UCL ARTIFICIAL
INTELLIGENCE SOCIETY

TUTORIAL #6

Session 6

Introduction to *Computer Vision*

UCL ARTIFICIAL
INTELLIGENCE SOCIETY

LECTURE OVERVIEW

01

Image preprocessing

Images. How can we make them better?

02

Computer Vision

Theory behind this area of Machine Learning.

03

CV Implementation

Showcase of the implementing CV models.

04

Competition

Try out everything you learned so far!

COMPETITION

@louisdewardt

@charlieharrison943

@Zaki

@Polly2327

@ladams

@asubedi

@chloemt03

@Gen1Code

@andrew

@FranekNowak

@y_kimz

@louiskwok

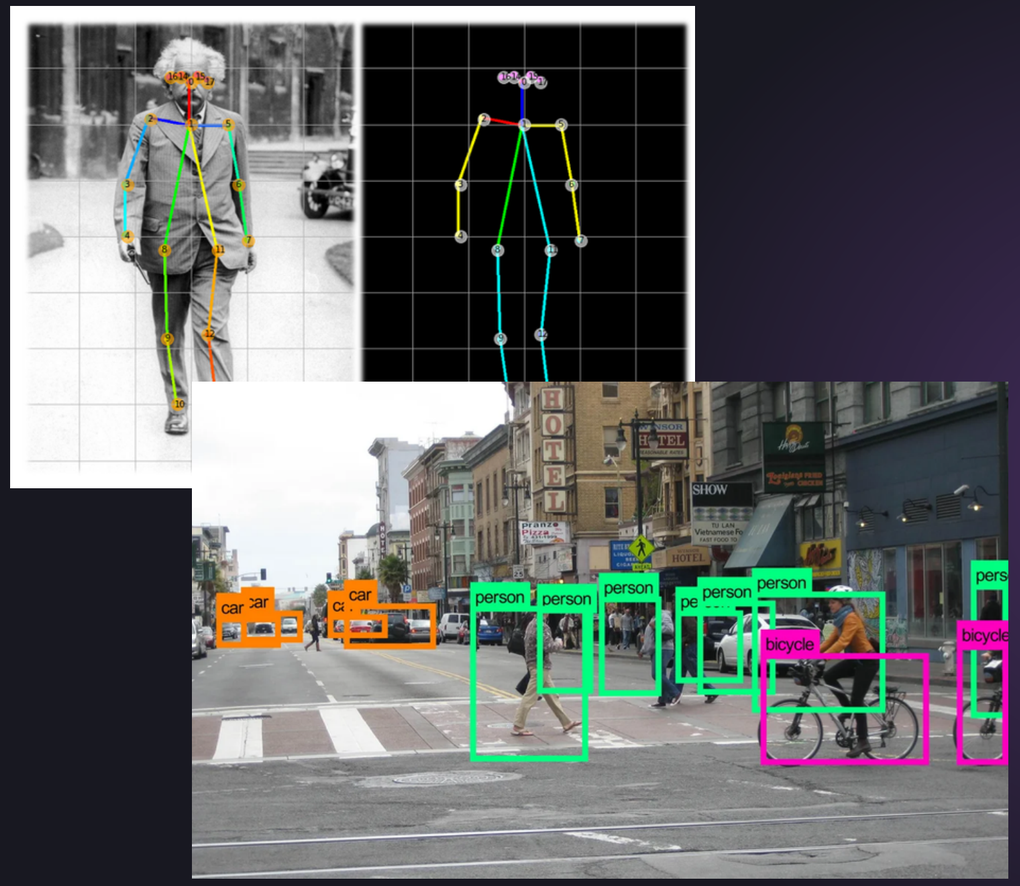
@louisdewardt

Theory

COMPUTER VISION

"Field of AI that enables computers to derive meaningful information from digital images and other visual inputs"

If AI enables computers to think, computer vision enables them to see, observe and understand.



Theory

COMPUTER VISION

A classical and central problem in computer vision is face recognition. Humans perform this task effortlessly, rapidly, reliably, and unconsciously.

(We don’t even know quite how we do it; like so many tasks for which our neural resources are so formidable, we have little “cognitive penetrance” or understanding of how we actually perform face recognition.)

Theory

COMPUTER VISION

Consider these three facial images:

Which two pictures show the same person?

Most algorithms for computer vision select 1 and 2 as the same person, since those images are more similar than 1 and 3.

Theory

COMPUTER VISION

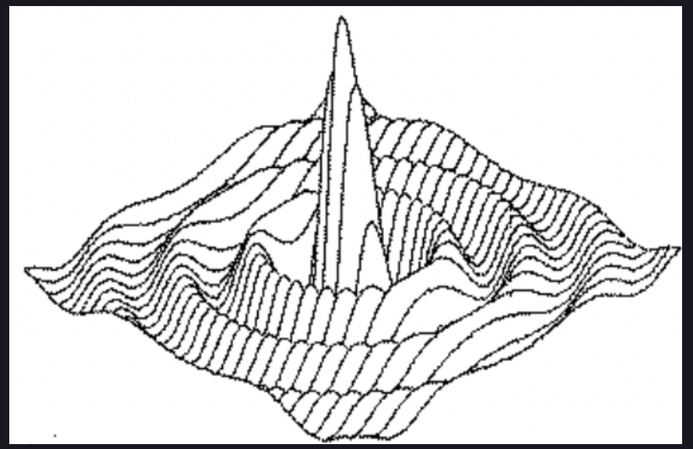
It is often difficult to simply train CV models on raw data. This is a discipline that often requires abstraction and preprocessing to make training techniques work.

