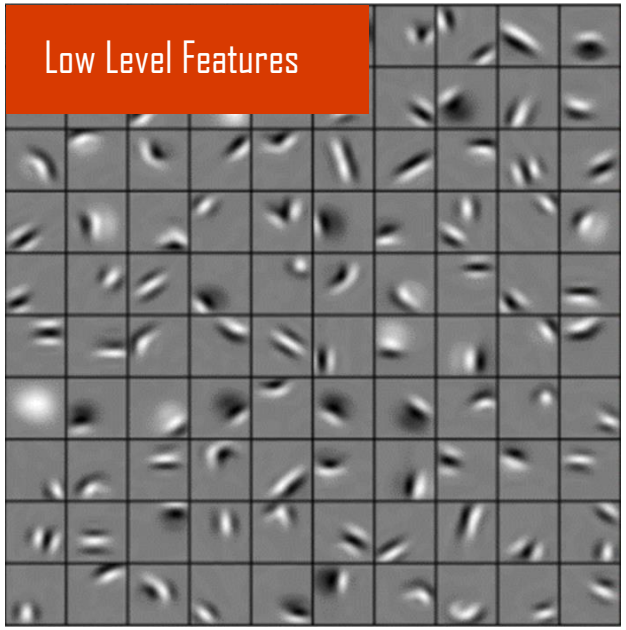
Deep Learning

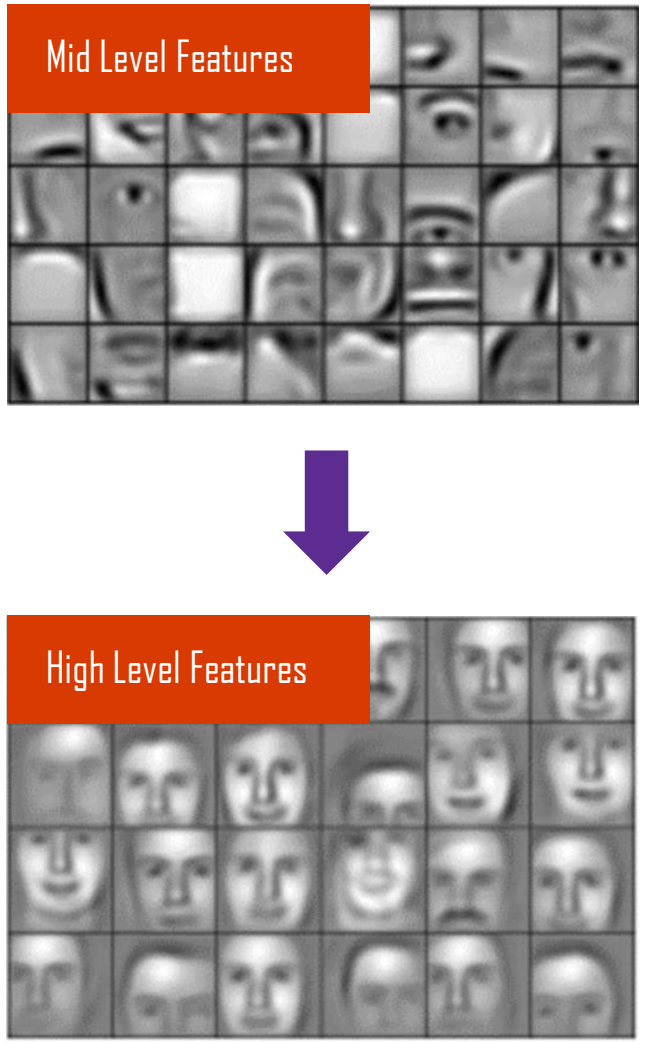
Representations are hierarchical and trained automatically

Low Level Features → Mid Level Features → High Level Features → Trainable Classifier

Deep Learning = learning a representation

Microsoft



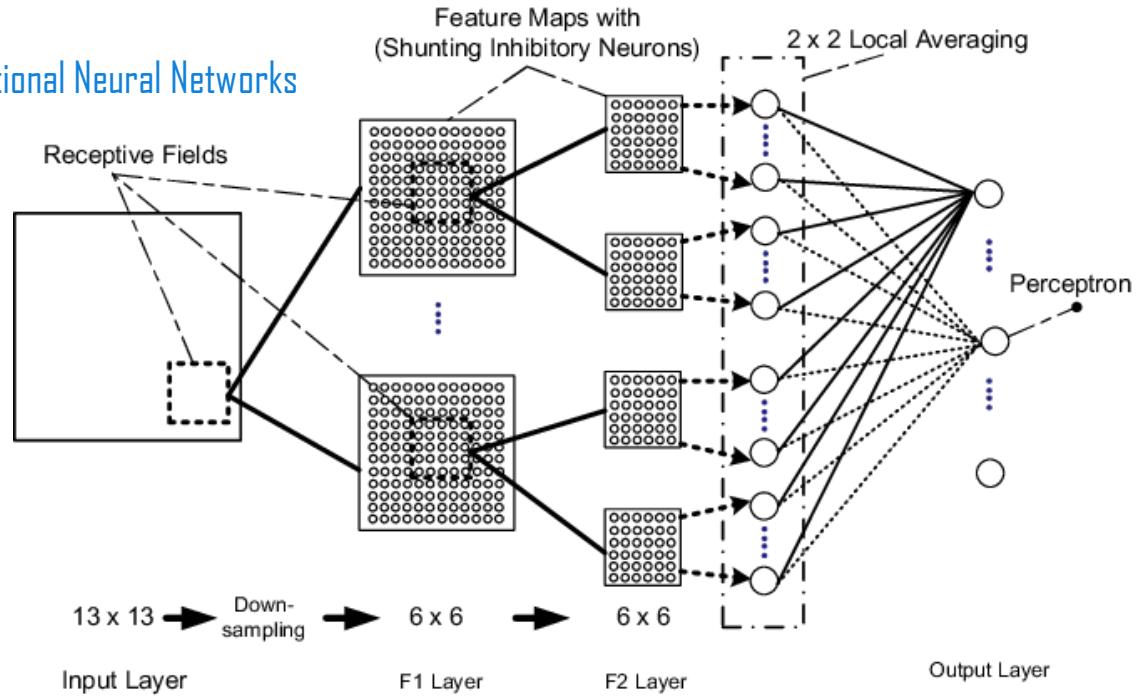


Intermediate representations are learned automatically

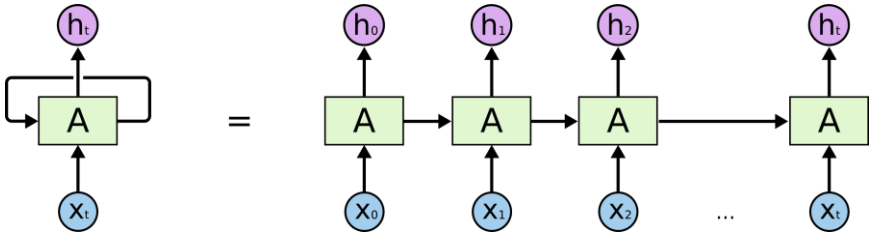
Unlike other algorithms, NNs can natively encode useful and obvious relationships in the data domain

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

Convolutional Neural Networks



Recurrent Neural Networks



NATIVE DATA-DOMAIN FEATURES

Microsoft

Microsoft

Universal functional approximators

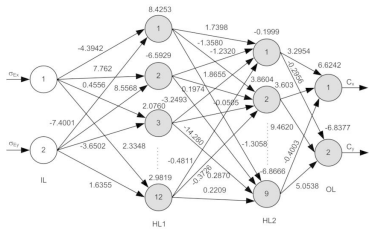
STRUCTURED

SPEECH

LANGUAGE

IMAGES

VIDEO



STRUCTURED

SPEECH

LANGUAGE

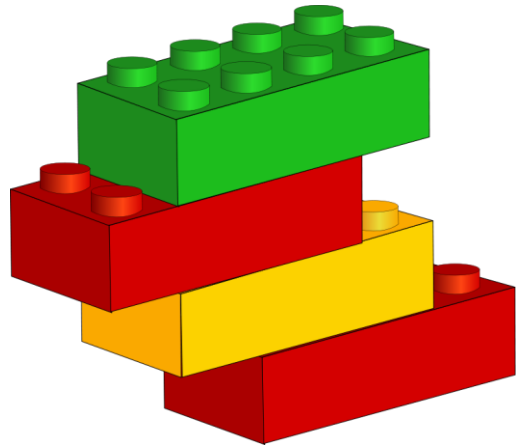
IMAGES

VIDEO

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

Microsoft

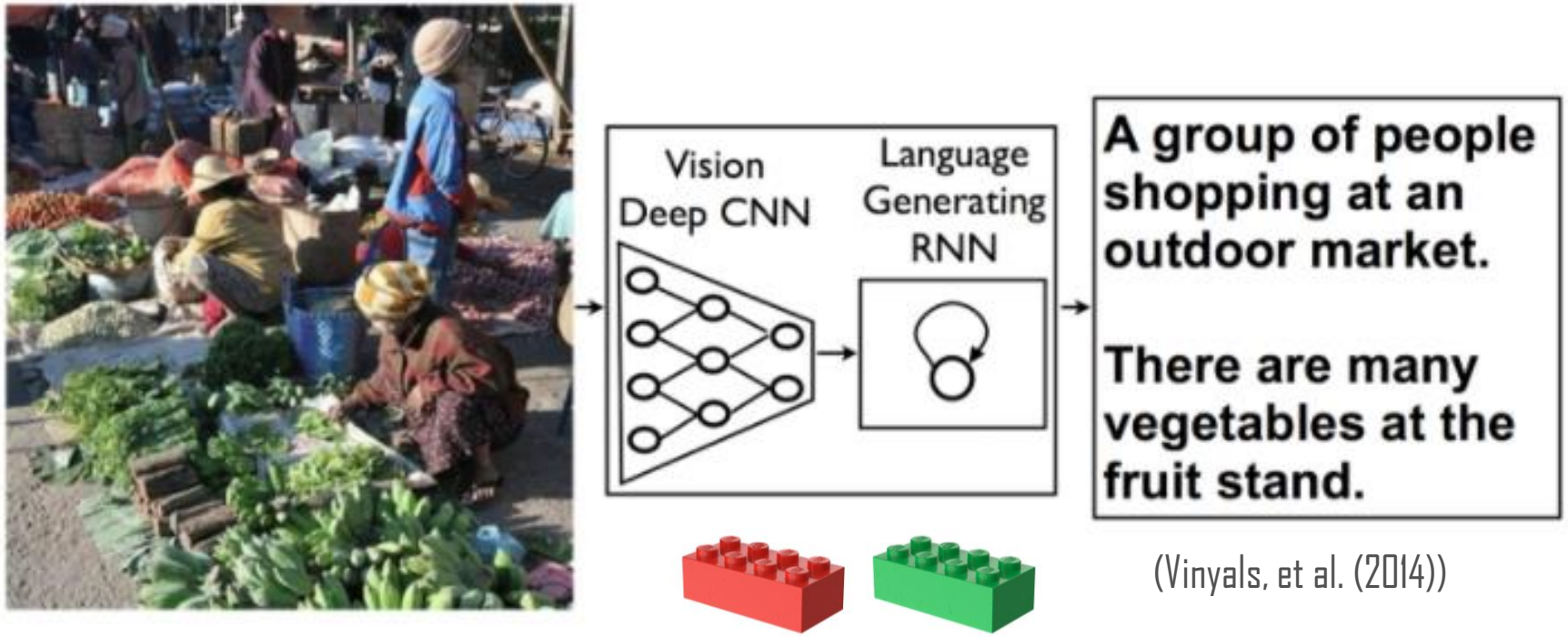
Composability



- 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

Microsoft

Building predictive architectures like lego blocks

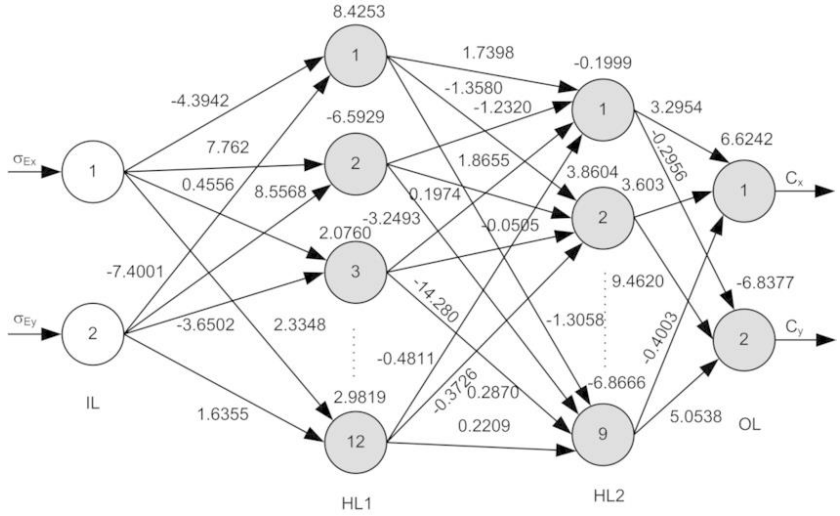


(Vinyals, et al. (2014))

Microsoft

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



https://medium.com/@karpathy/software-2-0-a64152b37c35

Microsoft

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

Microsoft

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
- Structured Data (sometimes)

Start with logistic regression

Epoch 003,928

Learning rate: 0.03 | Activation: ReLU | Regularization: None | Regularization rate: 0 | Problem type: Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$

3 HIDDEN LAYERS

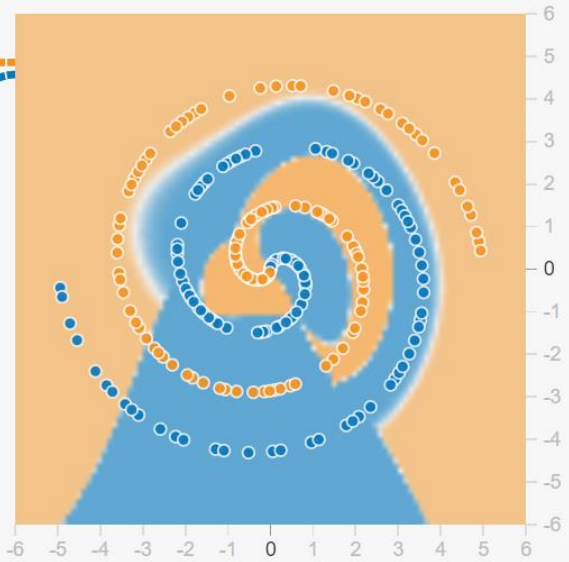
5 neurons | 5 neurons | 2 neurons

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

OUTPUT

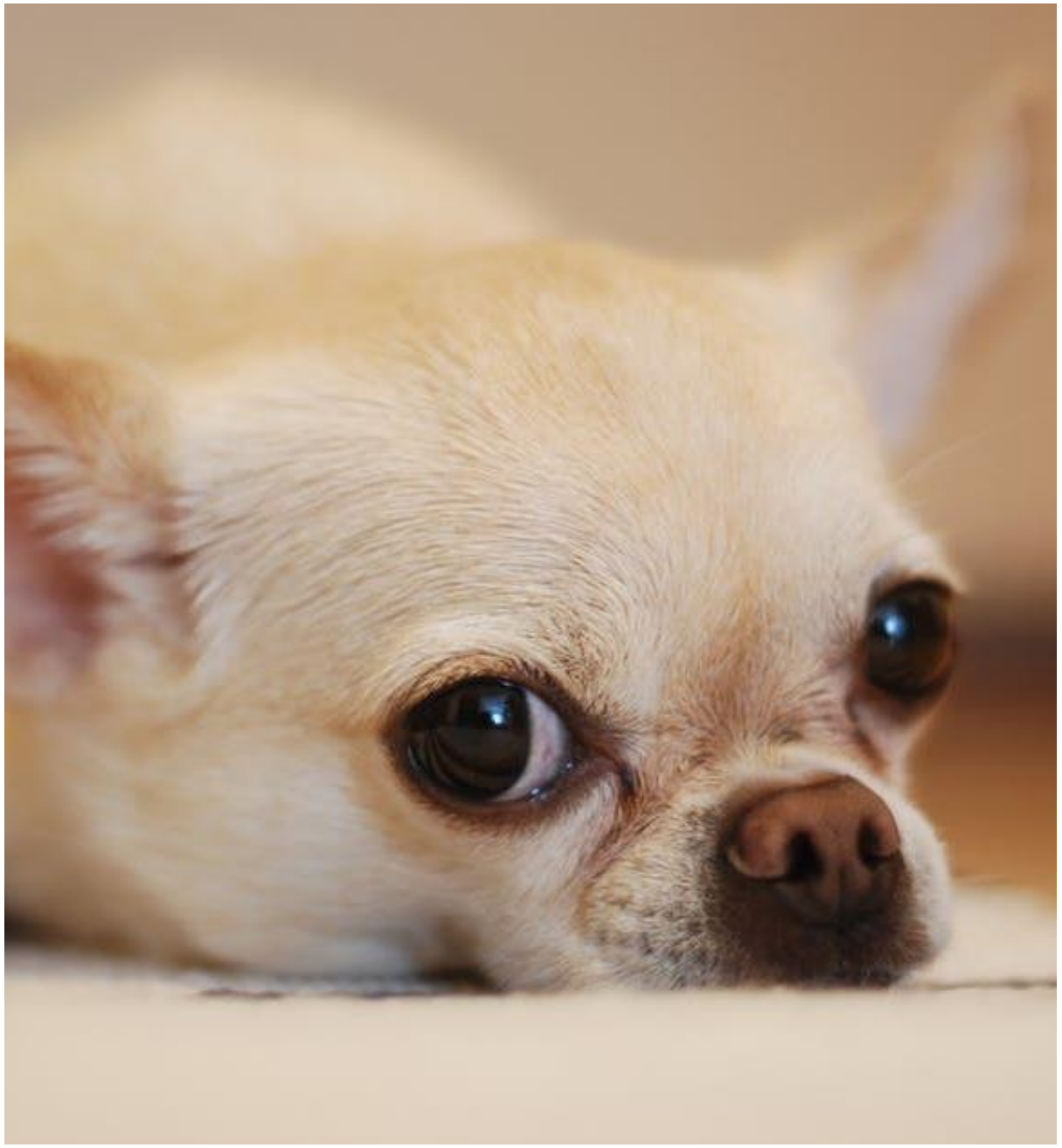
Test loss 0.211
Training loss 0.185



Colors shows data, neuron and weight values.

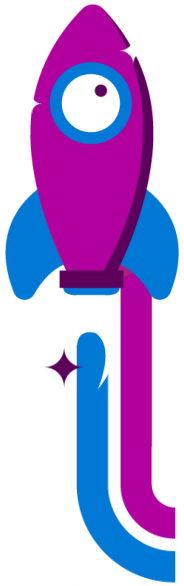
-1 0 1

Show test data | Discretize output



```
IF PICTURE.CONTAINS(2 EYES && NOSE)
    RETURN CHIHUAHUA
ELSE IF PICTURE.CONTAINS(PAPERCUP)
    RETURN MUFFIN
ELSE
    RETURN I HAVE NO CLUE
```

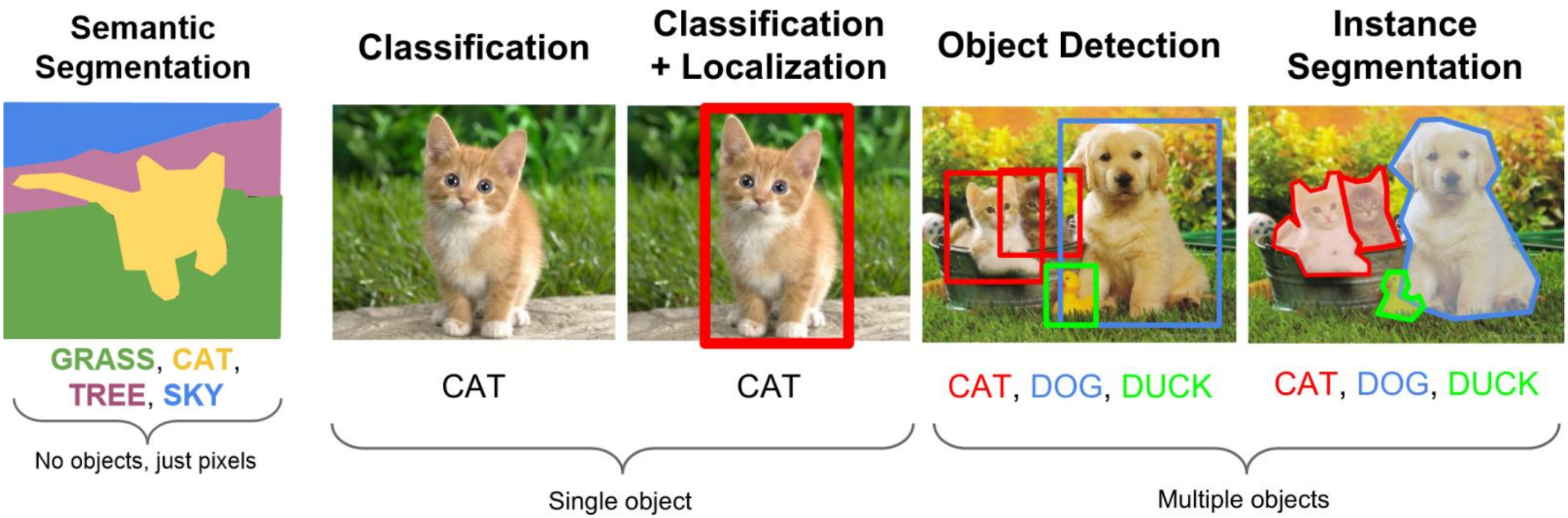
from @teenybiscuit



What can you do with computer vision?

Microsoft

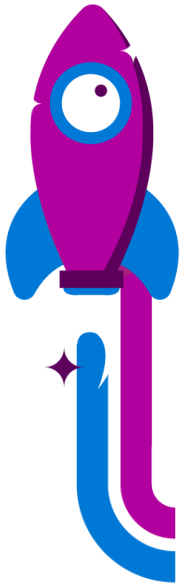
DISCRIMINATIVE ARCHITECTURES



Microsoft

GENERATIVE ARCHITECTURES

Microsoft

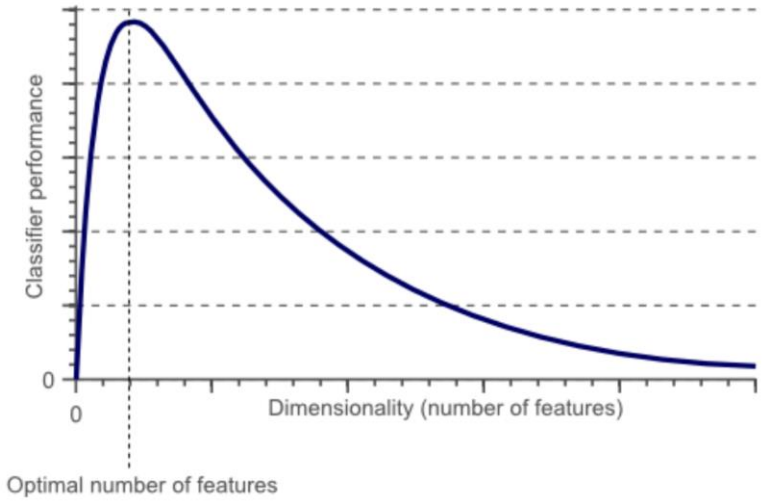


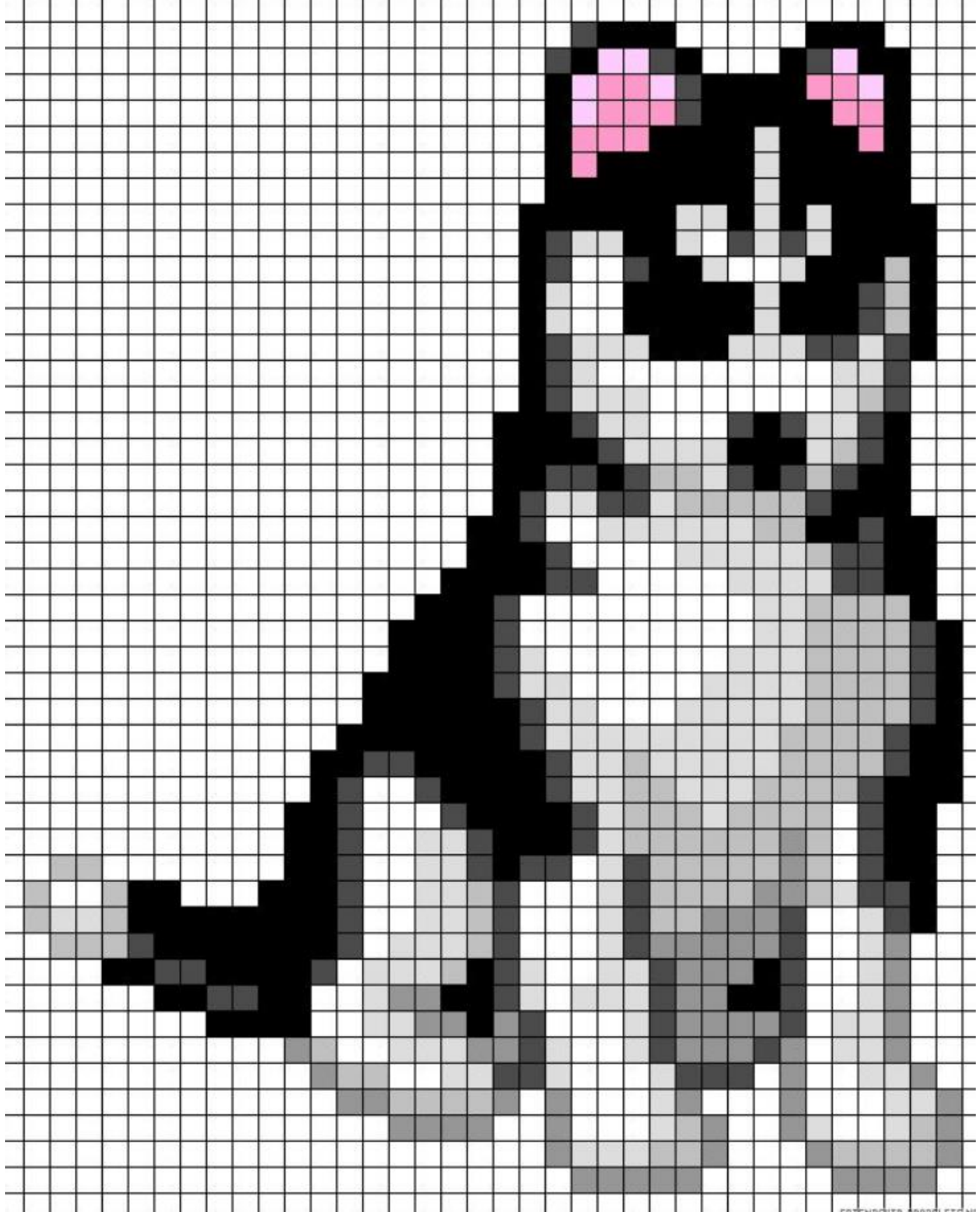
Vision before CNNs?

