

# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Welcome to Modern Computer Vision™

A Comprehensive Course on Computer Vision for 2022 onwards!

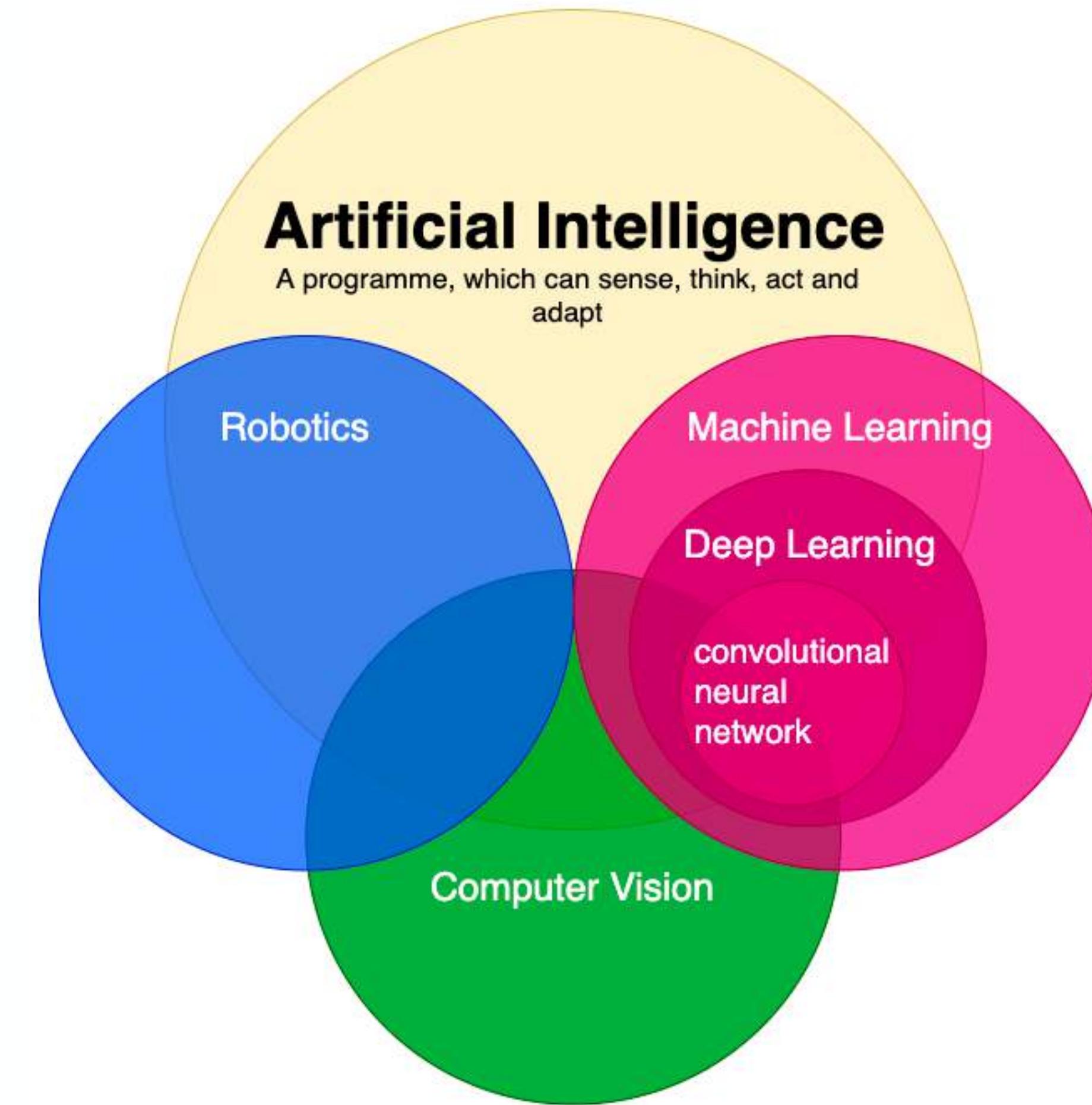
# So what exactly is Computer Vision?

“An interdisciplinary field that aims to enable computers to gain **understanding of what is being seen in** images and videos.”



Source - Terminator 2: Judgement Day (1991)

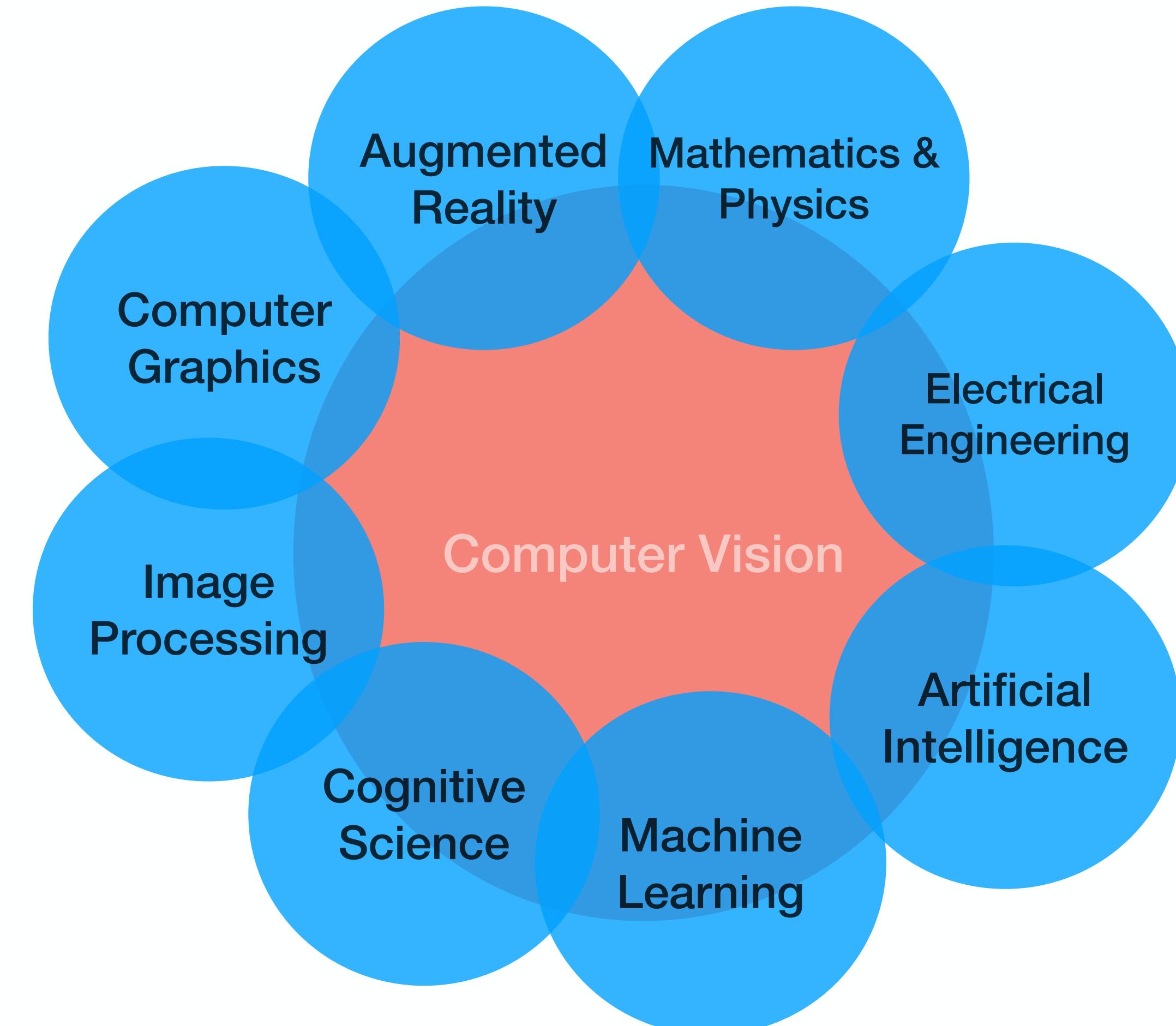
# Is Computer Vision Artificial Intelligence?



Relation between Artificial Intelligence, Machine Learning and Deep Learning, Computer Vision.

[https://www.researchgate.net/figure/Relation-between-Artificial-Intelligence-Machine-Learning-and-Deep-Learning-Computer\\_fig1\\_342978934](https://www.researchgate.net/figure/Relation-between-Artificial-Intelligence-Machine-Learning-and-Deep-Learning-Computer_fig1_342978934)

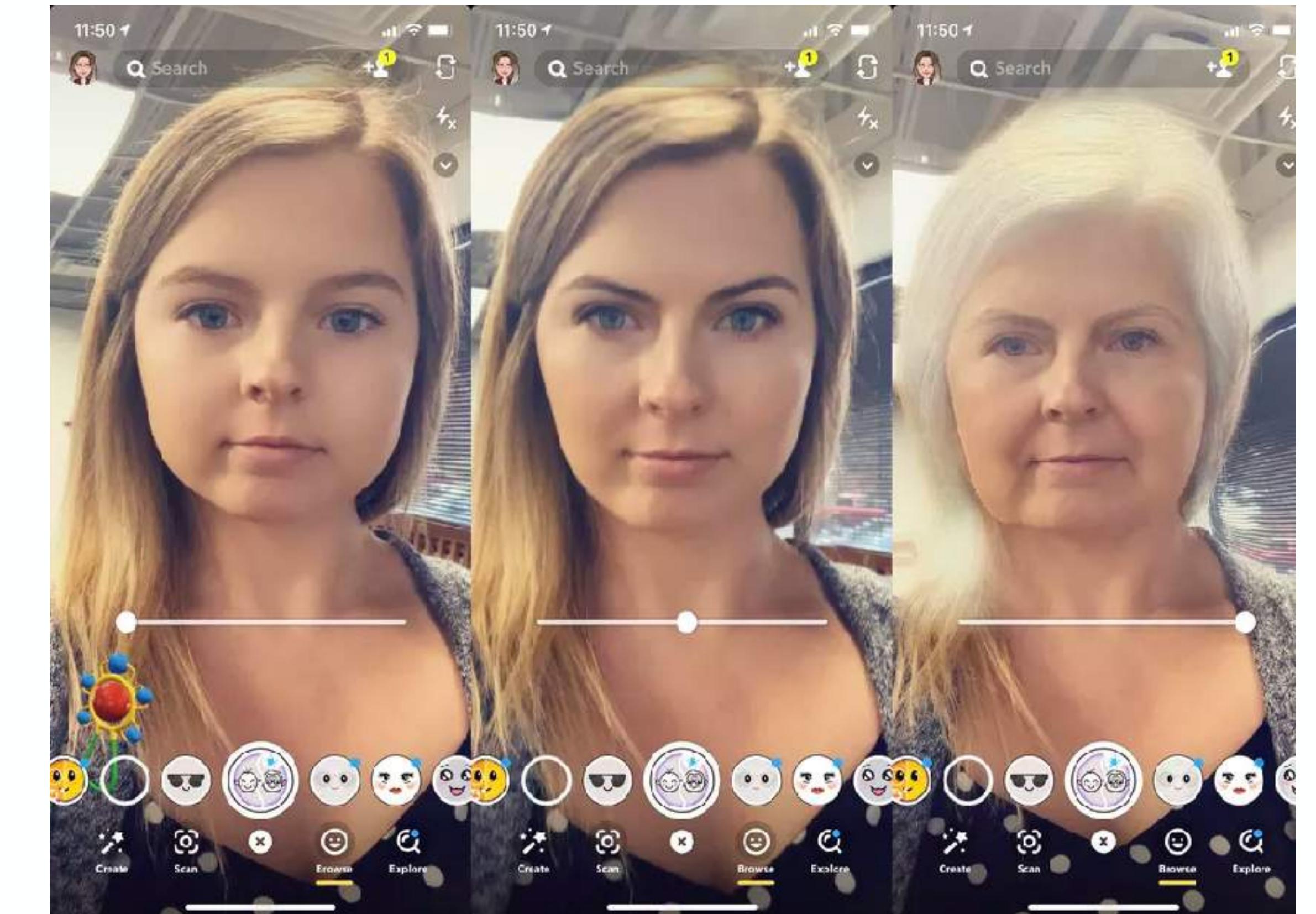
# Computer Vision is an amalgamation of many fields



# So what can Computer Vision do?

## You might be familiar with these...

- Snapchat and Instagram filters
- Optical Character Recognition (OCR)
- Licence Plate Reading
- Self-driving cars
- Sporting Analysis
- Facial Recognition

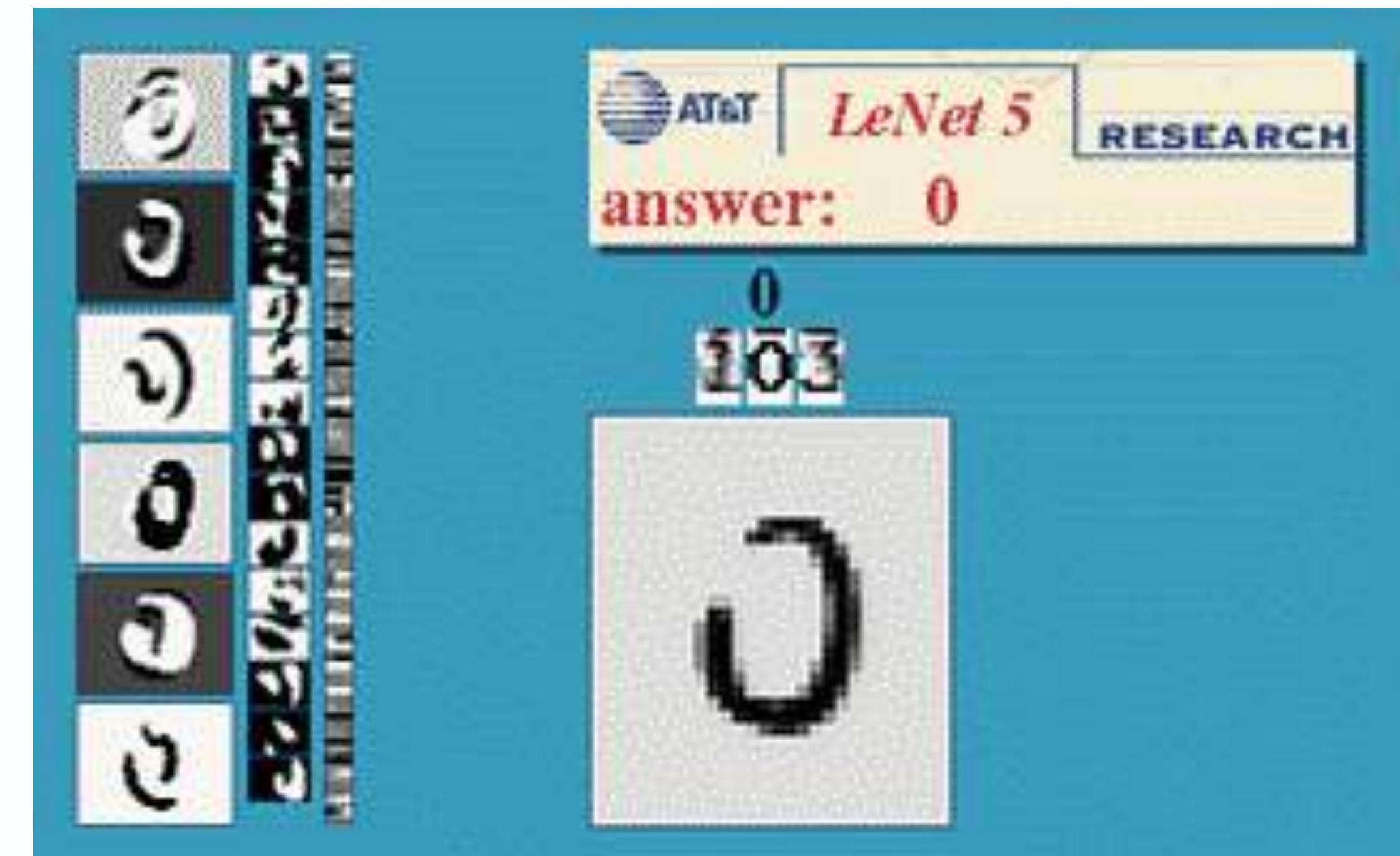


Source - Cnet - Snapchat's Time Machine AR lens creepily shows what you'll look like old

# So what can Computer Vision do?

You might be familiar with these...

- Snapchat and Instagram filters
- **Optical Character Recognition (OCR)**
- Licence Plate Reading
- Self-driving cars
- Sporting Analysis
- Facial Recognition



Source -AT&T's LeNet OCR for Handwritten Digits

# So what can Computer Vision do?

You might be familiar with these...

- Snapchat and Instagram filters
- Optical Character Recognition (OCR)
- **Licence Plate Reading**
- Self-driving cars
- Sporting Analysis
- Facial Recognition



# So what can Computer Vision do?

You might be familiar with these...

- Snapchat and Instagram filters
- Optical Character Recognition (OCR)
- Licence Plate Reading
- **Self-driving cars**
- Sporting Analysis
- Facial Recognition



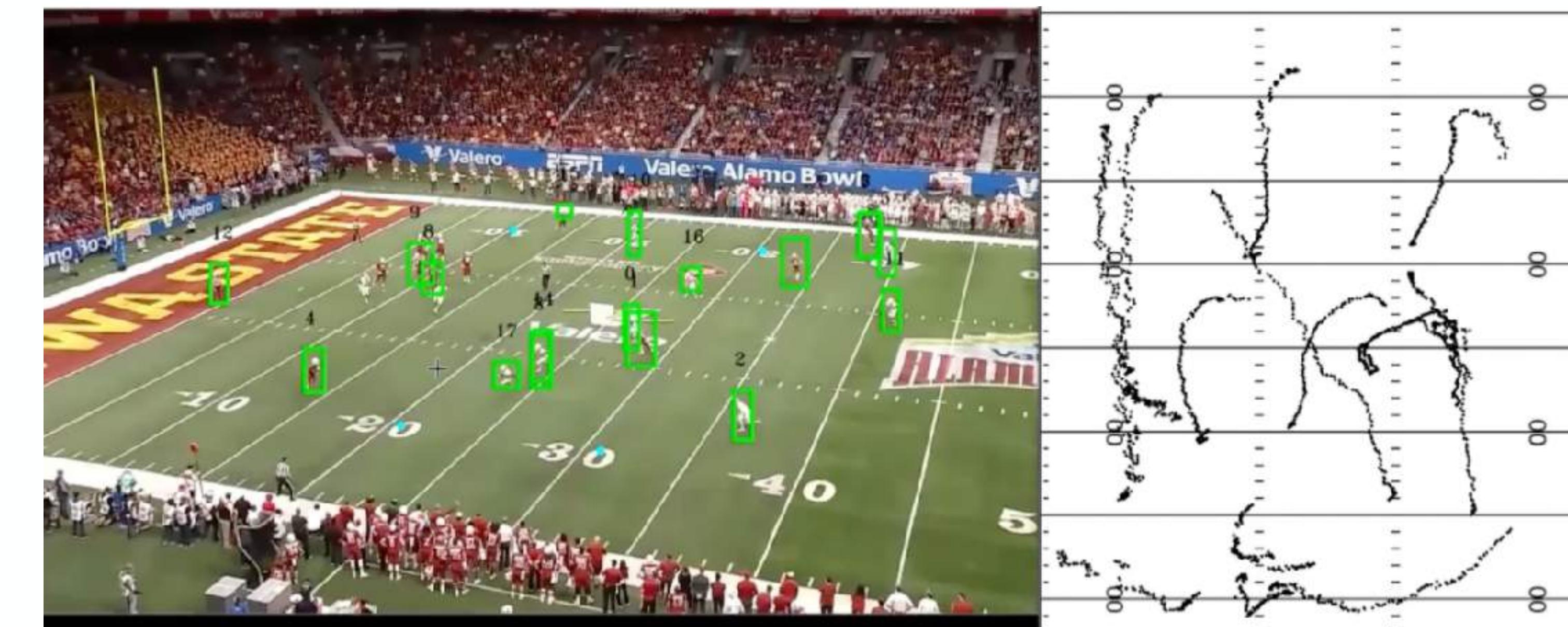
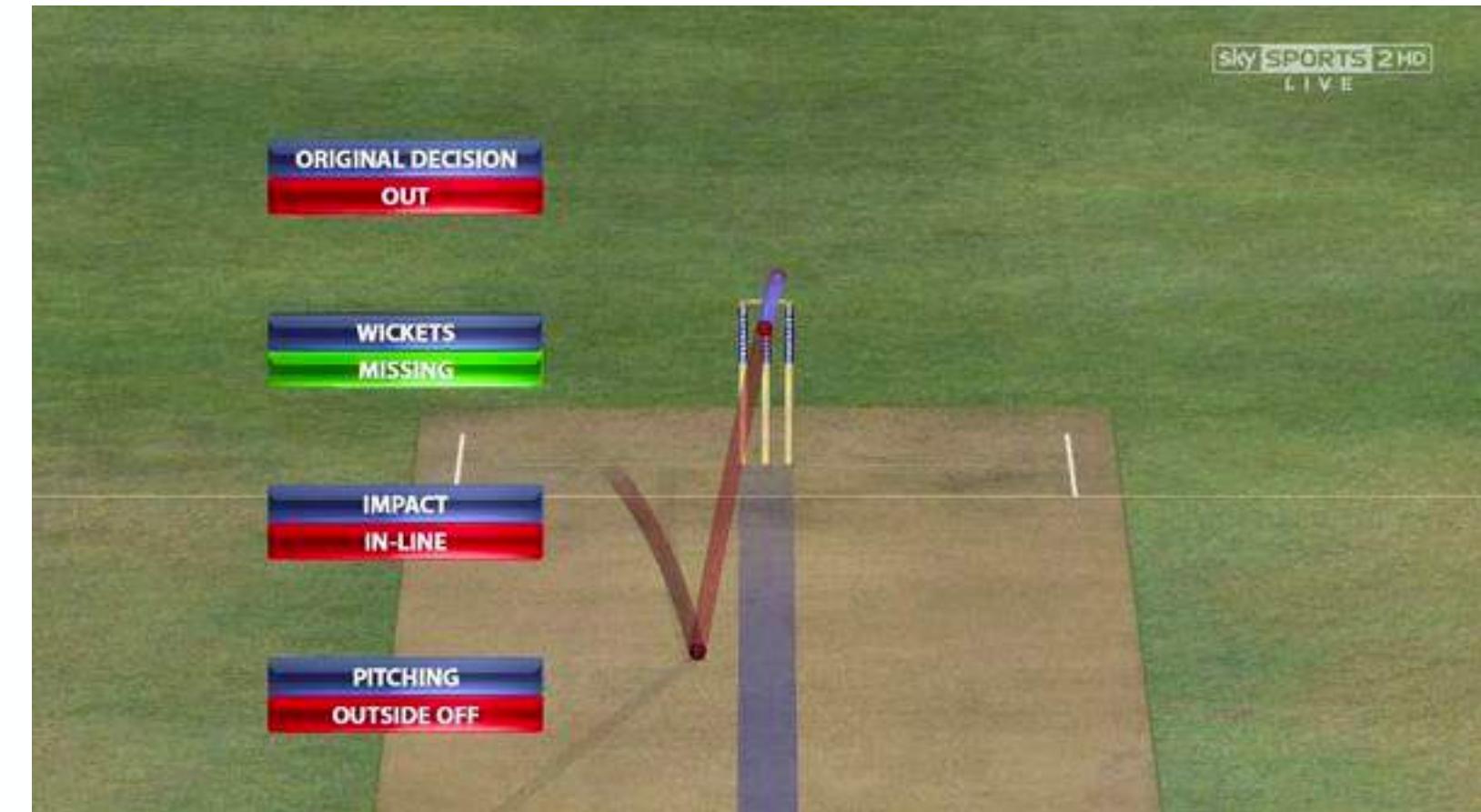
Source Inc. - The Tech That Powers Your Self-Driving Car Might  
Be Built Using People Playing Games on Their Phones

# So what can Computer Vision do?

You might be familiar with these...

- Snapchat and Instagram filters
- Optical Character Recognition (OCR)
- Licence Plate Reading
- Self-driving cars
- **Sporting Analysis**
- Facial Recognition

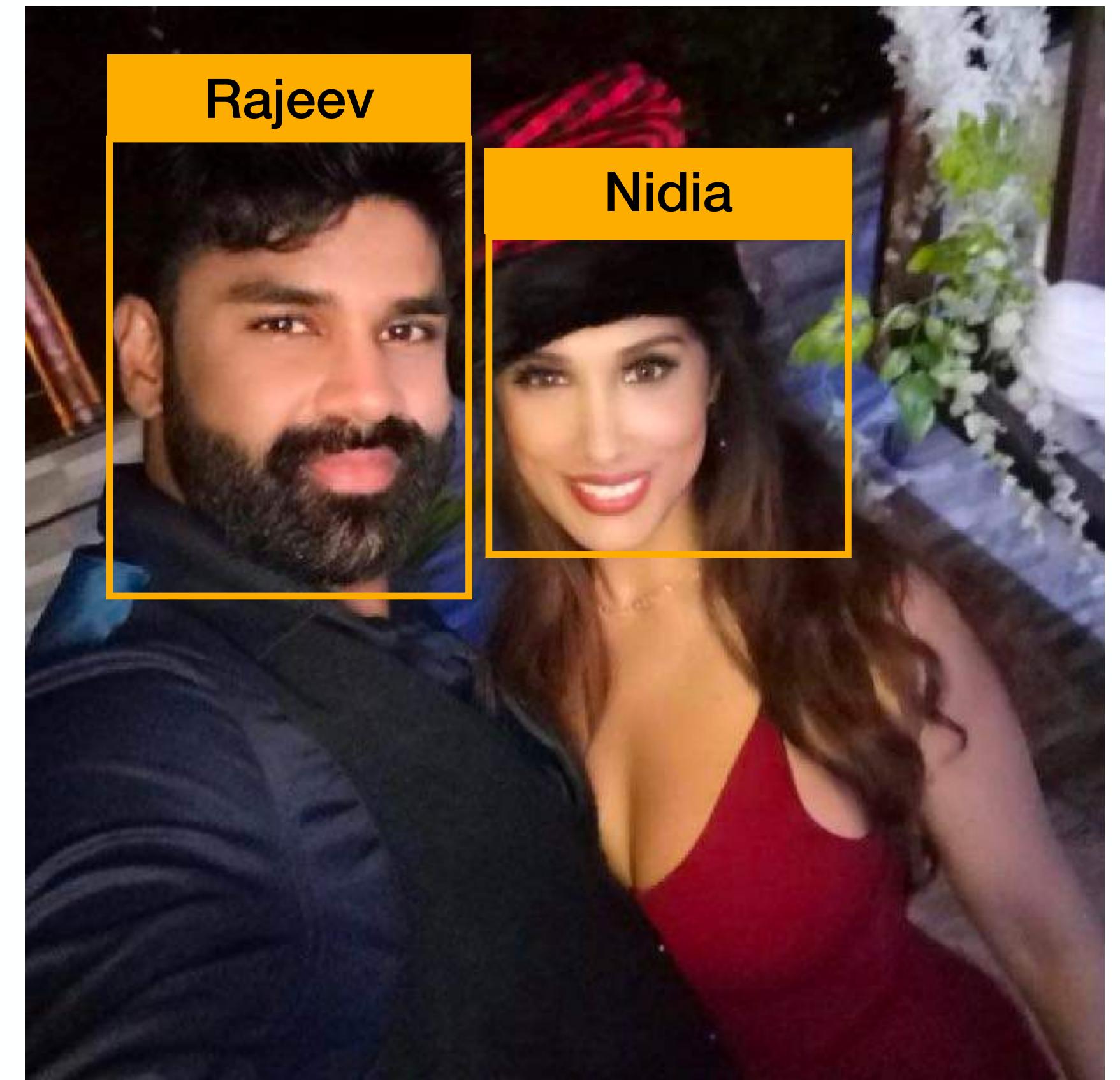
HawkEye in Cricket



# So what can Computer Vision do?

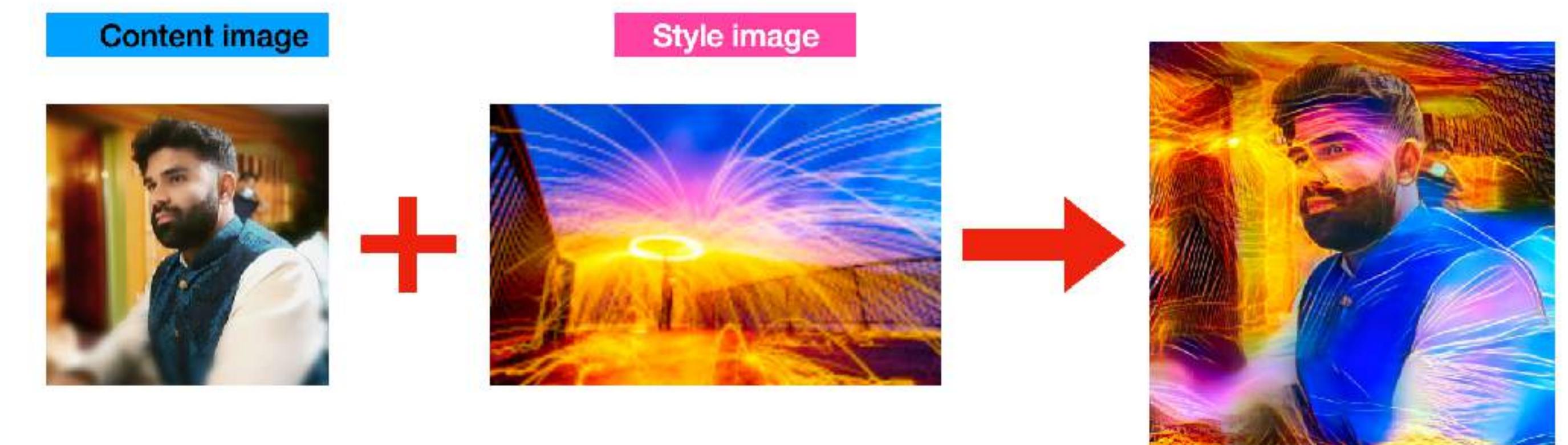
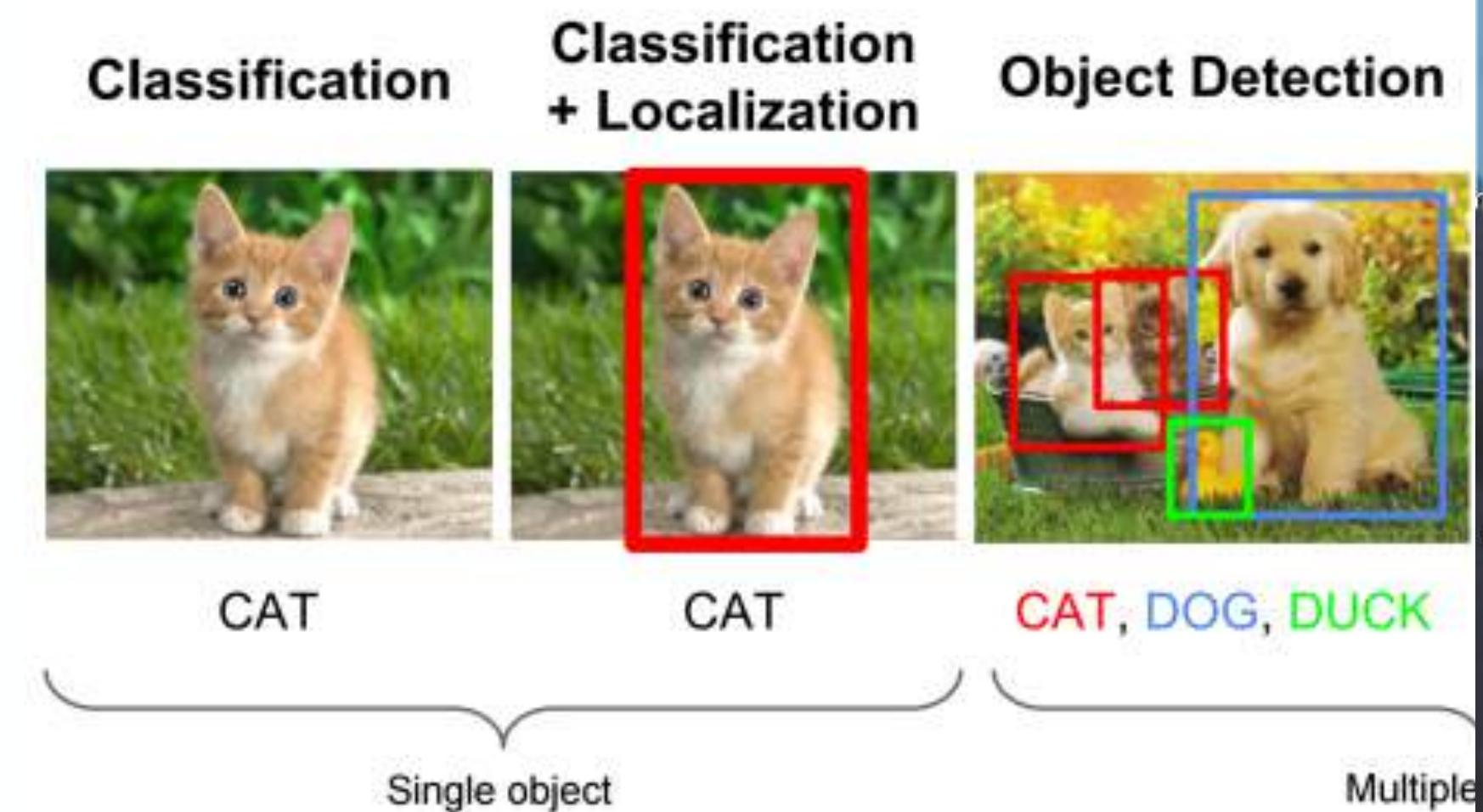
You might be familiar with these...

- Snapchat and Instagram filters
- Optical Character Recognition (OCR)
- Licence Plate Reading
- Self-driving cars
- Sporting Analysis
- **Facial Recognition**



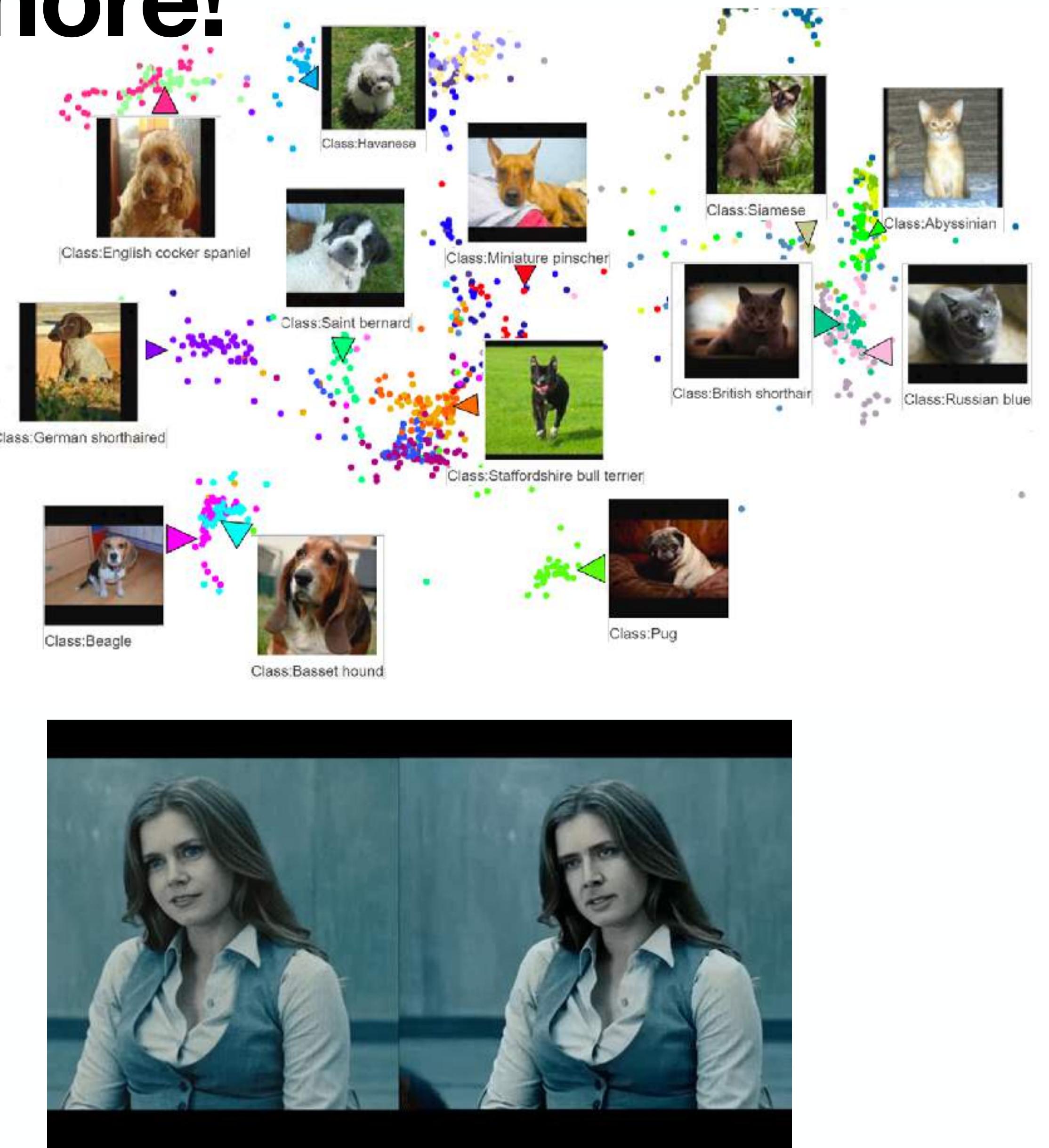
# But there's so much more!

- **Image Recognition**
- **Object Detection**
- **Segmentation**
- **AI Art**
- **Image Similarity**
- **Deep Fakes**
- **Body Pose Detection**
- **Image Generation**



# But there's so much more!

- Image Recognition
- Object Detection
- Segmentation
- AI Art
- **Image Similarity**
- **Deep Fakes**
- Body Pose Detection
- Image Generation



# But there's so much more!

- Image Recognition
- Object Detection
- Segmentation
- AI Art
- Image Similarity
- Deep Fakes
- **Body Pose Detection**
- **Image Generation**



# Computer Vision Applications are endless!

Electric Scooter ID

## Gas Leak Detection

Document Digitization

Plant Phenotyping

Flare Stack Monitoring

Resume Parsing

Augmented Reality

## Weed Detection

Microscopy

Bean Counting

Garbage Cleanup

Drone Video Analysis

## Conveyer Belt Debris

Traffic Counter

Pothole Identification

Soccer Player Tracker

Steelyard Throughput

Security Cam Analysis

Self Driving Cars

## Fish Measuring

Remote Tech Support

Tennis Line Tracking

Know Your Customer

Endangered Species Tracking

Inventory Management

## Hard Hat Detection

Pest Identification

OCR Math

Basketball Shot Tracking

Logo Identification

## Satellite Imagery

Traffic Cone Finder

## Airplane Maintenance

Tumor Detection

D&D Dice Counter

Plant Disease Finder

X-Ray Analysis

## Roof Damage Estimator

City Bus Tracking

Board Game Helpers

Dental Cavity Detection

Drought Tracking

Hog Confinements

## Sushi Identifier

Oil Storage Estimator

Car Wheel Finder

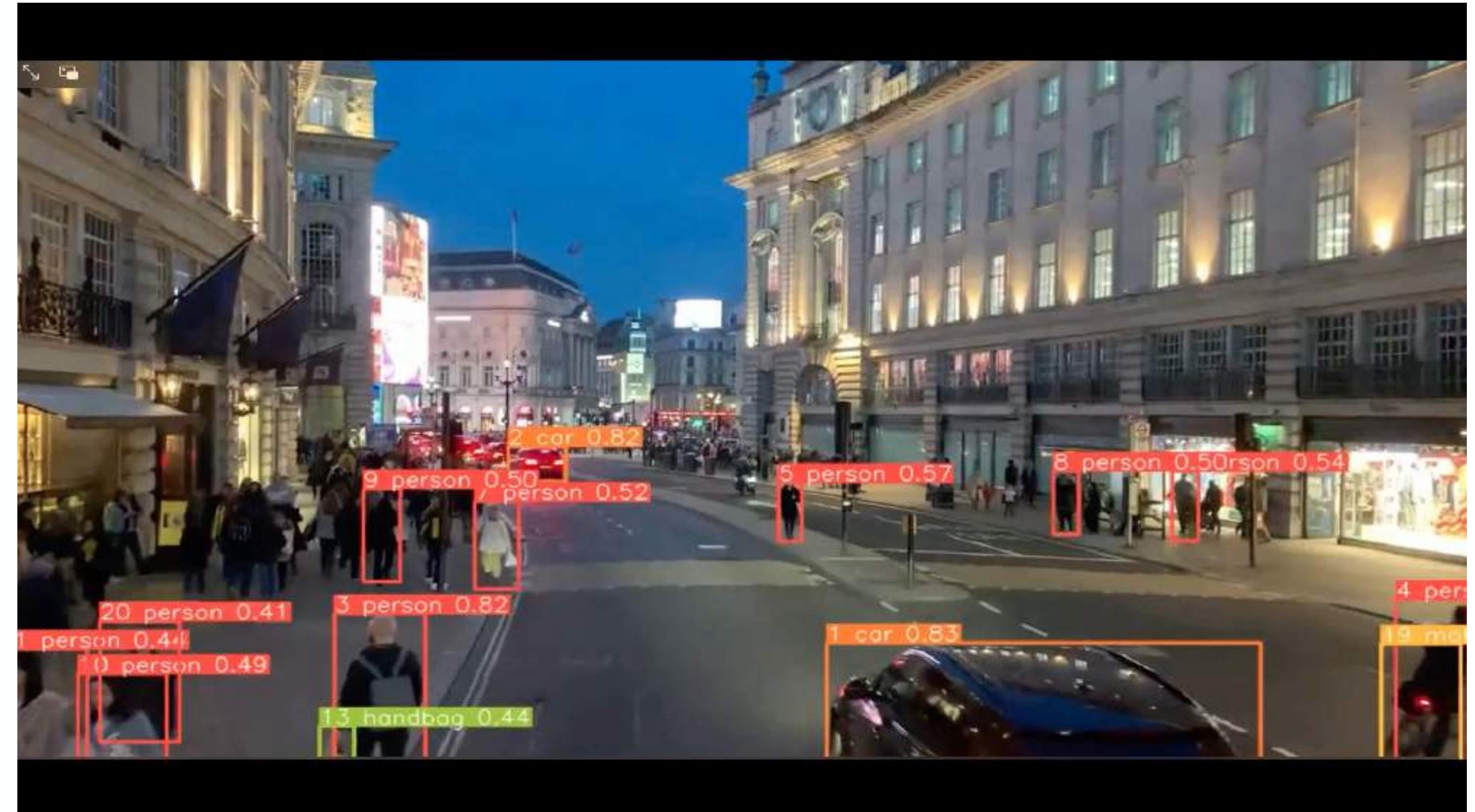
License Plate Reader

Exercise Counter

# Why do this course?

What are you going to learn exactly?

- Foundation in **Classical Computer Vision** with **OpenCV**
- **Deep Learning** applied to Computer Vision
  - PyTorch
  - TensorFlow Keras



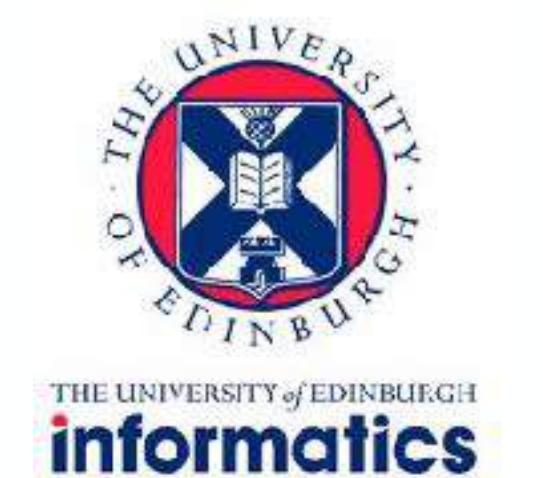
# Who should do this course?

- **College Students** (BSc., MSc or PhD) looking to start understanding and implementing computer vision applications
- **High School Students** or hobbyists
- **Software Engineers/Developers & Data Scientists** looking to get into Computer Vision



# About me

- **Electrical & Computer Engineer** with **7 years** experience as a Radio Frequency Engineer
- **MSc in Artificial Intelligence** from the **University of Edinburgh**
- **6 years experience** working in **Computer Vision**
- Worked at several **London startups** and even co-founded my own company
- Created several **Udemy courses** in Computer Vision
- Now **Senior Computer Vision Engineer**



# My Udemy Courses

## Instructor Rating of 4.3/5

**INSTRUCTOR**

### Rajeev D. Ratan

Data Scientist, Computer Vision Expert & Electrical Engineer

Total students    Reviews

**49,022**    **7,559**


**About me**

Hi I'm Rajeev, a **Data Scientist**, and **Computer Vision Engineer**.

I have a BSc in Computer & Electrical Engineering and an **MSc in Artificial Intelligence from the University of Edinburgh** where I gained extensive knowledge of machine learning, computer vision, and intelligent robotics.

I have published research on using data-driven methods for **Probabilistic Stochastic Modeling for Public Transport** and even was part of a group that won a robotics competition at the University of Edinburgh.

I launched my own computer vision startup that was based on using deep learning in education since then I've been contributing to 2 more startups in computer vision domains and one multinational company in Data Science.

Previously, I worked for 8 years at two of the Caribbean's largest telecommunication operators where he gained experience in managing technical staff and deploying complex telecommunications projects.

Show less ^

Facebook

LinkedIn

**My courses (7)**

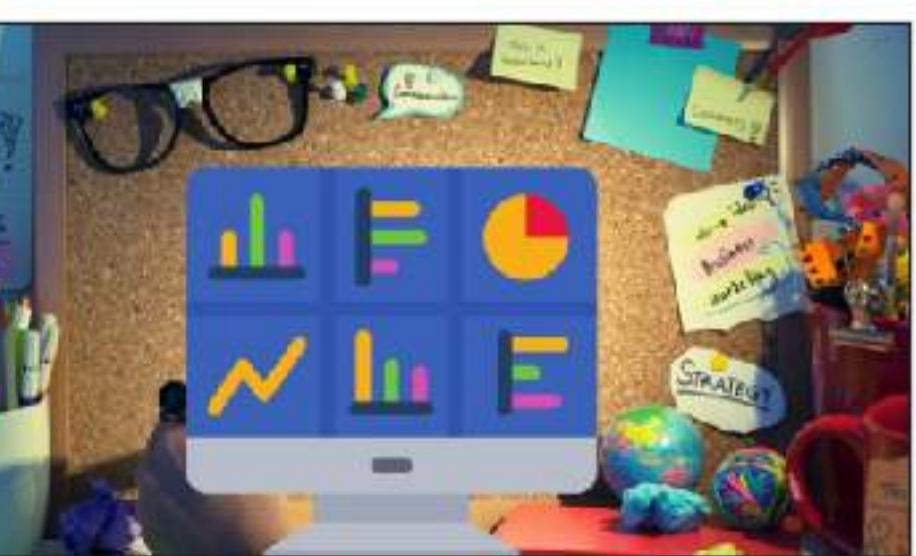

#### Deep Learning Computer Vision™ CNN, OpenCV, YOLO, SSD & GANs

Rajeev D. Ratan

**4.3 ★★★★★** (1,979)

14.5 total hours • 178 lectures • Intermediate

**£21.99** £79.99



#### Data Science, Analytics & AI for Business & the Real World™

Rajeev D. Ratan, Nidia Sahjara

**4.4 ★★★★★** (316)

30.6 total hours • 248 lectures • Beginner

**£47.99** £174.99

Bestseller



#### Data Science & Deep Learning for Business™ 20 Case Studies

Rajeev D. Ratan, Nidia Sahjara

**4.5 ★★★★★** (867)

21 total hours • 190 lectures • All Levels

**£15.99** £59.99

Bestseller



#### Master Computer Vision™ OpenCV4 in Python with Deep Learning

Rajeev D. Ratan

**4.2 ★★★★★** (3,506)

11 total hours • 116 lectures • All Levels

**£15.99** £59.99

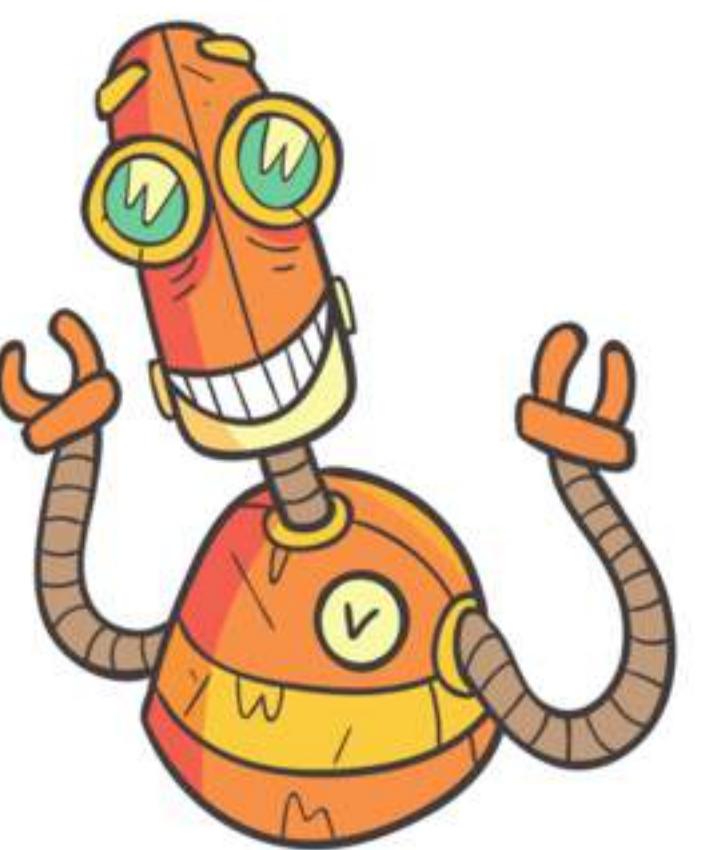
# Why did I make this Course?

- My previous courses launched in (2016 & 2018) were good but had many deficiencies. This course aims to fix everything that was wrong or broken in my older courses.
  - Taught using **Google Colab** (no messy installs and Virtual Machine setups)
  - Code and methodology are **up to date**
  - Covers **all key areas** in Computer Vision
  - Taught using both **TensorFlow Keras** and **PyTorch**
  - Includes both **OpenCV** and **Deep Learning** Modules in **One Single** 27+ hour course!



# Requirements?

- **Internet connection** (preferably at least 2mb) and web browser
- **High school level math** knowledge (preferred, not required)
- **Basic programming** knowledge is **not required** but preferred (especially if it's Python)
- **Enthusiasm** about Artificial Intelligence!



# High Level Overview

## OpenCV - Classical Computer Vision Outline

- 1.Image Manipulations
- 2.Segmentation
- 3.Feature Extraction
- 4.Object Detection with Haar Cascade Classifiers
- 5.Image analysis and Transformations
- 6.Background Removal
- 7.Tracking - Optical Flow and MeanShift
- 8.Face Swapping
- 9.OCR and text detection
- 10.QR and Barcode reading
- 11.YOLOv3 & SSDs in OpenCV
- 12.Neural Style Transfer
- 13.Colorising Black and White Photos
- 14.Computation Photography (noise removal, in-painting)
- 15.Facial Recognition
- 16.Working with Video and Video Stream

# High Level Overview

## Deep Learning Outline

- 1. Introduction to Deep Learning in Computer Vision
- 2. Deep Learning Basics with PyTorch and TensorFlow Keras
- 3. Comparing Libraries and Analysing Performance
- 4. Regularisation and Overfitting
- 5. Understanding What CNN's See
- 6. Designing CNNs and Overview of Modern CNN Architectures
- 7. PyTorch Lightning
- 8. Transfer Learning
- 9. Google DeepDream
- 10. Neural Style Transfer
- 11. Autoencoders
- 12. Generative Adversarial Networks
- 13. Siamese Networks
- 14. Facial Recognition
- 15. Object Detection
- 16. Deep Segmentation
- 17. Tracking with Deep Learning
- 18. Vision Transformers
- 19. Depth Estimation
- 20. Video Classification
- 21. Point Cloud Classification and Segmentation
- 22. 3D Image Classification
- 23. OCR Readers
- 24. Image Captioning
- 25. Computer Vision APIs

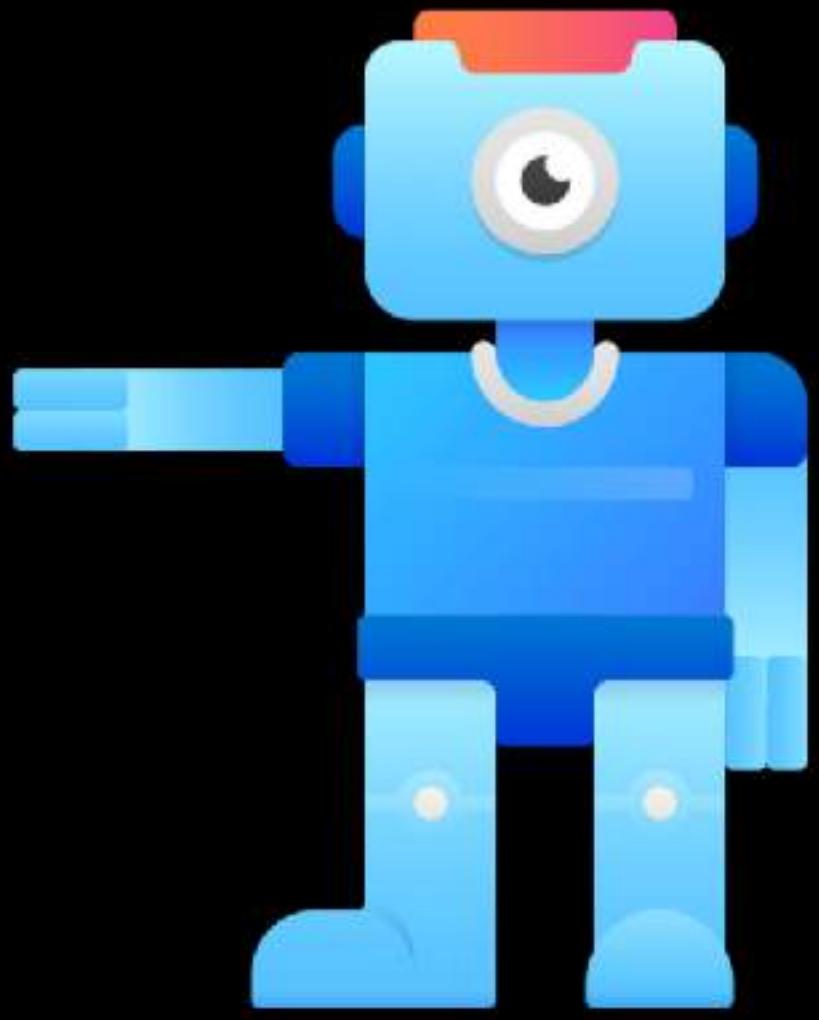


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

## Course Outline Overview



# MODERN COMPUTER VISION

BY RAJEEV RATAN

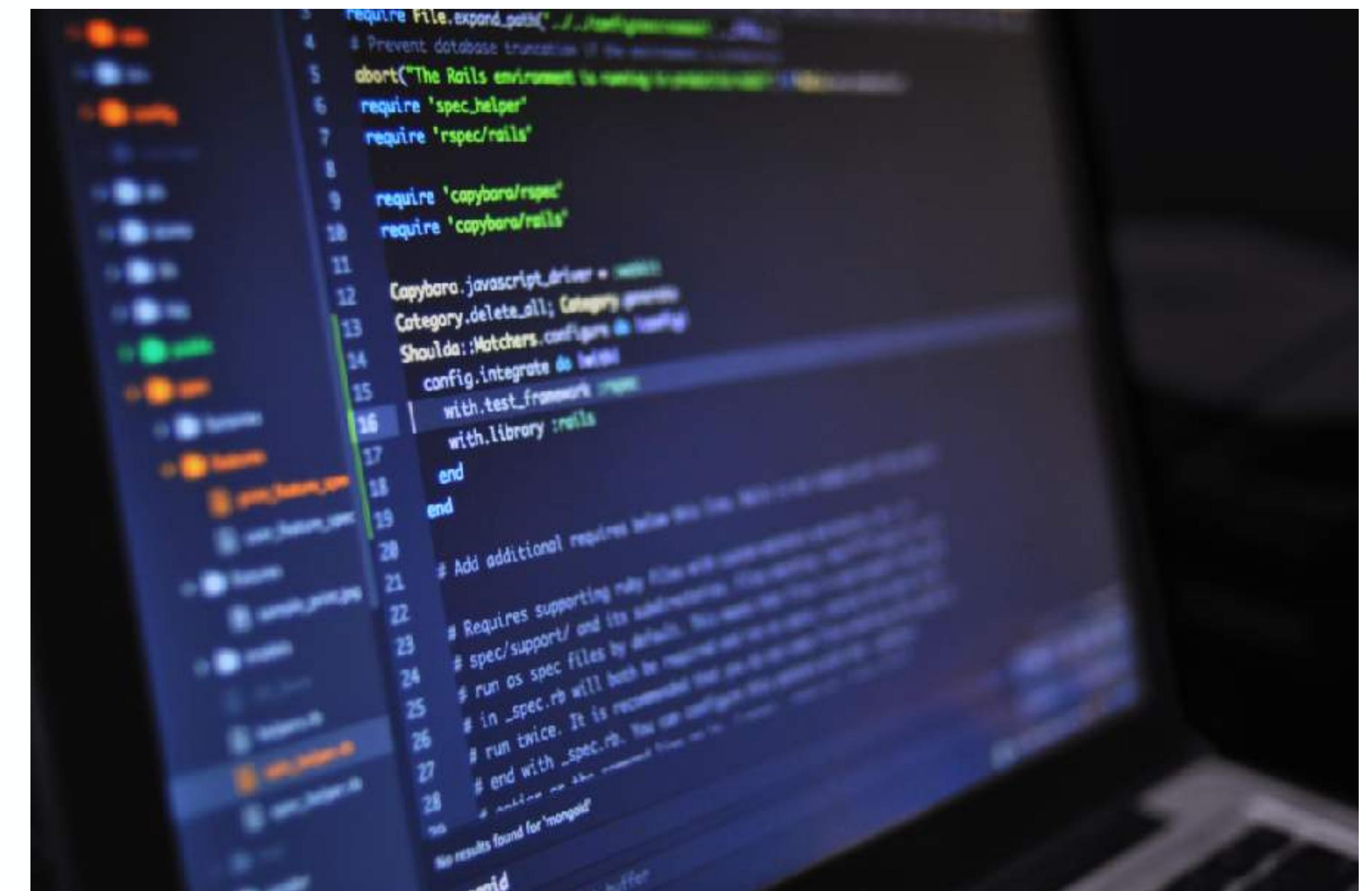
## Course Overview

This is a BIG course! Let's get started!

# So how do we do Computer Vision?

## What makes it possible?

- We need tools! Namely a programming language
- Many exist such as:
  - Matlab
  - C++ & Java
  - **Python**



```
1 require File.expand_path('../..', __FILE__)
2 # Prevent database truncation if the environment is test or
3 # test杠
4 abort("The Rails environment is running in production mode!
5 # This will have the adverse effect of
6 # making your test slower. You can
7 # change this behavior by changing
8 # RAILS_ENV = 'test' in config/environments/production.rb
9 # or you can turn off tests entirely
10 # by changing
11 # TEST杠 = false in config/environments/production.rb
12 # Note that this will mean that
13 # test杠 will be disabled for all
14 # environments on your machine
15 # It's not recommended to turn
16 # this off unless you know what
17 # you're doing
18 # Add additional require statements below this line
19 # Requires supporting files within the
20 # same directory as this file or,
21 # if you're dynamic, from a
22 # relative path, like
23 # require 'foo/bar/baz'
24 # run as spec_files by default, or
25 # in _spec.rb will both be
26 # # run twice. It is
27 # # end with _spec.rb. You
28 # # can also make it
29 # # run once by
30 # # require 'spec/spec'
31 # # run as
32 # # in _spec.rb will both be
33 # # run twice. It is
34 # # end with _spec.rb. You
35 # # can also make it
36 # # run once by
37 # # require 'spec/spec'
38 # # run as
39 # # in _spec.rb will both be
40 # # run twice. It is
41 # # end with _spec.rb. You
42 # # can also make it
43 # # run once by
44 # # require 'spec/spec'
45 # # run as
46 # # in _spec.rb will both be
47 # # run twice. It is
48 # # end with _spec.rb. You
49 # # can also make it
50 # # run once by
51 # # require 'spec/spec'
52 # # run as
53 # # in _spec.rb will both be
54 # # run twice. It is
55 # # end with _spec.rb. You
56 # # can also make it
57 # # run once by
58 # # require 'spec/spec'
59 # # run as
60 # # in _spec.rb will both be
61 # # run twice. It is
62 # # end with _spec.rb. You
63 # # can also make it
64 # # run once by
65 # # require 'spec/spec'
66 # # run as
67 # # in _spec.rb will both be
68 # # run twice. It is
69 # # end with _spec.rb. You
70 # # can also make it
71 # # run once by
72 # # require 'spec/spec'
73 # # run as
74 # # in _spec.rb will both be
75 # # run twice. It is
76 # # end with _spec.rb. You
77 # # can also make it
78 # # run once by
79 # # require 'spec/spec'
80 # # run as
81 # # in _spec.rb will both be
82 # # run twice. It is
83 # # end with _spec.rb. You
84 # # can also make it
85 # # run once by
86 # # require 'spec/spec'
87 # # run as
88 # # in _spec.rb will both be
89 # # run twice. It is
90 # # end with _spec.rb. You
91 # # can also make it
92 # # run once by
93 # # require 'spec/spec'
94 # # run as
95 # # in _spec.rb will both be
96 # # run twice. It is
97 # # end with _spec.rb. You
98 # # can also make it
99 # # run once by
100 # # require 'spec/spec'
101 # # run as
102 # # in _spec.rb will both be
103 # # run twice. It is
104 # # end with _spec.rb. You
105 # # can also make it
106 # # run once by
107 # # require 'spec/spec'
108 # # run as
109 # # in _spec.rb will both be
110 # # run twice. It is
111 # # end with _spec.rb. You
112 # # can also make it
113 # # run once by
114 # # require 'spec/spec'
115 # # run as
116 # # in _spec.rb will both be
117 # # run twice. It is
118 # # end with _spec.rb. You
119 # # can also make it
120 # # run once by
121 # # require 'spec/spec'
122 # # run as
123 # # in _spec.rb will both be
124 # # run twice. It is
125 # # end with _spec.rb. You
126 # # can also make it
127 # # run once by
128 # # require 'spec/spec'
129 # # run as
130 # # in _spec.rb will both be
131 # # run twice. It is
132 # # end with _spec.rb. You
133 # # can also make it
134 # # run once by
135 # # require 'spec/spec'
136 # # run as
137 # # in _spec.rb will both be
138 # # run twice. It is
139 # # end with _spec.rb. You
140 # # can also make it
141 # # run once by
142 # # require 'spec/spec'
143 # # run as
144 # # in _spec.rb will both be
145 # # run twice. It is
146 # # end with _spec.rb. You
147 # # can also make it
148 # # run once by
149 # # require 'spec/spec'
150 # # run as
151 # # in _spec.rb will both be
152 # # run twice. It is
153 # # end with _spec.rb. You
154 # # can also make it
155 # # run once by
156 # # require 'spec/spec'
157 # # run as
158 # # in _spec.rb will both be
159 # # run twice. It is
160 # # end with _spec.rb. You
161 # # can also make it
162 # # run once by
163 # # require 'spec/spec'
164 # # run as
165 # # in _spec.rb will both be
166 # # run twice. It is
167 # # end with _spec.rb. You
168 # # can also make it
169 # # run once by
170 # # require 'spec/spec'
171 # # run as
172 # # in _spec.rb will both be
173 # # run twice. It is
174 # # end with _spec.rb. You
175 # # can also make it
176 # # run once by
177 # # require 'spec/spec'
178 # # run as
179 # # in _spec.rb will both be
180 # # run twice. It is
181 # # end with _spec.rb. You
182 # # can also make it
183 # # run once by
184 # # require 'spec/spec'
185 # # run as
186 # # in _spec.rb will both be
187 # # run twice. It is
188 # # end with _spec.rb. You
189 # # can also make it
190 # # run once by
191 # # require 'spec/spec'
192 # # run as
193 # # in _spec.rb will both be
194 # # run twice. It is
195 # # end with _spec.rb. You
196 # # can also make it
197 # # run once by
198 # # require 'spec/spec'
199 # # run as
200 # # in _spec.rb will both be
201 # # run twice. It is
202 # # end with _spec.rb. You
203 # # can also make it
204 # # run once by
205 # # require 'spec/spec'
206 # # run as
207 # # in _spec.rb will both be
208 # # run twice. It is
209 # # end with _spec.rb. You
210 # # can also make it
211 # # run once by
212 # # require 'spec/spec'
213 # # run as
214 # # in _spec.rb will both be
215 # # run twice. It is
216 # # end with _spec.rb. You
217 # # can also make it
218 # # run once by
219 # # require 'spec/spec'
220 # # run as
221 # # in _spec.rb will both be
222 # # run twice. It is
223 # # end with _spec.rb. You
224 # # can also make it
225 # # run once by
226 # # require 'spec/spec'
227 # # run as
228 # # in _spec.rb will both be
229 # # run twice. It is
230 # # end with _spec.rb. You
231 # # can also make it
232 # # run once by
233 # # require 'spec/spec'
234 # # run as
235 # # in _spec.rb will both be
236 # # run twice. It is
237 # # end with _spec.rb. You
238 # # can also make it
239 # # run once by
240 # # require 'spec/spec'
241 # # run as
242 # # in _spec.rb will both be
243 # # run twice. It is
244 # # end with _spec.rb. You
245 # # can also make it
246 # # run once by
247 # # require 'spec/spec'
248 # # run as
249 # # in _spec.rb will both be
250 # # run twice. It is
251 # # end with _spec.rb. You
252 # # can also make it
253 # # run once by
254 # # require 'spec/spec'
255 # # run as
256 # # in _spec.rb will both be
257 # # run twice. It is
258 # # end with _spec.rb. You
259 # # can also make it
260 # # run once by
261 # # require 'spec/spec'
262 # # run as
263 # # in _spec.rb will both be
264 # # run twice. It is
265 # # end with _spec.rb. You
266 # # can also make it
267 # # run once by
268 # # require 'spec/spec'
269 # # run as
270 # # in _spec.rb will both be
271 # # run twice. It is
272 # # end with _spec.rb. You
273 # # can also make it
274 # # run once by
275 # # require 'spec/spec'
276 # # run as
277 # # in _spec.rb will both be
278 # # run twice. It is
279 # # end with _spec.rb. You
280 # # can also make it
281 # # run once by
282 # # require 'spec/spec'
283 # # run as
284 # # in _spec.rb will both be
285 # # run twice. It is
286 # # end with _spec.rb. You
287 # # can also make it
288 # # run once by
289 # # require 'spec/spec'
290 # # run as
291 # # in _spec.rb will both be
292 # # run twice. It is
293 # # end with _spec.rb. You
294 # # can also make it
295 # # run once by
296 # # require 'spec/spec'
297 # # run as
298 # # in _spec.rb will both be
299 # # run twice. It is
300 # # end with _spec.rb. You
301 # # can also make it
302 # # run once by
303 # # require 'spec/spec'
304 # # run as
305 # # in _spec.rb will both be
306 # # run twice. It is
307 # # end with _spec.rb. You
308 # # can also make it
309 # # run once by
310 # # require 'spec/spec'
311 # # run as
312 # # in _spec.rb will both be
313 # # run twice. It is
314 # # end with _spec.rb. You
315 # # can also make it
316 # # run once by
317 # # require 'spec/spec'
318 # # run as
319 # # in _spec.rb will both be
320 # # run twice. It is
321 # # end with _spec.rb. You
322 # # can also make it
323 # # run once by
324 # # require 'spec/spec'
325 # # run as
326 # # in _spec.rb will both be
327 # # run twice. It is
328 # # end with _spec.rb. You
329 # # can also make it
330 # # run once by
331 # # require 'spec/spec'
332 # # run as
333 # # in _spec.rb will both be
334 # # run twice. It is
335 # # end with _spec.rb. You
336 # # can also make it
337 # # run once by
338 # # require 'spec/spec'
339 # # run as
340 # # in _spec.rb will both be
341 # # run twice. It is
342 # # end with _spec.rb. You
343 # # can also make it
344 # # run once by
345 # # require 'spec/spec'
346 # # run as
347 # # in _spec.rb will both be
348 # # run twice. It is
349 # # end with _spec.rb. You
350 # # can also make it
351 # # run once by
352 # # require 'spec/spec'
353 # # run as
354 # # in _spec.rb will both be
355 # # run twice. It is
356 # # end with _spec.rb. You
357 # # can also make it
358 # # run once by
359 # # require 'spec/spec'
360 # # run as
361 # # in _spec.rb will both be
362 # # run twice. It is
363 # # end with _spec.rb. You
364 # # can also make it
365 # # run once by
366 # # require 'spec/spec'
367 # # run as
368 # # in _spec.rb will both be
369 # # run twice. It is
370 # # end with _spec.rb. You
371 # # can also make it
372 # # run once by
373 # # require 'spec/spec'
374 # # run as
375 # # in _spec.rb will both be
376 # # run twice. It is
377 # # end with _spec.rb. You
378 # # can also make it
379 # # run once by
380 # # require 'spec/spec'
381 # # run as
382 # # in _spec.rb will both be
383 # # run twice. It is
384 # # end with _spec.rb. You
385 # # can also make it
386 # # run once by
387 # # require 'spec/spec'
388 # # run as
389 # # in _spec.rb will both be
390 # # run twice. It is
391 # # end with _spec.rb. You
392 # # can also make it
393 # # run once by
394 # # require 'spec/spec'
395 # # run as
396 # # in _spec.rb will both be
397 # # run twice. It is
398 # # end with _spec.rb. You
399 # # can also make it
400 # # run once by
401 # # require 'spec/spec'
402 # # run as
403 # # in _spec.rb will both be
404 # # run twice. It is
405 # # end with _spec.rb. You
406 # # can also make it
407 # # run once by
408 # # require 'spec/spec'
409 # # run as
410 # # in _spec.rb will both be
411 # # run twice. It is
412 # # end with _spec.rb. You
413 # # can also make it
414 # # run once by
415 # # require 'spec/spec'
416 # # run as
417 # # in _spec.rb will both be
418 # # run twice. It is
419 # # end with _spec.rb. You
420 # # can also make it
421 # # run once by
422 # # require 'spec/spec'
423 # # run as
424 # # in _spec.rb will both be
425 # # run twice. It is
426 # # end with _spec.rb. You
427 # # can also make it
428 # # run once by
429 # # require 'spec/spec'
430 # # run as
431 # # in _spec.rb will both be
432 # # run twice. It is
433 # # end with _spec.rb. You
434 # # can also make it
435 # # run once by
436 # # require 'spec/spec'
437 # # run as
438 # # in _spec.rb will both be
439 # # run twice. It is
440 # # end with _spec.rb. You
441 # # can also make it
442 # # run once by
443 # # require 'spec/spec'
444 # # run as
445 # # in _spec.rb will both be
446 # # run twice. It is
447 # # end with _spec.rb. You
448 # # can also make it
449 # # run once by
450 # # require 'spec/spec'
451 # # run as
452 # # in _spec.rb will both be
453 # # run twice. It is
454 # # end with _spec.rb. You
455 # # can also make it
456 # # run once by
457 # # require 'spec/spec'
458 # # run as
459 # # in _spec.rb will both be
460 # # run twice. It is
461 # # end with _spec.rb. You
462 # # can also make it
463 # # run once by
464 # # require 'spec/spec'
465 # # run as
466 # # in _spec.rb will both be
467 # # run twice. It is
468 # # end with _spec.rb. You
469 # # can also make it
470 # # run once by
471 # # require 'spec/spec'
472 # # run as
473 # # in _spec.rb will both be
474 # # run twice. It is
475 # # end with _spec.rb. You
476 # # can also make it
477 # # run once by
478 # # require 'spec/spec'
479 # # run as
480 # # in _spec.rb will both be
481 # # run twice. It is
482 # # end with _spec.rb. You
483 # # can also make it
484 # # run once by
485 # # require 'spec/spec'
486 # # run as
487 # # in _spec.rb will both be
488 # # run twice. It is
489 # # end with _spec.rb. You
490 # # can also make it
491 # # run once by
492 # # require 'spec/spec'
493 # # run as
494 # # in _spec.rb will both be
495 # # run twice. It is
496 # # end with _spec.rb. You
497 # # can also make it
498 # # run once by
499 # # require 'spec/spec'
500 # # run as
501 # # in _spec.rb will both be
502 # # run twice. It is
503 # # end with _spec.rb. You
504 # # can also make it
505 # # run once by
506 # # require 'spec/spec'
507 # # run as
508 # # in _spec.rb will both be
509 # # run twice. It is
510 # # end with _spec.rb. You
511 # # can also make it
512 # # run once by
513 # # require 'spec/spec'
514 # # run as
515 # # in _spec.rb will both be
516 # # run twice. It is
517 # # end with _spec.rb. You
518 # # can also make it
519 # # run once by
520 # # require 'spec/spec'
521 # # run as
522 # # in _spec.rb will both be
523 # # run twice. It is
524 # # end with _spec.rb. You
525 # # can also make it
526 # # run once by
527 # # require 'spec/spec'
528 # # run as
529 # # in _spec.rb will both be
530 # # run twice. It is
531 # # end with _spec.rb. You
532 # # can also make it
533 # # run once by
534 # # require 'spec/spec'
535 # # run as
536 # # in _spec.rb will both be
537 # # run twice. It is
538 # # end with _spec.rb. You
539 # # can also make it
540 # # run once by
541 # # require 'spec/spec'
542 # # run as
543 # # in _spec.rb will both be
544 # # run twice. It is
545 # # end with _spec.rb. You
546 # # can also make it
547 # # run once by
548 # # require 'spec/spec'
549 # # run as
550 # # in _spec.rb will both be
551 # # run twice. It is
552 # # end with _spec.rb. You
553 # # can also make it
554 # # run once by
555 # # require 'spec/spec'
556 # # run as
557 # # in _spec.rb will both be
558 # # run twice. It is
559 # # end with _spec.rb. You
560 # # can also make it
561 # # run once by
562 # # require 'spec/spec'
563 # # run as
564 # # in _spec.rb will both be
565 # # run twice. It is
566 # # end with _spec.rb. You
567 # # can also make it
568 # # run once by
569 # # require 'spec/spec'
570 # # run as
571 # # in _spec.rb will both be
572 # # run twice. It is
573 # # end with _spec.rb. You
574 # # can also make it
575 # # run once by
576 # # require 'spec/spec'
577 # # run as
578 # # in _spec.rb will both be
579 # # run twice. It is
580 # # end with _spec.rb. You
581 # # can also make it
582 # # run once by
583 # # require 'spec/spec'
584 # # run as
585 # # in _spec.rb will both be
586 # # run twice. It is
587 # # end with _spec.rb. You
588 # # can also make it
589 # # run once by
590 # # require 'spec/spec'
591 # # run as
592 # # in _spec.rb will both be
593 # # run twice. It is
594 # # end with _spec.rb. You
595 # # can also make it
596 # # run once by
597 # # require 'spec/spec'
598 # # run as
599 # # in _spec.rb will both be
600 # # run twice. It is
601 # # end with _spec.rb. You
620 # # can also make it
621 # # run once by
622 # # require 'spec/spec'
623 # # run as
624 # # in _spec.rb will both be
625 # # run twice. It is
626 # # end with _spec.rb. You
627 # # can also make it
628 # # run once by
629 # # require 'spec/spec'
630 # # run as
631 # # in _spec.rb will both be
632 # # run twice. It is
633 # # end with _spec.rb. You
634 # # can also make it
635 # # run once by
636 # # require 'spec/spec'
637 # # run as
638 # # in _spec.rb will both be
639 # # run twice. It is
640 # # end with _spec.rb. You
641 # # can also make it
642 # # run once by
643 # # require 'spec/spec'
644 # # run as
645 # # in _spec.rb will both be
646 # # run twice. It is
647 # # end with _spec.rb. You
648 # # can also make it
649 # # run once by
650 # # require 'spec/spec'
651 # # run as
652 # # in _spec.rb will both be
653 # # run twice. It is
654 # # end with _spec.rb. You
655 # # can also make it
656 # # run once by
657 # # require 'spec/spec'
658 # # run as
659 # # in _spec.rb will both be
660 # # run twice. It is
661 # # end with _spec.rb. You
662 # # can also make it
663 # # run once by
664 # # require 'spec/spec'
665 # # run as
666 # # in _spec.rb will both be
667 # # run twice. It is
668 # # end with _spec.rb. You
669 # # can also make it
670 # # run once by
671 # # require 'spec/spec'
672 # # run as
673 # # in _spec.rb will both be
674 # # run twice. It is
675 # # end with _spec.rb. You
676 # # can also make it
677 # # run once by
678 # # require 'spec/spec'
679 # # run as
680 # # in _spec.rb will both be
681 # # run twice. It is
682 # # end with _spec.rb. You
683 # # can also make it
684 # # run once by
685 # # require 'spec/spec'
686 # # run as
687 # # in _spec.rb will both be
688 # # run twice. It is
689 # # end with _spec.rb. You
690 # # can also make it
691 # # run once by
692 # # require 'spec/spec'
693 # # run as
694 # # in _spec.rb will both be
695 # # run twice. It is
696 # # end with _spec.rb. You
697 # # can also make it
698 # # run once by
699 # # require 'spec/spec'
700 # # run as
701 # # in _spec.rb will both be
702 # # run twice. It is
703 # # end with _spec.rb. You
704 # # can also make it
705 # # run once by
706 # # require 'spec/spec'
707 # # run as
708 # # in _spec.rb will both be
709 # # run twice. It is
710 # # end with _spec.rb. You
711 # # can also make it
720 # # can also make it
721 # # run once by
722 # # require 'spec/spec'
723 # # run as
724 # # in _spec.rb will both be
725 # # run twice. It is
726 # # end with _spec.rb. You
727 # # can also make it
728 # # run once by
729 # # require 'spec/spec'
730 # # run as
731 # # in _spec.rb will both be
732 # # run twice. It is
733 # # end with _spec.rb. You
734 # # can also make it
735 # # run once by
736 # # require 'spec/spec'
737 # # run as
738 # # in _spec.rb will both
```

# What makes Python so great for Computer Vision?

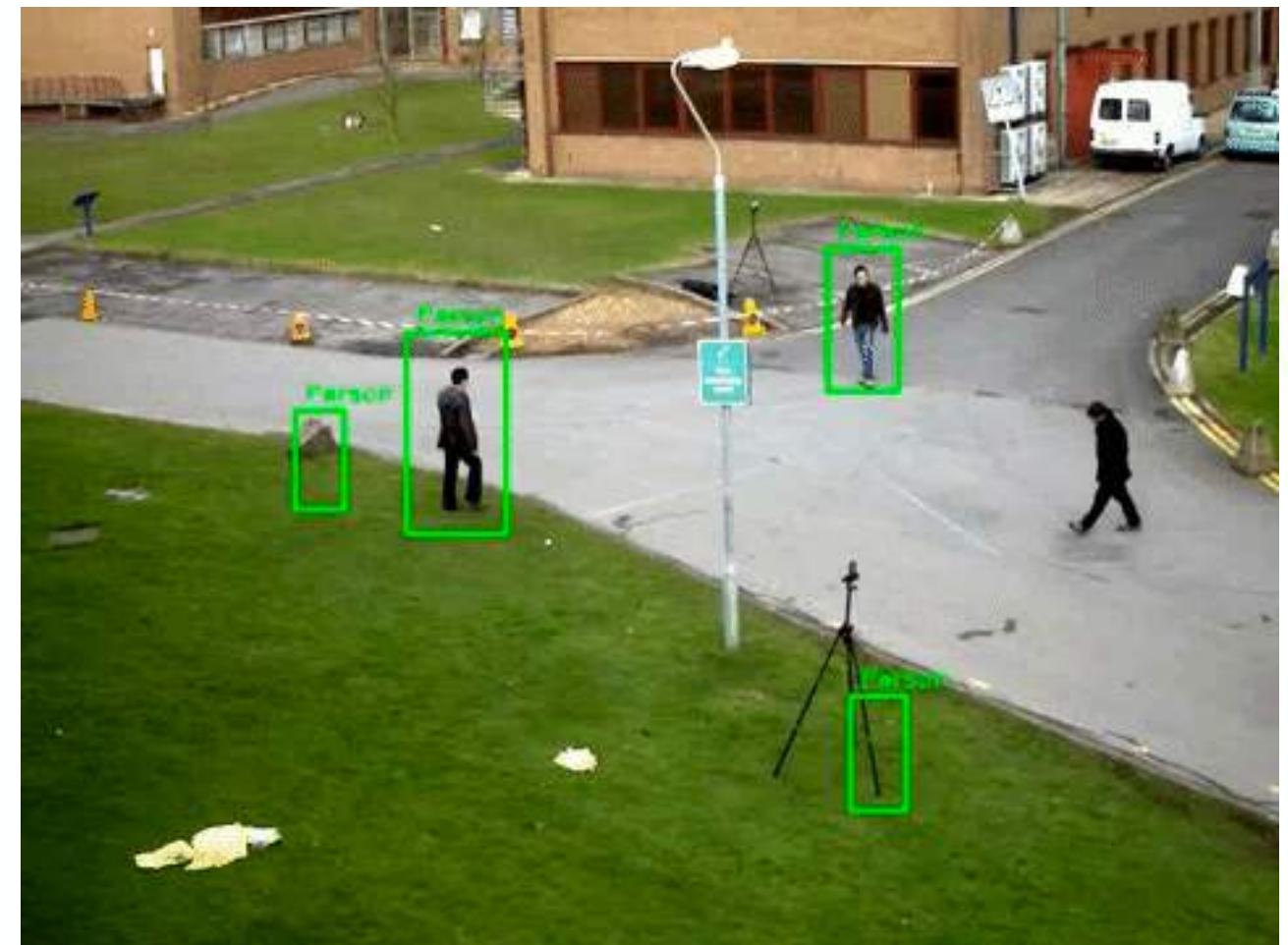
- Easy to learn and accessible
- Language of Artificial Intelligence
- The best and easiest to use Libraries, such as:
  - **OpenCV** for Classical Computer Vision
  - **PyTorch** (Facebook) and **TensorFlow with Keras** (Google)



TensorFlow

# Classical Computer Vision?

- What is meant by **Classical Computer Vision?**
- It encompasses Computer Vision algorithms that **do not** involve Machine Learning
- Before the advent of Machine Learning and Deep Learning, Computer Vision was a deeply explored field and many useful algorithms were developed for things like **feature extraction, OCR, Segmentation and simple transformations.**
- **OpenCV** is the Classical Computer Vision library of choice!