Theory

COMPUTER VISION

A few of the tasks we need to solve in vision are:

- infering depth properties from an image
- infering surface properties from image properties
- infering colours in an illuminant-invariant manner
- infering structure from motion, shading, texture, shadows
- infering a 3D shape unambiguously from a 2D line drawing



Theory

IMAGE PREPROCESSING

As with any Supervised Machine Learning approach, we have to ask ourselves about the data.

- Why do we need to preprocess our data?
- What kinds of image preprocessing are there?

Theory

IMAGE PREPROCESSING

As with any Supervised Machine Learning approach, we have to ask ourselves about the data.

- Why do we need to preprocess our data?
- What kinds of image preprocessing are there?

Theory

IMAGE PREPROCESSING

What kinds of image preprocessing are there?

To reduce the complexity of most tasks and thus increases models accuracy. Furthemore, to make the data better fit our task.

What kinds of image preprocessing are there?

The preprocessing includes everything from denoising the image, colour balancing or sharpening the image to cropping or rotating it.

Theory

IMAGE PREPROCESSING

Changing colorspace

There are several color spaces:
- RGB
- Grayscale
- CMYK

Each metric in color space is one channel, which you can change

RGB

CMYK

Grayscale

Theory

IMAGE PREPROCESSING

Changing colorspace

Gray scale

Sometimes, you might want to build a model that doesn't need color information. Reducing the images to gray scale can save computations for a model.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
imshow(gray)
```

Theory

IMAGE PREPROCESSING

Thresholding

Changing the colour of pixels into just few values

Thresholding has several techniques, which can be useful for different tasks (binary, adaptive, ...)

Threshold = "Where is the line between when pixels are given one value and when they are given another"

Theory

IMAGE PREPROCESSING

Thresholding

Binary thresholding

```
# if the pixel value is smaller than 200, make it 0
# else, make it 250
(T, thresh) = cv2.threshold(gray, 200, 250, cv2.THRESH_BINARY)
imshow(thresh)
```

Theory

IMAGE PREPROCESSING

Geometric transformations

Scaling

Rotation

Flipping

Theory

IMAGE PREPROCESSING

Geometric transformations

Shifting / Translation

```
shift_w = 100  # shift to right by 100
whift_h = 50  # shift to bottom by 50
shift_matrix = np.float32([[1, 0, shift_w], [0, 1, whift_h]])
shifted = cv2.warpAffine(image, shift_matrix, (w, h))

imshow(shifted)
```

Flipping

```
flip_vertical = cv2.flip(image, 0)
imshow(flip_vertical)
```

```
flip_horizontal = cv2.flip(image, 1)
imshow(flip_horizontal)
```

Rotation

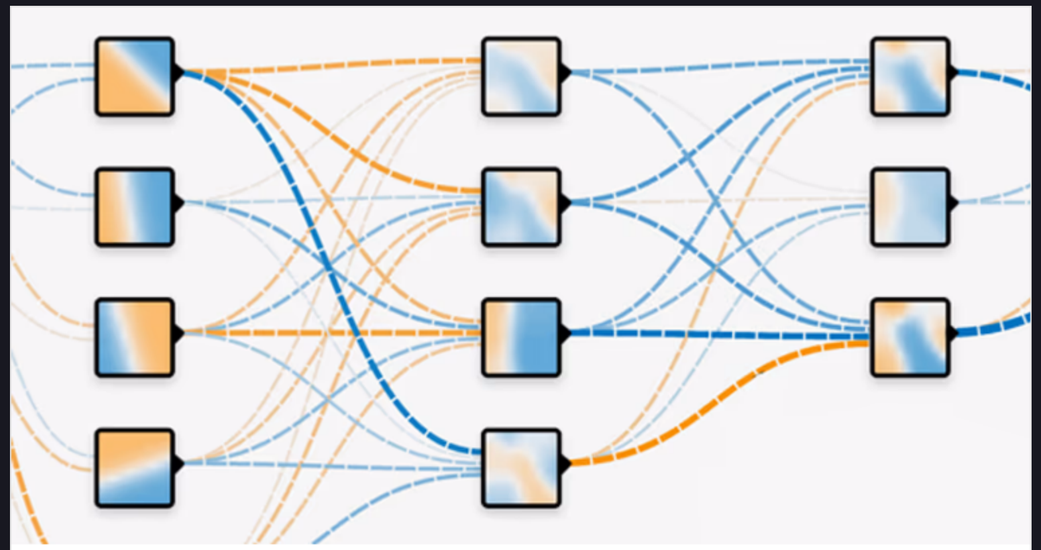
```
(h, w) = image.shape[:2]
(cX, cY) = (w // 2, h // 2)
M = cv2.getRotationMatrix2D((cX, cY), 30, 1.0)
rotated = cv2.warpAffine(image, M, (w, h)) # rot