Microsoft

PIXEL-WISE LEARNING

- Not “translation” invariant
- Not scale invariant
- Not rotational invariant
- Curse of dimensionality



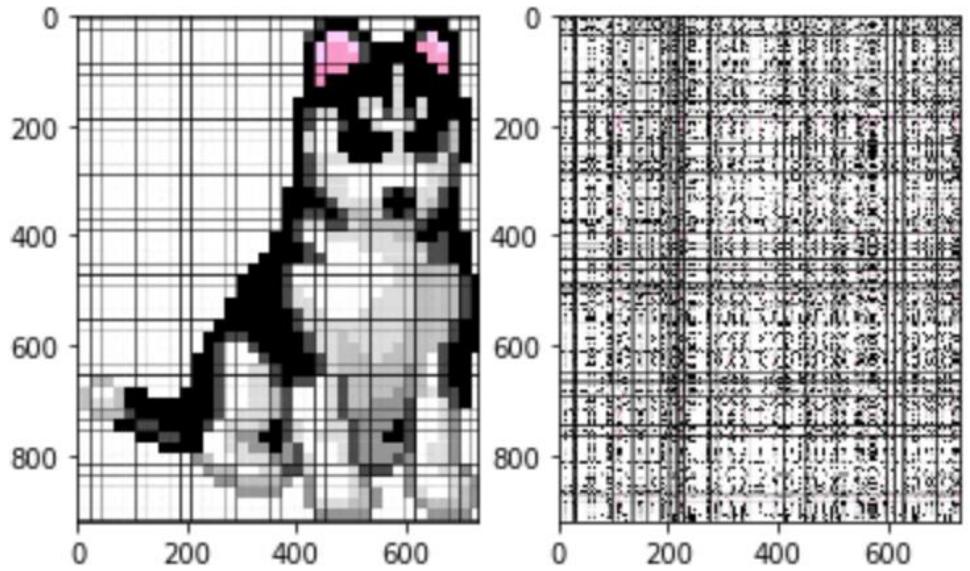


Microsoft

WHAT A CLASSICAL ALGORITHM SEES (1)

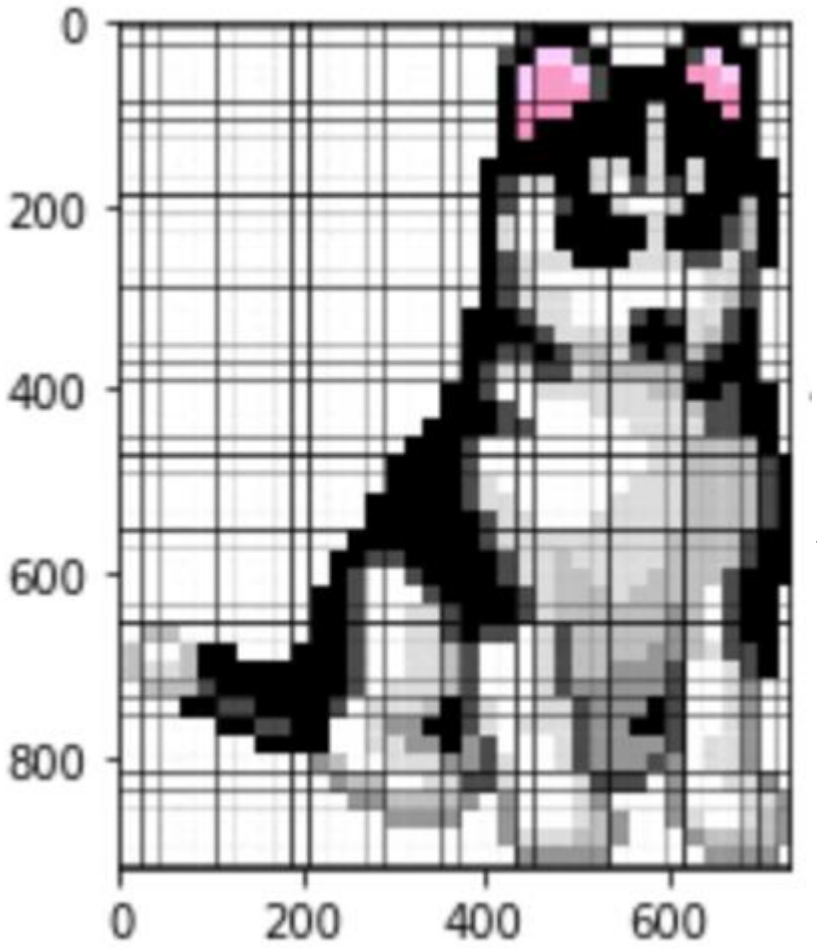
```
img=mpimg.imread('dog.png')

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

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

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



Microsoft

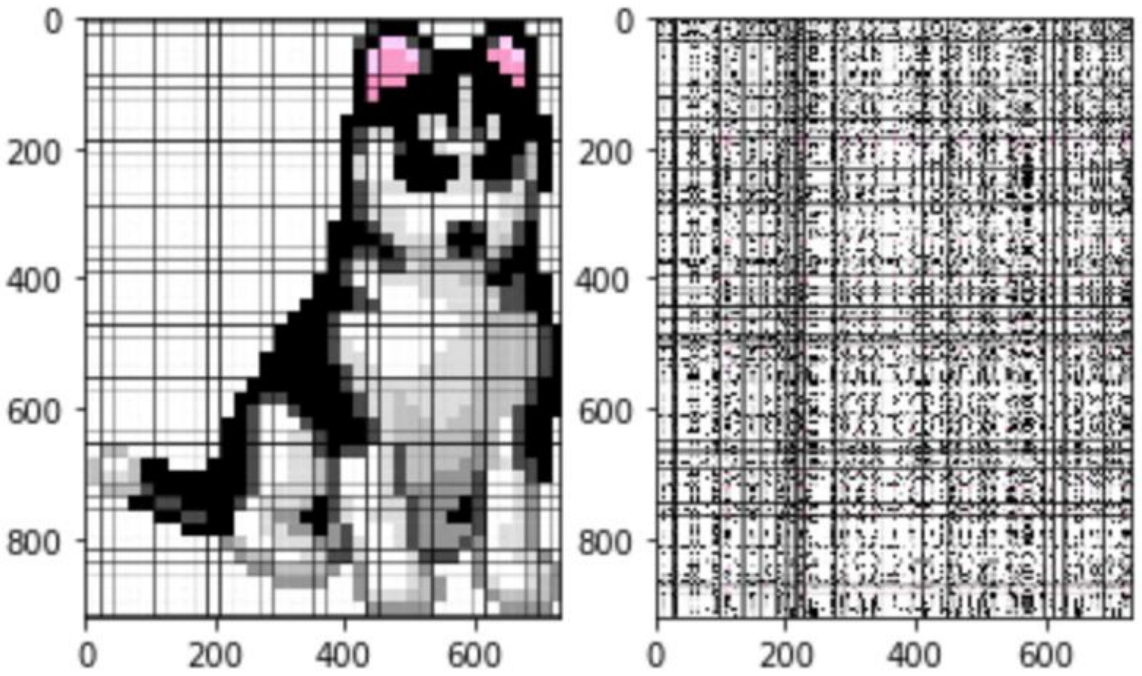
WHAT A CLASSICAL ALGORITHM SEES (2)



Flatten!

WHAT A CLASSICAL ALGORITHM SEES (3)

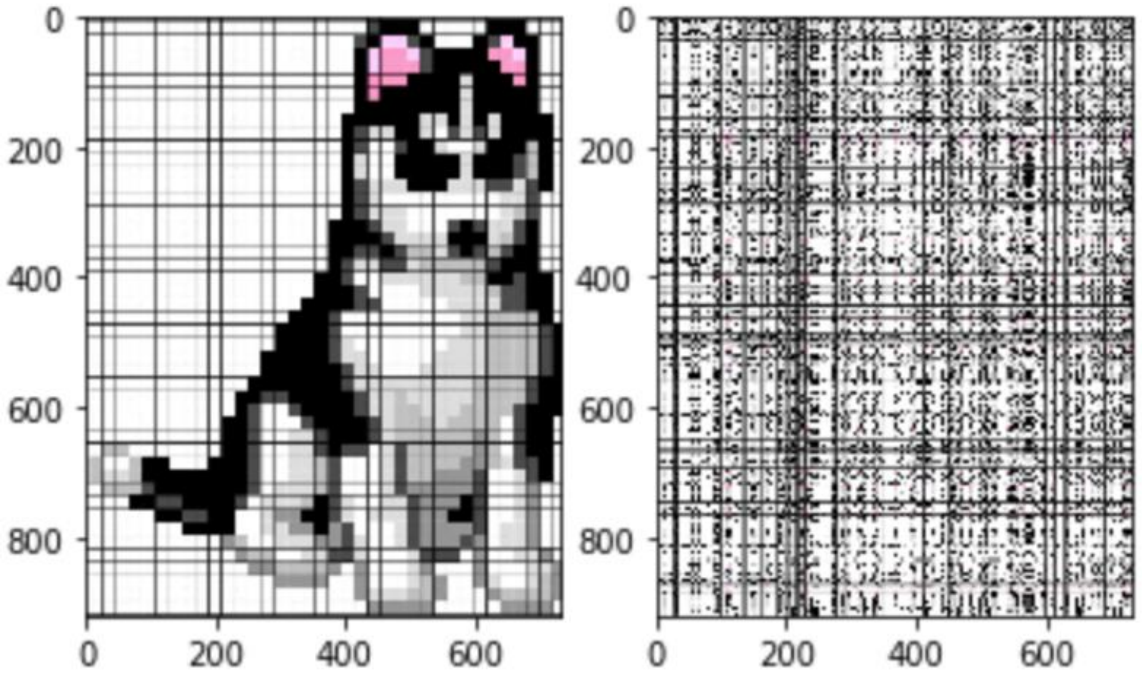
- There are two key types of information in an image;
  - the pixel values themselves and …
  - **their spatial relationships/context**



Microsoft

WHAT A CLASSICAL ALGORITHM SEES (3)

- When you shuffle the pixels of an image, **all the meaning is lost** but the statistical **distribution remains the same**.

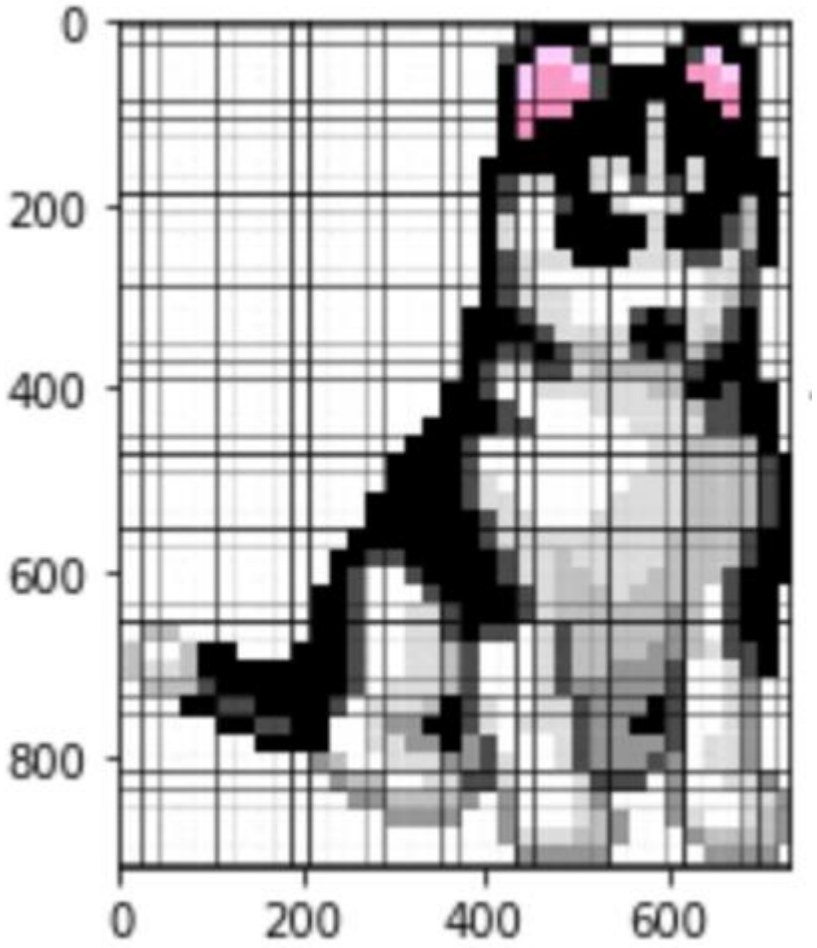


Microsoft

IMAGES ARE…

- Statistically undescriptive
- Sparse
- Huge

So we need a way to transform the information **"out of the space domain"**

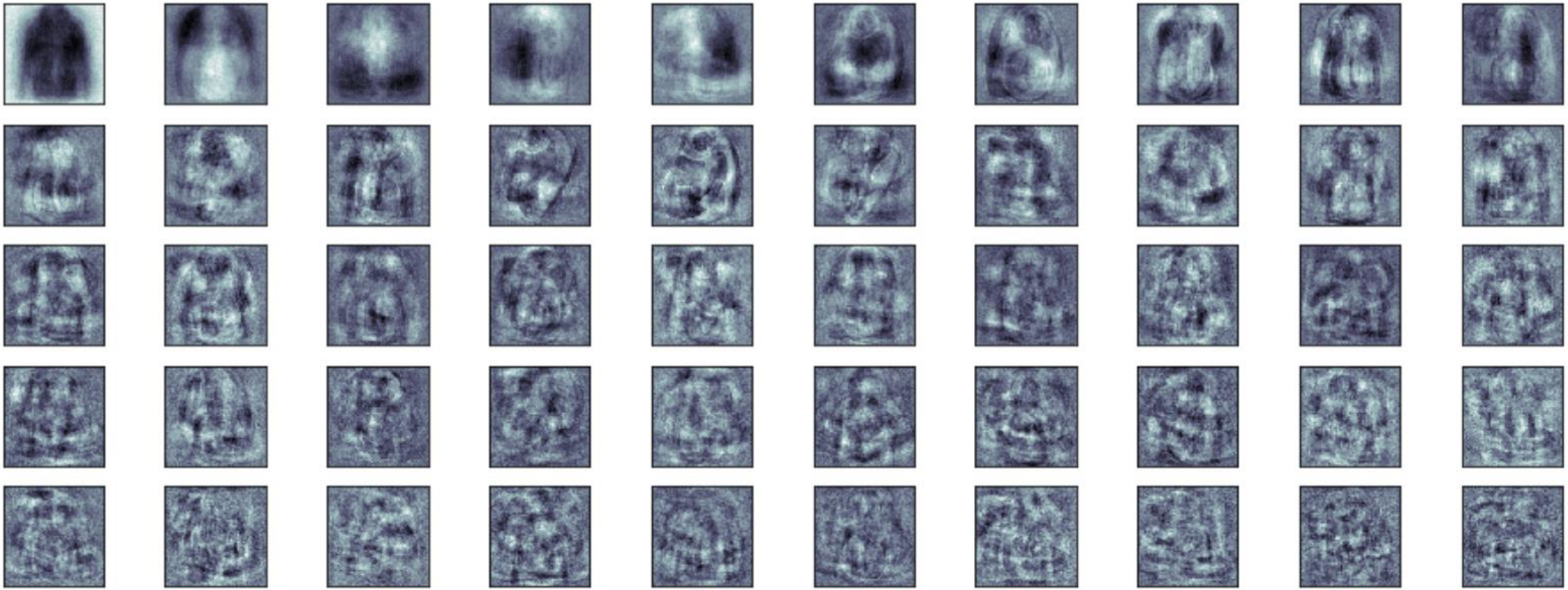


600*800 ~ .5M dimensions!

Microsoft

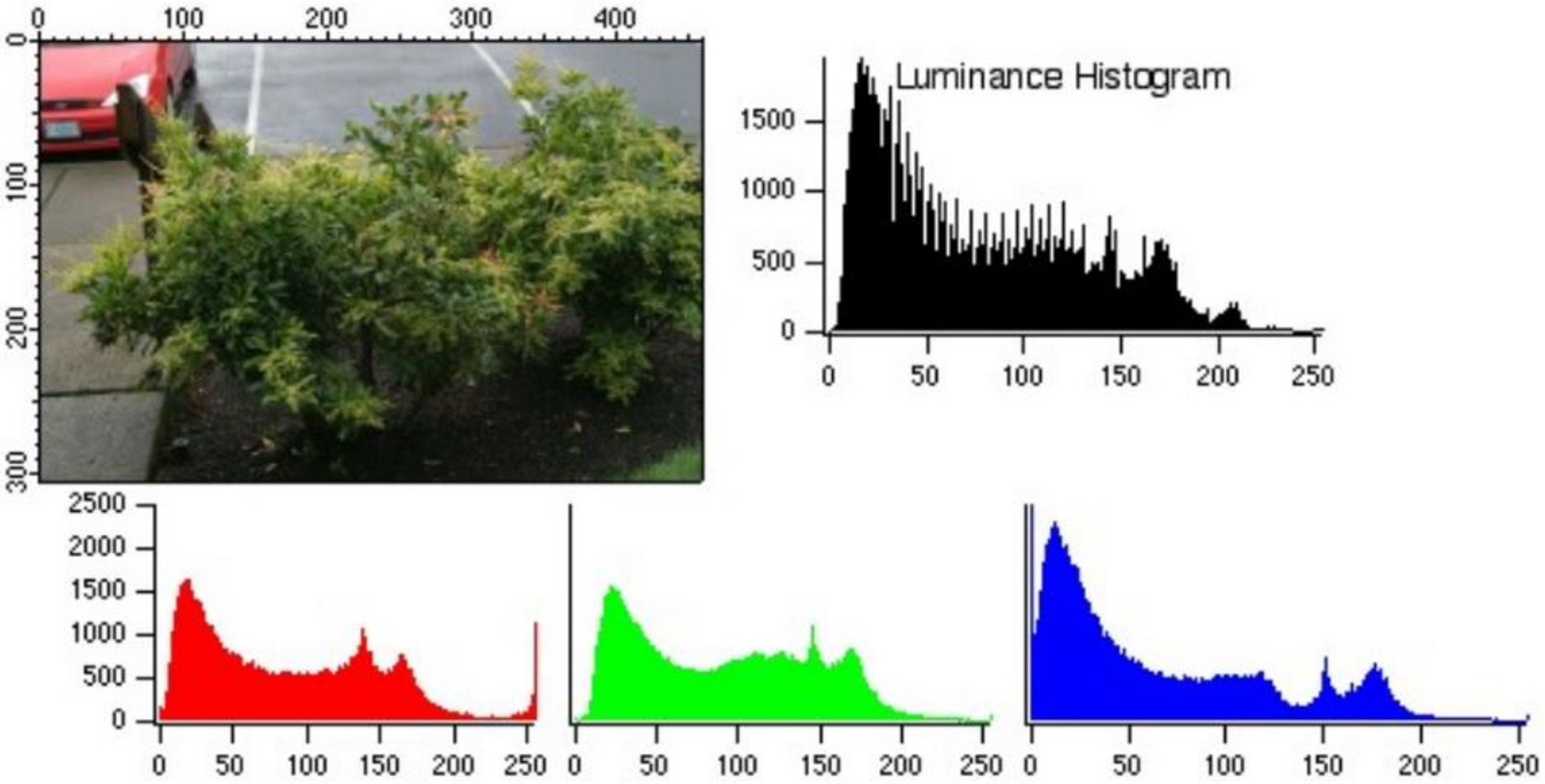
PCA

In [45]:
```
from sklearn import decomposition

def PCA(xtr, xte, n_components=50, flat_size = 128*128):

    pca = decomposition.PCA(n_components=n_components)
    pca.fit(xtr.reshape(xtr.shape[0],flat_size))

    xtr_pca = pca.transform(xtr.reshape(xtr.shape[0],flat_size))
    xte_pca = pca.transform(xte.reshape(xte.shape[0],flat_size))