# Deep Learning Computer Vision

- Deep Learning was used in Computer Vision since the 1990s, however due to the computational requirements and intricate design, it remained on the sidelines for decades.
- Until the mid 2010s...which brought **two important building blocks** together.
  - **Mature Deep Learning libraries** (TensorFlow, Keras, Theano, Caffe)
  - **Accessible GPU processing** (NVIDIA's CUDA)



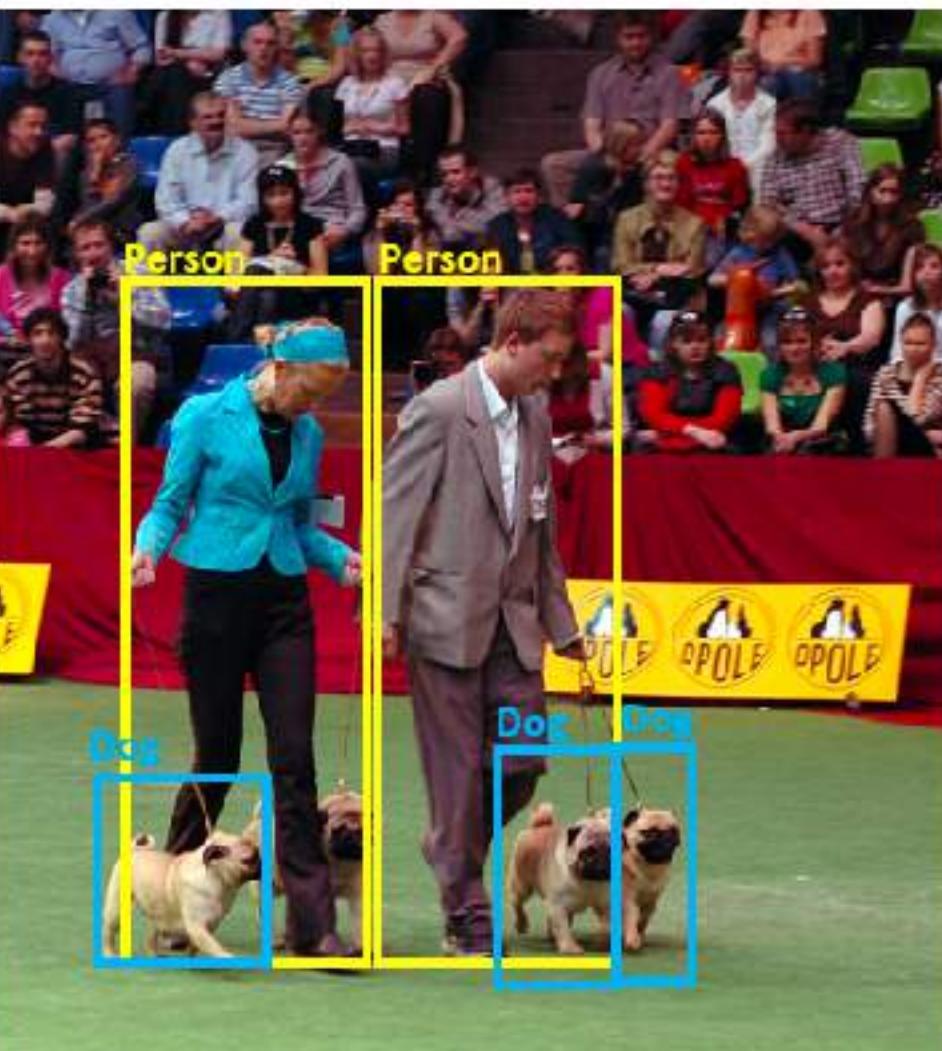
# Deep Learning Examples

Image Classification



{Dog}

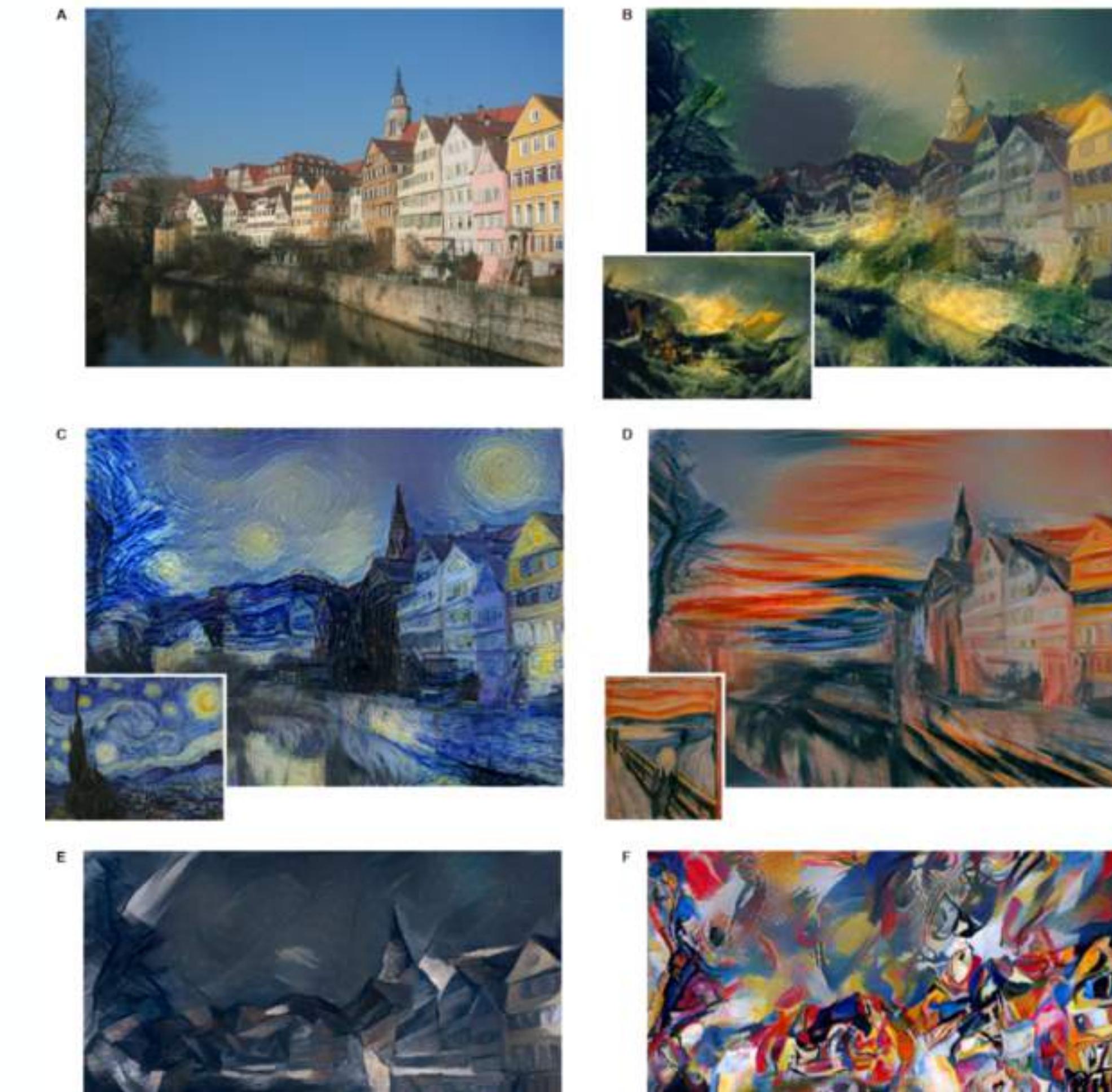
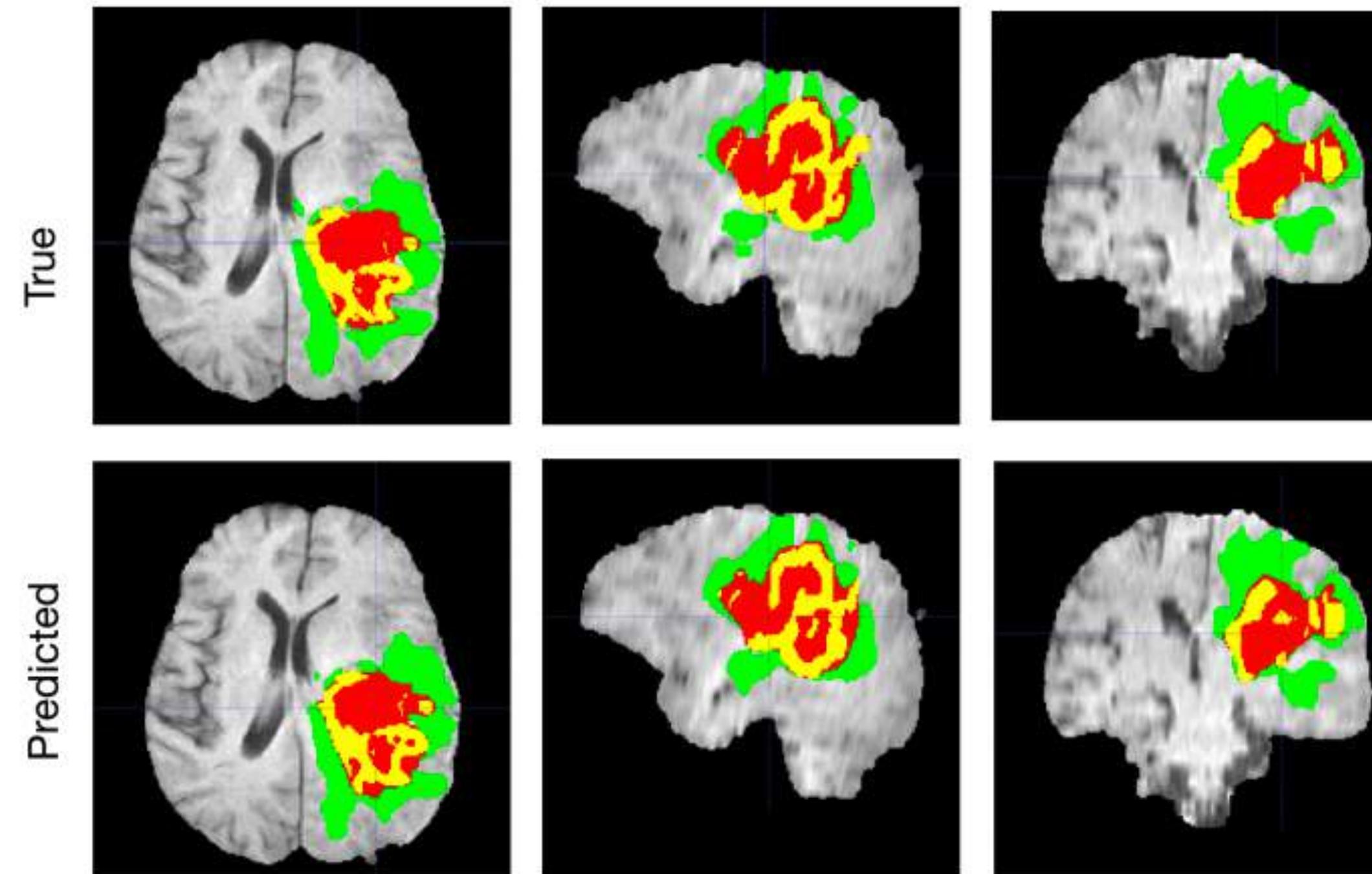
Object Detection



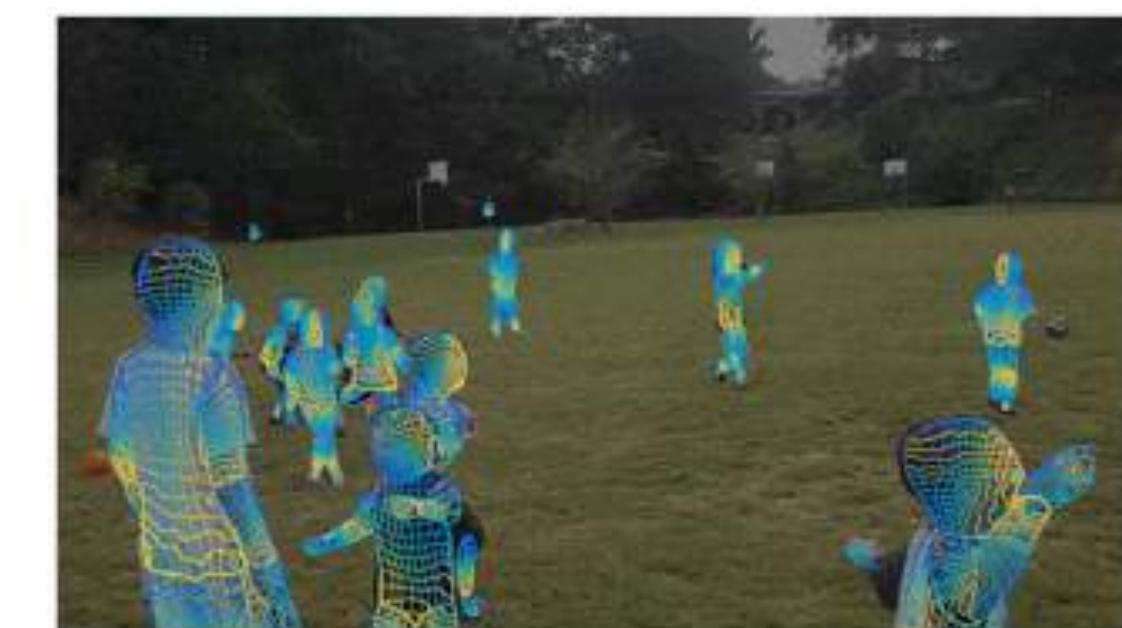
{Dog, Dog, Dog, Person, Person}



# Deep Learning Examples



# Deep Learning Examples



# Deep Learning CV vs Classical CV

<b>Deep Learning</b>	<b>Classical Computer Vision</b>
Adapts to new images well (assuming it's similar to the data it was trained on)	Small changes can have big negative impacts
Requires Models to be trained	Doesn't require training and can be used once coded
Model weights learn to adapt to varying image conditions	Relies on hardcoded features and parameters
Requires GPU hardware (most times)	Can be run on CPU

# Our OpenCV Outline

1. Getting Started with OpenCV
2. Grey-scaling Images
3. Color Spaces (HSV & RGB)
4. Drawing on Images
5. Transformations - Translations and Rotations
6. Scaling and re-sizing and Cropping
7. Arithmetic and Bitwise Operations
8. Convolutions, Blurring and Sharpening
9. Thresholding & Binarization
10. Dilation, Erosion and Edge Detection
11. Contours - Drawing and Hierarchy and Modes
12. Moments, Matching and Sorting Contours
13. Line, Circle, Blob Detection
14. Counting Circles, Ellipses and Finding Waldo
15. Finding Corners
16. Face and Eye Detection with HAAR Cascade Classifiers
17. Vehicle & Pedestrian Detection
18. Perspective Transforms
19. Histograms and K-means clustering for finding dominant colours
20. Comparing Images with MSE and Structural Similarity

# Our OpenCV Outline

21. Filtering Colors
22. Watershed Algorithm marker-based image segmentation
23. Background and Foreground Subtraction
24. Motion tracking using Mean Shift and CAM-Shift
25. Optical Flow Object Tracking
26. Simple Object Tracking by Colour
27. Facial Landmarks Detection with Dlib
28. Face Swapping with Dlib
29. Tilt Shift Effect
30. Grabcut Algorithm for Background Removal
31. OCR with PyTesseract and EasyOCR
32. Barcode and QR generation and reading
33. YOLOv3 in OpenCV
34. Neural Style Transfer with OpenCV
35. SSDs in OpenCV
36. Colorise Black and White Photos
37. Repair Damaged Photos with Inpainting
38. Add and remove Noise, Fix Contrast with Histogram Equalisation
39. Detect Blur in Images
40. Facial Recognition

# Our OpenCV Outline

## Video Section

1. Using Webcam to do Sketch
2. Opening Video Files
3. Saving and Recording Videos
4. Video Streams RTSP and IP
5. Auto Reconnect to Stream
6. Capturing video using screenshots
7. Import Youtube Video into OpenCV

# Deep Learning Outline

## Introduction to Deep Learning in Computer Vision

- Convolution Neural Networks
- Feature Detectors
- Convolution on Color Images
- Kernel Size and Depth
- Padding
- Stride
- Activation Layer ReLU
- Pooling
- Fully Connected Layer
- Softmax Layer
- Building a CNN
- Parameter Counts in CNNs
- Why CNNs work so well for Images
- How to Train a CNN
- Loss Functions
- Back Propagation
- Gradient Descent
- Optimisers
- Summary of CNNs

# Deep Learning Outline

## Deep Learning Basics with PyTorch and TensorFlow Keras

- Deep Learning History
- Deep Learning Libraries
- Building and Training Your First CNN with PyTorch
- PyTorch Transformations
- PyTorch Inspect and Visualise Dataset
- PyTorch Dataloader
- PyTorch Building our CNN
- PyTorch Loss Function and Optimiser
- PyTorch Training
- PyTorch Results
- PyTorch Plot Results
- Building and Training Your First CNN with Keras in TensorFlow 2 - Load Data
- Keras - Inspecting and Visualising
- Keras - Preprocessing
- Keras Constructing our CNN
- Keras - Training our Model
- Keras - Plotting our Results
- Keras - Saving and Loading, Visualising Results

## Comparing Libraries and Analysing Performance

- Deep Learning Library Comparisons
- Assessing Model Performance
- Confusion Matrix and Classification Reports
- Keras - Loading Data and Viewign Misclassifications
- Keras - Confusion Matrix and Classification Report
- PyTorch - Loading Data and Viewign Misclassifications

# Deep Learning Outline

## Regularisation and Overfitting

- Overfitting and Generalisation
- Regularisation Methods
- L1 and L2 Regularisation
- Dropout
- Data Augmentation
- Early Stopping
- Batch Normalisation
- When to use regularisation
- Keras - FNIST No Regularisation
- Keras - FNIST with Regularisation
- PyTorch - FNIST No Regularisation
- PyTorch - FNIST with Regularisation

## Understanding What CNN's See

- Visualising Filters of CNNs
- Filter Activations of CNNS
- Keras - Visualise Filter and Feature Maps
- Maximising Filters
- Maximising Class Activations
- Keras - Filter and Class Maximisation
- GradCAM
- Keras - GradCAM

# Deep Learning Outline

## Designing CNNs and Overview of Modern CNN Architectures

- Basic CNN Design Principles
- CNN History
- LeNet
- AlexNet
- VGGNet
- ResNet
- Why do ResNets Work
- MobileNet
- Inception
- SqueezeNet
- EfficientNet
- DenseNet
- ImageNet Dataset
- Keras LeNET and AlexNet
- PyTorch Pretrained Networks
- Keras Pretrained Networks
- Top-1 and Top-5 Accuracies
- PyTorch Rank N
- Keras Rank N
- Callbacks with PyTorch
- Callbacks with Keras

# Deep Learning Outline

## PyTorch Lightning

- PyTorch Lightning Setup and Class Design
- AutoBatch Selection & Auto LR Selection, Tensor-boards
- Callbacks, Saving and Inference
- mGPU, TPU more
- Transfer Learning Lightning

## Transfer Learning

- Keras Transfer Learning and Fine Tuning
- Keras Using CNN as a Feature Extactor
- PyTorch Transfer Learning and Fine Tuning
- PyTorch Feature Extraction Transfer Learning
- PyTorch Feature Extraction

## Google DeepDream

- Google DeepDream with PyTorch
- Google DeepDream with Keras

## Neural Style Transfer

- Neural Style Transfer with PyTorch
- Neural Style Transfer with Keras

## Autoencoders

- Autoencoders with Keras
- Autoencoders with PyTorch

# Deep Learning Outline

## Generative Adversarial Networks

- How do GANs Work?
- Training GANs
- Where do we use GANs?
- Keras DCGAN MNIST
- PyTorch DCGAN MNIST
- Super Resolution Keras
- Anime Characters - StyleGAN
- Horses to Zebras - CycleGAN
- ArcaneGAN

## Siamese Networks

- Training Siamese Networks
- Keras Siamese
- PyTorch Siamese

## Facial Recognition

- Facial Similarity Keras VGGFace
- Face Recognition Keras One Shot learning and Friends
- Facial Similarity PyTorch - FaceNet
- DeepFace - Age, Gender, Pose

# Deep Learning Outline

## Object Detection

- Early Object Detectors
- Intersection Over Union - Assessing Object Detector Performance
- Mean Average Precision (mAP)
- Non-Maximum Suppression
- Object Detection - R-CNNs - COCO
- Object Detection - SSDs
- Object Detection - Introduction to the YOLO Object Detectors
- Object Detectors - How Does YOLO Work?
- Object Detectors - Training YOLO
- Object Detectors - YOLO Architecture and Evolution from YOLOv3 to v5
- Object Detection - EfficientDet
- Object Detection - Detectron2
- Object Detection - Weapons Recognition - Guns SclaedYOLOv4
- Object Detection - Mask Detection - MobileNetV2 SSD
- Object Detection - Sign Language - TFODAPI - EfficientNetD0-D7
- Object Detection - Mushroom Detection - Detectron2
- Object Detection - Pothole Detection - TinyYOLOv4
- Object Detection - Website Region - YOLOv4 Darknet
- Object Detection - Drone Maritime Faster R-CNN
- Object Detection - Chess Pieces YOLOv3 PyTorch
- Object Detection - Hardhat Detection using EfficientDet
- Object Detection - Bloodcell Detector YOLOv3
- Object Detection - Plant Doctor YOLOv5



# Deep Learning Outline

## Deep Segmentation

- Deep Segmentation - U-Net & SegNet - Nuclie
- Deep Segmentation - SegNet Medical
- MaskCNN Demo
- Detectron2 MaskRCNN
- Detectron2 Bodypose
- MaskCNN Shapes

## Tracking with Deep Learning

- DeepSort - Track Cars

# Deep Learning Outline

## Miscellaneous Deep Learning Tutorials

- DeepFakes
- Vision Transformers
- ViT PyTorch
- ViT Keras
- BiT Keras
- Monocular depth estimation
- Image Similarity using Metric Learning
- Image Captioning
- Video Classification with a CNN-RNN Architecture
- Video Classification using Transformers
- Point Cloud Classification
- Point Cloud Segmentation
- 3D image classification from CT scans
- X-Ray Pneumonia Classification
- OCR model for reading Captchas
- WebApp - Computer Vision using Flask
- WebApp - Computer Vision using Flask Site

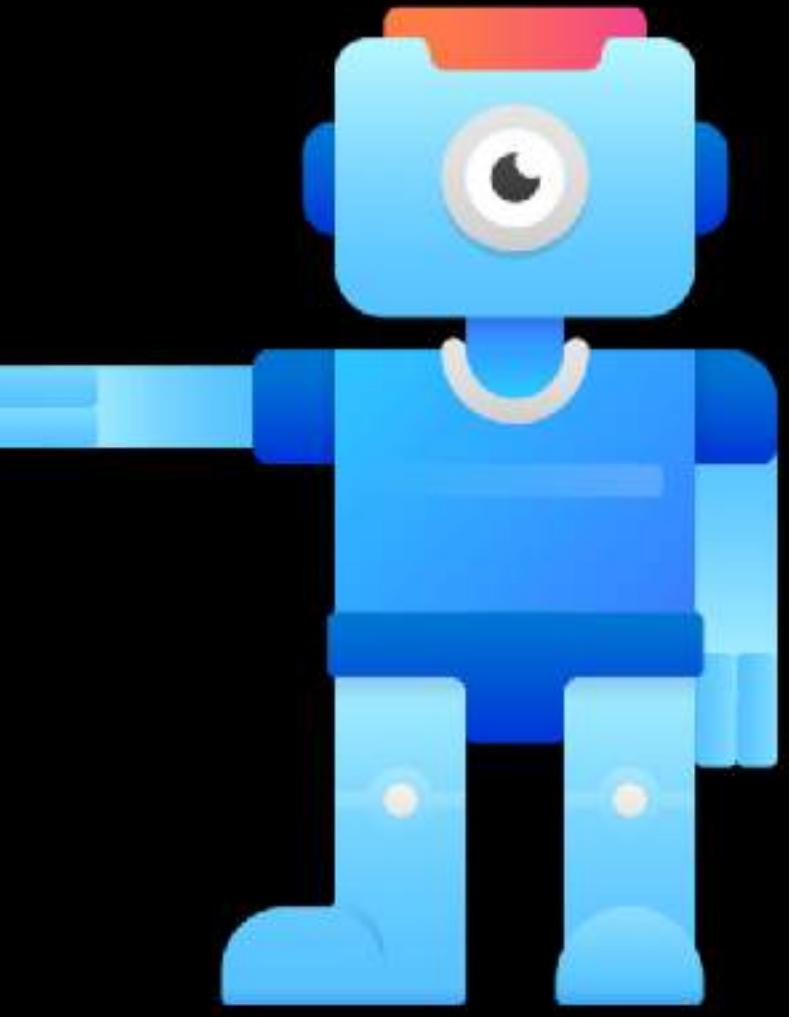


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

## Introduction to Computer Vision



# MODERN COMPUTER VISION

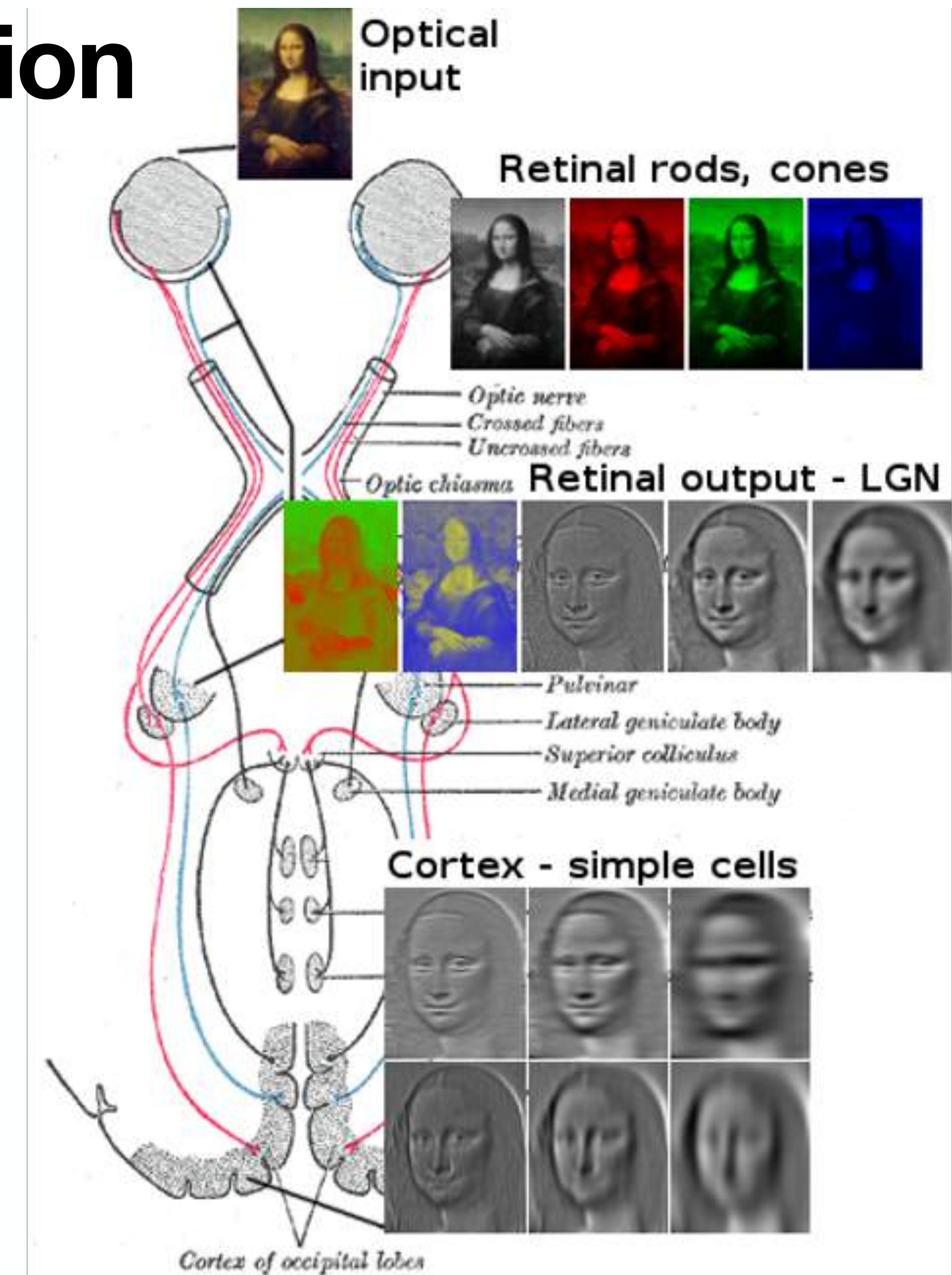
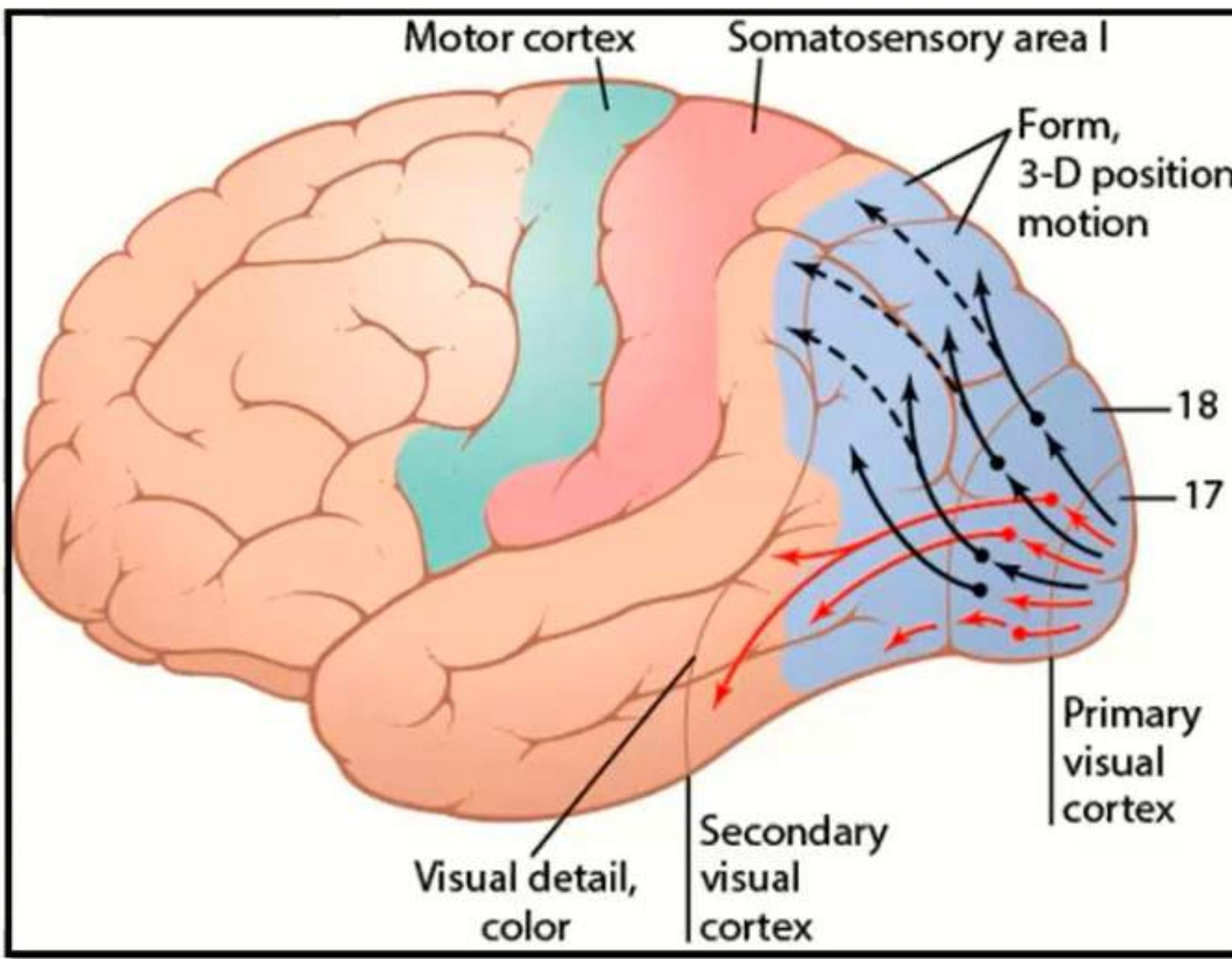
BY RAJEEV RATAN

# What Makes Computer Vision Hard?

**Nothing worth having comes easy**

# Our Brains are amazing at Vision

The **visual cortex** (located in the occipital lobe) is the primary cortical region of the brain that receives, integrates, and processes visual information relayed from the retinas.



# Can Artificial Intelligence Come Close?

The AI Dream, a machine that can see and understand better than humans.



Source - Terminator 2: Judgement Day (1991)

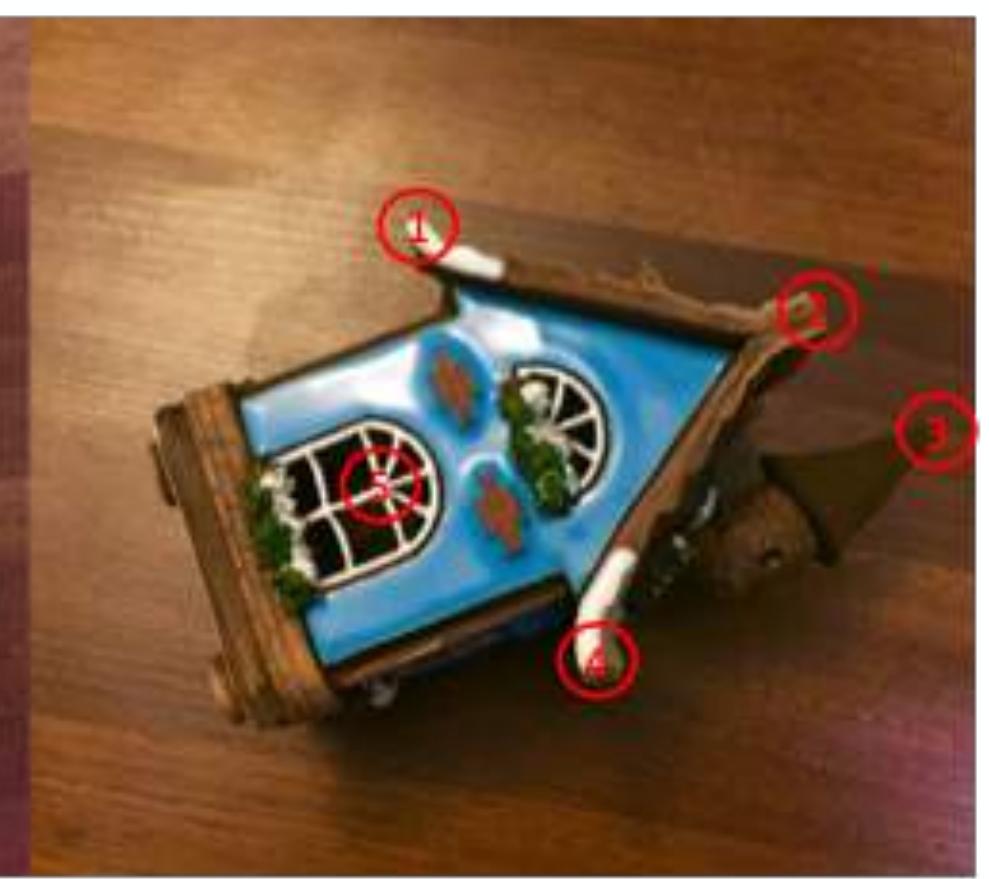
# What Makes Computer Vision Hard?

## Camera and Sensor Limitations



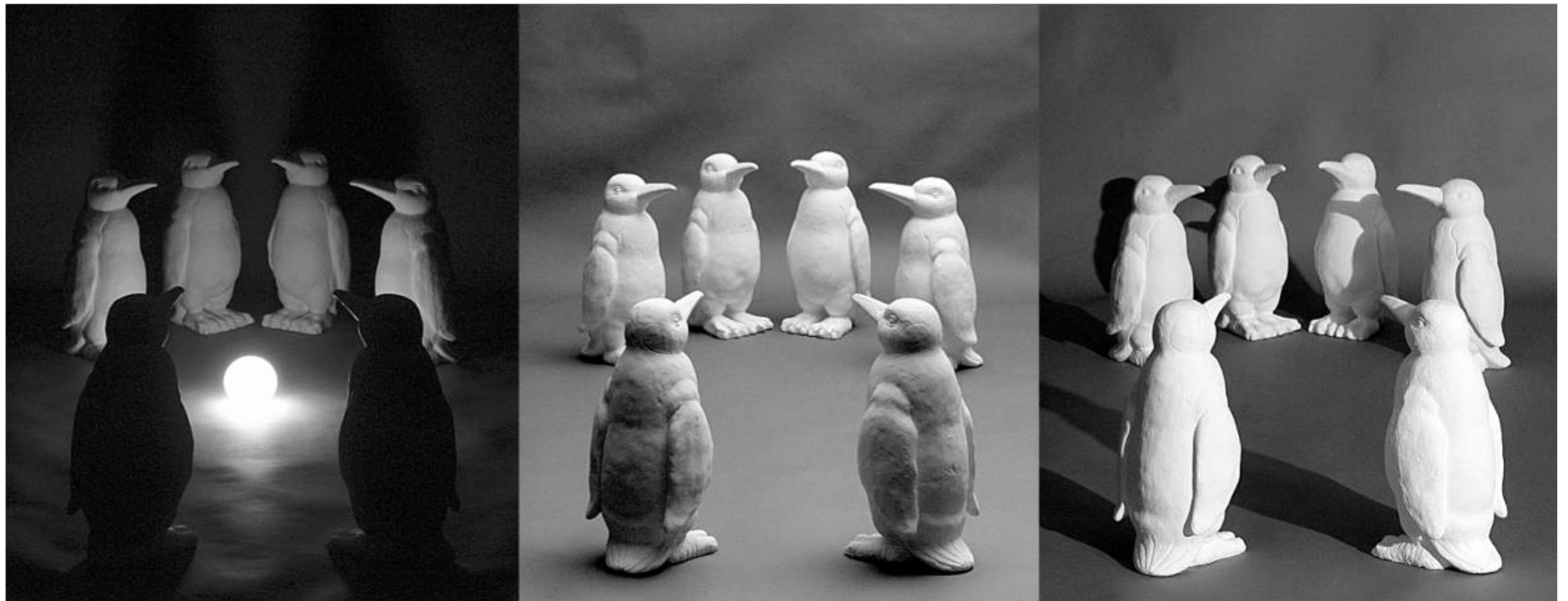
# What Makes Computer Vision Hard?

## Viewpoint Variations



# What Makes Computer Vision Hard?

## Changing Lighting Conditions



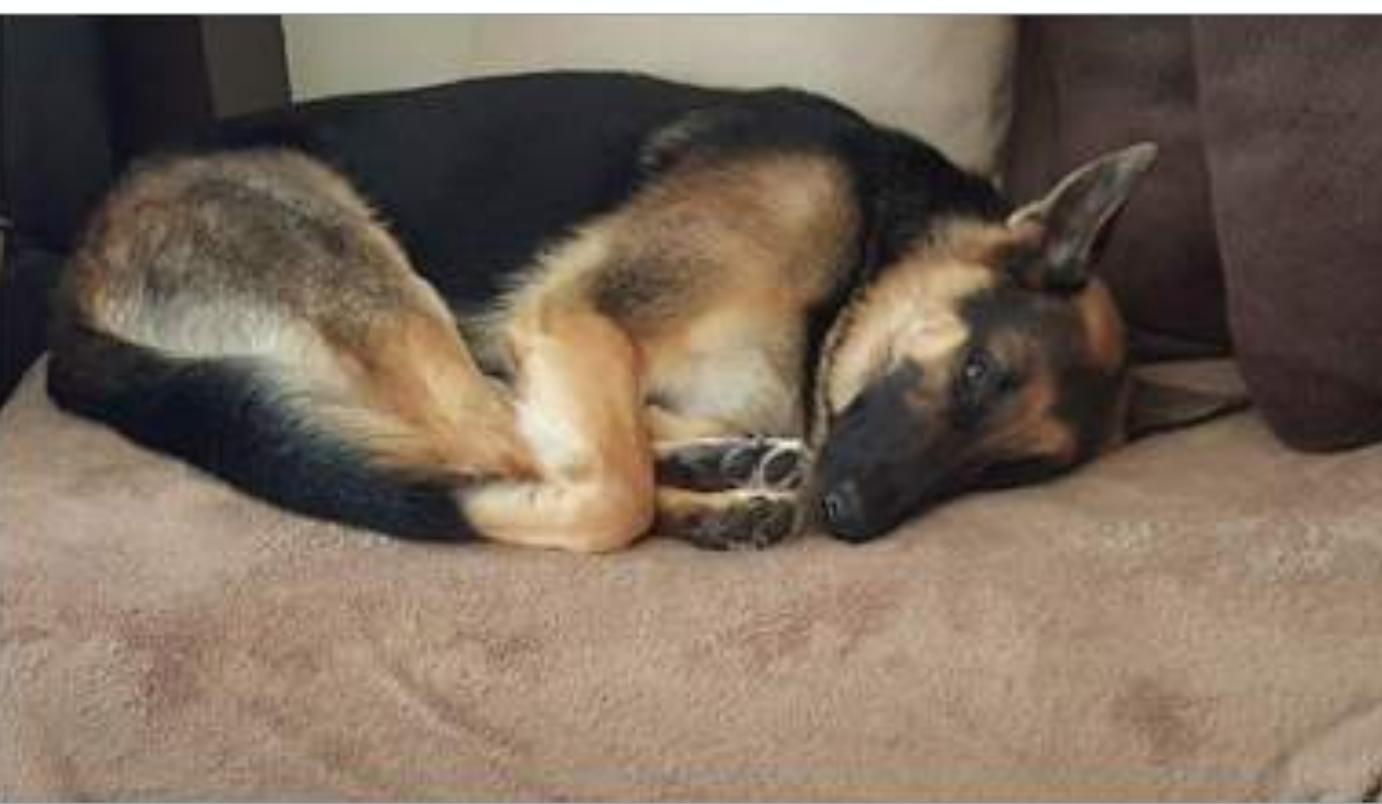
# What Makes Computer Vision Hard?

## Scaling Issues



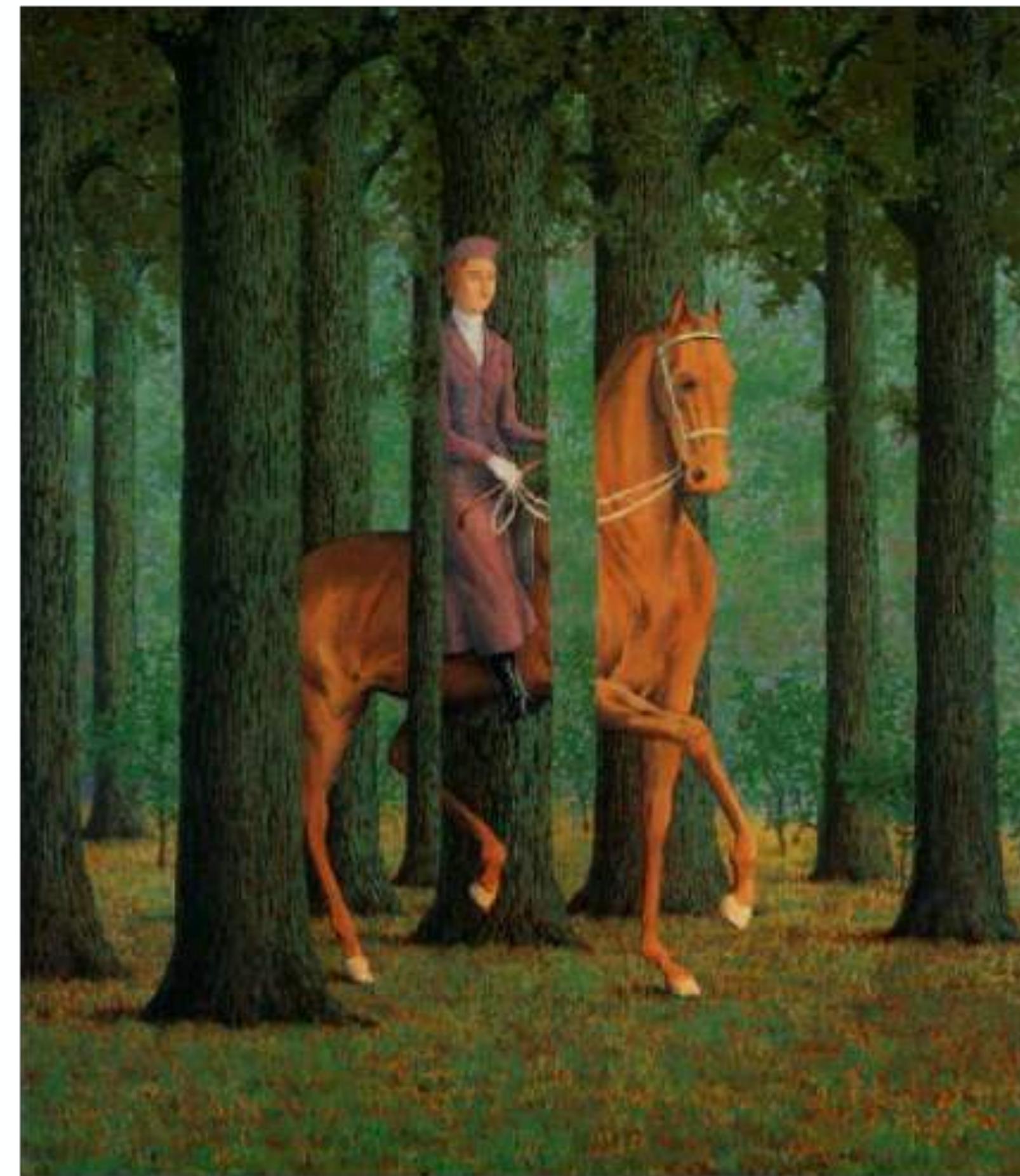
# What Makes Computer Vision Hard?

## Non-rigid Deformations



# What Makes Computer Vision Hard?

## Occlusion



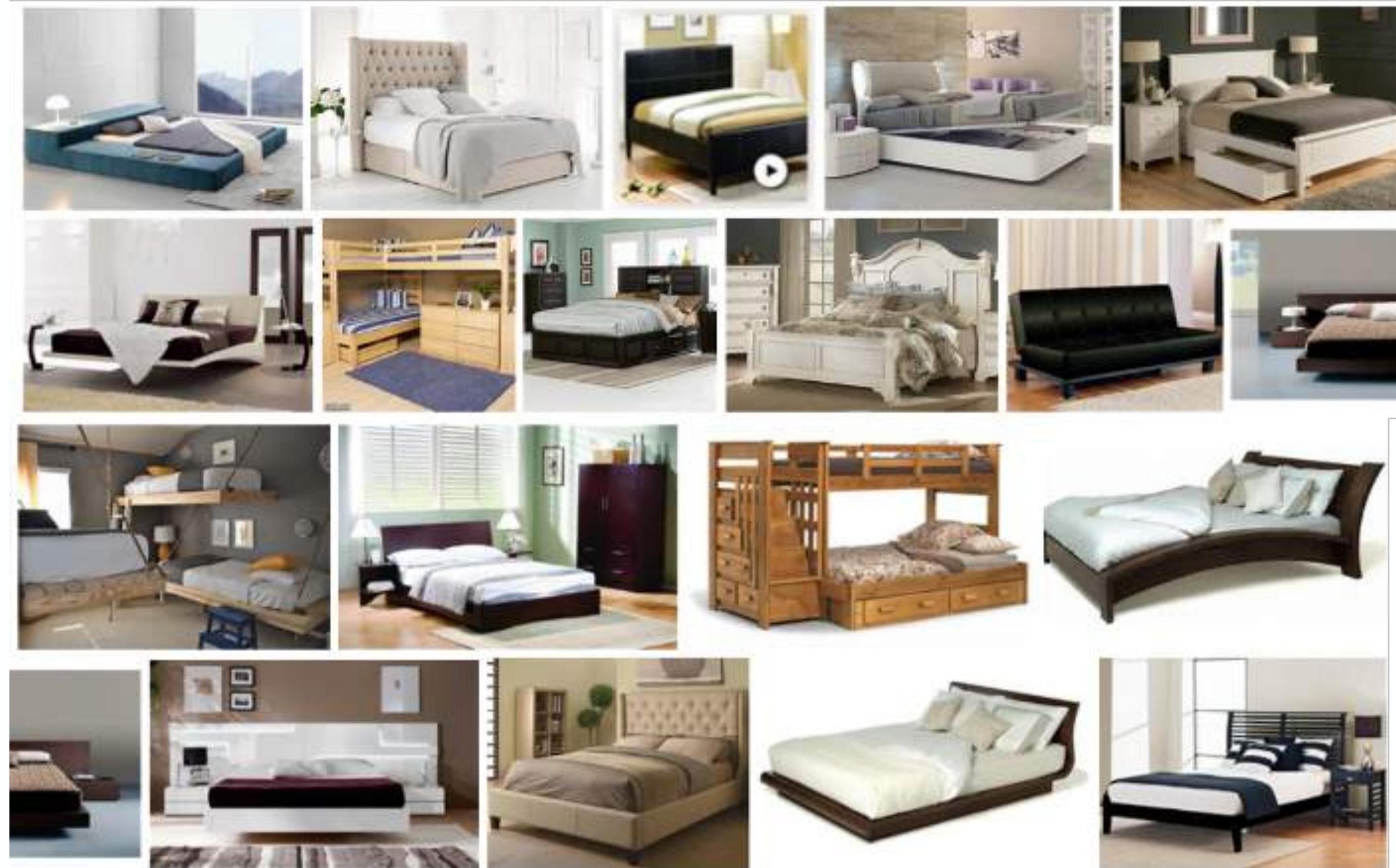
# What Makes Computer Vision Hard?

## Clutter



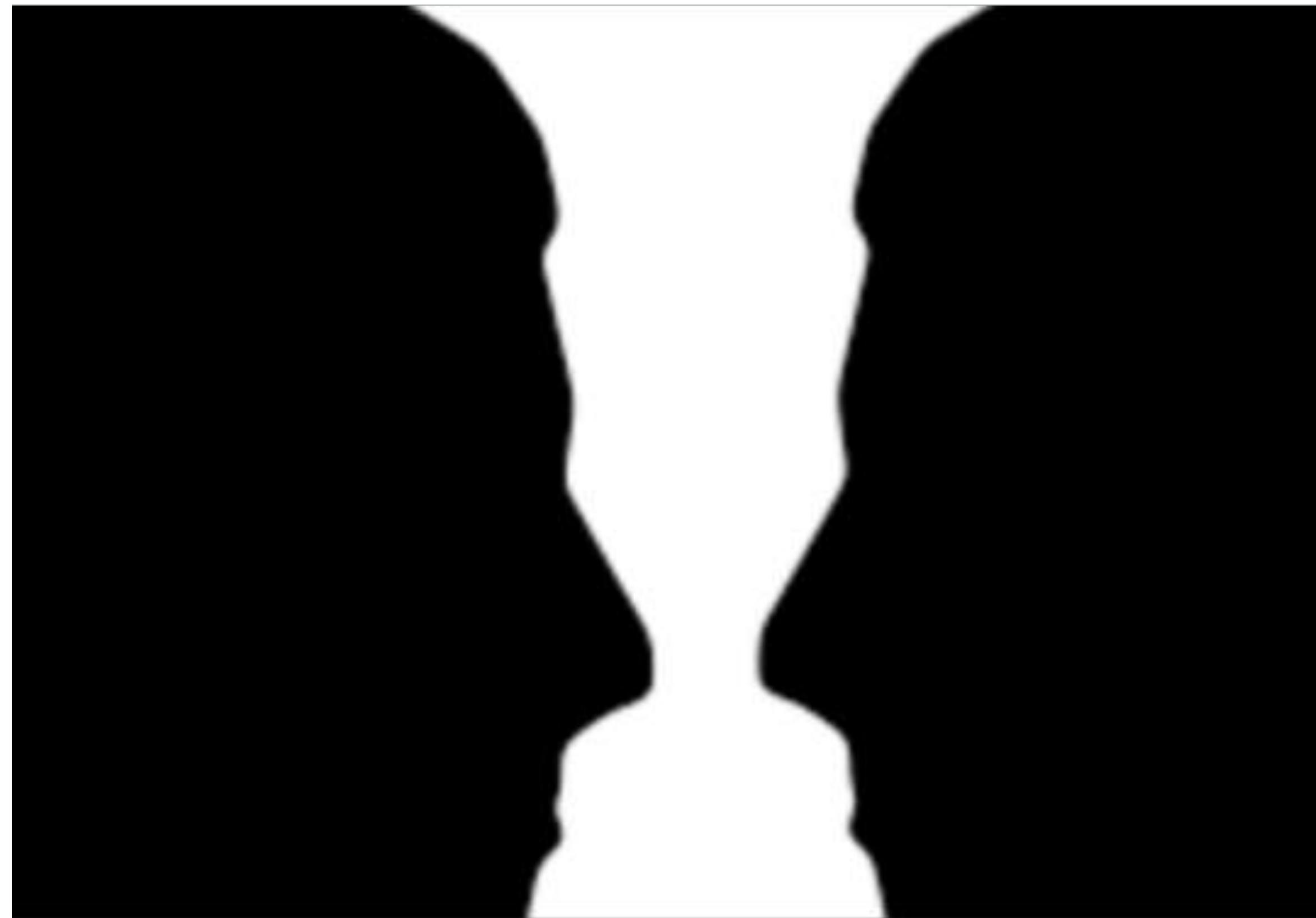
# What Makes Computer Vision Hard?

## Object Class Variation



# What Makes Computer Vision Hard?

## Ambiguous Optical Illusions



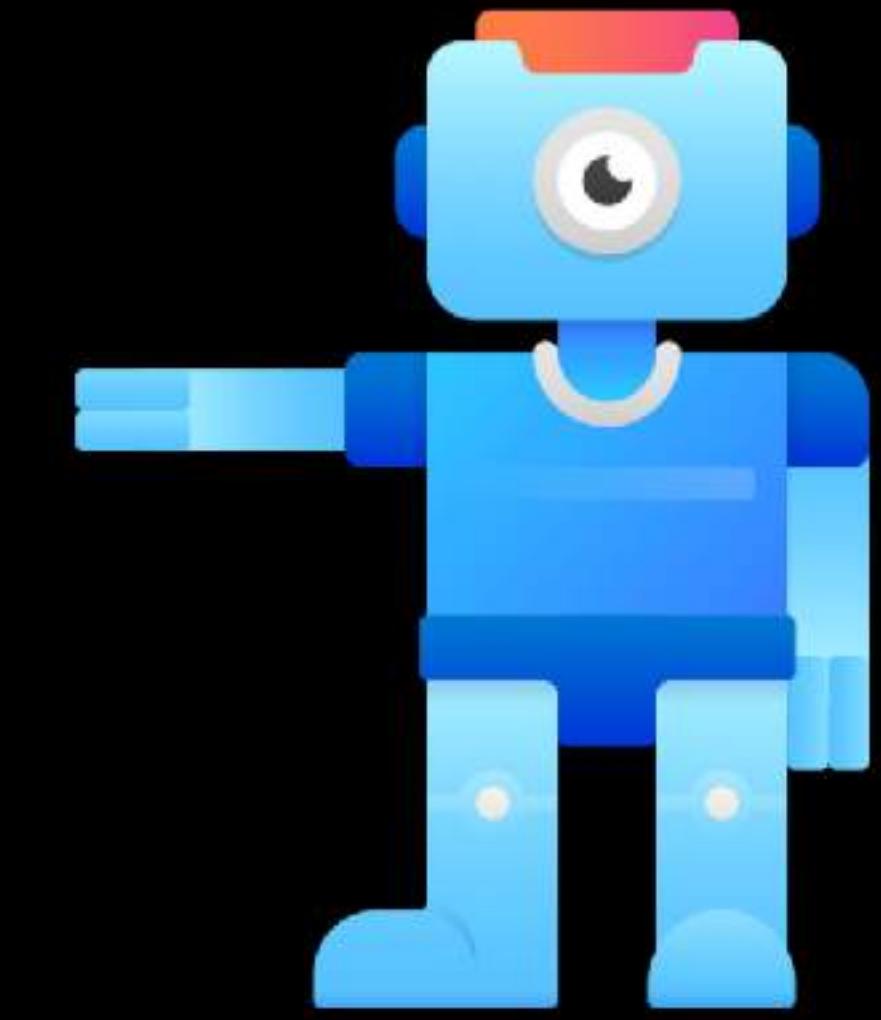


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**What are Images?**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

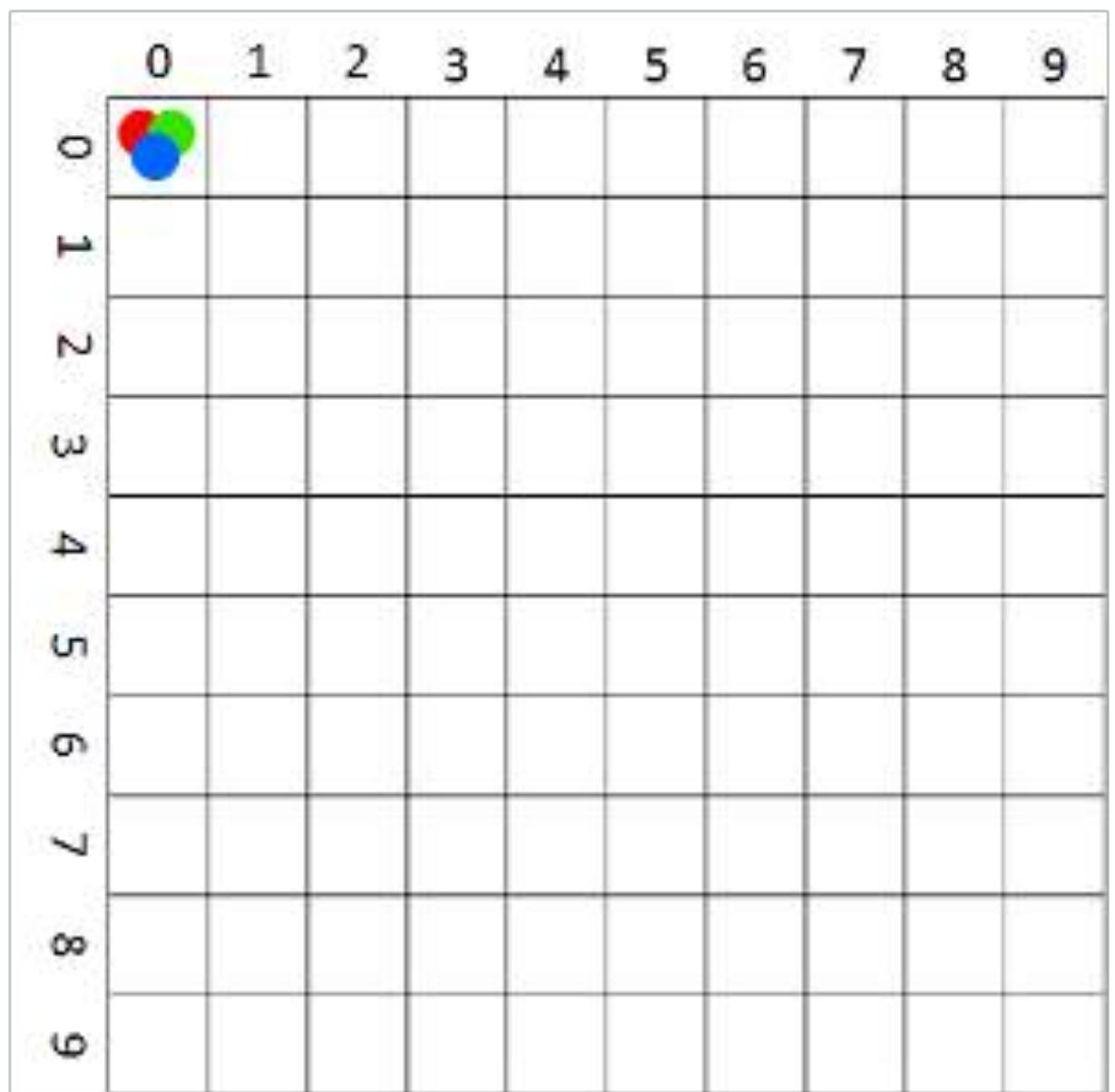
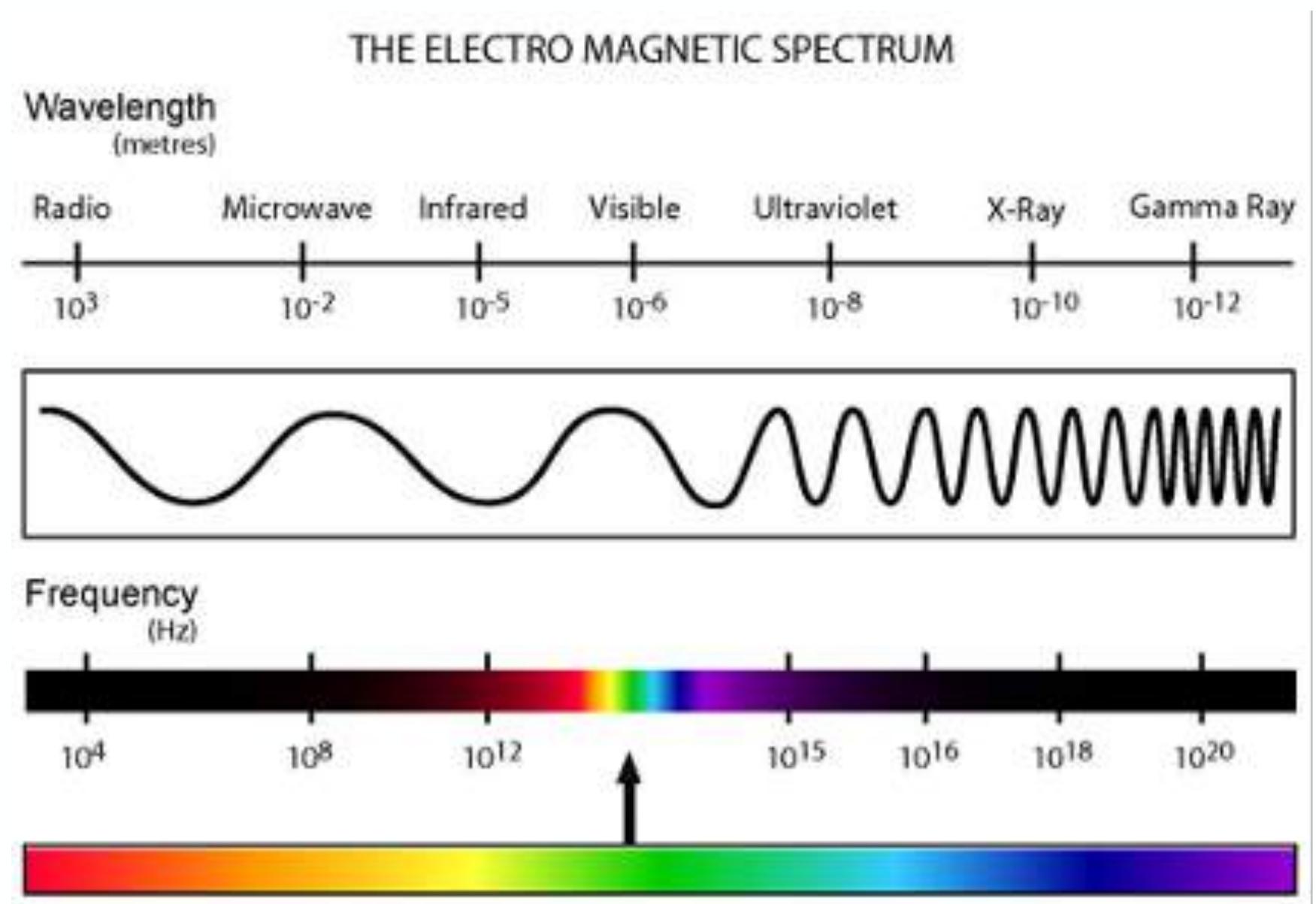
## What Are Images? A foundation in understanding digital images

# How do Computers ‘see’ Images?

4	17	32	85	22	42	75	157	285	147	52	221	37	251	289	111	26	6	38	132	248	224	224	138	52	130	205	17	82	155	107	27	90	256	94	26	219	31	36	262	26	73	45	50	241	82	109	210	342	27	192	166	243	55	26	133	95	72	240	220	96	234	37	88	235	31	138	89	218	30	39	43	120															
95	24	74	53	88	281	54	246	153	180	88	265	247	176	158	160	77	213	58	141	4	43	33	183	223	16	126	84	6	305	240	95	96	22	18	248	96	26	249	43	198	247	95	138	228	2	231	58	105	36	253	121	201	95	227	34	122	83	230	82	160	81	178	52	154	57	116	111	40	344	253	25	180	93	62	97	68	4	105	158	121	139	93	161	33	78	36	119
83	32	168	7	185	89	162	45	44	159	54	44	21	132	37	21	21	154	68	101	57	186	227	121	226	113	119	89	216	208	59	6	57	63	63	111	34	46	93	128	78	39	155	105	128	119	61	168	154	66	80	103	129	107	147	129	125	121	25	174	129	107	98	140	162	153	85	28	54	44	81																	
31	188	252	105	12	126	159	140	88	44	248	146	156	35	201	147	161	140	29	48	95	158	203	28	118	8	72	54	93	220	188	89	79	189	10	45	121	32	35	237	155	180	111	33	128	144	76	138	243	15	30	72	252	69	167	80	188	85	166	75	9	21	25	184	16	240	109	150	43	14	19	255	258	111	176	151	68											
88	285	22	48	189	180	159	35	222	79	1	71	182	34	74	162	129	19	88	104	82	97	119	155	105	25	143	182	29	24	201	65	12	97	102	57	25	198	127	246	108	188	124	24	206	92	12	73	26	188	164	254	188	94	11	14	164	145	74	189	18	4	8	200	89	58	12	36	182	68	33	56	246															
51	54	126	19	246	4	128	23	39	122	25	45	237	34	186	230	198	128	194	76	36	241	52	184	184	24	223	100	207	101	123	62	125	76	82	101	32	122	154	64	35	22	109	121	126	12	136	153	154	126	128	120	182	265	121	234	246	58	103	25	224	89	78	52	21	53	45	25	177	161	126	130																
203	146	255	180	57	134	11	76	59	189	203	140	253	29	186	163	170	153	5	75	144	44	52	107	29	18	67	231	39	209	68	154	38	184	13	27	181	156	177	24	159	155	121	131	8	159	181	143	55	31	8	38	173	247	129	180	21	31	41	57	105	161	201	125	26	183	179	135	56	78	188																	
54	36	64	152	232	40	172	245	233	70	25	165	1	155	138	164	86	261	62	1	51	206	203	204	156	50	1	127	65	136	93	155	128	124	34	109	25	65	32	92	58	99	244	82	180	48	131	11	171	157	32	124	186	180	246	50	208	179	247	111	248	29	23	128	107	160	101	193	227	54	38	18																
25	19	33	249	216	162	97	180	57	45	46	42	285	26	28	35	222	85	39	8	79	65	12	57	20	183	194	79	89	205	53	197	94	242	110	42	147	106	15	17	102	143	37	71	145	124	167	77	192	239	18	21	149	18	230	136	184	203	1	12	161	246	77	41	10	98	162	217	18	133																		
52	172	234	177	83	162	10	183	25	85	94	23	251	72	186	159	170	156	280	30	31	2	240	226	290	175	29	7	79	37	258	76	91	65	43	25	34	43	216	62	185	95	22	93	20	236	18	101	12	162	188	180	53	167	35	42	84	0	229	240	43	180	240	57	197	49	247	31	34	225	31	28	22	8	98	138	92	17	6									
30	185	35	167	233	46	125	23	87	137	154	155	178	246	62	233	128	45	158	139	212	30	88	57	20	185	1	113	12	162	88	73	2	218	58	183	125	17	159	58	183	100	55	152	227	8	203	77	162	227	25	42	54	21	52	57	3	47	199	62	63	27	22	245	95	140	96																					
202	133	188	232	175	20	28	40	37	1	180	254	87	147	239	195	204	141	236	17	85	177	202	209	159	1	4	229	34	186	163	103	85	184	201	4	184	39	31	84	23	188	126	98	10	183	181	180	111	85	35	179	234	212	3	18	174	155	162	111	86	79																										
4	19	22	36	1	48	54	245	196	174	201	31	235	0	29	40	163	143	156	180	14	227	138	88	54	79	234	21	9	243	224	97	186	307	72	181	282	193	162	180	182																																															

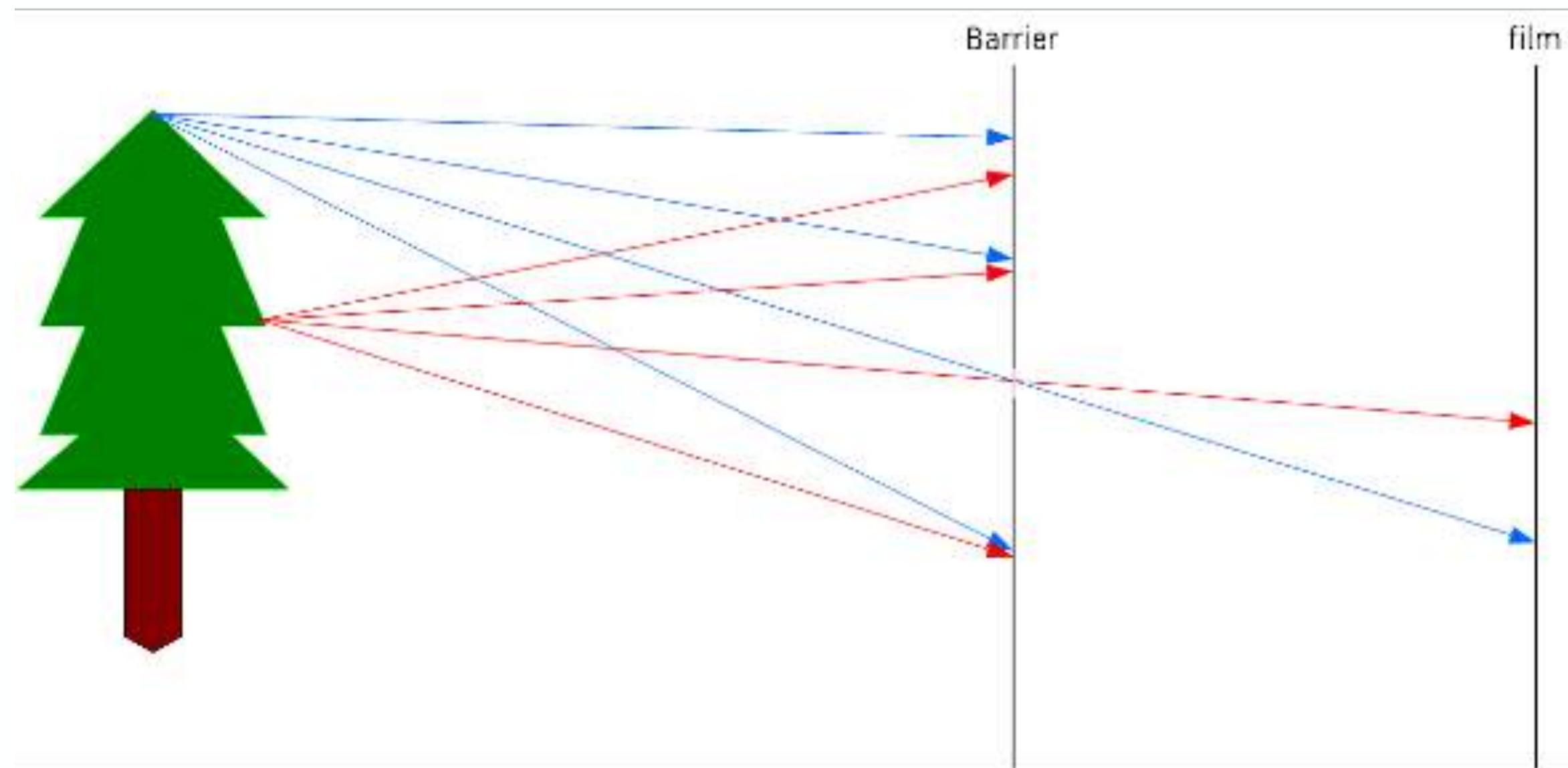
# What are Images?