imshow(rotated)
```

Theory

CNN

First lets recap a NN



Tensorflow — Neural Network Playground

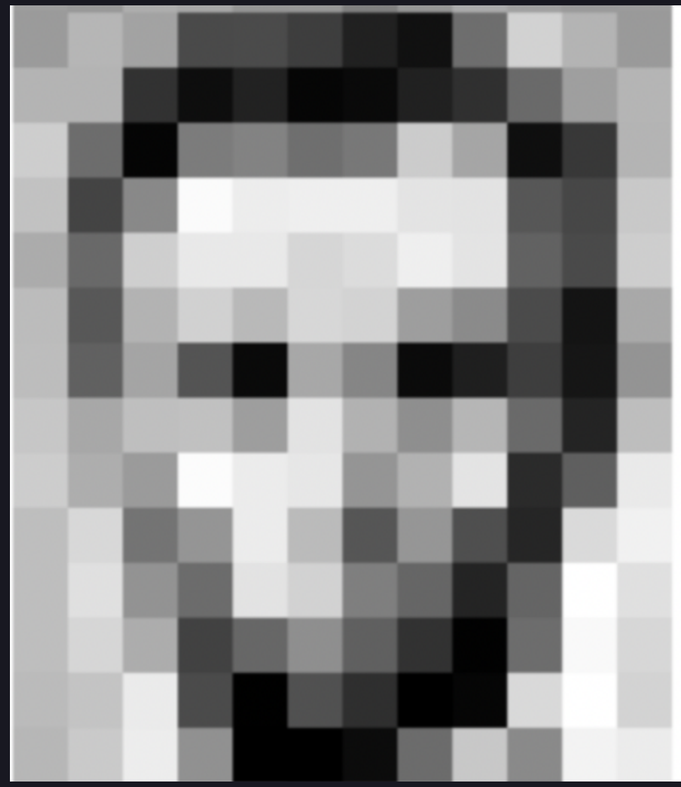
Tinker with a real neural network right here in your browser.

tensorflow.org /

Theory

CNN

Image representation



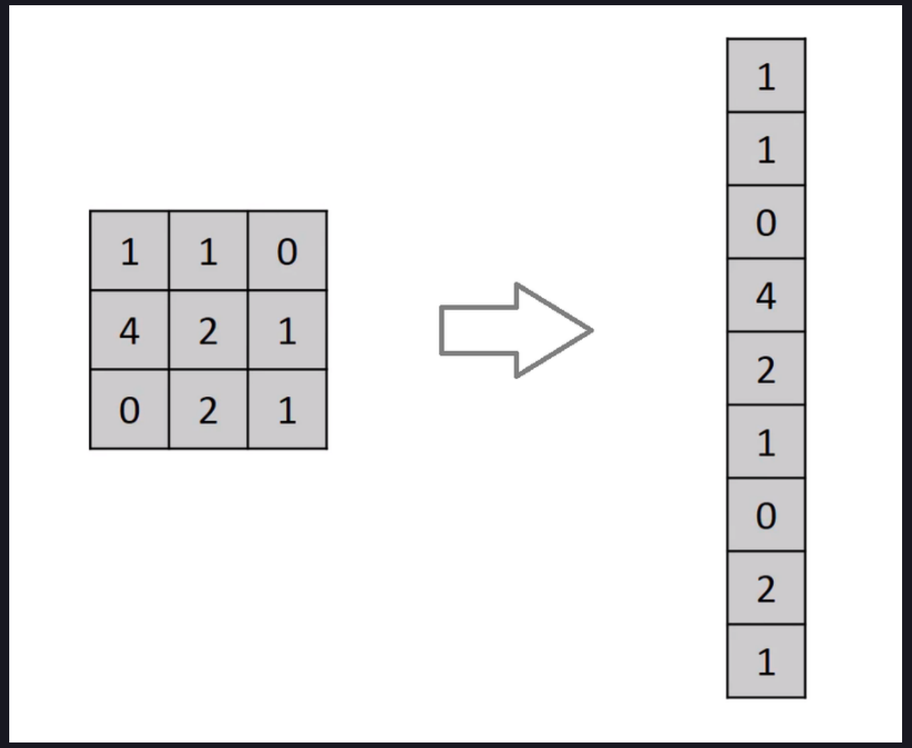
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148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

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

Theory

CNN

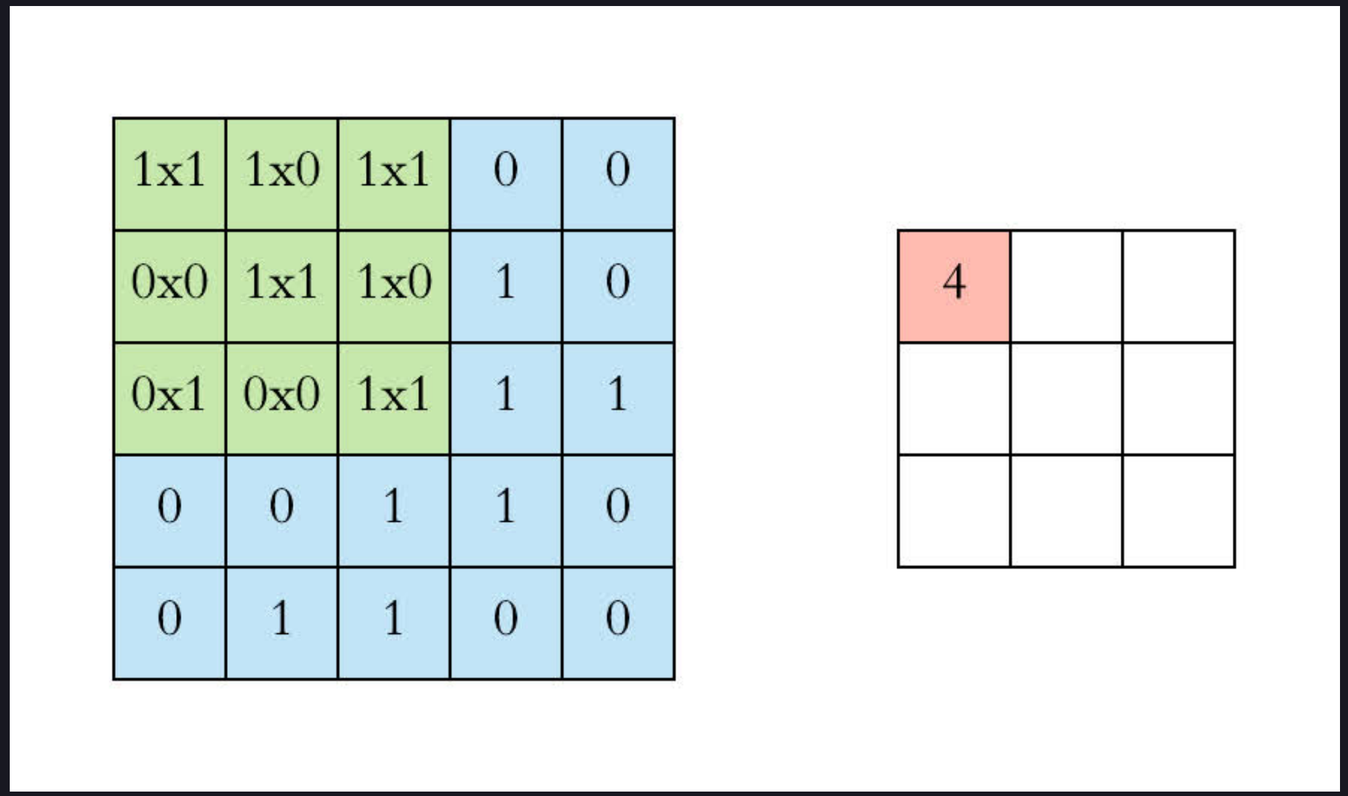
Image representation



Theory

CNN

CNN = FILTER/KERNEL + NN



Theory

CNN

Why apply kernels?

Applying kernels is a way to reduce the image size while highlighting various features in that image.

Theory

CNN

Why apply kernels? – Example: Edge Detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