    return xtr_pca, xte_pca, pca
```



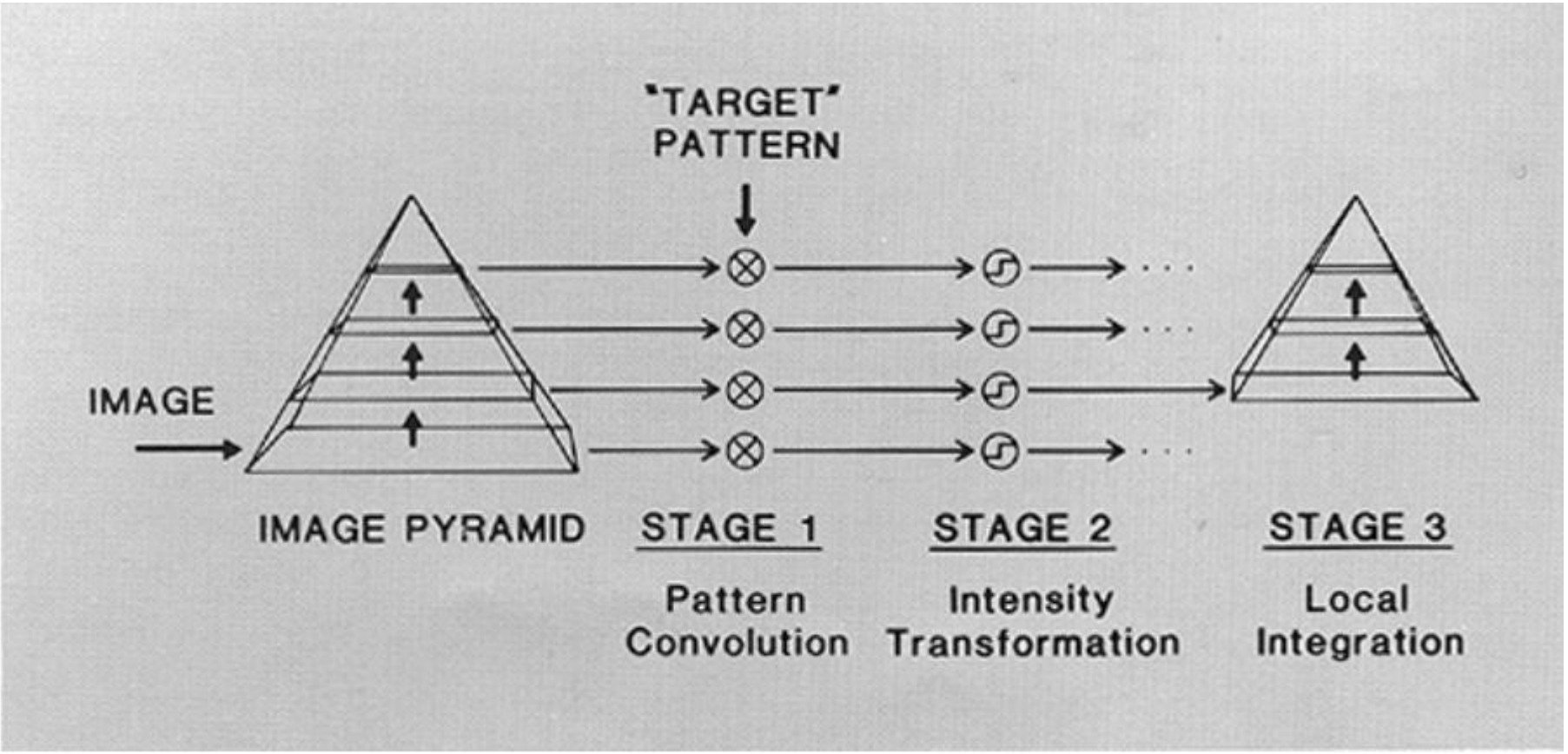
Microsoft

HISTOGRAMS A FEATURES



Microsoft

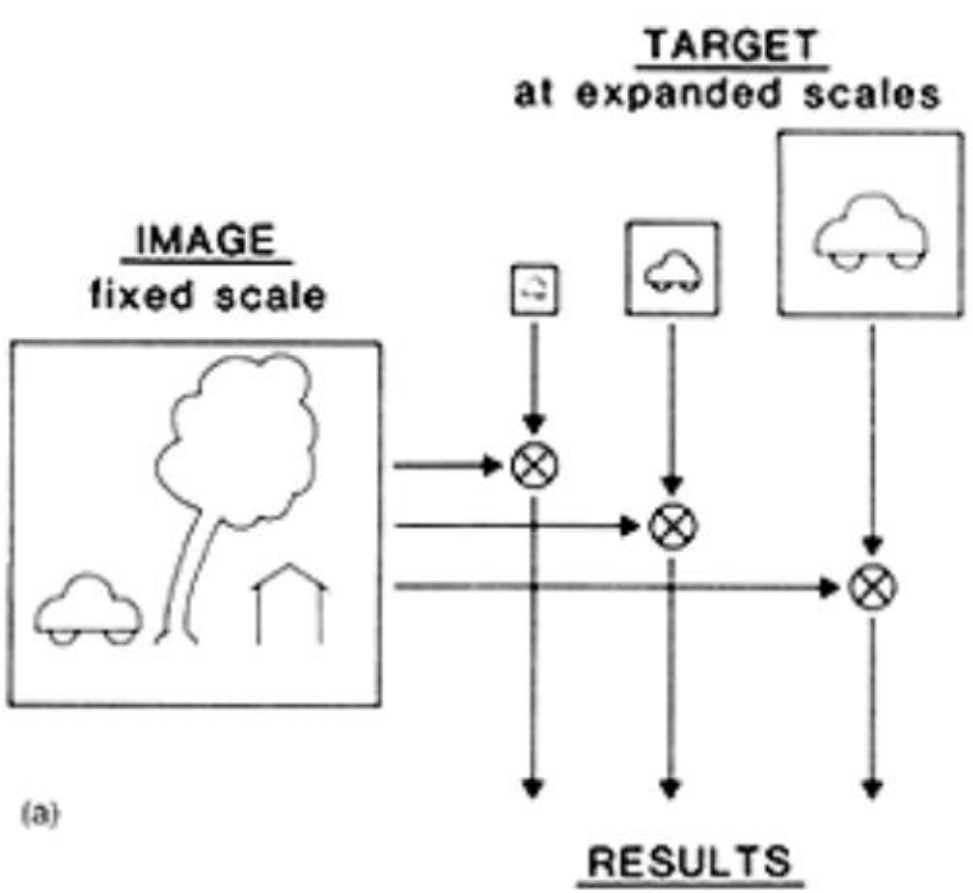
IMAGE PYRAMID (1)

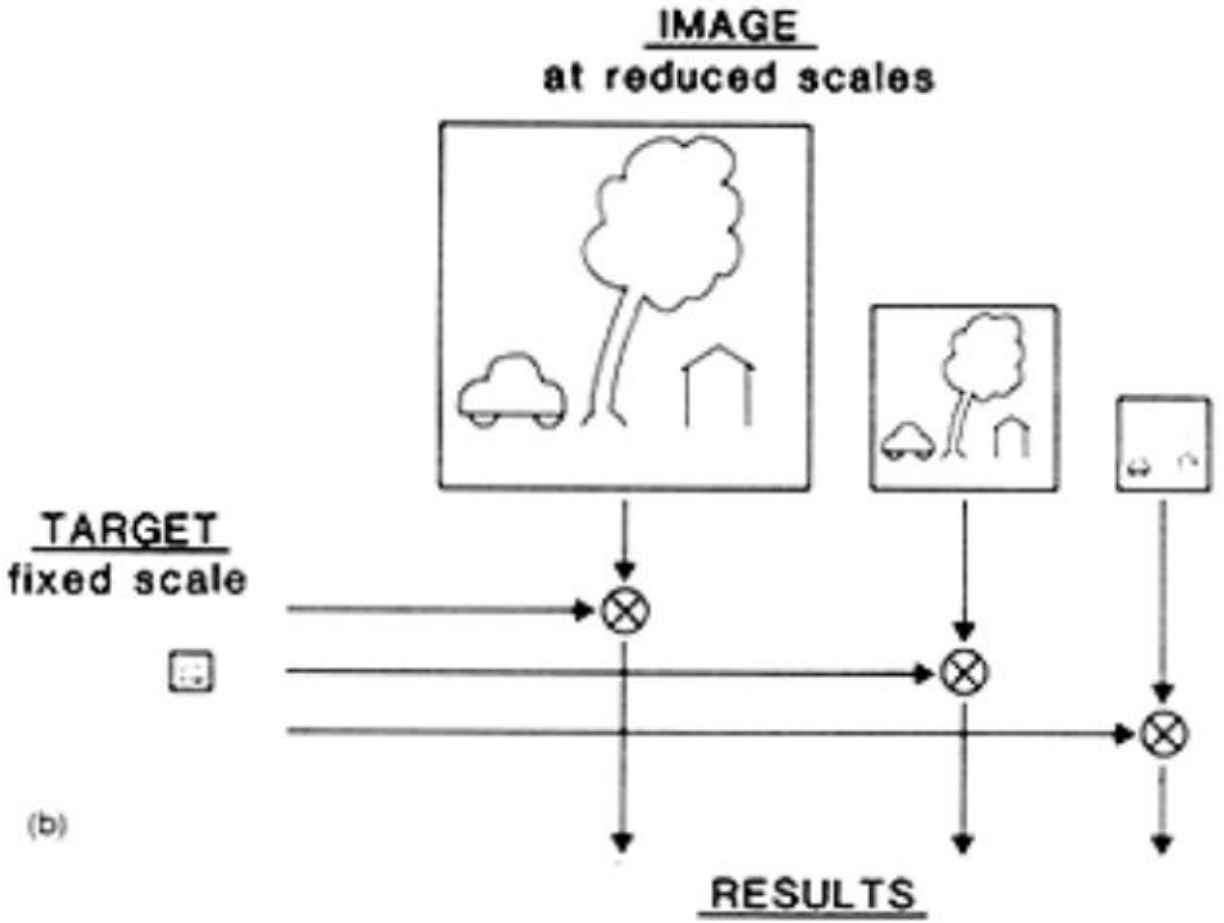


Adelson et al '84

Microsoft

IMAGE PYRAMID (2)

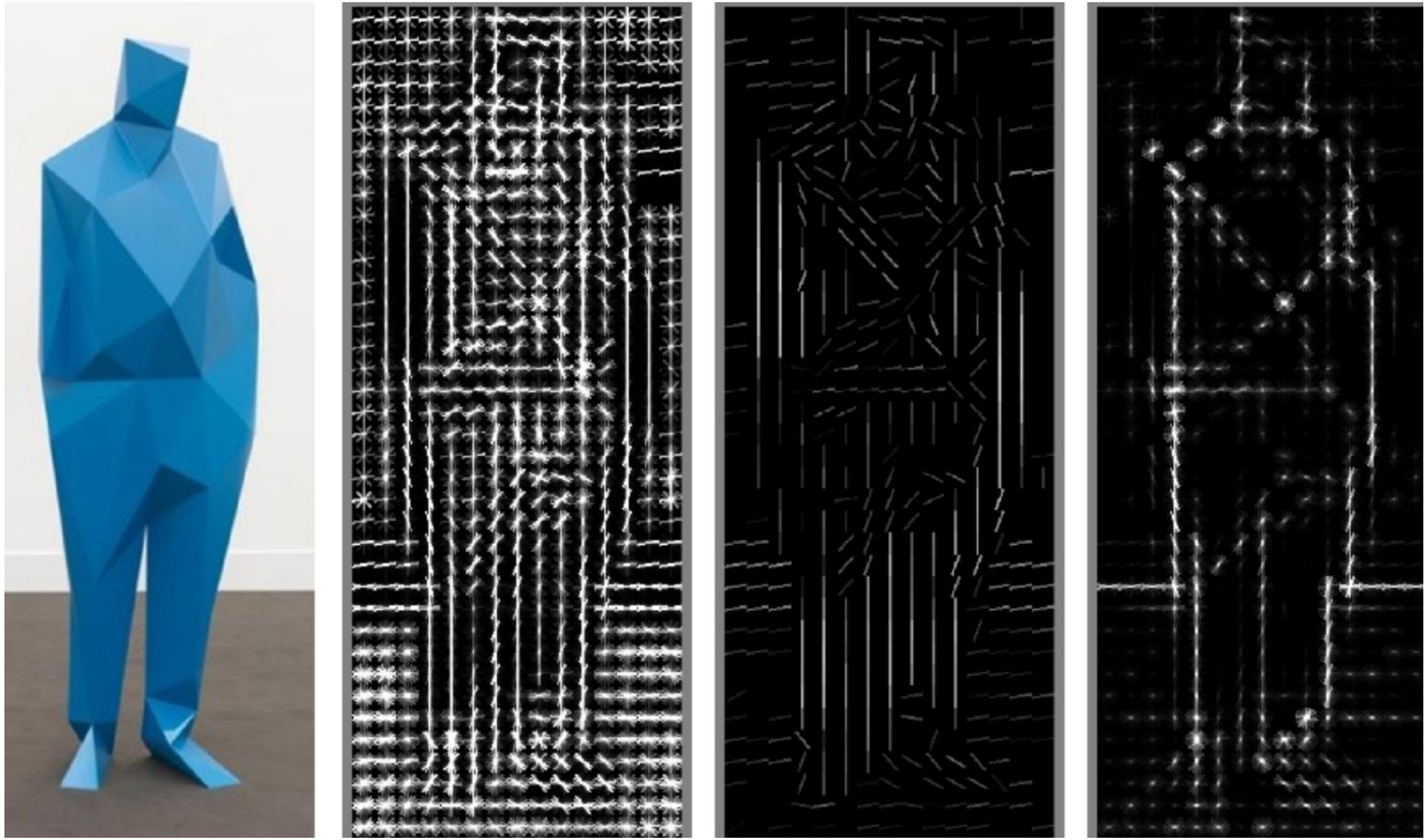




Adelson et al ‘84

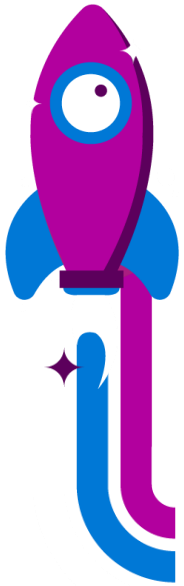
Microsoft

HISTOGRAMS OF GRADIENTS (HOG)



Microsoft

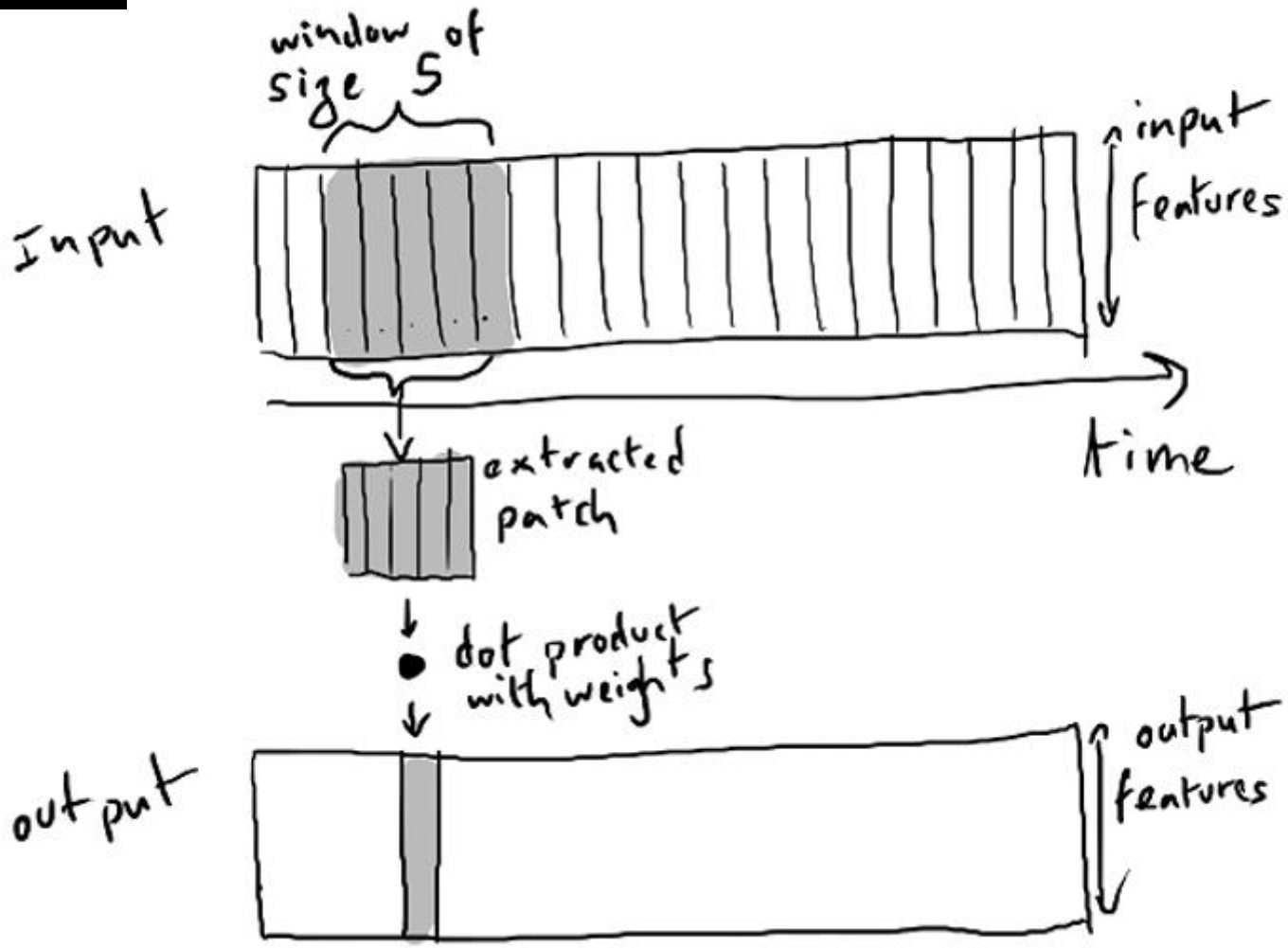
EDGE DETECTION (WITH CONVOLUTION)



How do convolutional neural networks work?

Microsoft

CONVOLUTION

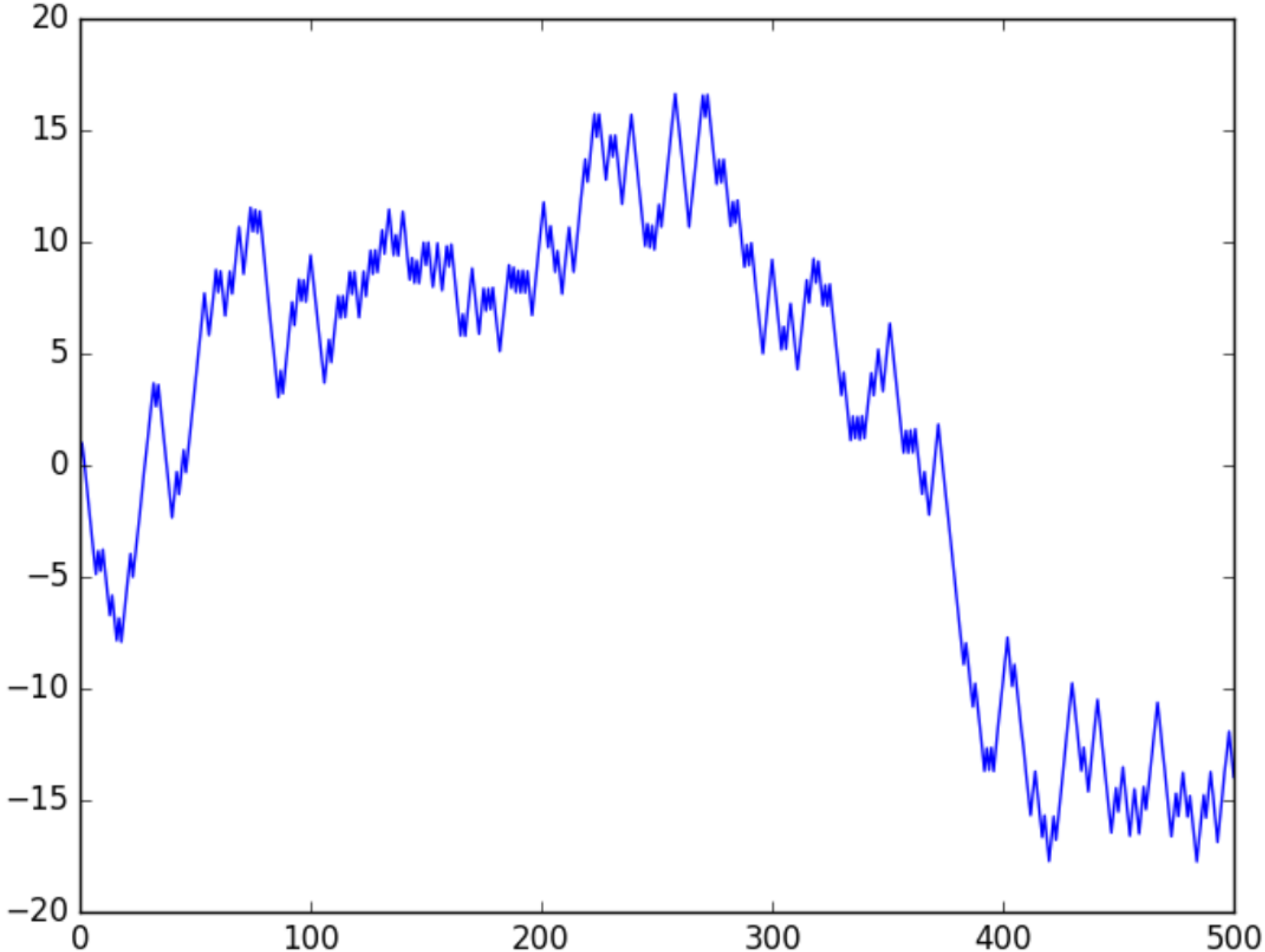


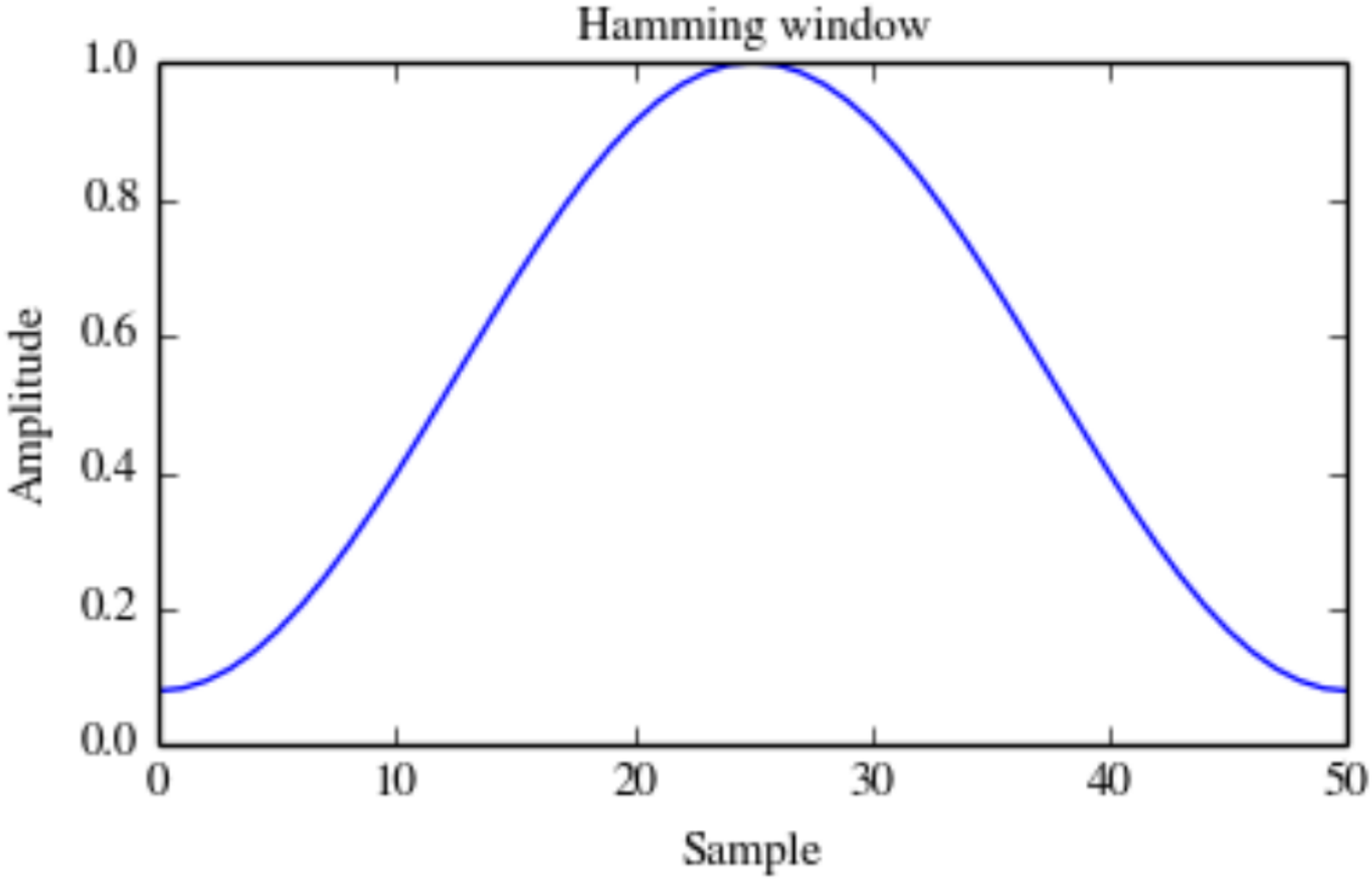
© Francois Chollet

Microsoft

Microsoft

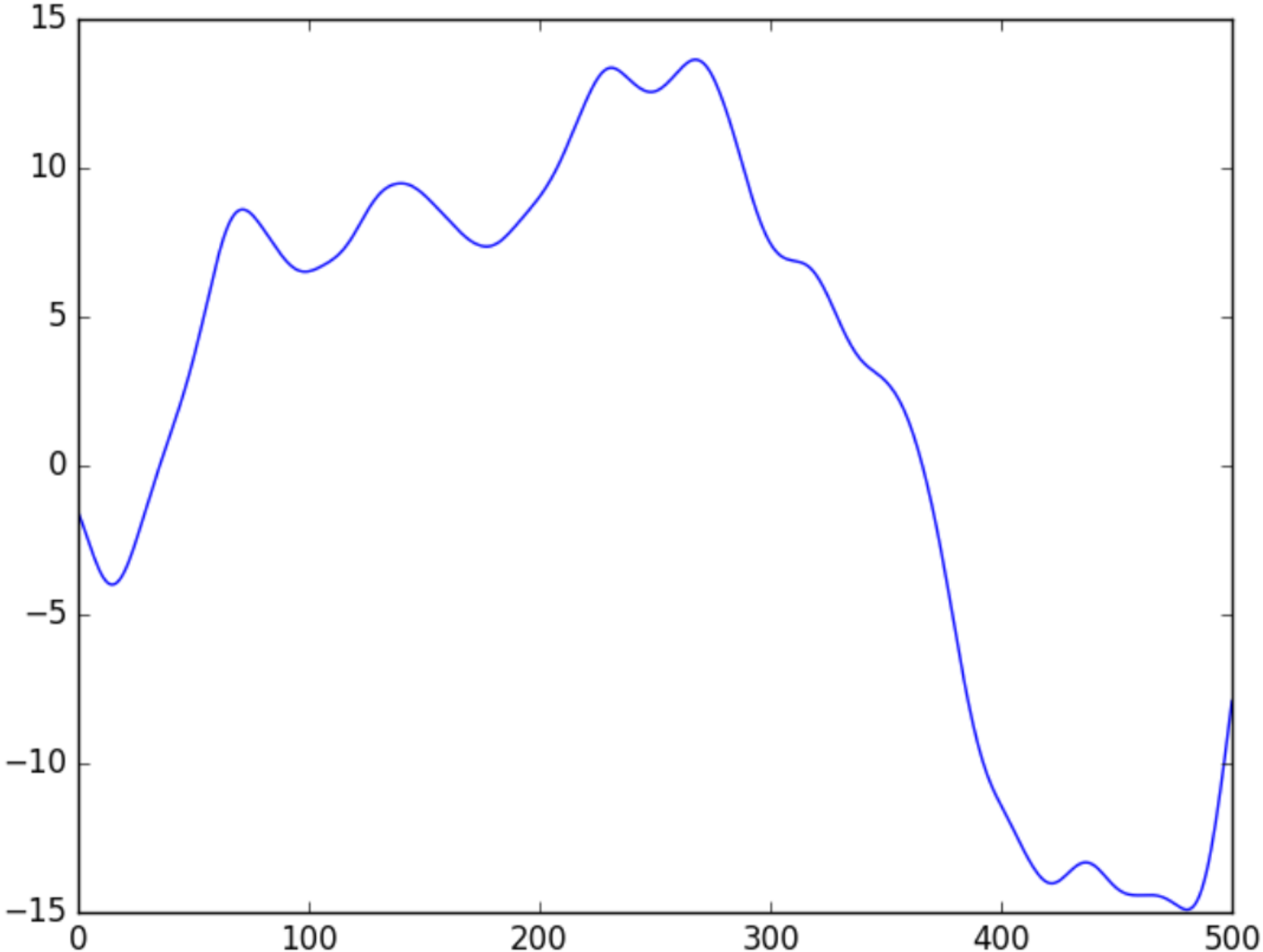
```
x = [0]

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

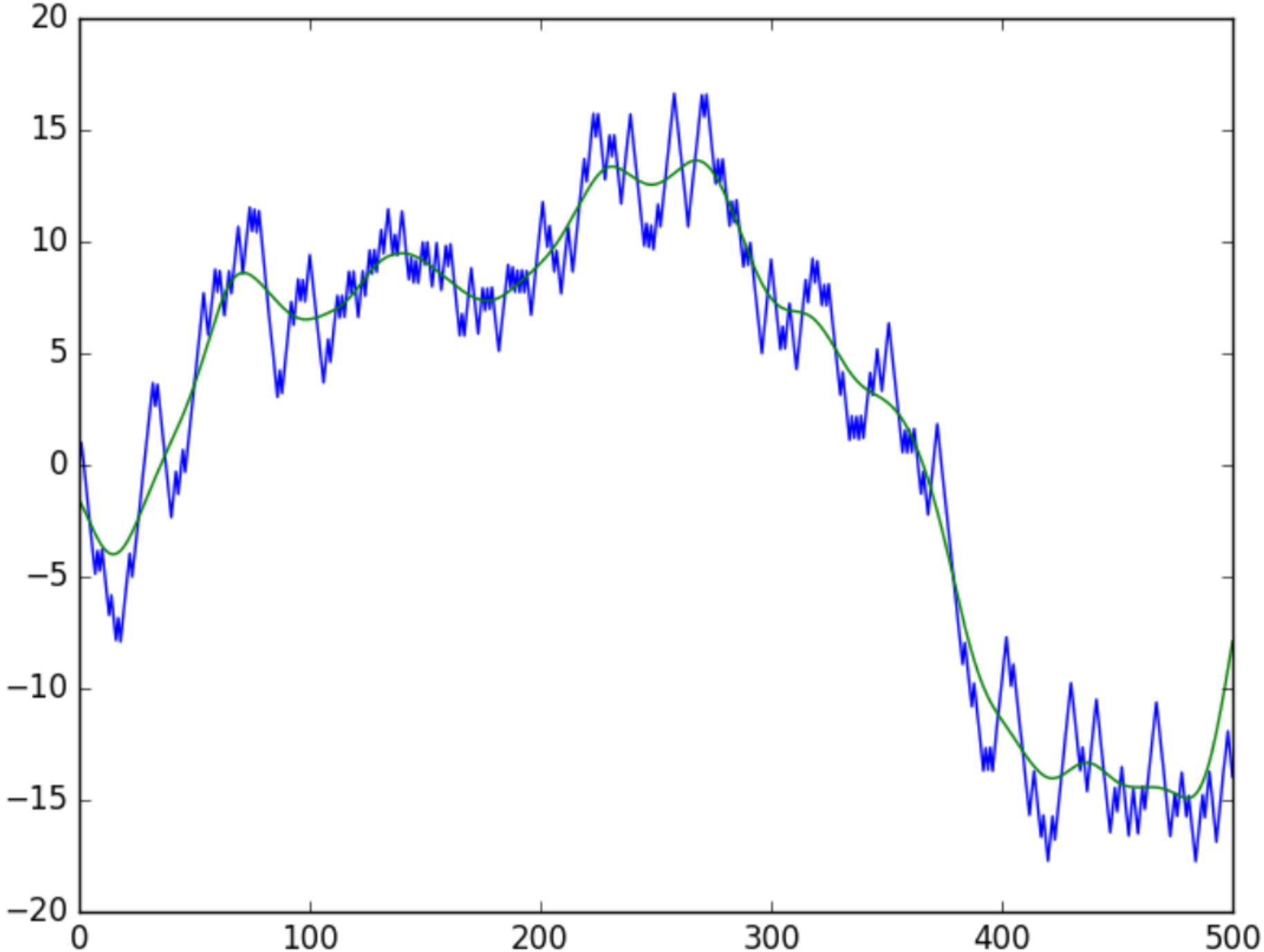

20
15
10
5
0
−5
−10
−15
−20
0
100
200
300
400
500



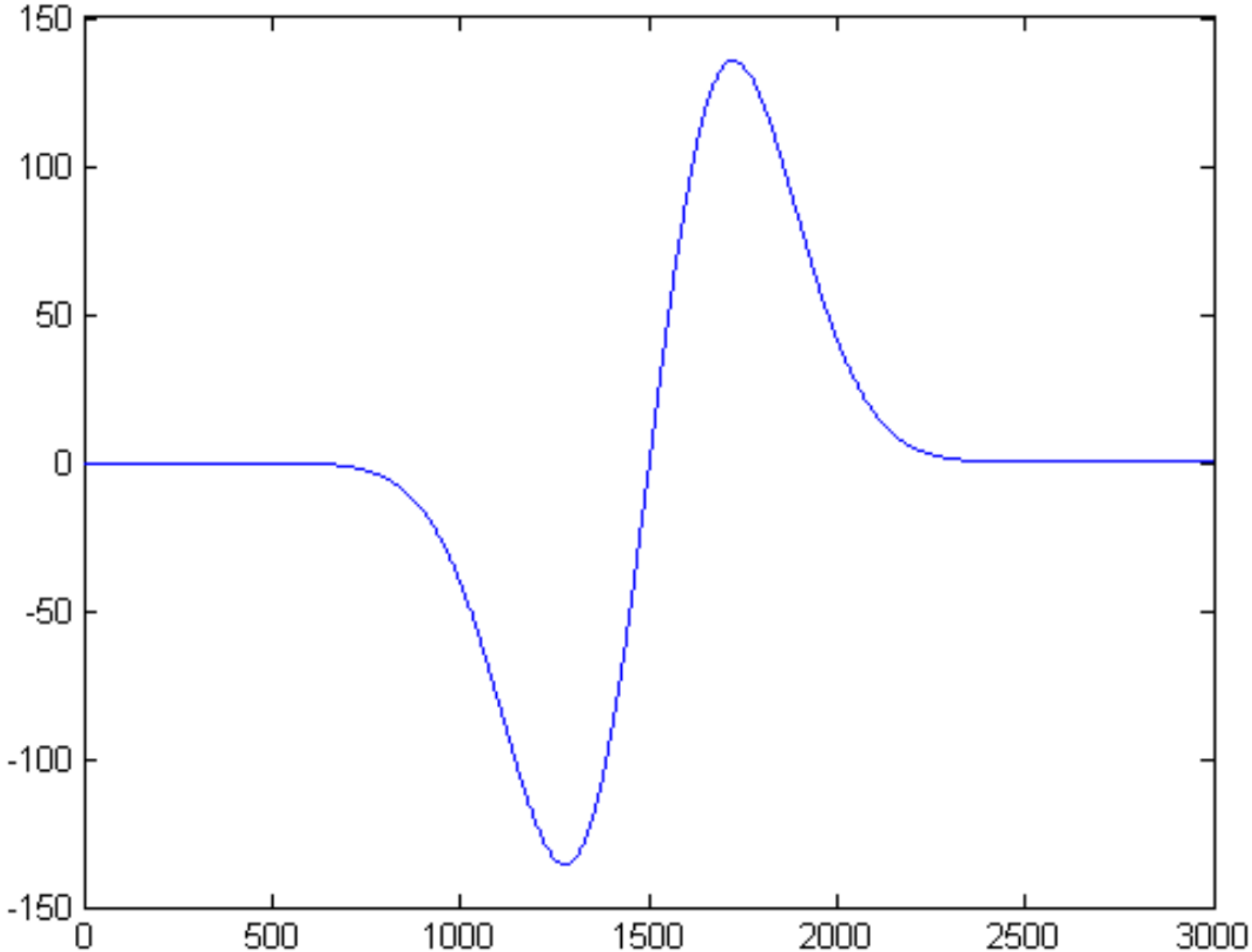
$$w(n) = 0.54 - 0.46\cos\left(2\pi\frac{n}{N}\right), \quad 0 \le n \le N.$$