- Images are 2-D representations of the **visible light spectrum**



# Image Creation

- Digital images are created either by using software on computer (think clip art or logos) or by digital cameras (or scanners).

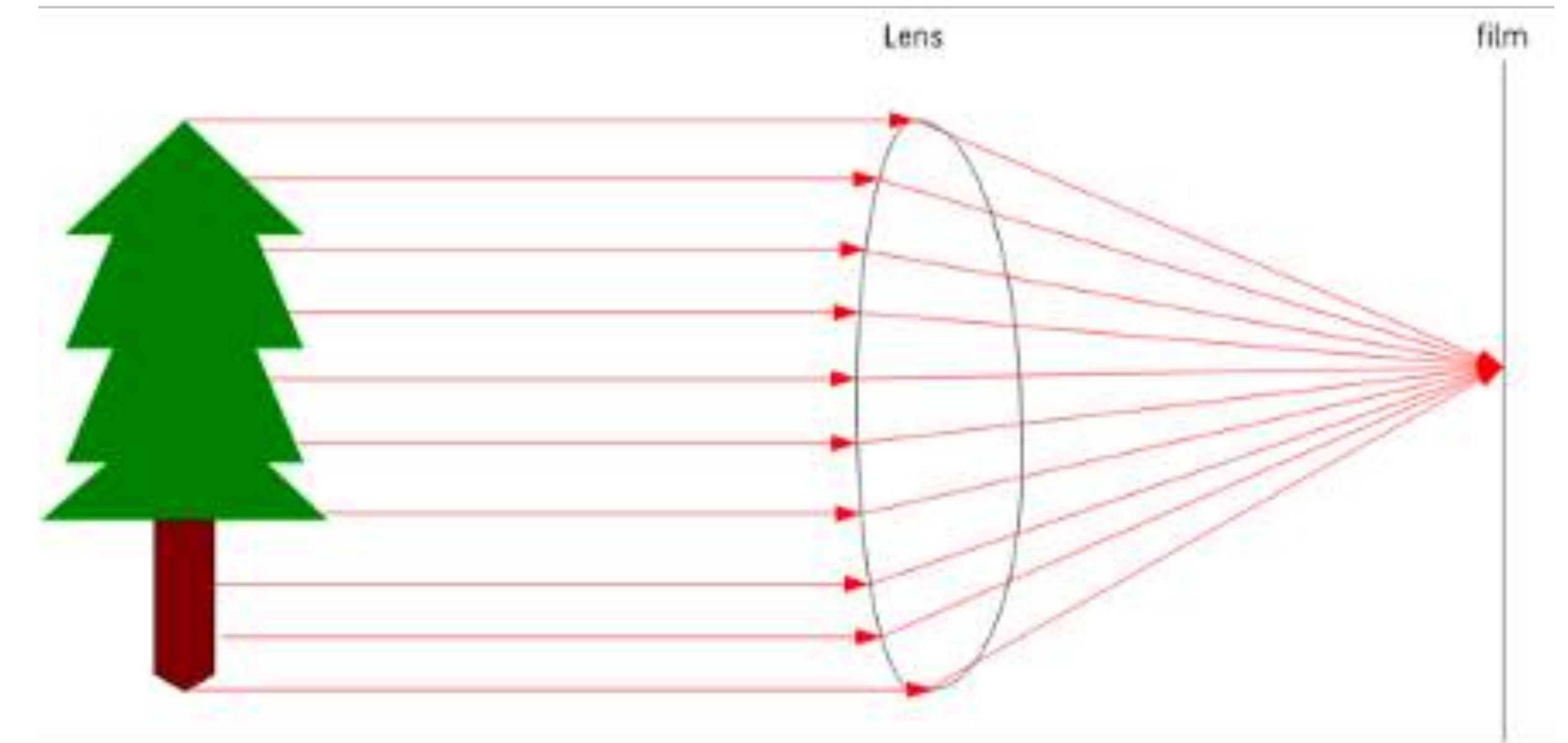


- The pinhole camera model is the basis of how we form an image using a photosensitive sensor

# Modern Cameras

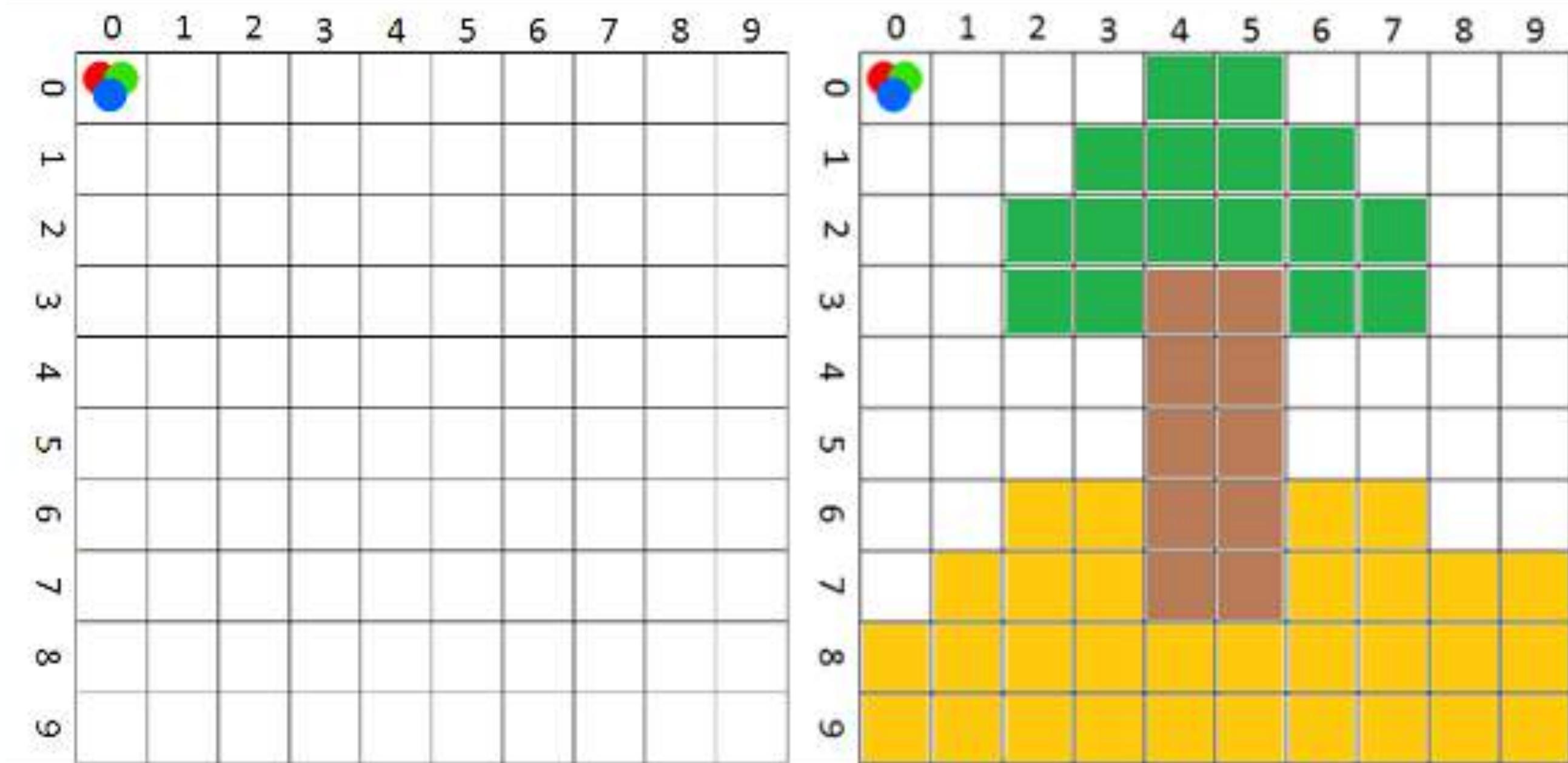
Our eyes as well as cameras use an **adaptive lens** to control many aspects of image formation such as:

- **Aperture Size**
  - Controls the amount of light allowed through (f-stops in cameras)
  - Depth of Field (Bokeh)
- **Lens width** - Adjusts focus distance (near or far)



# Digital Images

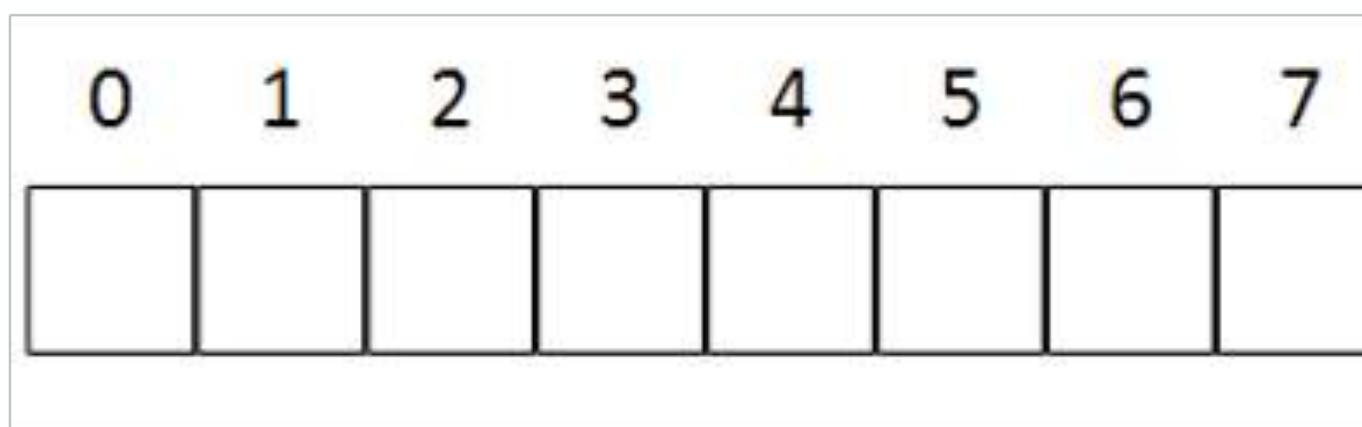
- Computers store images in a variety of formats, however they all involve mixing some of the following such as: colours, hues, brightness or saturation.
  - The one most commonly used in Computer Vision is **RGB (Red, Green, Blue)**
  - Each pixel is a combination of the brightness of blue, green and red, ranging from 0 to 255 (brightest)
  - Yellow is represented as:
    - Red – 255
    - Green – 255
    - Blue - 0



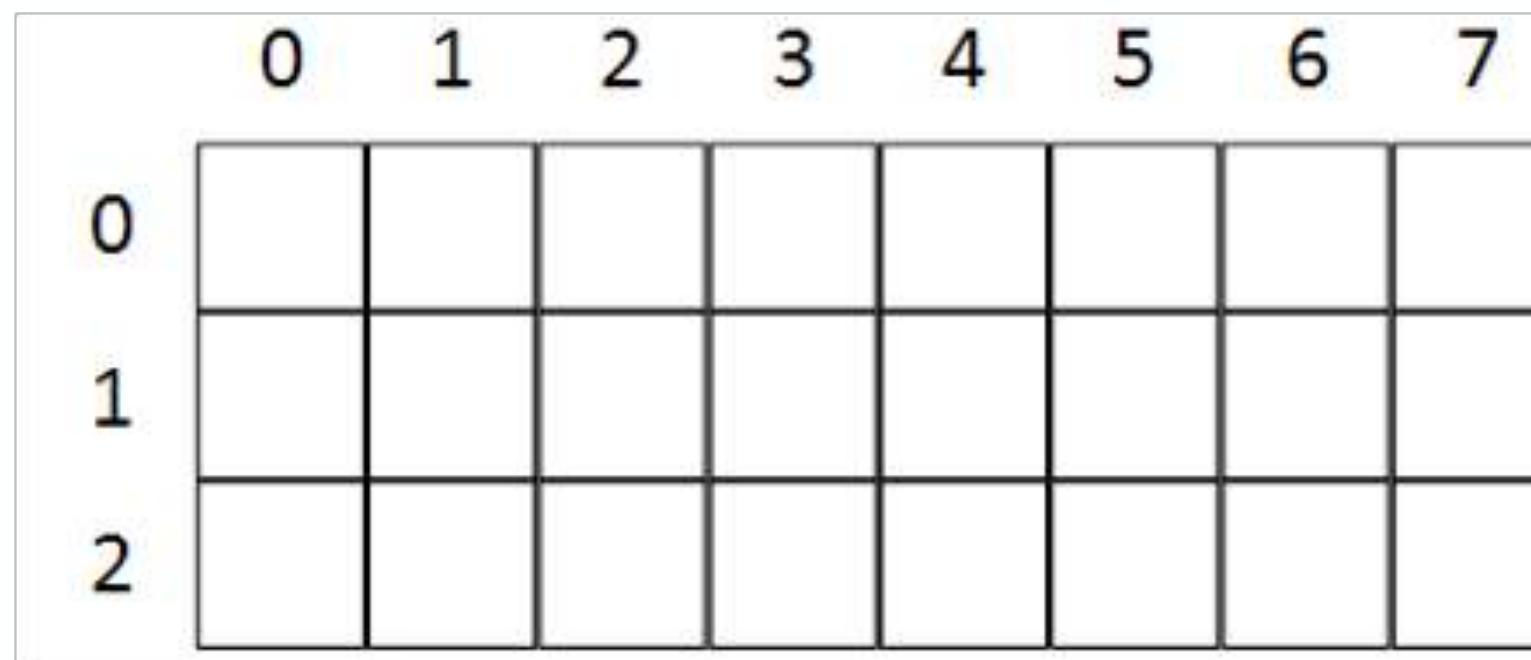
# Digital Images Format

Images are stored in Multi-Dimensional Arrays

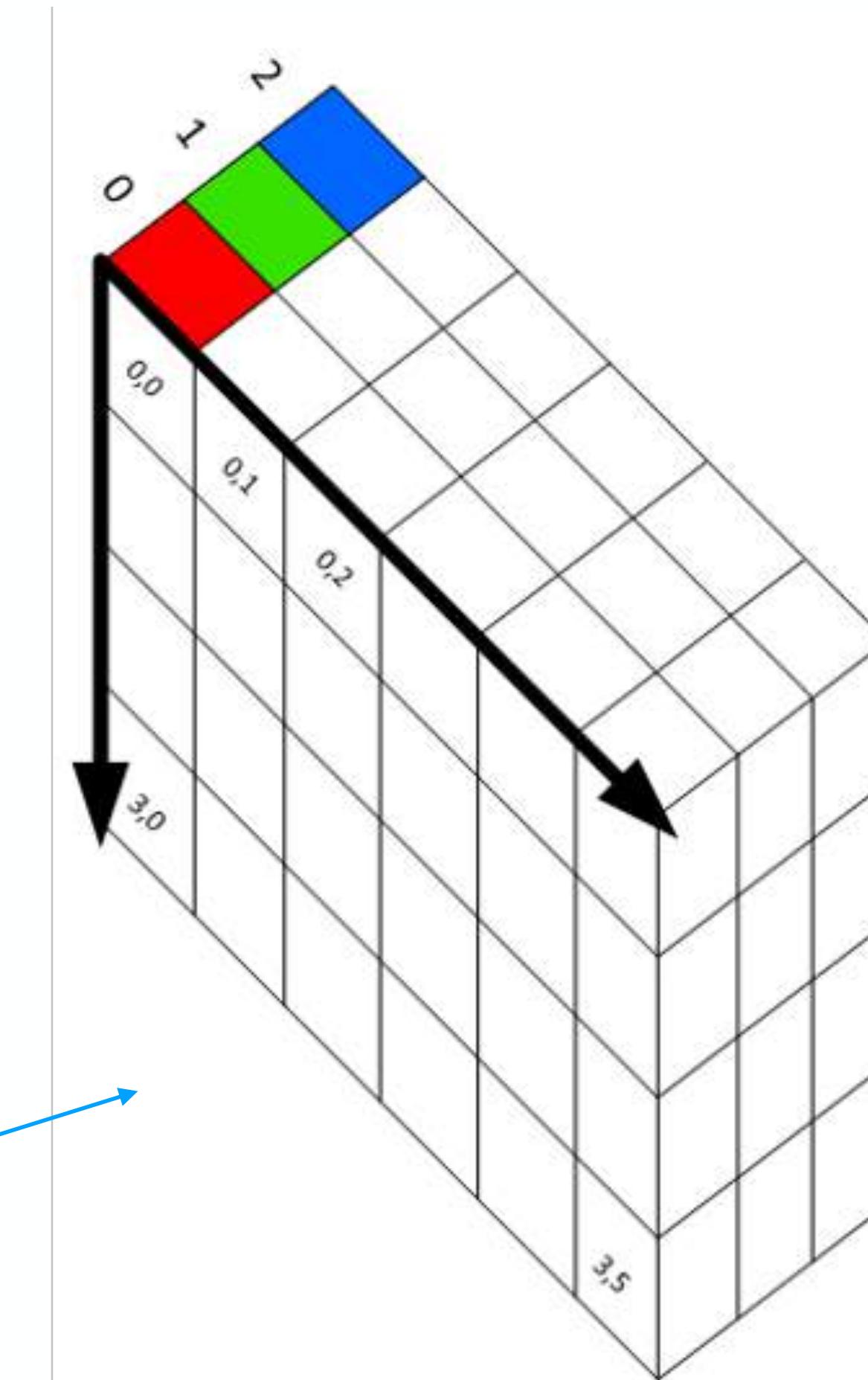
- A 1-Dimensional array looks like this



- A 2-Dimensional array looks like this



- A 3-Dimensional array looks like this



# A Grayscale Image

Sometimes referred to as black and white image

- Info is stored in a 2-Dim arra

# Image File Formats



## Raster images

Pixel-based graphics  
Resolution dependent  
Photos & web graphics

**JPG**

Web & print  
photos and  
quick previews

**GIF**

Animation &  
transparency in  
limited colors

**PNG**

Transparency  
with millions  
of colors

**TIFF**

High quality  
print graphics  
and scans

**RAW**

Unprocessed  
data from  
digital cameras

**PSD**

Layered Adobe  
Photoshop  
design files



## Vector images

Curve-based graphics  
Resolution independent  
Logos, icons, & type

**PDF**

Print files and  
web-based  
documents

**EPS**

Individual  
vector design  
elements

**AI**

Original Adobe  
Illustrator  
design files

**SVG**

Vector files  
for web  
publishing

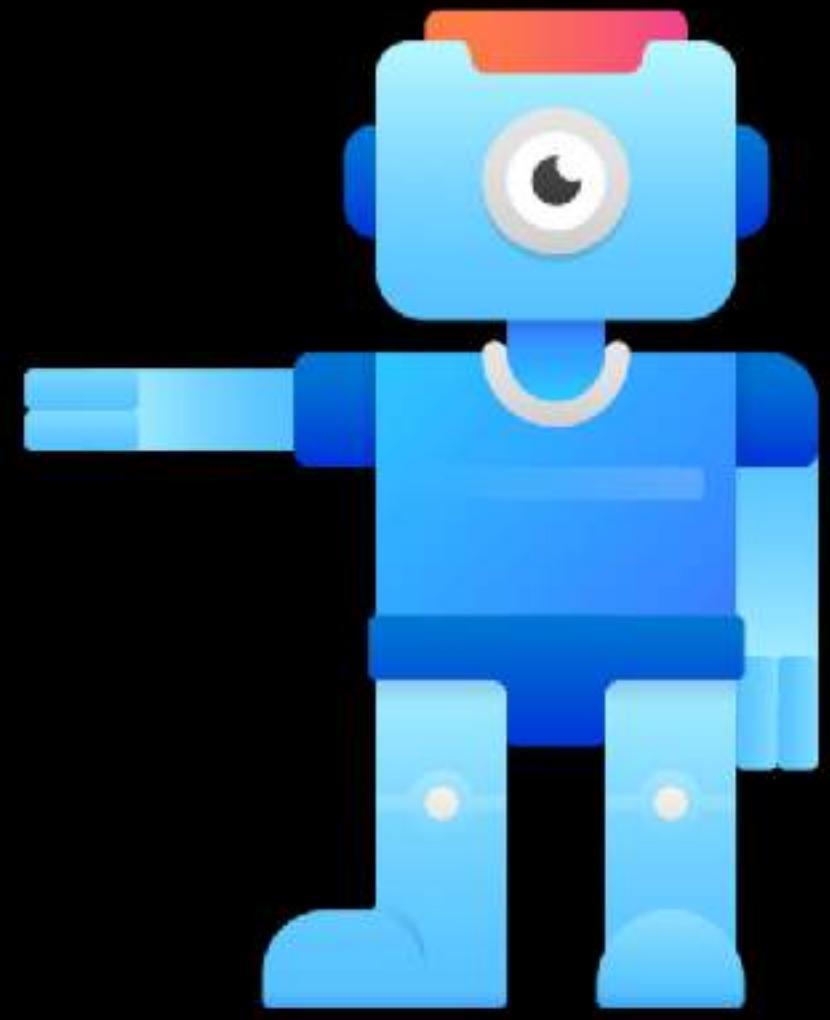


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Let's get Started with Google Colab and the Course setup**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Introduction to Convolutional Neural Networks

## A Gentle Overview of CNNs

# Understanding Images

Predicted to be a Cat



Predicted to be a Dog

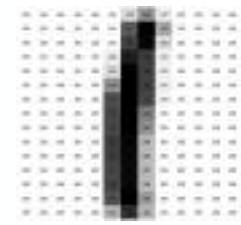


# What Do We See?

- Many things indicated it was a cat or dog
- Whiskers, shape, eyes, fur, colour etc.
- Can an Algorithm or Predictive Model do this?

# **Convolutional Neural Networks**

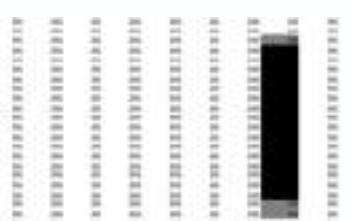
# Building an Intuition for CNNs



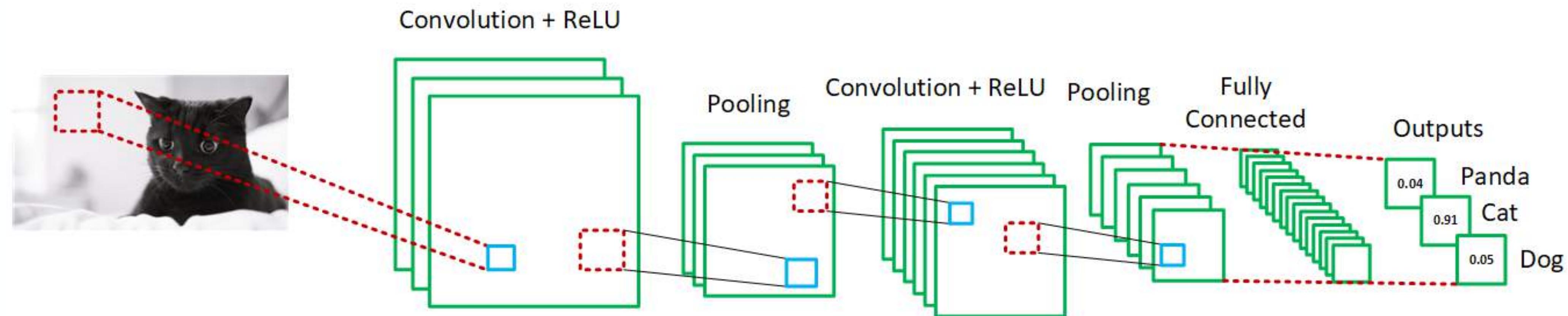
- What digit is this?
- How would we program a computer to know this?
  - Overall shape?



- Maybe if something lies in this region only it's a one?
- But what if it's shifted?



# What If We Had Filters that Could Scan All Parts of An Image?

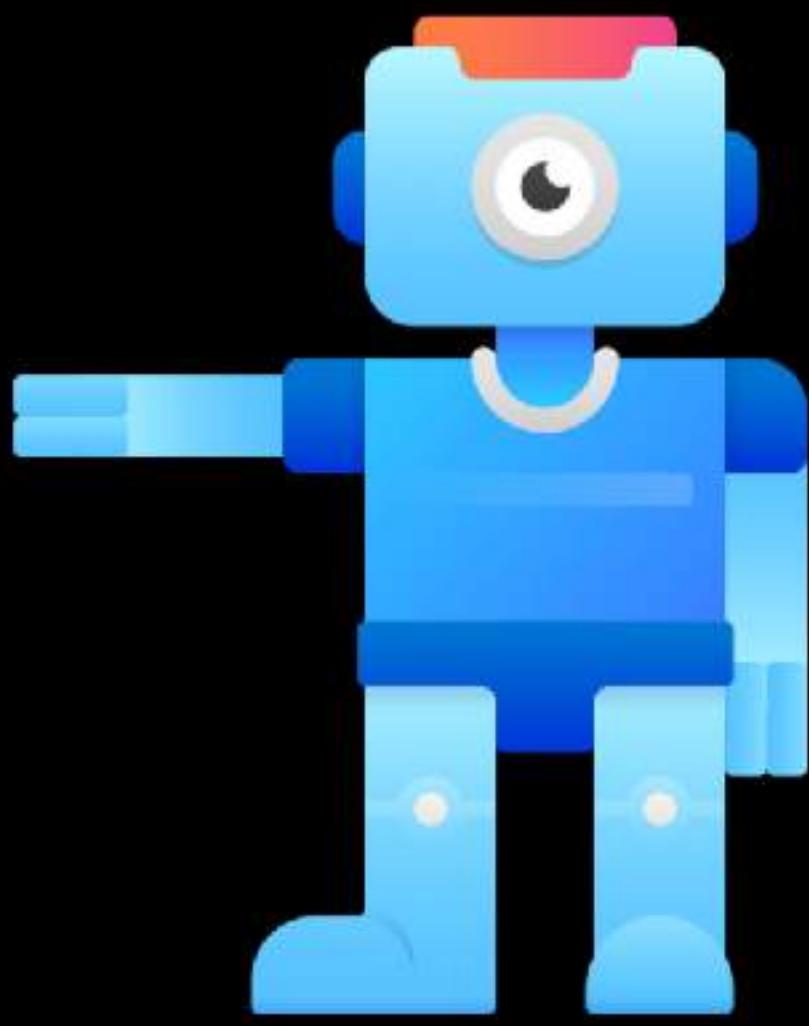


# Overview of our CNN Chapter

- Convolutions
- Feature Detectors
- 3D Conv Filters - Convolution on Color Images
- Kernel Size and Depth
- Padding
- Stride
- Activation Layer - ReLU
- Pooling
- Fully Connected Layer
- Softmax
- Building a CNN
- Parameter Counts in CNNs
- Why CNNs Work so Well For Images
- The Training Process Part 1 - Loss Functions
- The Training Process Part 2 - Back Propagation
- The Training Process Part 3 - Gradient Descent
- Optimisers and Learning Rate Schedules
- Summary of CNNs

# Convolutions

How We Detect Features in Images



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# The Convolution Operation

- Mathematical term to describe the process of combining **two** functions to produce a **third**
- In our situation, the output is called a **Feature Map**
- We use a matrix, called a **Filter** or **Kernel** that is applied to our Image
- So the first ‘**Function**’ is the **image** that is combined with the **Kernel** or **Filter** which produces a **Feature Map**

Image  $\times$  Kernel = Feature Map

# Example of a Convolution Operation

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	3
2	1	1

Input Image

Filter or Kernel

Output or Feature Map



**Recall grayscale images are just intensity values ranging from black to white**

# Example of a Convolution Operation

$$(1 \times 0) + (0 \times 1) + (1 \times 0) + (1 \times 1) + (0 \times 0) + (0 \times -1) + (0 \times 0) + (1 \times 1) + (1 \times 0) = 2$$

1x0	0x1	1x0	0	1
1x1	0x0	0x-1	1	1
0x0	1x1	1x0	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2		

Input Image

Filter or Kernel

Output or Feature Map

# Example of a Convolution Operation

$$(0 \times 0) + (1 \times 1) + (0 \times 0) + (0 \times 1) + (0 \times 0) + (1 \times -1) + (1 \times 0) + (1 \times 1) + (0 \times 0) = 1$$

1	0x0	1x1	0x0	1
1	0x1	0x0	1x-1	1
0	1x0	1x1	0x0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	

Input Image

Filter or Kernel

Output or Feature Map

# Example of a Convolution Operation

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

\*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1

Output or Feature Map

# Example of a Convolution Operation

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

0	1	0
1	0	-1
0	1	0

\*

=

2	1	-1
-1		

Filter or Kernel

Output or Feature Map

# Example of a Convolution Operation

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	

Input Image

Filter or Kernel

Output or Feature Map

# Example of a Convolution Operation

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

0	1	0
1	0	-1
0	1	0

\*

=

2	1	-1
-1	1	3

Filter or Kernel

Output or Feature Map

# Example of a Convolution Operation

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	3
2		

Input Image

Filter or Kernel

Output or Feature Map

# Example of a Convolution Operation

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	3
2	1	

Input Image

Filter or Kernel

Output or Feature Map

# Example of a Convolution Operation

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	3
2	1	1

Input Image

Filter or Kernel

Output or Feature Map

# Image Features

- Our feature maps are actually **Feature Detectors**
- **Why did we do this?**
- Because Convolution Filters or Kernels **detect features** in images

# Our Convolution Filter as an Edge Detector



1	1	0	0	0
1	1	0	0	0
1	1	0	0	0
1	1	0	0	0
1	1	0	0	0

\*

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

=



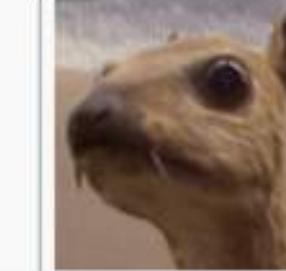
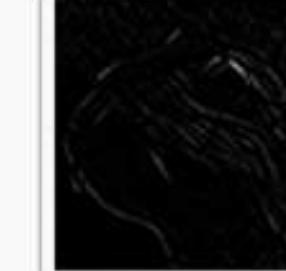
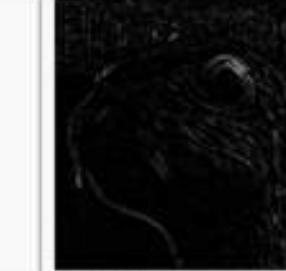
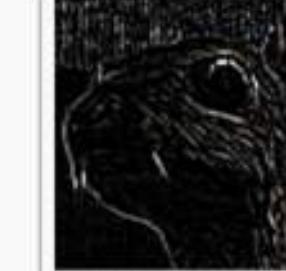
3	3	0
3	3	0
3	3	0

Input Image

Filter or Kernel

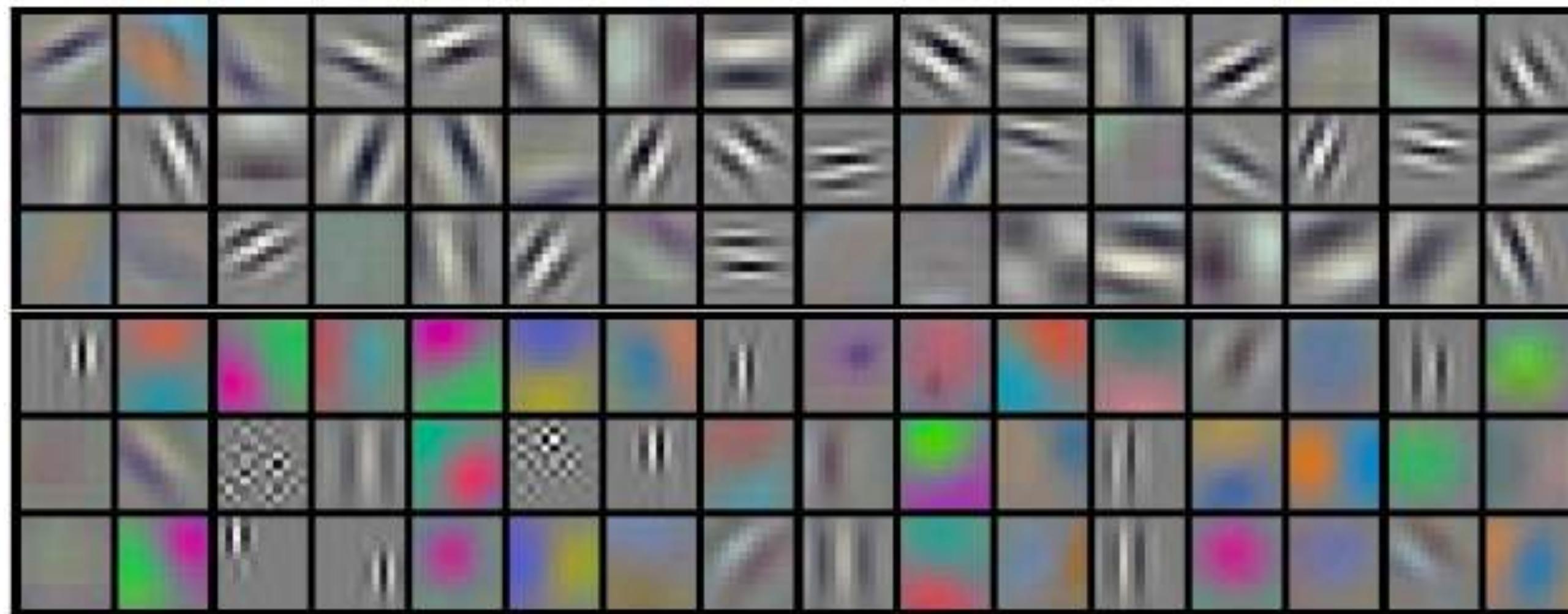
Output or Feature Map

# Convolution Filters Detect Many Features

Operation	Filter	Convolved Image
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 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

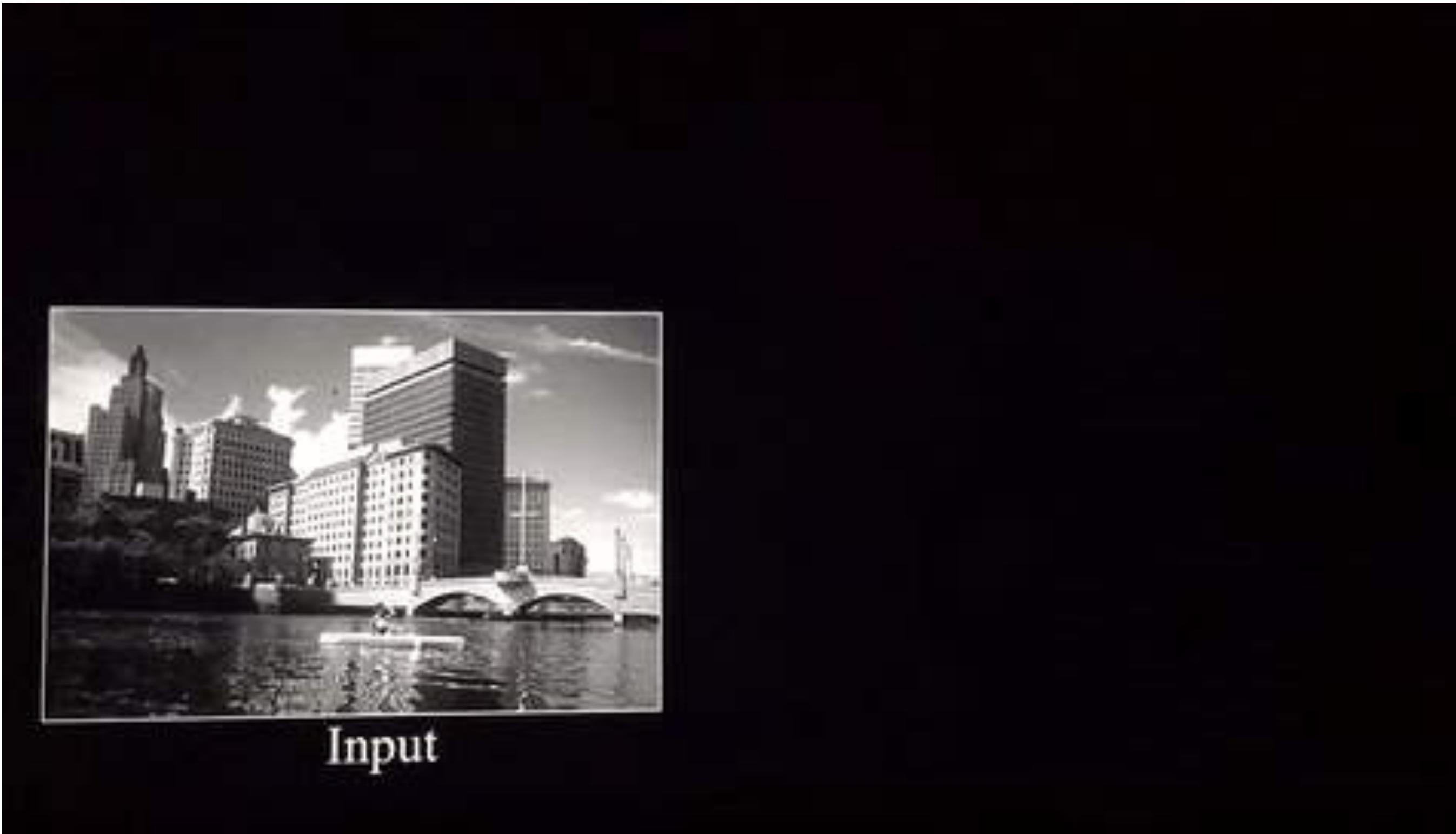
[https://en.wikipedia.org/wiki/Kernel\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

# Convolution Filters Detect Many Features



Example filters learned by Krizhevsky et al.

# Convolution Filters Detect Many Features



Input

Source – Deep Learning Methods for Vision  
[https://cs.nyu.edu/~fergus/tutorials/deep\\_learning\\_cvpr12/](https://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/)

# Calculating Feature Map Size

$$\text{Feature Map Size} = n - f + 1 = m$$

$$\text{Feature Map Size} = 5 - 3 + 1 = 3$$

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	3
2	1	1

$5 \times 5$

$n \times n$

$3 \times 3$

$f \times f$

$3 \times 3$

$m \times m$



# MODERN COMPUTER VISION

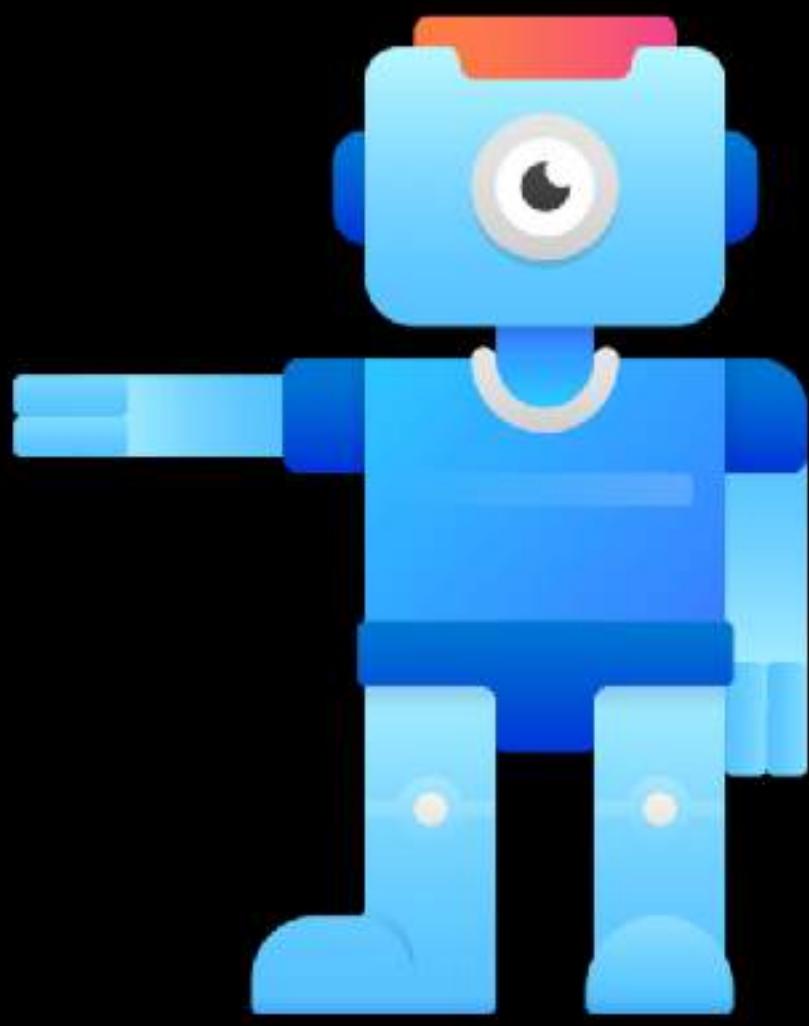
BY RAJEEV RATAN

# Next...

**Feature Detectors**

# Feature Detectors

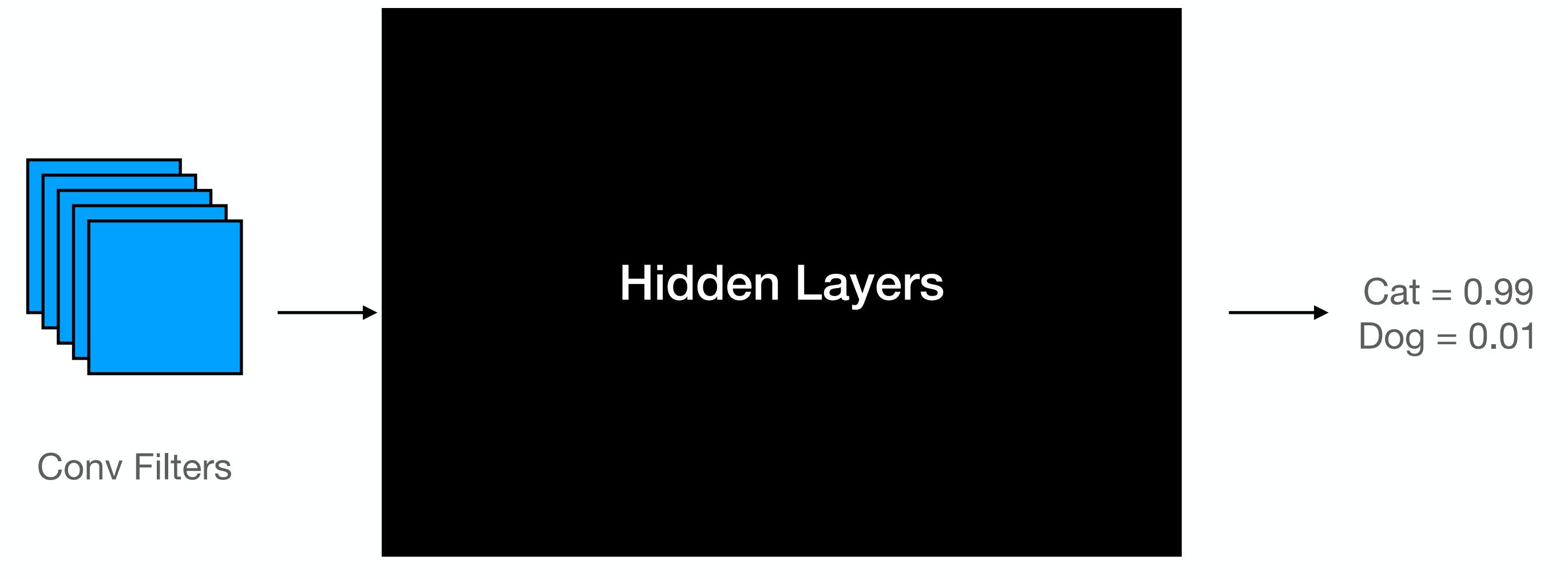
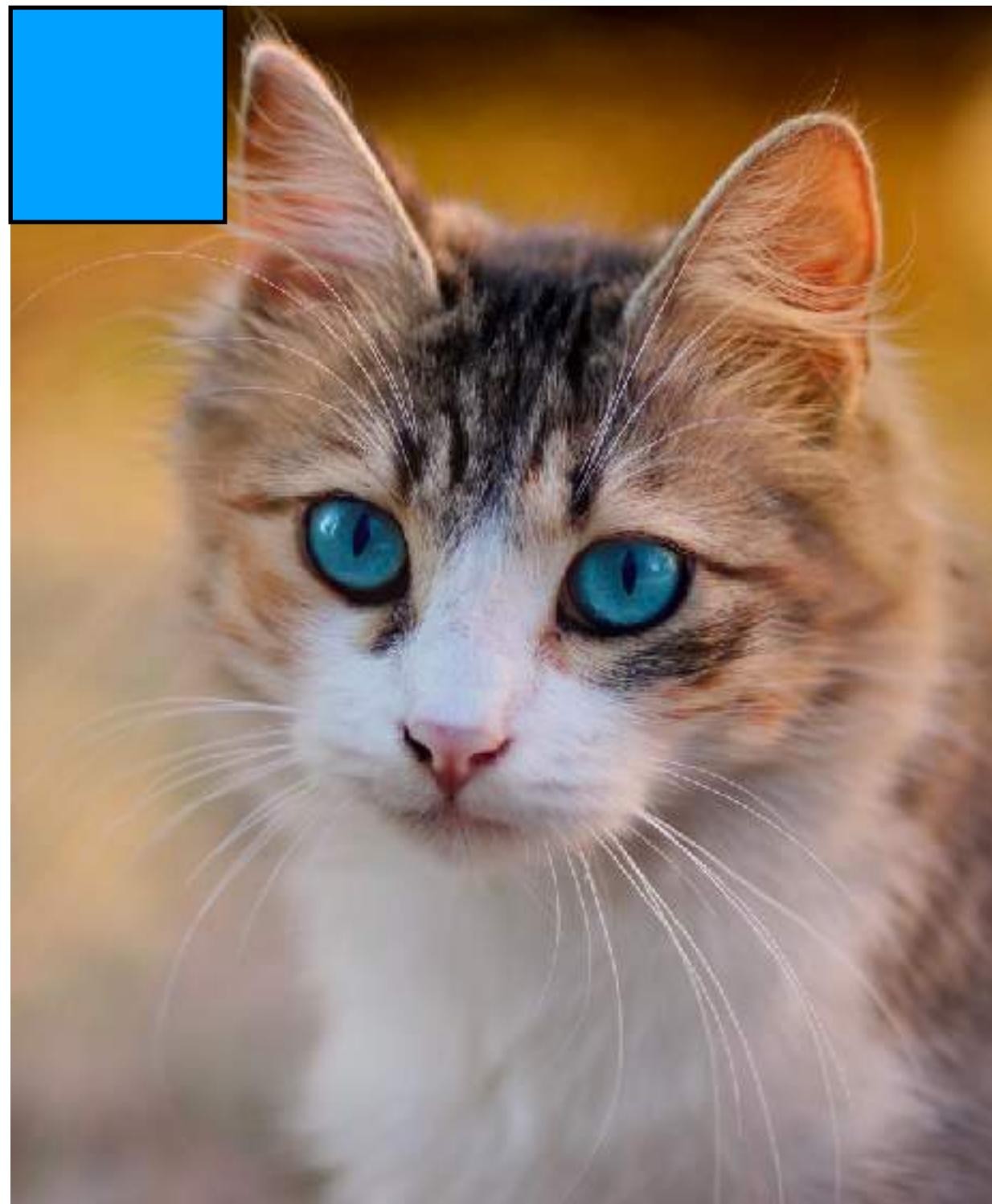
How Feature Detectors Help Classify Images



MODERN  
COMPUTER  
VISION

BY RAJEEV RATAN

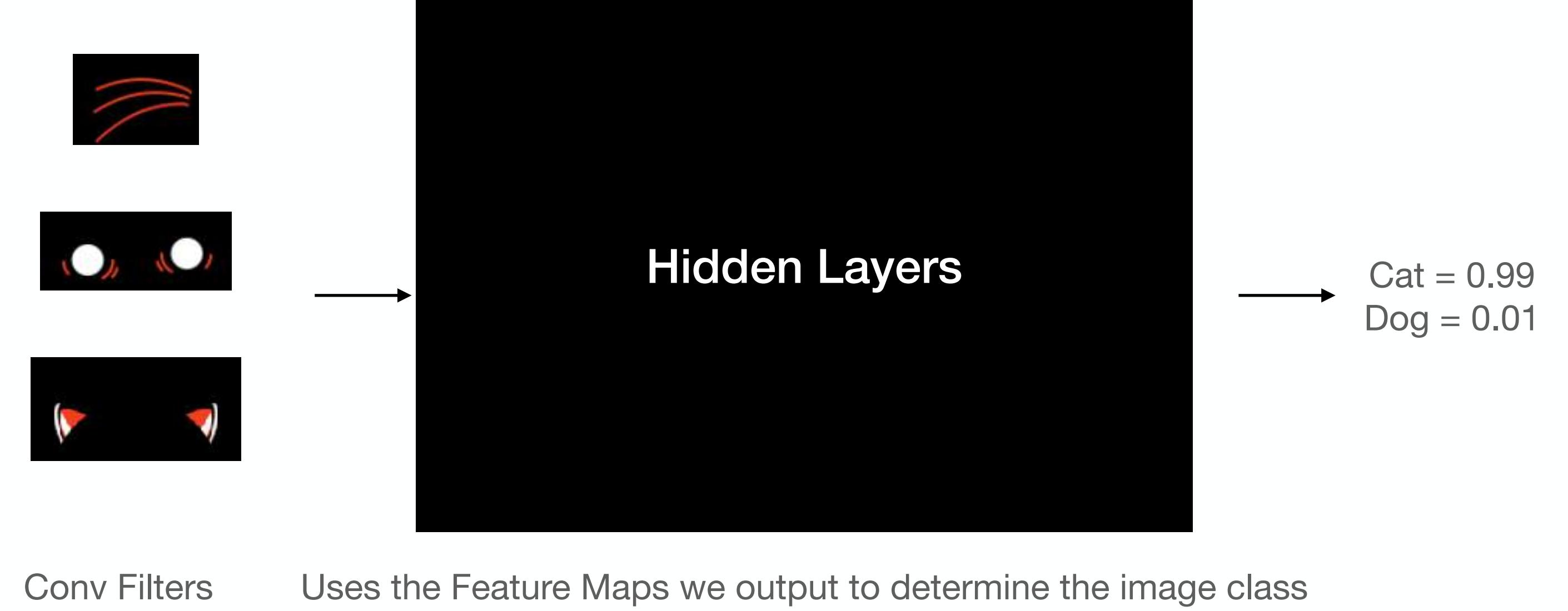
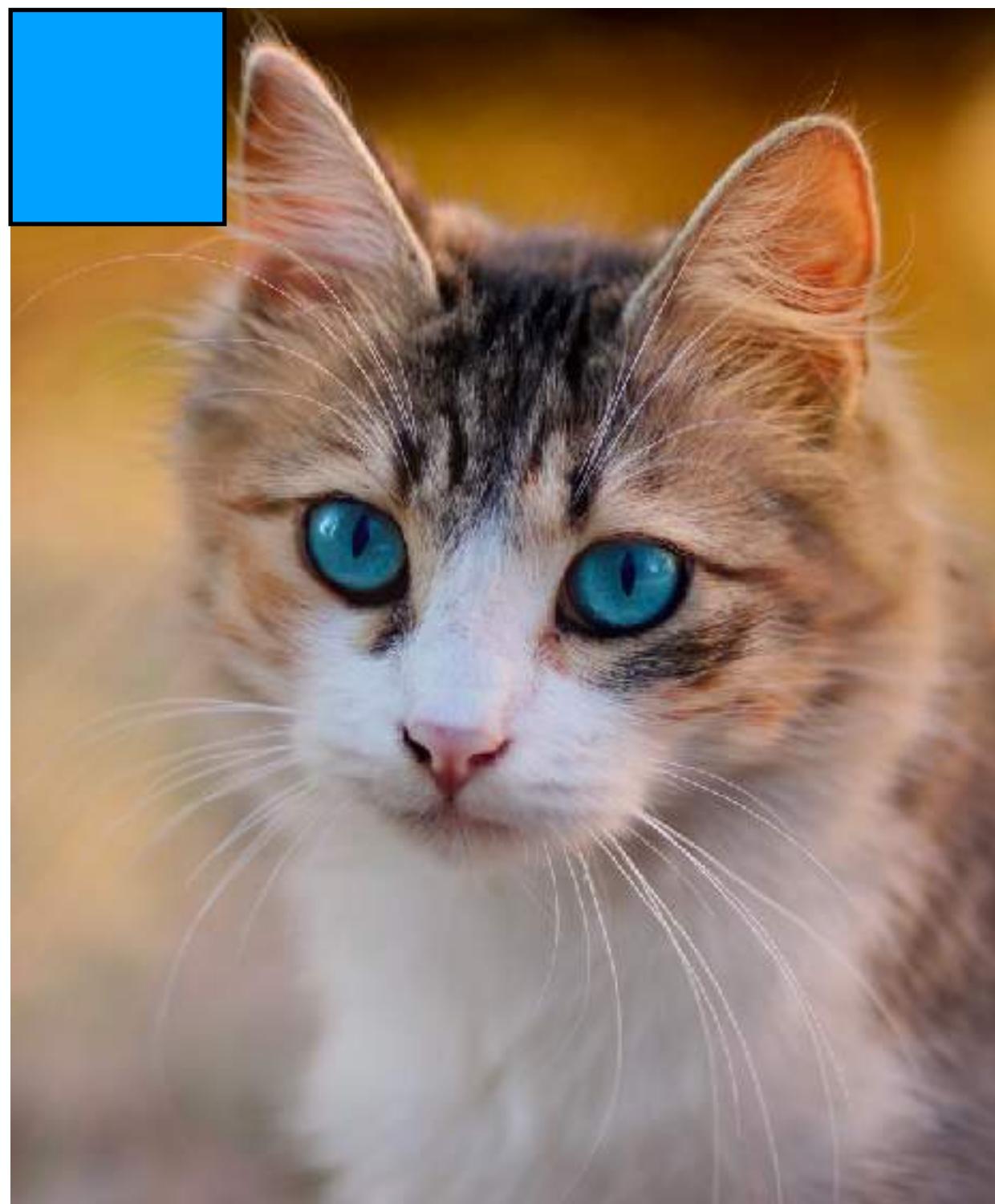
# How Classifiers Use Feature Detectors



Uses the Feature Maps we output to determine the image class

We perform Convolution operations with the input image and our filters

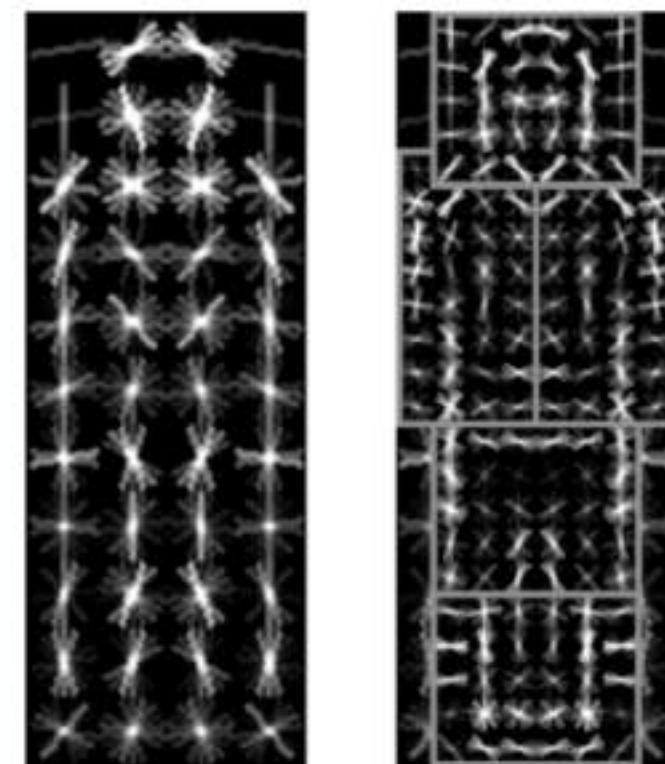
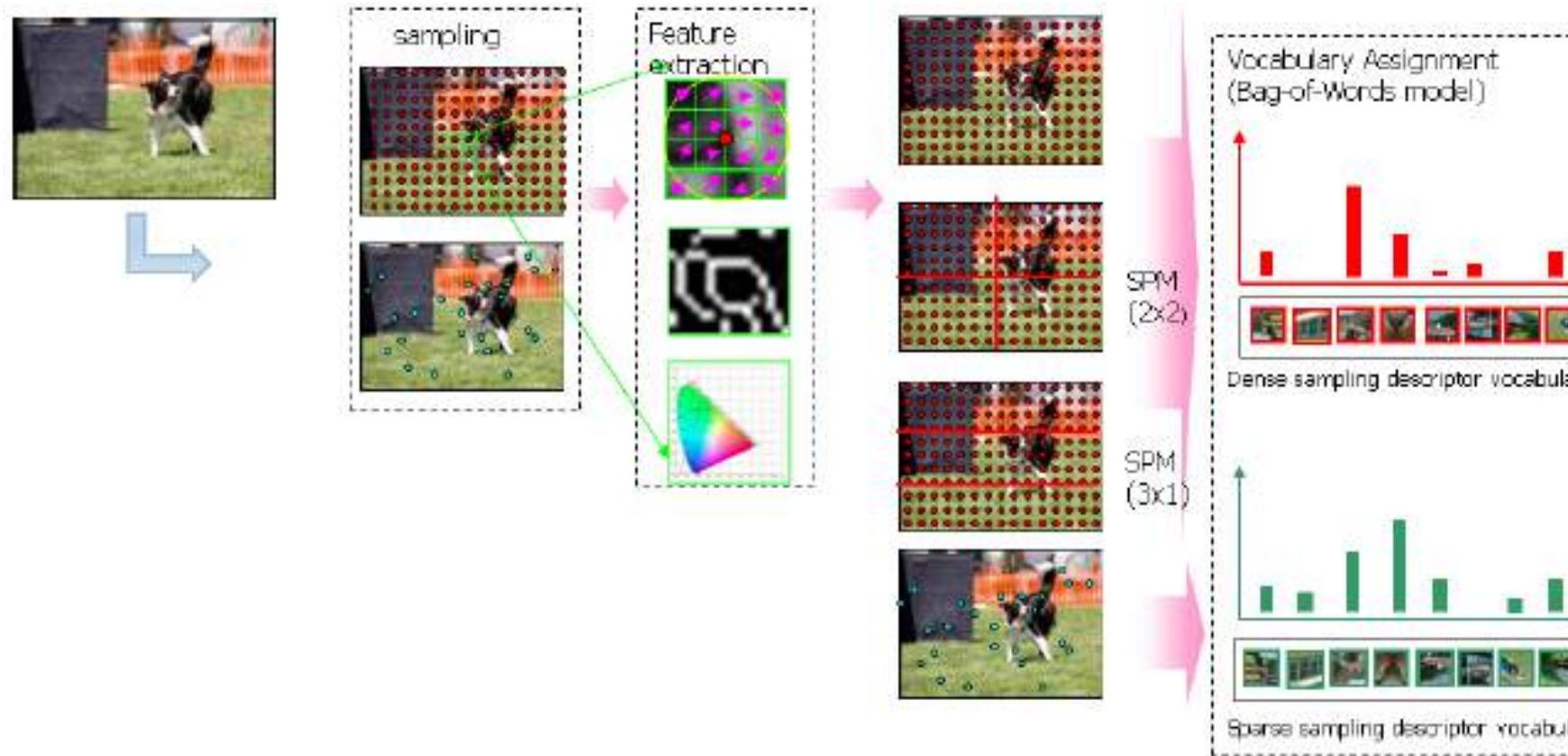
# How Classifiers Use Feature Detectors



We perform Convolution operations with the input image and our filters

# In the Past Hand Engineered Features Were Often Used

## HoGs, SIFT, SURF, ORB, PHOG, Haar etc.



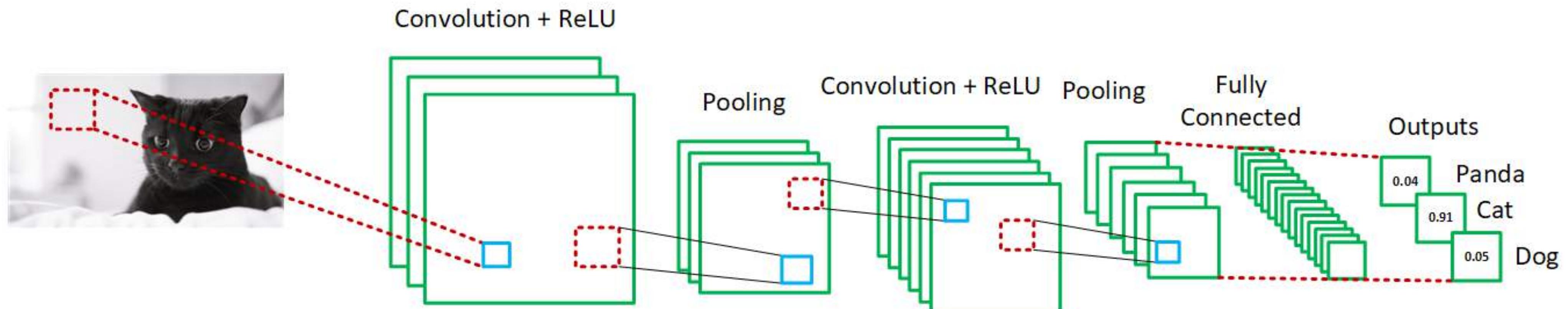
Yan & Huang  
(Winner of PASCAL 2010 classification competition)

Felzenszwalb, Girshick,  
McAllester and Ramanan, PAMI 2007

Hand crafting Features is **VERY** hard, messy and leads to often poor results...

CNNs solved this by having the ability to **Learn Features**

# The Layers of a CNN



- Convolution
- ReLU
- Pooling Layers
- Fully Connected or Dense Layer
- Output

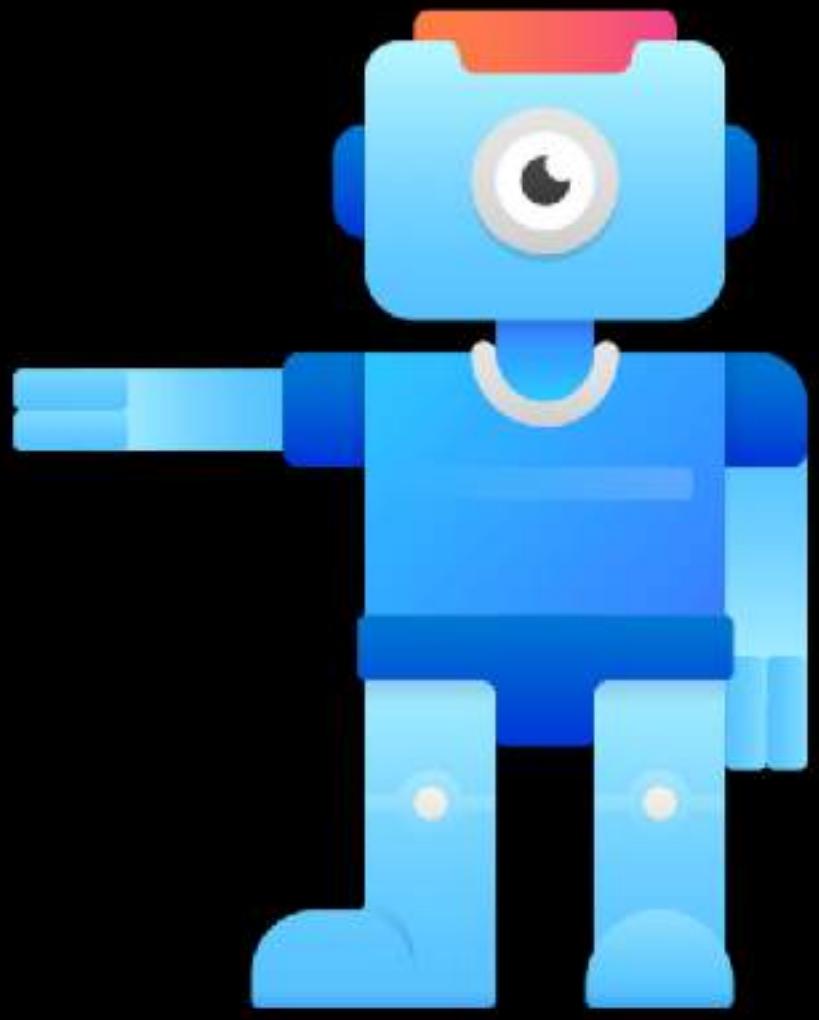


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Convolution on Color Images**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Convolution on Color Images

How we use 3D Volumes of Convolution Filters

# Convolution on our Grey Scale Image



1	1	0	0	0
1	1	0	0	0
1	1	0	0	0
1	1	0	0	0
1	1	0	0	0

\*

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

=



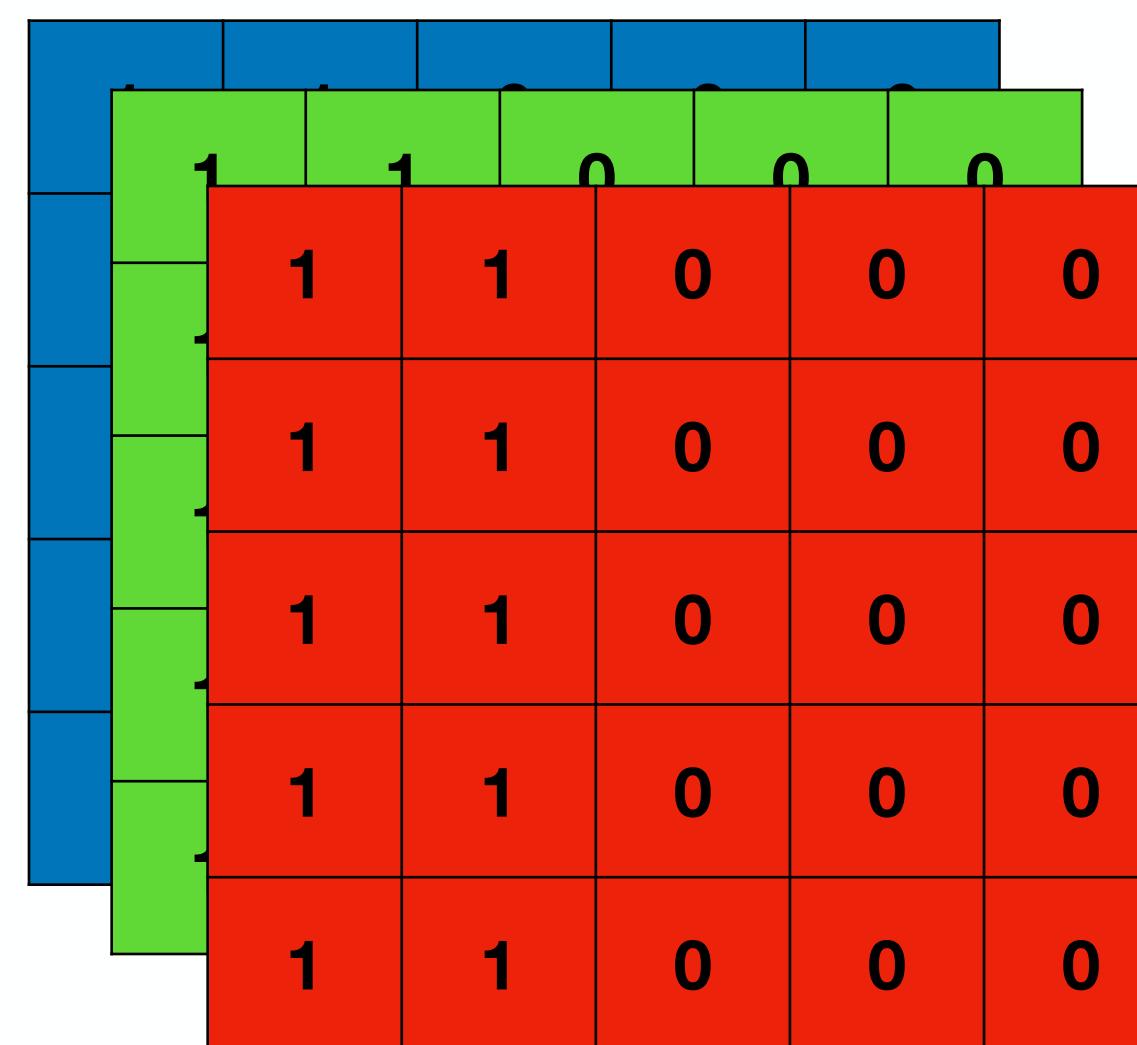
Input Image

Filter or Kernel

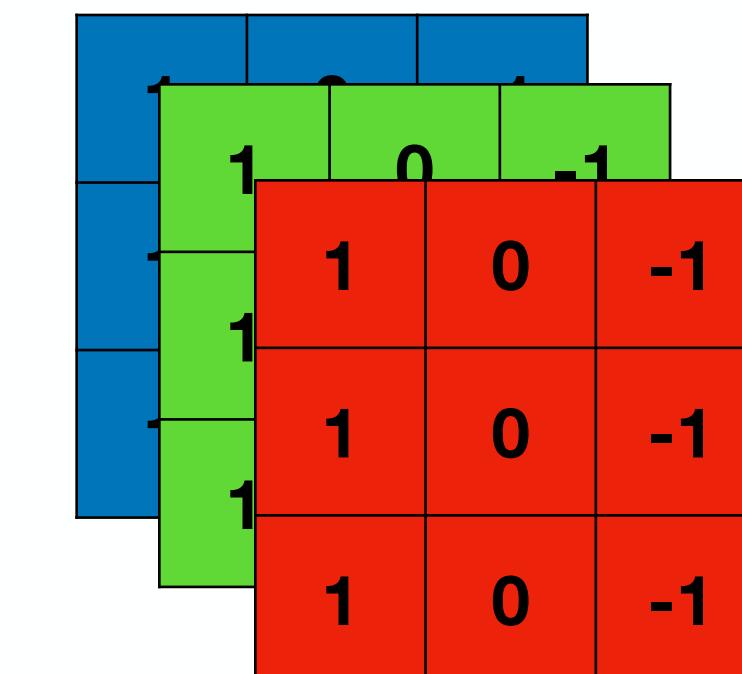
Output or Feature Map

3	3	0
3	3	0
3	3	0

# Convolution Operations on Color Images



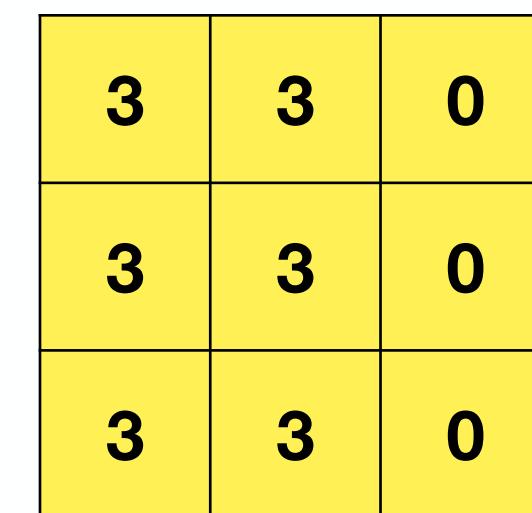
\*



Filter or Kernel

Sum all the numbers to get the Feature Map Output

=



Input Image

Output or Feature Map

# Advantages of Having a Filter For Each Colour

The diagram illustrates the convolution operation between an input image and a filter or kernel. The input image is a 7x6 grid with a 3x3 green receptive field highlighted. The filter or kernel is a 3x3 grid. The result of the convolution is a 3x3 output or feature map.