*

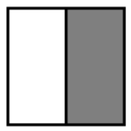
1	0	-1
1	0	-1
1	0	-1

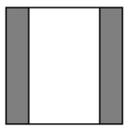
3 x 3

=

-0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

4 x 4



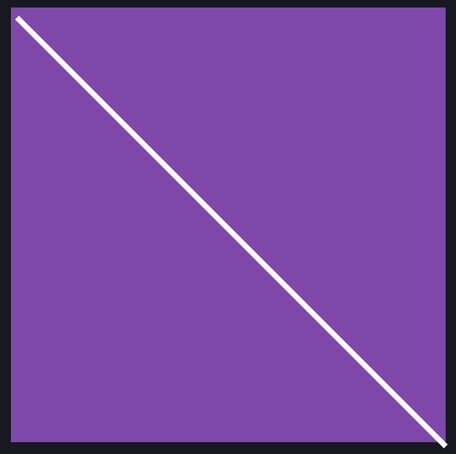


Theory

CNN

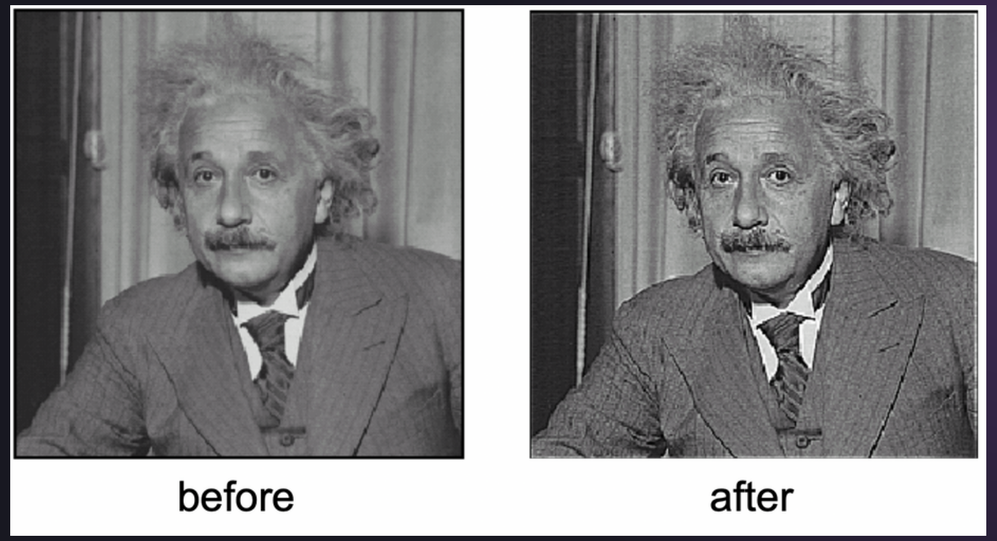
Exercise

Diagonal



Blur

Sharpen



Theory

CNN

Exercise

Diagonal

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

Blur

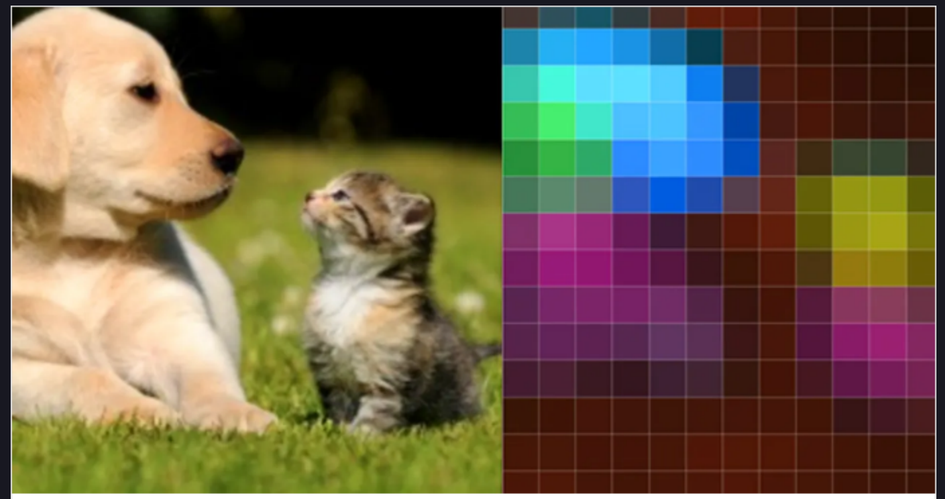
$$1/9 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpen

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - 1/9 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Theory

CNN



The Building Blocks of Interpretability

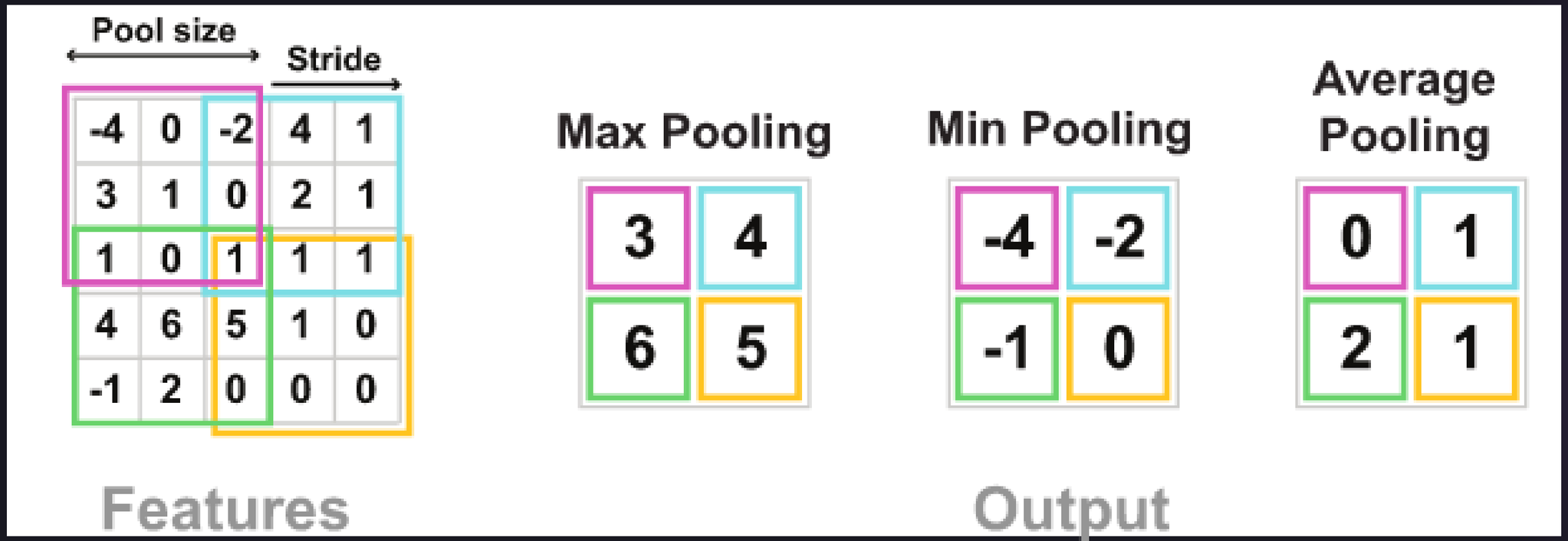