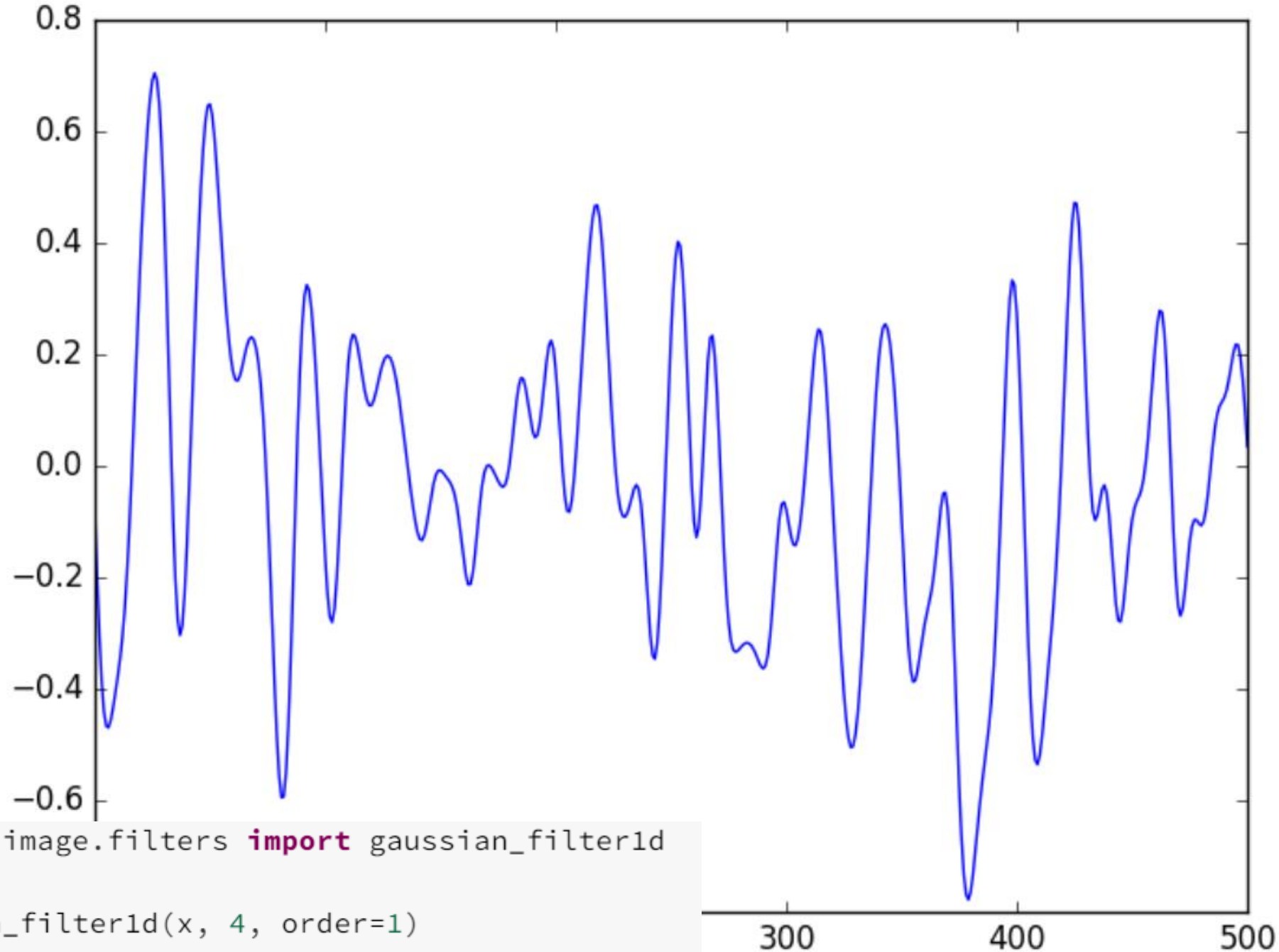
```
from scipy import signal

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



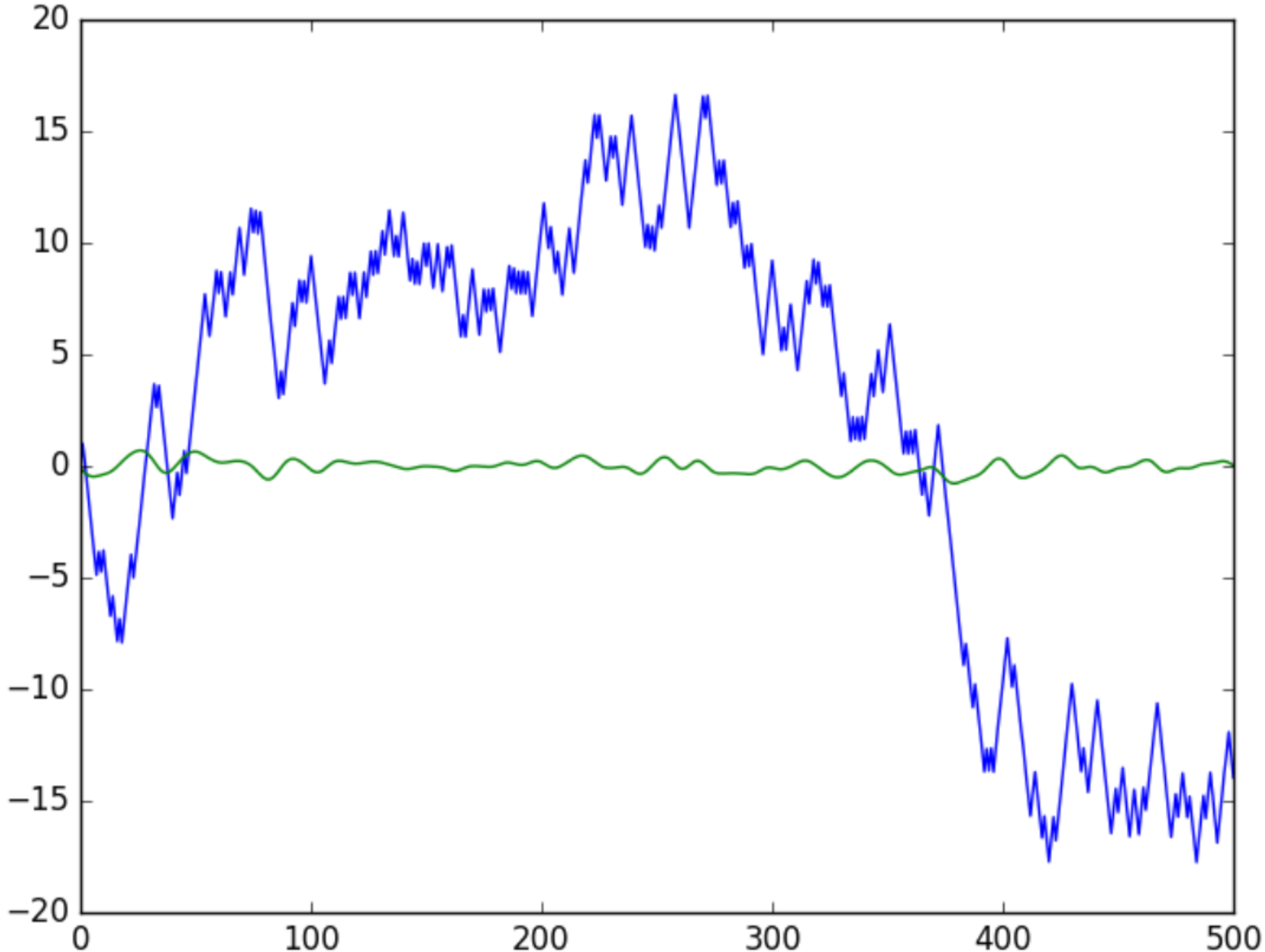


Microsoft