**Input Image**

1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0

**\***

**Filter or Kernel**

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

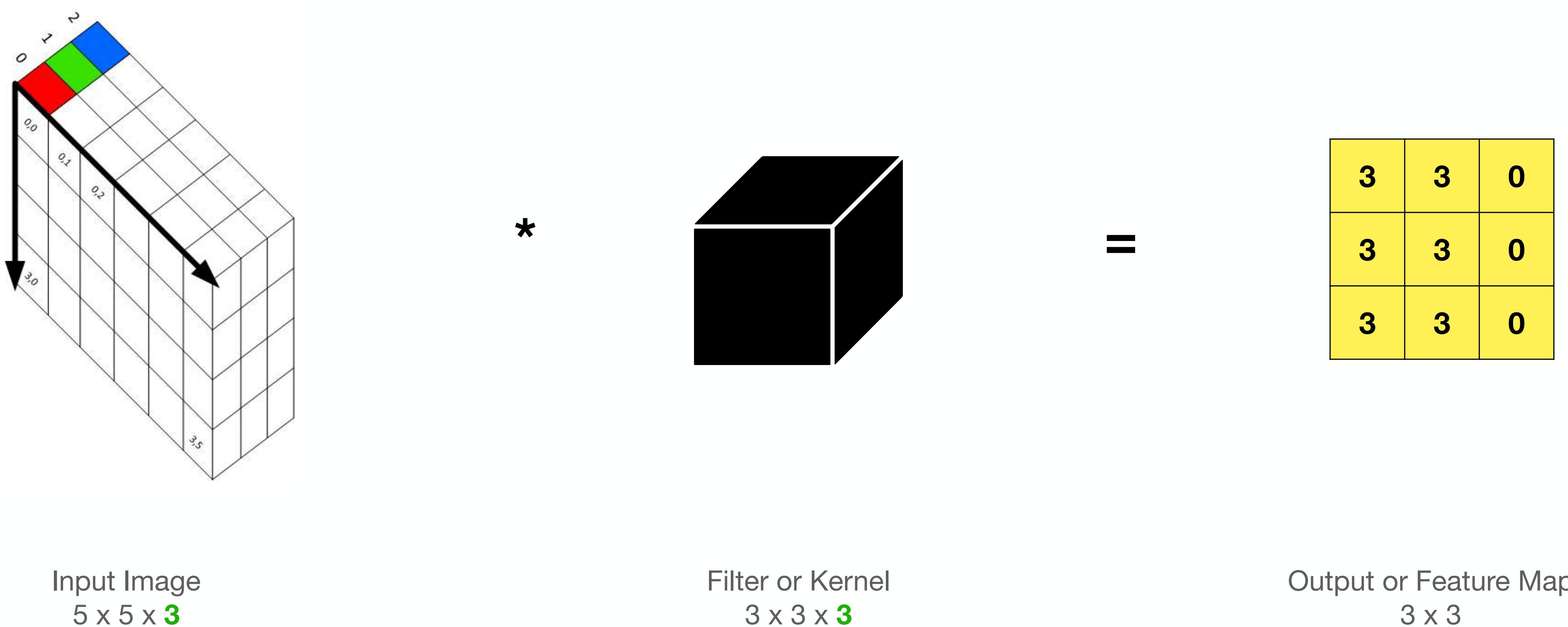
**=**

**Output or Feature Map**

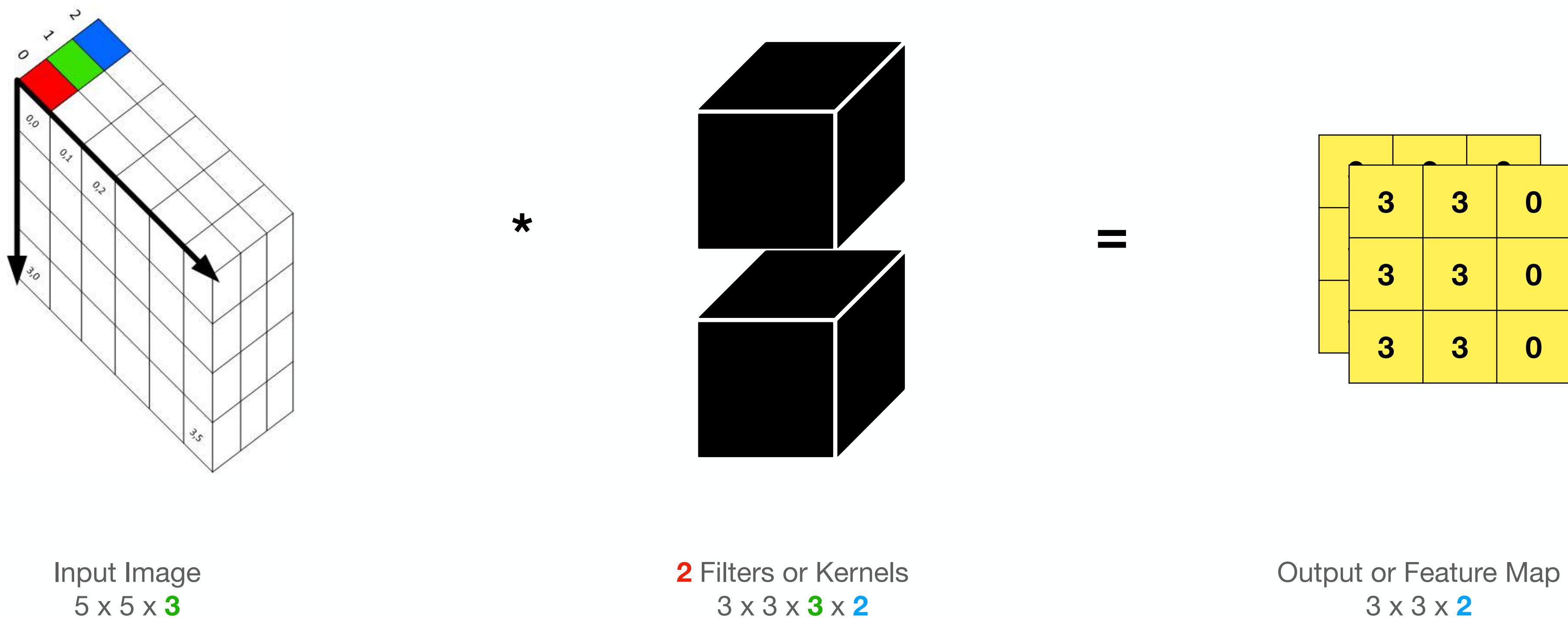
3	3	0
3	3	0
3	3	0

- We can detect features that are specific to a colour

# Considered 3D Volumes



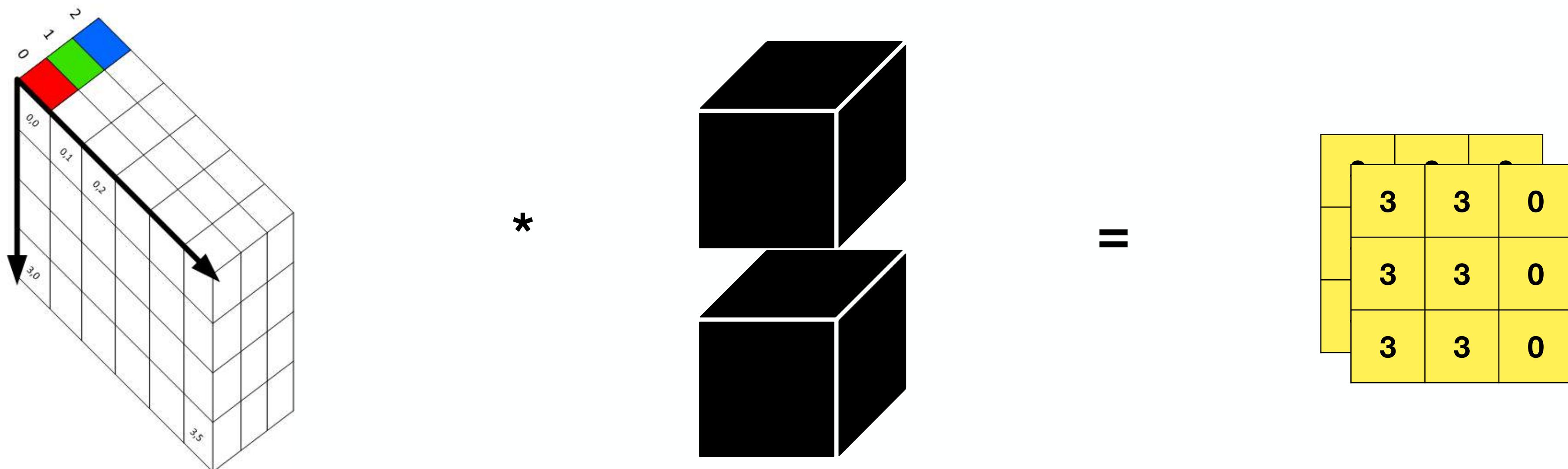
# How Multiple Filters Affect Our Output



# Calculating Output Size for 3D Conv Volumes

$$(n \times n \times n_c) * (f \times f \times n_c) = (n - f + 1) \times (n - f + 1) \times n_f$$

$$(5 \times 5 \times 3) * (3 \times 3 \times 3) = 3 \times 3 \times 2$$



Input Image  
 $5 \times 5 \times 3$

**2** Filters or Kernels  
 $3 \times 3 \times 3 \times 2$

Output or Feature Map  
 $3 \times 3 \times 2$

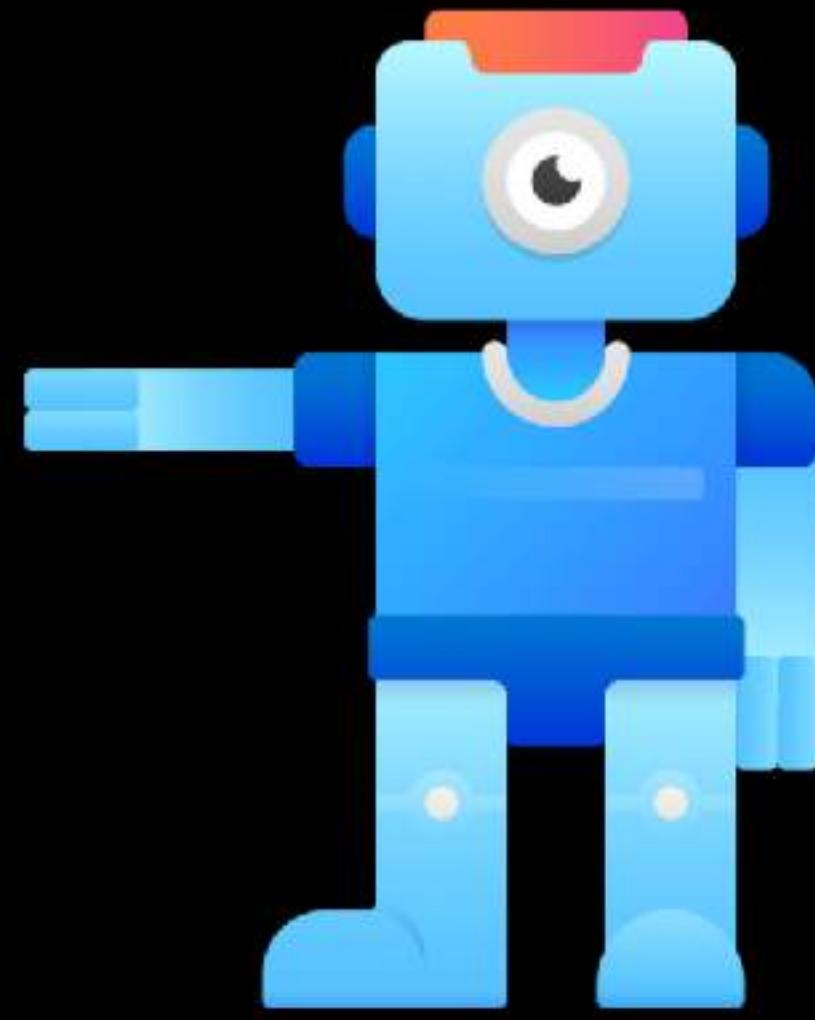


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Kernel Size and Depth**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Kernel Size and Depth

We look at some of parameters that define our Conv Filter

# Parameters that Control the Conv Filter

- Kernel Size ( $k \times k$ )
- Depth (1 for grayscale or 3 for RGB)
- Stride
- Padding

# Sizing Convolution Filters

- In the previous example we used a  $3 \times 3$  Filter or Kernel
- Can we use other sizes?

$$\begin{array}{|c|c|c|c|c|} \hline 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 \\ \hline \end{array} \quad * \quad
 \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \quad = \quad
 \begin{array}{|c|c|c|} \hline 3 & 3 & 0 \\ \hline 3 & 3 & 0 \\ \hline 3 & 3 & 0 \\ \hline \end{array}$$

# Sizing Convolution Filters

- Yes we can use larger filters/kernels

$$\text{Feature Map Size} = n - f + 1 = m$$

$$\text{Feature Map Size} = 6 - 5 + 1 = 2$$

1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0

\*

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

=

3	3
3	3

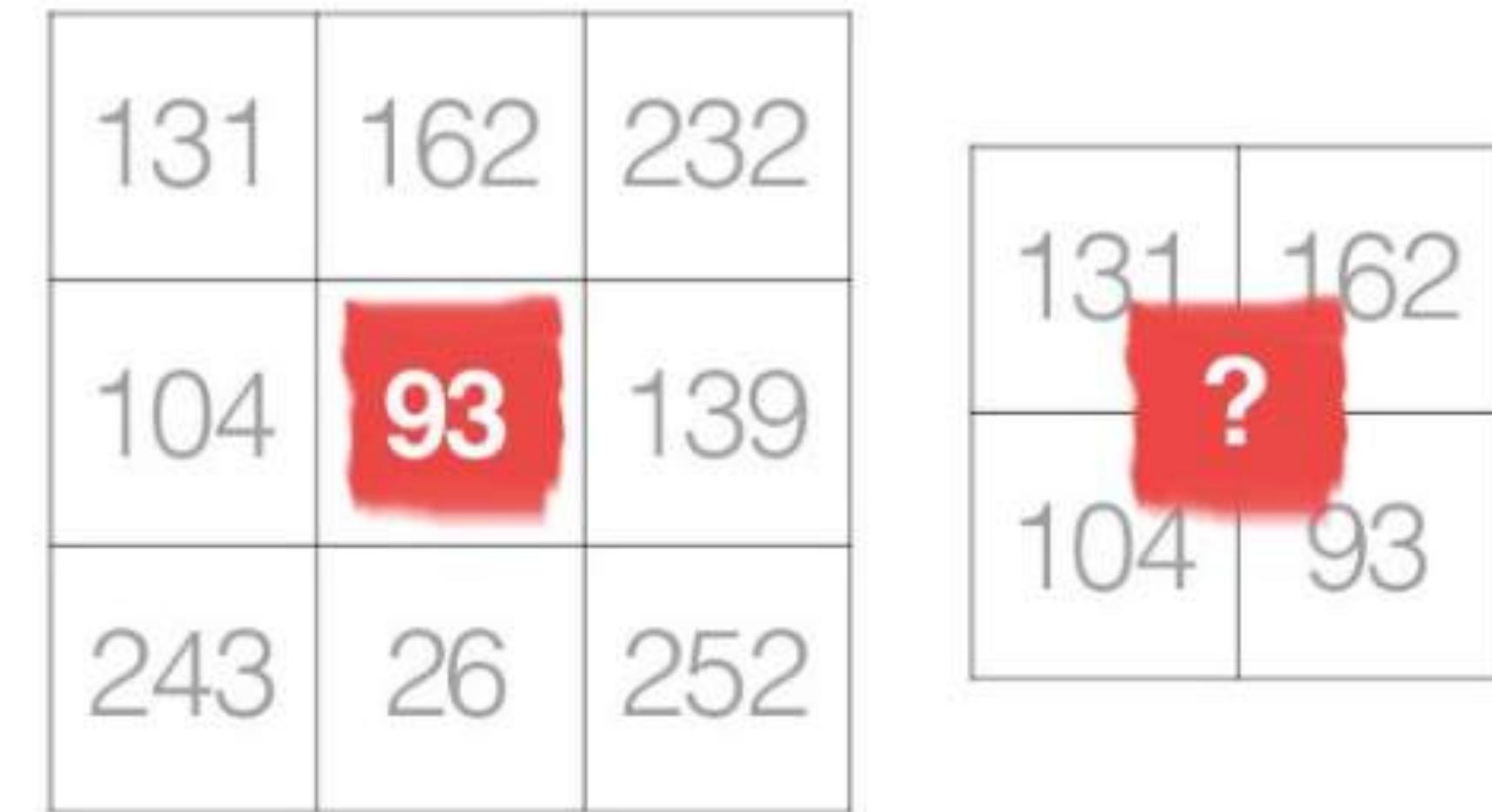
6 x 6

5 x 5

Output or Feature Map

# Odd Number vs Even Number for Filter Dimensions

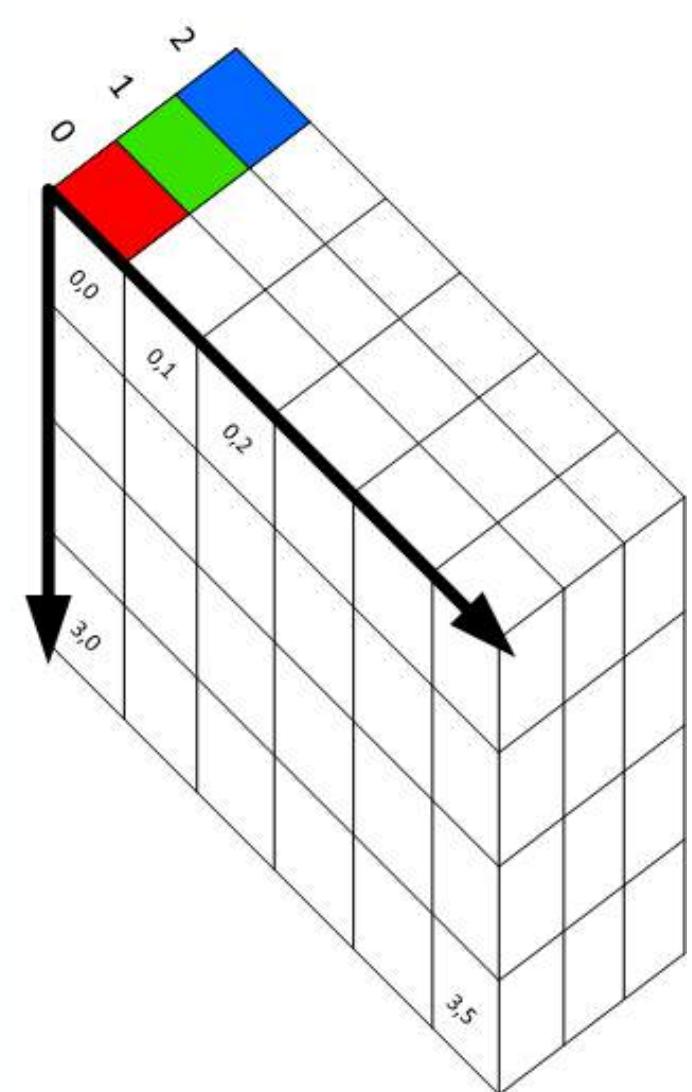
- Odd sized filters are symmetrical around the centre pixel or anchor point
- A lack of symmetry here results in distortions across layers



source:<https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363>

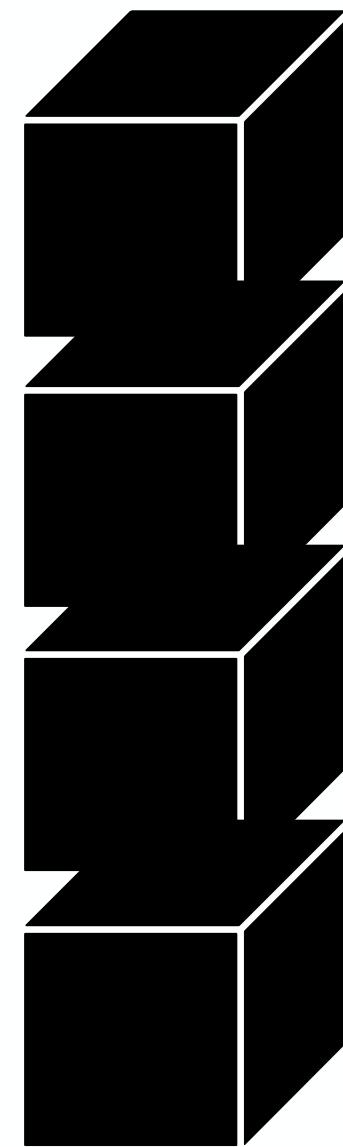
# Depth

- Depth typically refers to the **colour channels**
- However, in some nomenclature it can refer to the 3rd dimension of any layer in our CNN e.g. our Feature Map has depth of 4



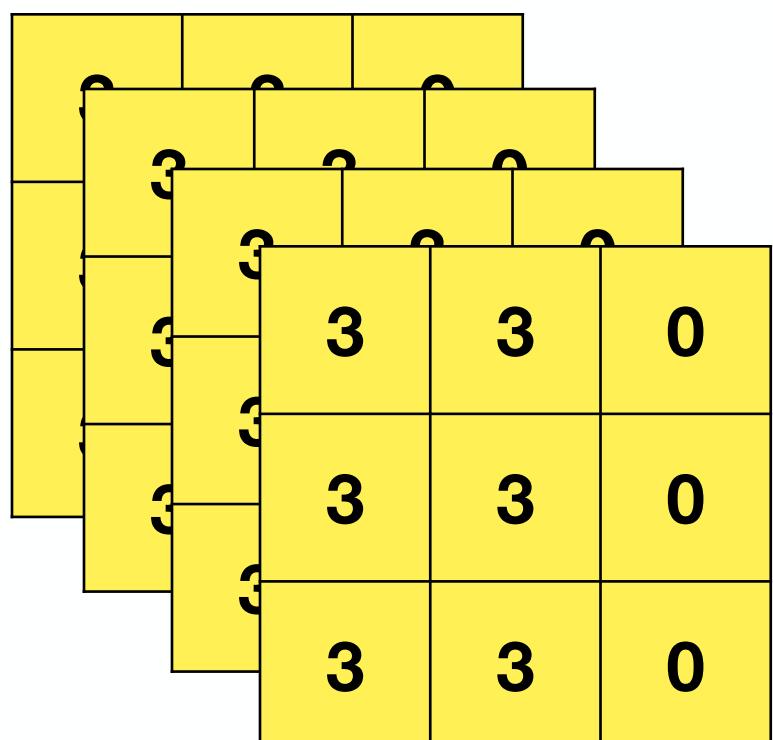
Input Image  
 $5 \times 5 \times 3$

\*



4 Filters or Kernels  
 $3 \times 3 \times 3 \times 4$

=



Output or Feature Map  
 $3 \times 3 \times 4$



# MODERN COMPUTER VISION

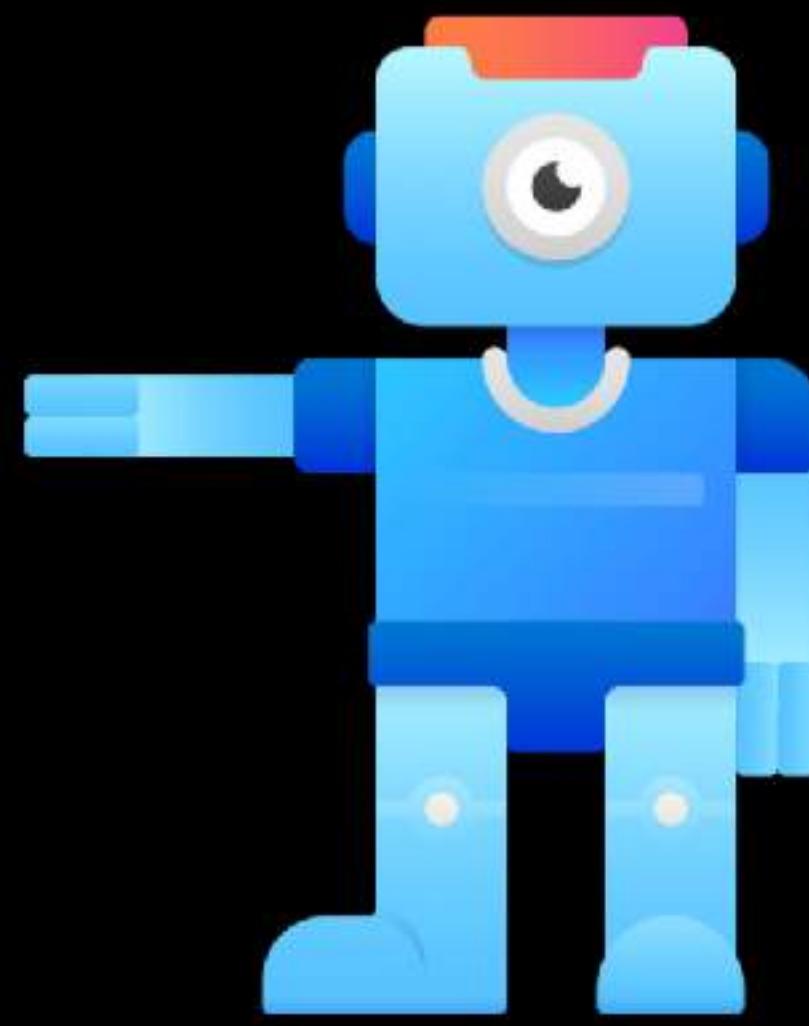
BY RAJEEV RATAN

# Next...

**Padding**

# Padding

Manipulating the input size



MODERN  
COMPUTER  
VISION

BY RAJEEV RATAN

# Padding

**Notice How Conv Filters Produce an Output Smaller than the Input**

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	3
2	1	1

$5 \times 5$

$n \times n$

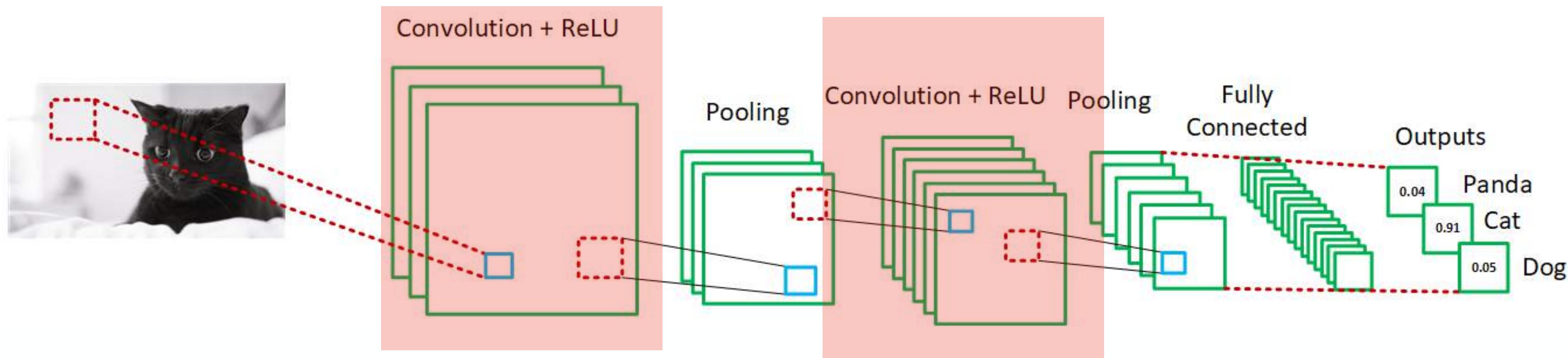
$3 \times 3$

$f \times f$

$3 \times 3$

$m \times m$

# CNNs can have several sequences of Convolution layers



# Consecutive Conv layers would keep shrinking the output

Can we preserve our image size?

We've added a 1 pixel pad of zeros (zero padding) around our input

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

\*

0	1	0
1	0	-1
0	1	0

=

$7 \times 7$

$n \times n$

$3 \times 3$

$f \times f$

# Padding

Let's Perform our Convolution with the Padding

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

\*

0	1	0
1	0	-1
0	1	0

=

$7 \times 7$

$n \times n$

$3 \times 3$

$f \times f$

# Padding

Let's Perform our Convolution with the Padding

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

\*

0	1	0
1	0	-1
0	1	0

=

$7 \times 7$

$n \times n$

$3 \times 3$

$f \times f$

# Padding

Let's Perform our Convolution with the Padding

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

\*

0	1	0
1	0	-1
0	1	0

=

$7 \times 7$

$n \times n$

$3 \times 3$

$f \times f$

# Padding

Let's Perform our Convolution with the Padding

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

\*

0	1	0
1	0	-1
0	1	0

=

$7 \times 7$

$n \times n$

$3 \times 3$

$f \times f$

# Padding

Let's Perform our Convolution with the Padding

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

\*

0	1	0
1	0	-1
0	1	0

=

$7 \times 7$

$n \times n$

$3 \times 3$

$f \times f$

# Padding

Let's Perform our Convolution with the Padding

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

\*

0	1	0
1	0	-1
0	1	0

=

$7 \times 7$

$n \times n$

$3 \times 3$

$f \times f$

# Padding

## Let's Perform our Convolution with the Padding

$$\text{Feature Map Size} = n - f + 1 = m$$

$$\text{Feature Map Size} = 7 - 3 + 1 = 5$$

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1	2	2
-1	1	3	2	1
2	1	1	1	2
1	1	1	0	2
2	0	2	3	1

$7 \times 7$

$n \times n$

$3 \times 3$

$f \times f$

$5 \times 5$

$m \times m$

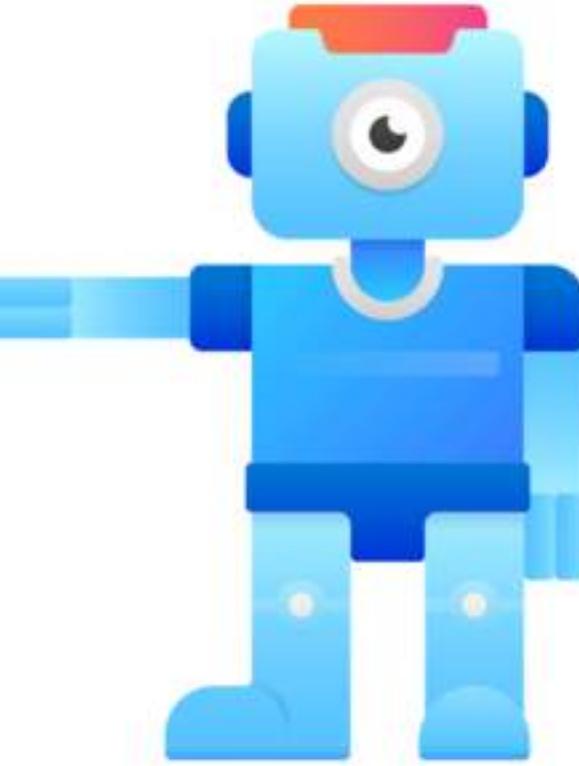
# Why Use Padding?

- For very deep networks we don't want to keep reducing the size
- Pixels at the edges contribute less to the output Feature Maps, thus we're throwing away information from them

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Without padding, our **top left** pixel is only touched by the Conv Filter once

Whereas, our **centre** pixel is passed over numerous times

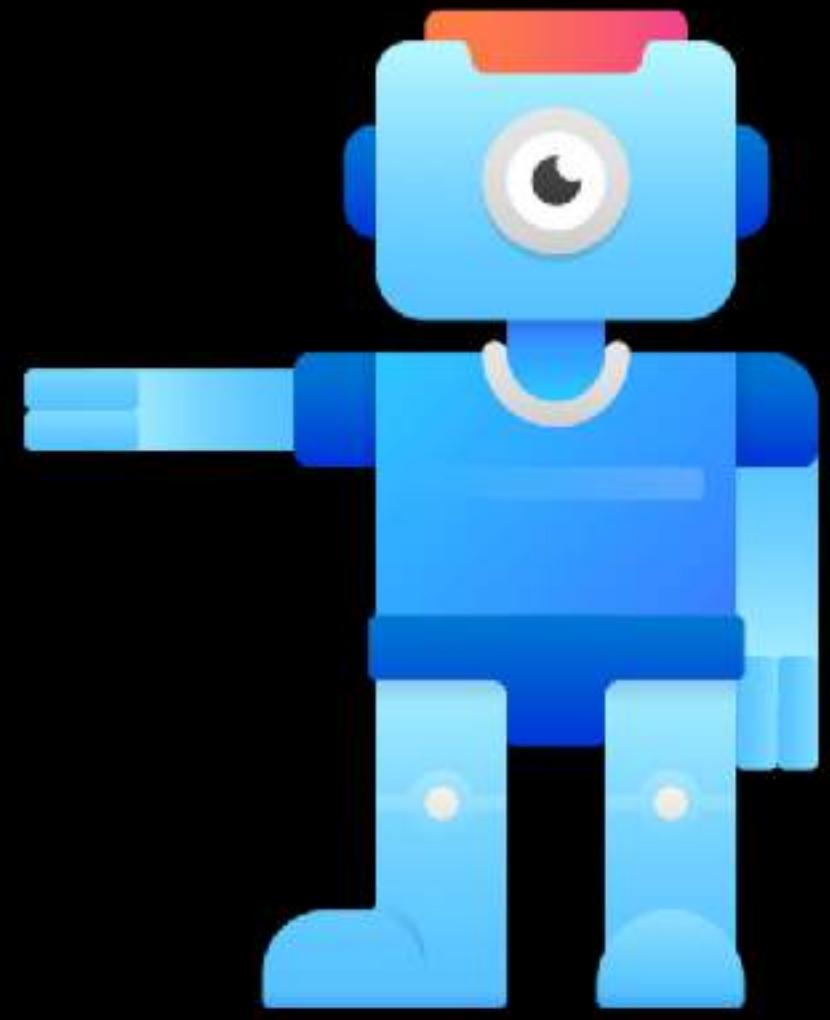


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Stride**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Stride

We look at a parameter that defines how we move our Conv Filter

# Stride

## Our Step Size

- Stride defines how many steps we take when sliding our Convolution Window across the input image

# What a Stride of 1 Looks Like

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

\*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2		

Output or Feature Map

# What a Stride of 1 Looks Like

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

\*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	

Output or Feature Map

# What a Stride of 1 Looks Like

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

\*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1

Output or Feature Map

# What a Stride of 1 Looks Like

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

\*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1		

Output or Feature Map

# What a Stride of 1 Looks Like

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

\*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	

Output or Feature Map

# What a Stride of 1 Looks Like

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

\*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	3

Output or Feature Map

# What a Stride of 1 Looks Like

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	3
2		

Input Image

Filter or Kernel

Output or Feature Map

# What a Stride of 1 Looks Like

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	3
2	1	

Input Image

Filter or Kernel

Output or Feature Map

# What a Stride of 1 Looks Like

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	3
2	1	1

Input Image

Filter or Kernel

Output or Feature Map

**What about a  
Stride of 2?**

# What a Stride of 2 Looks Like

We start off in the same position

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

\*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	

Output or Feature Map

# What a Stride of 2 Looks Like

Now we jump two spots to the left

→

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	-1

Input Image

Filter or Kernel

Output or Feature Map

# What a Stride of 2 Looks Like

Now we go down, but by also 2 spots

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

\*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	-1
2	

Output or Feature Map

# What a Stride of 2 Looks Like

Now we jump two spots to the left

→

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

\*

0	1	0
1	0	-1
0	1	0

=

2	-1
2	1

Input Image

Filter or Kernel

Output or Feature Map

# Stride Observations

- A larger Stride produced a **smaller** Feature Map output
- Larger Stride has **less overlap**
- We can use stride to **control the size of the Feature Map output**

# Calculating Output Size

## Using Stride and Padding

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image  
5 x 5

0	1	0
1	0	-1
0	1	0

Filter or Kernel  
3 x 3

**Stride = 2**  
**Padding = 0**

2	-1
2	1

$$(n \times n) * (f \times f) = \left( \frac{n + 2p - f}{s} + 1 \right) \times \left( \frac{n + 2p - f}{s} + 1 \right) = \left( \frac{5 + (2 \times 0) - 3}{2} + 1 \right) \times \left( \frac{5 + (2 \times 0) - 3}{2} + 1 \right) = 2 \times 2$$

# Calculating Output Size

## Using Stride and Padding

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image  
5 x 5

0	1	0
1	0	-1
0	1	0

Filter or Kernel  
3 x 3

**Stride = 1**  
**Padding = 0**

2	-1	1
2	1	0
1	-1	1

$$(n \times n) * (f \times f) = \left( \frac{n + 2p - f}{s} + 1 \right) \times \left( \frac{n + 2p - f}{s} + 1 \right) = \left( \frac{5 + (2 \times 0) - 3}{1} + 1 \right) \times \left( \frac{5 + (2 \times 0) - 3}{1} + 1 \right) = 3 \times 3$$

# Calculating Output Size

## When using Stride & Padding

$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline
 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hline
 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline
 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline
 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ \hline
 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline
 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
 \end{array}$$

Input Image  
 $7 \times 7$

$$\begin{array}{|c|c|c|} \hline
 0 & 1 & 0 \\ \hline
 1 & 0 & -1 \\ \hline
 0 & 1 & 0 \\ \hline
 \end{array}$$

Filter or Kernel  
 $3 \times 3$

**Stride = 1**  
**Padding = 1**

$$\begin{array}{|c|c|c|c|c|} \hline
 2 & -1 & 1 & 1 & 0 \\ \hline
 2 & 1 & 0 & 1 & 2 \\ \hline
 1 & -1 & 1 & 0 & 1 \\ \hline
 0 & 1 & 2 & 1 & 1 \\ \hline
 2 & 0 & 1 & 0 & 2 \\ \hline
 \end{array}$$

$$(n \times n) * (f \times f) = \left( \frac{n + 2p - f}{s} + 1 \right) \times \left( \frac{n + 2p - f}{s} + 1 \right) = \left( \frac{5 + (2 \times 1) - 3}{1} + 1 \right) \times \left( \frac{5 + (2 \times 1) - 3}{1} + 1 \right) = 5 \times 5$$

Note if we get non integer value for our output size, we found it down to the nearest integer

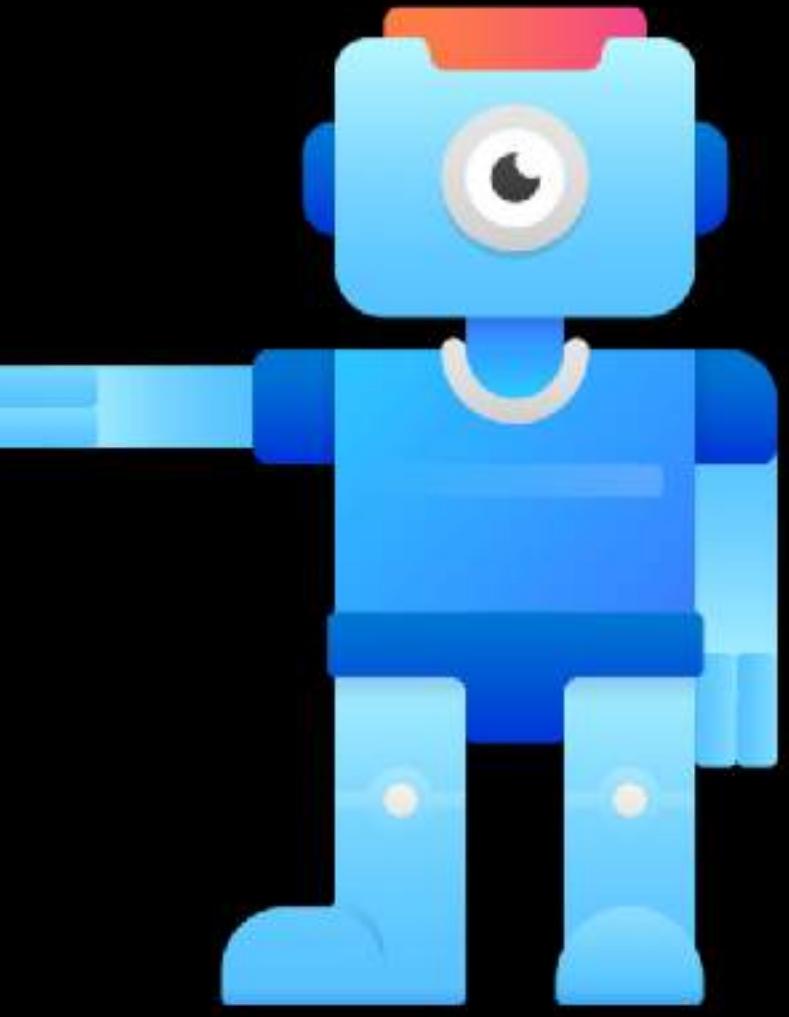


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Activation Layer ReLU**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

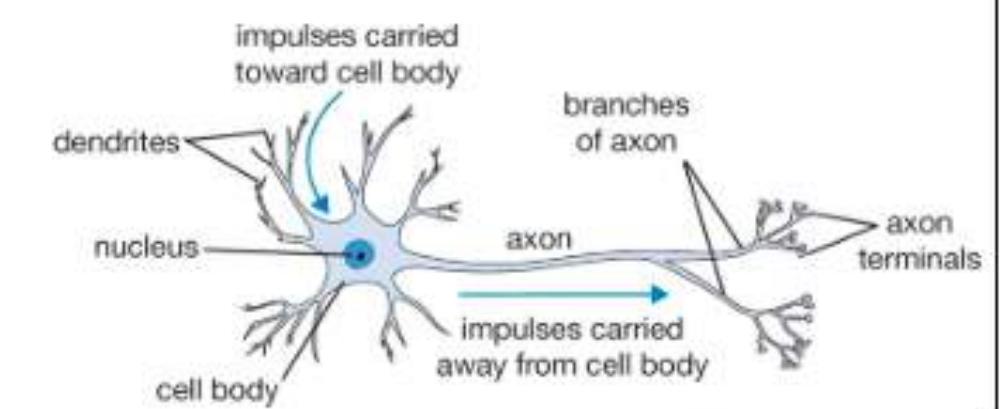
## Activation Layer ReLU

**What are Activation Functions and their importance**

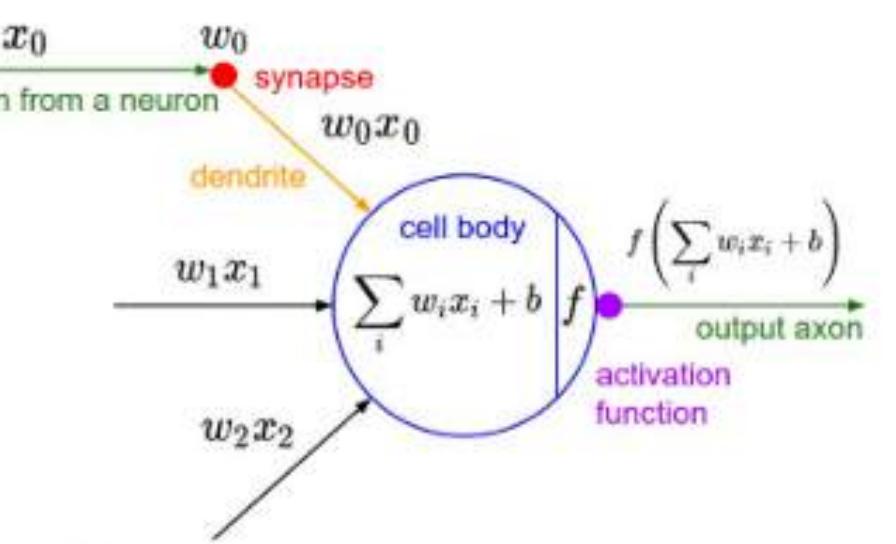
# Purpose of Activation Functions

To enable the learning of **complex patterns** in our data

- Biological neurons fire (activate) on certain inputs, these are then fed into other neurons
- Introduces **non-linearity** to our network
- This allows a non-linear decision boundary via non-linear combinations of the weight and inputs



A cartoon drawing of a biological neuron (left) and its mathematical model (right).



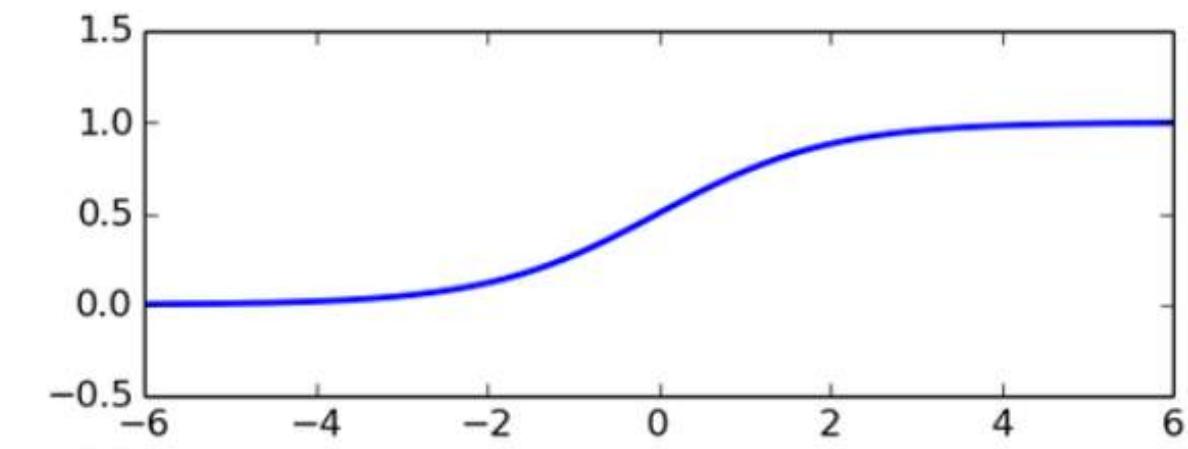
Stanford's CS231 Course

# Types of Activation Functions

There are several activation functions we can use in our CNN. However, Rectified Linear Units (**ReLU**) have become the activation function of choice for CNNs.

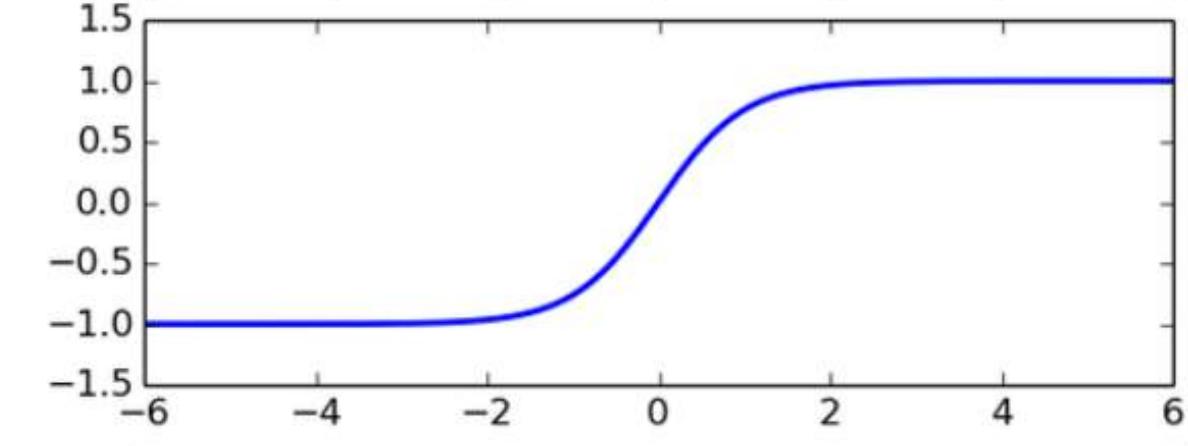
ReLU is advantageous in CNN Training:

- Simple Computation (fast to train)
- Does not saturate



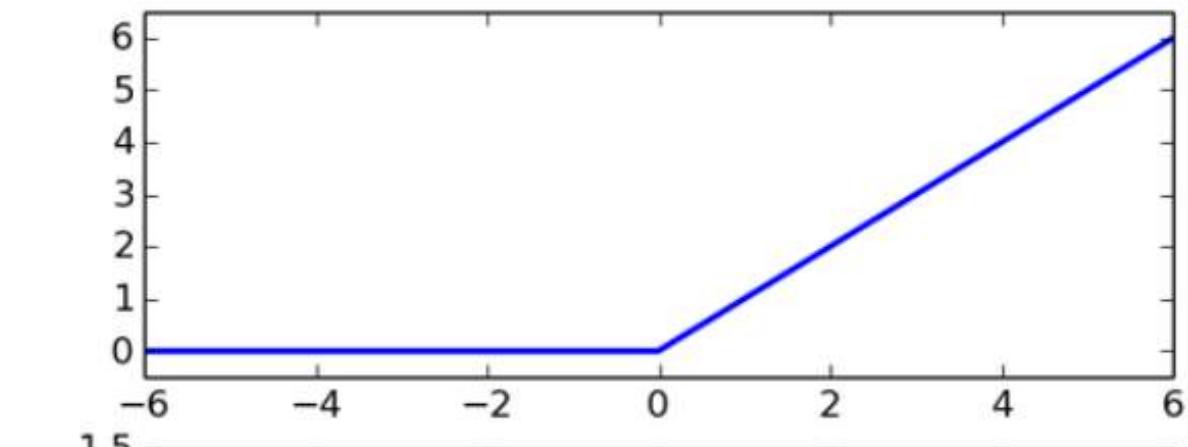
Sigmoid

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



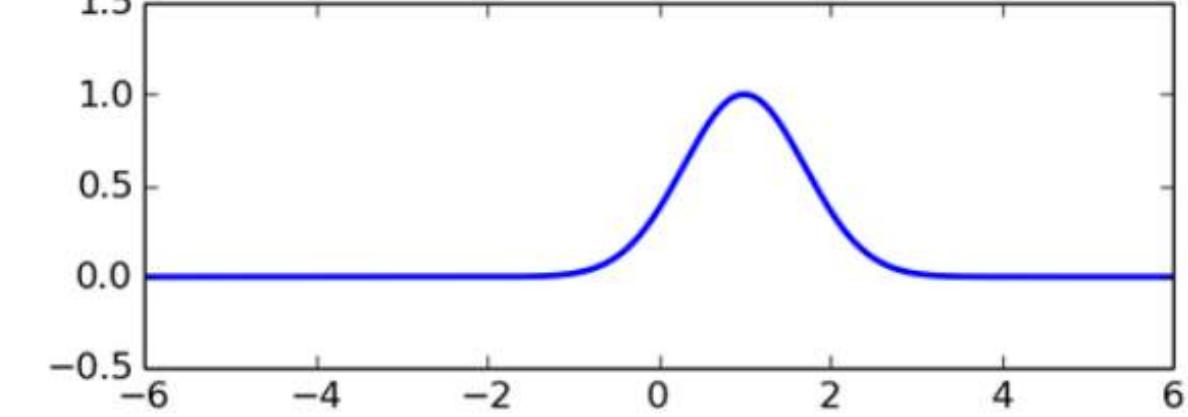
Hyperbolic Tangent

$$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



Rectified Linear

$$\phi(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$



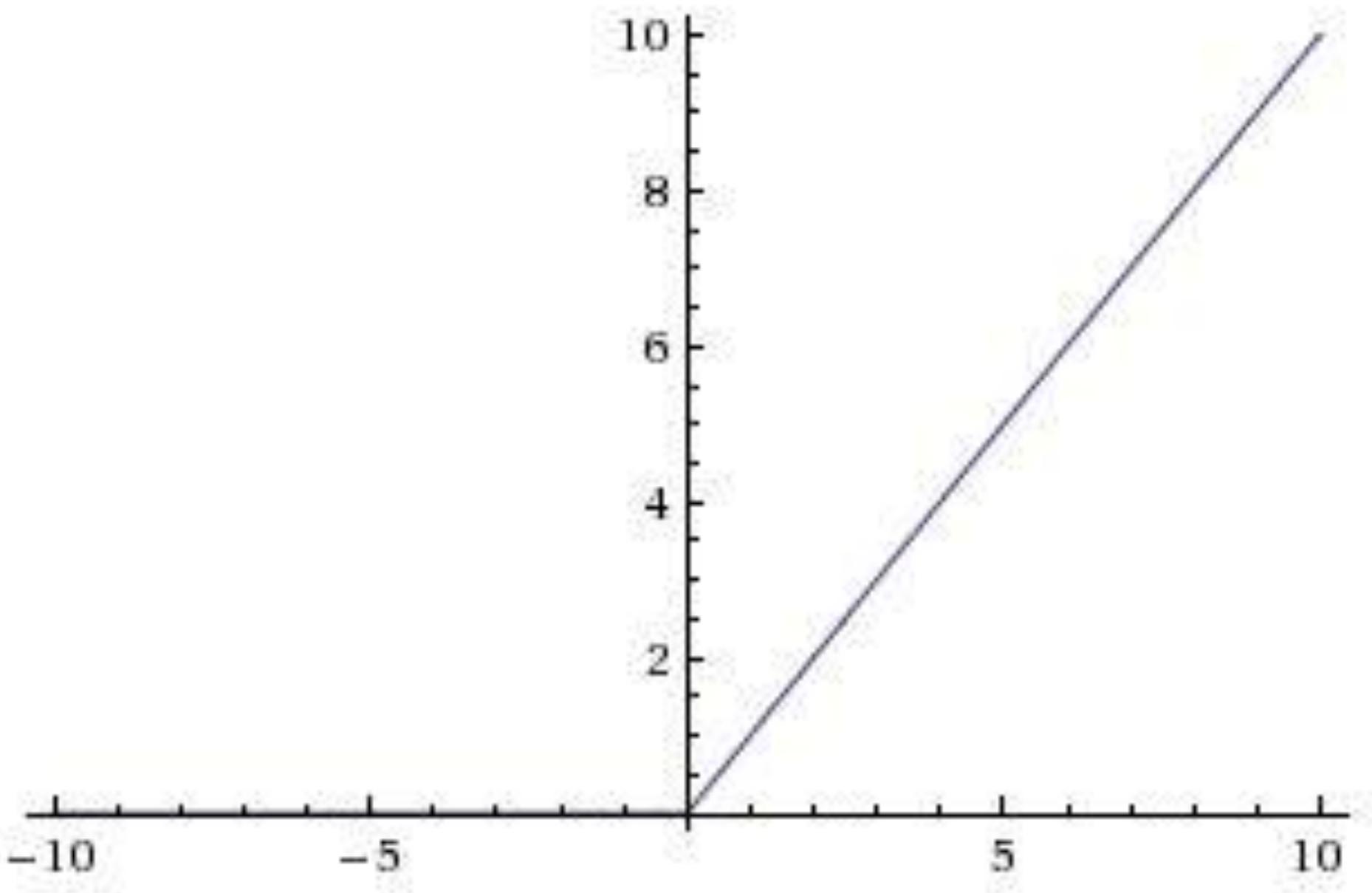
Radial Basis Function

$$\phi(z, c) = e^{-(\epsilon \|z - c\|)^2}$$

# The ReLU Operation

- Change all negative values to 0
- Leave all positive Values alone

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



# Applying the ReLU Activation

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

0	1	0
1	0	-1
0	1	0

Filter or Kernel

2	1	-1
-1	1	3
2	1	-5

Output or Feature Map

ReLU

2	1	0
0	1	3
2	1	0

# Applying the ReLU Activation

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

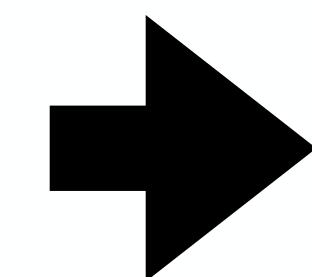
\*

0	1	0
1	0	-1
0	1	0

=

2	1	-1
-1	1	3
2	1	-5

ReLU



2	1	0
0	1	3
2	1	0

Input Image

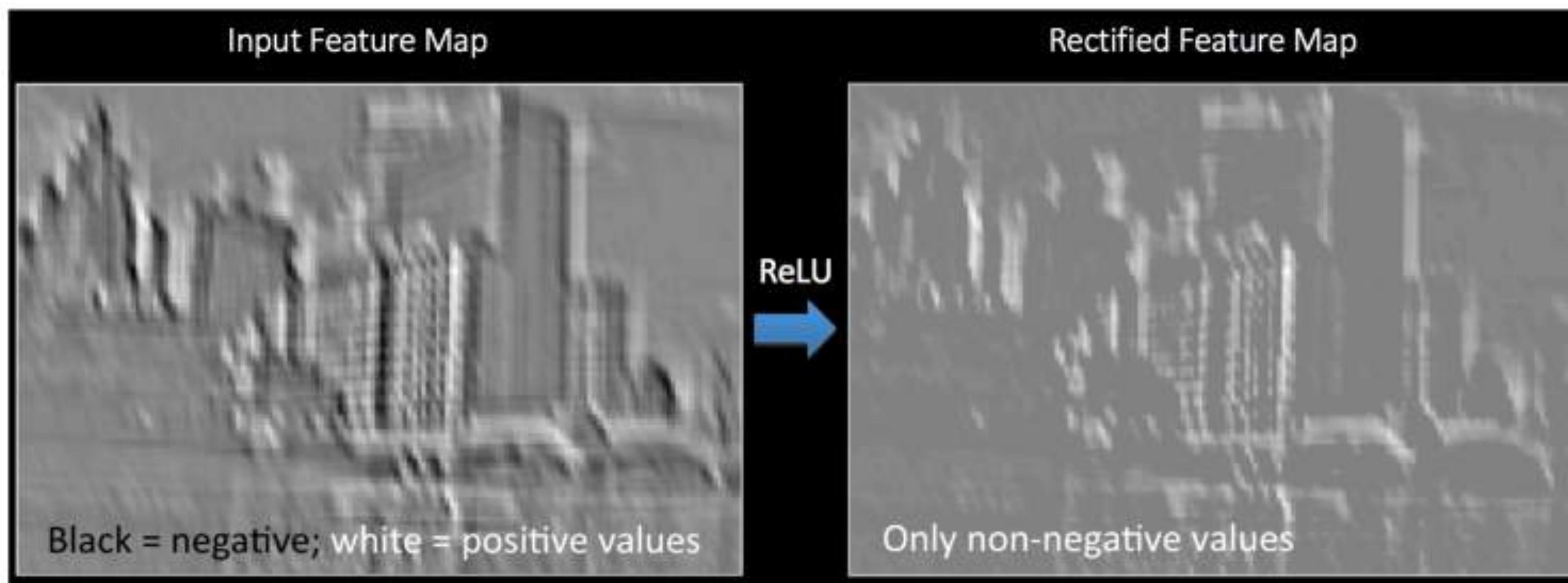
Filter or Kernel

Output or Feature Map

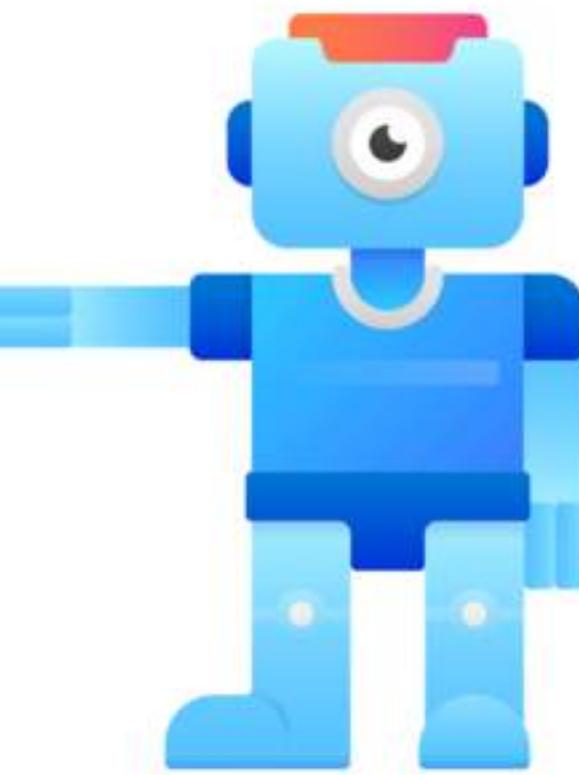
Rectified Feature Map

One Layer

# Example of a Rectified Linear Map



Source - [http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus\\_1.pdf](http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf)



# MODERN COMPUTER VISION

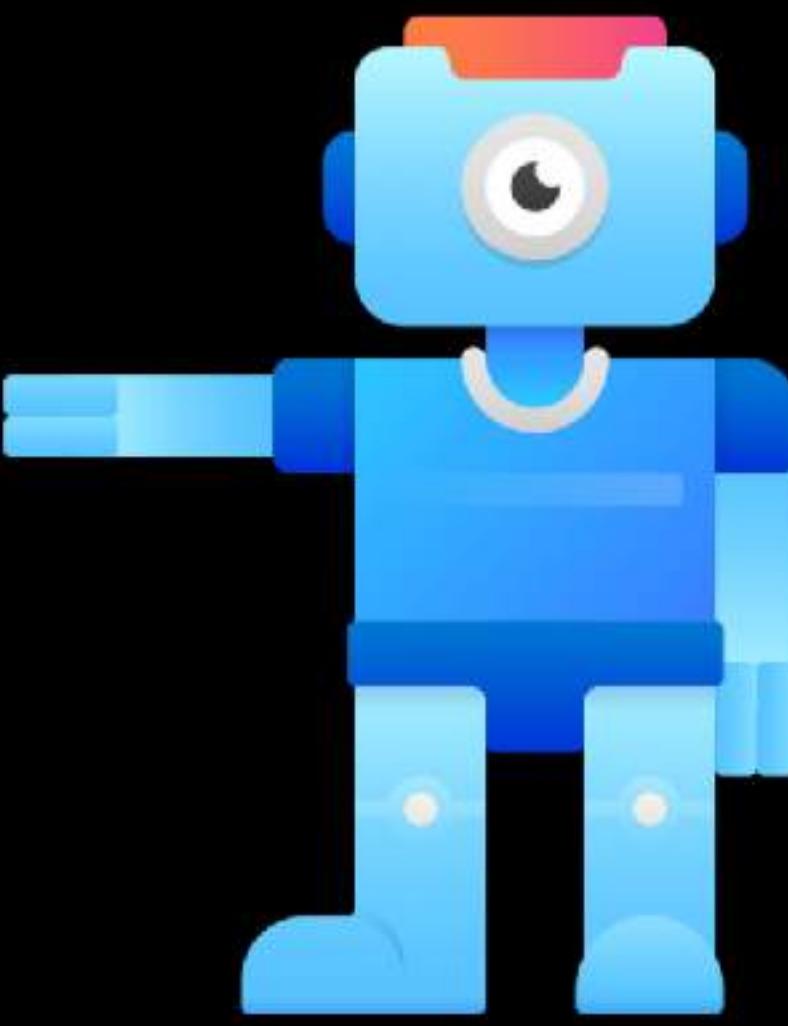
BY RAJEEV RATAN

# Next...

**Pooling**

# Pooling

We explore the Max Pool Layer and it's purpose



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Pooling

- Pooling is the process whereby we reduce the size or dimensionality of the Feature Map
- This allows us to reduce the number of Parameters in our Network whilst retaining important features
- Also called Subsampling or Downsampling

# Example of Max Pooling

4	123	1	34
56	99	222	253
45	122	165	12
21	187	133	124

**MaxPool Operation**



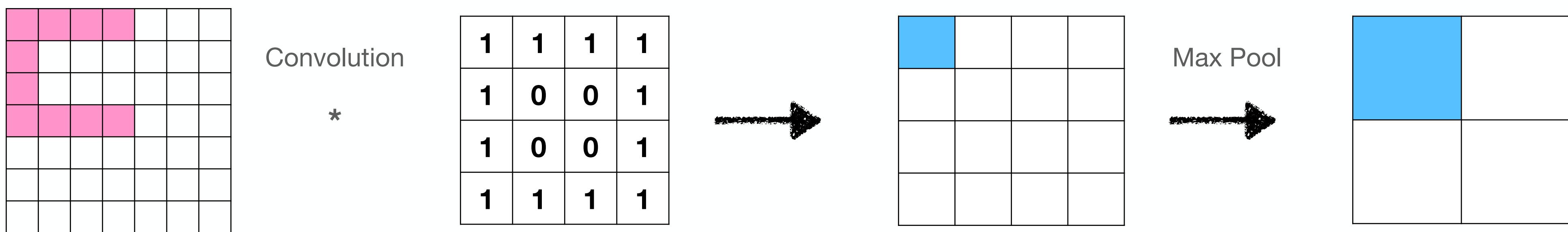
**Stride = 2**  
**Kernel = 2x2**

123	167
187	165

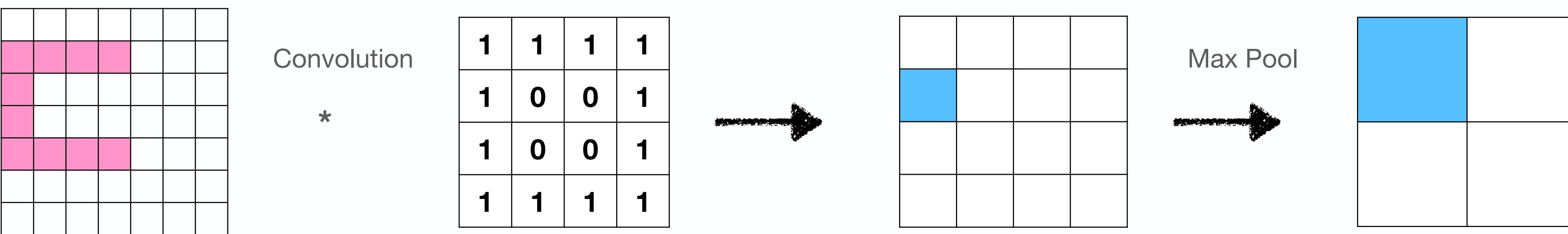
# More on Pooling

- Typically we use **2x2** kernels and a Stride of **2** with no padding.
- With the above setting, pooling reduces the **dimensionality by a factor of 2** (width and height).
- Pooling makes our model more **invariant** to minor **transformations** and **distortions** in our image.
- We can also use Average Pooling or Sum Pooling

# How Max Pooling Achieves Translation Invariance



Shifting Our C down one pixel



# Why Pooling Works

Pooling reduces our feature map size by half in most cases, is that ok?

- Neighbouring pixels are **strongly correlated**, especially in lowest layers
- Remember further apart two pixels are from each other, the less correlated
- Therefore, we can **reduce the size** of the output by subsampling (**pooling**) the filter response **without losing information**
- A big stride in the pooling layer leads to high information loss
- In practice, a stride of 2 and a kernel size 2x2 for the pooling layer was found to be effective in practice

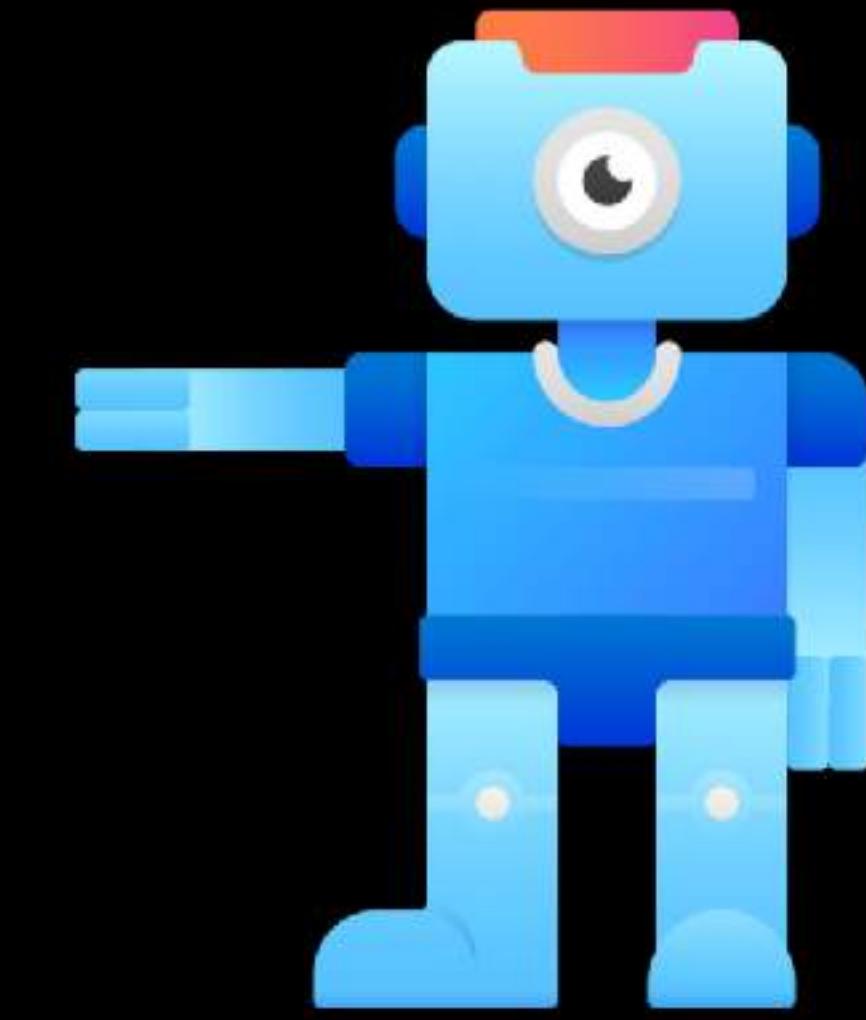


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Fully Connected Layer**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

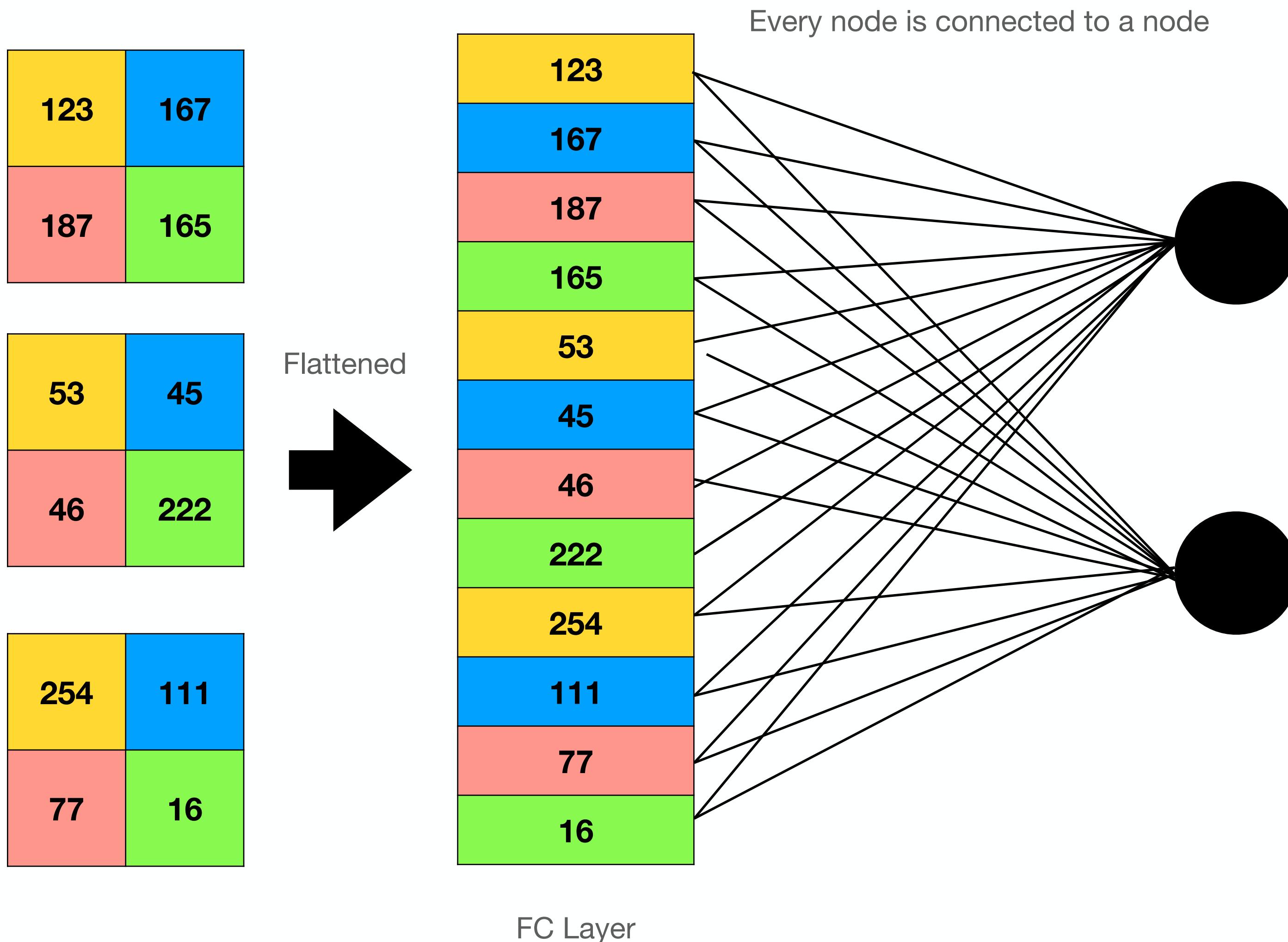
# Fully Connected Layer

We explore the Fully Connected Layer and it's purpose

# What does Fully Connected Mean?

- It means all nodes in one layer are connected to the outputs of the next layer.
- It takes the 3D Volume output of the previous layer and **flattens** it into a single vector that is used for input in the next layer.
- It's sometimes called the Dense Layer

# FC Layer - The Max Pool Layer is Flattened



# Purpose of the Fully Connected Layer

- It compiles the data/outputs extracted from previous layers to form the final output
- It's an easy way of learning non-linear combinations of these features



# MODERN COMPUTER VISION

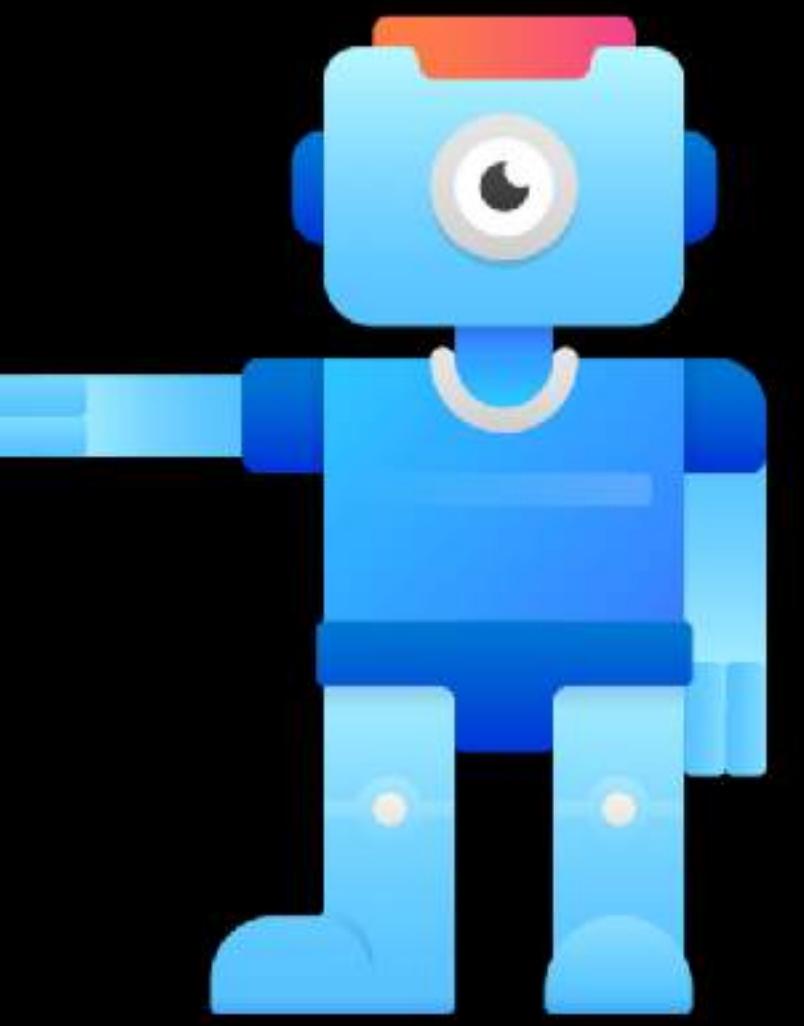
BY RAJEEV RATAN

# Next...

**Softmax Layer**

# Softmax Layer

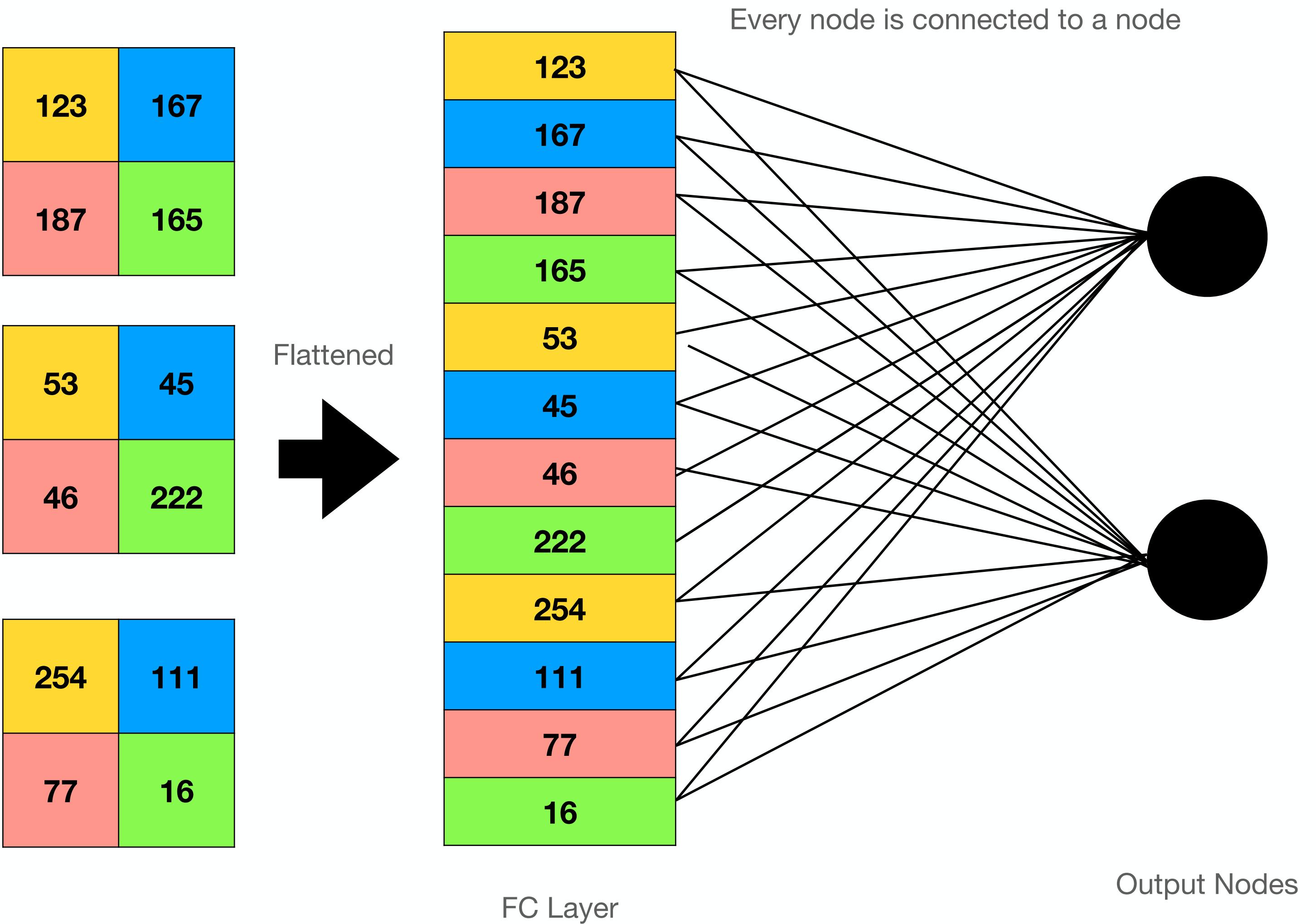
**Softmax Layer to Probabilities**



**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# The Fully Connected Layer



# Softmax Layer

- We now need to produce probability outcomes for each class in Network.
- Softmax converts the Logs into Probabilities
- It takes the exponents of every output and the normalises each output by the sum of the exponents.
- It guarantees a well behaved distribution i.e. all **sum to 1** and no values are zero.