Interpretability techniques are normally studied in isolation. We explore the powerful interfaces that arise when you combine them -- and the rich structure of this combinatorial space.

Distill

Theory

CNN

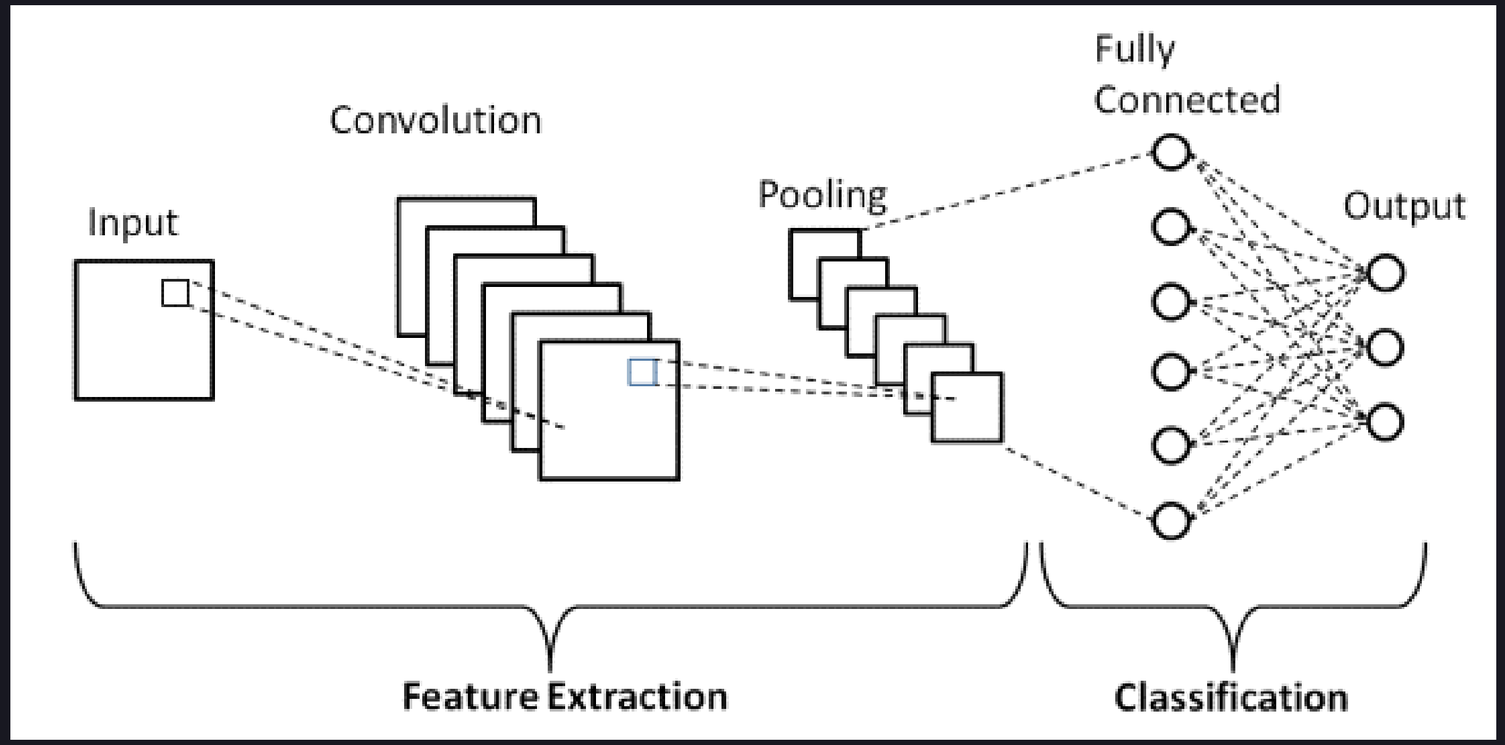
Pooling



Theory

CNN

CNN Architecture



Theory

CNN

Advantages

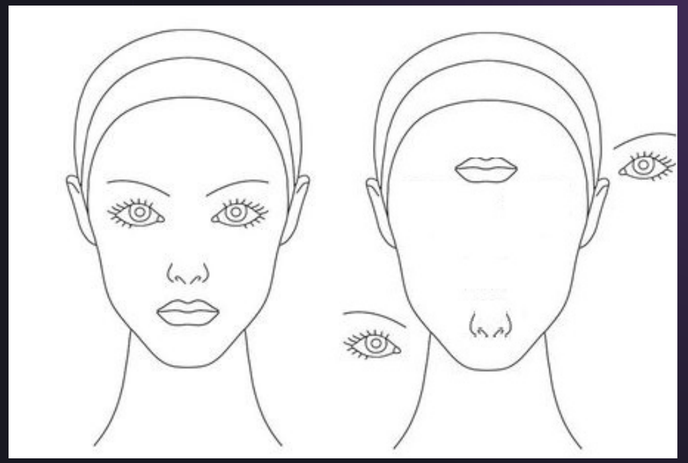
1. local connectivity
2. shared weights (n filters per convolution layer)
3. pooling reduces #weights that need to be learned

Theory

CNN

Disadvantages

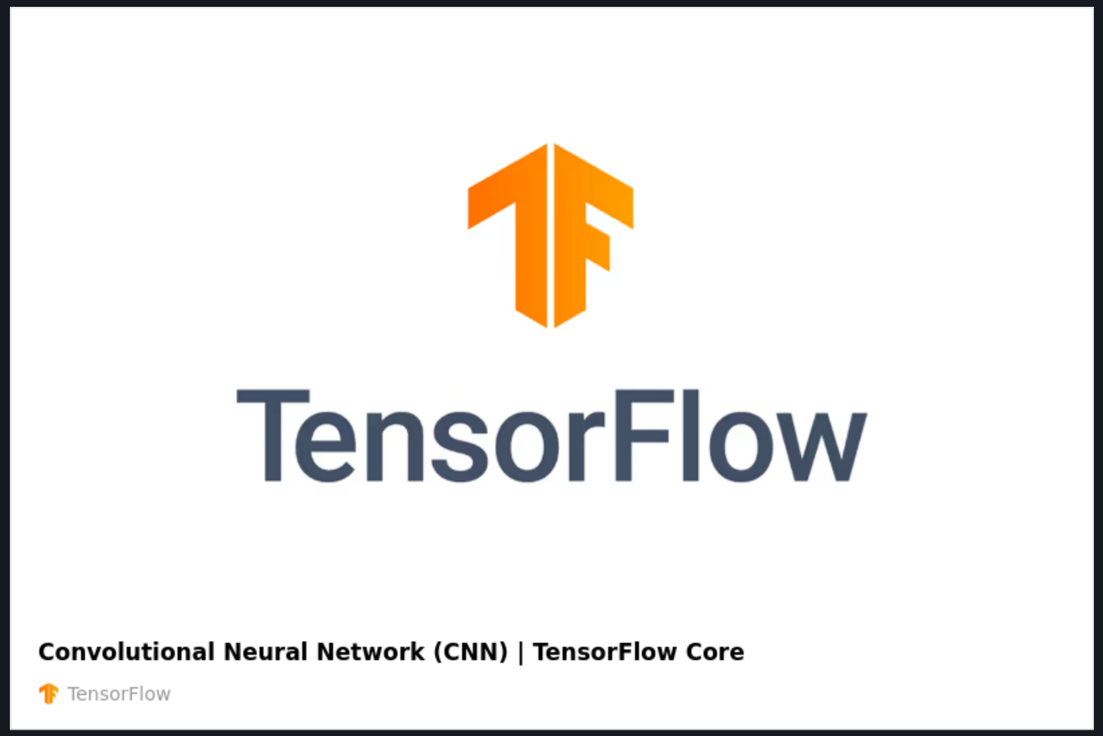
1. NOT invariant of Orientation(translation, angle) and pixel values (colour, intensities).
2. Relative position of components of an image
3. Classifying multiple objects in a single image



Theory

CNN

CODE



QUESTIONS?

IT'S TIME FOR THE NEXT CHALLENGE

PREVIOUS CHALLENGE