```python
from scipy.ndimage.filters import gaussian_filter1d

x2 = gaussian_filter1d(x, 4, order=1)
```

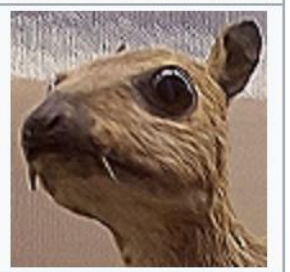


Microsoft

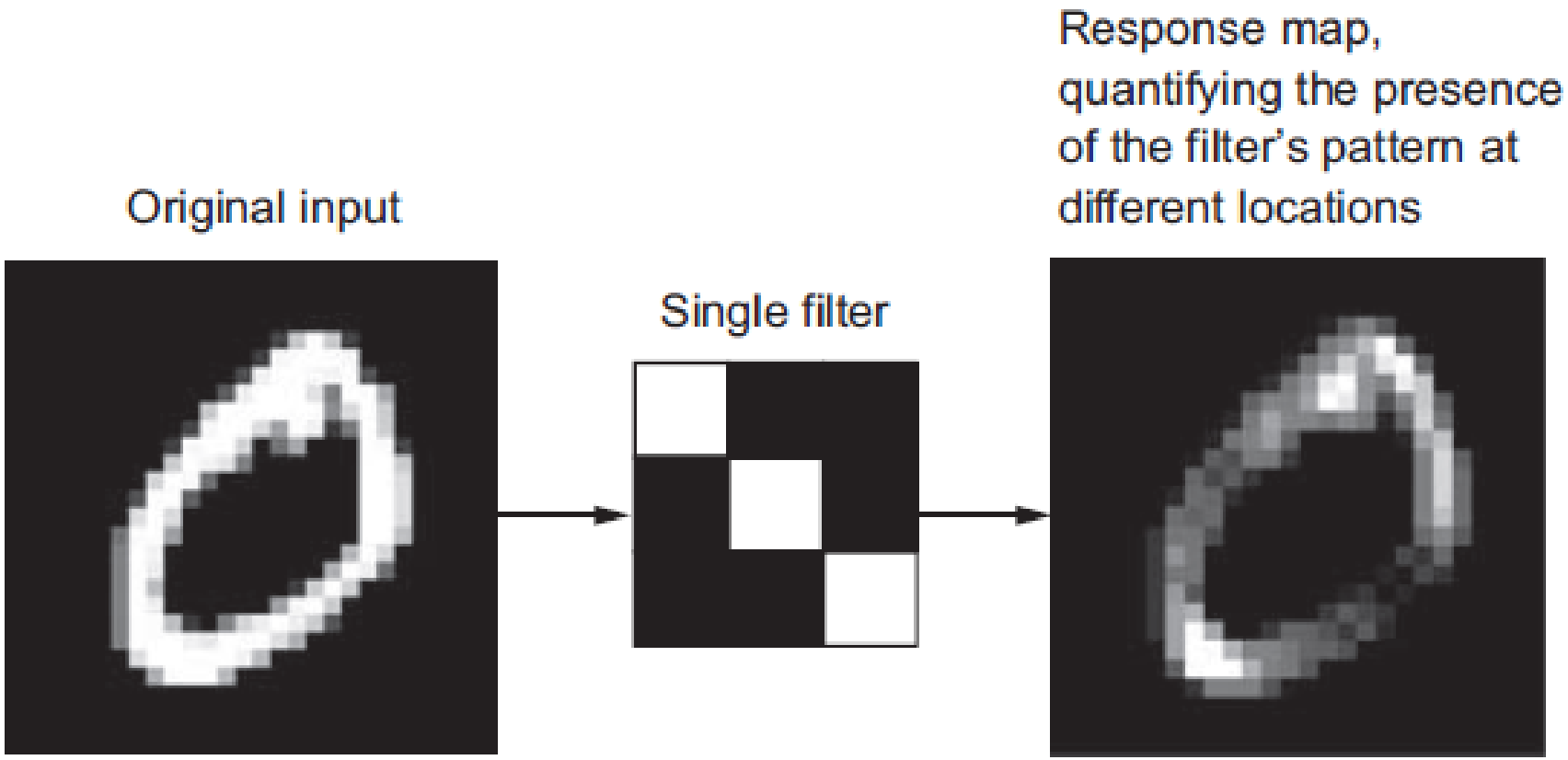
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

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)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

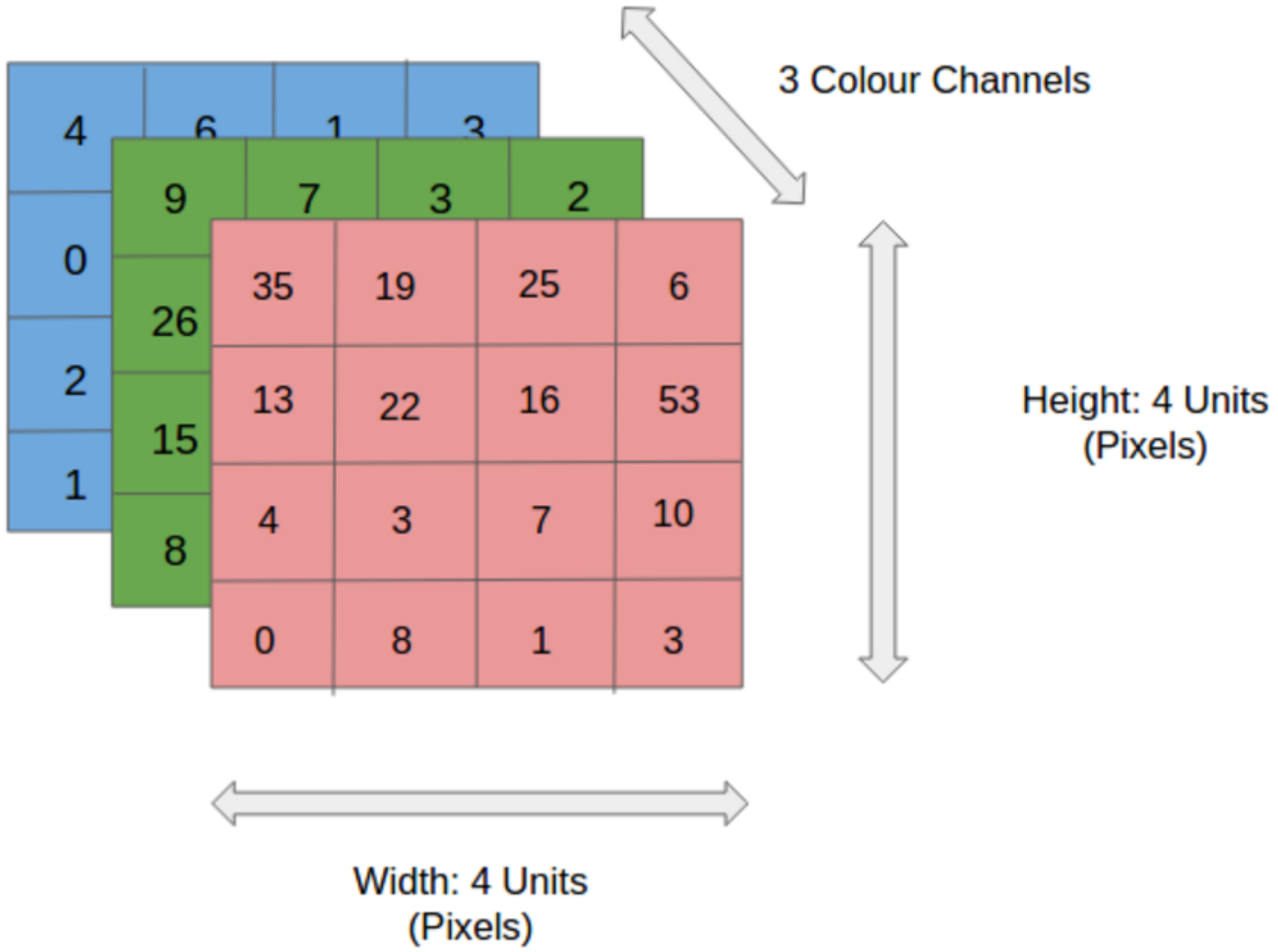
DEMO

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
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	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Microsoft



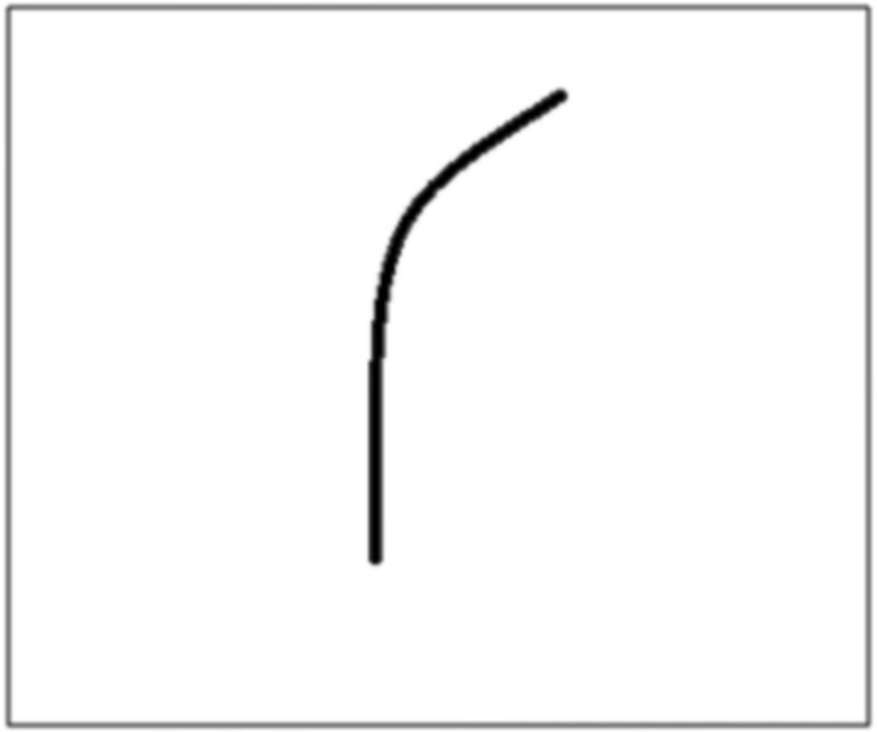
PREPARE DATASET OF IMAGES



Microsoft

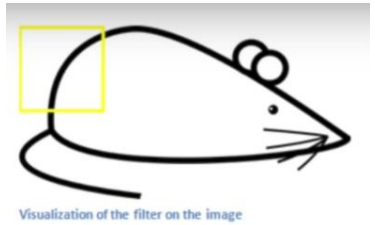
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

CONVOLUTION FILTER



Visualization of the filter on the image

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

Visualization of the receptive field

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	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 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

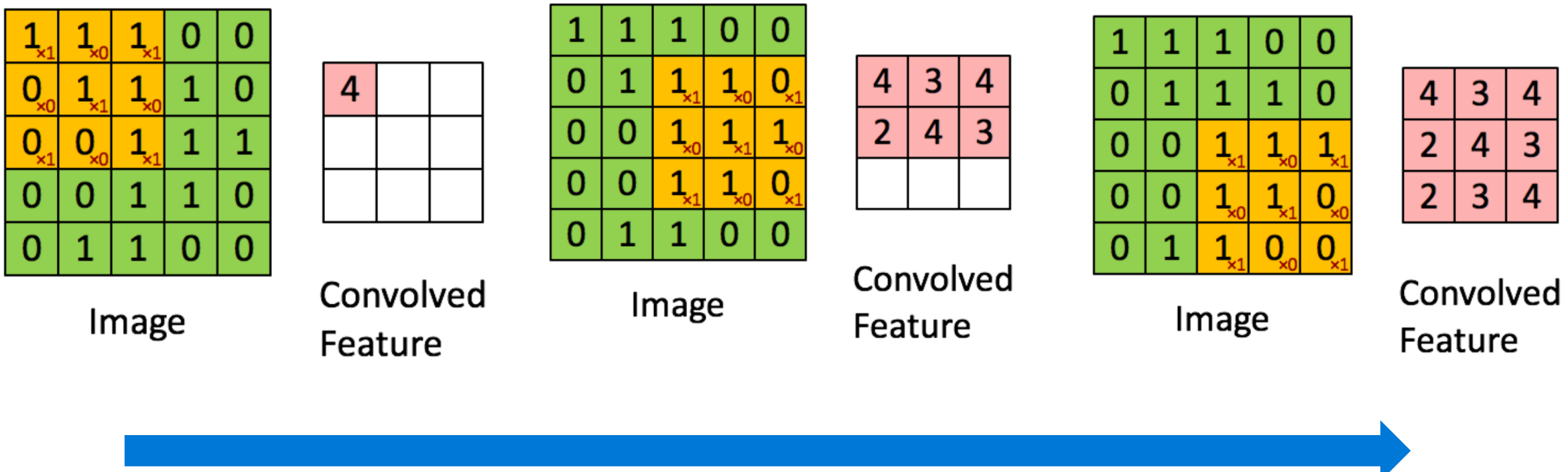
Pixel representation of receptive field

*

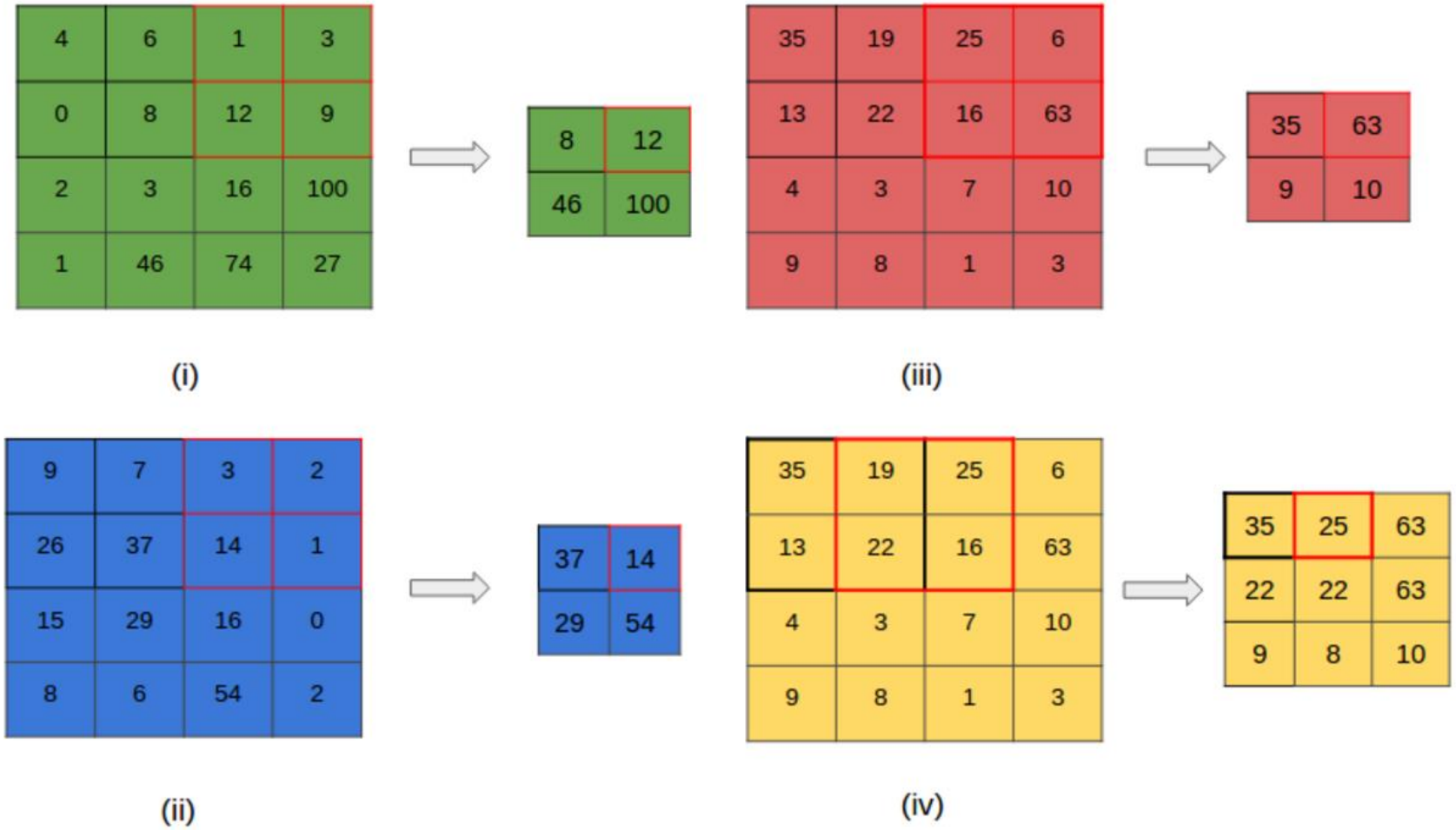
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

CONVOLUTION FILTER NO MATCH

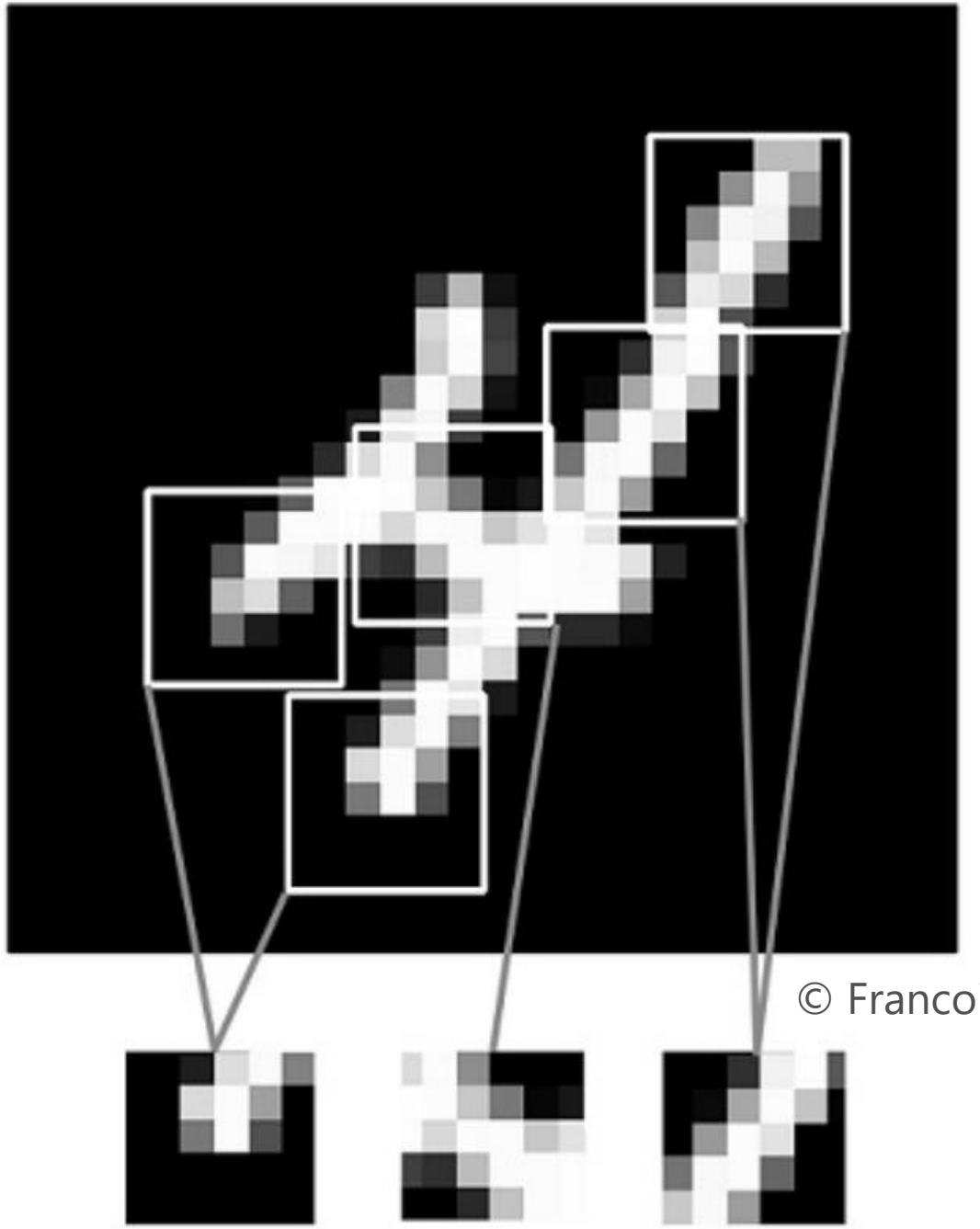


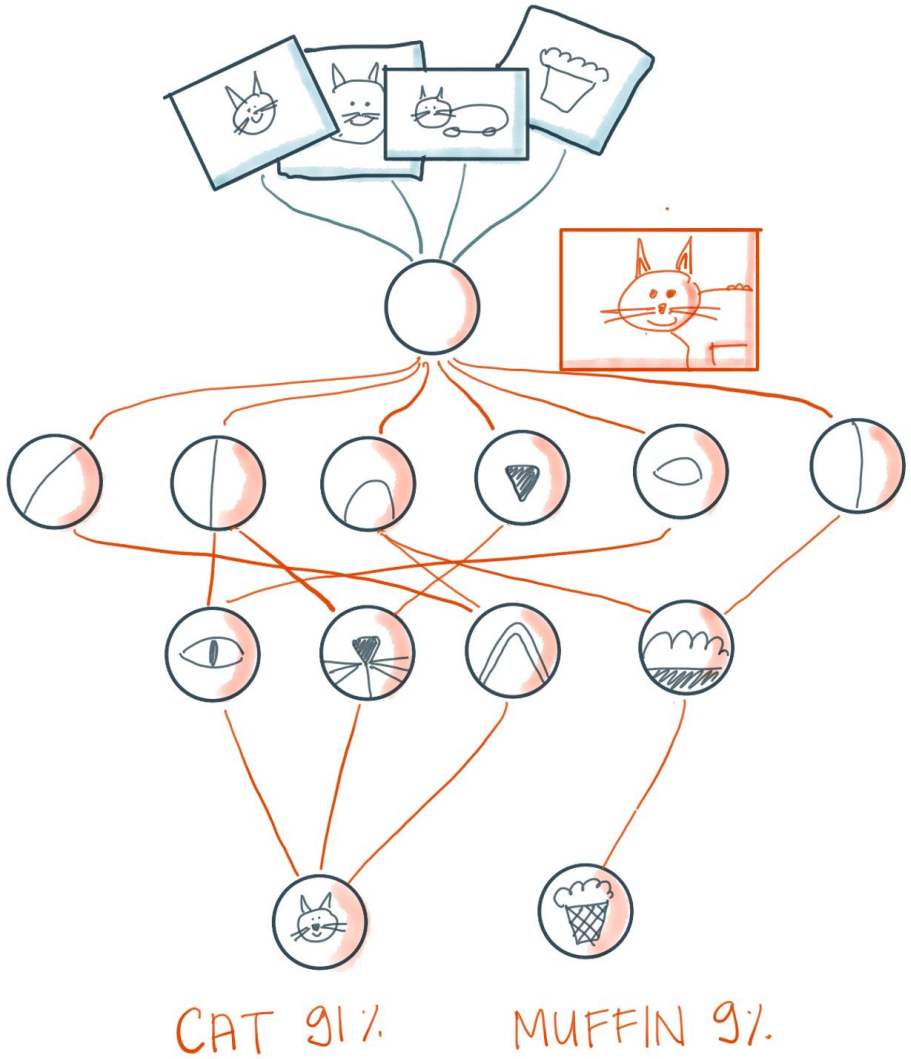
CONVOLUTION


4 6 1 3
0 8 12 9
2 3 16 100
1 46 74 27
8 12
46 100
(i)
35 19 25 6
13 22 16 63
4 3 7 10
9 8 1 3
35 63
9 10
(iii)
9 7 3 2
26 37 14 1
15 29 16 0
8 6 54 2
37 14
29 54
(ii)
35 19 25 6
13 22 16 63
4 3 7 10
9 8 1 3
35 25 63
22 22 63
9 8 10
(iv)

POOLING

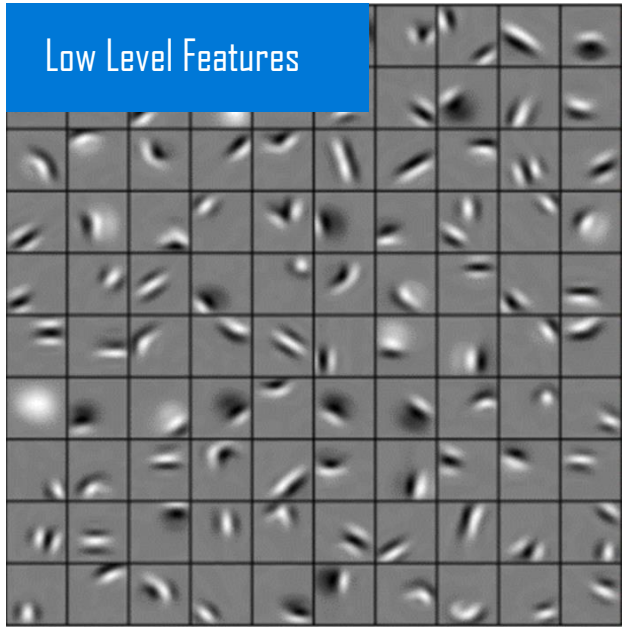
- Translation Invariant
- Spatial hierarchies of patterns
- Sort of scale invariant
- Not rotationally invariant!



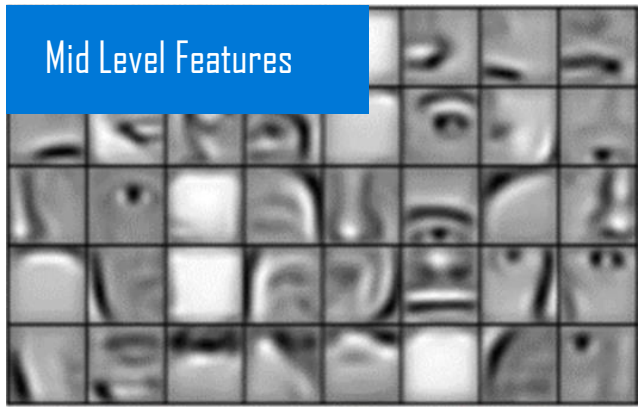


SPATIAL HIERARCHY OF PATTERNS

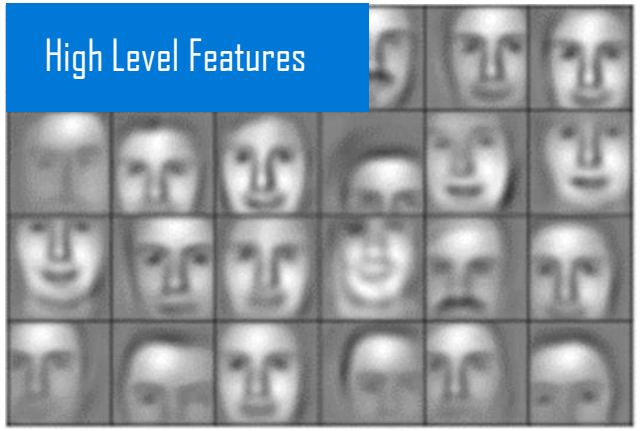
Low Level Features



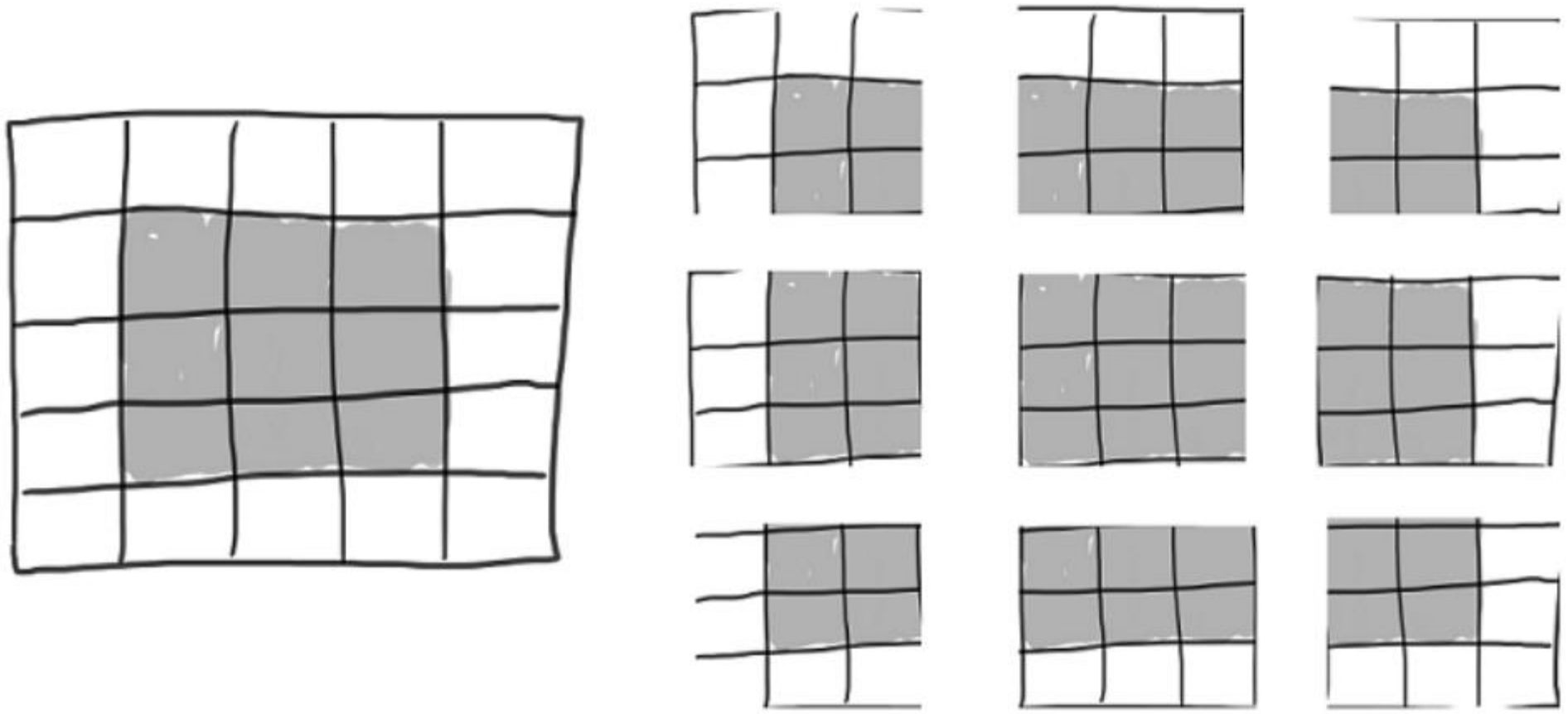
Mid Level Features



High Level Features



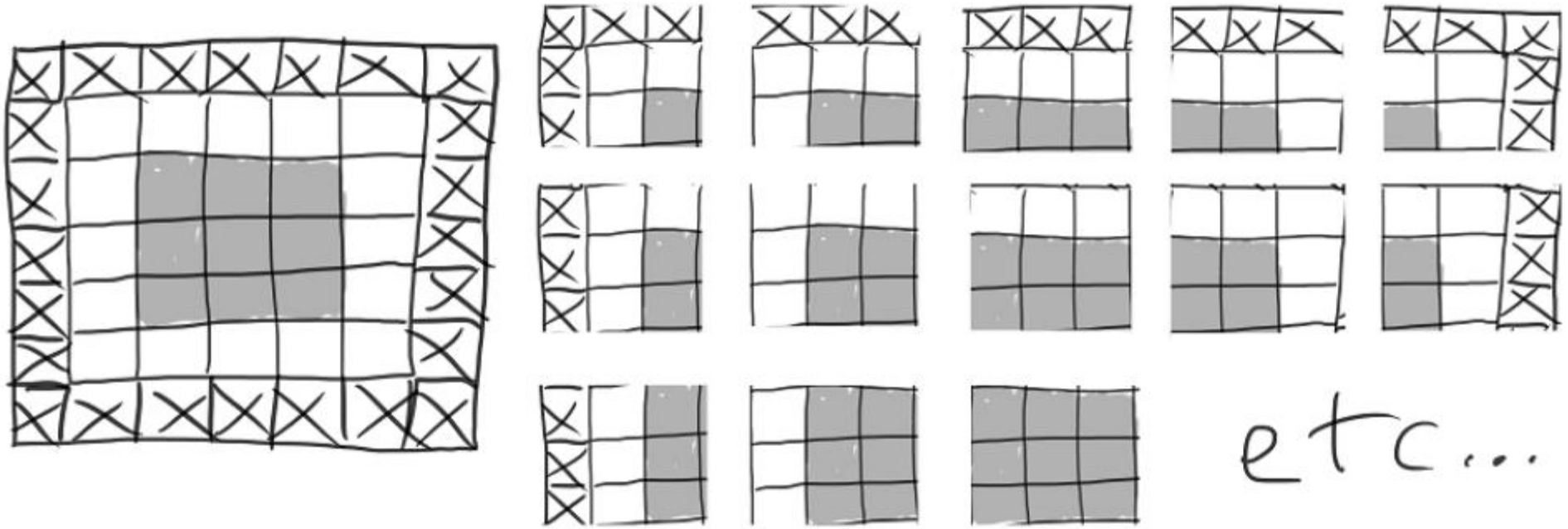
LEARNING REPRESENTATIONS



© Francois Chollet

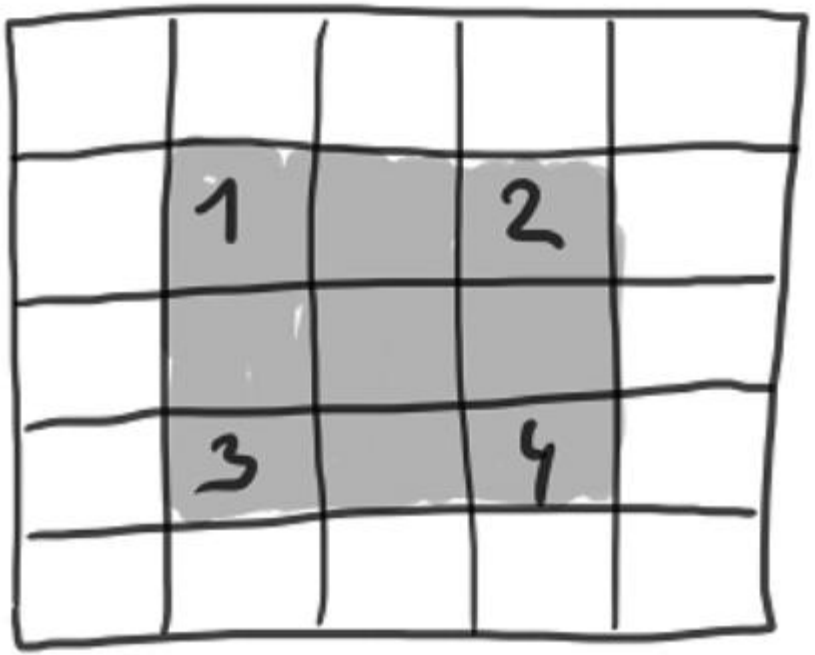
CONVOLUTION EXAMPLE

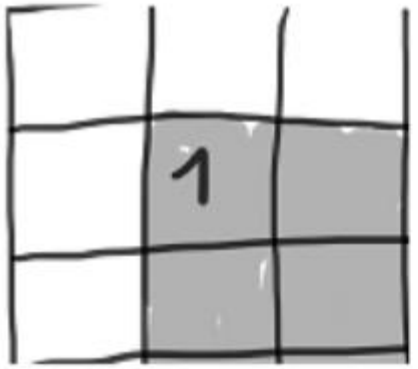
© Francois Chollet

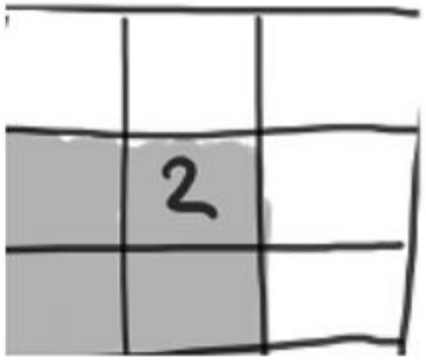


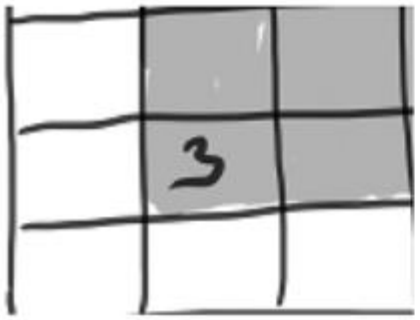
BORDER EFFECTS: 'VALID' VS 'SAME'

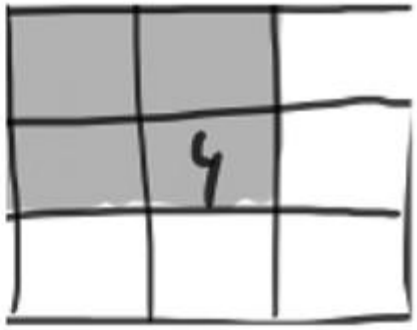
© Francois Chollet





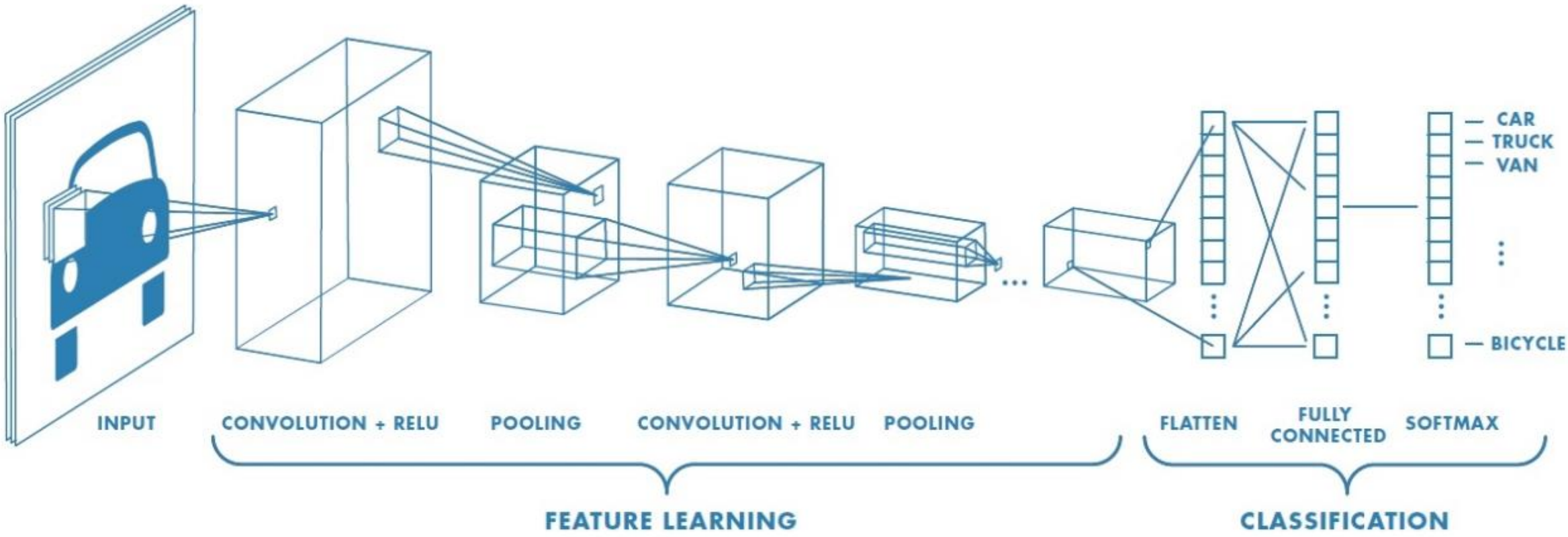


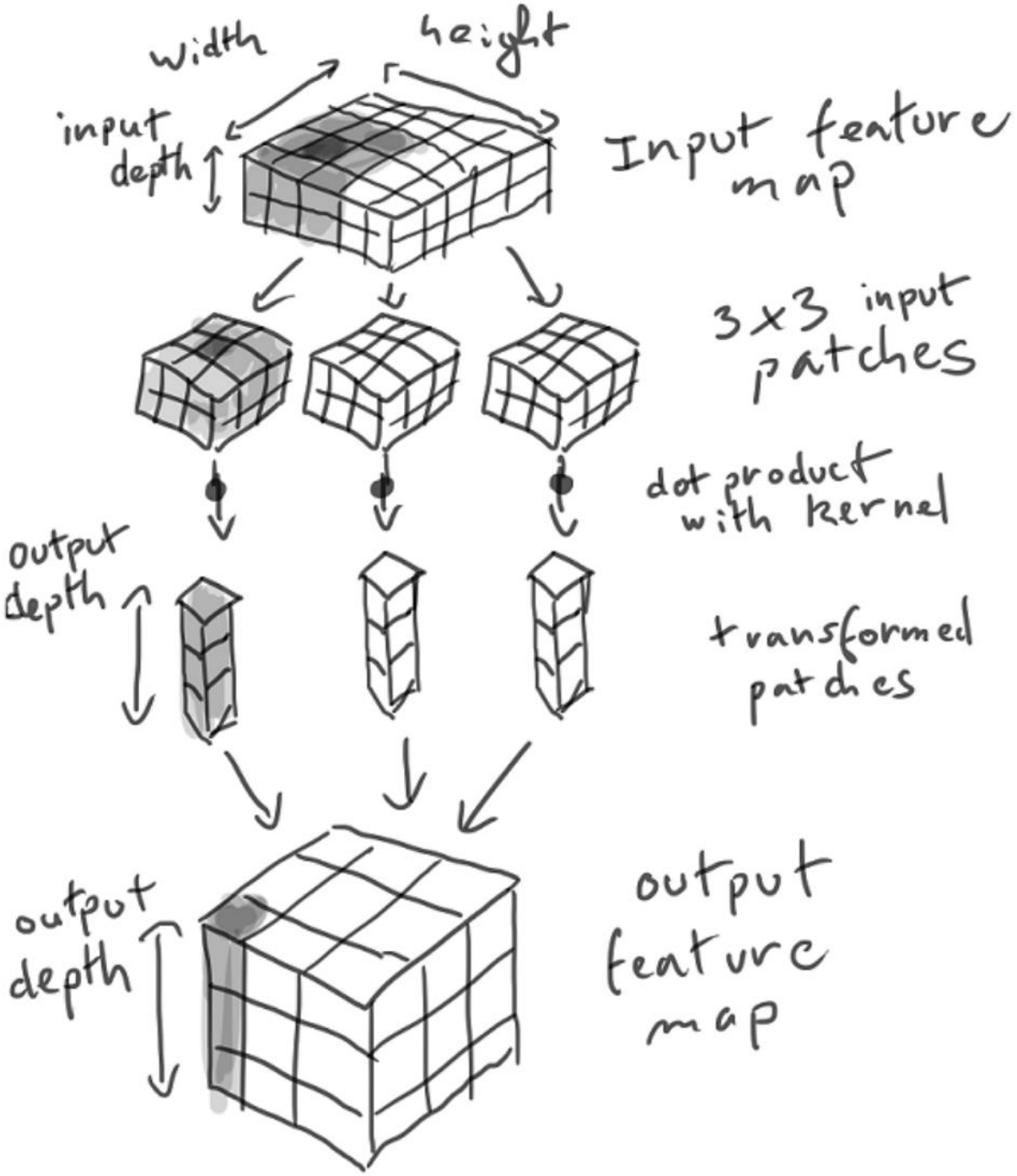




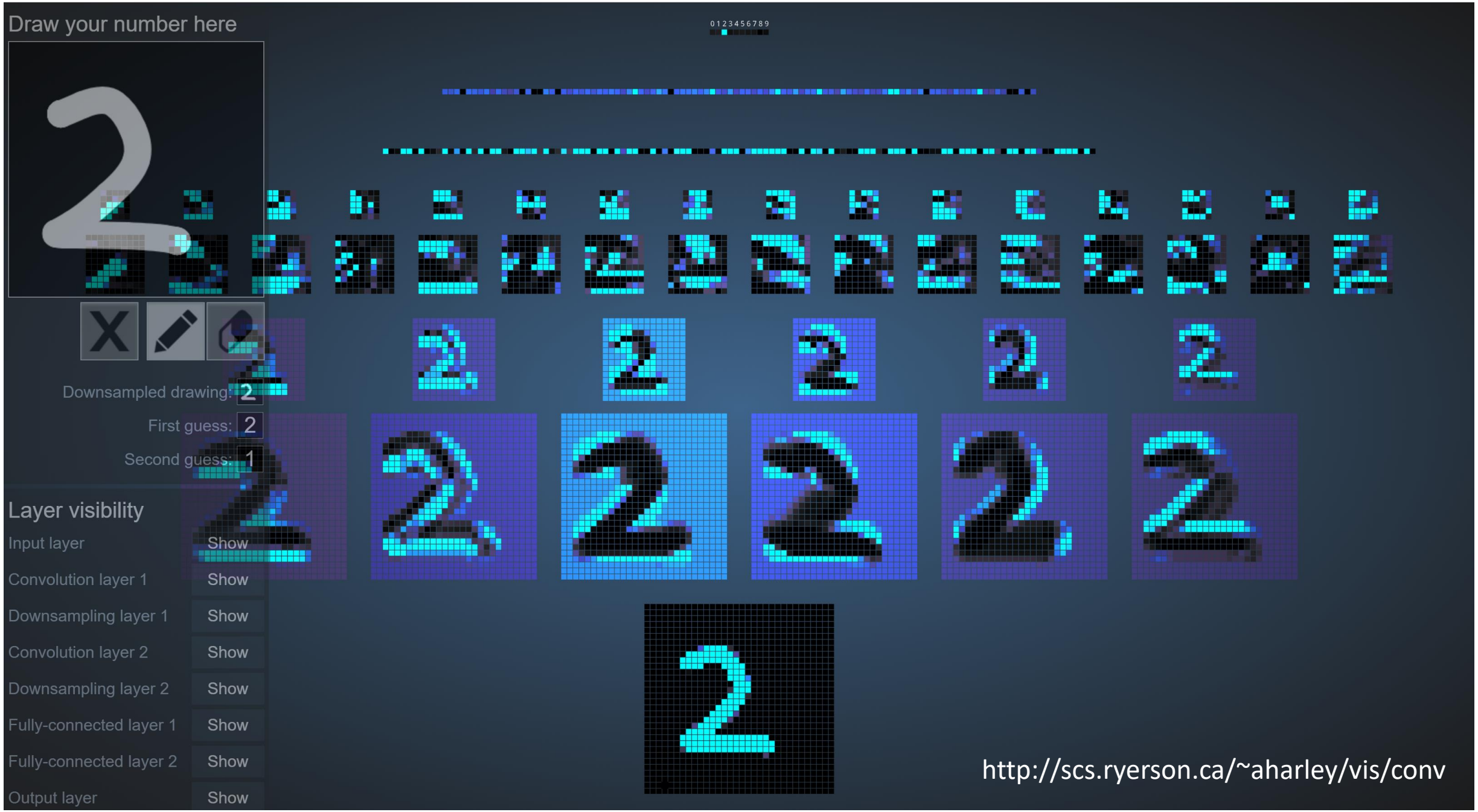
STRIDING

IMAGE CLASSIFICATION EXAMPLE

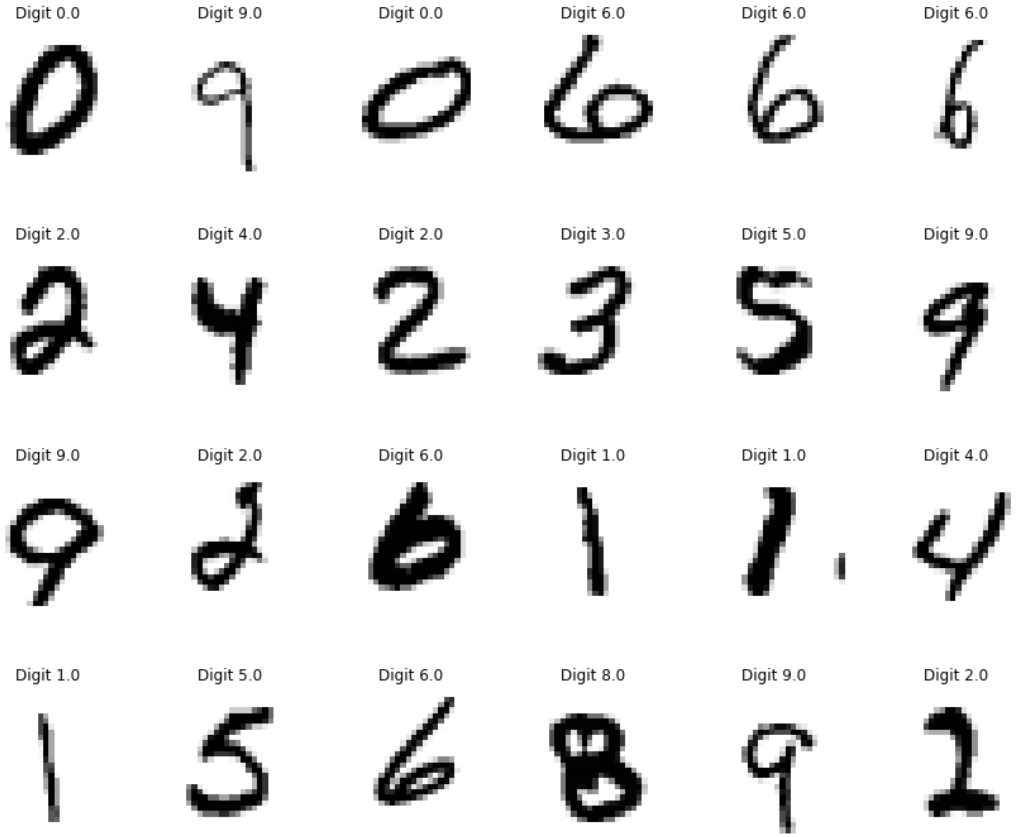


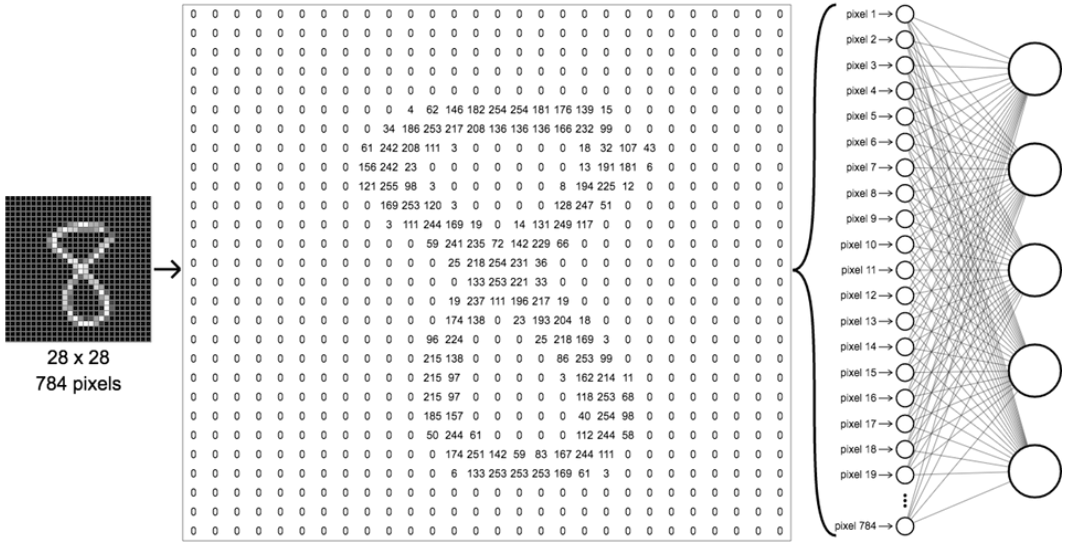


EACH LAYER DOES THIS


Draw your number here
0 1 2 3 4 5 6 7 8 9
Downsampled drawing:
First guess: 2
Second guess: 1
Layer visibility
Input layer Show
Convolution layer 1 Show
Downsampling layer 1 Show
Convolution layer 2 Show
Downsampling layer 2 Show
Fully-connected layer 1 Show
Fully-connected layer 2 Show
Output layer Show
http://scs.ryerson.ca/~aharley/vis/conv

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()                                    input_shape=(28, 28, 1))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
      Conv2D(32, (3, 3)
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
      Conv2D(64, (3, 3),
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
=======Conv2D(64, (3, 3),=======================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

(32 * 3 * 3) + 32 = 320

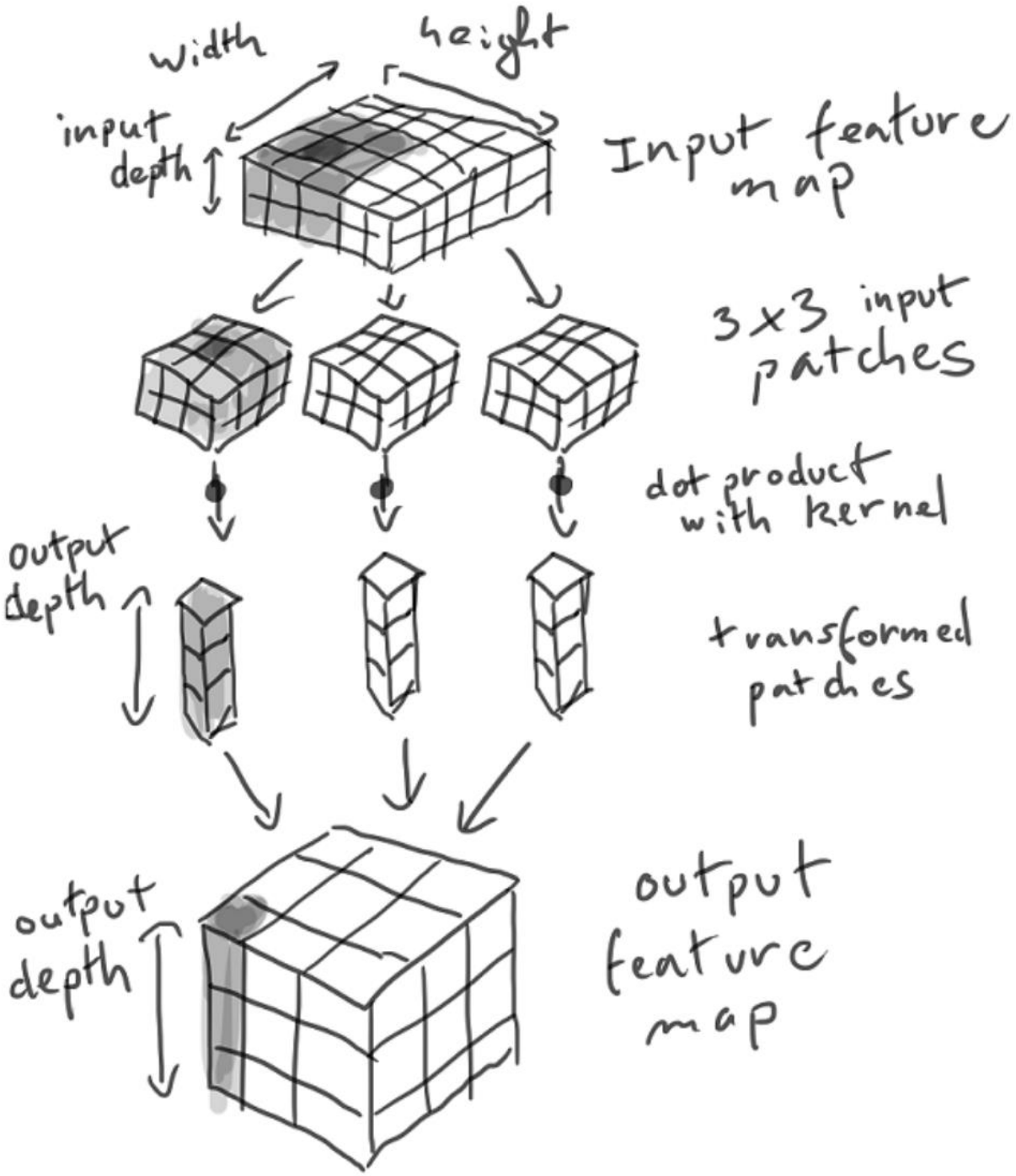
(3 * 3 * 32 + 1) * 64

(3 * 3 * 64 + 1) * 64

CNN # params independent of input size

total_params =
(filter_height * filter_width * input_image_channels + 1) * number_of_filters

Remember this?



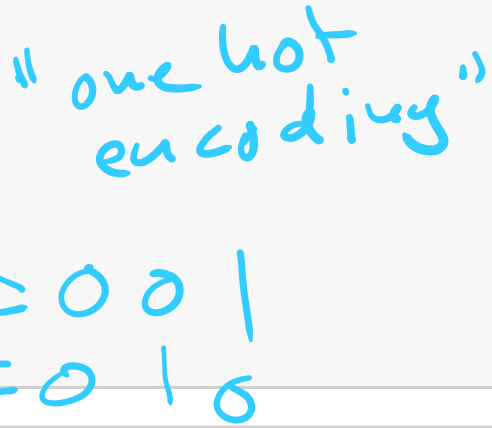
EACH LAYER DOES THIS