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

# Softmax Layer

Logits Scores

2.0

1.0

0.1

Probabilities

0.7

0.2

0.1

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



# MODERN COMPUTER VISION

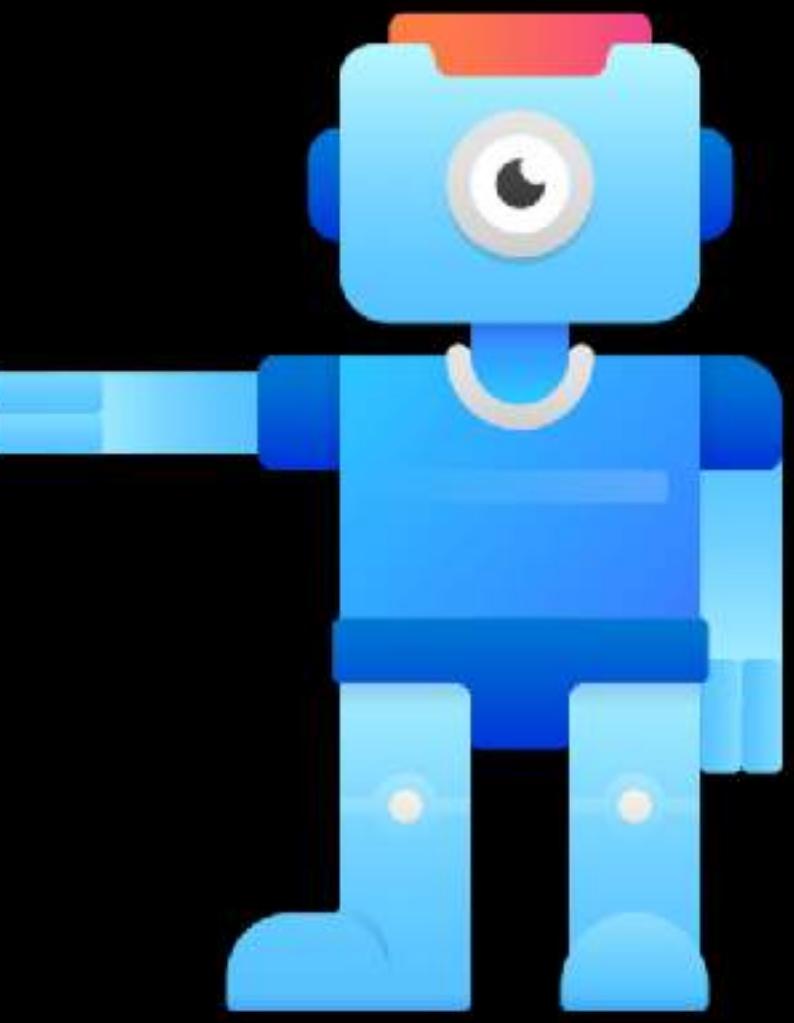
BY RAJEEV RATAN

# Next...

**Building a CNN**

# Building a CNN

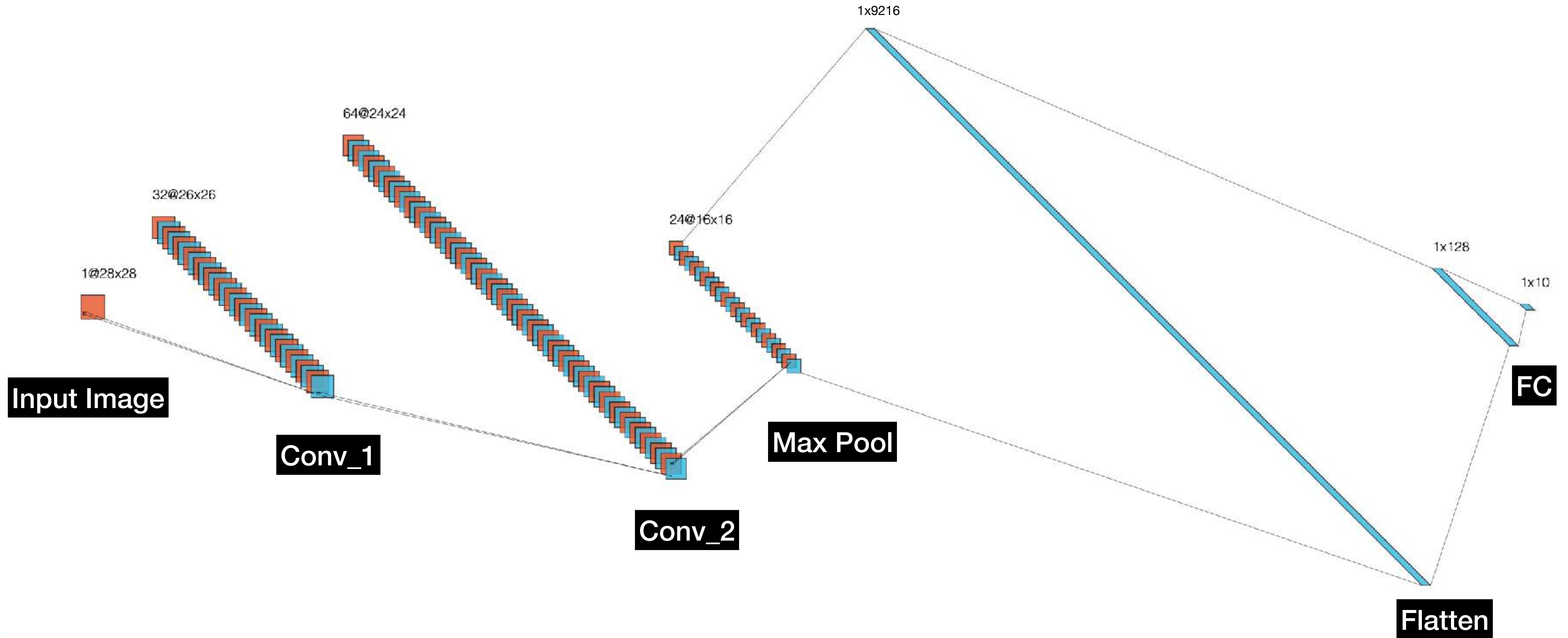
Let's put all the pieces together and build a CNN



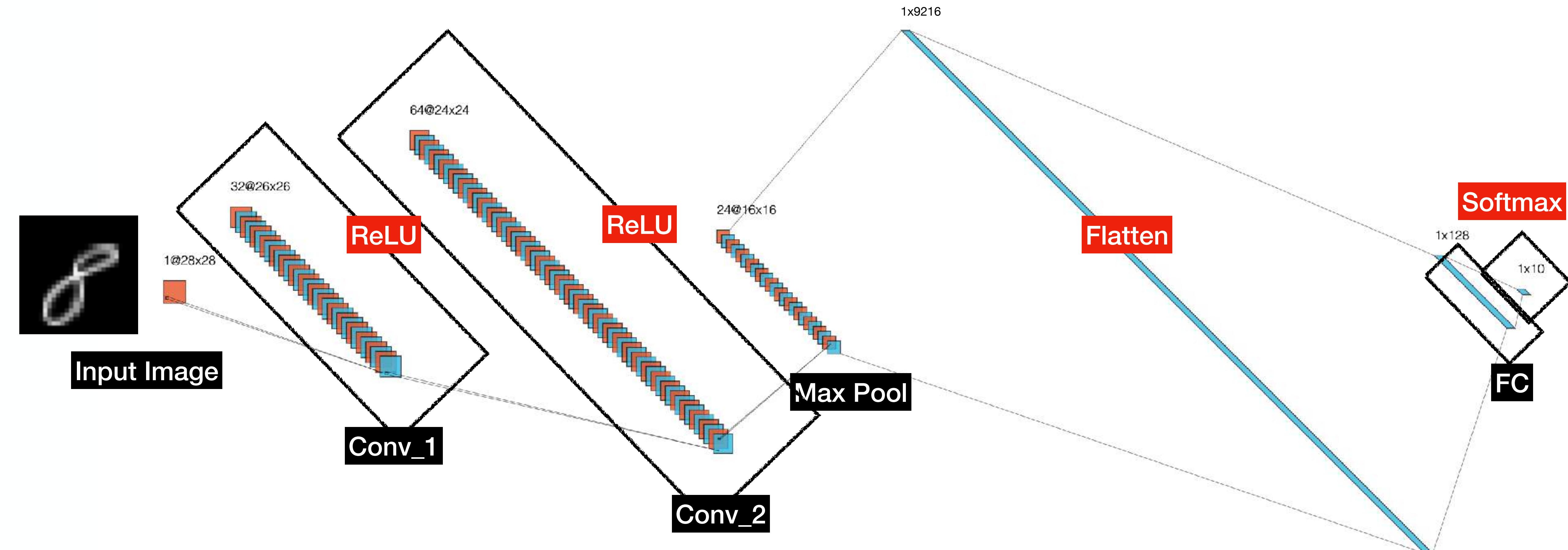
# MODERN COMPUTER VISION

BY RAJEEV RATAN

# A Simple CNN

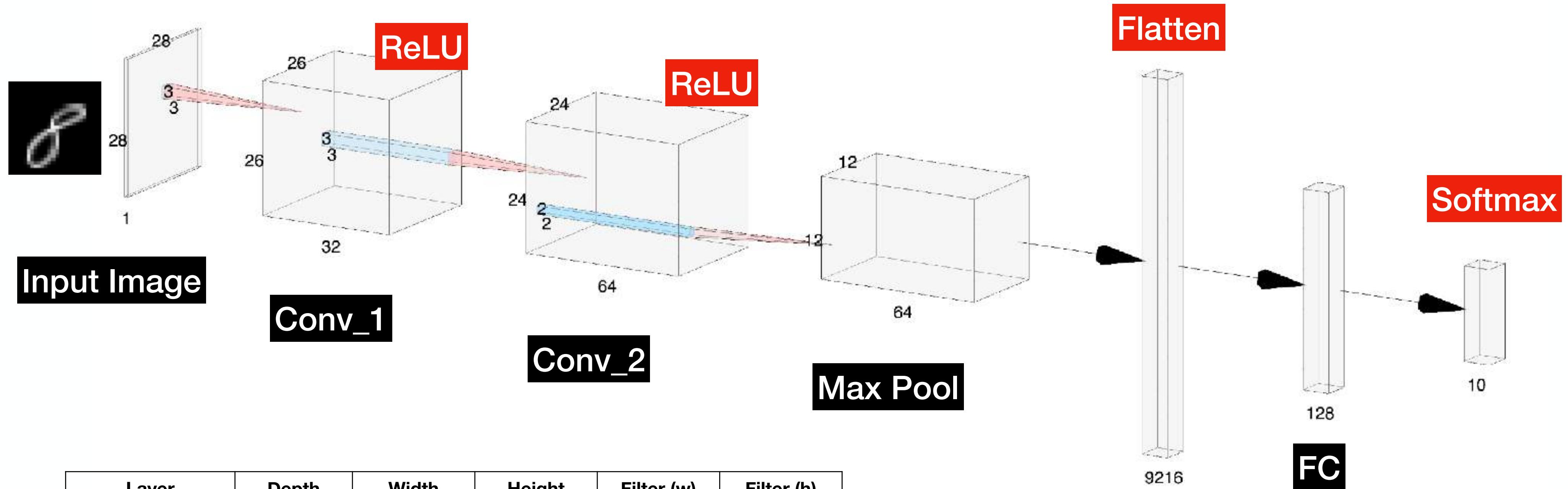


# A 4 Layer Deep CNN for MNIST



Example of a 4 Layer CNN we can use for the MNIST Dataset

# Another Representation

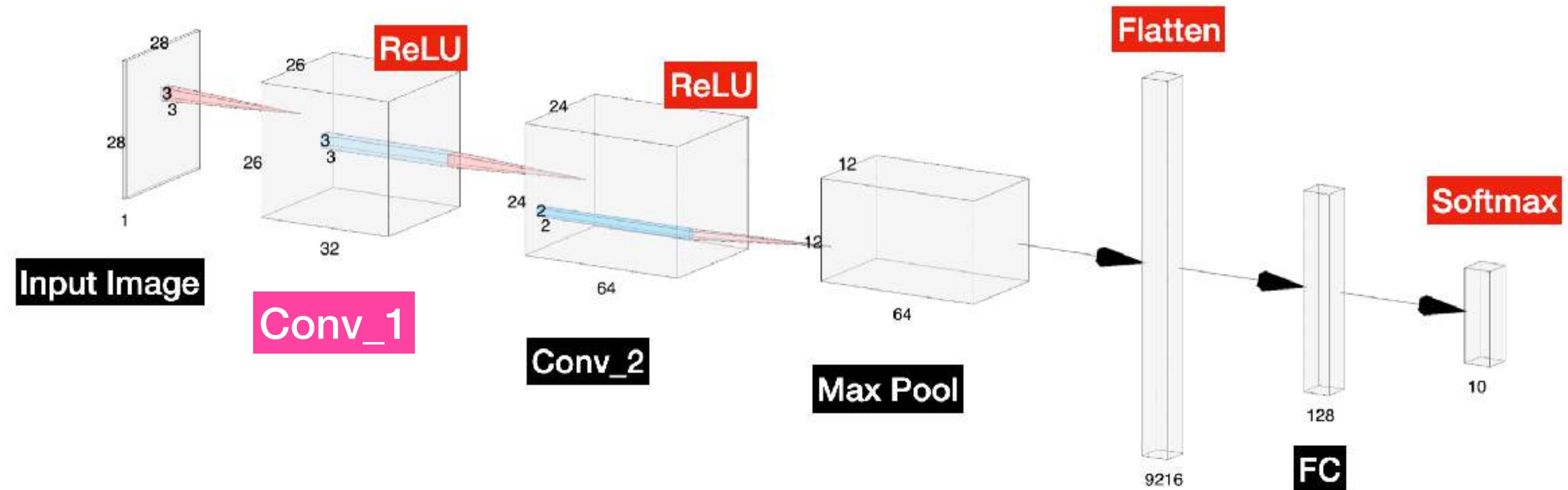


Layer	Depth	Width	Height	Filter (w)	Filter (h)
<b>Input</b>	1	28	28		
<b>Conv_1</b>	32	26	26	3	3
<b>Conv_2</b>	64	24	24	3	3
<b>Max Pool</b>	64	12	12	2	2
<b>Flatten</b>	9216	1	1		
<b>Fully Connected</b>	128	1	1		
<b>Output</b>	10	1	1		

## Notes:

- We choose 32 & 64 Filters or Kernels for Conv\_1 & Conv\_2
- We choose to
- The Feature Maps are shown as Conv\_1 and Conv\_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

# Calculating the Output Size of Conv\_1



## Notes:

- We choose 32 Filters or Kernels for Conv\_1
- The Feature Maps are shown as Conv\_1 & Conv\_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$(n \times n) * (f \times f) = \left( \frac{n + 2p - f}{s} + 1 \right) \times \left( \frac{n + 2p - f}{s} + 1 \right) = \left( \frac{28 + (2 \times 0) - 3}{1} + 1 \right) \times \left( \frac{28 + (2 \times 0) - 3}{1} + 1 \right) = 26 \times 26$$

Where

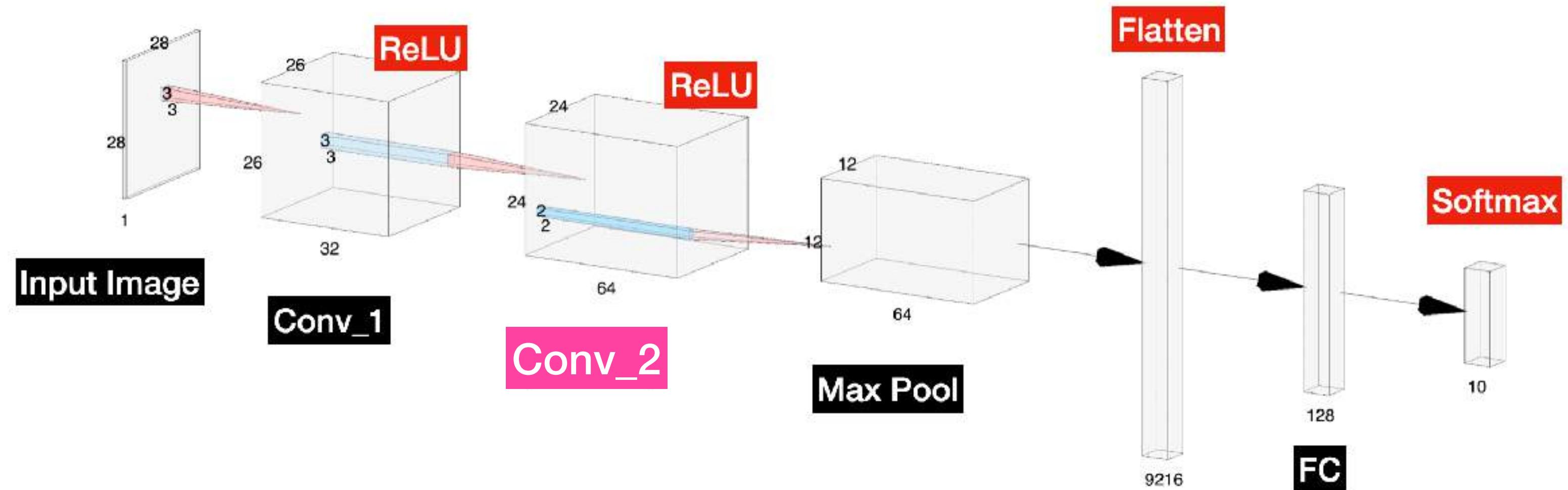
$$n = 28$$

$$f = 3$$

$$s = 1$$

$$p = 0$$

# Calculating the Output Size of Conv\_2



## Notes:

- We choose 64 Filters or Kernels for Conv\_2
- The Feature Maps are shown as Conv\_1 & Conv\_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$(n \times n) * (f \times f) = \left( \frac{n + 2p - f}{s} + 1 \right) \times \left( \frac{n + 2p - f}{s} + 1 \right) = \left( \frac{26 + (2 \times 0) - 3}{1} + 1 \right) \times \left( \frac{26 + (2 \times 0) - 3}{1} + 1 \right) = 24 \times 24$$

Where

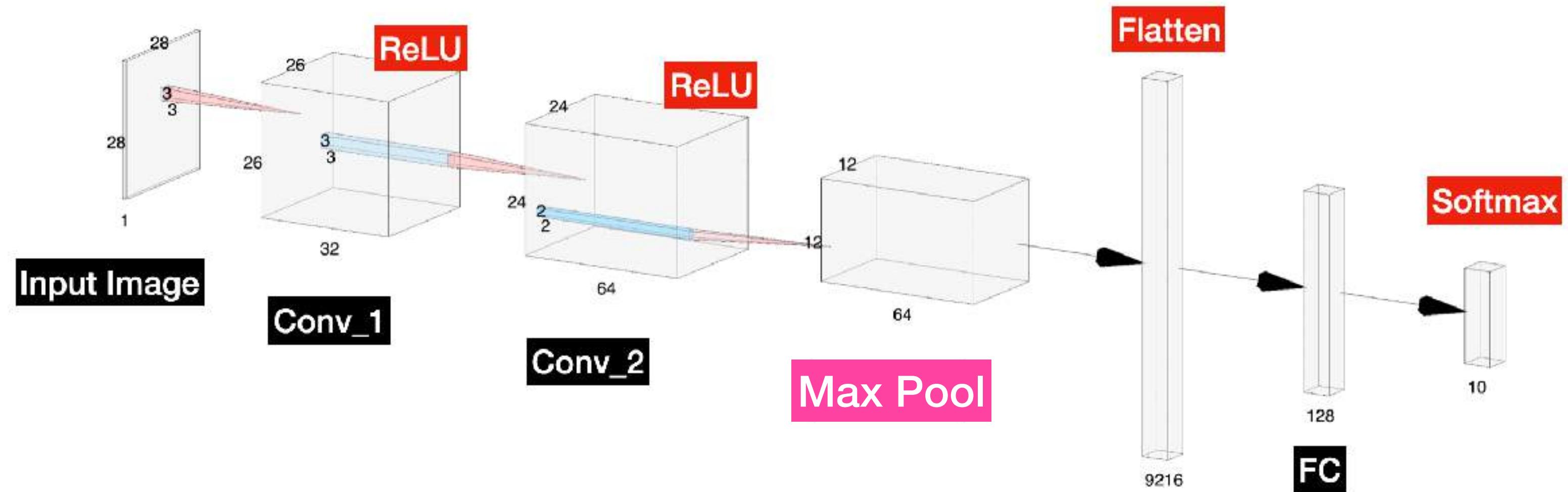
$$n = 26$$

$$f = 3$$

$$s = 1$$

$$p = 0$$

# Calculating the Output Size of the Max Pool Layer



## Notes:

- We choose 64 Filters or Kernels for Conv\_2
- The Feature Maps are shown as Conv\_1 & Conv\_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$(n \times n) * (f \times f) = \left( \frac{n + 2p - f}{s} + 1 \right) \times \left( \frac{n + 2p - f}{s} + 1 \right) = \left( \frac{24 + (2 \times 0) - 2}{2} + 1 \right) \times \left( \frac{24 + (2 \times 0) - 2}{2} + 1 \right) = 12 \times 12 \quad 12 \times 12$$

Where

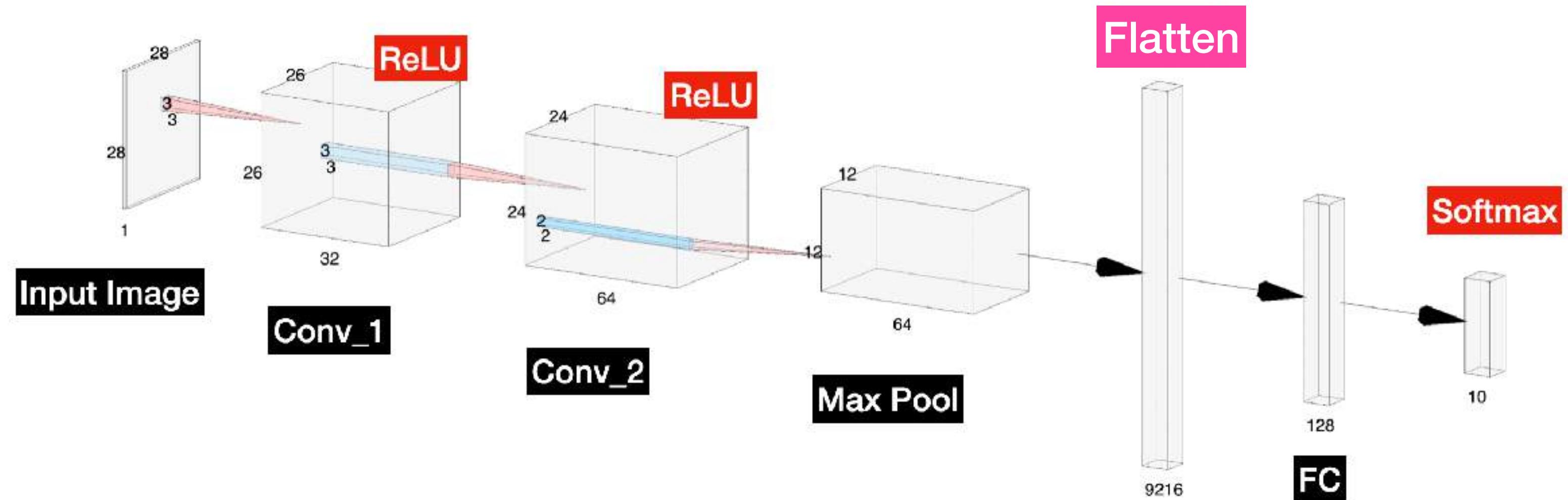
$$n = 24$$

$$f = 2$$

$$s = 2$$

$$p = 0$$

# Calculating the Output Size of Flattened Layer



## Notes:

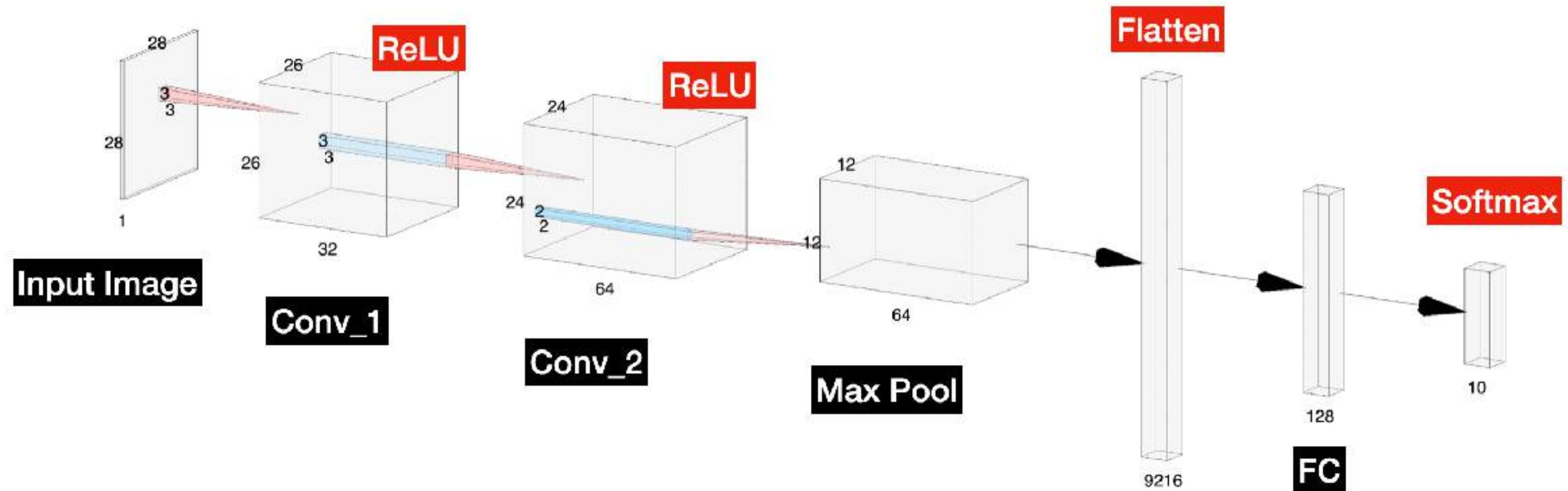
- We choose 64 Filters or Kernels for Conv\_2
- The Feature Maps are shown as Conv\_1 & Conv\_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$12 \times 12 \times 64 = 9216$$

Where

$$n = 12$$

# The Rest of the Our CNN



## Notes:

- We choose 64 Filters or Kernels for Conv\_2
- The Feature Maps are shown as Conv\_1 & Conv\_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2
- We choose 128 nodes in our FC Layer
- Our Dataset has 10 classes, hence why the final layer has 10 nodes

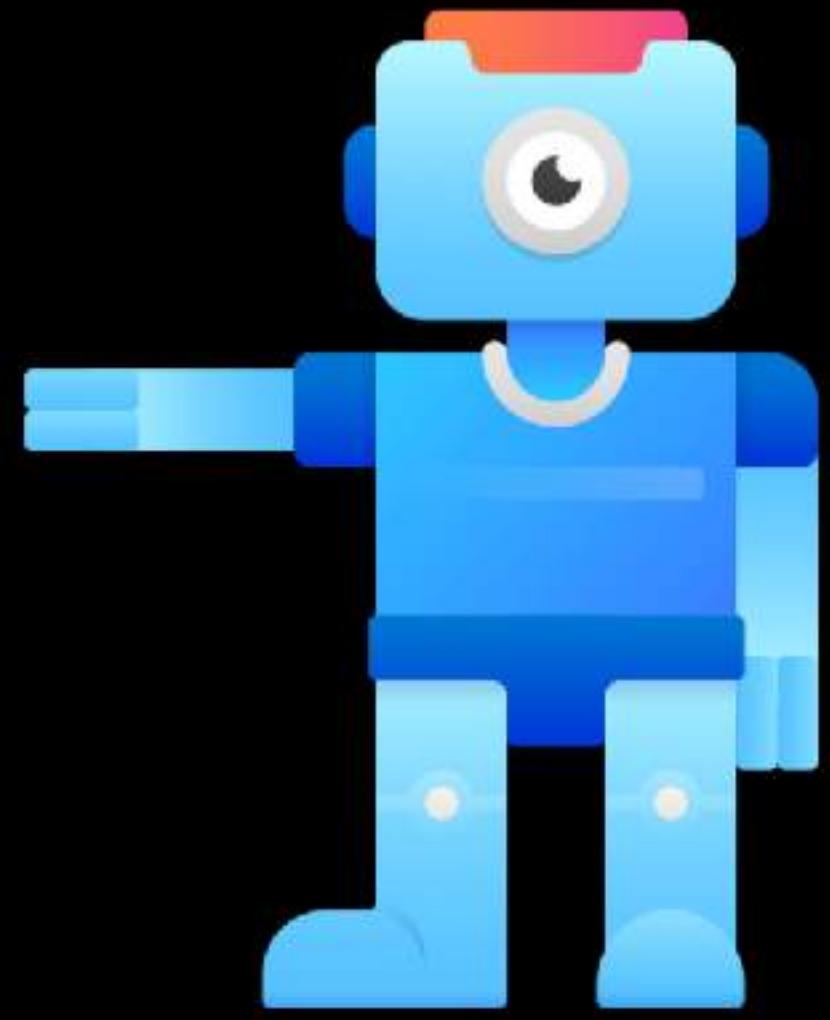


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Parameter Counts in CNNs**



MODERN  
COMPUTER  
VISION

BY RAJEEV RATAN

# Parameter Counts in CNNs

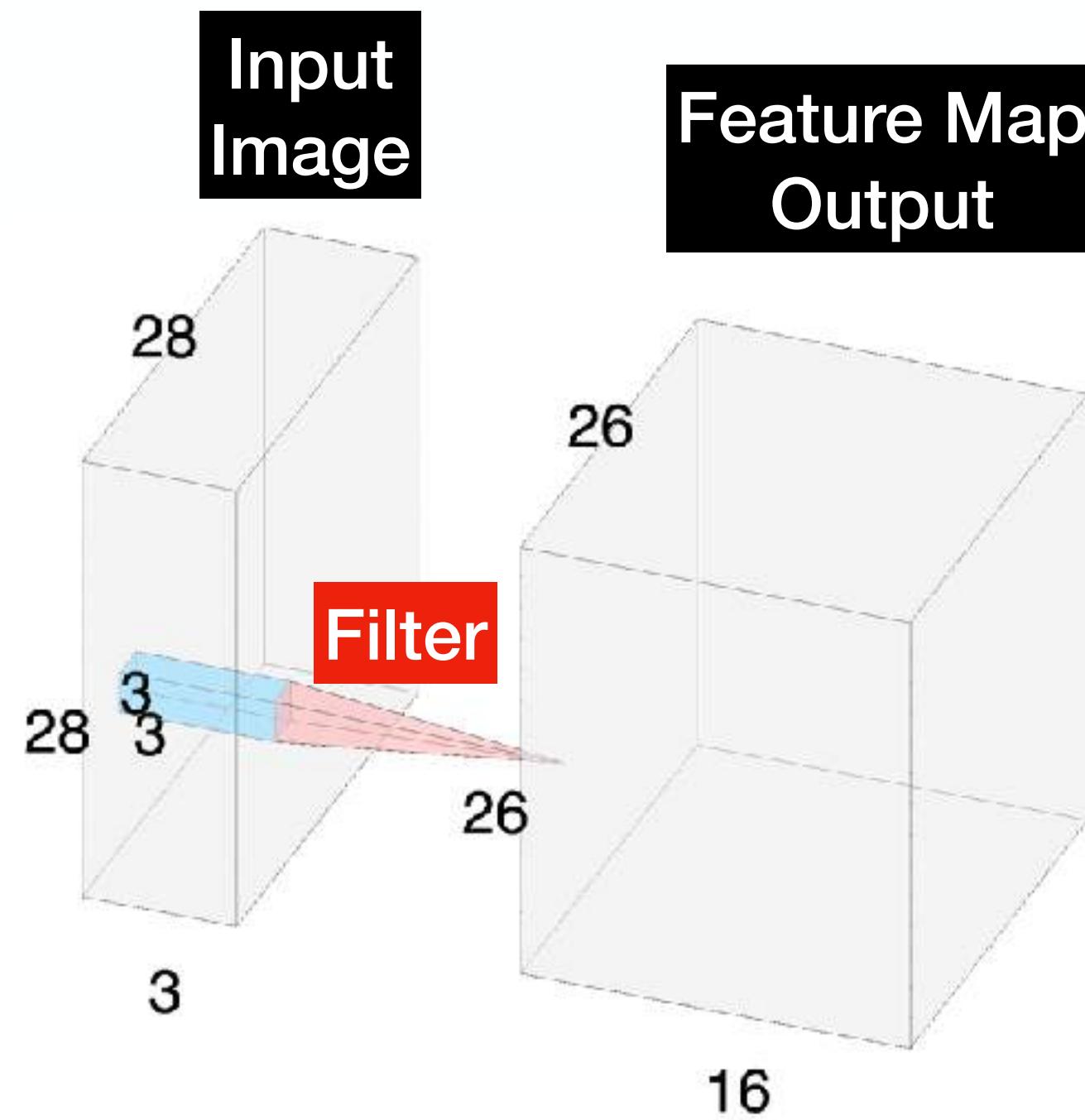
We explore what makes Convolution Neural Networks well suited for images

# Parameters

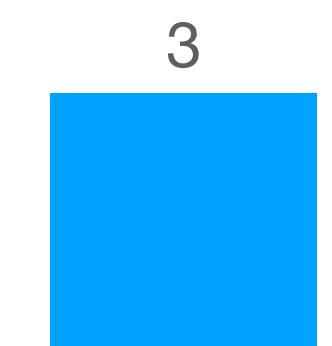
## What are Parameters or weights?

- They are the variables that need to be learnt when training a Model
- Often called learnable parameters or weights
- Our hidden layers like the Convolution or Fully Connected Layers have weights

# Calculating Learnable Parameters in a Conv Filter



- How many parameters are in 16 Convolution Filters with 3x3 kernels?
- Does the input image dimension matter? No, only it's depth
- All that matters is that we have 16 Conv Filters of 3x3 kernel size



**Parameters for One Filter**

$$(Height \times Width \times Depth) + bias$$

$$(3 \times 3 \times 1) + 1 = \boxed{10}$$

**Parameters for 16 Filters**

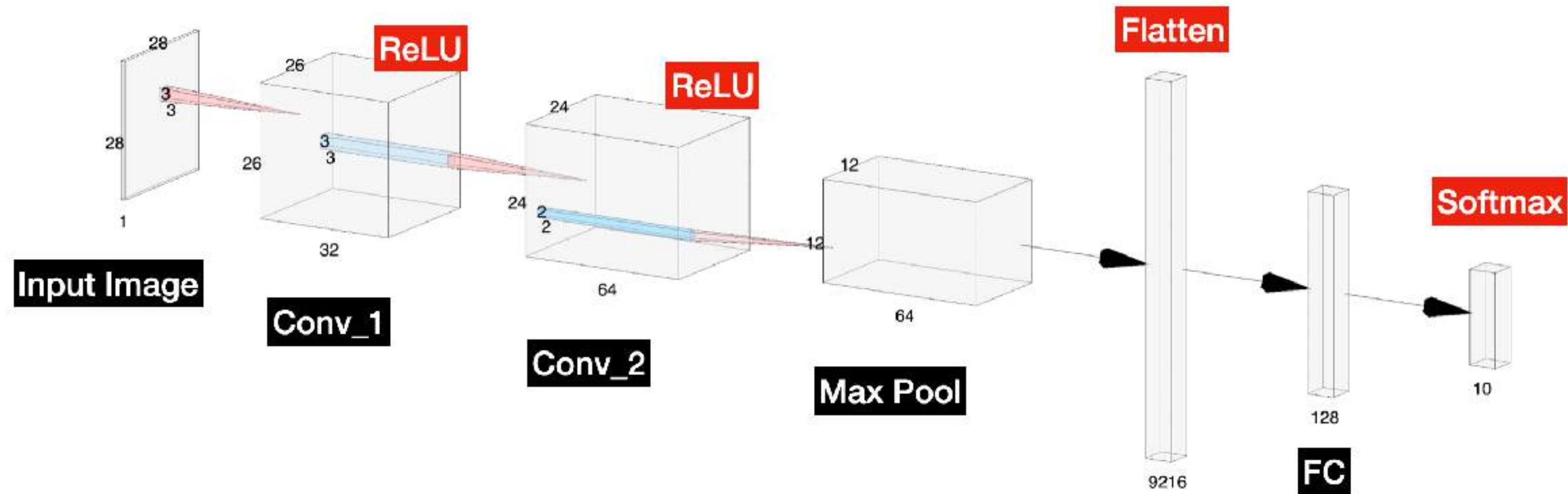
$$((Height \times Width \times Depth) + bias) \times 16$$

$$((3 \times 3 \times 1) + 1) \times 16 = \boxed{160}$$

# Biases

- Biases allow us to shift our activation function (left or right) by adding a constant value
- Biases are per ‘**neuron**’, so per **filter** in our case (shared in the case with colour RGB images as the input)
- These shared biases per ‘neuron’ allow the filter to detect the same feature
- They are a learnable parameter

# Let's Calculate the Number of Parameters in Our Simple CNN



Layer	Parameters
Conv_1 + ReLU	320
Conv_2 + ReLU	18494
Max Pool	0
Flatten	0
FC_1	1,179,776
FC_2 (Output)	1,290
Total	1,199,882

**Conv\_1**

$$((\text{Height} \times \text{Width} \times \text{Depth}) + \text{bias}) \times N_f$$

$$((3 \times 3 \times 1) + 1) \times 32 = 320$$

**Conv\_2**

$$((\text{Height} \times \text{Width} \times \text{Depth}) + \text{bias}) \times N_f$$

$$((3 \times 3 \times 32) + 1) \times 64 = 18,494$$

**No Trainable Parameters**

- Max Pool
- Flatten
- ReLU

**Fully Connected/Dense**

$$(\text{Length} + \text{bias}) \times N_{\text{nodes}}$$

$$(9216 + 1) \times 128 = 1,179,776$$

**Final Output (FC/Dense)**

$$(\text{Length} + \text{bias}) \times N_{\text{nodes}}$$

$$(128 + 1) \times 10 = 1,290$$

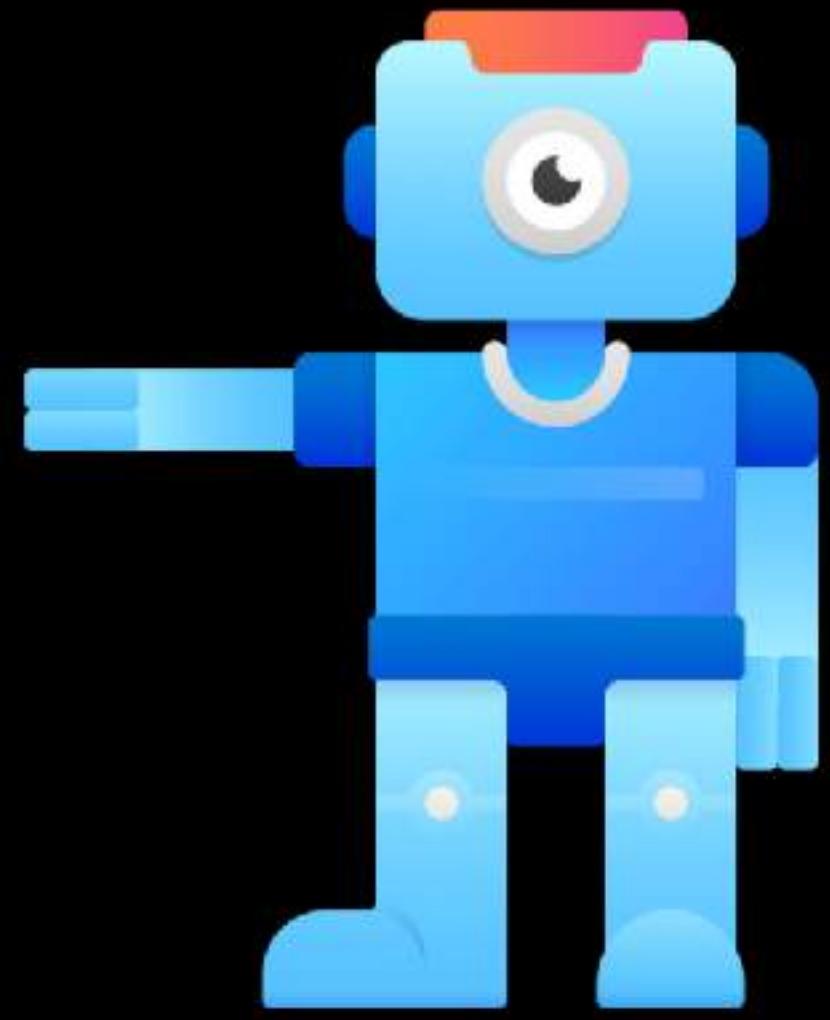


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Why CNNs work so well for Images**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

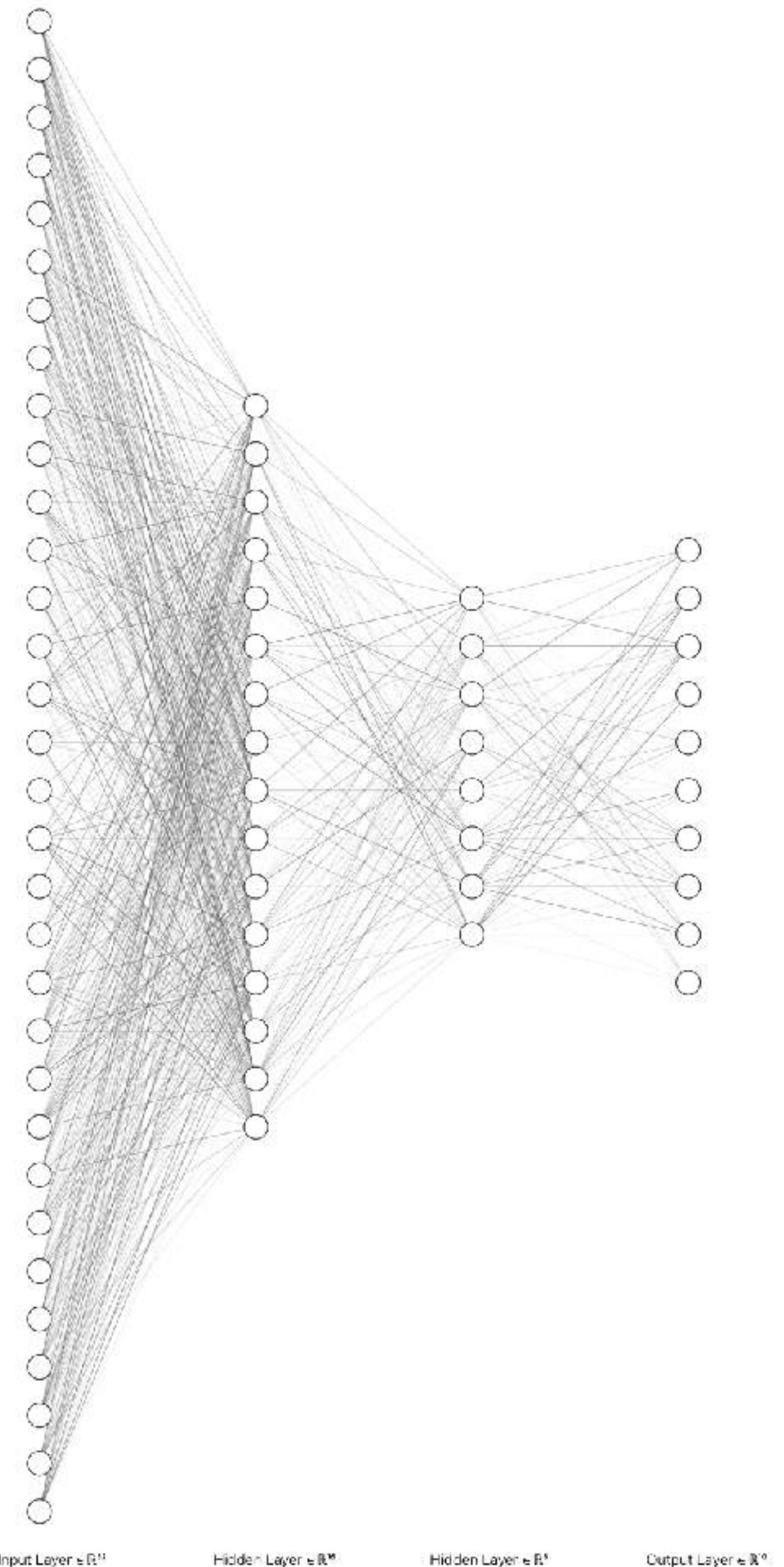
## Why CNNs Work So Well For Images

We explore design choices in CNNs and how it relates to their performance in the real world

# Standard Neural Networks

## A thought experiment...

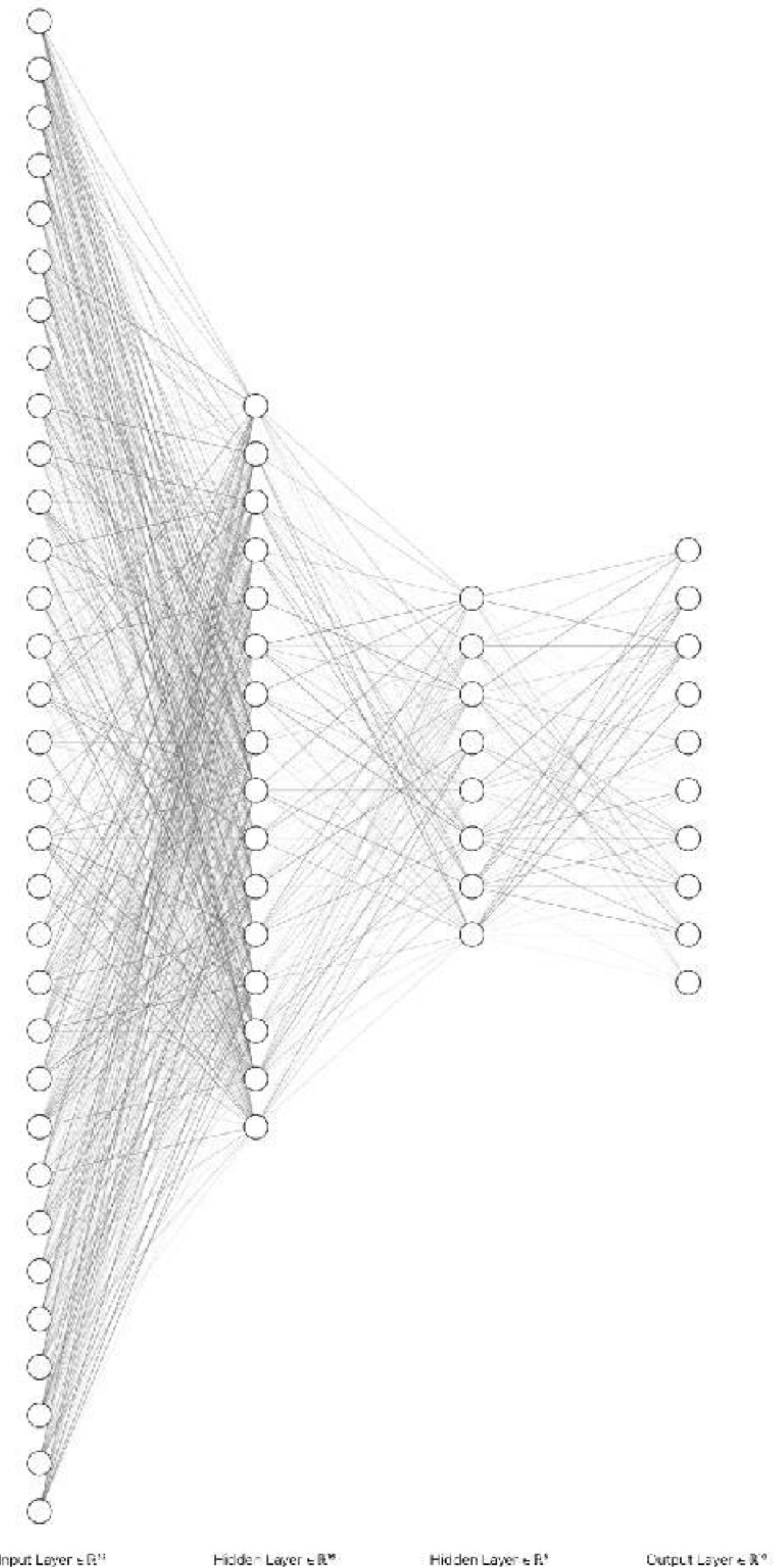
- Standard Neural Networks don't have Convolution Filter Inputs
- For Images every pixel will be it's own input
- Therefore, a small image that's  $28 \times 28$  would have 784 input nodes for our first layer



# Standard Neural Networks

## A thought experiment...

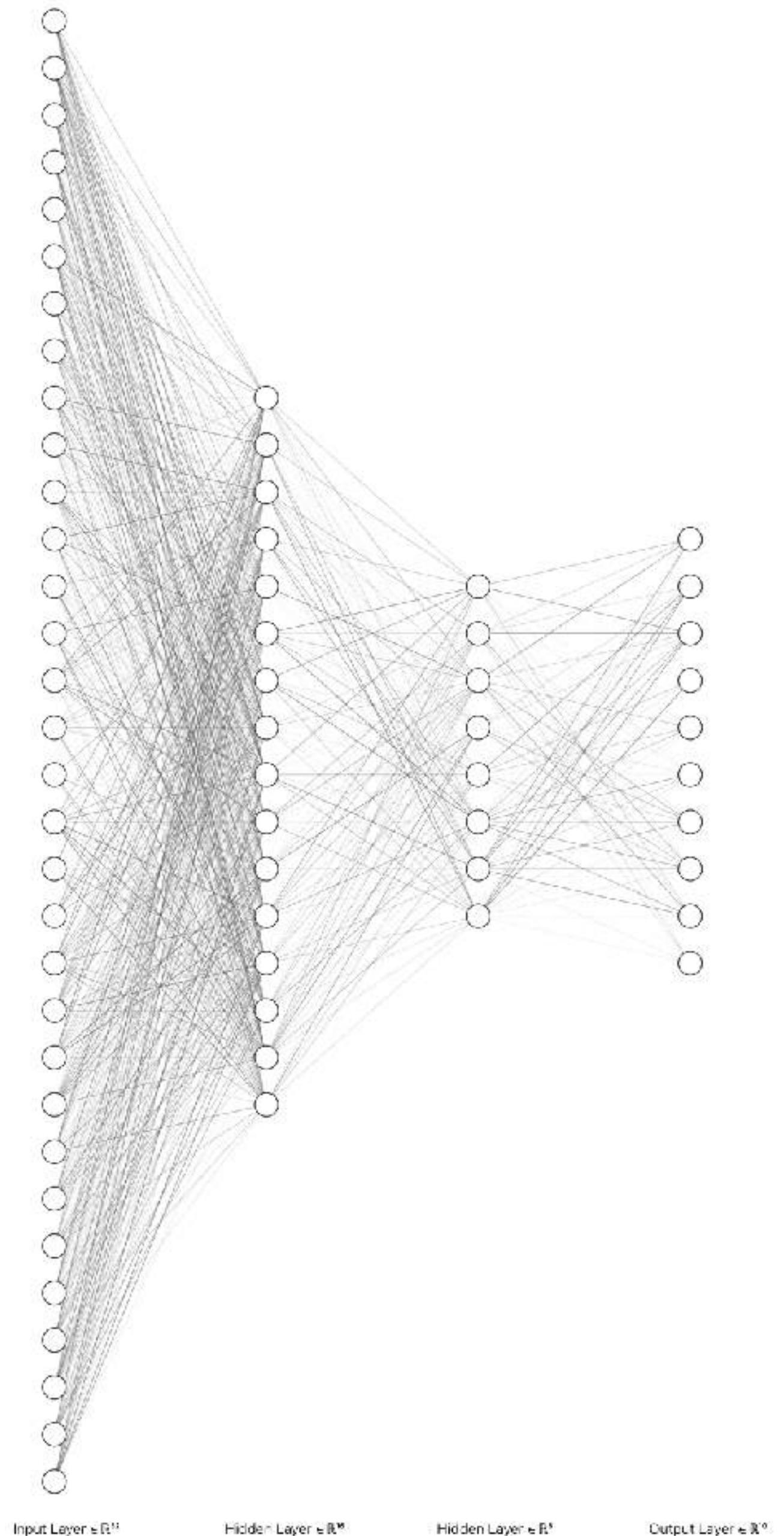
- Our second layer, if we breakdown our previous CNN model would be:
  - 32 Filters for  $26 \times 26$  Feature Maps
  - 13,312
- If they were fully connected, our weight matrix would be:
  - $784 \times 13,312 = \mathbf{10,436,608}$
- For just one hidden layer!



# Standard Neural Networks

## Not Feasible for Image Classification!

- Not scalable to large data inputs
- Overfitting



# Advantages of Convolution Neural Networks

- **Parameter sharing** - where a single filter can be used all parts of an image
- **Sparsity of connections** - As we saw, fully connected layers in a typical Neural Network result in a weight matrix with large number of parameters.
- **Invariance** - Remember our Max Pool Example

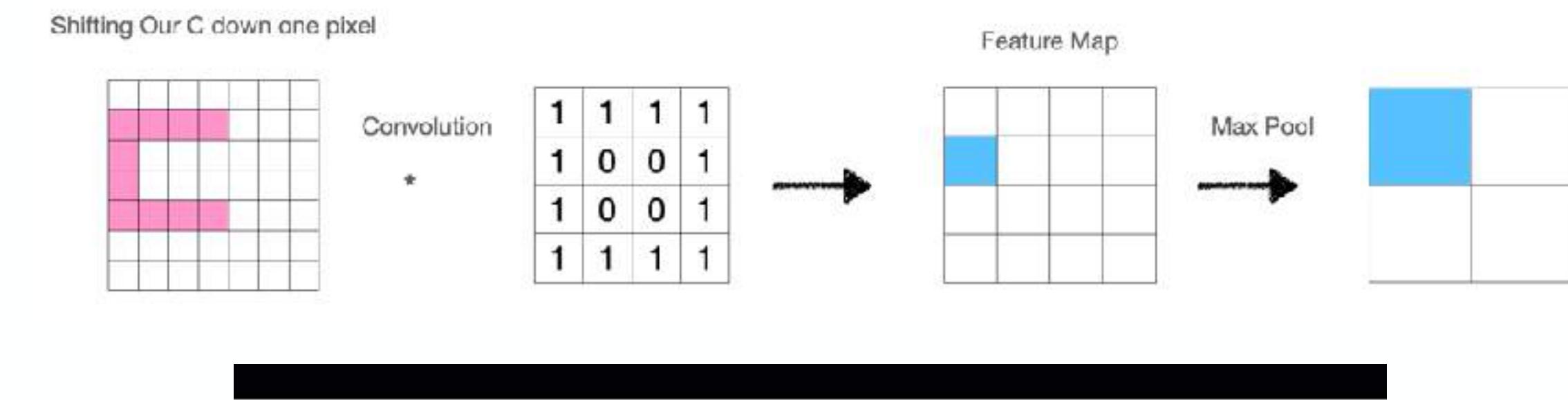
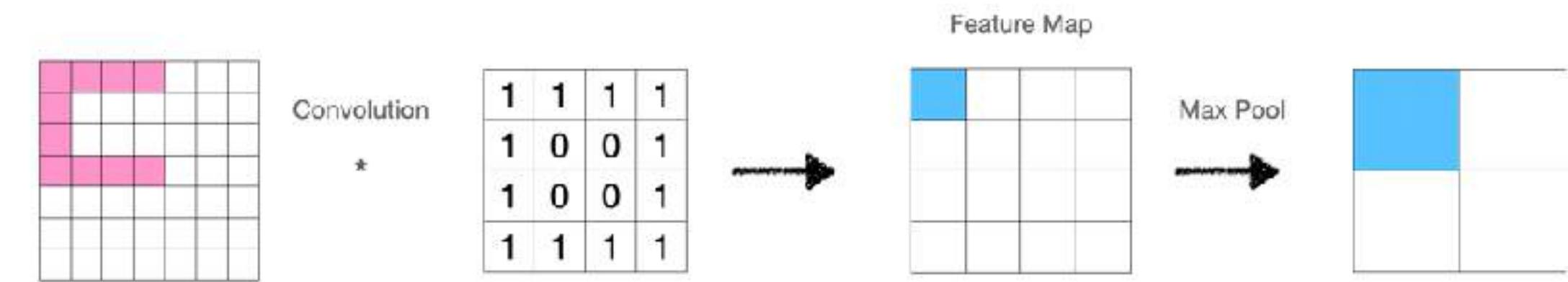


Diagram illustrating the backward pass (gradient flow) of a convolutional neural network layer:

Input (Image): A 5x5 grid with a pink "L" shape.

Filter (Weight Matrix):

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Operation:

Filter (Weight Matrix):

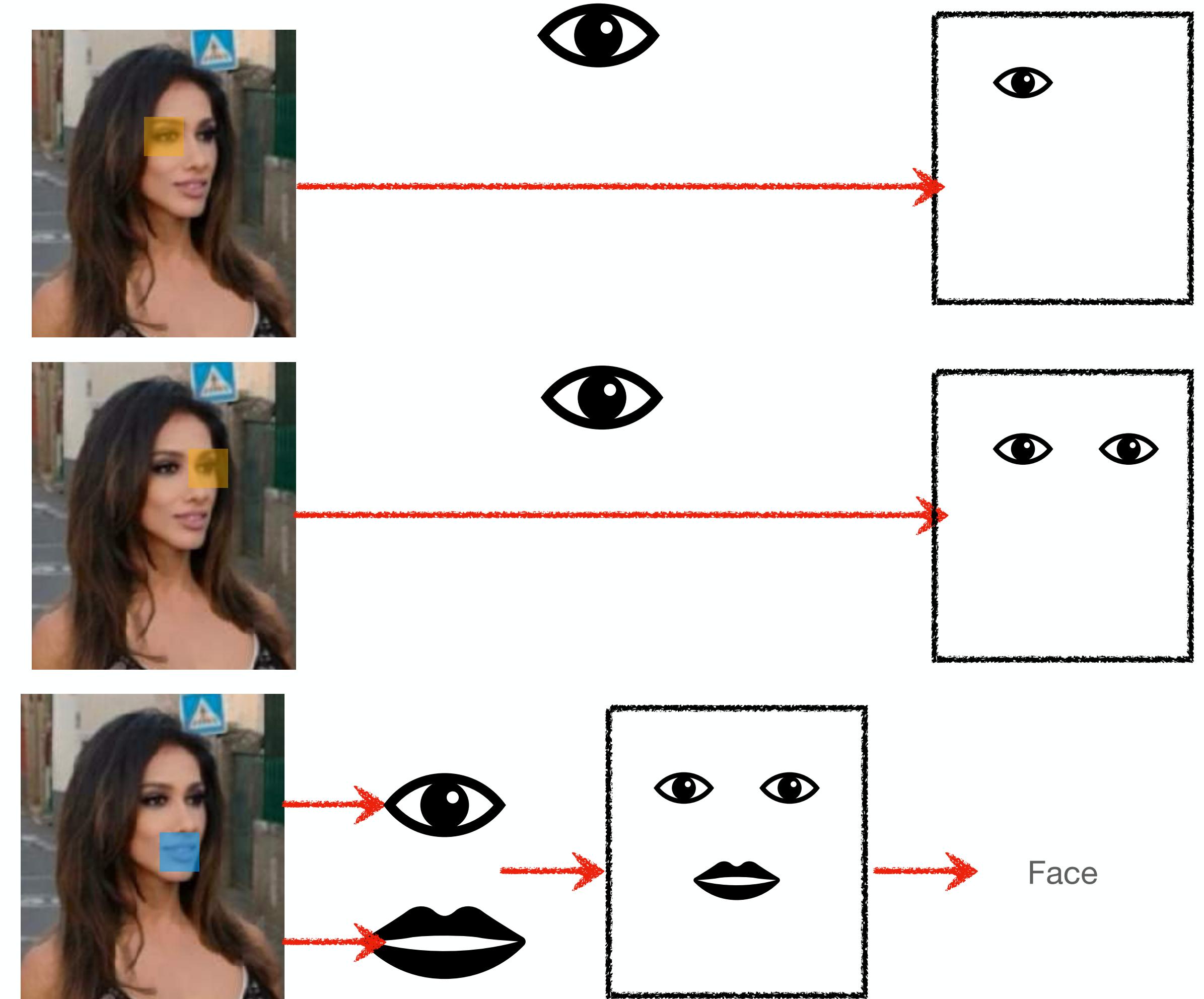
0	1	0
1	0	-1
0	1	0

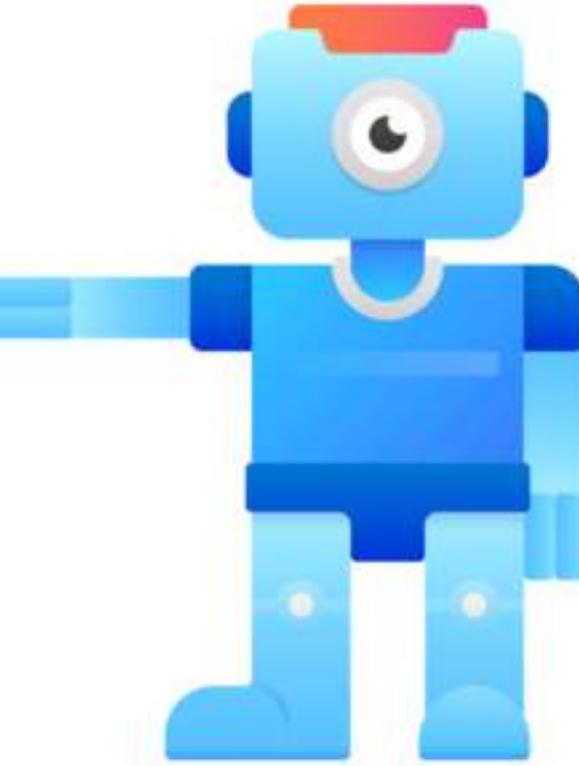
The diagram shows the gradient flow from the input image through the convolution operation to the filter matrix.

---

# Convolution Neural Networks Assumptions

- Low-level features are local
- Features are translational invariant
- High-level features are made up of low-level features



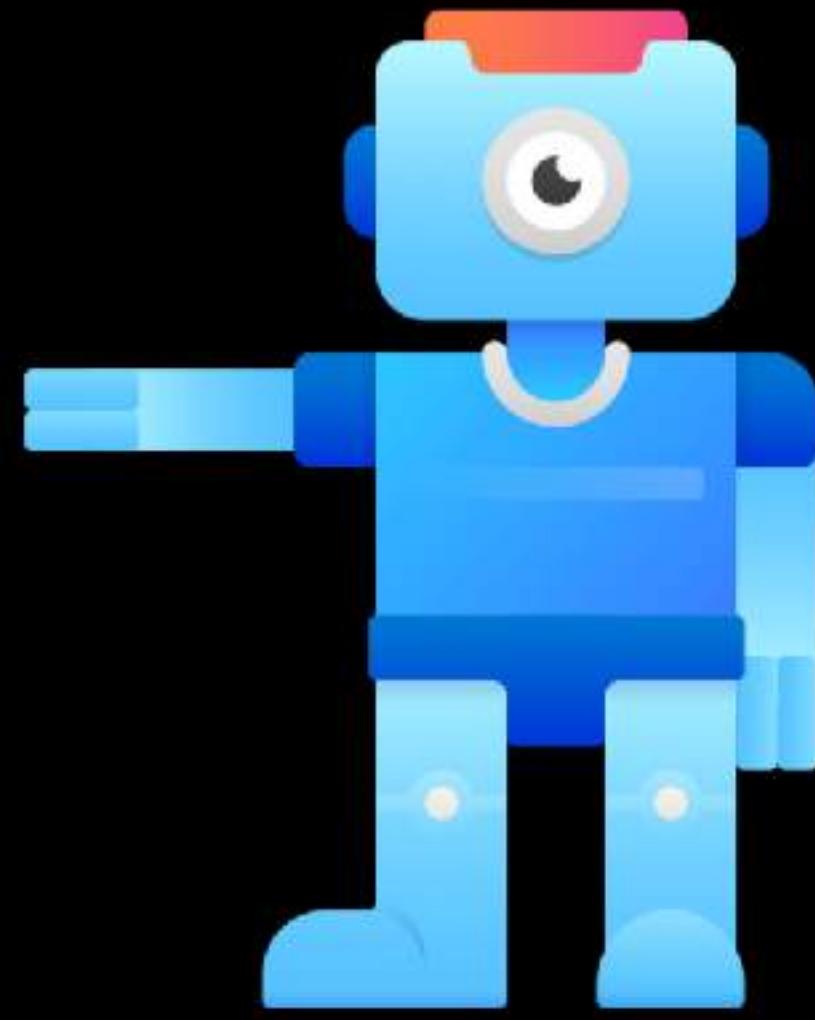


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**How to Train a CNN**



# MODERN COMPUTER VISION

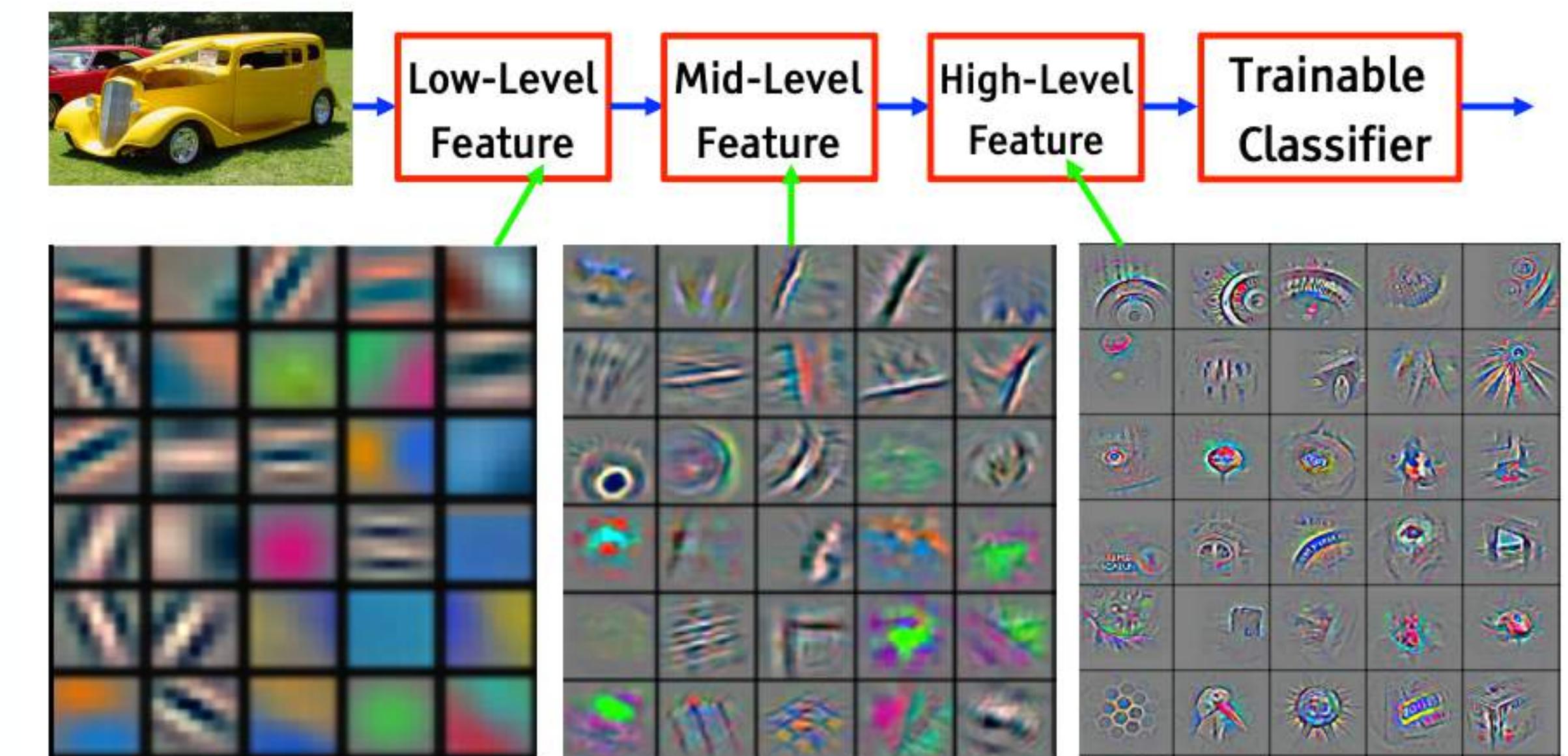
BY RAJEEV RATAN

## How to Train a CNN

We now explore a high level view of how the training process works

# What Conv Filters Learn

- Typically early layers of our CNN learn **low level** features (like edges, or lines)
- Mid-level layers learn **simple patterns**
- High-level layers learn more **structured complex patterns**
- **How is this done?**

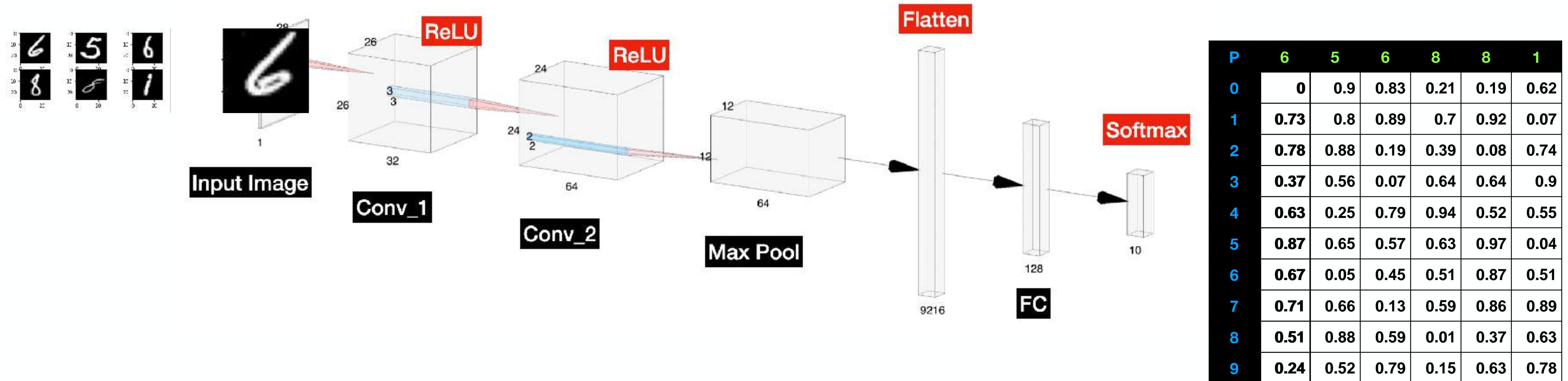


<http://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf>

# What Happens During Training?

- **Initialise random weights** values for our trainable parameters
- **Forward propagate** an image or batch of images through our network
- Calculate the **total error**
- Use **Back Propagation** to update our gradients (weights) via Gradient Descent
- **Propagate more images** (or batch) and update weights, until all images have been propagated (one epoch)
- **Repeat a few more epochs** (i.e. passing all image batches through our Network) until our loss reaches satisfactory values

# The Training Process



# We need to Learn from our Results

- How correct are our results?
- We need a way tell the model it needs to do better

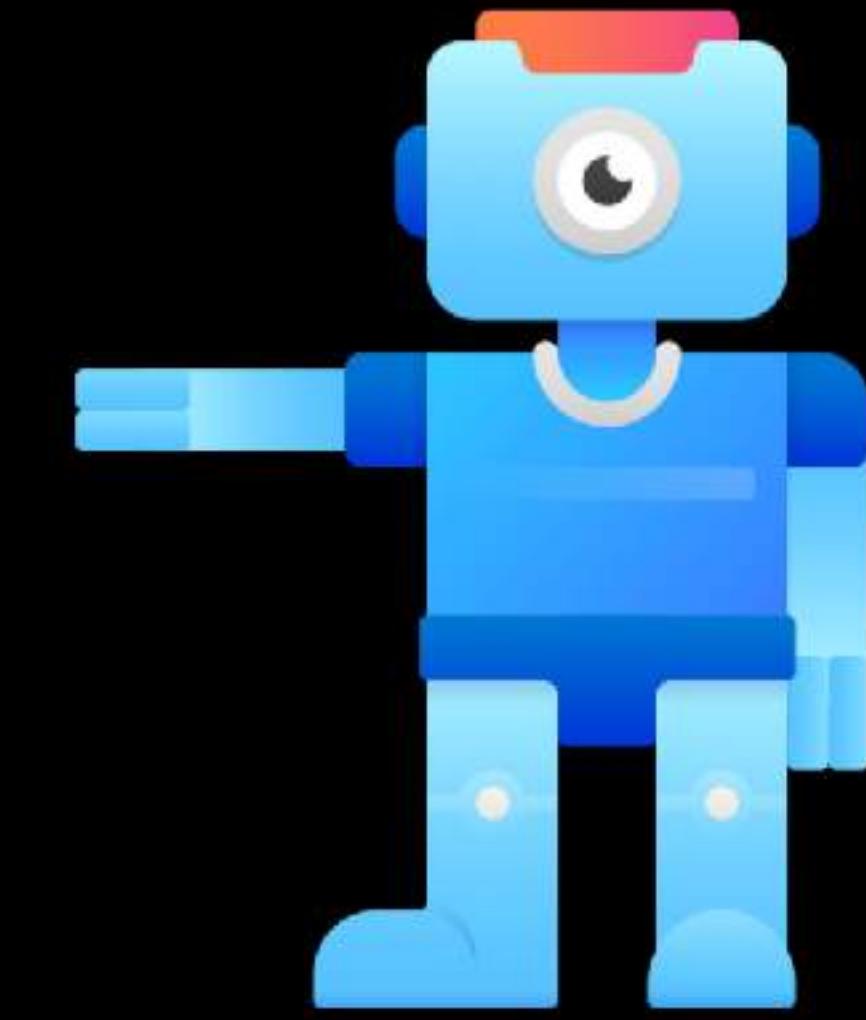


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

## Loss Functions



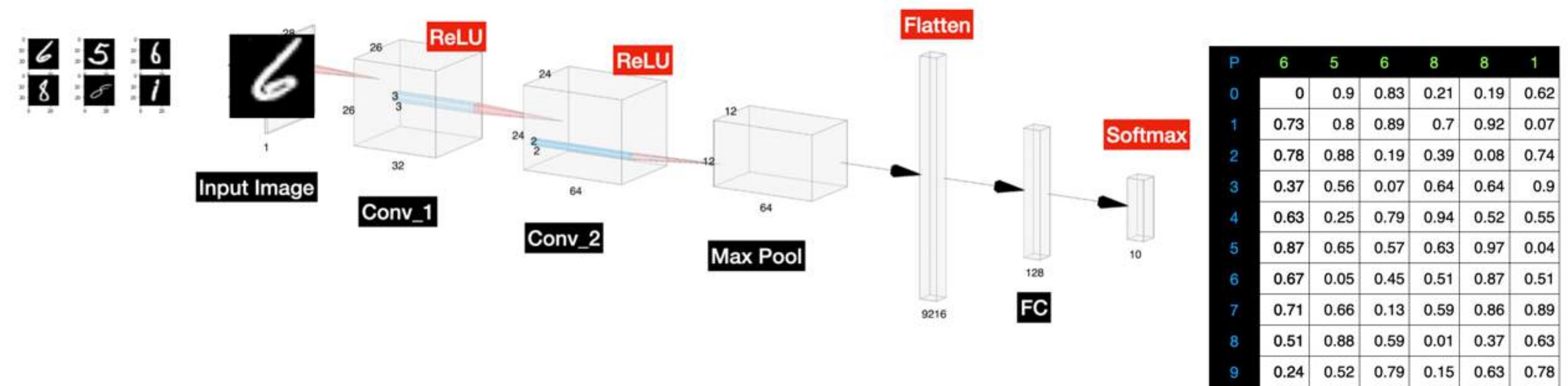
# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Loss Functions

Loss Functions are essential to training

# Quantifying Loss



- How bad are the probabilities we predicted?
- How do we quantify the degree our prediction is off by?

# Cross Entropy Loss or Categorical Cross Entropy Loss

Class	Predicted Probabilities	Ground Truth
0	0.1	0
1	0.2	0
2	0.1	0
3	0.05	0
4	0.05	0
5	0.05	0
6	0.05	0
7	0.3	1
8	0.05	0
9	0.05	0

- Cross Entropy Loss uses two distributions, our ground truth distribution  $p(x)$  and  $q(x)$  our predicted distribution.
- $L = - y \cdot \log(\hat{y})$
- Where  $y$  is the ground truth vector,  $\hat{y}$  is the predicted distribution and ‘ . ‘ is the inner product.

# Cross Entropy Loss a Simpler Example

Class	Predicted Probabilities	Ground Truth
0	0.3	0
1	0.6	1
2	0.1	0

- $L = -y \cdot \log(\hat{y})$
- $L = -(0 \times \log(0.3) + 1 \times \log(0.6) + 0 \times \log(0.1))$
- $L = -(0 + 1 \times -0.222 + 0) = 0.222$
- **NOTE:**
  - Multi-class log loss rewards/penalises the correct classes only

$$L = \frac{1}{N} \sum_{i=1}^N L_i$$

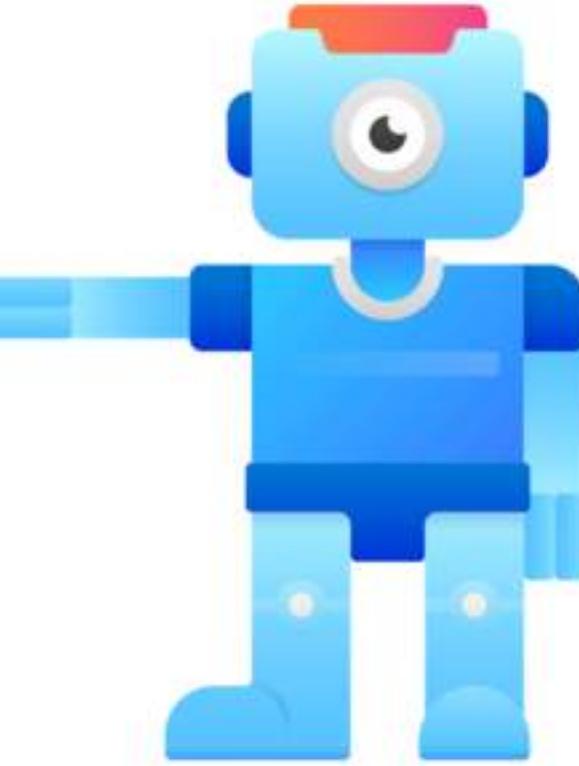
-

# Other Loss Functions

- Loss Functions are sometimes called **Cost Functions**
- For Binary Classification problems we use **Binary Cross-Entropy Loss** (same as categorical cross-entropy loss except it uses just one output node)
- For Regressions we often use the **Mean Square Error (MSE)**
  - Mean Square Error (MSE) =  $(\text{Target} - \text{Predicted})^2$ 
    - $MSE = \frac{1}{n} \sum (Y_i - \hat{Y}_i)^2$
- Other loss functions that are sometimes used:
  - L1, L2
  - Hinge Loss
  - Mean Absolute Error (MAE)

# What do we do with our Quantified Loss?

- Updating all the weights of our model is not trivial
- How do we correctly update our weights to minimise loss?
- We use **Back Propagation**
- And we use the loss value for this!