UCLAIS Challenge 1: Stroke Prediction

For the first DOXA challenge of the UCLAIS tutorial series, you are being challenged to predict whether someone has suffered a brain stroke or not by applying some of the machine learning techniques you have learned over the past few tutorial lectures.

Finished · 17 entrants

OVERVIEW

SCOREBOARD

SUBMISSION GUIDE

JUPYTER NOTEBOOK

UCLAIS WEBSITE

Scoreboard

Search

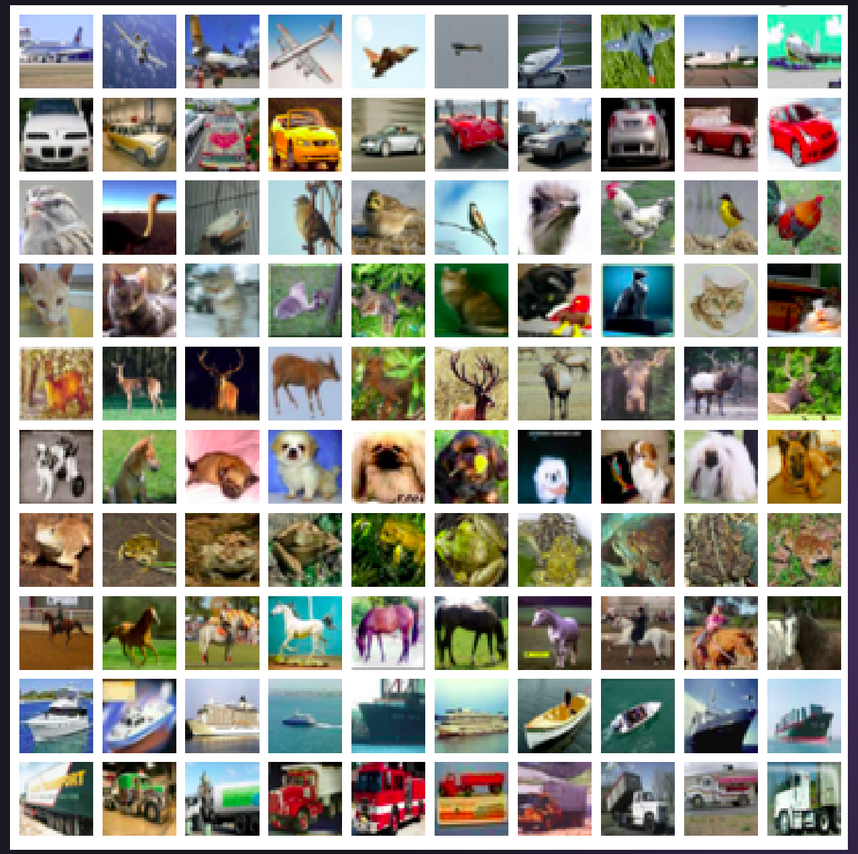
#	Participant	Submitted	Accuracy
1	@vedal	6 days ago	0.98752
2	@Rosasinensis	2 weeks ago	0.93762
3	@maria	2 weeks ago	0.93762
4	@jakedorman	2 weeks ago	0.93762
5	@Aaaashvin	2 weeks ago	0.93762
6	@chloemt03	1 week ago	0.93762
7	@Gen1Code	1 week ago	0.93762
8	@andrew	1 week ago	0.93762
9	@FranekNowak	5 days ago	0.93762
10	@y_kimz	4 days ago	0.93762