```
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 #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (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
=================================================================
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)
```




```
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 representations (i.e. "features") 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 #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
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.
```

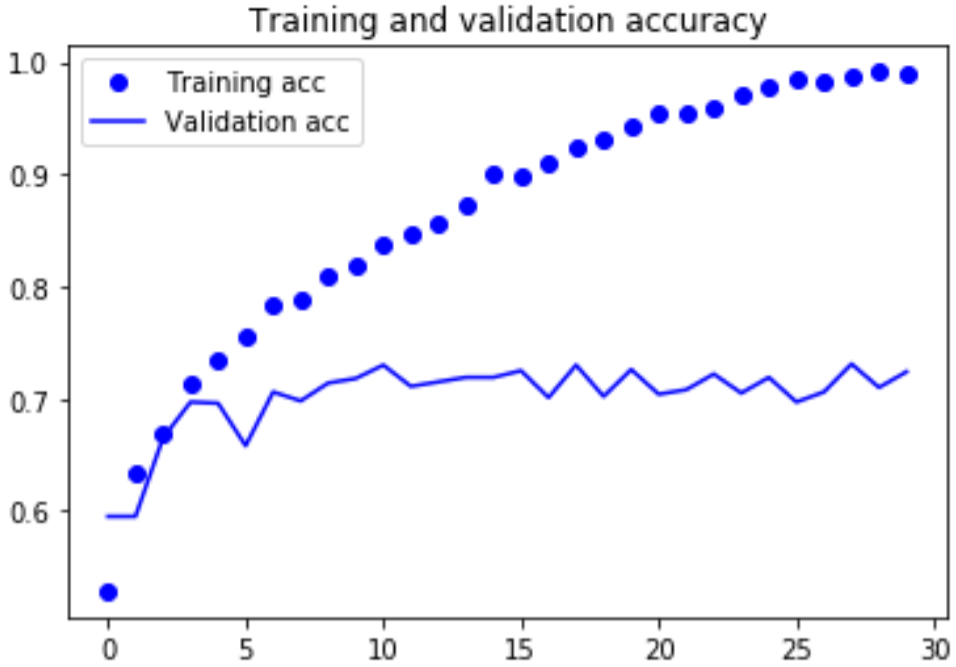
```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

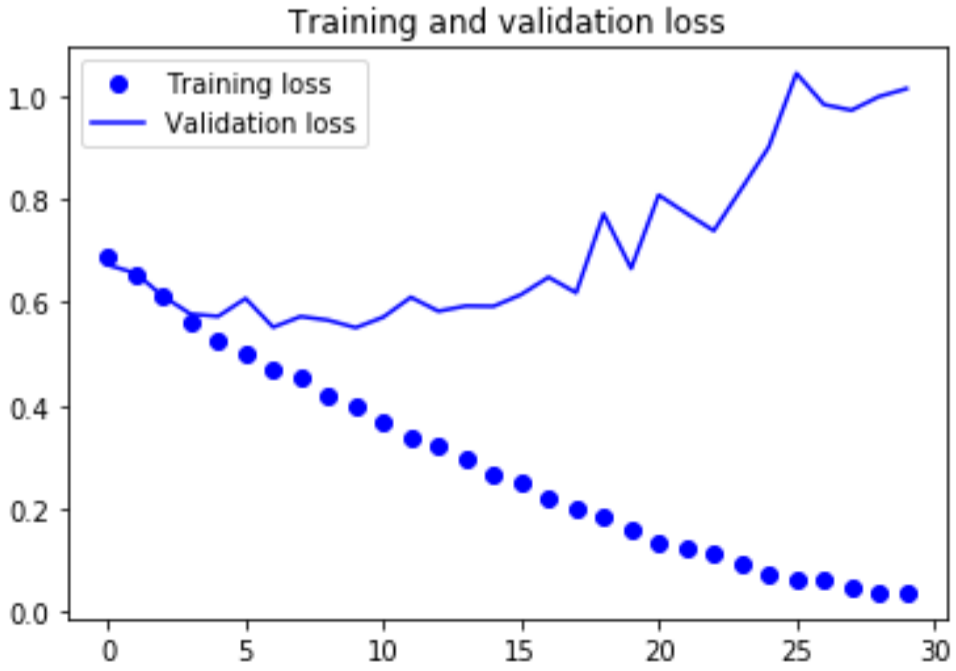
```
Epoch 1/30
100/100 [==============================] - 9s - loss: 0.6898 - acc: 0.5285 - val_loss: 0.6724 - val_acc:
0.5950
Epoch 2/30
100/100 [==============================] - 8s - loss: 0.6543 - acc: 0.6340 - val_loss: 0.6565 - val_acc:
0.5950
Epoch 3/30
100/100 [==============================] - 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()
```

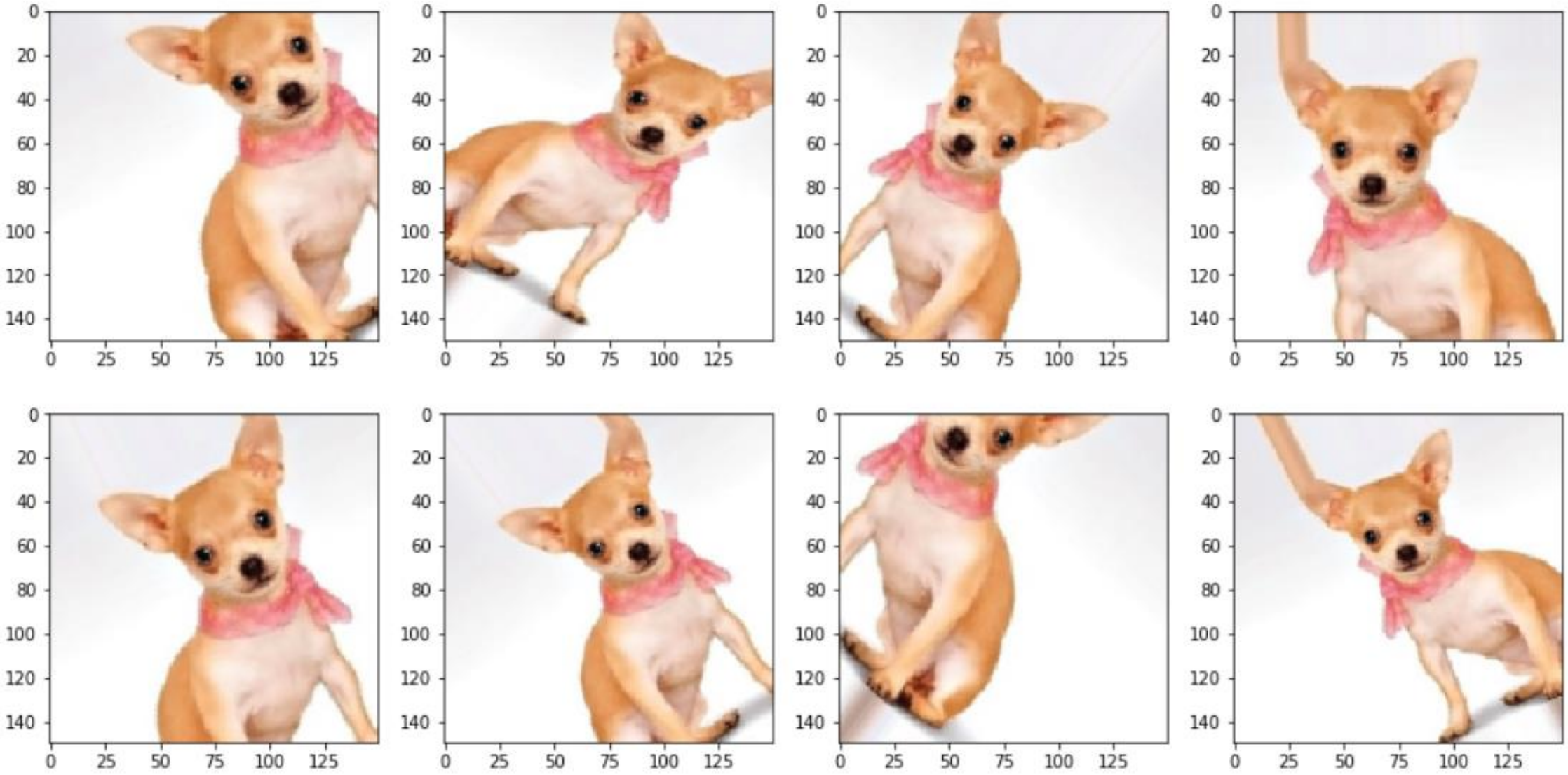




- Val acc of 70%
- Overfitting
- Dropout
- L2 regularisation
- Now... Augmentation

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

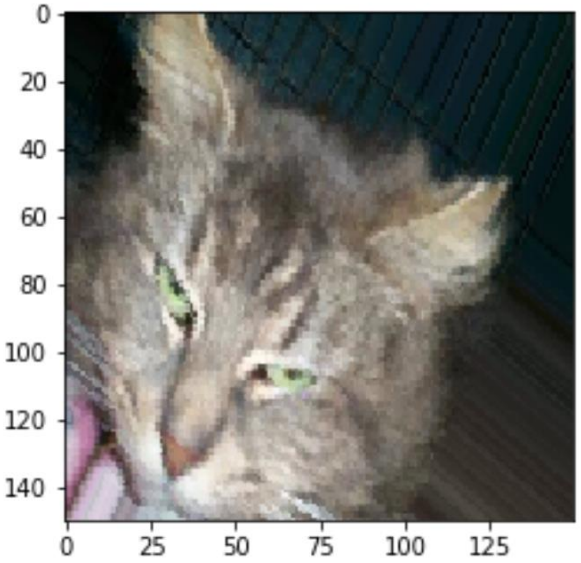
[DATA AUGMENTATION]