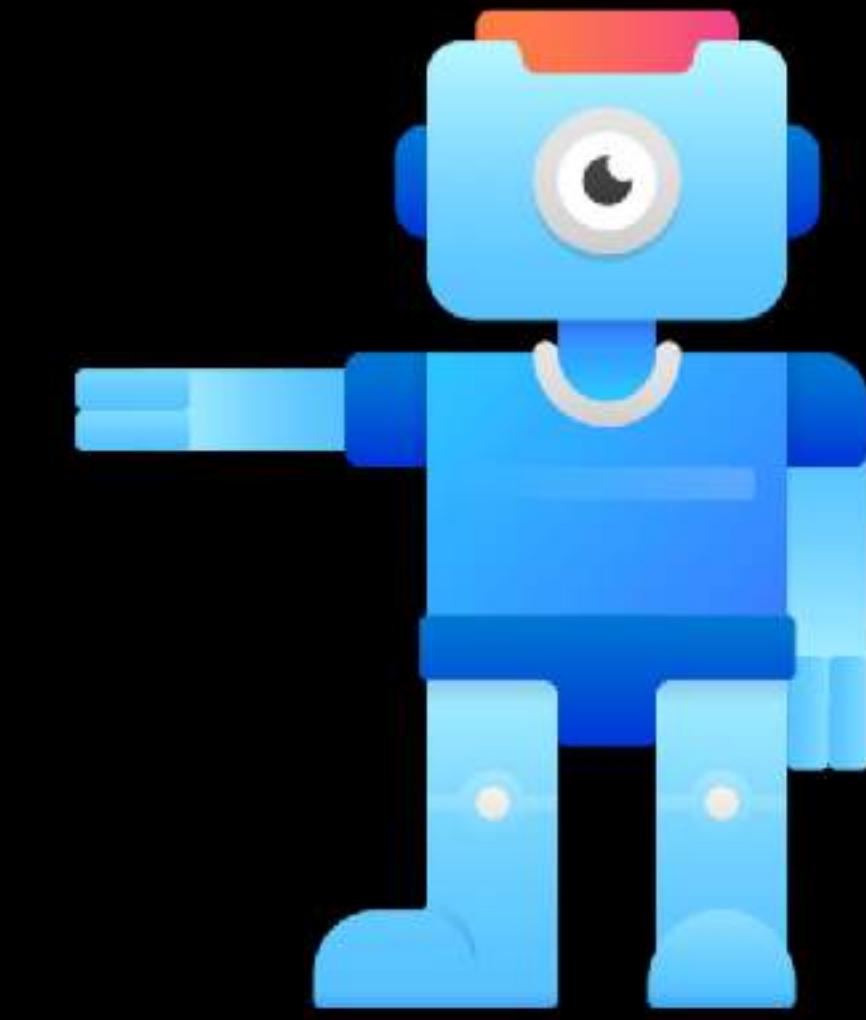


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Back Propagation**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Back Propagation

Back Propagation makes Neural Networks Trainable

# Back Propagation

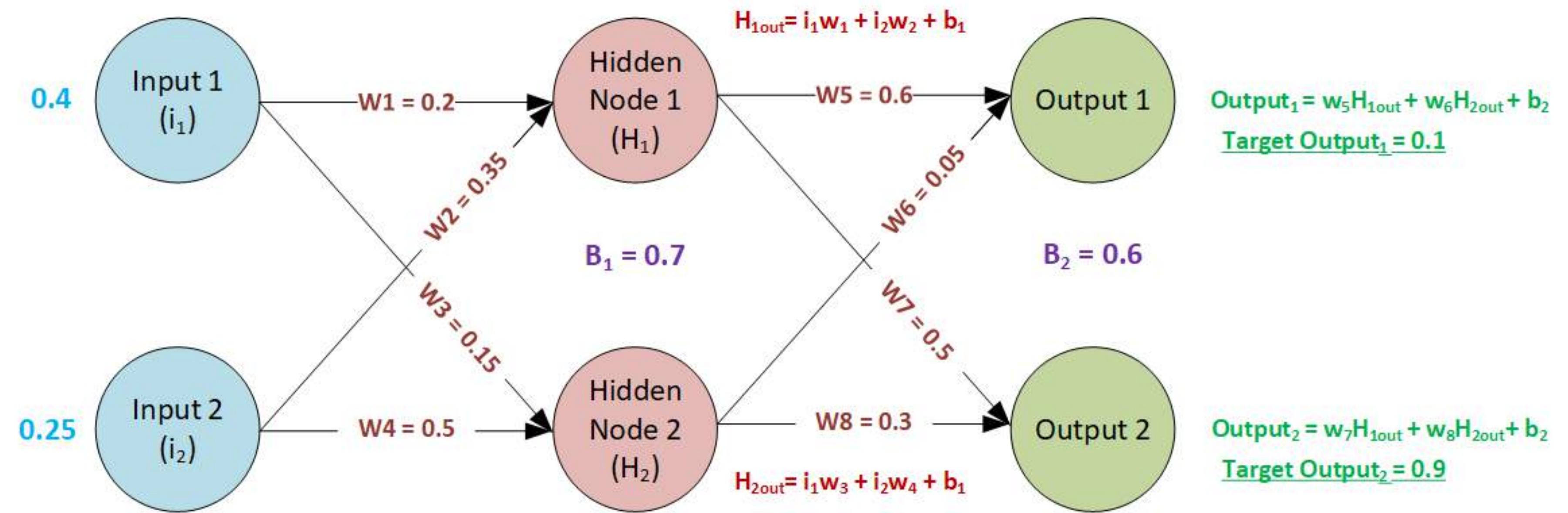
This is what makes Neural Networks Trainable :)

- The importance of Back Propagation cannot be understated
- Using the loss, it tells us how much to change/update the gradients by so that we reduce the overall loss



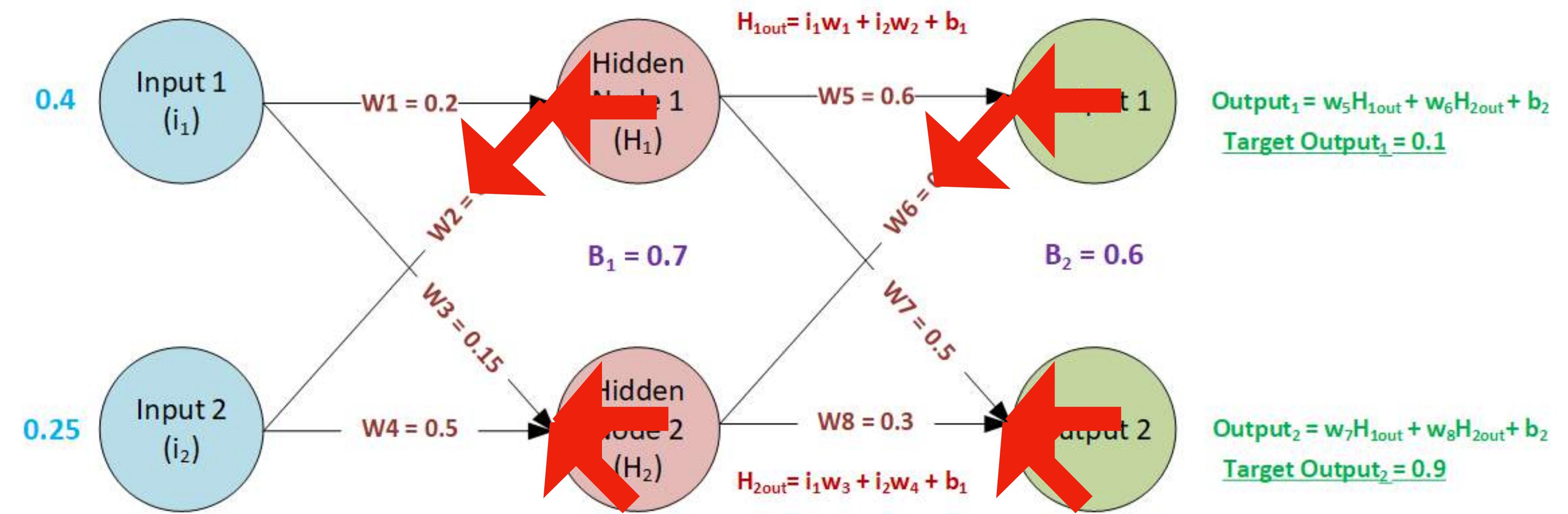
# Back Propagation Example

## Explained Using Neural Networks



- Let's look at a regular Neural Network above.
- Using the Loss value, Back Propagation can tell us whether a **small increase** of  $W_5$  to 0.6001 or a **small decrease** 0.5999 will lead to a **reduction in the overall loss**

# Back Propagation Example



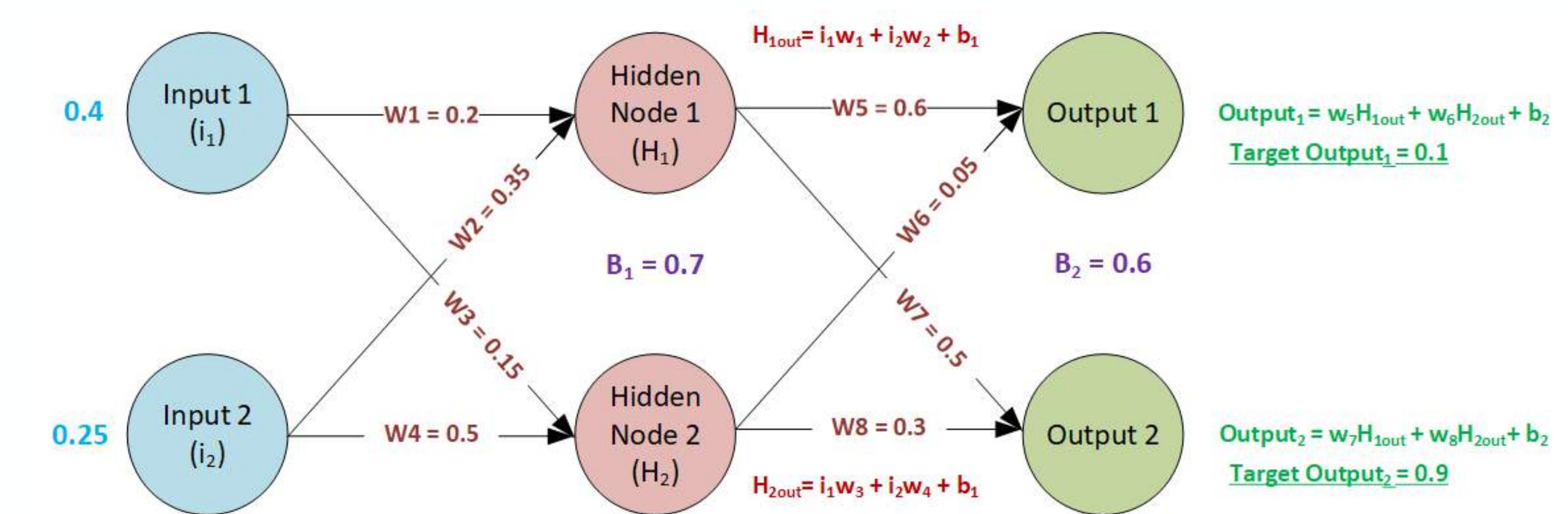
- Moving **right to left**
- Back Propagation gives us the new gradient or weight values for each node so that the overall loss is decreased
- This is done for all nodes

# Back Propagation Process

- By **forward** propagating input data we can use back propagation to **lower** the weights to **lower the loss**
- **But**, this simply tunes the weights for that particular input (or batch of inputs)
- We improve **Generalisation** (ability to make good predictions on unseen data) by using all data in our training dataset
- By continuously changing the weights for each data input (or batch of images) we are lowering the overall loss for our training data.

# What do our Weights or Gradients Look Like?

## Let's look at a Simple Neural Network



- The output from Hidden Node 1 is:
  - $H_{1out} = i_1w_1 + i_2w_2 + b_1$

# For a Convolutional Neural Network

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

\*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	3
2	1	1

Output or Feature Map

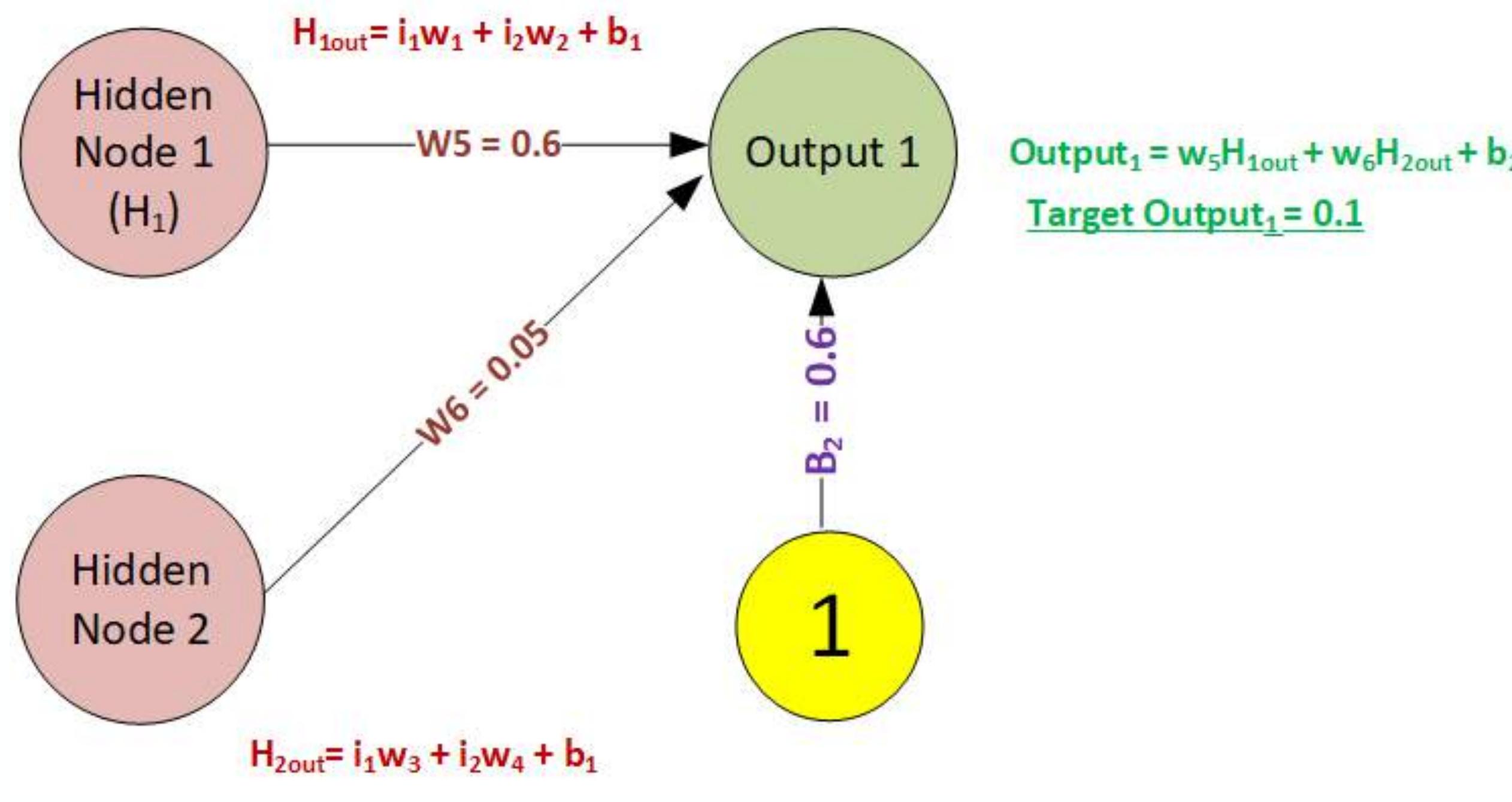
- The values of our Filter/Kernel are the weights!

# How does Back Propagation Work?

- **Chain Rule!**
- If we have two functions  $y = f(u)$  and  $u = g(x)$  then the derivative of  $y$  is:

$$\cdot \frac{dy}{dx} = \frac{dy}{du} \times \frac{du}{dx}$$

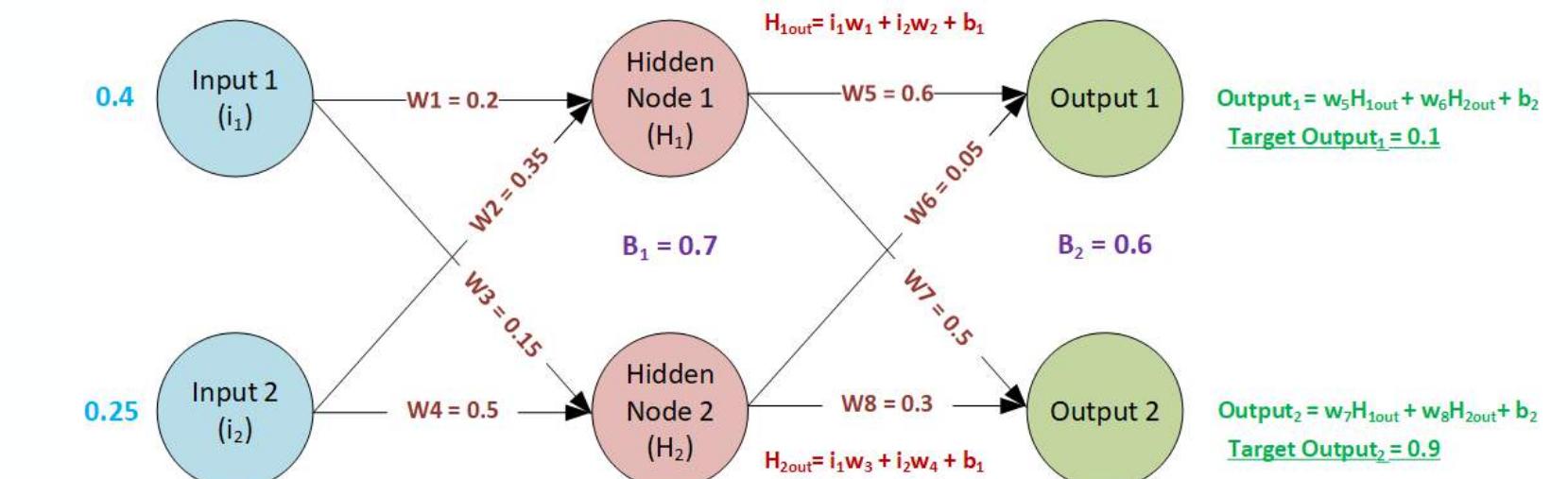
# A Simple Back Propagation Example



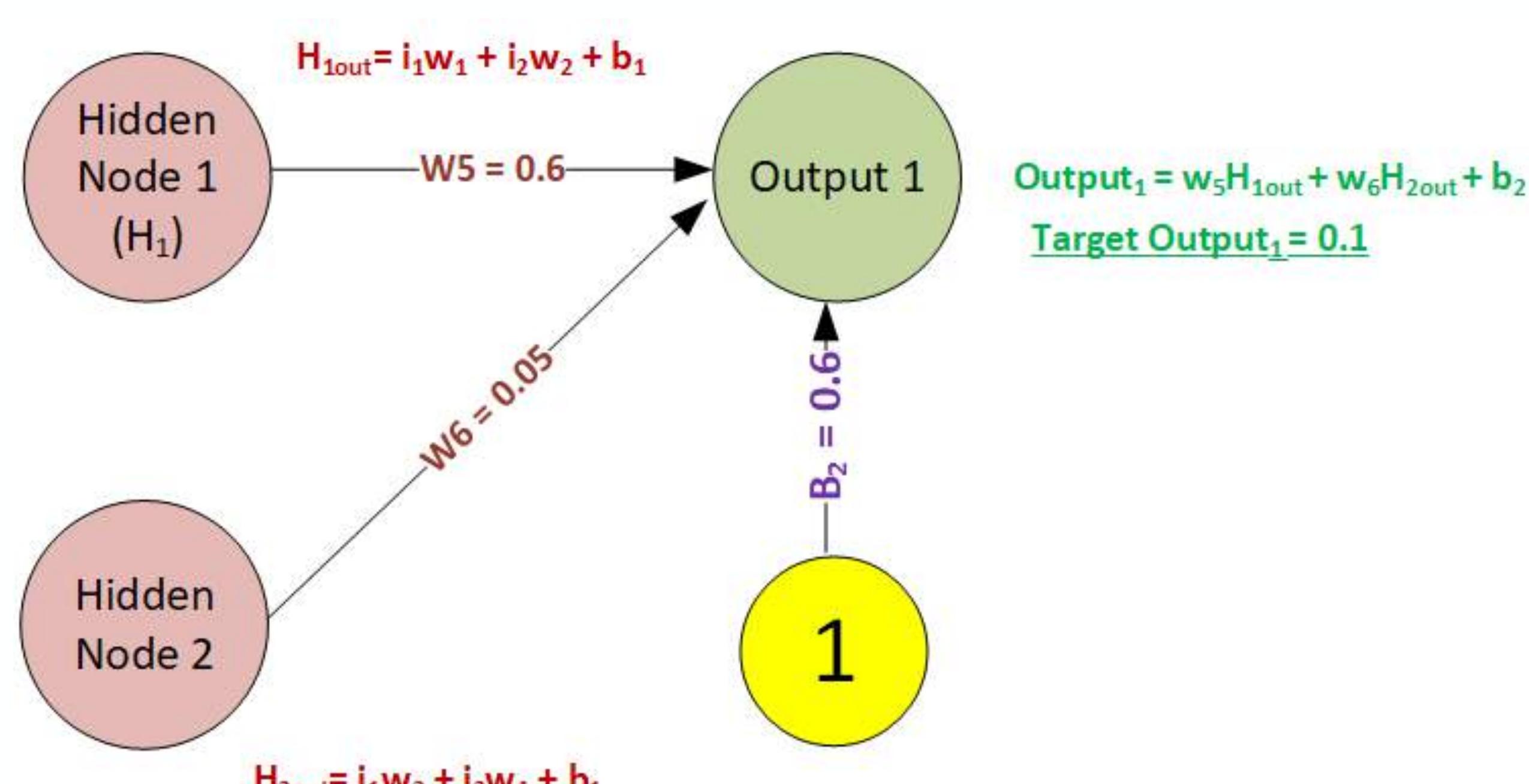
- We want to know how much changing  $W_5$  changes the **Total Error**.
- That is given by:

$$\frac{dE_T}{dW_5}$$

- Where  $E_T$  is the sum of the error from Outputs 1 and 2 (see below)

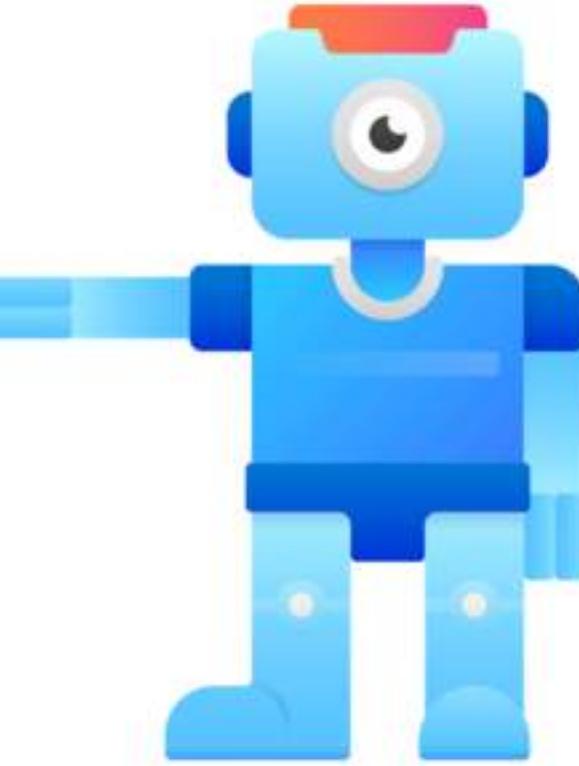


# A Simple Back Propagation Example



$$\text{New } W_5 = -\lambda \times \frac{dE_T}{dW_5}$$

- Note we introduced a new parameter  $\lambda$
- $\lambda$  is our learning rate
- It controls how a big a jump (positive or negative) we take when updating  $W_5$
- Large learning rates train faster, but can get stuck in a Global Minimum
- Small learning rates train more slowly



# MODERN COMPUTER VISION

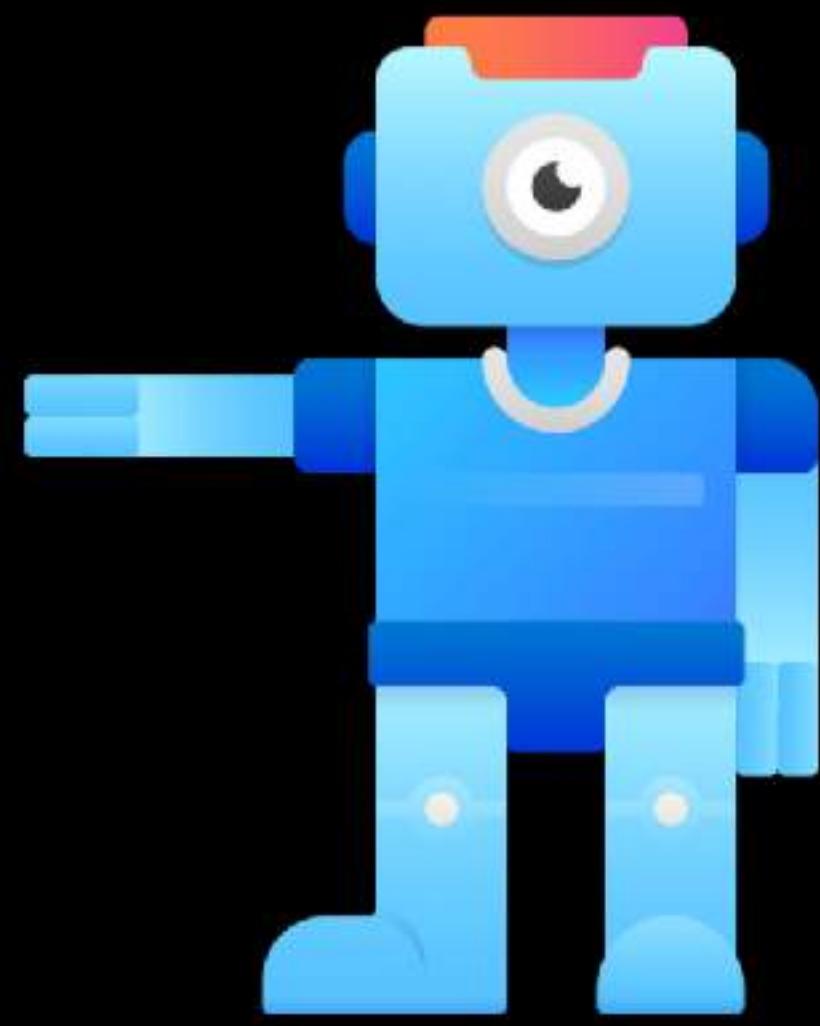
BY RAJEEV RATAN

# Next...

**Gradient Descent**

# Gradient Descent

Finding the optimal weights



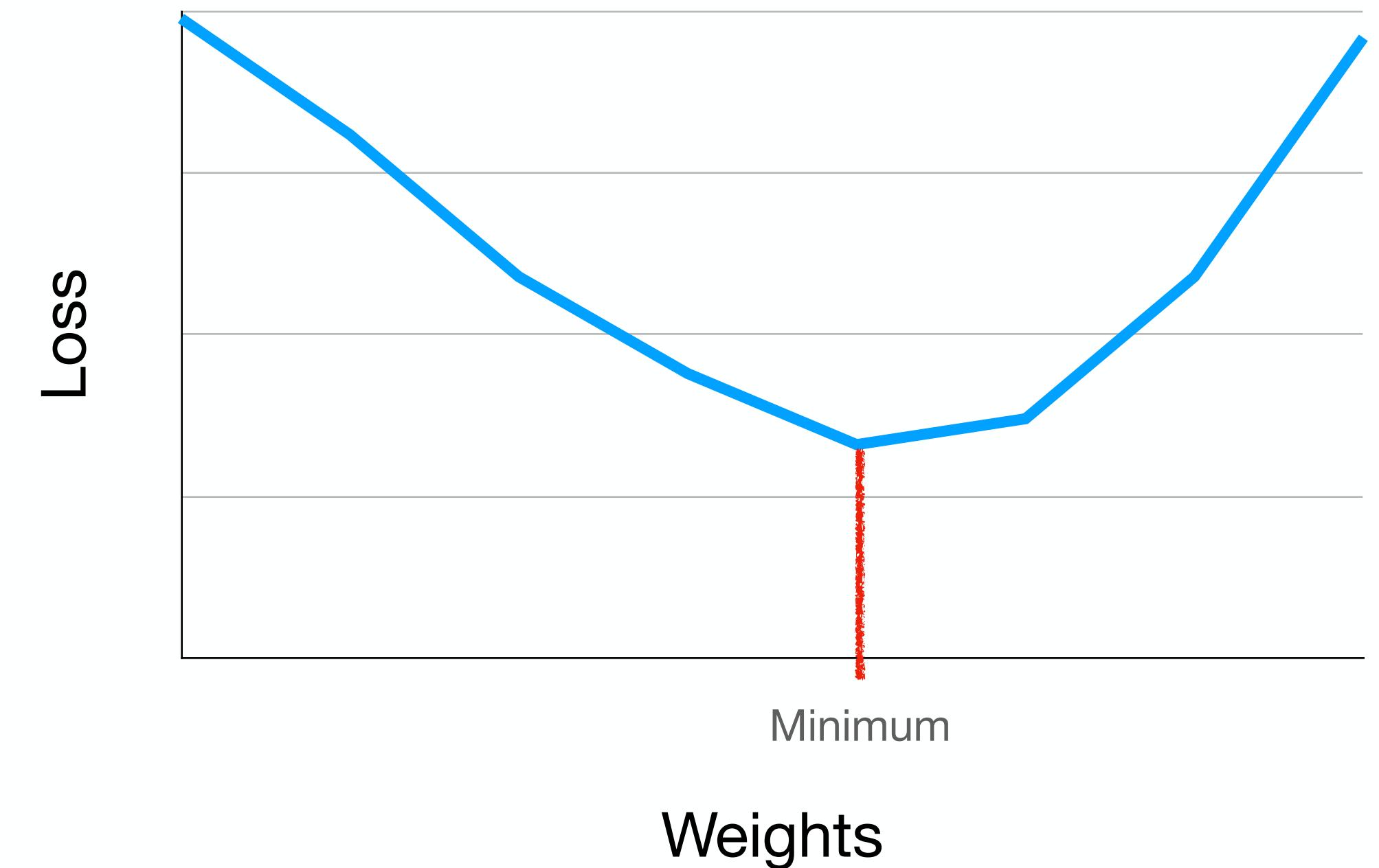
# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Loss Functions

## How do we find the lowest loss?

- Back Propagation is the process we use to update the individual weights or gradients
  - $wx + b$
- Our goal is finding the right value of weights where the loss is lowest
- The method by which we achieve this goal (i.e. updating all weights to lower the total loss) is called **Gradient Descent**
- It's the point at which we find the **optimal weights** such that **loss is near lowest**



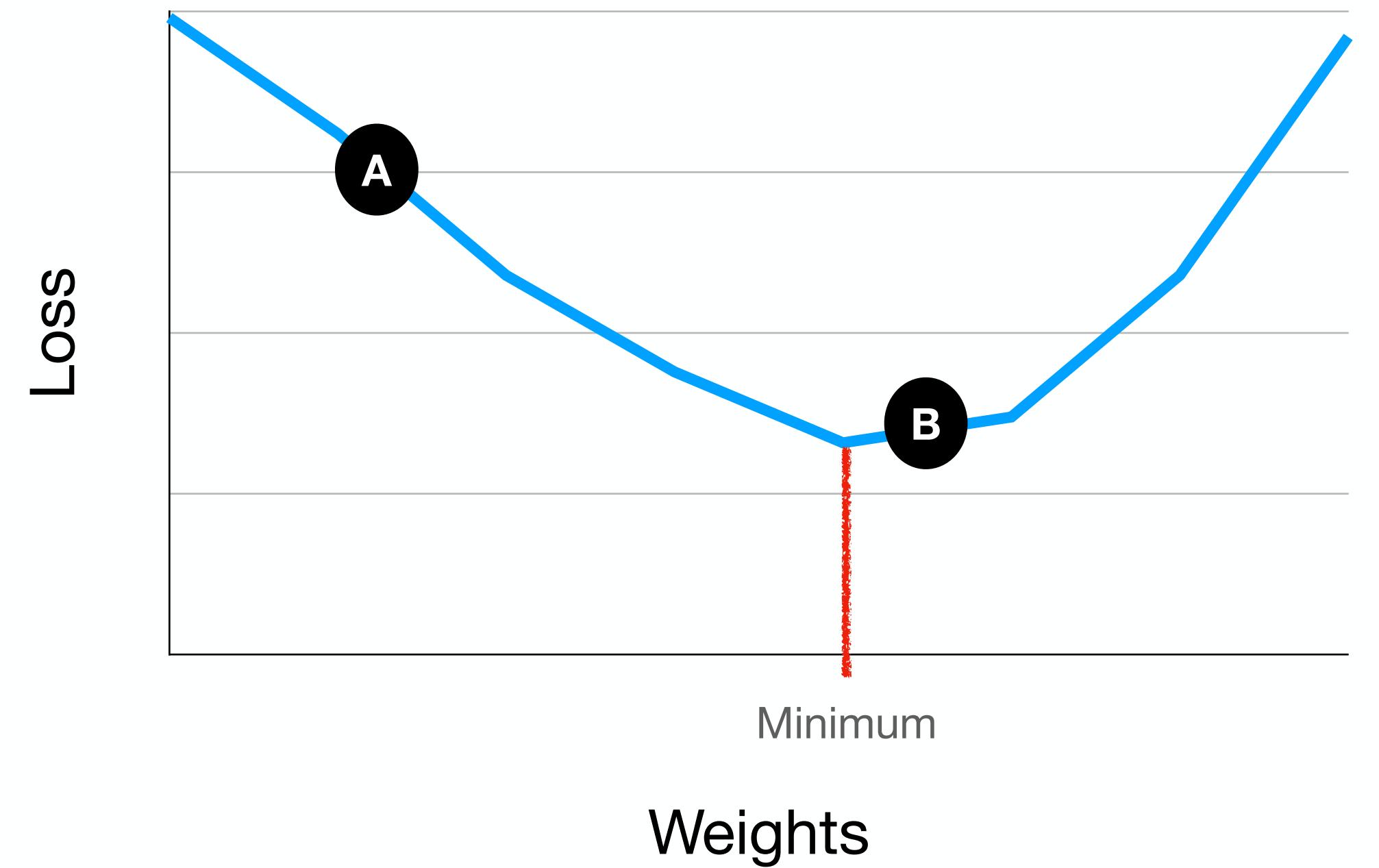
# Gradients

## Gradients are the derivative of a function

- It tells us the rate of change of one variable with respect to the other e.g.

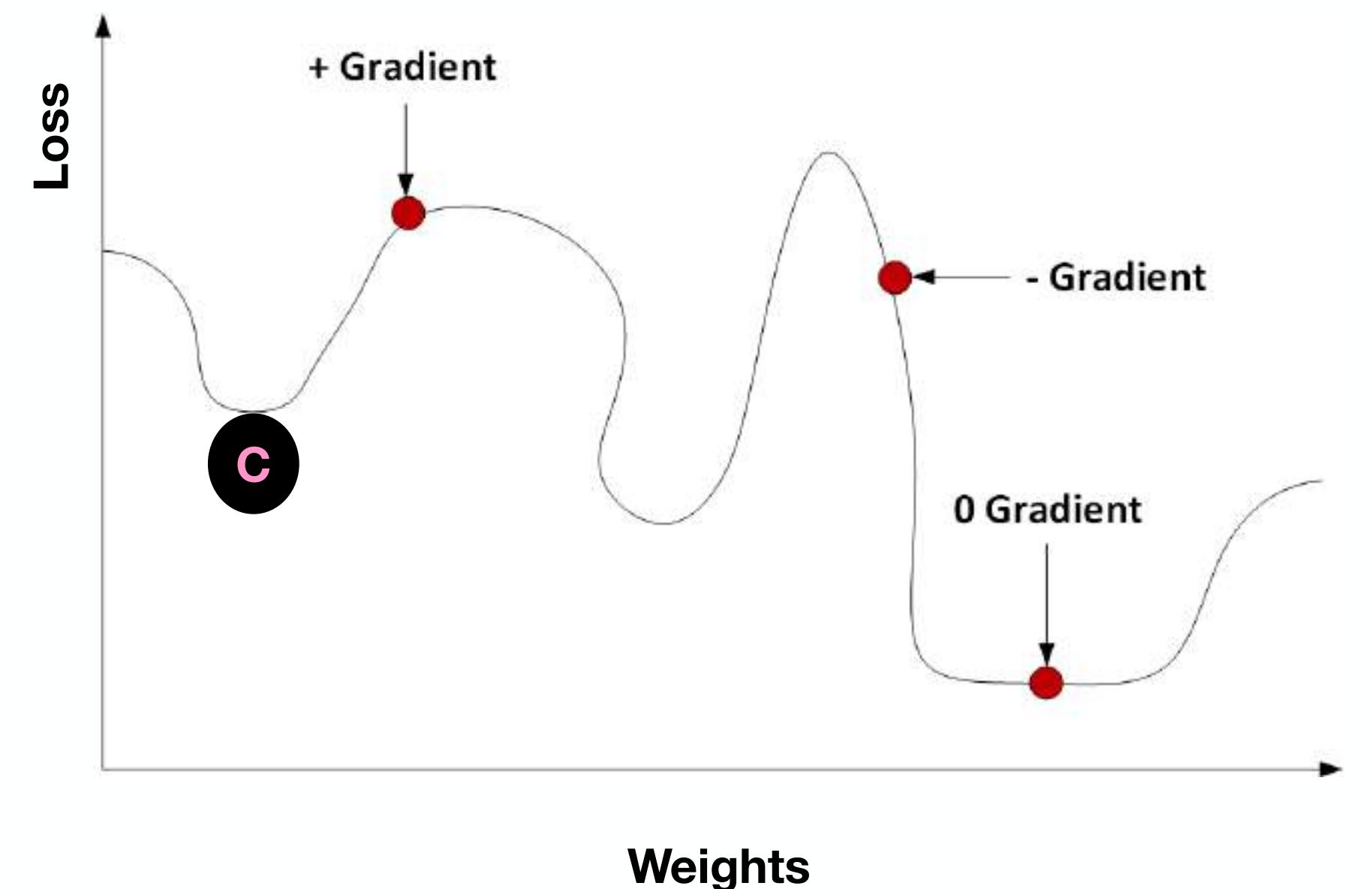
$$\text{Gradient} = \frac{dE}{dw}, \text{ where } E \text{ is the Error or Loss and } w \text{ is the weight}$$

- A positive gradient means, loss increases if weights increase
- A negative gradient means, loss decreases if weights increase
- At point A, moving right increases our weights and decreases our loss, -ve
- At point B, moving right increases our weights and increases our loss, +ve
- Therefore, the **negative** of our gradient tells us the direction we should be moving



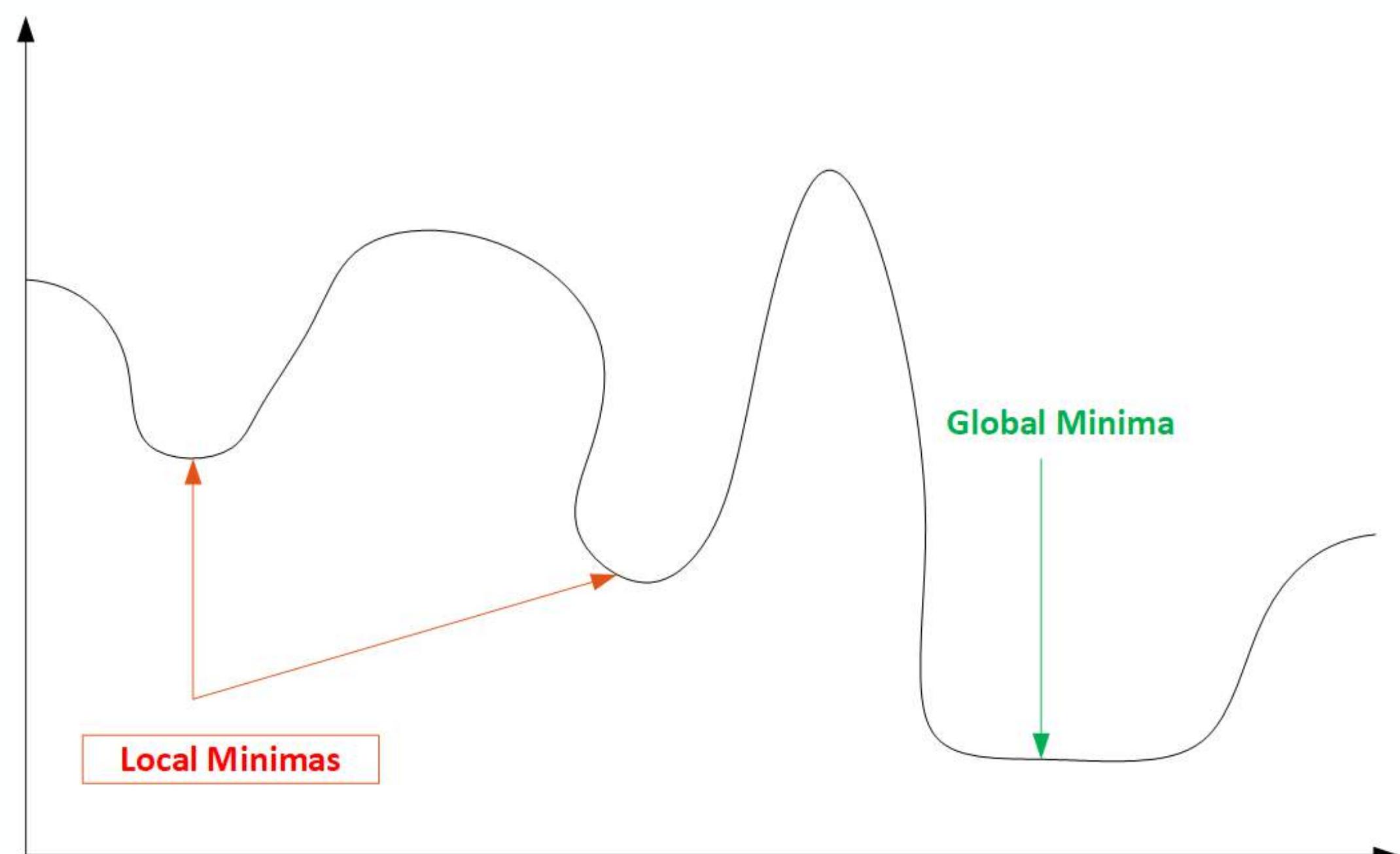
# More on Gradients

- The point at which a Gradient is zero means that small changes to the left or right don't change the loss
- In training Neural Networks this is good and bad
- At point, C, very small changes to the left or right don't change the Loss
- This means, our network gets stuck during training.
- This is called getting **stuck in a Local Minima**



# Local and Global Minimas

- We always want to find the Global Minima during training.
- That is the point where the combination of weights give the lowest loss

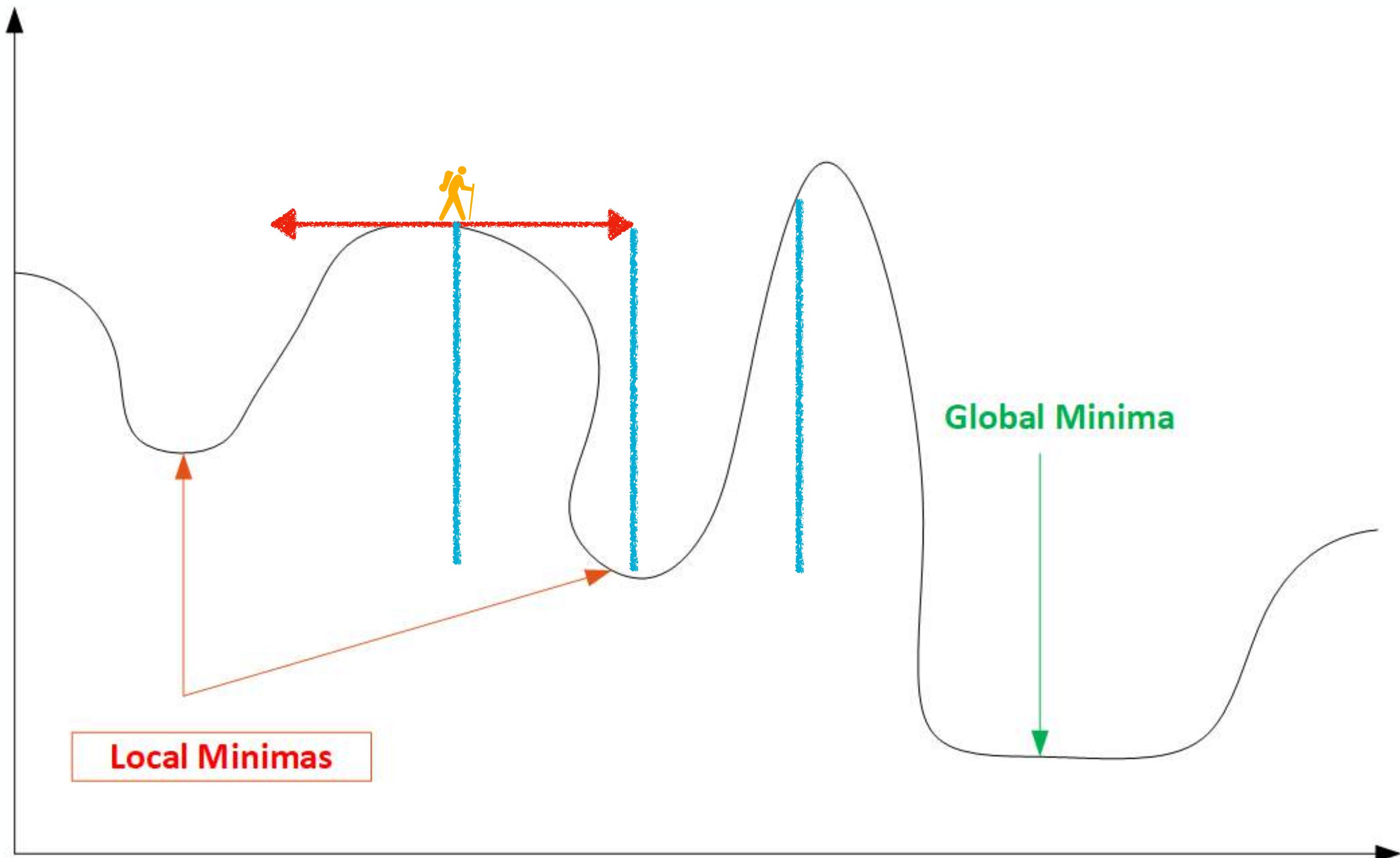


# Gradient Descent

- Imagine being a really tiny person and you're traversing down the slope of this old, rough bowl.
- There'd be peaks, valleys, troughs etc.
- How do you know when you're truly at the bottom?
- Possibly take large steps so you don't get stuck in a valley
- But then you risk jumping over the Global Minima



# Step Size is Important



# Learning Rates

## Recall our Back Propagation Weight Update Formula

- $W_5 = -\lambda \times \frac{dE_T}{dW_5}$
- $\lambda$  is our learning rate
- Learning Rates allow us to adjust the magnitude of jumps in weights
- Finding an optimal value will avoid us finding **Local Minimums** and while preventing us from jumping over the **Global Minimum**.

# Gradient Descent Methods

- **Naive Gradient Descent** - Passes the entire dataset through our network then updates the weights.
  - It is computationally expensive and slow.
- **Stochastic Gradient Descent (SGD)** - Updates weights after each data sample (image) is forward propagated through our network.
  - This leads to noisy fluctuating loss values and is also slow to train
- **Mini-Batch Gradient Descent** - Combines both methods, it takes a batch of data points (images) and forward propagates all, then updates the gradients.
  - This leads to faster training and convergence to the Global Minima
  - Batches are typically 8 to 256 in size

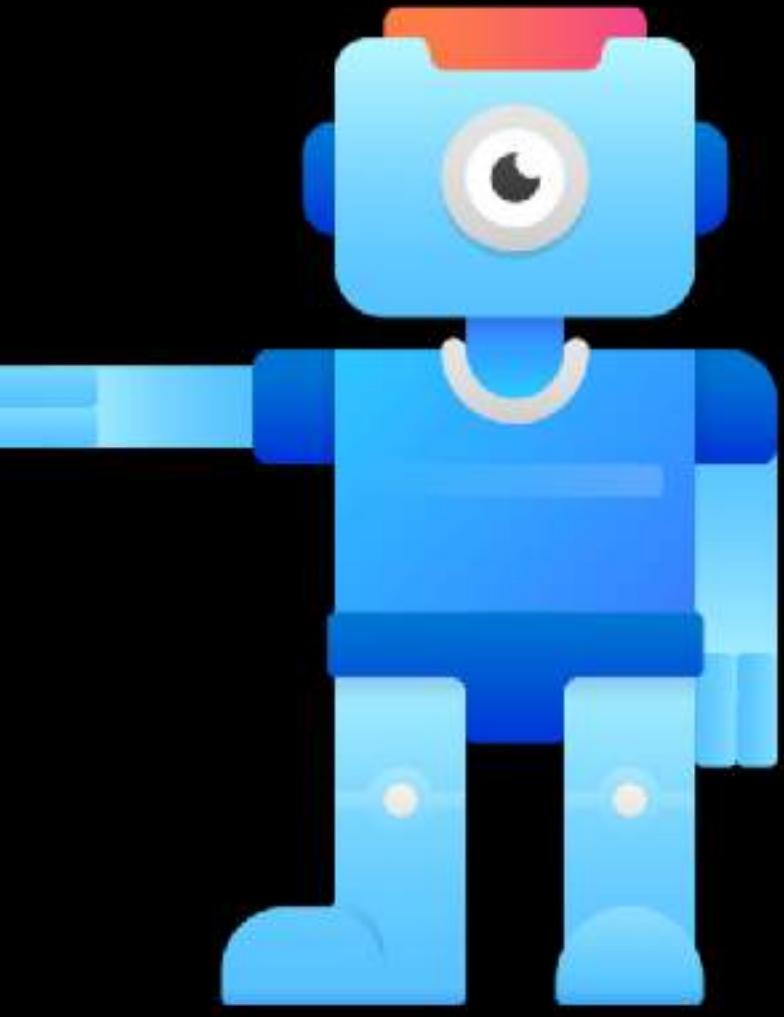


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Optimisers**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

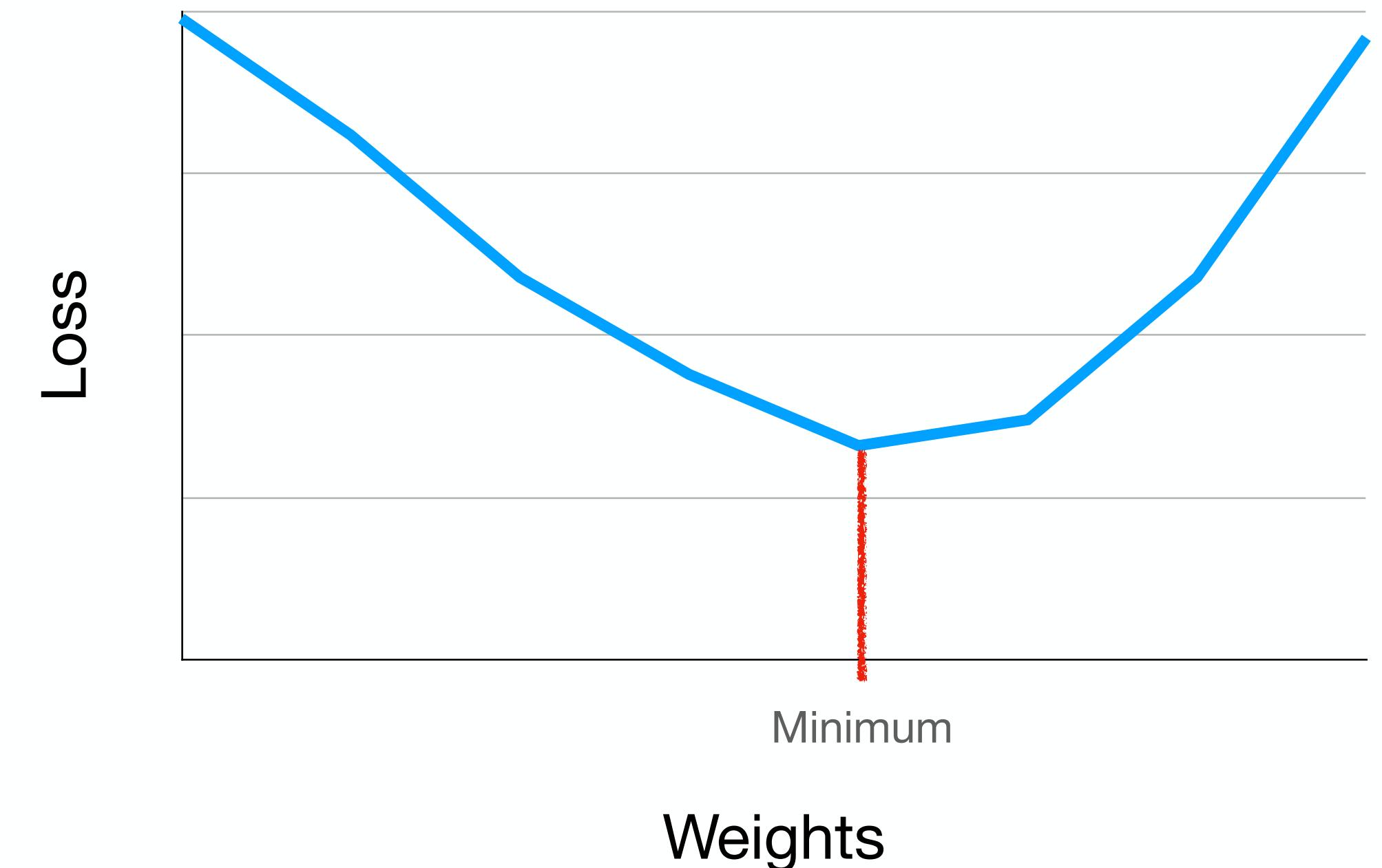
## Optimisers & Learning Rate Schedules

Methods or Algorithms used in finding optimal weights

# Optimisers

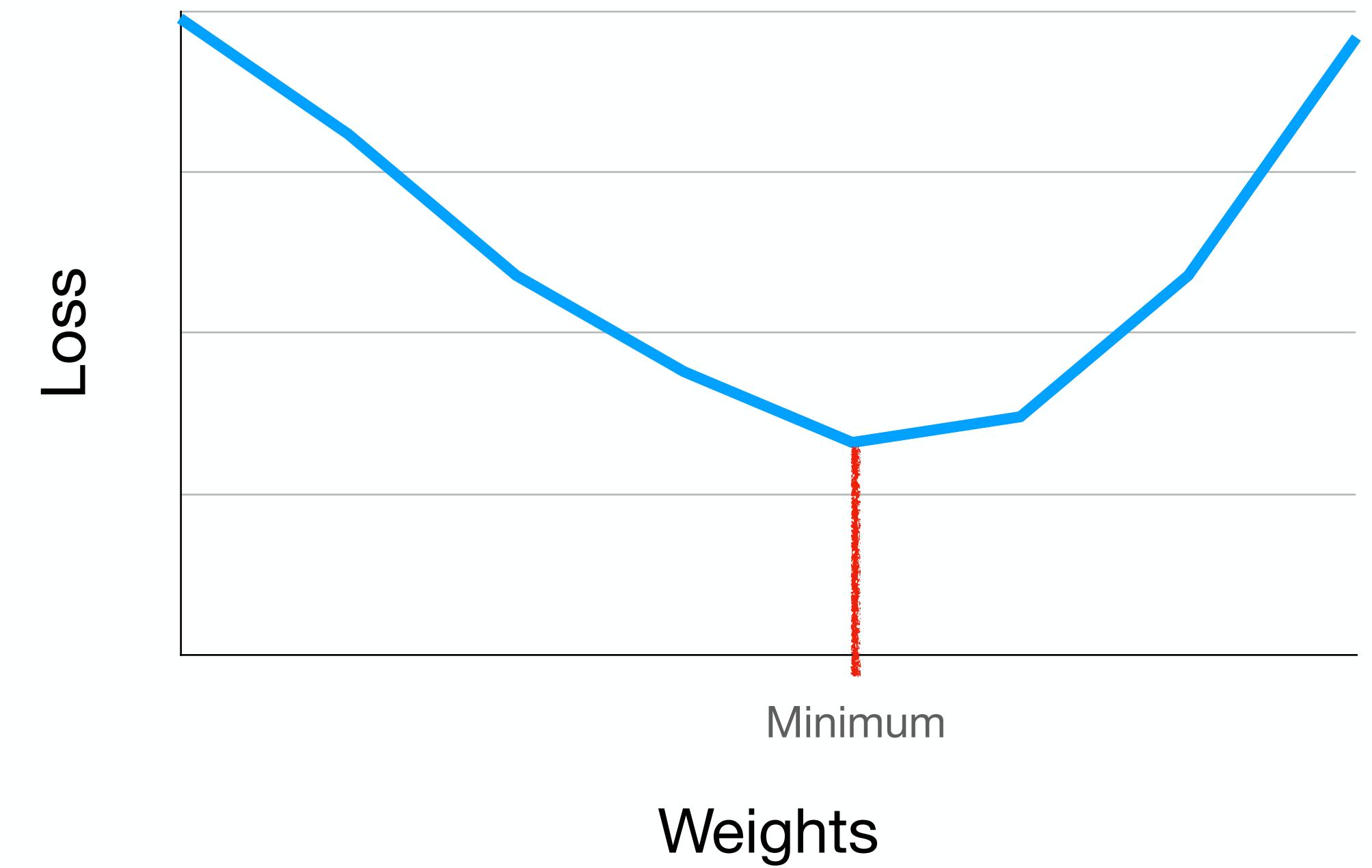
**The algorithm we use to update the weights**

- You have already been introduced to Gradient Descent which is an example of a first order optimisation algorithm.
- In this section we will explore some alternatives to Stochastic Gradient Descent and take a look at a few other optimisation algorithms.



# Problems with standard SGD

- Choosing an appropriate Learning Rate (LR), deciding Learning Rate Schedules,
- Using the same learning rate for all parameter updates (as is the case with sparse data), but most importantly SGD is susceptible to getting trapped in Local Minimas or Saddle Points (where one dimensions slopes up and another slopes down).
- To solve some of these issues several other algorithms have been developed including some extensions to SGD which include Momentum and Nestor's Acceleration.



# Momentum

- One of the issues with SGD are areas where our hyper-plane is much steeper in one direction.
- This results in SGD oscillating around these slopes making little progress to the minimum point.
- Momentum increases the strength of the updates for dimensions whose gradients switch directions. It also dampens oscillations. Typically we use a Momentum value of 0.9.

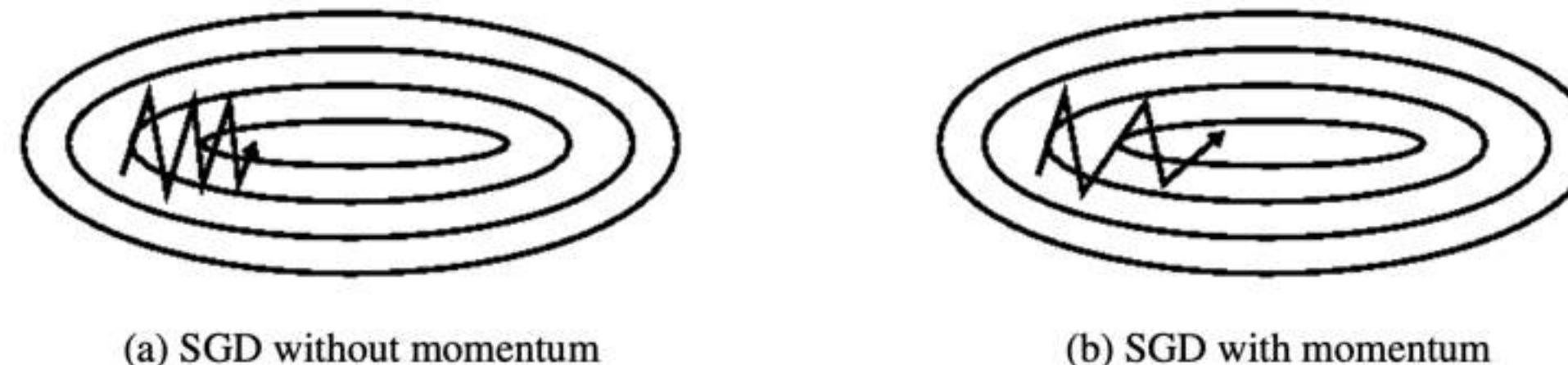


Figure 2: Source: Genevieve B. Orr

# Nesterov's Acceleration

- One problem introduced by Momentum is overshooting the local minimum.
- Nesterov's Acceleration is effectively a corrective update to the momentum which lets us obtain an approximate idea of where our parameters will be after the update.
- Below we show the corrected Gradient updates (in green)

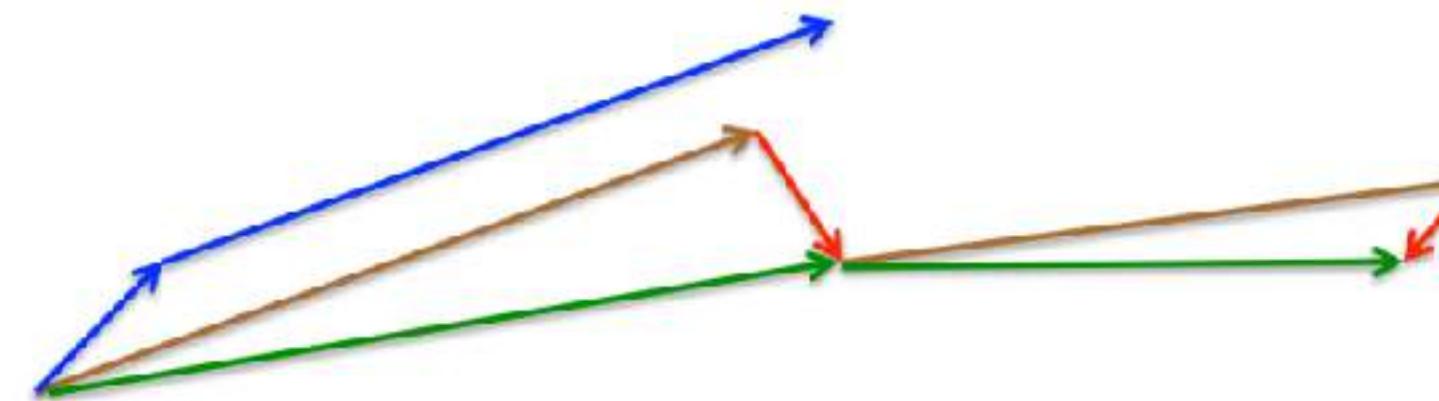
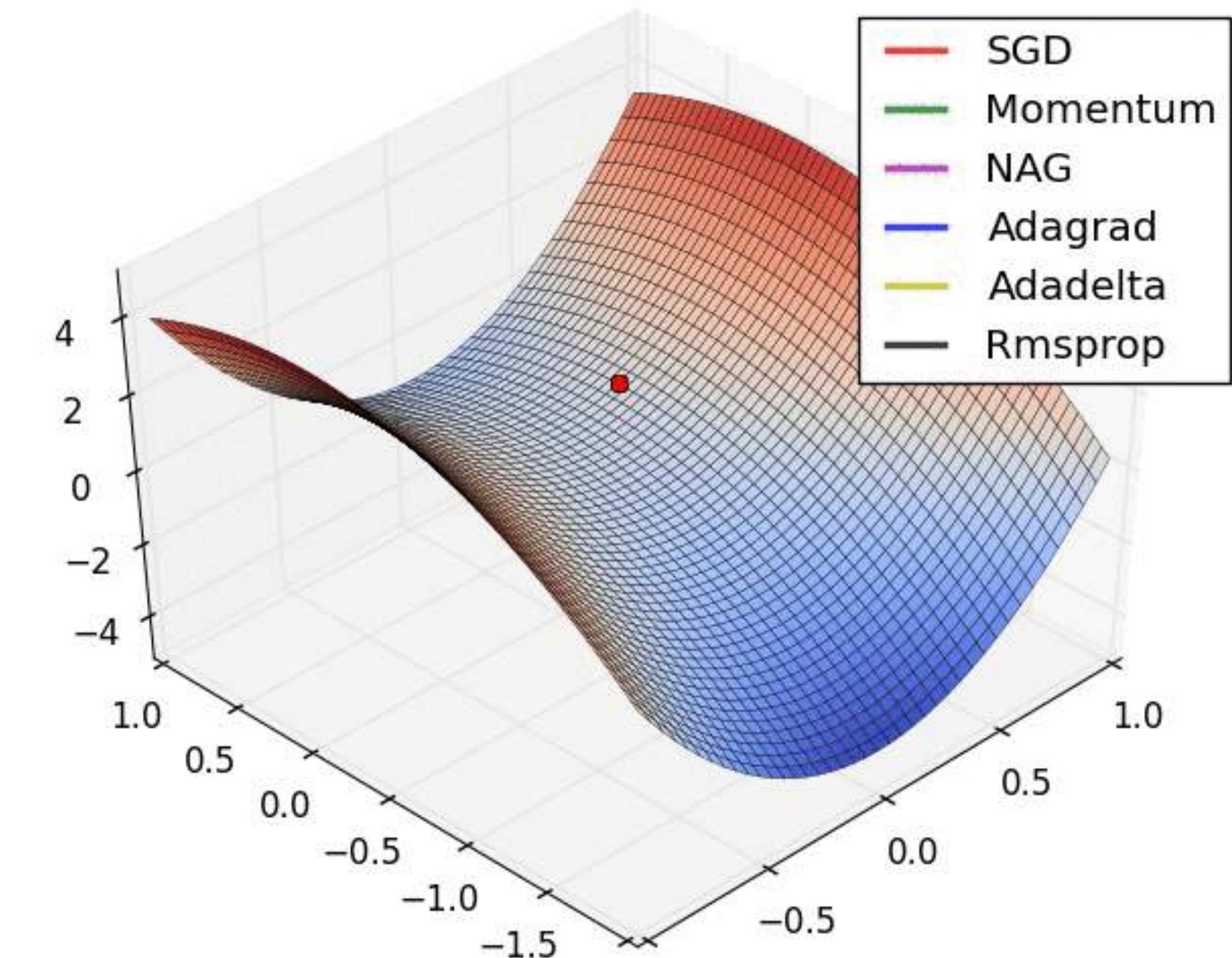
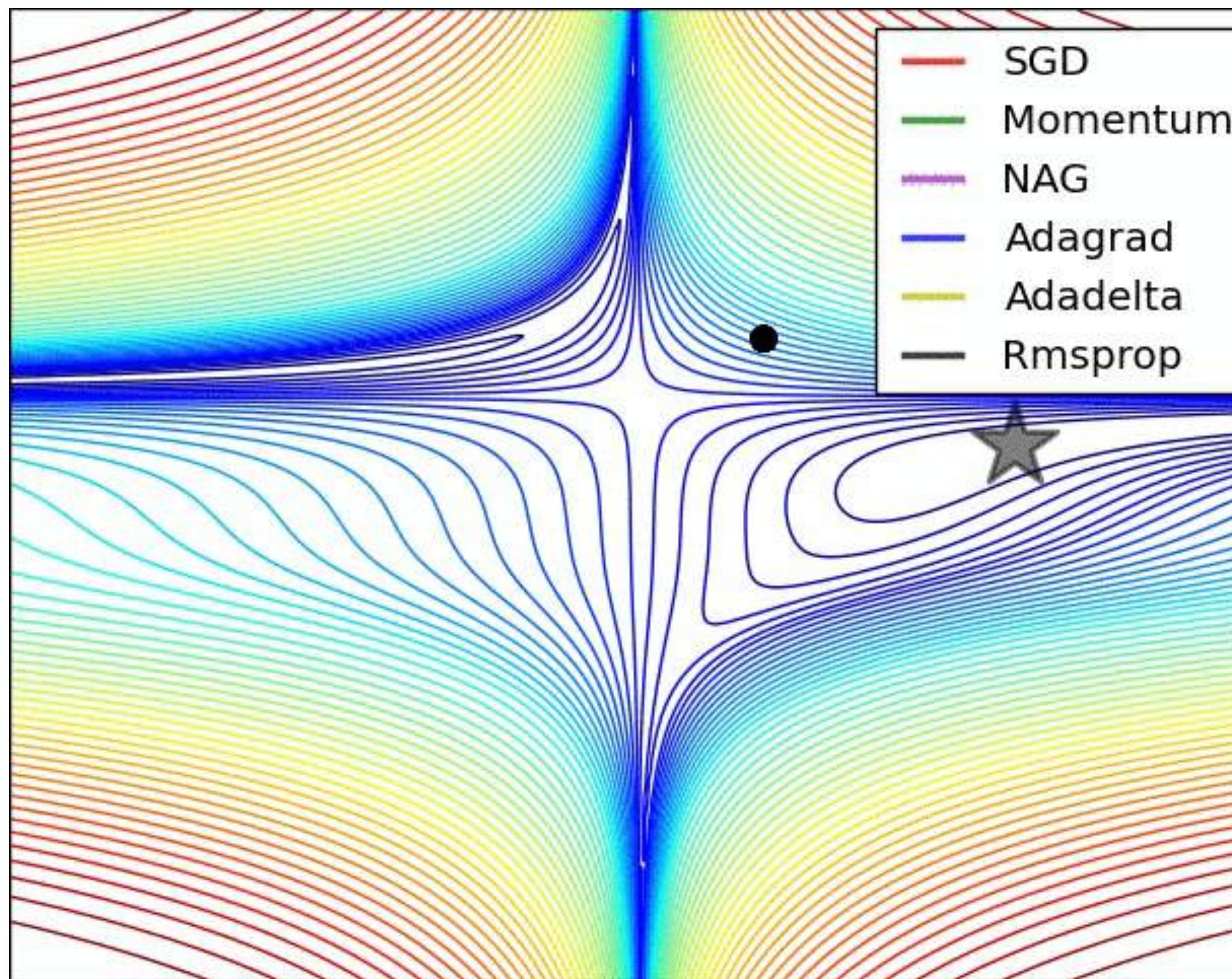


Figure 3: Nesterov update (Source: G. Hinton's lecture 6c)

# Other Optimisers

- **Adagrad** - Good for Sparse data. It adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features.
- **Adadelta** - Extension of Adagrad that reduces its aggressive, monotonically decreasing learning rate.
- **Adam** - Adaptive Moment Estimation is a method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum.
- **RMSProp** - Similar to adadelta,
- **AdaMax** - It is a variant of Adam based on the infinity norm.
- **Nadam** - Nesterov accelerated gradient (NAG) is superior to vanilla momentum. Nadam (Nesterov-accelerated Adaptive Moment Estimation) thus combines Adam and NAG. In order to incorporate NAG into Adam, we need to modify its momentum term
- **AMSGrad** - AMSGrad is an extension to the Adam version of gradient descent that attempts to improve the convergence properties of the algorithm, avoiding large abrupt changes in the learning rate for each input variable

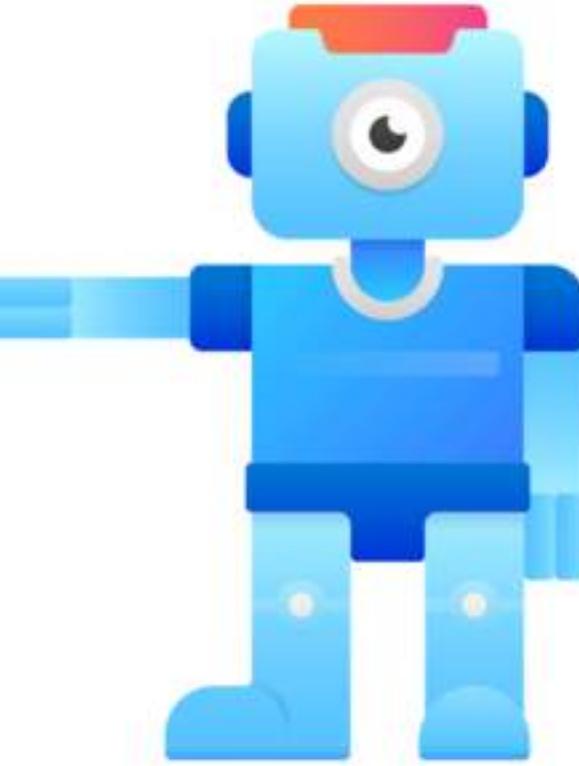
# Visual Comparison of some Optimisers



<http://louistiao.me/notes/visualizing-and-animating-optimization-algorithms-with-matplotlib/>

# Learning Rate Schedules

- A preset list of learning rates used for each epoch.
- Progressively reduce over time.
- We use LR Schedules because if our LR is too high, it can overshoot the minimum points (areas of lowest loss).
- Applying a progressively decreasing learning rate allows the network to take smaller steps (when gradients are updated) allowing our network to find the point of lowest loss instead of jumping over it.
- Early in the training process, we can afford to take big steps, however as the decrease in loss slows, it is often better to use smaller learning rates to avoid oscillations.
- Learning rate schedules are simply to implement with our Deep learning libraries (PyTorch or Keras/ Tensorflow) as they incorporate a decay parameter in optimisers that support LR schedules, such as SGD.