11	@louiskwok	3 days ago	0.93762
12	@louisdewardt	2 days ago	0.93762
13	@charlieharrison943	1 day ago	0.93762
14	@Zaki	22 hours ago	0.93762
15	@Polly2327	3 hours ago	0.93762
16	@ladams	5 days ago	0.93555
17	@asubedi	6 days ago	0.91060

CONGRATS!

UCLAIS DOXA Challenge 2

IMAGE CLASSIFICATION - CIFAR10

CIFAR-10 is an established computer-vision dataset used for object recognition. It is one of the most widely used datasets for machine learning research



Challenge

Predict the corresponding class of a given image. There are a total of 10 classes that we need to predict from.

CLASS

Cat, Dog, Truck, Bird, Airplane, Ship,
Frog, Horse, Deer, Automobile

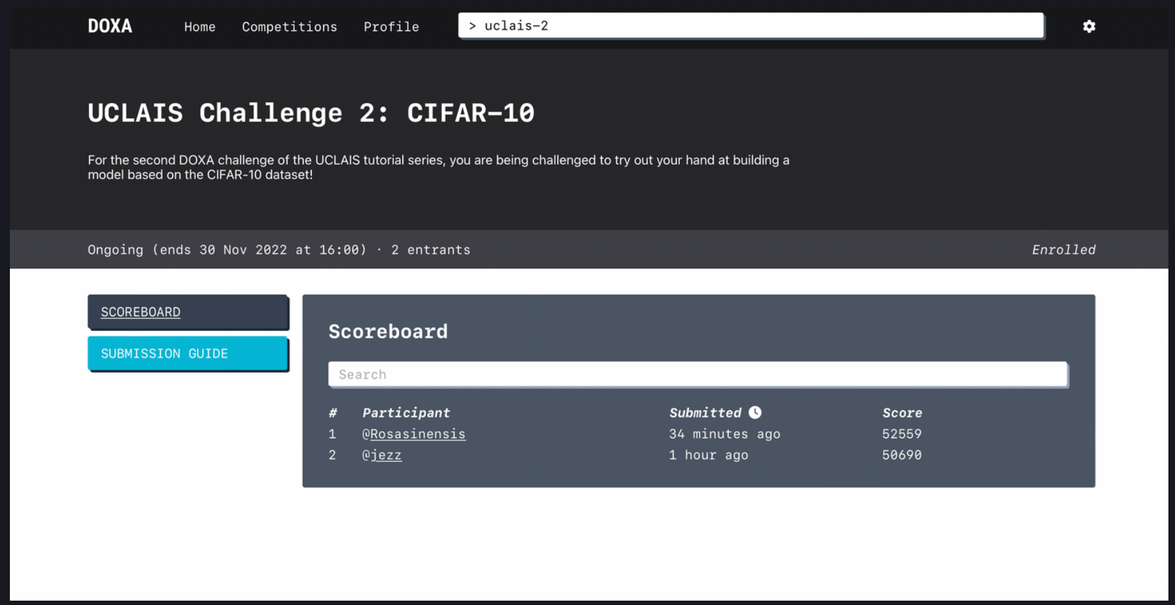
Training Set:

Small Dataset: 15 000 images; 1 500 images for each class
Large Dataset: 50 000 images; 5 000 images for each class

Test Set: 10,000 images (BUT, there is no need to worry about this!)

HOW TO TAKE PART

1. Sign up for an account on **doxaai.com**
2. Enrol for the competition at **https://doxaai.com/competition/uclais-2**
3. Open the Jupyter notebook in Google Colab (**https://colab.research.google.com/**)
4. Run all the cells (making sure to approve access for the DOXA CLI submission tool)
5. Find yourself on the DOXA scoreboard!
6. Improve on the model you were provided with using some of the ideas you have learned over the past few weeks.



DOXA: https://doxaai.com/
GitHub: https://github.com/UCLAIS/doxa-challenges/tree/main/Challenge-2
Notebook: https://github.com/UCLAIS/doxa-challenges/blob/main/Challenge-2/starter.ipynb

A DEMO 

SEE YOU NEXT TIME