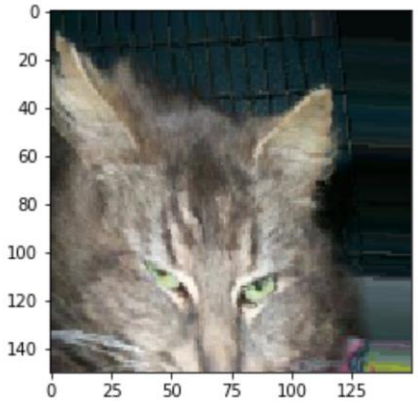


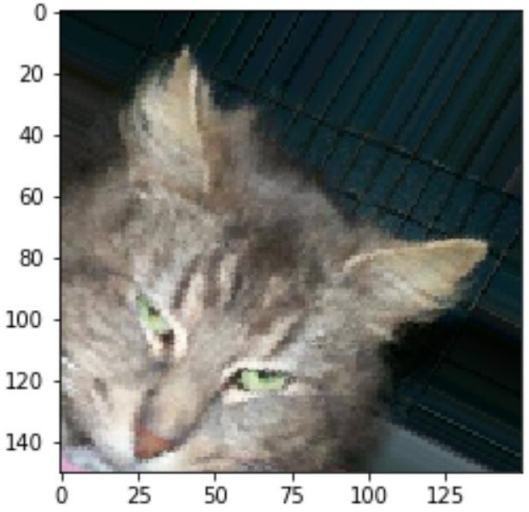
Chihuahua the movie

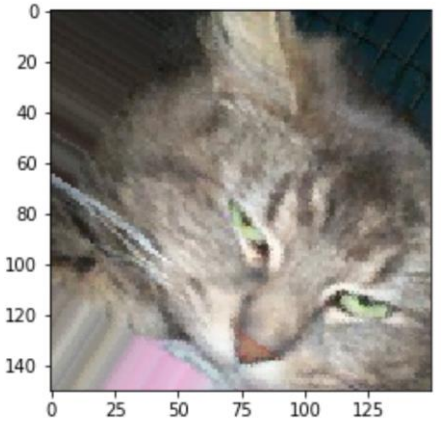
```
datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

AUGMENTED IMAGES

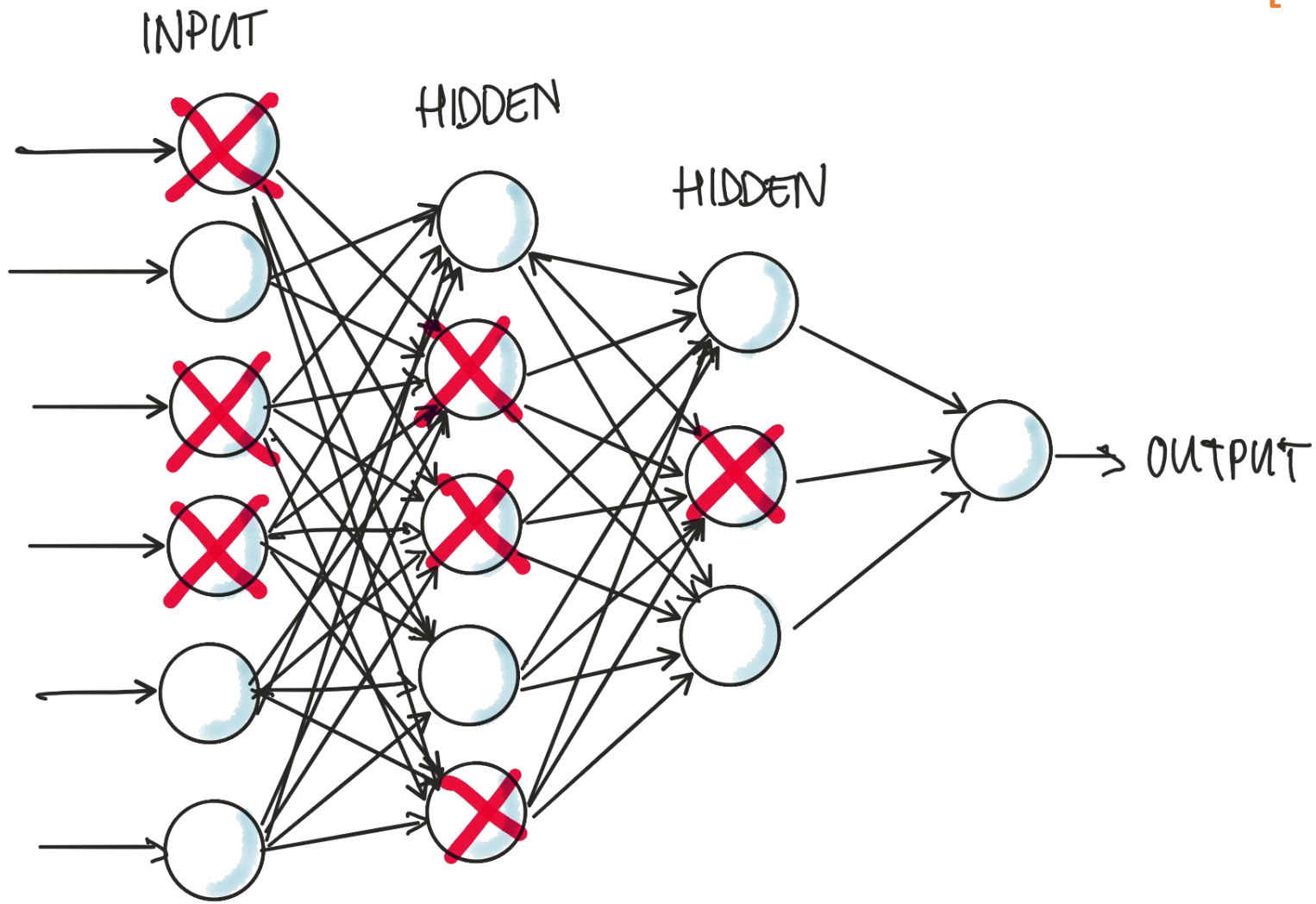








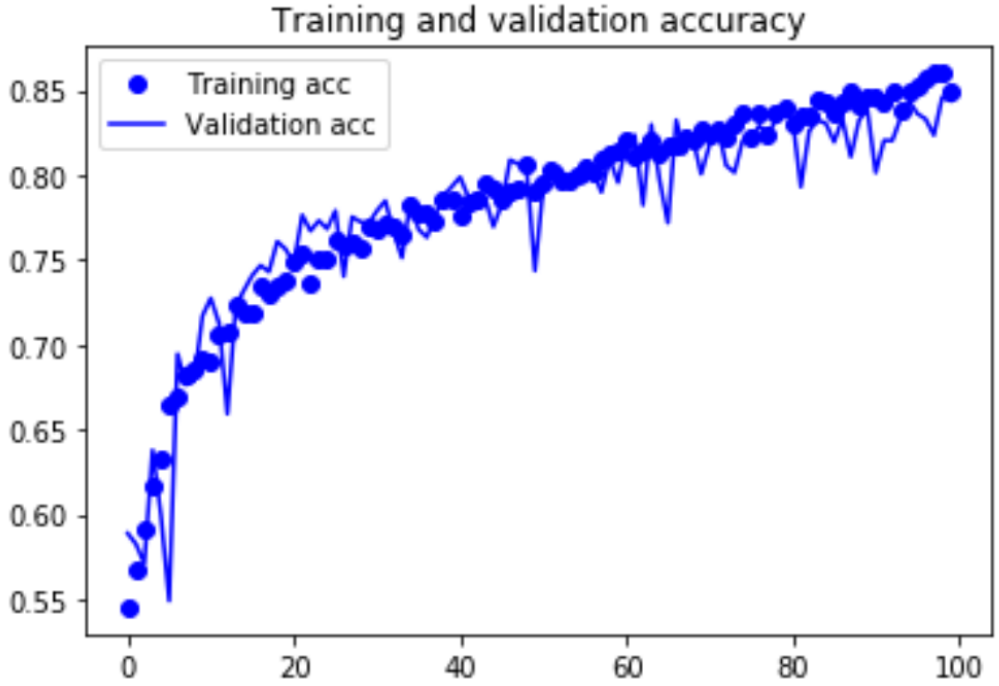
[DROPOUT]

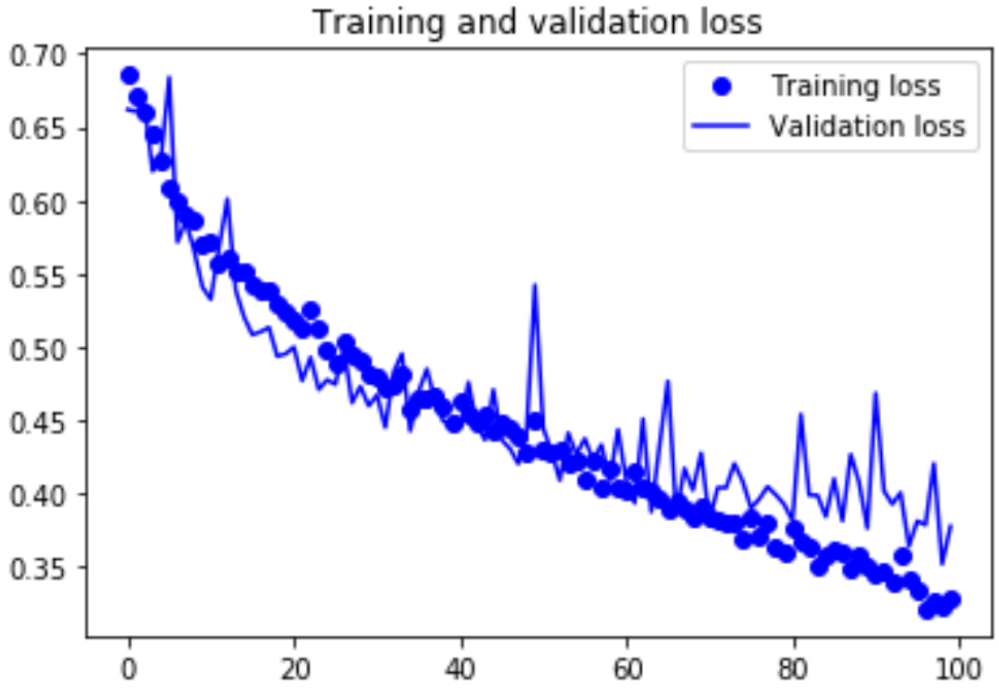


TRAIN AGAIN WITH DROPOUT AND AUGMENTATION

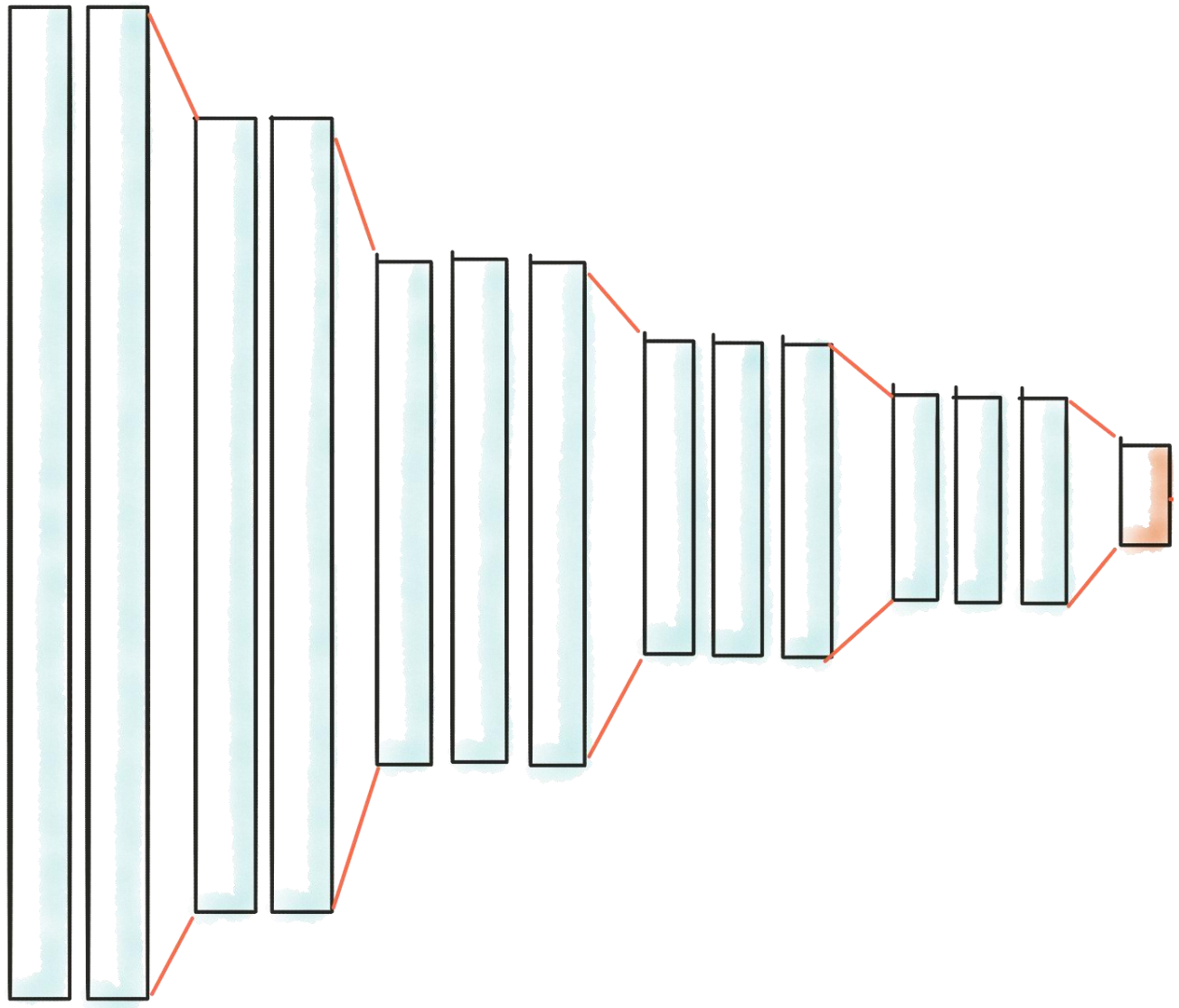
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (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'])
```

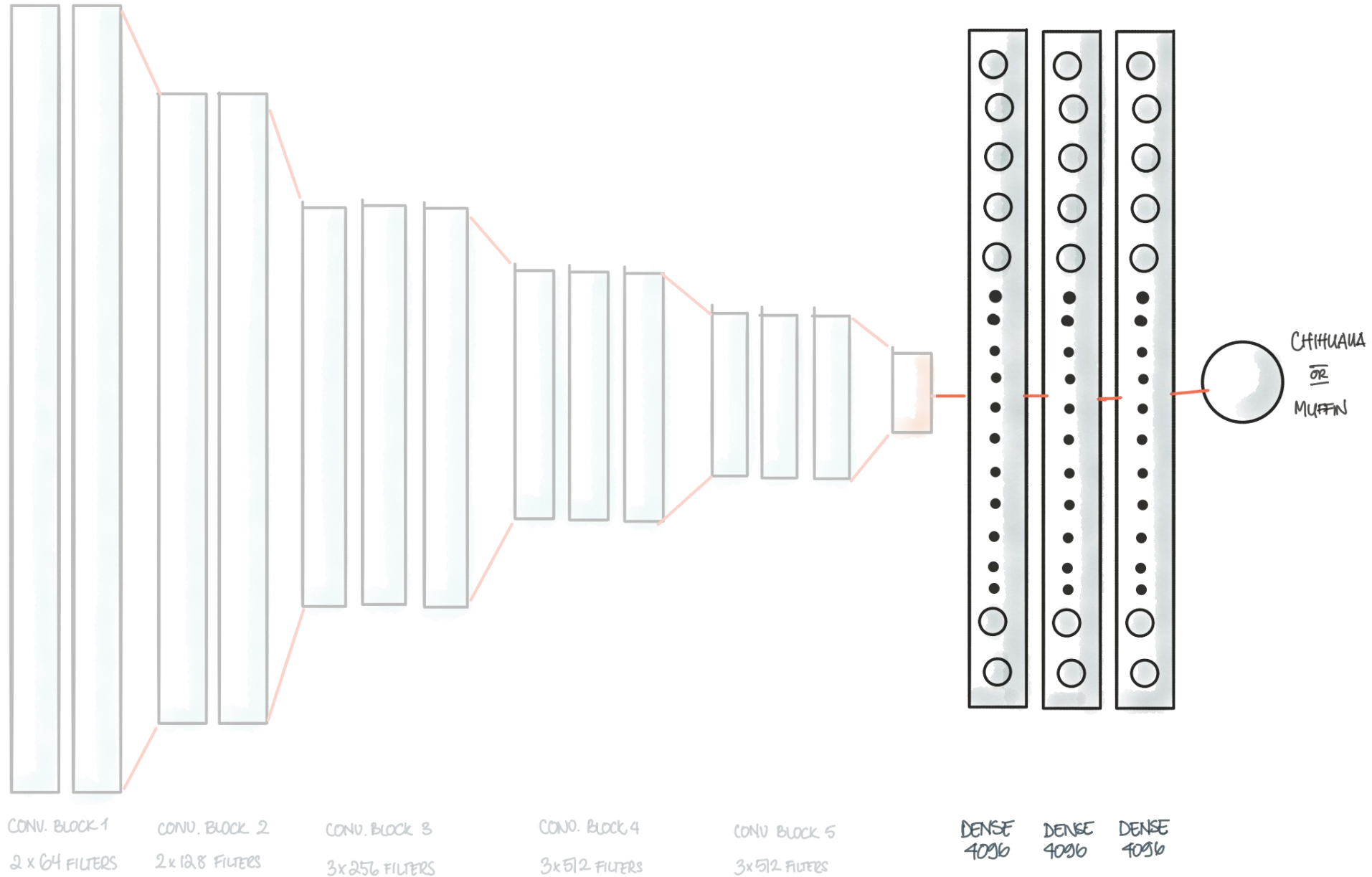
NO OVERFITTING!

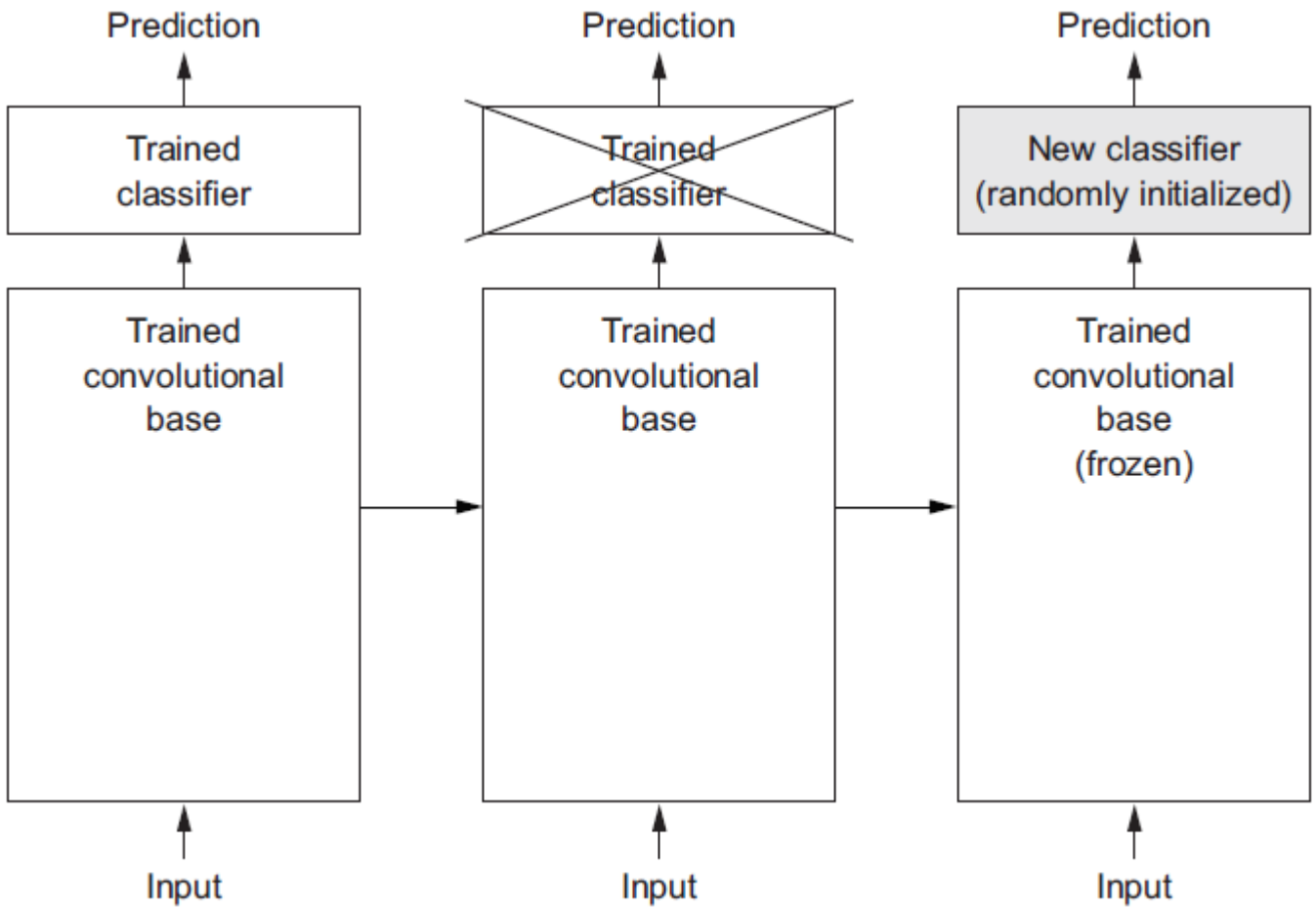




- VAL acc of 82!
- 15% relative


CONV. BLOCK 1
2 x 64 FILTERS
CONV. BLOCK 2
2 x 128 FILTERS
CONV. BLOCK 3
3 x 256 FILTERS
CONV. BLOCK 4
3 x 512 FILTERS
CONV BLOCK 5
3 x 512 FILTERS


CONV. BLOCK 1
2 x 64 FILTERS
CONV. BLOCK 2
2 x 128 FILTERS
CONV. BLOCK 3
3 x 256 FILTERS
CONV. BLOCK 4
3 x 512 FILTERS
CONV BLOCK 5
3 x 512 FILTERS
DENSE 4096
DENSE 4096
DENSE 4096
CHIHUAHUA
OR
MUFFIN



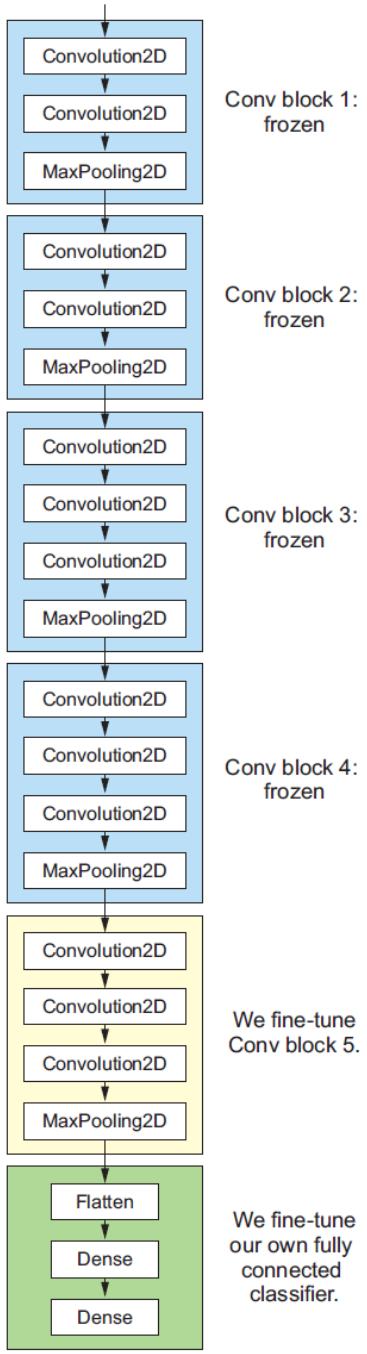


Figure 5.19 Fine-tuning the last convolutional block of the VGG16 network

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

```
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
_________________________________________________________________
```

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'))
```

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
```

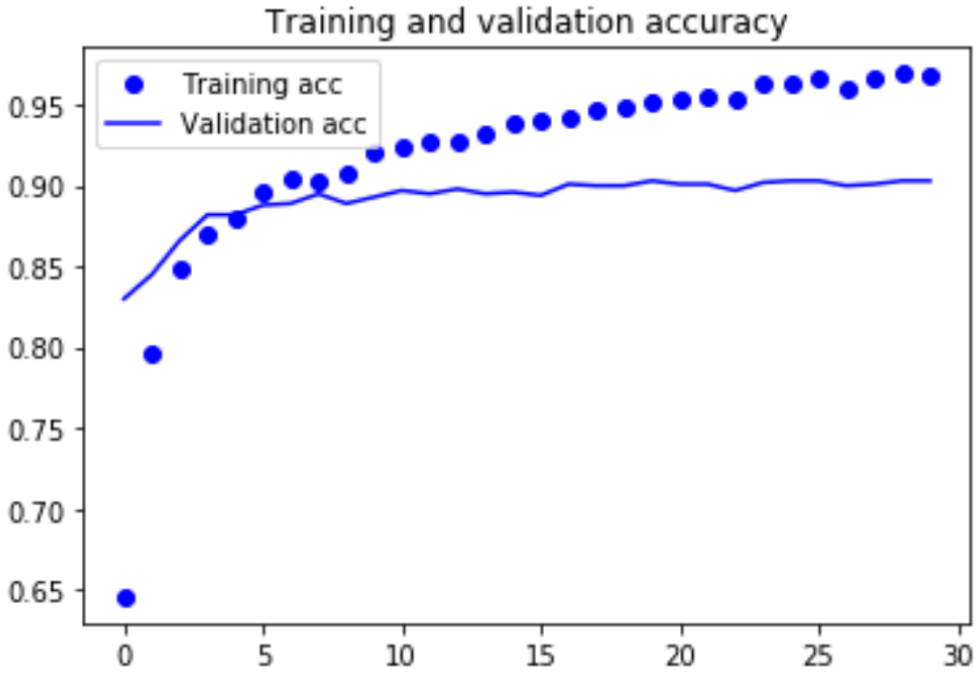
In [12]:
```
print('This is the number of trainable weights '
      'after freezing the conv base:', len(model.trainable_weights))
```

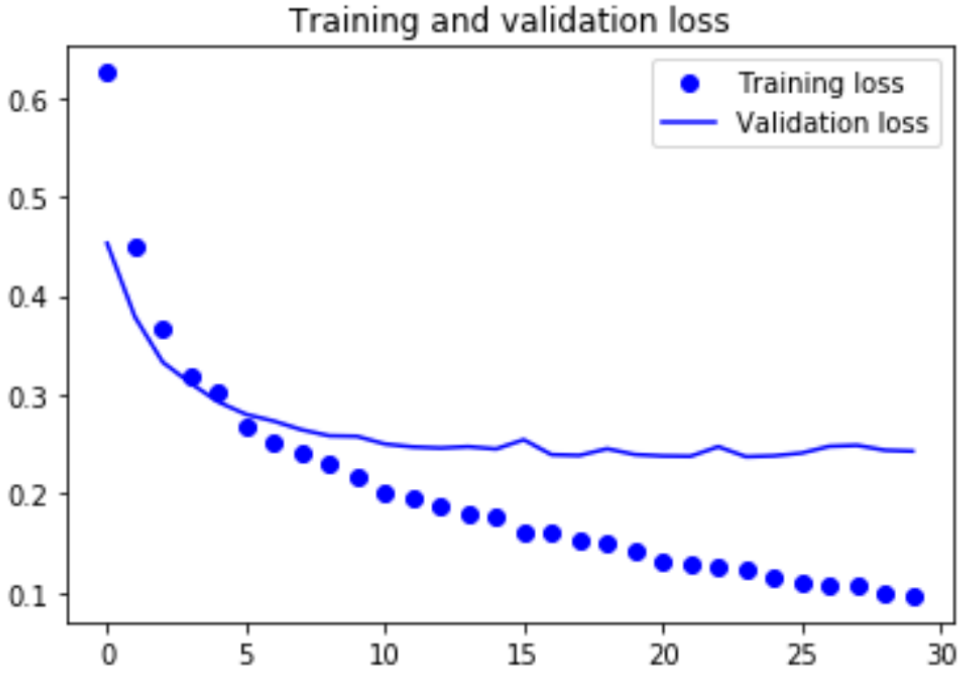
```
This is the number of trainable weights after freezing the conv base: 4
```

We don't train into the conv base until we have trained the classifier, so we don't destroy the sensitive representations learned in the base

```
model.summary()
```