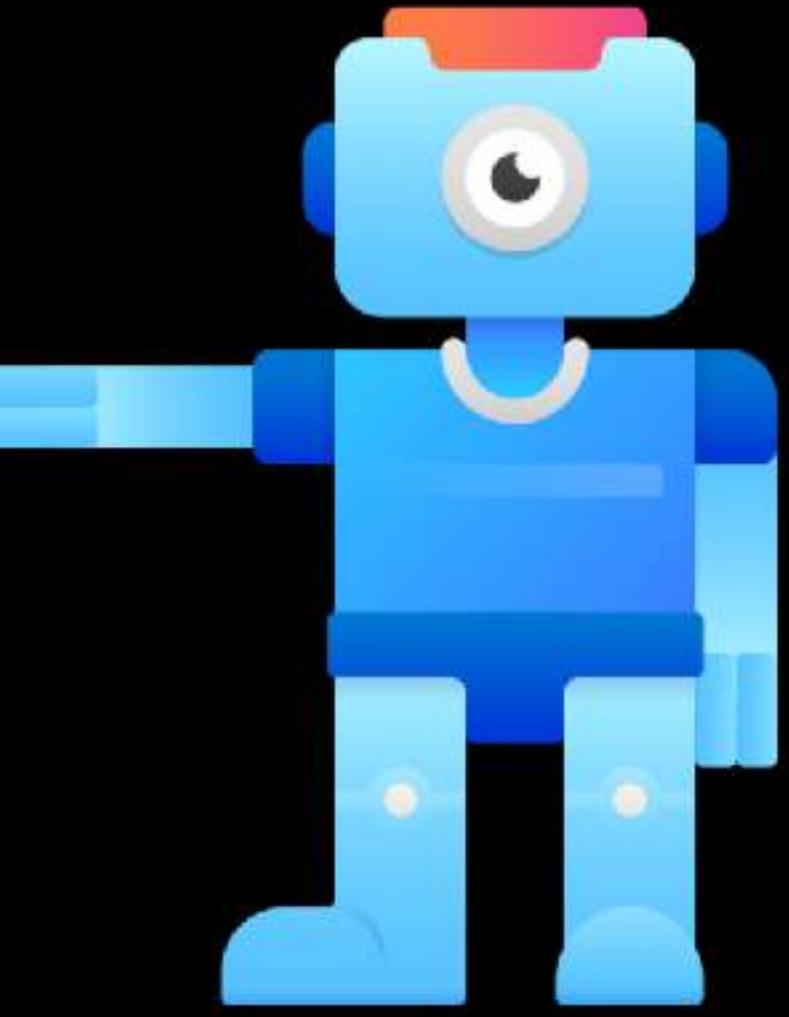


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

A Review of all CNN Theory



# MODERN COMPUTER VISION

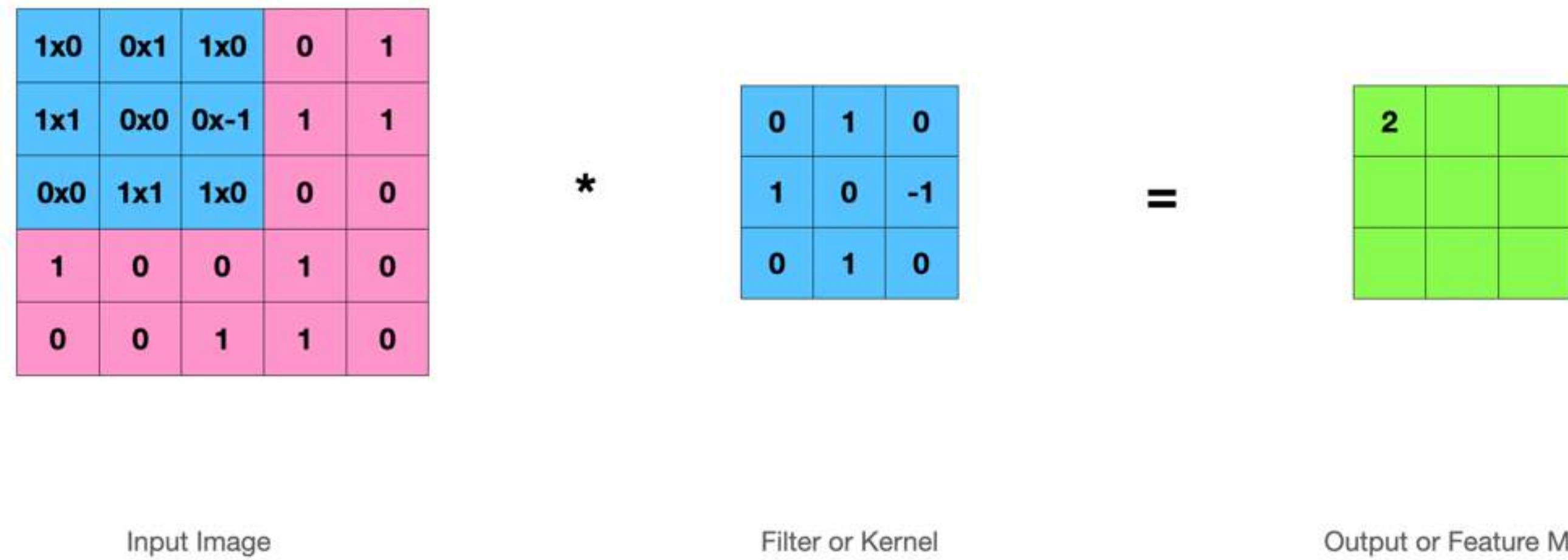
BY RAJEEV RATAN

## Summary of Convolutional Neural Networks (CNNs)

Putting it all together

# Conv Layers

$$(1 \times 0) + (0 \times 1) + (1 \times 0) + (1 \times 1) + (0 \times 0) + (0 \times -1) + (0 \times 0) + (1 \times 1) + (1 \times 0) = 2$$



- Convolution Operations occur when we Convolve our Filters with the input image by sliding it over our image
- This produces an output called a Feature Map
- Feature Maps are now the inputs to the next layer of our CNN

# Stride, Padding and Kernel Size

$$\text{Feature Map Size} = n - f + 1 = m$$

$$\text{Feature Map Size} = 7 - 3 + 1 \equiv 5$$

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \quad * \quad
 \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 0 & -1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \quad = \quad
 \begin{array}{|c|c|c|c|c|} \hline 2 & 1 & -1 & 2 & 2 \\ \hline -1 & 1 & 3 & 2 & 1 \\ \hline 2 & 1 & 1 & 1 & 2 \\ \hline 1 & 1 & 1 & 0 & 2 \\ \hline 2 & 0 & 2 & 3 & 1 \\ \hline \end{array}$$

- We use Stride, Padding and Kernel size to control the output size of our Feature Map

$$\bullet \quad (n \times n) * (f \times f) = (\frac{n+2p-f}{s} + 1) \times (\frac{n+2p-f}{s} + 1)$$

# Activation Layer ReLU - Adds Non-Linearity to our Network

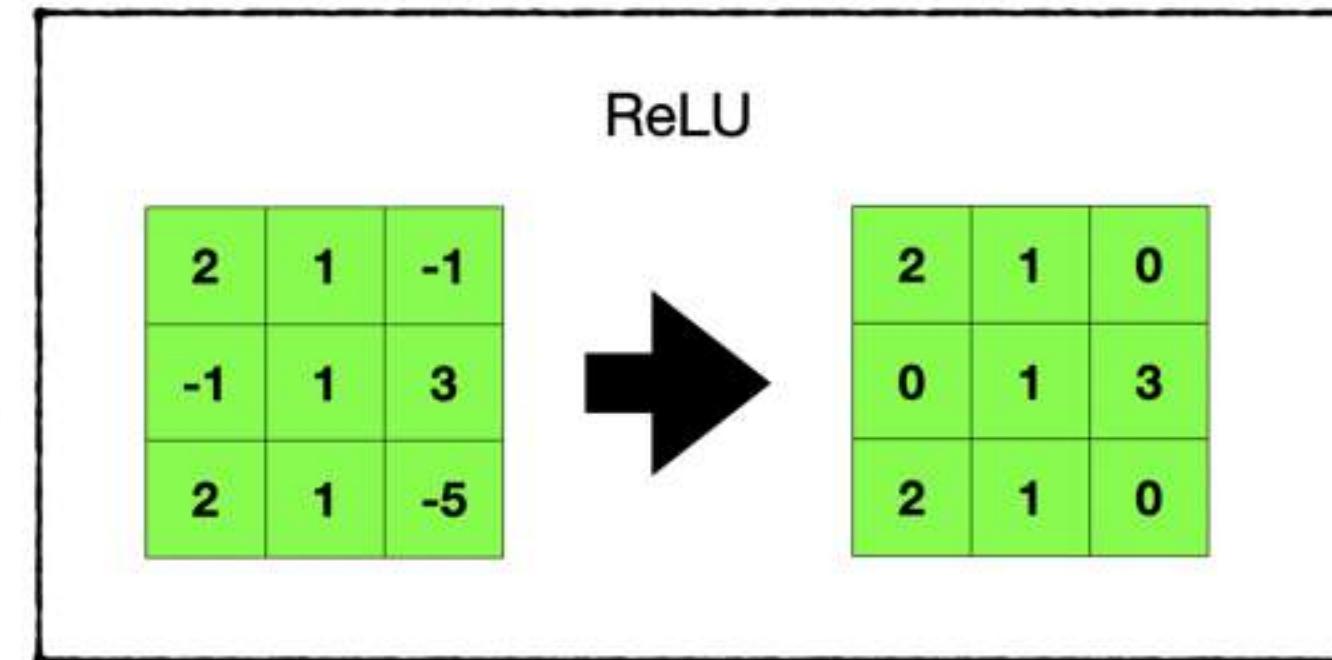
1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

0	1	0
1	0	-1
0	1	0

\*

=



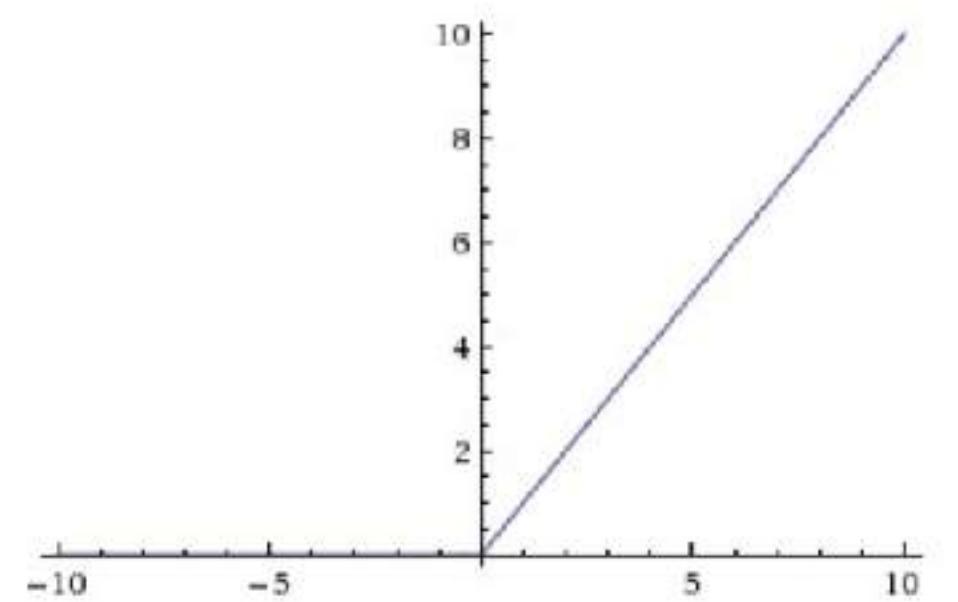
Filter or Kernel

Output or Feature Map

Rectified Feature Map

One Layer

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



# Max Pooling - Reduce Dimensionality

4	123	1	34
56	99	222	253
45	122	165	12
21	187	133	124

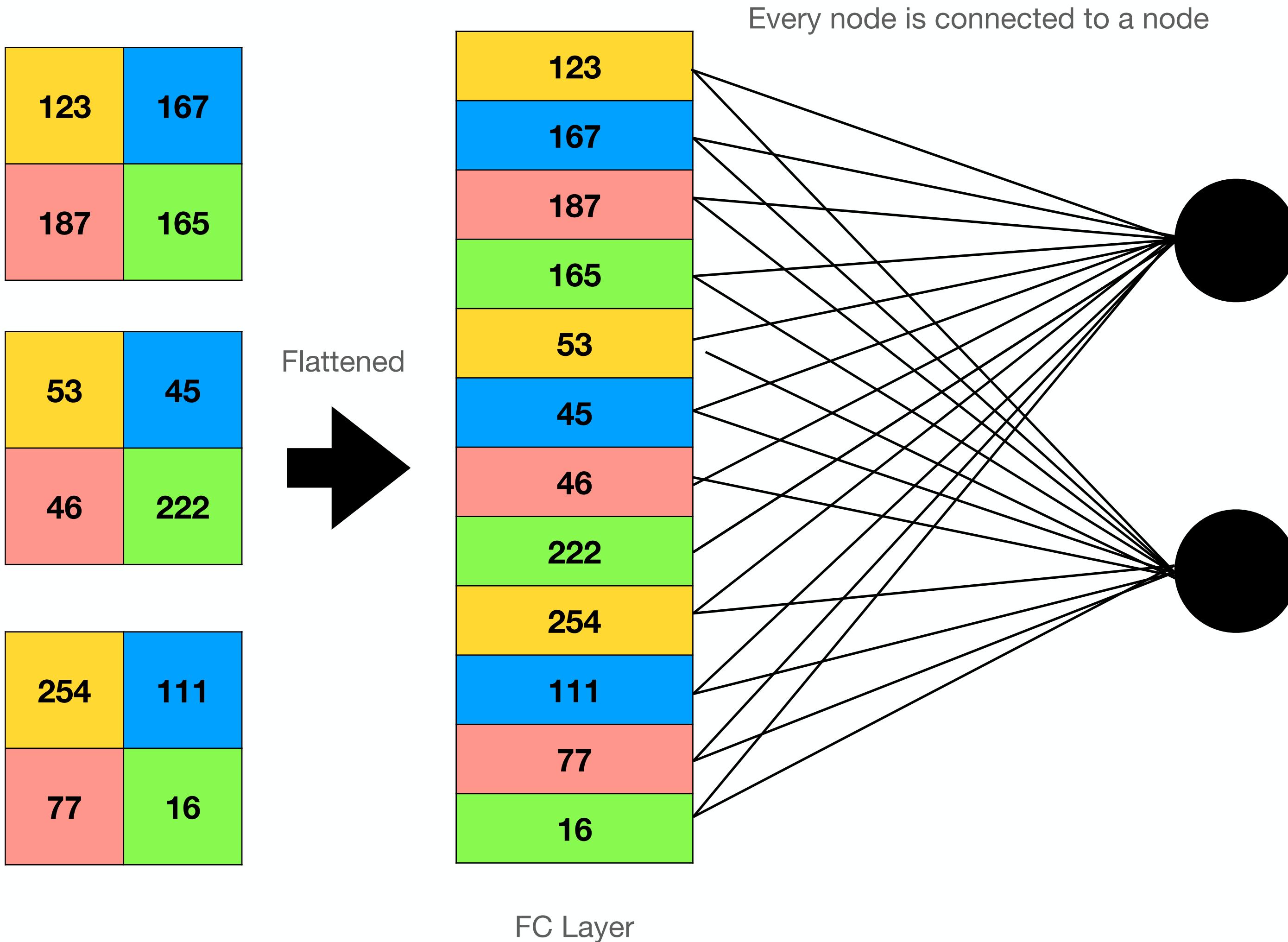
**MaxPool Operation**



**Stride = 2**  
**Kernel = 2x2**

123	167
187	165

# Fully Connected Layer - Max Pool Layer is Flattened



# Softmax Layer

Logits Scores

2.0

1.0

0.1

Probabilities

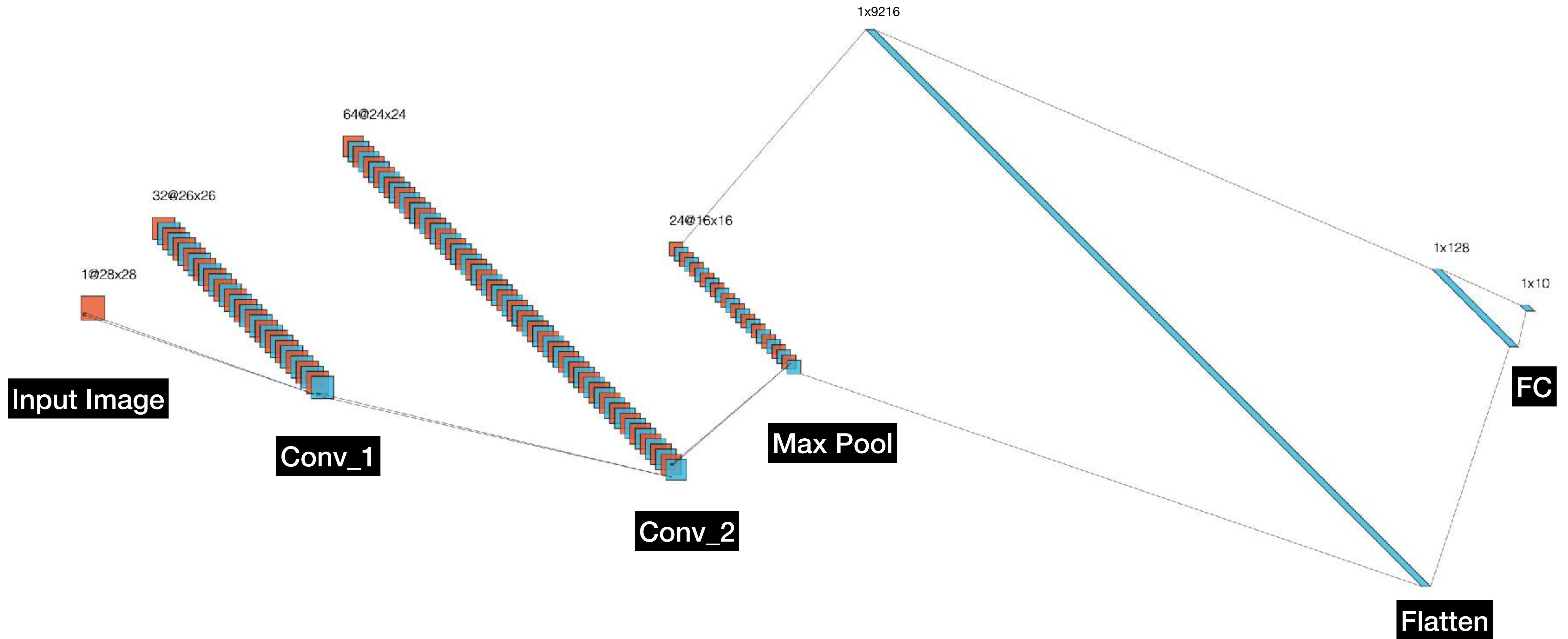
0.7

0.2

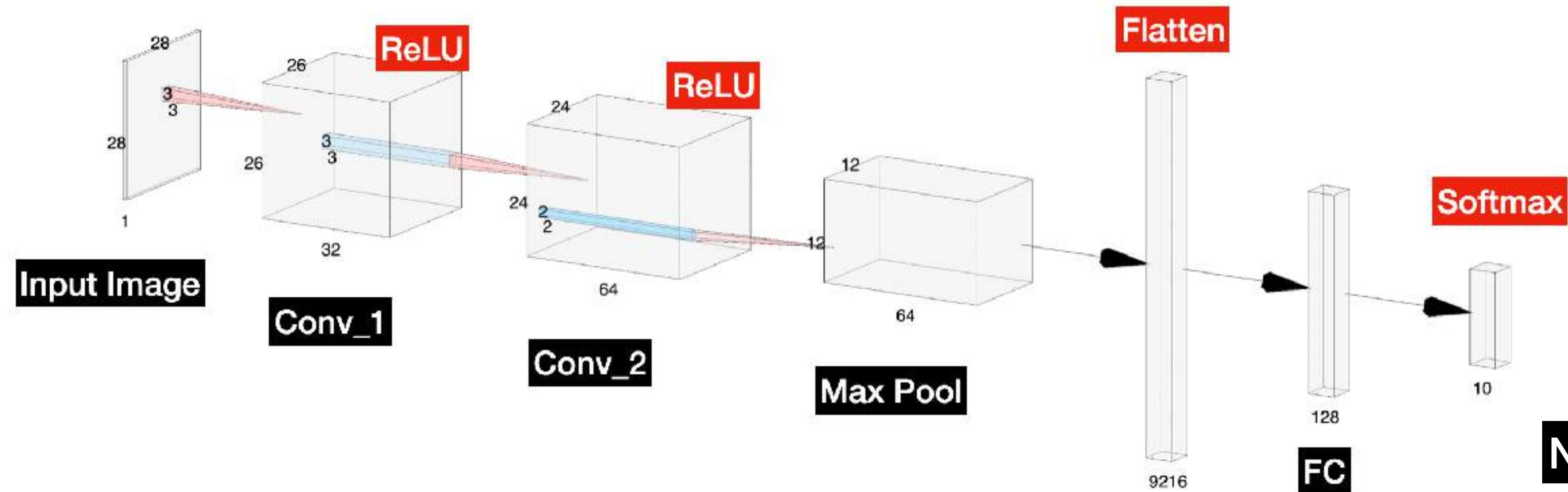
0.1

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

# Our Basic CNN



# Parameters in our Basic CNN



Layer	Parameters
Conv_1 + ReLU	320
Conv_2 + ReLU	18494
Max Pool	0
Flatten	0
FC_1	1,179,776
FC_2 (Output)	1,290
Total	1,199,882

**Conv\_1**

$$((\text{Height} \times \text{Width} \times \text{Depth}) + \text{bias}) \times N_f$$

$$((3 \times 3 \times 1) + 1) \times 32 = 320$$

**Conv\_2**

$$((\text{Height} \times \text{Width} \times \text{Depth}) + \text{bias}) \times N_f$$

$$((3 \times 3 \times 32) + 1) \times 64 = 18,494$$

**No Trainable Parameters**

- Max Pool
- Flatten
- ReLU

**Fully Connected/Dense**

$$(\text{Length} + \text{bias}) \times N_{\text{nodes}}$$

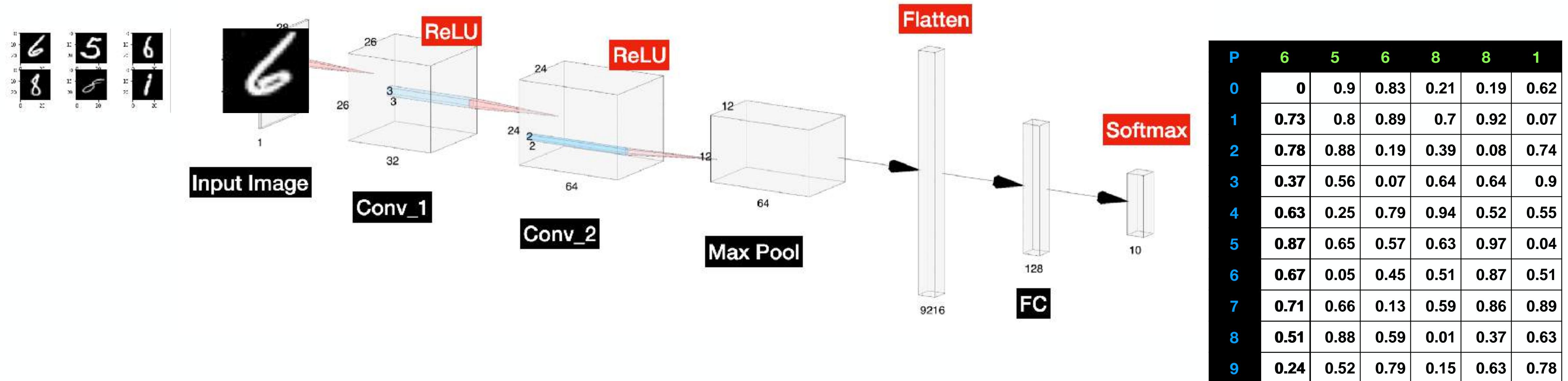
$$(9216 + 1) \times 128 = 1,179,776$$

**Final Output (FC/Dense)**

$$(\text{Length} + \text{bias}) \times N_{\text{nodes}}$$

$$(128 + 1) \times 10 = 1,290$$

# The Training Process



# Overview on Training

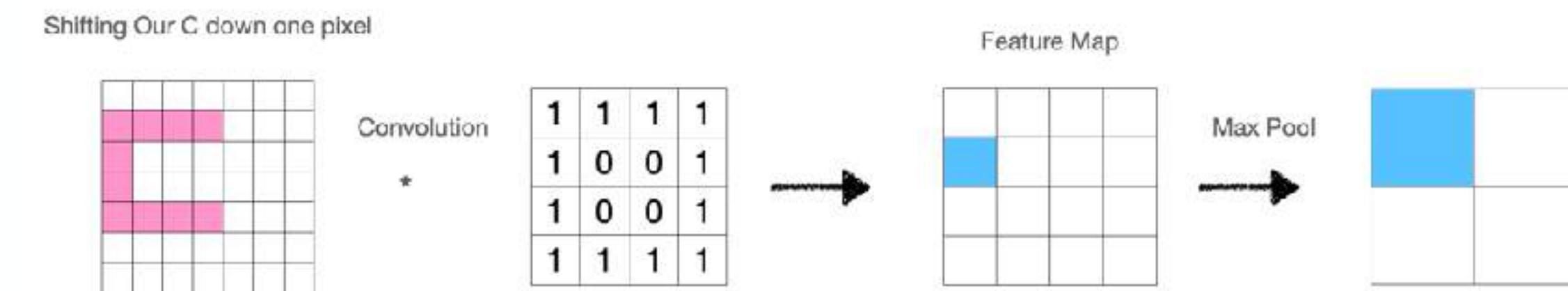
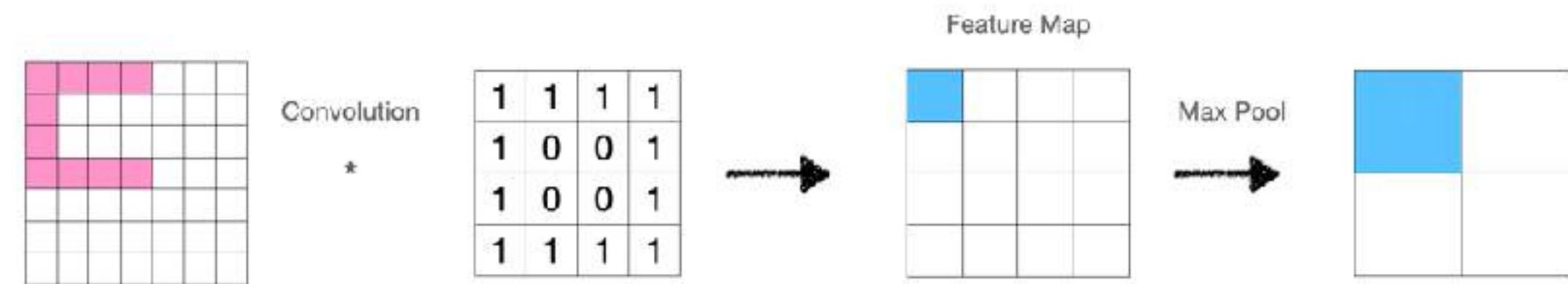
- CNN Model is **designed and defined**
- Weights are **initialised with random values**
- Batches of Images (typically 8 to 256) are **forward propagated** through our CNN Model
- Using **Back Propagation** with **Mini-Batch Gradient Descent** we update the individual weights (right to left)
- Using we update all our weights so that we have a lower loss
- When the entire dataset of images is forward propagated, we've completed an **Epoch**
- We train for **5 to 50** Epochs and stop when Loss stops decreasing

# Batches, Mini-Batches, Iterations & Epochs

- We can feed images one at a time, however using mini-batches is better

# Advantages of Convolution Neural Networks

- **Invariance** - Remember our Max Pool Example
- **Parameter sharing** - where a single filter can be used all parts of an image
- **Sparsity of connections** - As we saw, fully connected layers in a typical Neural Network result in a weight matrix with large number of parameters.



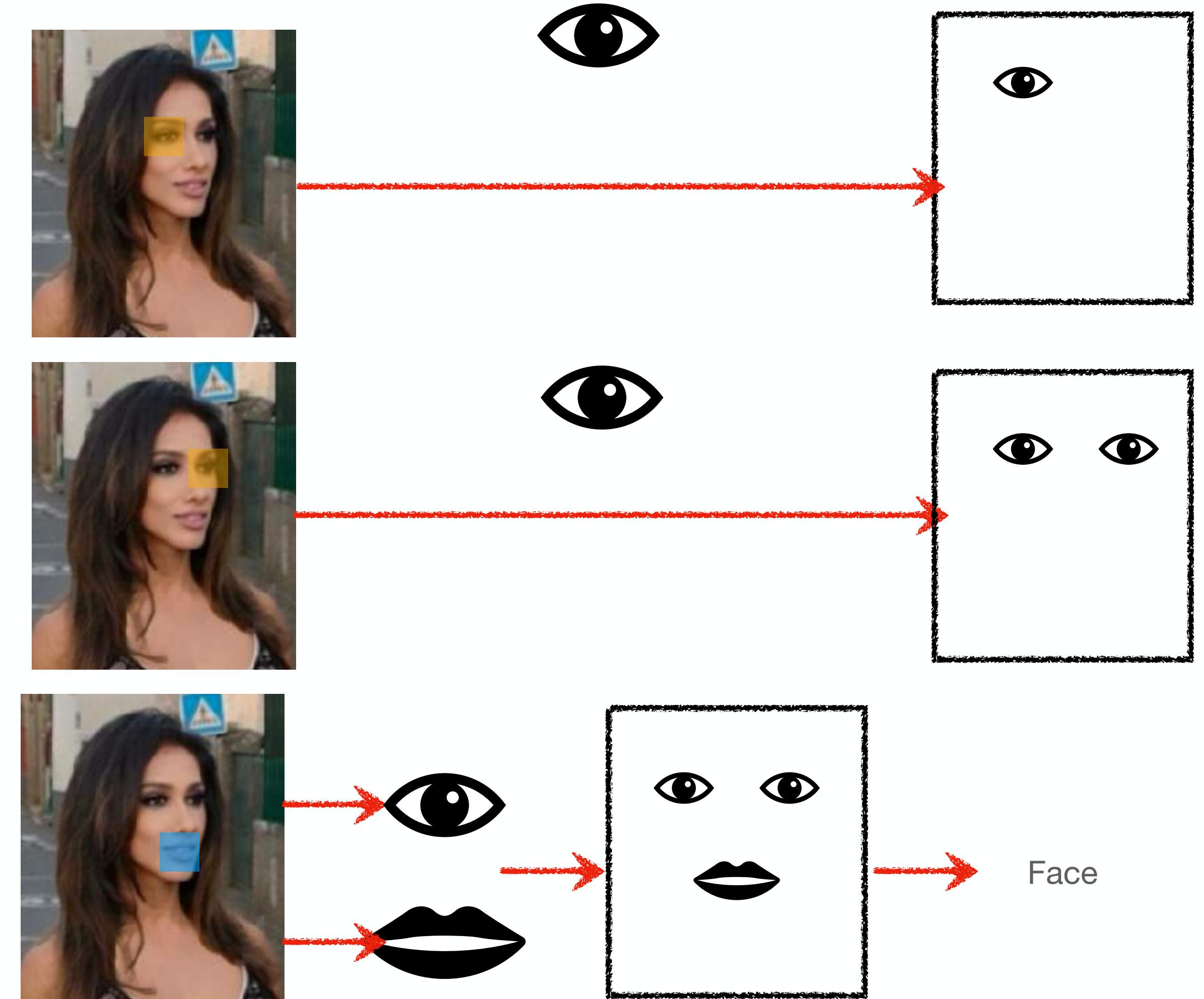
---


$$\begin{matrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{matrix} * \begin{matrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{matrix}$$


---

# Convolution Neural Networks Assumptions

- Low-level features are local
- Features are translational invariant
- High-level features are made up of low-level features



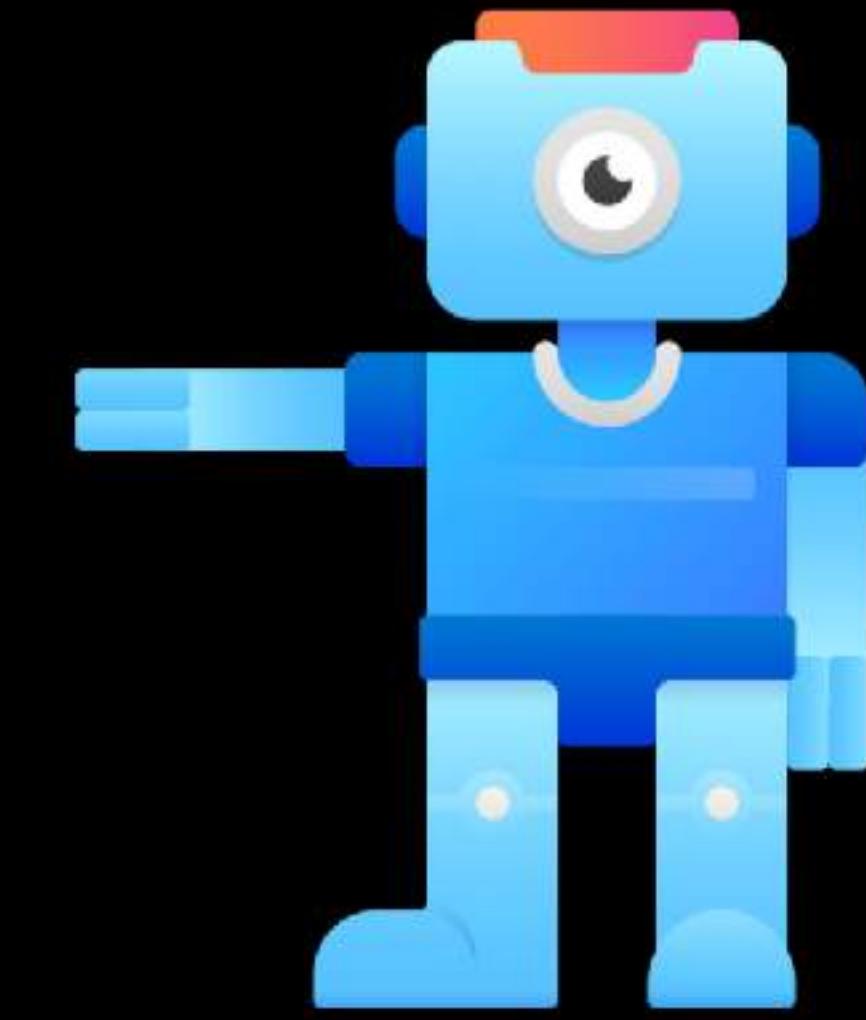


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**History of Deep Learning and AI**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Deep Learning History

**Why now is the best time to be a Deep Learning Practitioner**

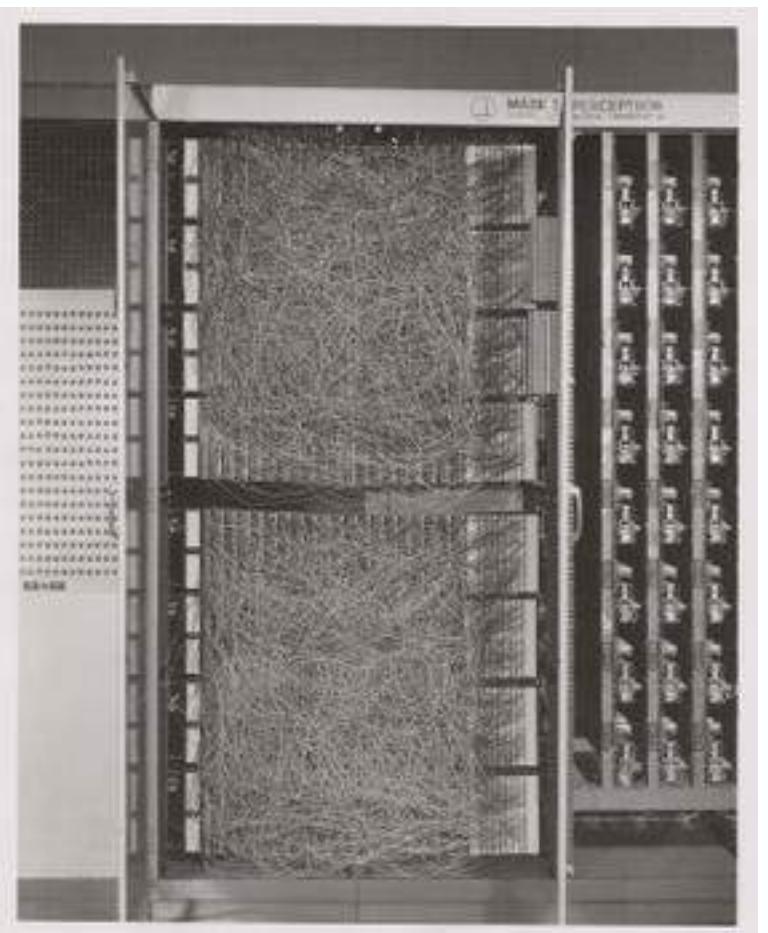
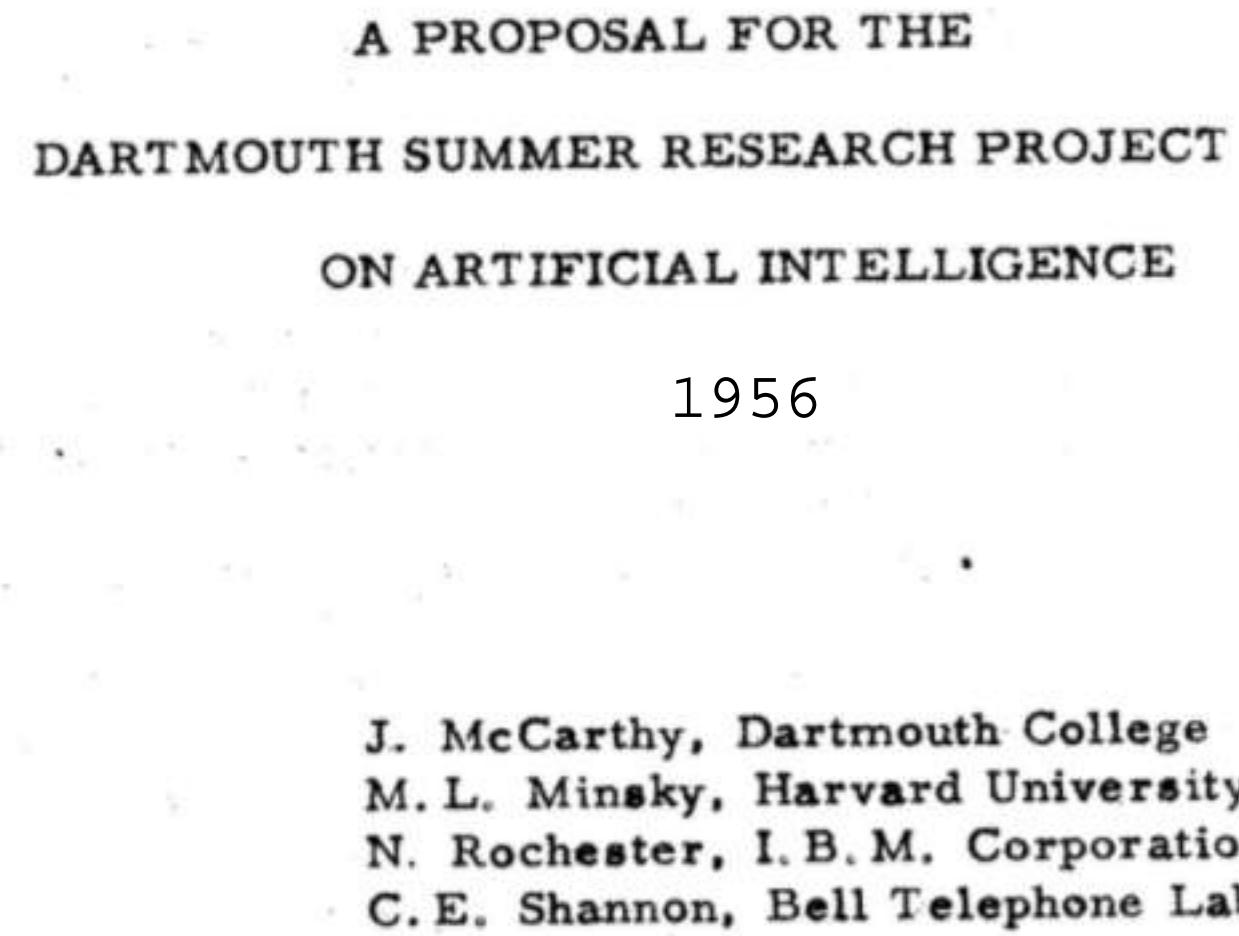
# Now is the best time to be a Deep Learning Practitioner

- We're in the Golden Age of AI
- Fast Graphics Processing Unit (**GPU**) hardware is relatively cheap and readily available
- Cheap even free cloud GPUs are available
- Deep Learning Libraries are mature and relatively easy to use
- But it wasn't always this way....



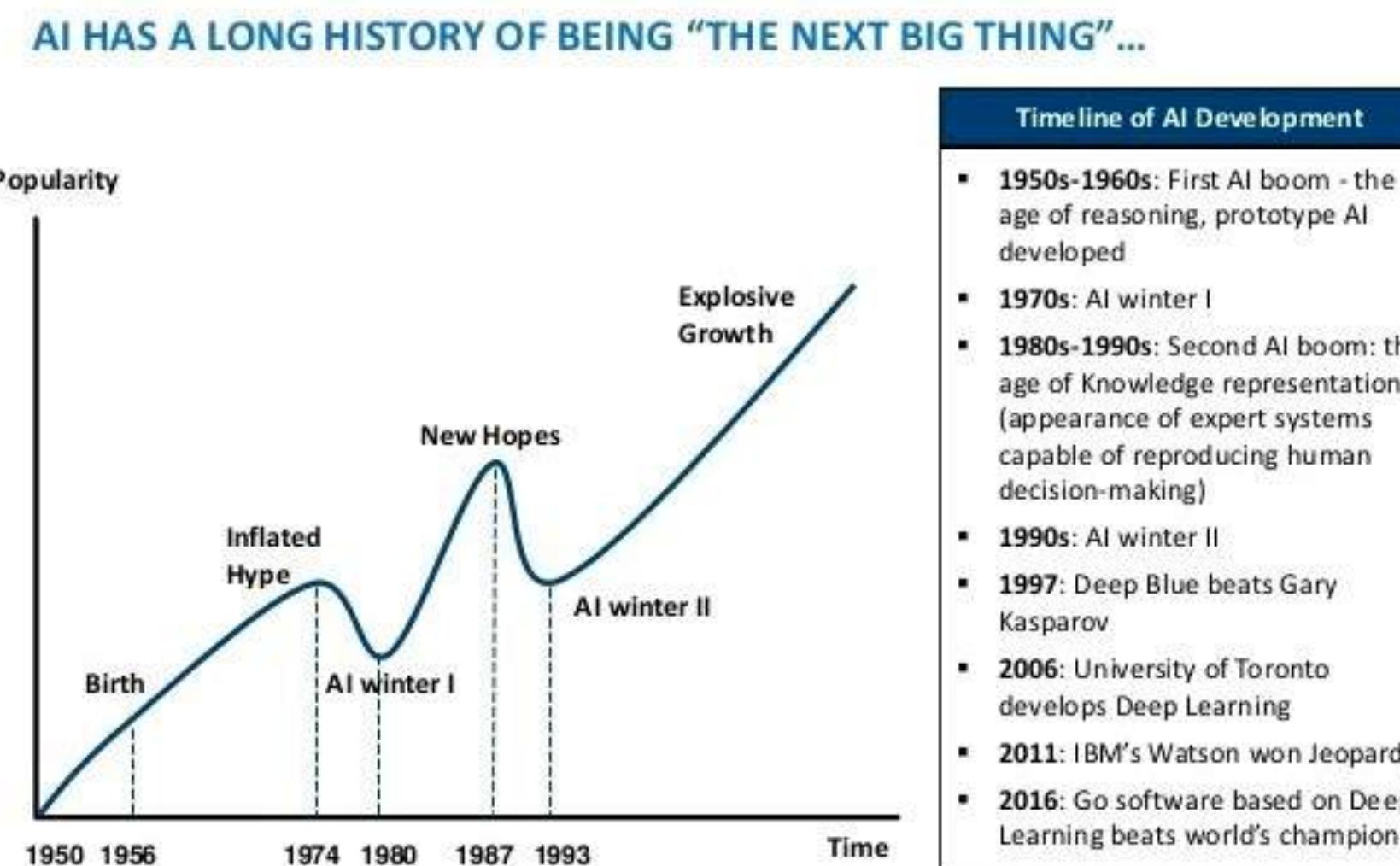
# History of Deep Learning

- The concept of using neuron like algorithms in Computer Science was first introduced in 1943 by Warren McCulloch and Walter Pitts.
- Progress was slow until 1960 when Henry J. Kelly developed the basics of a Continuous Back Propagation Model.
- Things progressed nicely as Statistical Modelling and early Machine Learning algorithms were introduced
- However, the execution of many of these algorithms were slow, inefficient and error prone.



# History of Deep Learning - AI Winter 1

- In the 1970s, the AI promises of the 1960s were unfilled and it lost favour with researchers at the time.
- Funding was slashed and many claimed **AI was just hype.**



# History of Deep Learning - Neocognitron

- Research still continued and in 1979, Kunihiko Fukushima designed the first Convolutional Neural Network, called **Neocognitron**.

Published: April 1980

Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position

[Kunihiko Fukushima](#)

[Biological Cybernetics](#) 36, 193–202(1980) | [Cite this article](#)

5885 Accesses | 1880 Citations | 33 Altmetric | [Metrics](#)

<https://link.springer.com/article/10.1007/BF00344251>

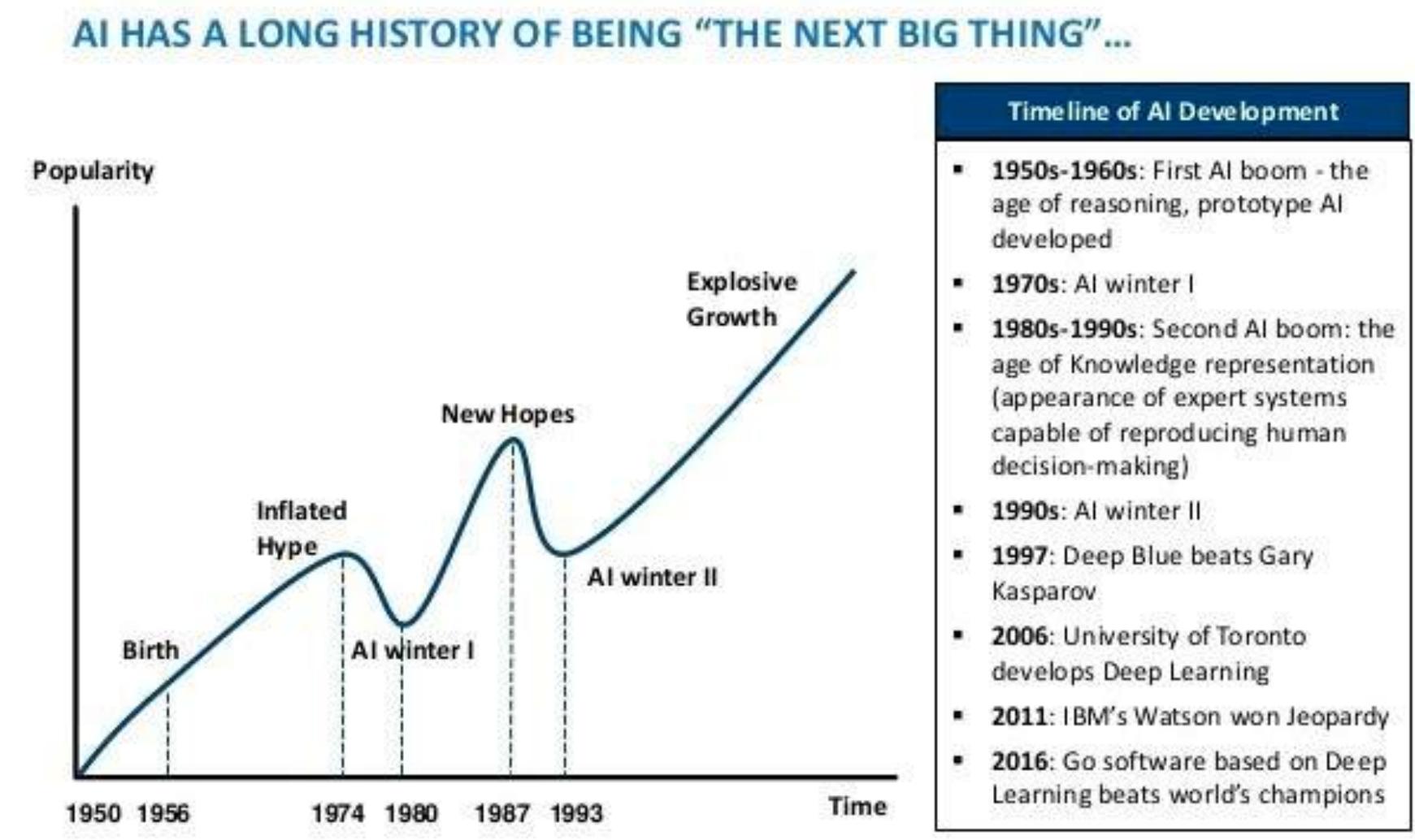
# Back Propagation in FORTRAN

- Though it was implemented in the 1970s, it was only in 1985 it was applied in Neural Networks
- Things picked up at this point, however, it led to over promising and under delivering thus leading to the second AI winter.

```

1 module kernel_m
2   interface zero
3     attributes(global) subroutine zero(a) &
4       bind(C,name='zero')
5     use iso_c_binding
6     real(c_float) :: a(*)
7   end subroutine zero
8 end interface
9 end module kernel_m
10
11 program fCallingC
12   use cudafor
13   use kernel_m
14   integer, parameter :: n = 4
15   real, device :: a_d(n)
16   real :: a(n)
17
18   a_d = 1.0
19   call zero<<<1,n>>>(a_d)
20   a = a_d
21   write(*,*) a
22 end program fCallingC

```



# GPUs and Deep Learning a Match Made in Heaven

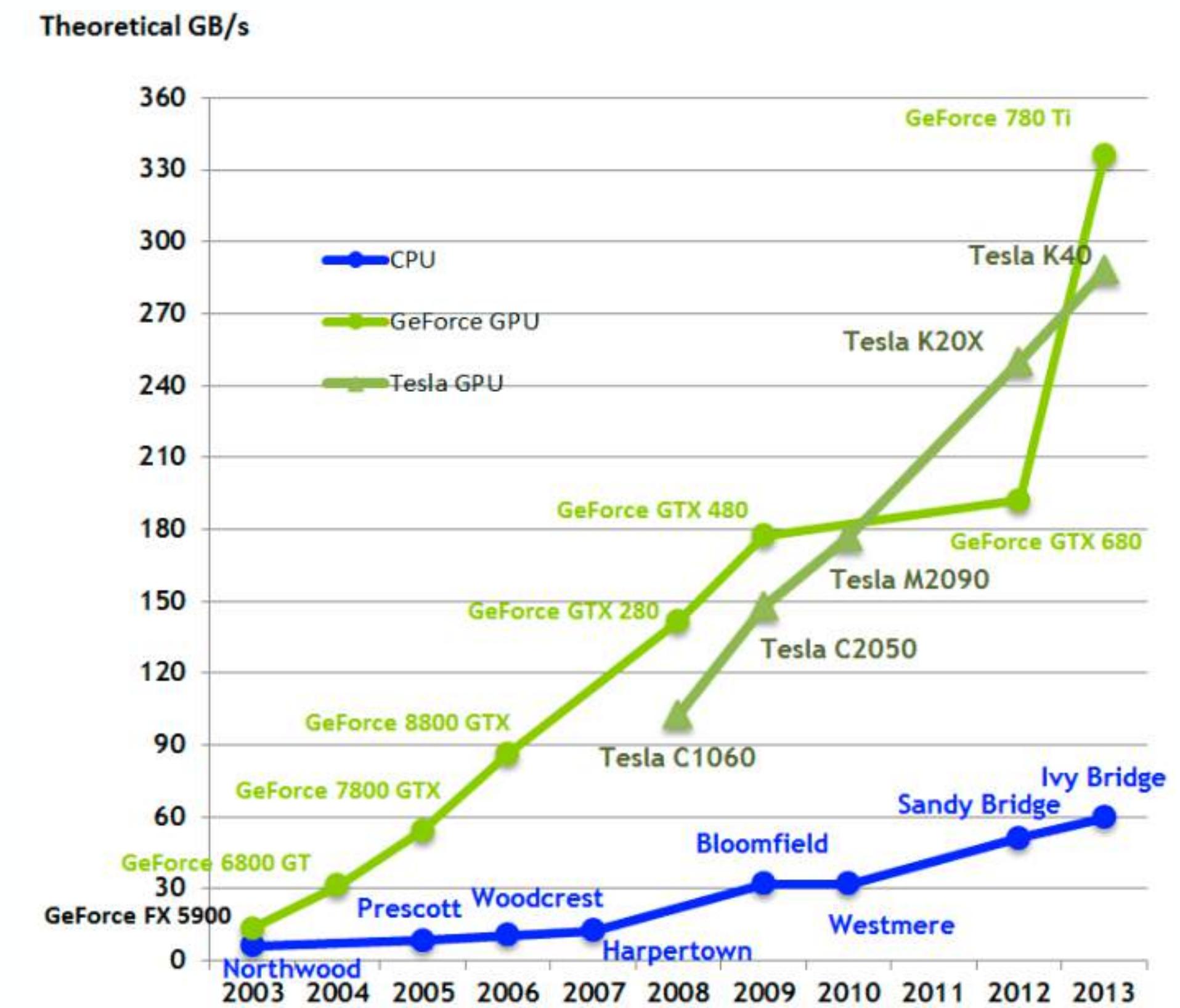
- In the late 1990s 3D gaming necessitated GPUs which early Deep Learning methods also utilised.
- GPUs are suited for both rendering graphics and Deep Learning as they provide specialised processors that perform floating point operations with dedicated memory (high bandwidth).
- Think of CPUs as a 4-door sedan, good all round vehicle
- Think of GPUs as huge Truck (good for one thing, not good at others)

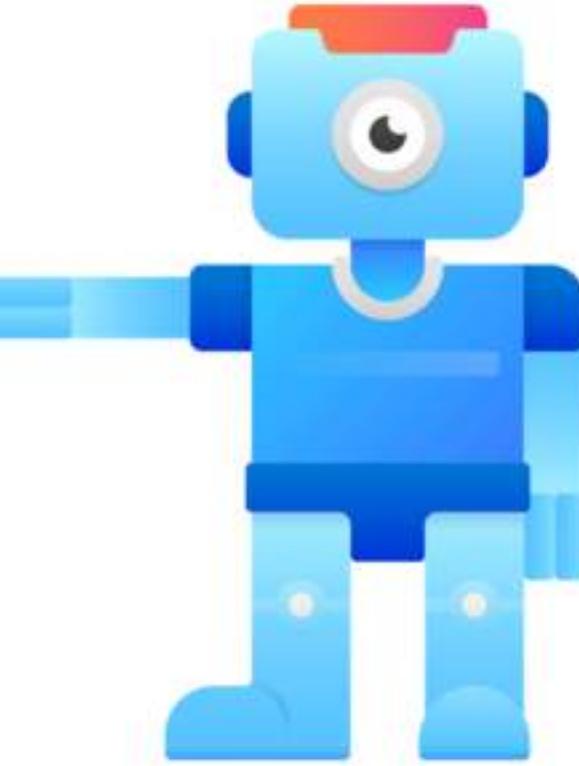


3DFX Voodoo

# GPUs have come a long way

- GPU speed continues to increase at a faster pace than CPU speed
- Keeping up pace with this was the development of Deep Learning Libraries...





# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

Deep Learning Libraries

# Deep Learning Libraries

Introduction to Deep Learning Libraries or APIs



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Deep Learning Computations

- Implementing deep learning on computers is difficult
- It requires a lot of moving parts
- Massive amounts of calculations
- Takes a lot of time and can be prone to breaking
- Complexity in implementing special features

**Don't Try Deep  
Learning At Home!**

**Actually You Should**

**Don't Try Making Your  
Own Deep Learning  
Low Level Code!**

**Because it's been done**

# Deep Learning Libraries to the Rescue!

- Google and Facebook have developed Open Source Deep Learning libraries.

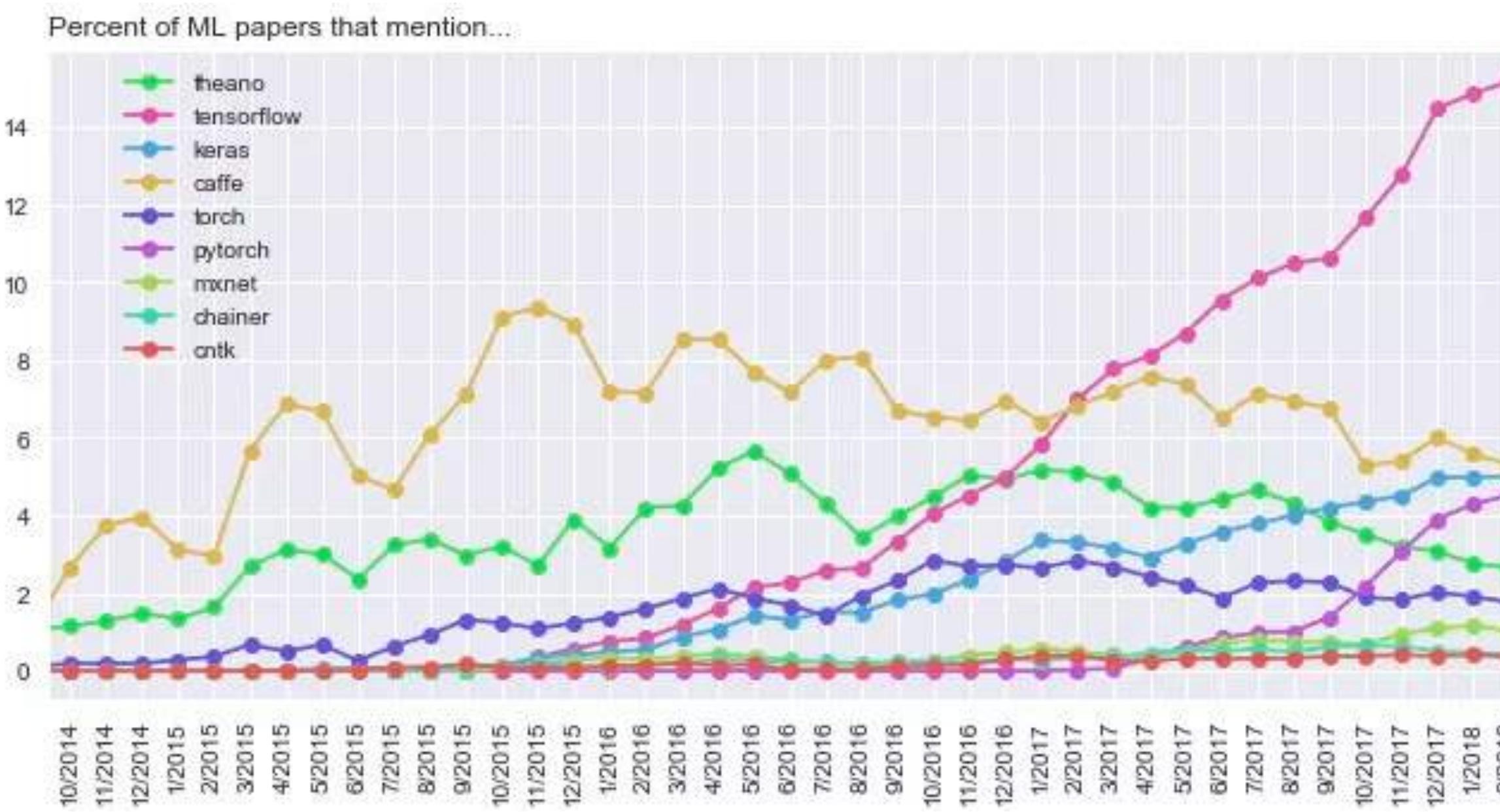


- These two brilliant libraries allow us to create Neural Networks in Python very easily.
- They are now quite mature and offer excellent speed, reliability, stability and support.

# Deep Learning Libraries

**PyTorch, TensorFlow, Keras and others!**

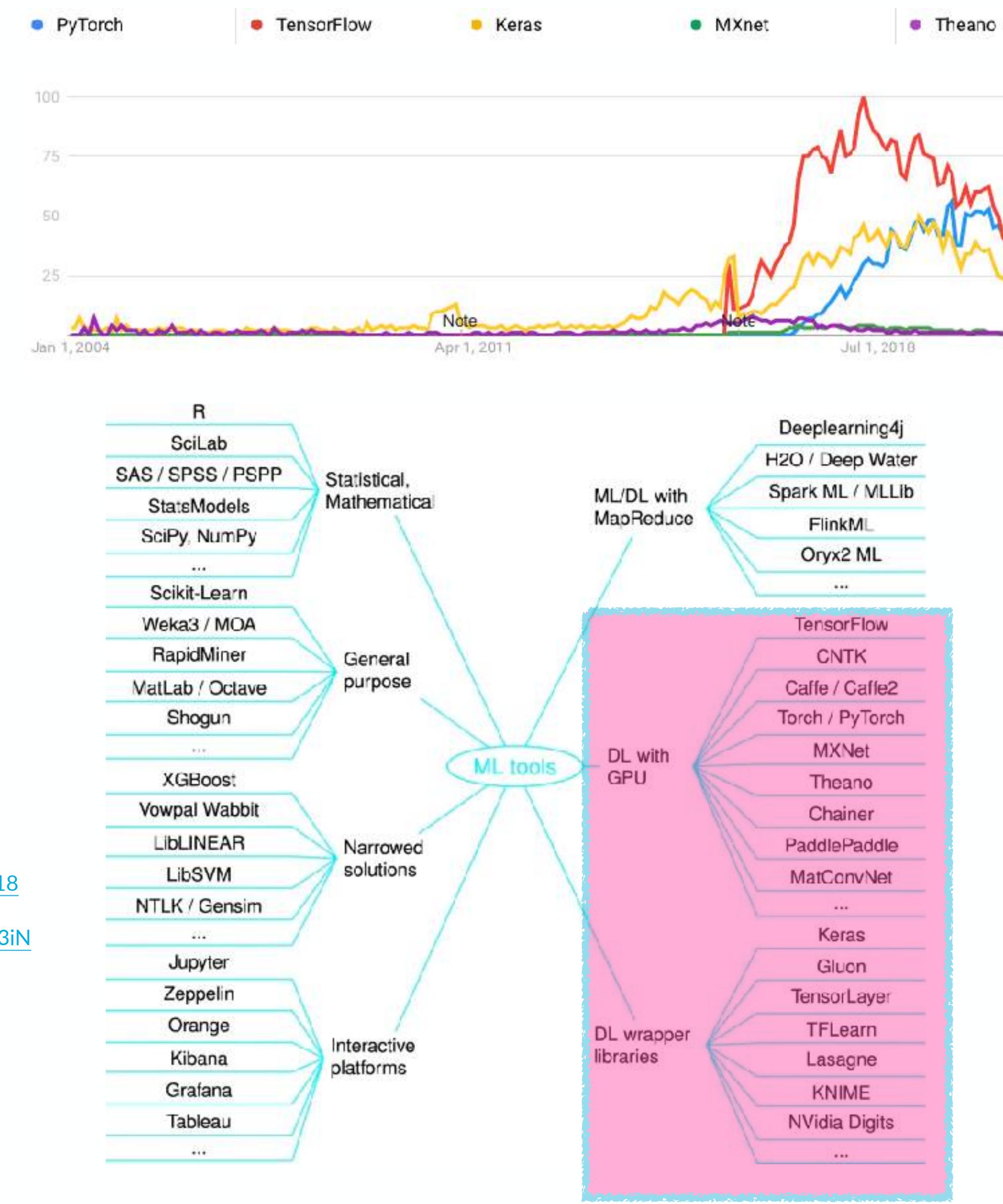
- While PyTorch and TensorFlow/Keras have come out on top, many others vied for this title such as Theano, Caffe, MXNet, PaddlePaddle



Andrej Karpathy

(@karpathy) [10 marzo 2018](https://twitter.com/YOYAvc33IN)

[pic.twitter.com/YOYAvc33IN](https://pic.twitter.com/YOYAvc33IN)



# TensorFlow

## Developed by Google



- Free and open-source Deep Learning Library
- Developed internally by Google Brain and open-sourced in 2015
- Written in C++ and CUDA (*NVIDIA's Compute Unified Device Architecture API for programming on GPUs*)
- Primarily used in Python but APIs/wrappers exist for several other languages (C++, Go, Java, JavaScript, Swift).
- TensorFlow 2.0 was released in 2019

# PyTorch

## Developed by Facebook



- Free and open-source Deep Learning Library
- Developed by Facebook's AI Research Team
- It is a C-based tensor Library written with CUDA capabilities
- Primarily meant for use in Python, though C++ is well supported
- PyTorch is Python based scientific computing package aimed at:
  1. A replacement for NumPy using the power of GPUs
  2. A Deep Learning research platform providing both flexibility and speed

# Keras

Developed by François Chollet



- Free and open-source Deep Learning API Interface for Python
- Initially supported several Deep Learning Backends (TensorFlow & Theano), as of Version 2.3+, its sole focus was TensorFlow.
- Written in Python...for Python
- Primarily meant to make TensorFlow more accessible by abstracting some of low level work
- It also contained several implementations of NN layers and tools
- As for 2019, Keras is now bundled with TensorFlow2.0

# Summary

- TensorFlow and PyTorch are fairly low-level
- Keras is simpler, quicker to implement and less prone to user errors
- TensorFlow has wider adoption in industry
- PyTorch due to flexibility, has gained more popularity in research and academia, taking over from Caffe



# Interested in Comparing Deep Learning Software?

[https://en.wikipedia.org/wiki/Comparison\\_of\\_deep-learning\\_software](https://en.wikipedia.org/wiki/Comparison_of_deep-learning_software)

## Comparison of deep-learning software

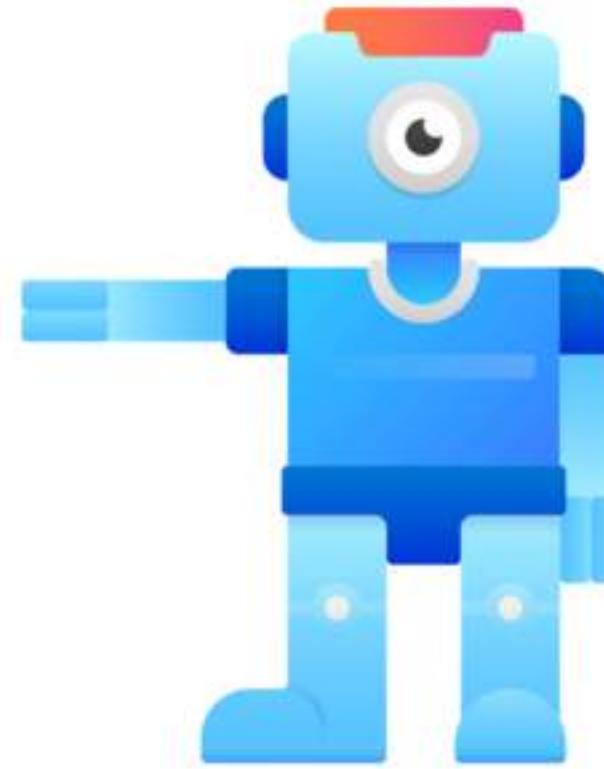
From Wikipedia, the free encyclopedia

The following table compares notable software frameworks, libraries and computer programs for deep learning.

## Contents [hide]

[Deep-learning software by name](#) [edit]

Software	Creator	Initial Release	Software license <sup>[a]</sup>	Open source	Platform	Written in	Interface	OpenMP support	OpenCL support
BigDL	Jason Dai (Intel)	2016	Apache 2.0	Yes	Apache Spark	Scala	Scala, Python		
Caffe	Berkeley Vision and Learning Center	2013	BSD	Yes	Linux, macOS, Windows <sup>[2]</sup>	C++	Python, MATLAB, C++	Yes	Under development <sup>[3]</sup>
Chainer	Preferred Networks	2015	BSD	Yes	Linux, macOS	Python	Python	No	No
Deeplearning4j	Skymind engineering team; Deeplearning4j community; originally Adam Gibson	2014	Apache 2.0	Yes	Linux, macOS, Windows, Android (Cross-platform)	C++, Java	Java, Scala, Clojure, Python (Keras), Kotlin	Yes	No <sup>[7]</sup>
Dlib	Davis King	2002	Boost Software License	Yes	Cross-platform	C++	C++, Python	Yes	No
Flux	Mike Iernes	2017	MIT license	Yes	Linux, MacOS, Windows (Cross-platform)	Julia	Julia		
Intel Data Analytics Acceleration Library	Intel	2015	Apache License 2.0	Yes	Linux, macOS, Windows on Intel CPU <sup>[13]</sup>	C++, Python, Java	C++, Python, Java <sup>[13]</sup>	Yes	No
Intel Math Kernel Library	Intel		Proprietary	No	Linux, macOS, Windows on Intel CPU <sup>[14]</sup>		C <sup>[15]</sup>	Yes <sup>[16]</sup>	No
Keras	François Chollet	2015	MIT license	Yes	Linux, macOS, Windows	Python	Python, R	Only if using Theano as backend	Can use Theano, Tensorflow or PlaidML as backends

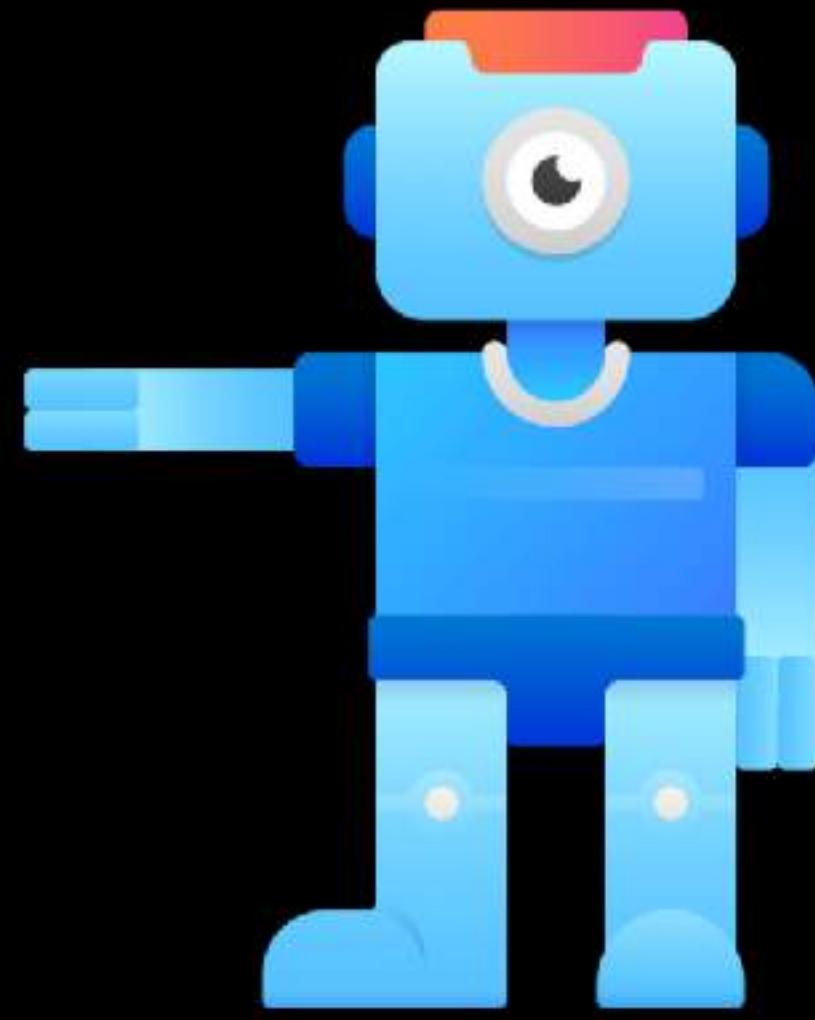


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Building and Training Your First Convolutional Neural Network with PyTorch**

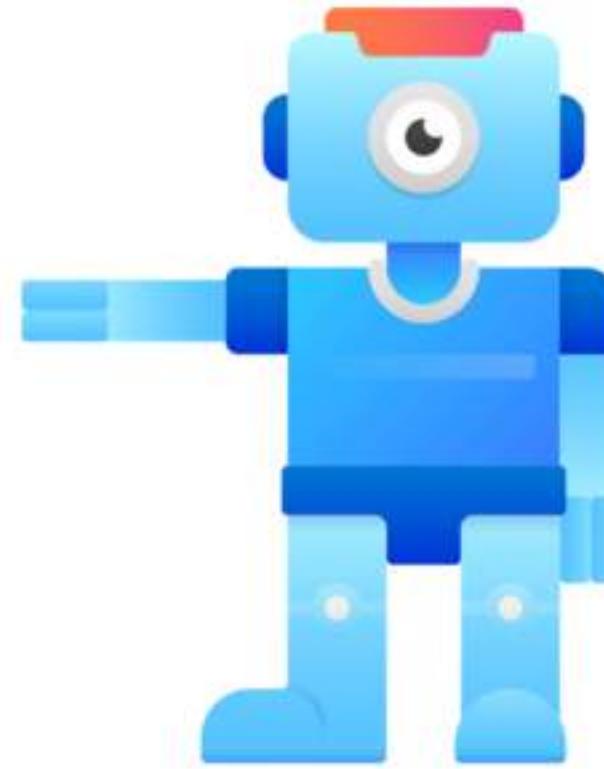


# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Building and Training Your First Convolutional Neural Network with PyTorch

A detailed introduction to using PyTorch

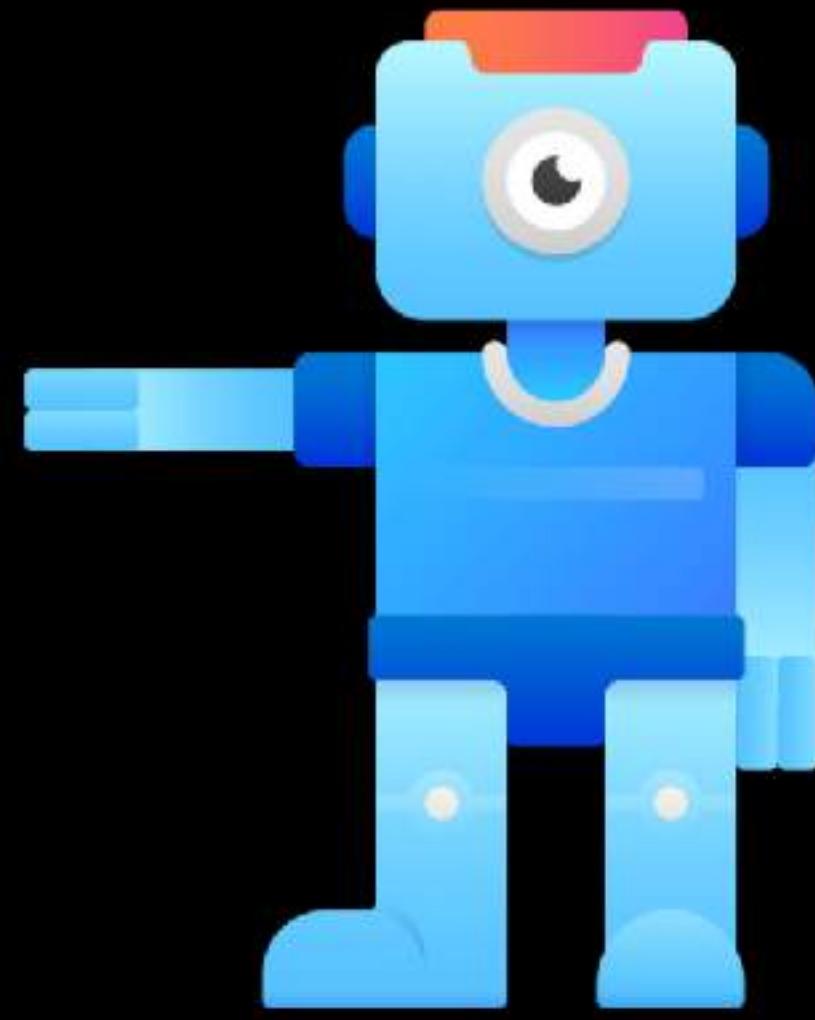


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Building and Training Your First Convolutional Neural Network with Keras in TensorFlow 2.0**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Building and Training Your First Convolutional Neural Network with Keras in TensorFlow 2.0

A detailed introduction to using PyTorch



**MODERN  
COMPUTER  
VISION**

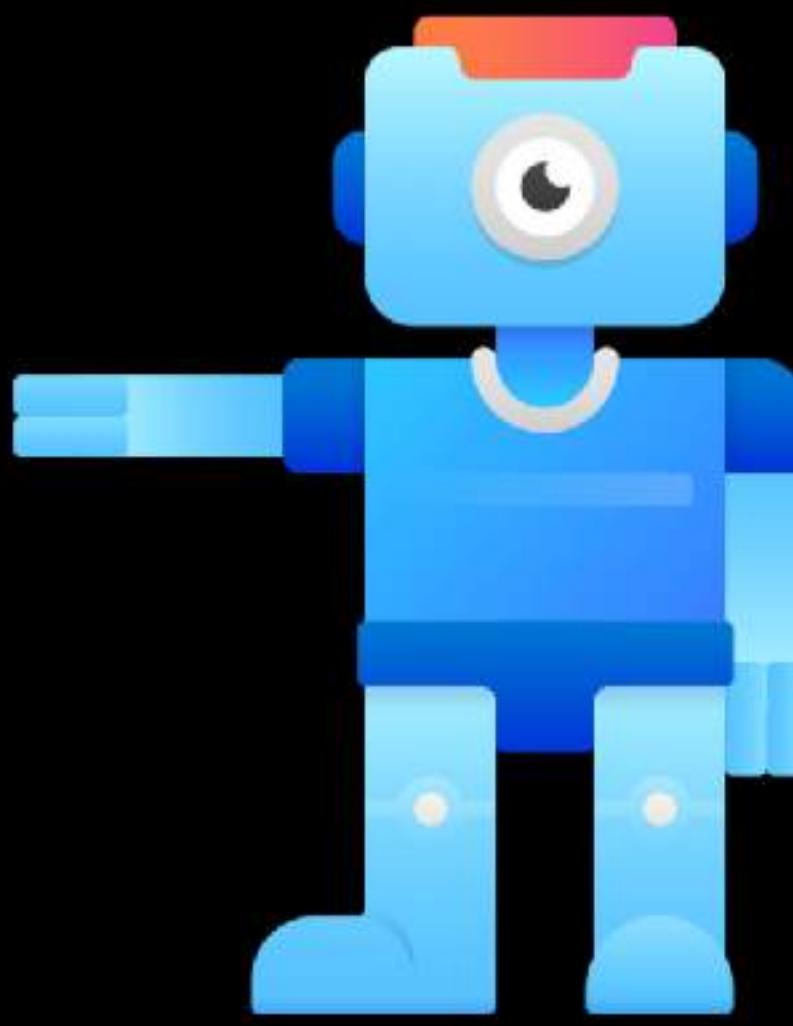
BY RAJEEV RATAN

# Next...

**PyTorch vs Keras vs TensorFlow 2.0**

# Deep Learning Libraries

**PyTorch vs Keras vs TensorFlow 2.0**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# A Quick Recap...

# TensorFlow

## Developed by Google



- Free and open-source Deep Learning Library
- Developed internally by Google Brain and open-sourced in 2015
- Written in C++ and CUDA (*NVIDIA's Compute Unified Device Architecture API for programming on GPUs*)
- Primarily used in Python but APIs/wrappers exist for several other languages (C++, Go, Java, JavaScript, Swift).
- TensorFlow 2.0 was released in 2019

# PyTorch

## Developed by Facebook



- Free and open-source Deep Learning Library
- Developed by Facebook's AI Research Team
- It is a C-based tensor Library written with CUDA capabilities
- Primarily meant for use in Python, though C++ is well supported
- A Python based scientific computing package aimed at:
  1. A replacement for NumPy using the power of GPUs
  2. A Deep Learning research platform providing both flexibility and speed

# Keras

Developed by François Chollet



- Free and open-source Deep Learning API Interface for Python
- Initially supported several Deep Learning Backends (TensorFlow & Theano), as of Version 2.3+, its sole focus was TensorFlow.
- Written in Python...for Python
- Primarily meant to make TensorFlow more accessible by abstracting some of low level work
- It also contained several implementations of NN layers and tools
- As for 2019, Keras is now bundled with TensorFlow2.0

# PyTorch vs Keras vs TensorFlow

- TensorFlow and PyTorch are fairly **low-level**
- Keras is **simpler, quicker to implement** and less prone to user errors
- TensorFlow has wider adoption in industry
- PyTorch due to flexibility, has gained more **popularity in research and academia**, taking over from Caffe

# More Differences between Keras vs PyTorch

- **Keras is a high-level API** that can use multiple backends, however as it has matured it has been integrated into TensorFlow 2.0
- **PyTorch is low-level** and requires more code and knowledge/details about your network to do the same things as Keras.
- After learning the theory behind CNN's PyTorch offers a more familiar approach whereas **Keras hides a lot of the nitty gritty details** (good and bad!)
- PyTorch offers a more **Pythonic design**