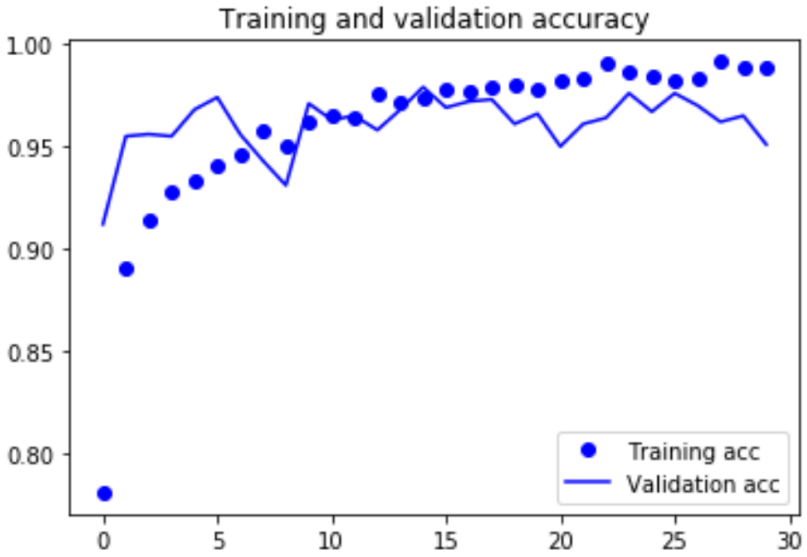
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
_________________________________________________________________
```

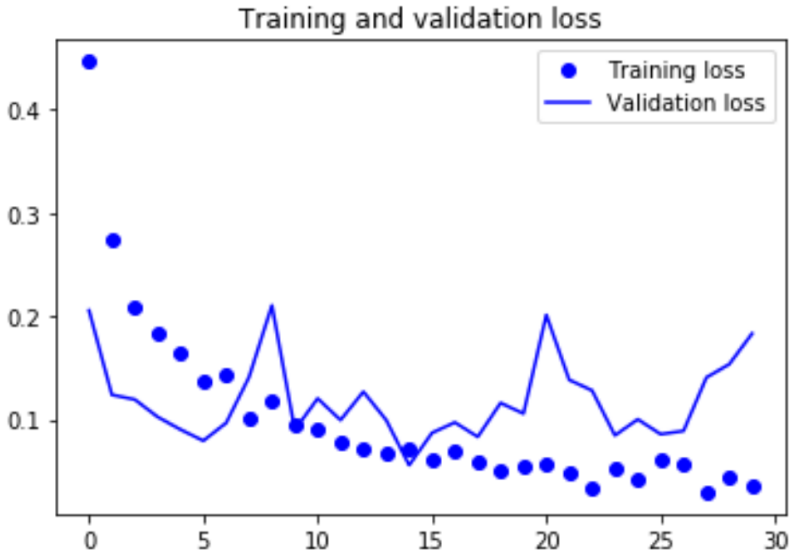




- Val now ~90%!
- Overfitting a lot
- Note this is with NO augmentation!

ADD THE AUGMENTATION BACK IN





- Val now ~96%!
- **Not overfitting**

RECAP ON VALIDATION PERFORMANCE

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

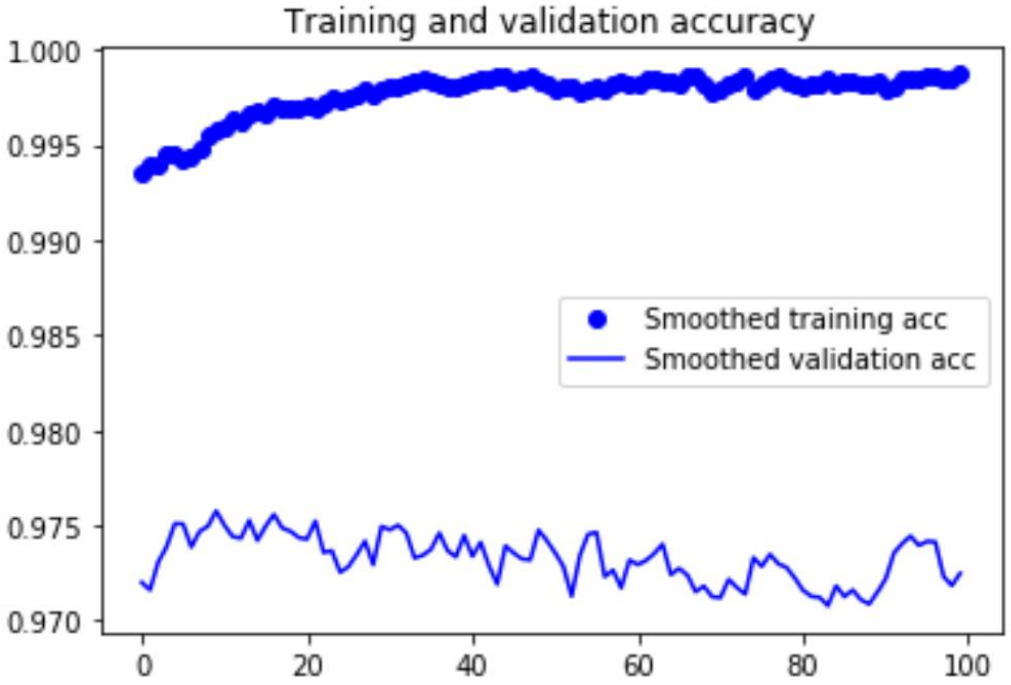
TRAINING INTO THE CONV BASE "FINE TUNING"

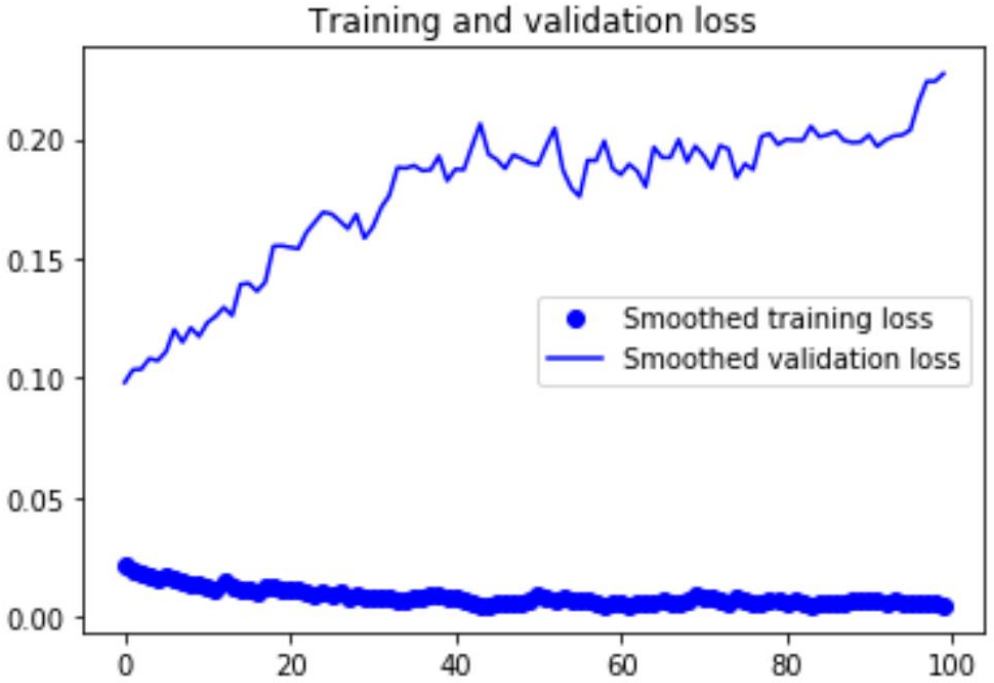
- Add your custom network on top of an already trained base network.
- Freeze the base network.
- Train the part you added.
- Unfreeze some layers in the base network.
- Jointly train both these layers and the part you added.

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!

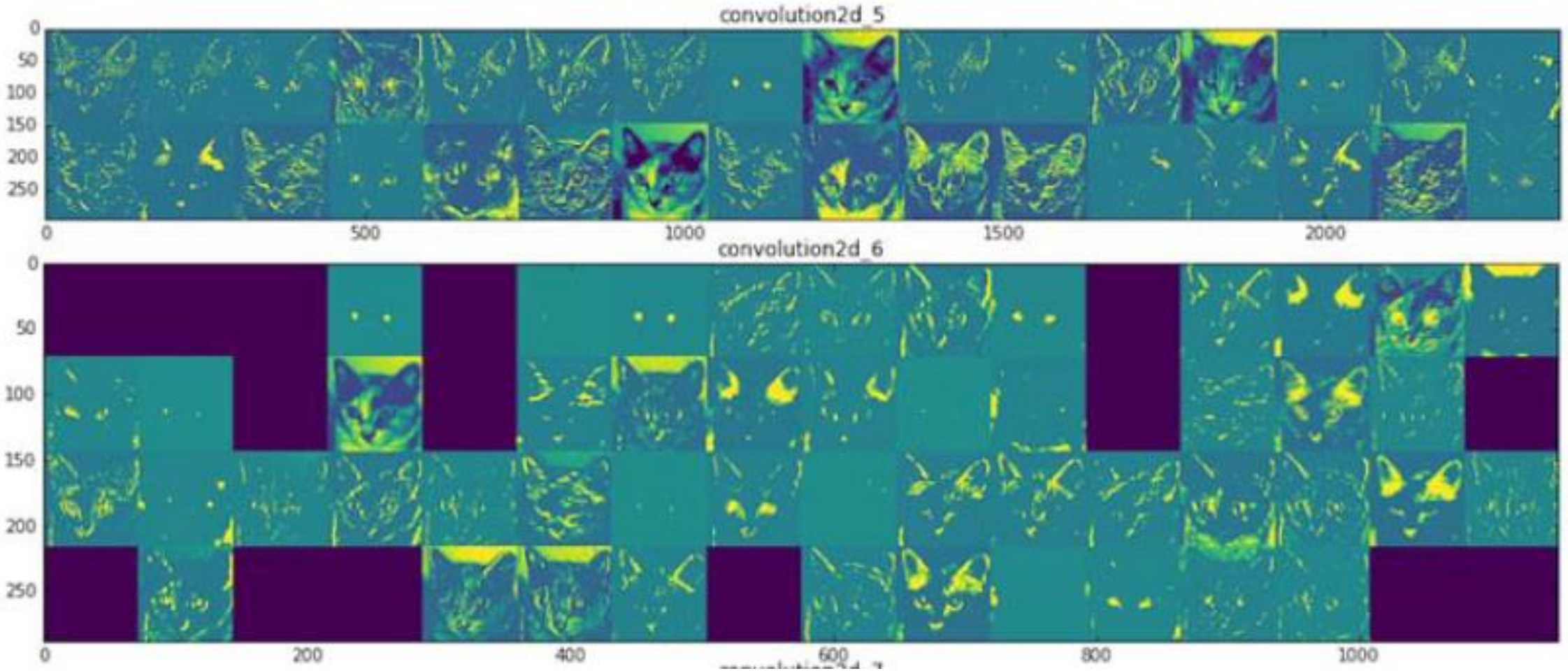
RECAP ON VALIDATION PERFORMANCE

- 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”

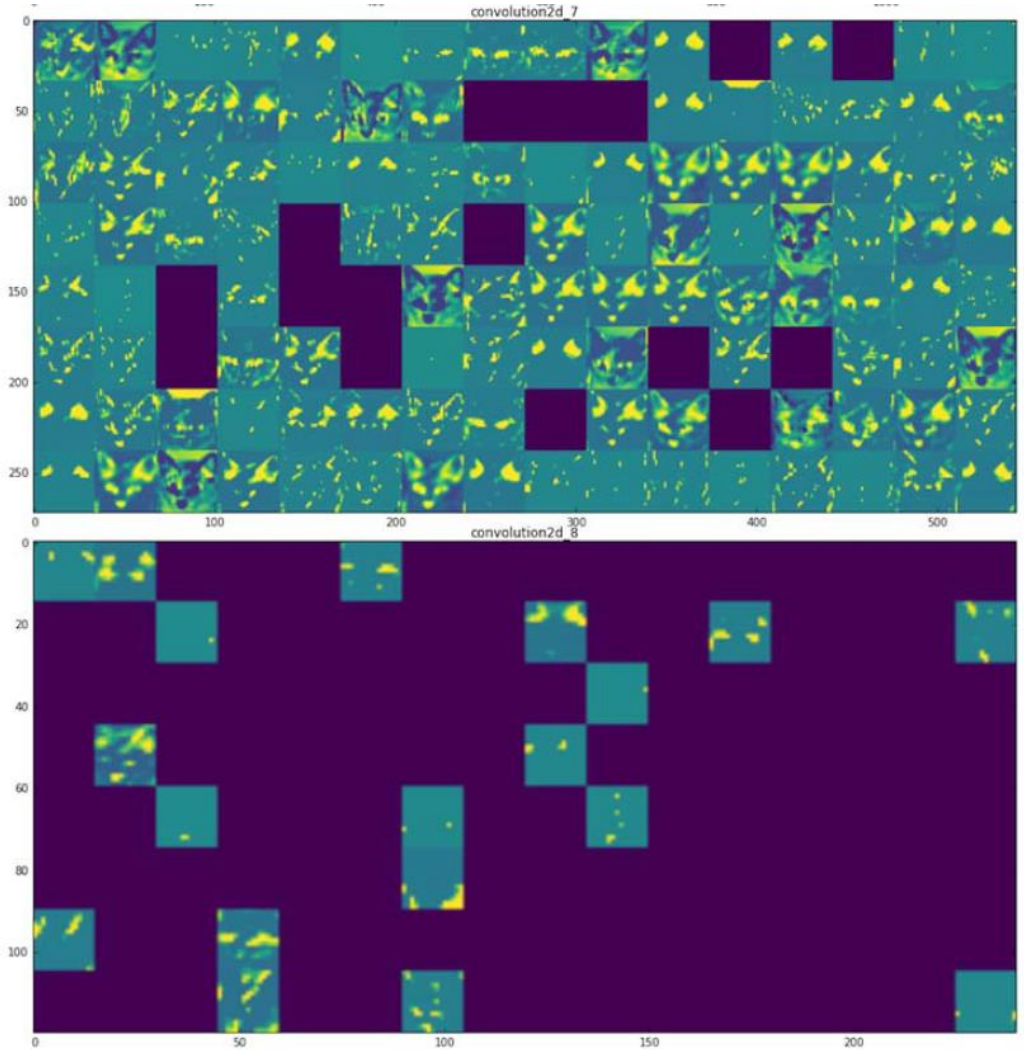
TAKE AWAYS

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

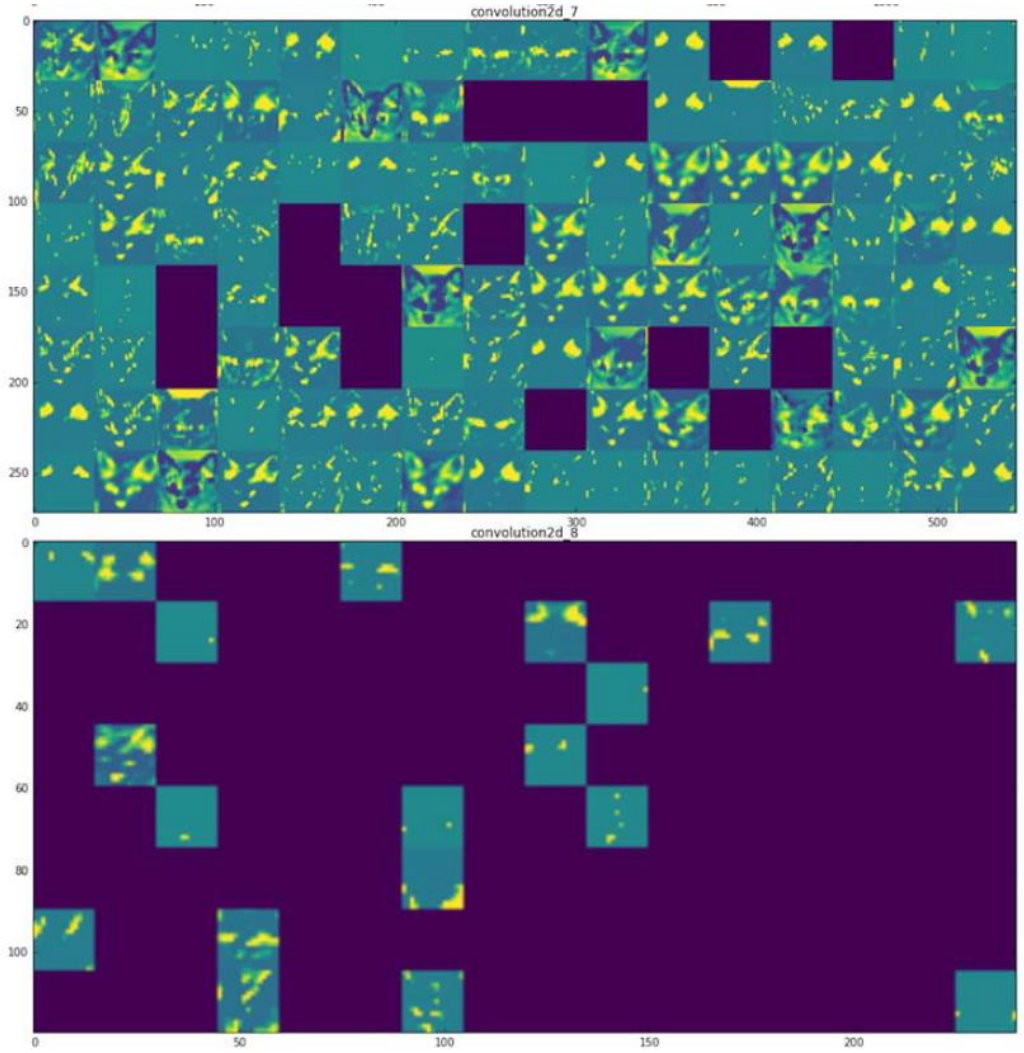
VISUALISING THE FILTERS (1)



VISUALISING THE FILTERS (2)



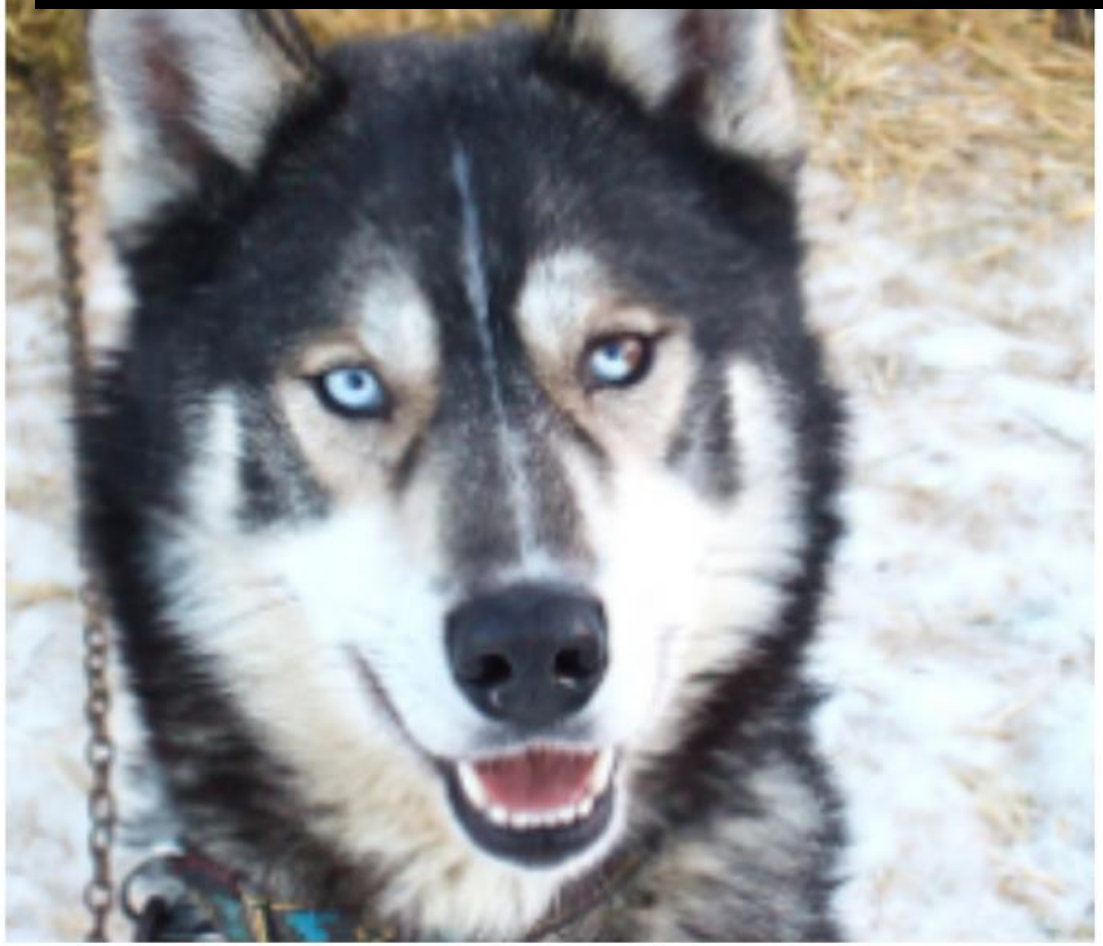
VISUALISING THE FILTERS (3)



- **Note these are actually the response maps, not filters**
- Early layers; content
- Later layers; class
- Sparsity

INTEPRETABILITY – LIME/SHAP

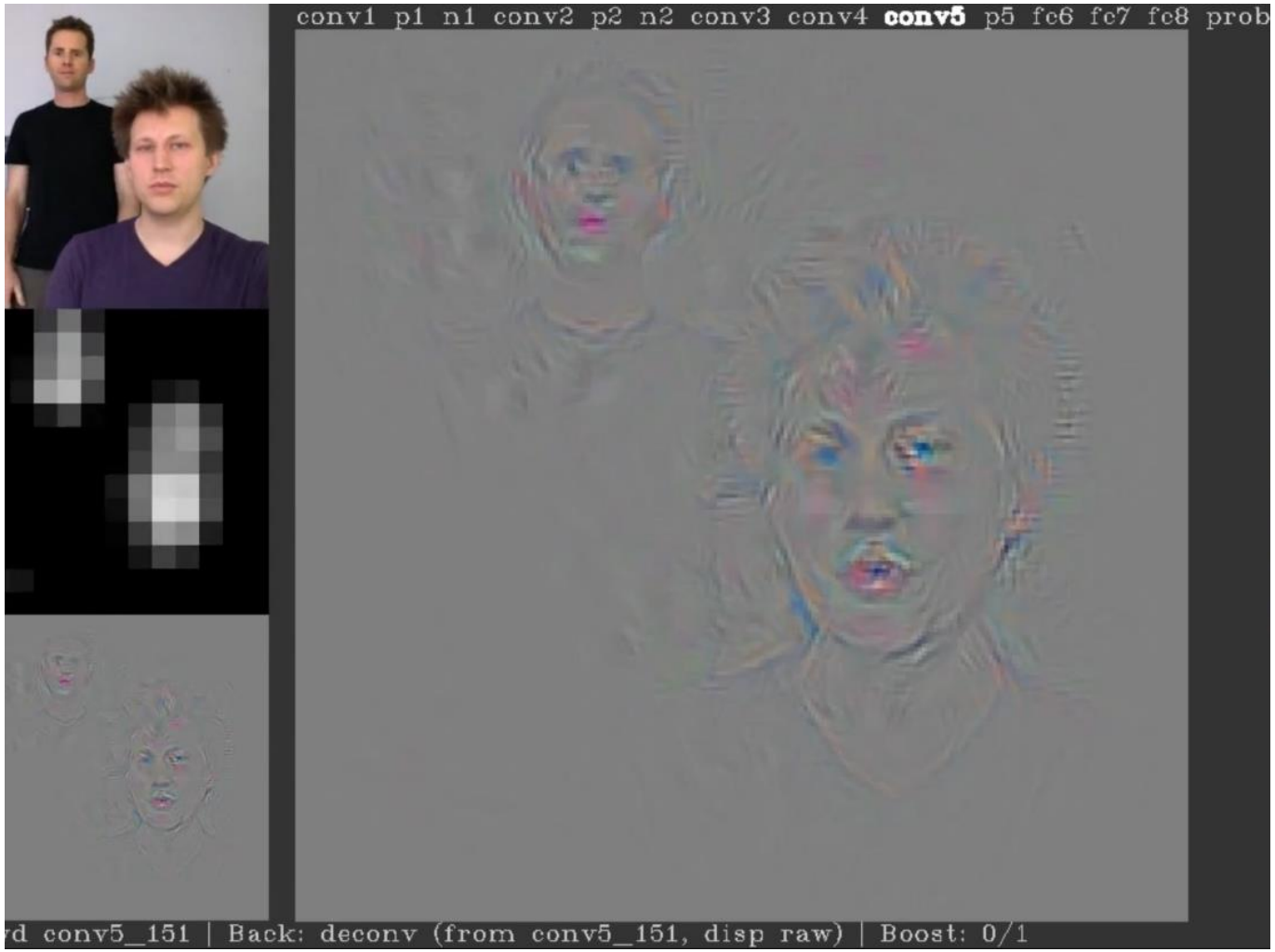
[LIME]



(a) Husky classified as wolf

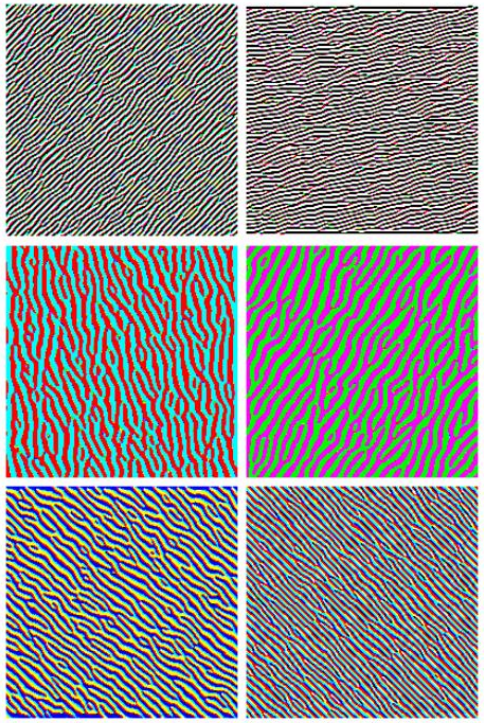
(b) Explanation

DEEP VISUALISATION TOOLBOX "SALIENCY MAPS"

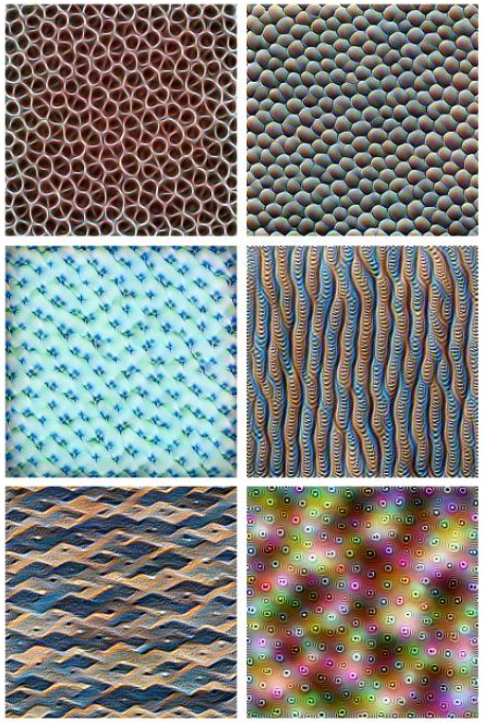


https://www.youtube.com/watch?v=AgkfIQ4IGaM

FEATURE VISUALISATION



Edges (layer conv2d0)



Textures (layer mixed3a)

Patterns (layer mixed4a)

Parts (layers mixed4b & mixed4c)

Objects (layers mixed4d & mixed4e)

Tim Scarfe

@ecsquendor

youtube.com/machinelearningatmicrosoft

THANK YOU!

Azure

Microsoft