# Examples - PyTorch vs Keras

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3)
        self.conv2 = nn.Conv2d(32, 64, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 12 * 12, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 12 * 12)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
net = Net()
net.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss = 'categorical_crossentropy',
              optimizer = SGD(0.001),
              metrics = ['accuracy'])
```

- PyTorch requires us to use **class inheritance** to create its model
- We need to define layers and then a **forward propagation function** that executes the sequence
- We need to **know the output sizes** of our layers in PyTorch
- Lastly, PyTorch requires more **extensive knowledge of Python** to use well
- Keras abstracts us away from these specifics and coding, allowing us to **easily add layers with less code**
- Keras uses a **simpler modular** type approach allowing easy prototyping

# Why use the low-level PyTorch?

- PyTorch is more flexible, allows for more intricate experimentation
- Using PyTorch encourages you have a stronger understanding of Deep Learning
- Popularity is growing fast! Especially among academics and researchers

# Comparison Table - Keras vs PyTorch

	Keras with TF2.0	PyTorch
<b>Ease of Use</b>	<b>Far easier to get up and running</b>	<b>Requires deeper knowledge of Deep Learning and Python</b>
<b>Speed</b>	<b>Very fast due to TensorFlow 2.0 Backend</b>	<b>Capable of being faster due better data parallelism as it relies on native support for asynchronous execution through Python</b>
<b>Debugging</b>	<b>Good, but due to it's C++ core debug messages are harder to understand and less informative</b>	<b>Better, more informative debug messages.</b>
<b>Customization</b>	<b>Very good, but doesn't easily allow for exotic network designs</b>	<b>Excellent, low level frame work allows for very customisable configurations</b>
<b>Documentation</b>	<b>Excellent especially now that is has been included within TensorFlow 2.0's online documentation</b>	<b>Very good as it's supported by Facebook</b>
<b>Future Proof</b>	<b>Safe as it's supported by Google and has a big head start with Deep Learning practitioners</b>	<b>Safe as it's supported by Facebook and has a very active and growing community in both industry and academia</b>
<b>Features</b>	<b>A lot of prebuilt useful Deep Learning Functions</b>	<b>Lots of prebuilt features and many exotic algorithms can be found in the community</b>
<b>Reach</b>	<b>As of early 2020, Keras claimed to have user base of 375K individuals and usage within companies like Netflix, Yelp, Instcart along with NASA &amp; CERN</b>	<b>Gaining popularity in academia</b>
<b>Support</b>	<b>Embedded in TensorFlow 2.0 which is the most popular Deep Learning library and is supported by Google</b>	<b>High-level frameworks exist, such as fast.ai, Flare and Ignite.</b>
<b>Deployment</b>	<b>TensorFlow Models have much better deployment options with TensorFlow Serving Framework.</b>	<b>PyTorch relies on users using Flask or Django to deploy models</b>

# When to Choose PyTorch

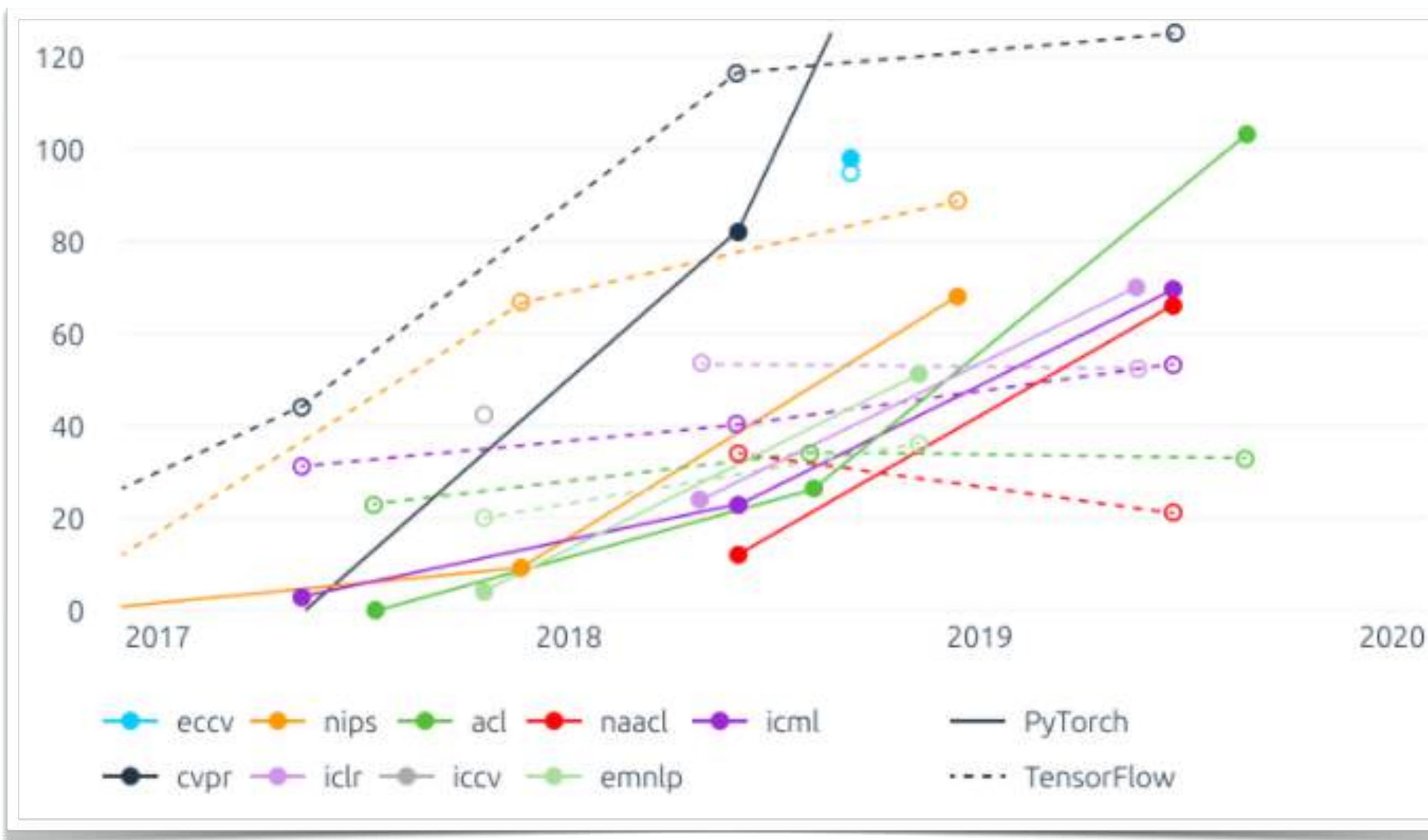
- If you want more visibility over your network
- Research custom/exotic models
- When precise control over your model/network is needed
- Altering an existing PyTorch Model
- Skilled with Python

# When to Choose Keras

- For quick prototyping
- When generic models suffice and you need to deploy quickly
- Maintaining consistency with using TensorFlow
- When you have less data
- When you'd like to experiment with various simple models

# Popularity

## Unique Mentions at Popular AI Research Conferences



Source: [horace.io](https://horace.io)

## Google Trend Analysis



Source: [horace.io](https://horace.io)

# Why aren't we learning pure TensorFlow?

- It is more low-level than PyTorch
- Requires more workarounds to get things working than PyTorch
- Keras provides 95% of the Deep Learning capabilities we'll be using with Computer Vision
- Learning Keras and PyTorch offer the best in terms of your capabilities and job market prospects
- Many companies primarily use Keras or PyTorch not pure TensorFlow

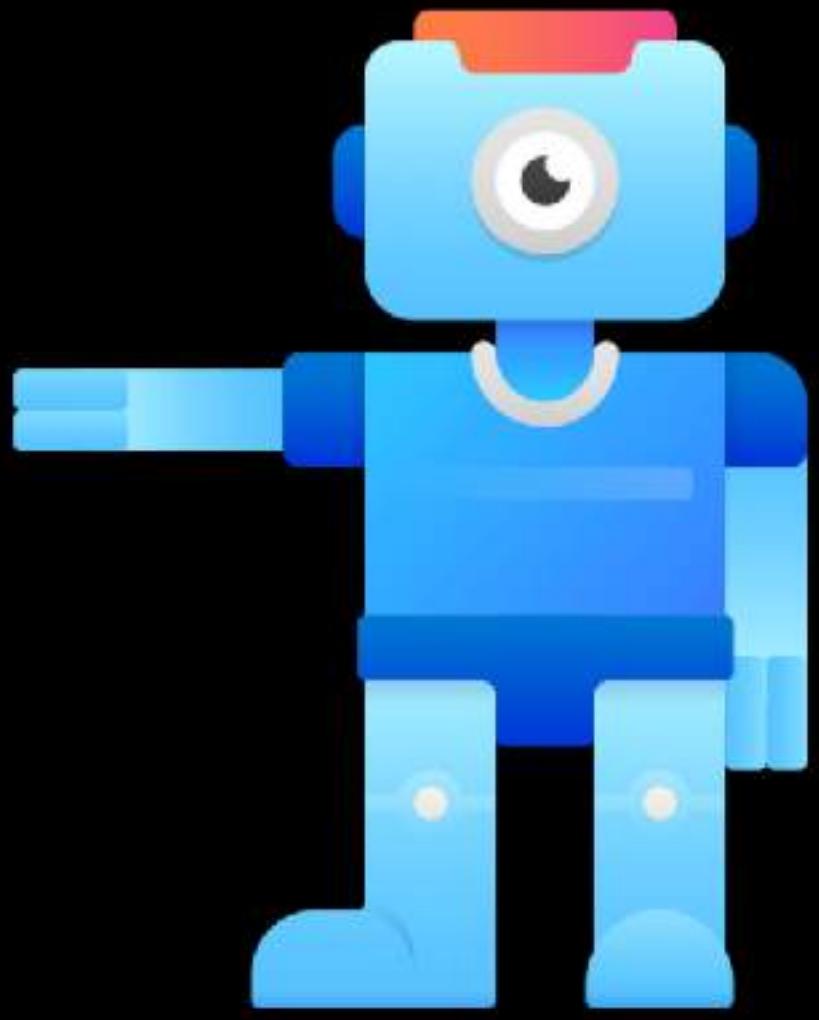


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

## Assessing Model Performance



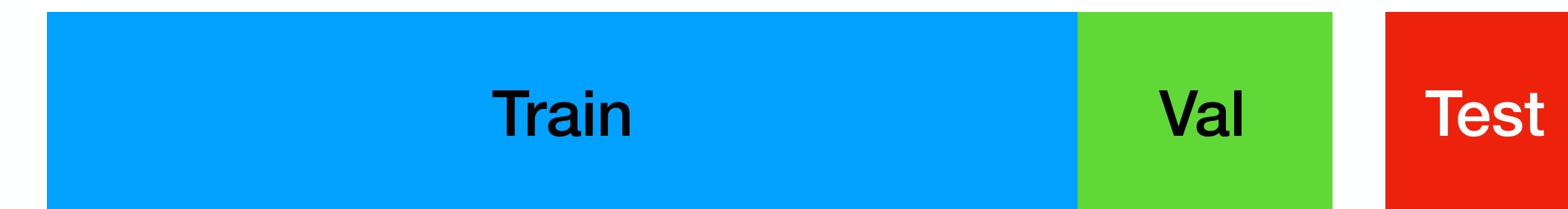
# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Assessing Model Performance

Metrics we use to assess the performance of a Model

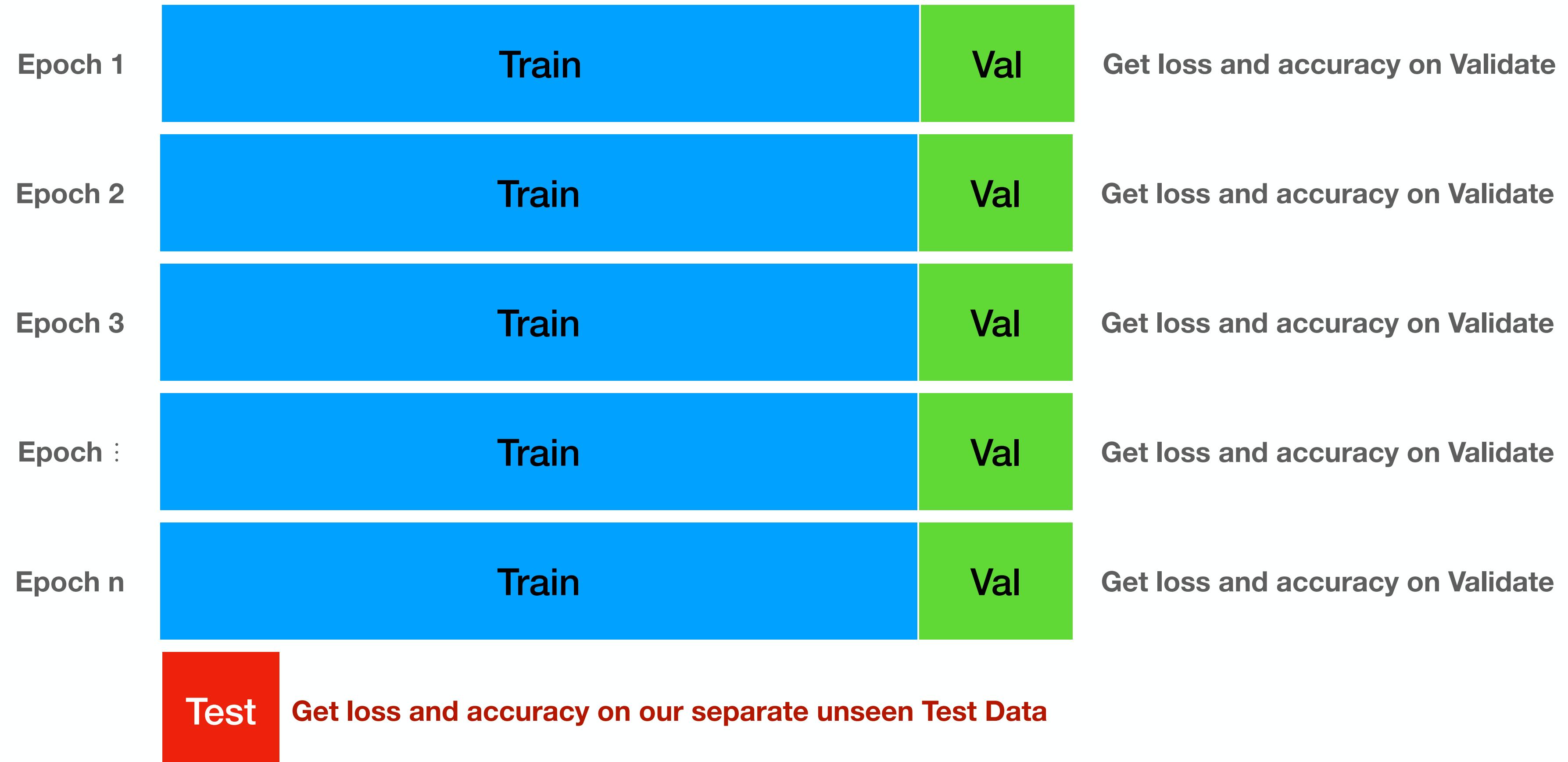
# Validation, Test and Training Datasets



- **Training Data** is what we use to **optimise the model parameters**
- Often you'll see me and others refer to a **Test** dataset or **Validation** dataset
- In ML nomenclature, **Test** dataset often refers to what we test or benchmark the model performance on.
- A **Validation** dataset is what we use to provide an unbiased evaluation of our model during training.
- Those performance metrics (test and accuracy) you see after each Epoch are based on the **Validation** dataset
- After training our model, we typically should have a **separate Test** dataset to do an evaluation on the **final model**.

# An Example

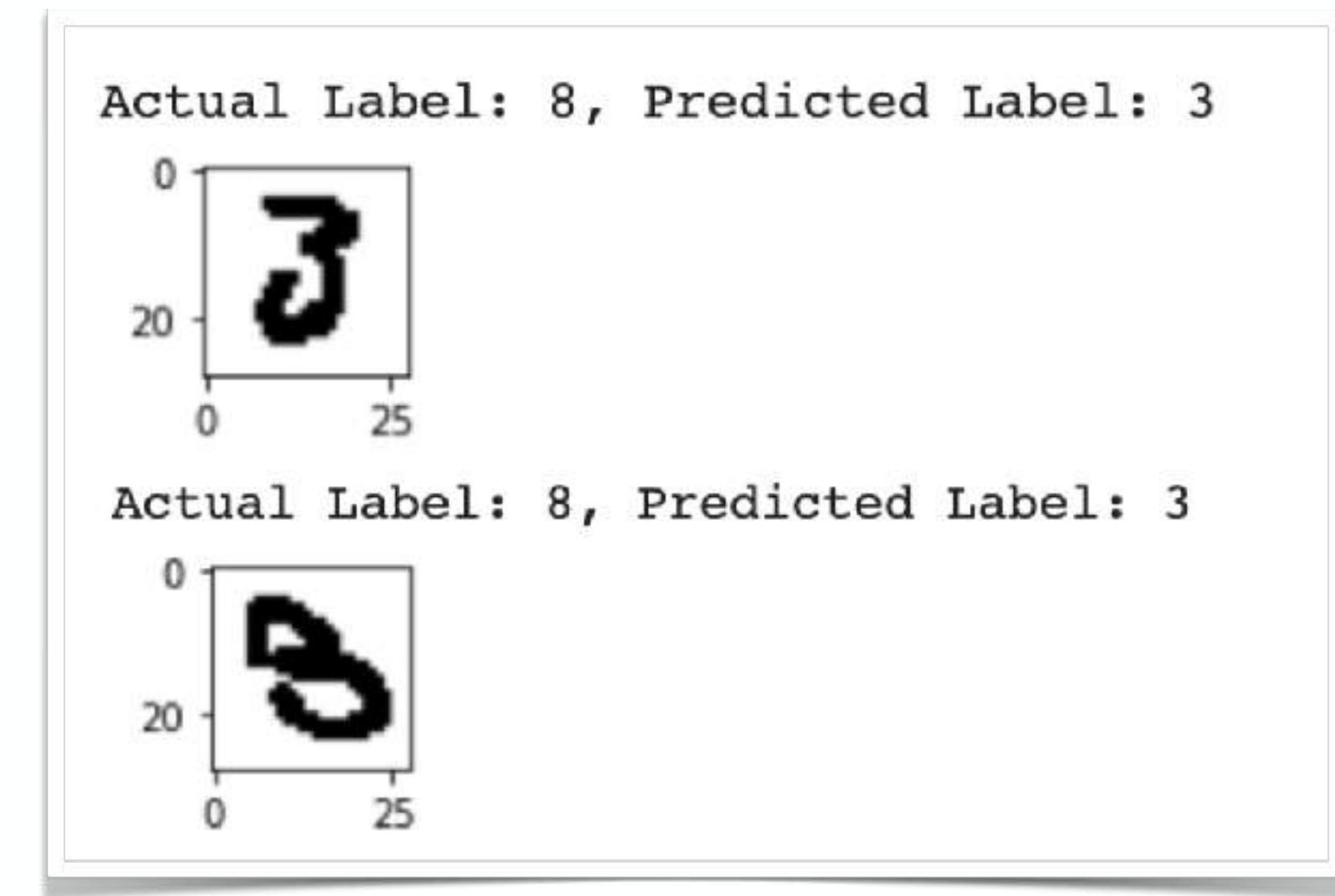
Optimize Weights using Gradient Descent and BackProp



# Basic Performance Metrics

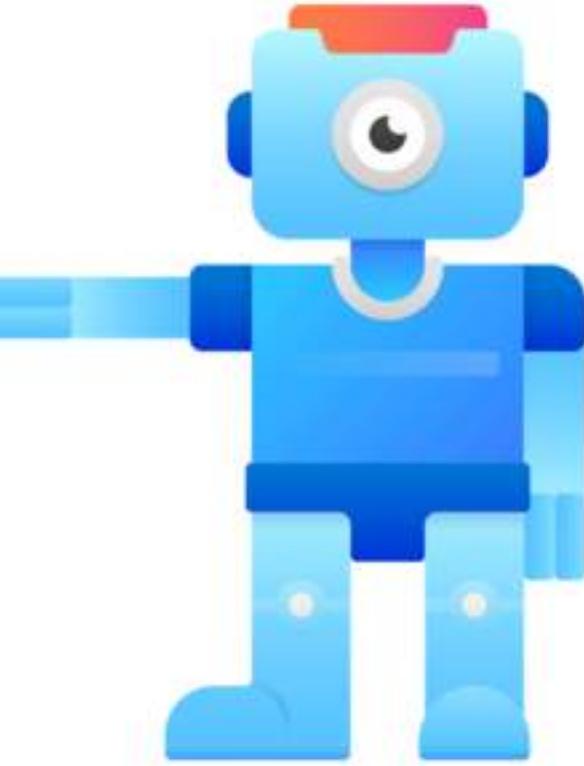
- Training Loss
- Training Accuracy
- Test/Validation Lost
- Test/Validation Accuracy

# What if our Hand Written Digit Model Had Issues confused 8s with 3s?



- How do we identify such problems with our model?
- How do we know which classes perform worst?
- How do we know if it has a predisposition for predicting 3s instead of 8s?

We can use the Confusion Matrix and Classification Report

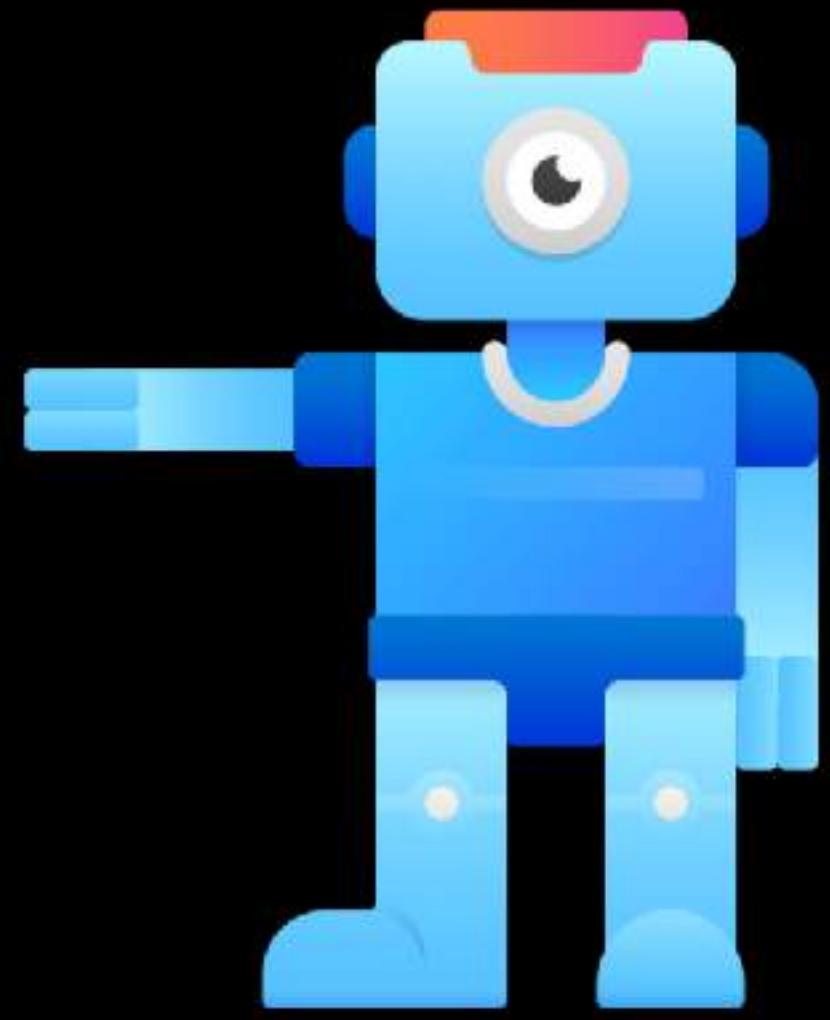


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

The Confusion Matrix and Classification Reports



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## The Confusion Matrix and Classification Reports

How we can analyse our model performance in more detail

# Confusion Matrix

- A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.
- We can generate this matrix easily by using scikit-learn's function

```
from sklearn.metrics import confusion_matrix
```

- It's generated using two input arguments:
  - All our Test dataset **ground truth** labels
  - All our Test dataset **predicted labels**, generated using our model

# MNIST Confusion Matrix Example

```
[ [ 973   0   2   0   0   1   1   1   2   0 ]  
  [ 0 1128   1   1   0   0   2   1   2   0 ]  
  [ 2   2 1018   1   1   0   0   4   4   0 ]  
  [ 0   0   0 1001   0   5   0   0   4   0 ]  
  [ 0   0   2   0 971   0   0   0   0   9 ]  
  [ 2   0   0   3   0 884   1   0   0   2 ]  
  [ 6   2   1   0   2   5 939   0   3   0 ]  
  [ 0   2   7   1   0   0   0 1014   2   2 ]  
  [ 5   0   3   0   0   1   1   2 959   3 ]  
  [ 2   2   0   3   5   3   0   5   3 986 ] ]
```

What do these numbers mean?

# MNIST Confusion Matrix Example

Predicted Labels										True Labels
0	1	2	3	4	5	6	7	8	9	
[ [ 973 0 2 0 0 1 1 1 2 0 ] ]	0									
[ 0 1128 1 1 0 0 2 1 2 0 ] ]	1									
[ 2 2 1018 1 1 0 0 4 4 0 ] ]	2									
[ 0 0 0 1001 0 5 0 0 4 0 ] ]	3									
[ 0 0 2 0 971 0 0 0 0 9 ] ]	4									
[ 2 0 0 3 0 884 1 0 0 2 ] ]	5									
[ 6 2 1 0 2 5 939 0 3 0 ] ]	6									
[ 0 2 7 1 0 0 0 1014 2 2 ] ]	7									
[ 5 0 3 0 0 1 1 2 959 3 ] ]	8									
[ 2 2 0 3 5 3 0 5 3 986 ] ]	9									

- **Green** - Our model predicted 884 5s correctly
- **Red** - Our model predicted 2 0s as 5s, 3 3s as 5, 1 6 as a 5 and 2 9s as 5s

# What Main Issues Can We Deduce?

Predicted Labels										True Labels
0	1	2	3	4	5	6	7	8	9	
[ [ 973	0	2	0	0	1	1	1	2	0 ]	0
[ 0 1128	1	1	0	0	2	1	2	0 ]	1	
[ 2 2 1018	1	1	1	0	0	4	4	0 ]	2	
[ 0 0 0 1001	0	5	0	0	0	4	0 ]	3		
[ 0 0 2 0 971	0	0	0	0	0	0	9 ]	4		
[ 2 0 0 3 0 884	1	0	0	0	0	2 ]	5			
[ 6 2 1 0 2 5 939	0	3	0 ]	6	0 ]	0 ]	6			
[ 0 2 7 1 0 0 0 1014	2	0 ]	7	0 ]	0 ]	2 ]	7			
[ 5 0 3 0 0 1 1 2 959	3 ]	0 ]	8	0 ]	1 ]	2 ]	8			
[ 2 2 0 3 5 3 0 5 3 986 ] ]	9	0 ]	9	0 ]	1 ]	2 ]	9			

- Our model seems to sometimes predict 4s as 9s
- Our model seems to sometimes predict 2s as 7s

# Binary Classification Problems

Imagine we had an Image Classifier that Predicted a Yes or No if a patient had a disease based on an X-Ray Scan

<b>N = 145</b>	<b>Predicted NO</b>	<b>Predicted YES</b>	
<b>True Label NO</b>	<b>True Negatives: 40</b>	<b>False Positives: 5</b>	<b>Total True Negatives: 45</b>
<b>True Label YES</b>	<b>False Negatives: 10</b>	<b>True Positives: 90</b>	<b>Total True Positives: 100</b>
	<b>Predicted Negatives: 50</b>	<b>Predicted Positives: 95</b>	

- **Accuracy** =  $\frac{(TP + TN)}{Total} = \frac{40 + 90}{145} = \frac{130}{145} = 0.897$
- **Misclassification or Error Rate** =  $1 - Accuracy = 1 - 0.897 = 0.103$

# Binary Classification Problems

Imagine we had an Image Classifier that Predicted a Yes or No if a patient had a disease based on an X-Ray Scan

<b>N = 145</b>	<b>Predicted NO</b>	<b>Predicted YES</b>	
<b>True Label NO</b>	<b>True Negatives: 40</b>	<b>False Positives: 5</b>	<b>Total True Negatives: 45</b>
<b>True Label YES</b>	<b>False Negatives: 10</b>	<b>True Positives: 90</b>	<b>Total True Positives: 100</b>
	<b>Predicted Negatives: 50</b>	<b>Predicted Positives: 95</b>	

- **True Positive Rate or Sensitivity or Recall** =  $\frac{TP}{TruePositiveLabels} = \frac{90}{100} = 0.9$  when it's a Yes, how often do we predict Yes
- **False Positive Rate** =  $\frac{FP}{TrueNegativeLabels} = \frac{5}{45} = 0.11$  when it's actually No, how often does it predict Yes.
- **True Negative Rate or Specificity** =  $\frac{TN}{TrueNegativeLabels} = \frac{40}{45} = 0.89$  when it's actually No, how often is it really a No.

# Binary Classification Problems

Imagine we had an Image Classifier that Predicted a Yes or No if a patient had a disease based on an X-Ray Scan

<b>N = 145</b>	<b>Predicted NO</b>	<b>Predicted YES</b>	
<b>True Label NO</b>	<b>True Negatives: 40</b>	<b>False Positives: 5</b>	<b>Total True Negatives: 45</b>
<b>True Label YES</b>	<b>False Negatives: 10</b>	<b>True Positives: 90</b>	<b>Total True Positives: 100</b>
	<b>Predicted Negatives: 50</b>	<b>Predicted Positives: 95</b>	

- Precision =  $\frac{TP}{PredictedYes} = \frac{90}{95} = 0.95$  when it's Yes, how often is it right?

# Precision, Recall, F1

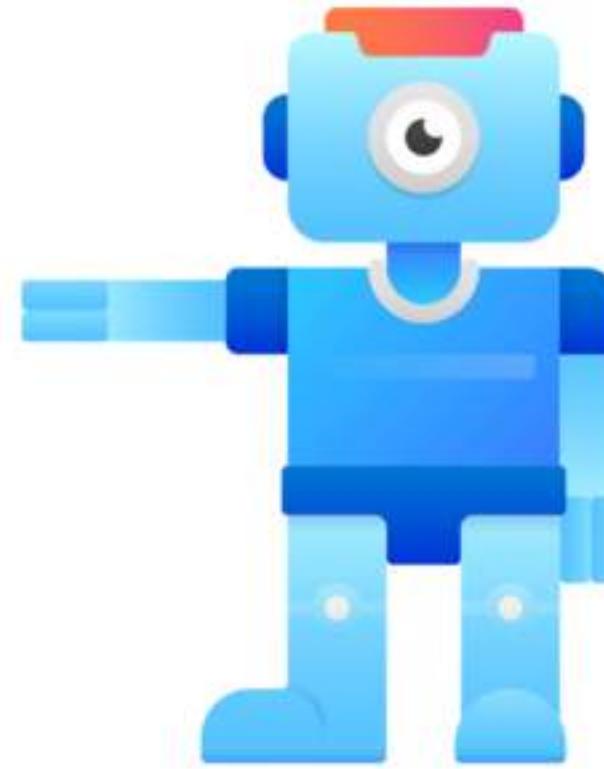
- **Recall** or **True Positive Rate** or **Sensitivity** - when it's a Yes, how often do we predict Yes. In other words how good is our classifier at identifying all occurrences of our class.
- **Precision** - When we predict a Yes, how often is that Yes correct.
- **Trade offs** - Sometimes we can accept a low precision with high recall. Example in disease prediction. E.g. with Covid-19 testing, we want to identify persons with the disease as much as possible at the expense of False Positives (recall).
- **F1 Score** - is the harmonic mean (punishes extreme values) of both **precision** and **recall**.  
**It's more informative than accuracy alone as it**

$$\bullet \quad F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

# Classification Reports

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.98	0.99	0.99	1032
3	0.99	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.98	0.99	0.99	892
6	0.99	0.98	0.99	958
7	0.99	0.99	0.99	1028
8	0.98	0.98	0.98	974
9	0.98	0.98	0.98	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

The support is the number of occurrences of each class

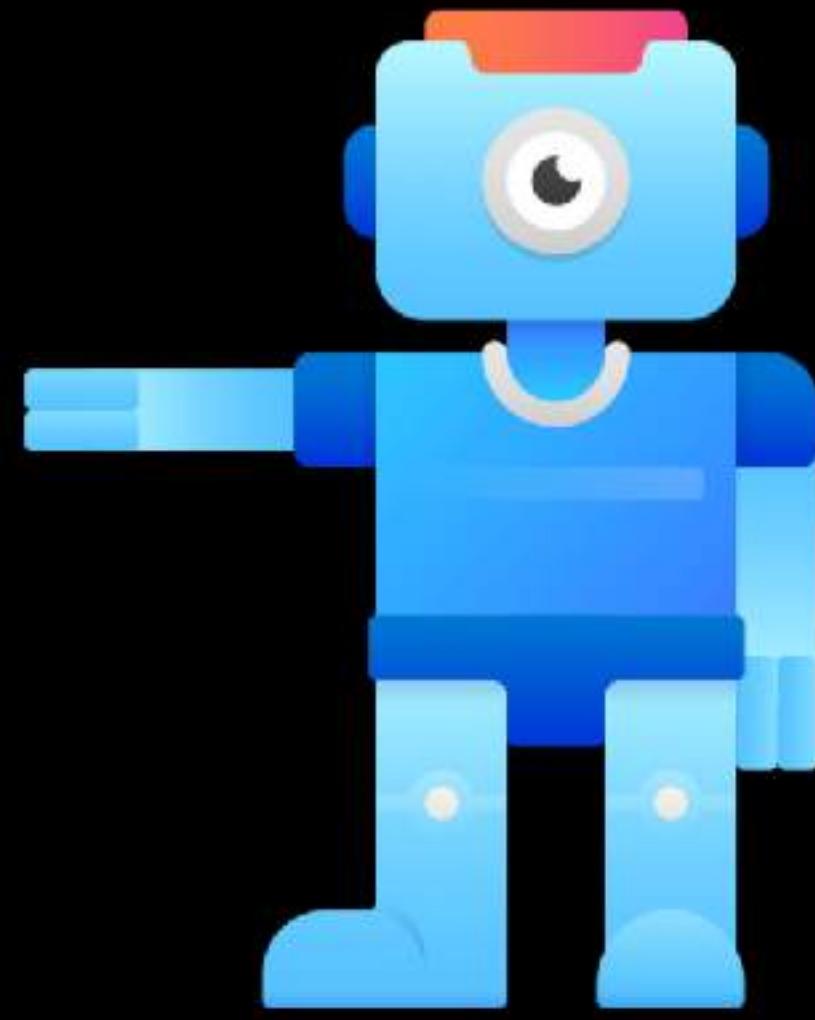


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Train our MNIST CNN Model and Create Classification and Confusion Matrix Reports in Keras and PyTorch**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Understanding Overfitting and Generalisation

We explore the ill effects of overfitting

# What Makes Ensures Our Model is Good?

- We just covered several metrics to assess model performance
- So how do we obtain good performance?
  - Use lots of data?
  - Deeper, more complex models?
  - Train for lots of Epochs?
- We use a few techniques to reduce **Overfitting** and improve **Generalisation**

# Overfitting

- A very common issue that plagues model development is overfitting.
- It occurs when a model has ‘over-trained’ on the training data and thus ends up performing poorly on our Test or Validation datasets.
- This often happens when we **don’t have enough data**, or have used too many features or developed an overly complex model

# Generalisation

- A measure of how well our model performs on new **unseen data**.
- Models that overfit to the training data perform poorly on new data, hence having poor Generalisation.
- A model that stores a lot of information has the potential to be more accurate by leveraging more features, but that also puts it at the risk of storing irrelevant features.

# An Example of Overfitting

- Imagine you are given 3 images of cars and 3 images of Fire Trucks and you are told

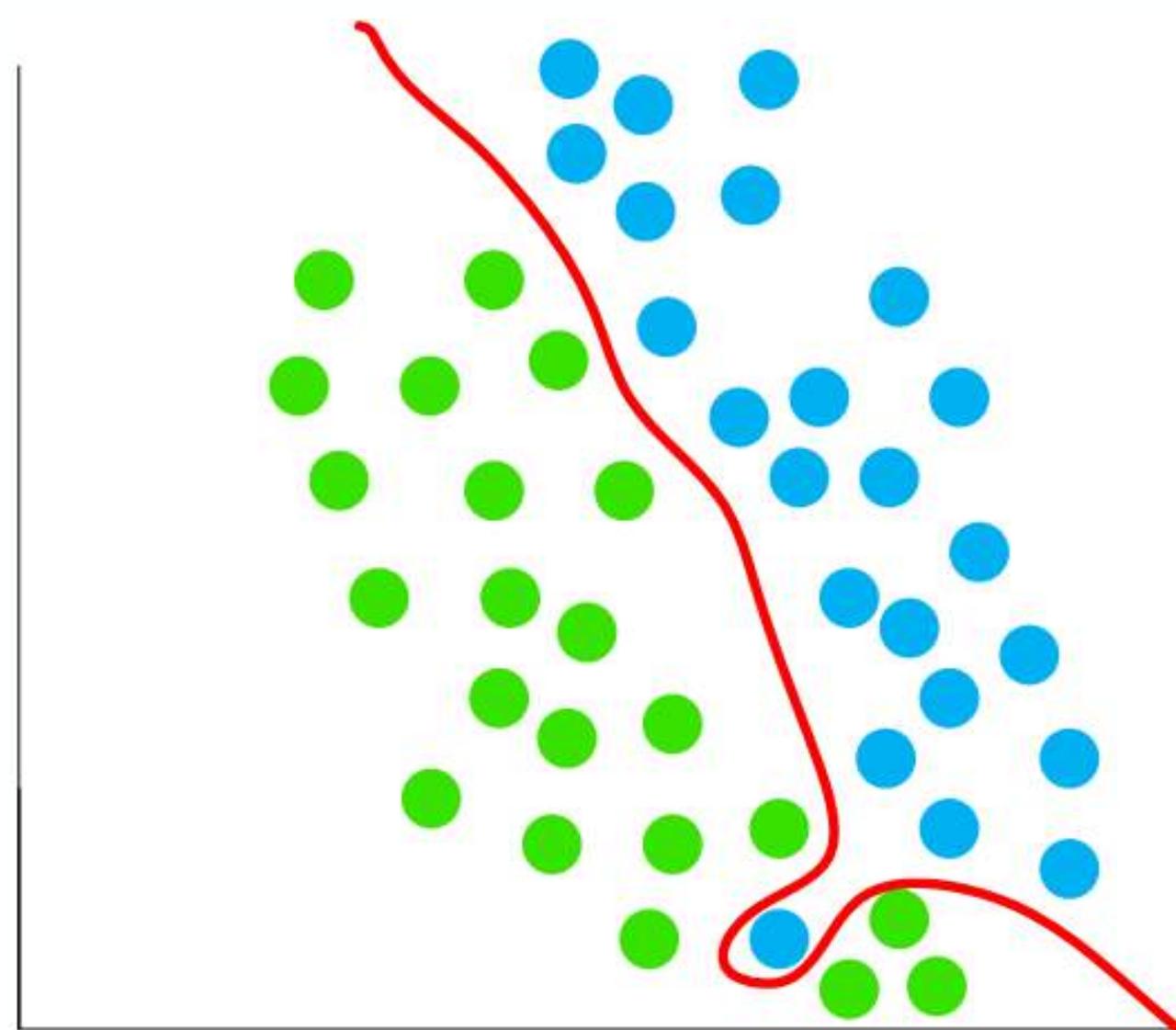


If your model learned to use the colour red as a feature to identify Firetrucks, then you might have classified this car as a Fire Truck. This means you have Overfit!

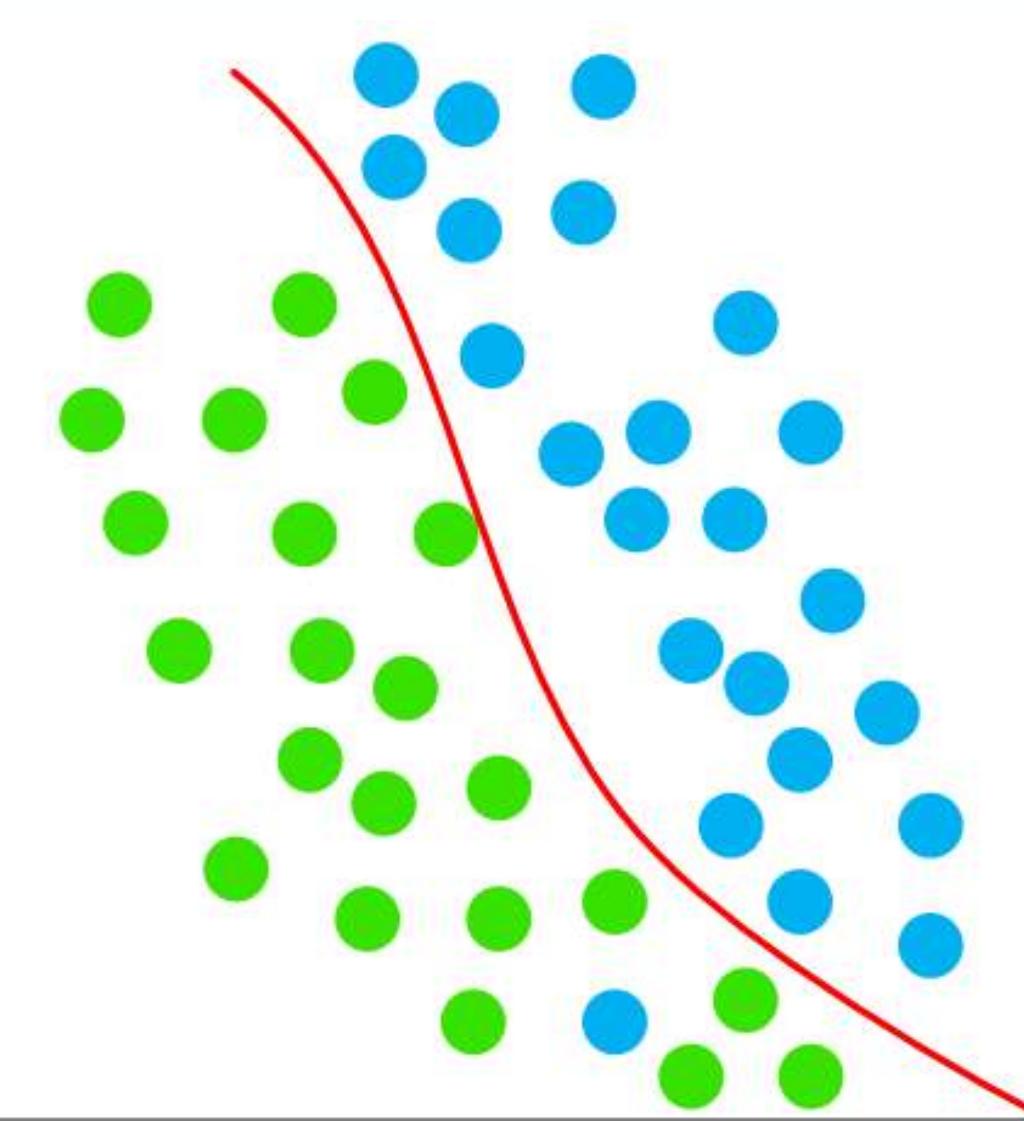
# An Example

## Which Model is Best?

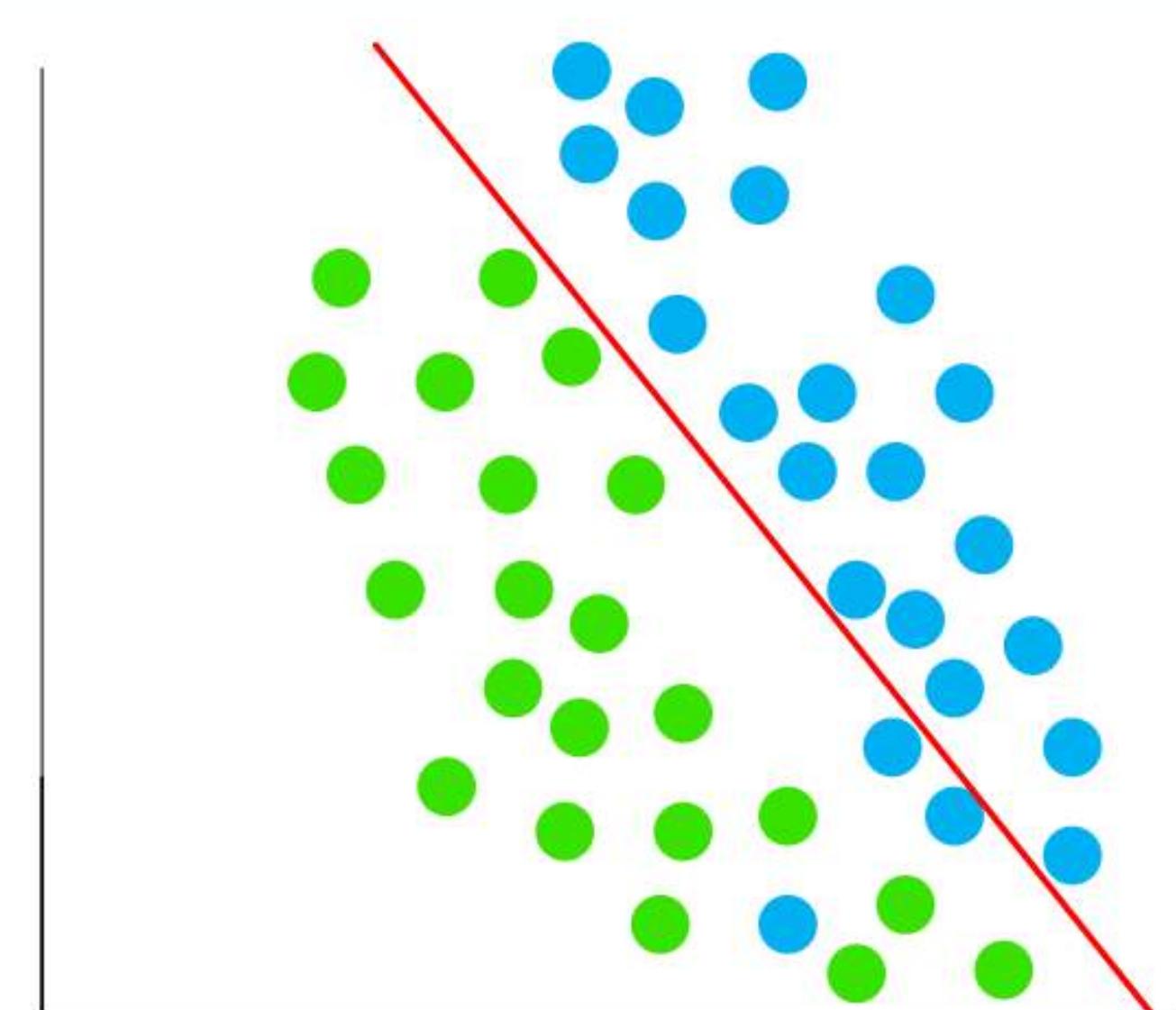
Model A



Model B



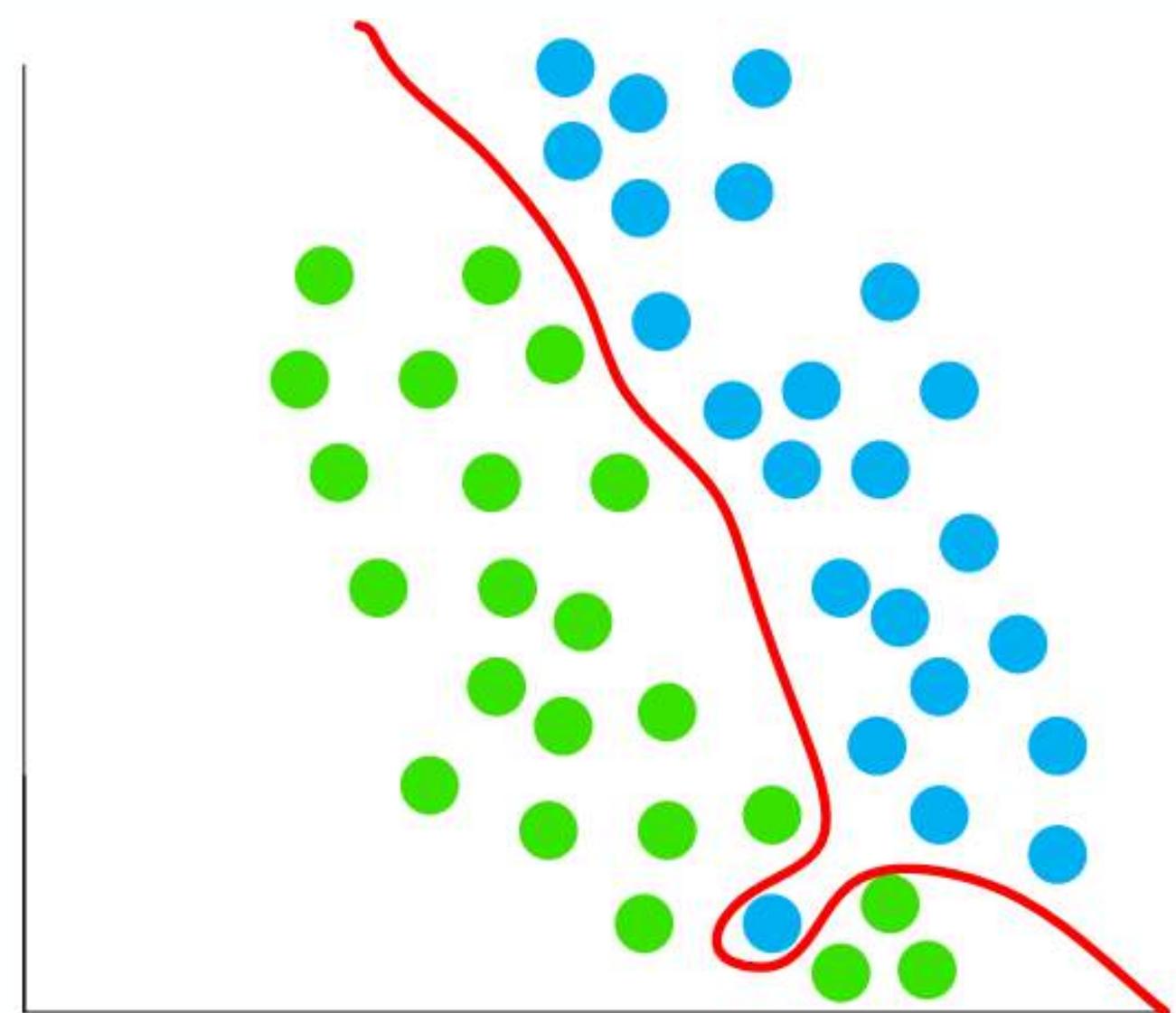
Model C



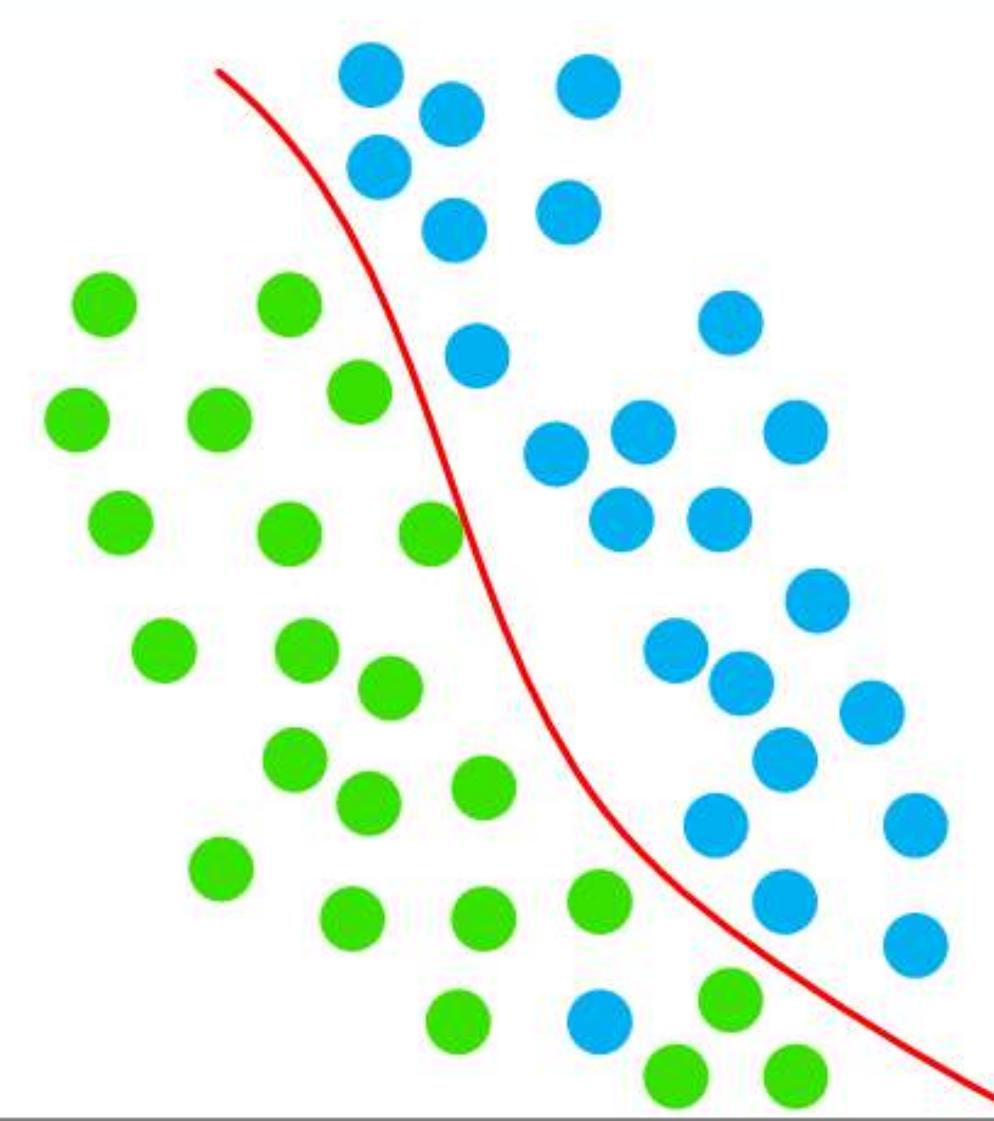
# An Example

## Model B was best

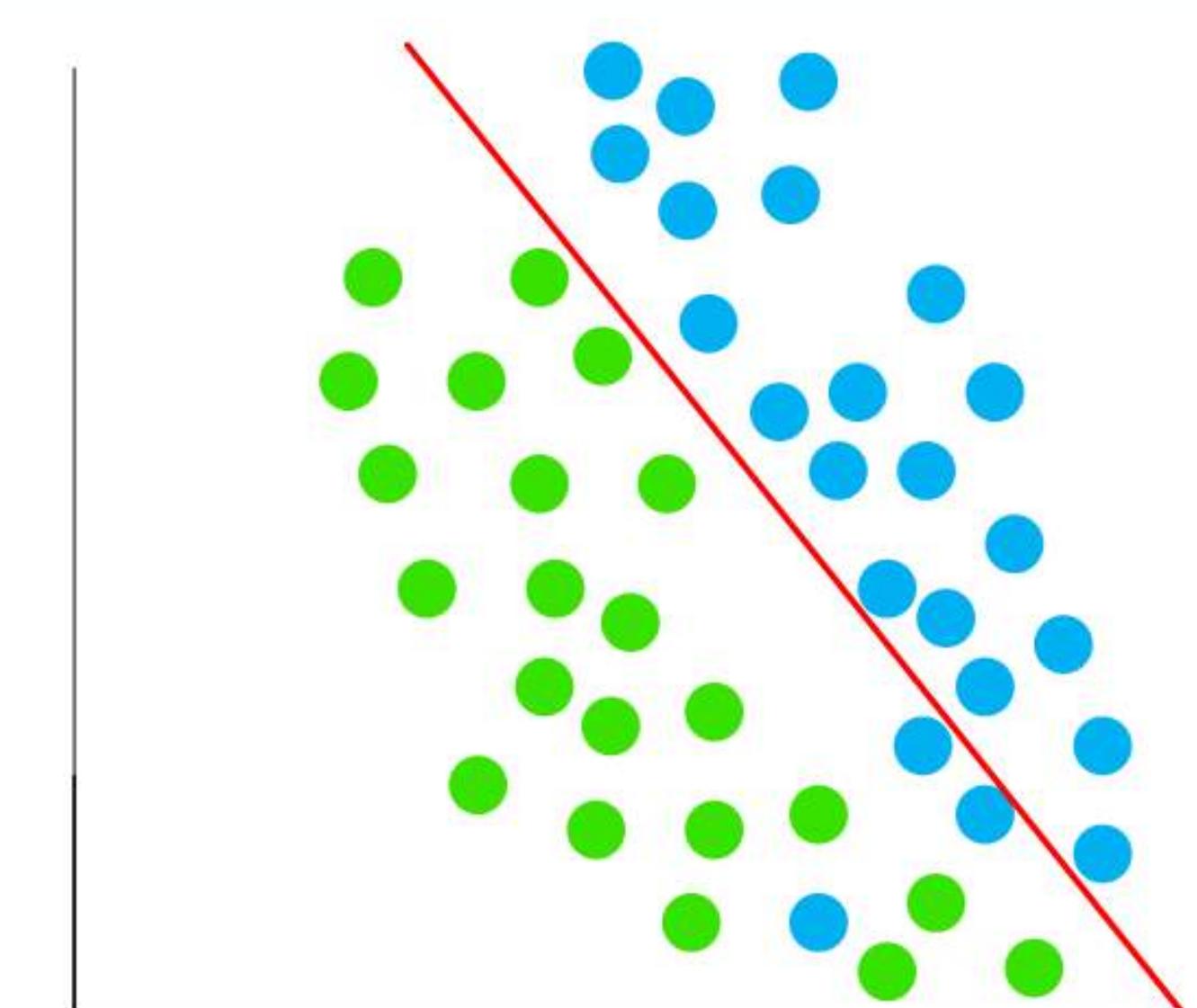
Overfit



Generalised Well



Under-fit

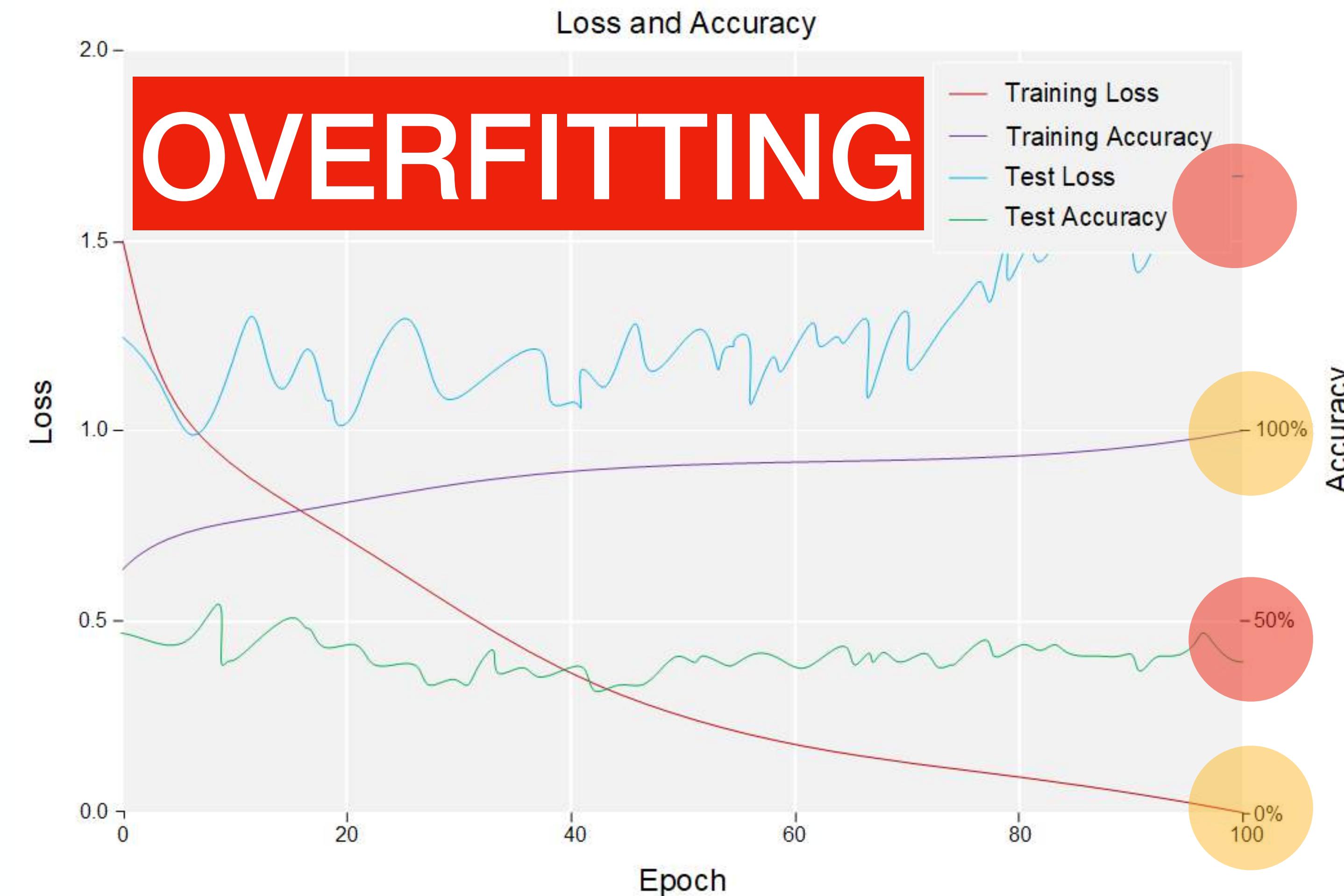


- High Variance
- Fits Noise

- Low Variance

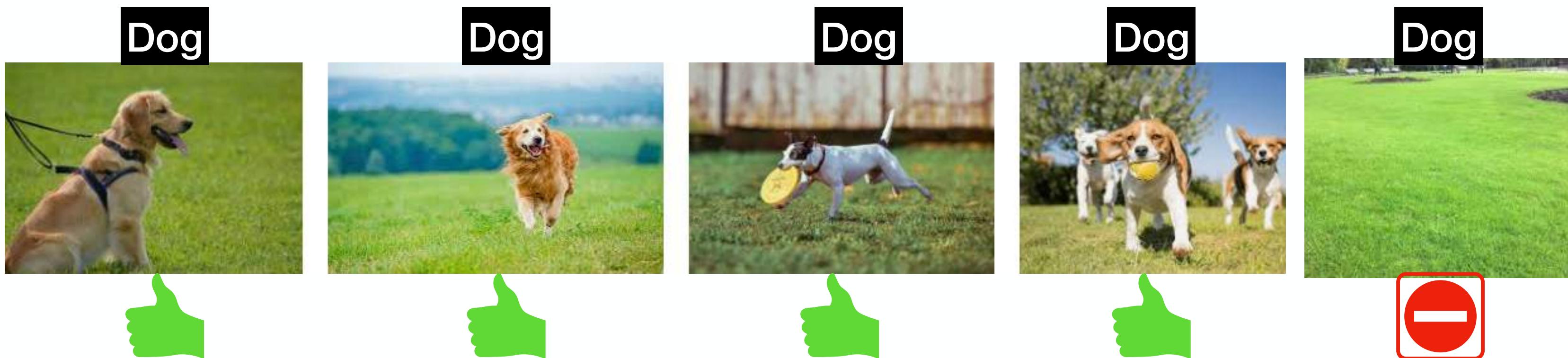
# Training Accuracy vs Test Accuracy

Problem - Training Accuracy can be very good, but Test Accuracy poor



# Avoiding Overfitting

- Using large datasets or reducing model complexity is one way, but it's better to use **Regularisation**
- **Regularisation** - is a method by which we **control the model complexity** and ensure the model is using the right features to classify objects.
- E.g. for a 'Dog or not Dog' classifier, we'd like our classifier to look at the overall shape, tail, nose, ears and mouth as important features in identifying dogs and NOT associate it with common things in the image such as trees, or grass.



# How to confuse machine learning:



<https://twitter.com/drjuliashaw/status/874293864814845952>

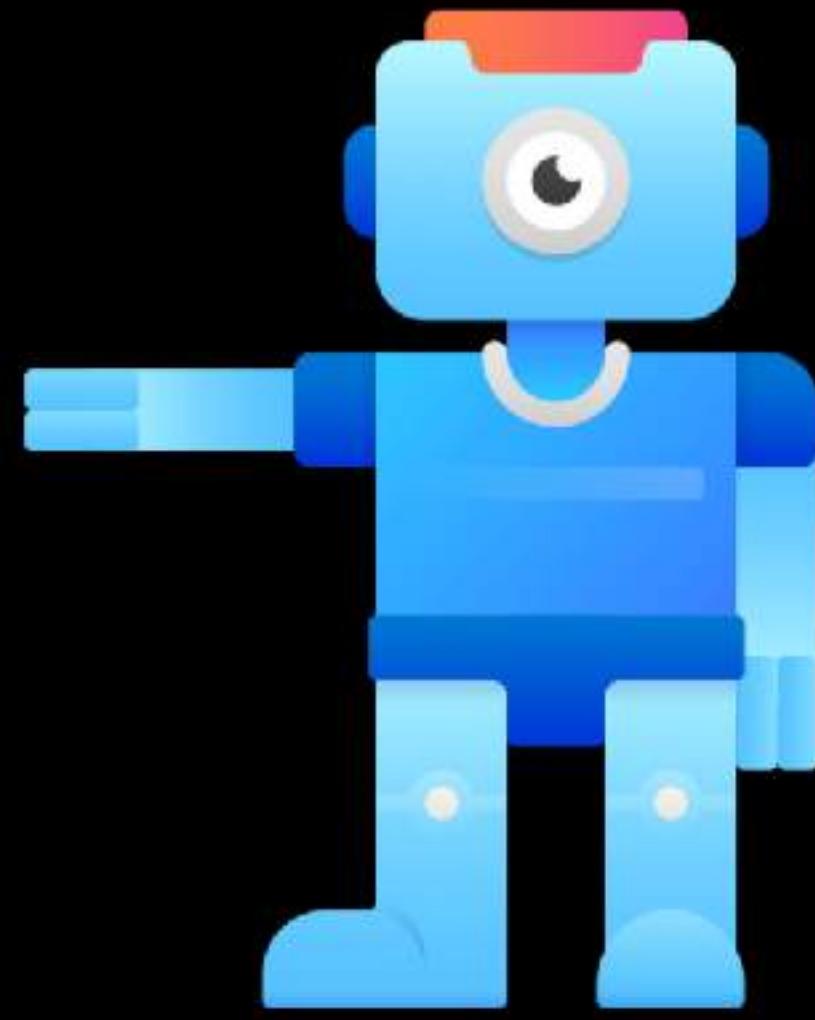


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

## Regularisation Methods



# MODERN COMPUTER VISION

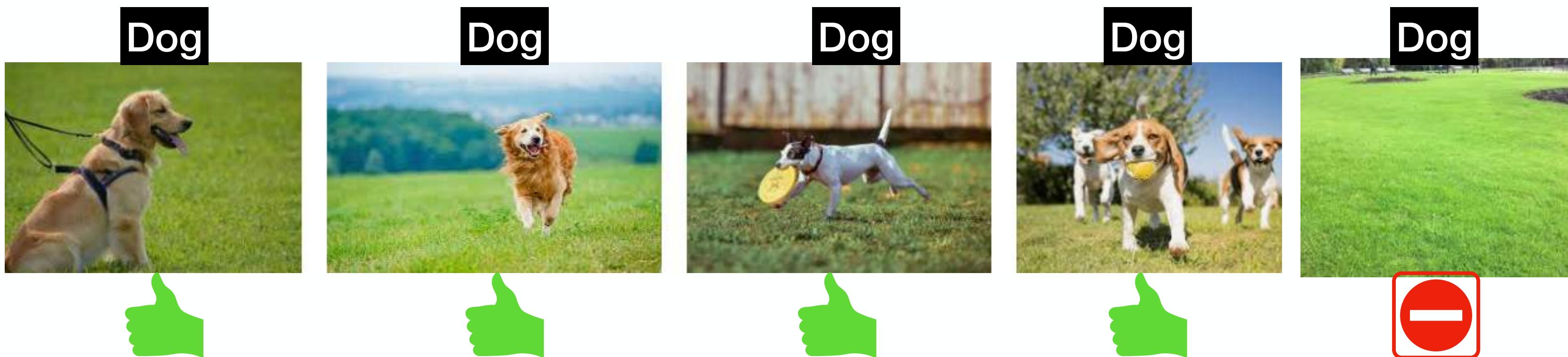
BY RAJEEV RATAN

## Regularisation Methods

We discuss how and the types of Regularisation that can be used to reduce Over Fitting

# Regularisation

- **Regularisation** - is a technique used as an attempt to **reduce overfitting**.
- It allows us to **control the model complexity** and ensure the model is using the **right features** to classify objects.
- E.g. for a ‘Dog or not Dog’ classifier, we’d like our classifier to look at the overall shape, tail, nose, ears and mouth as important features in identifying dogs and NOT associate it with common things in the image such as trees, or grass.



# Methods of Regularisation

## Commonly used in Computer Vision Deep Learning

- L1 & L2 Regularisation
- Data Augmentation
- Drop Out
- Early Stopping
- Batch Normalisation

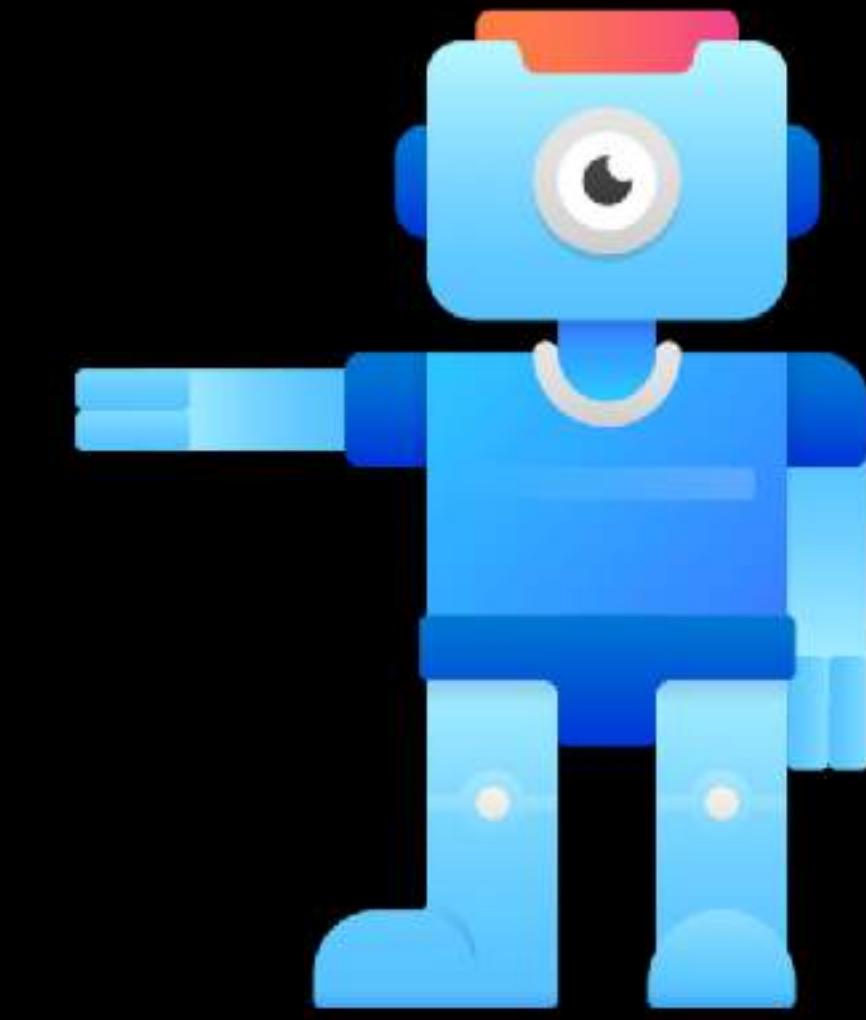


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**L1 and L2 Regularisation**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## L1 and L2 Regularisation

We explore L1 and L2 Regularisation

# L1 and L2 Regularisation

## Weight Constraining

- L1 and L2 Regularisation work by forcing parameters (weights and biases) to take small values.
- This works because by reducing the weights in our network, we decrease their effects on the activation function

# L1 Regularisation or L1-Norm or Lasso Regression

$$\text{Loss Function} + \lambda \sum_{j=1}^p |\beta_j|$$

- Where,  $\beta_j$  are our weights and  $\lambda$  controls the effect of the penalty applied (less than one)
- A large  $\lambda$  means we are making our penalty term larger
- L1 uses an **absolute value** as a penalty

# L2 Regularisation or Ridge Regression

$$LossFunction + \lambda \sum_{j=1}^p \beta_j^2$$

- Where,  $\beta_j$  are our weights and  $\lambda$  controls the effect of the penalty applied (less than one)
- A large  $\lambda$  means we are making our penalty term larger
- L2 uses a **squared magnitude** of weights as a penalty

# Differences between L1 and L2

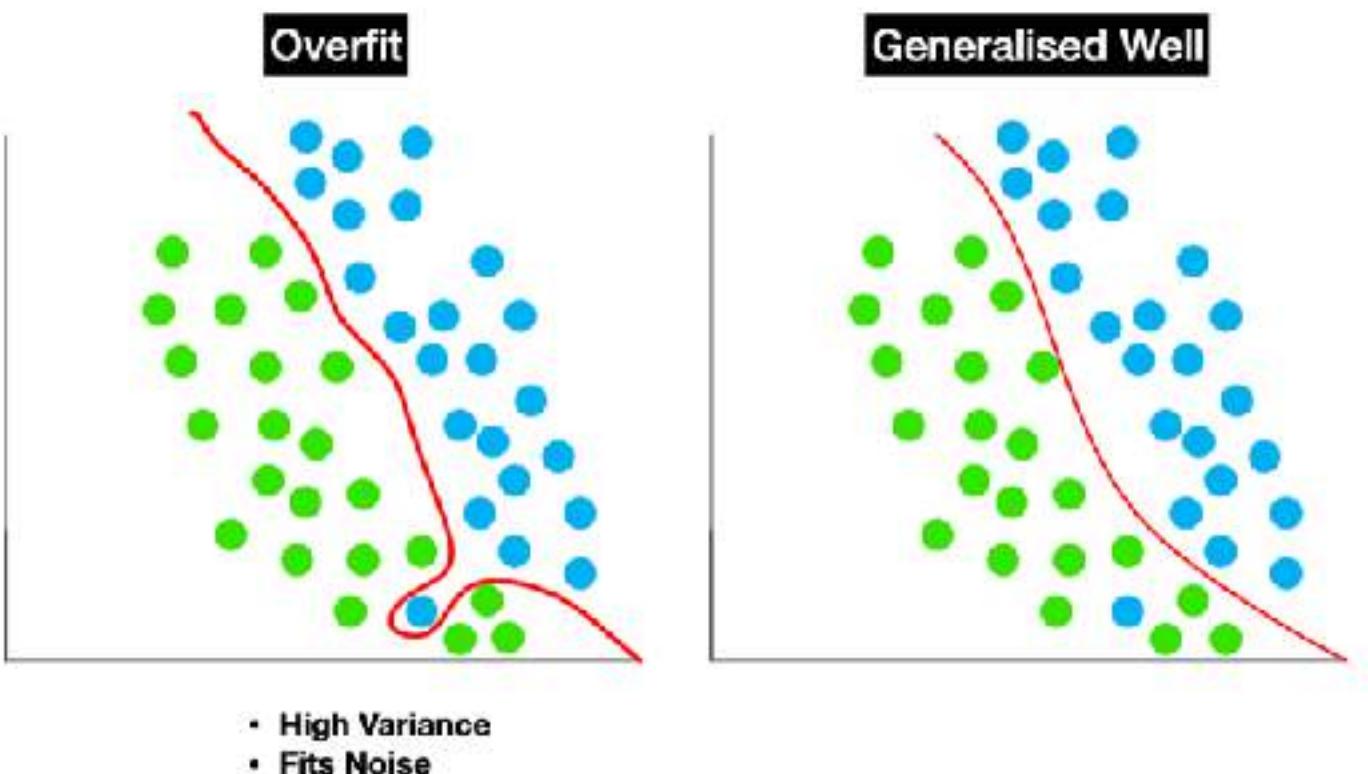
$$L1: LossFunction + \lambda \sum_{j=1}^p |\beta_j|$$

$$L2: LossFunction + \lambda \sum_{j=1}^p \beta_j^2$$

- In L1 weights shrink by a constant amount toward 0.
- By shrinking less important feature weights to zero, it acts like a feature selection algorithm, yielding sparse models.
- In L2, weights shrink by an amount proportional to the weights
- L2 penalises large weights more, smaller weights less.
- L1 tends to concentrate the weight of the network to a relatively small number but high important connections.

# Summary of L1 and L2

- L1 and L2 Regularisation make our network prefer to learn smaller weights.
- Large weights are allowed only if they considerably improve the first part of our cost function.
- This can be interpreted as a way of compromising between finding smaller weights and minimising the original loss function.
- The degree of compromise depends on  $\lambda$
- A small  $\lambda$  means we prefer to minimise the original lost function
- A large  $\lambda$  means we prefer smaller weights



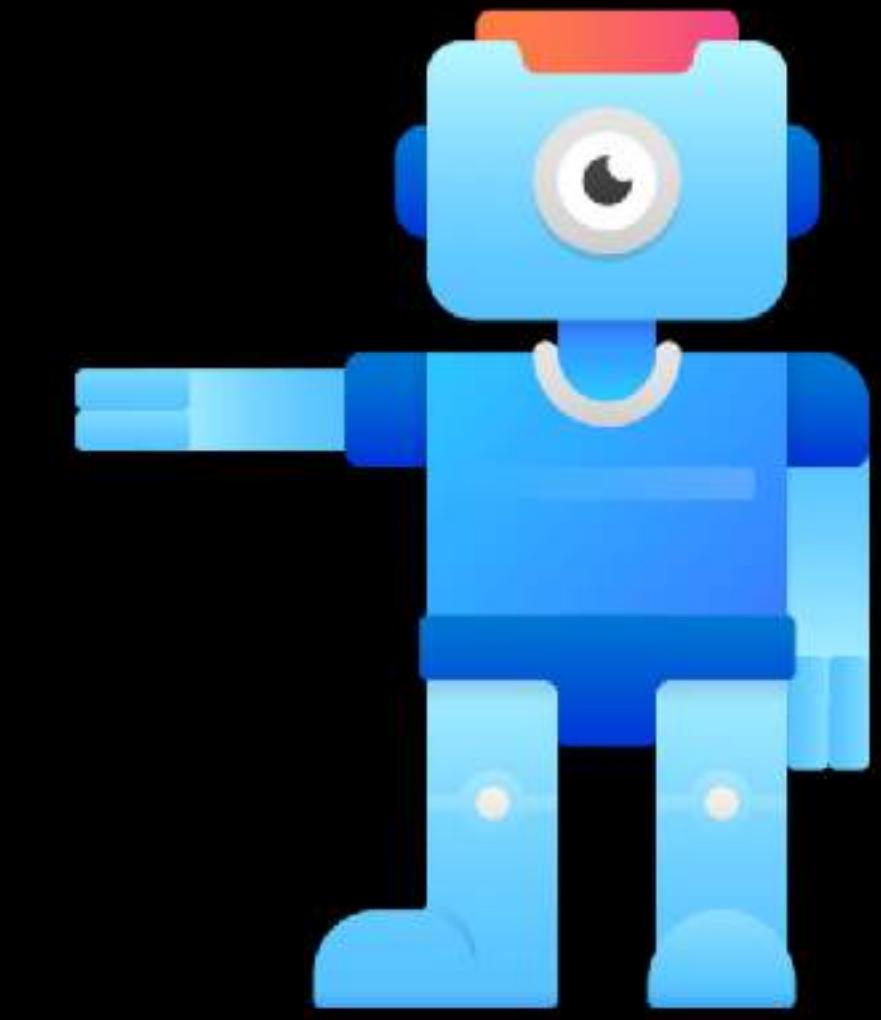


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Dropout**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

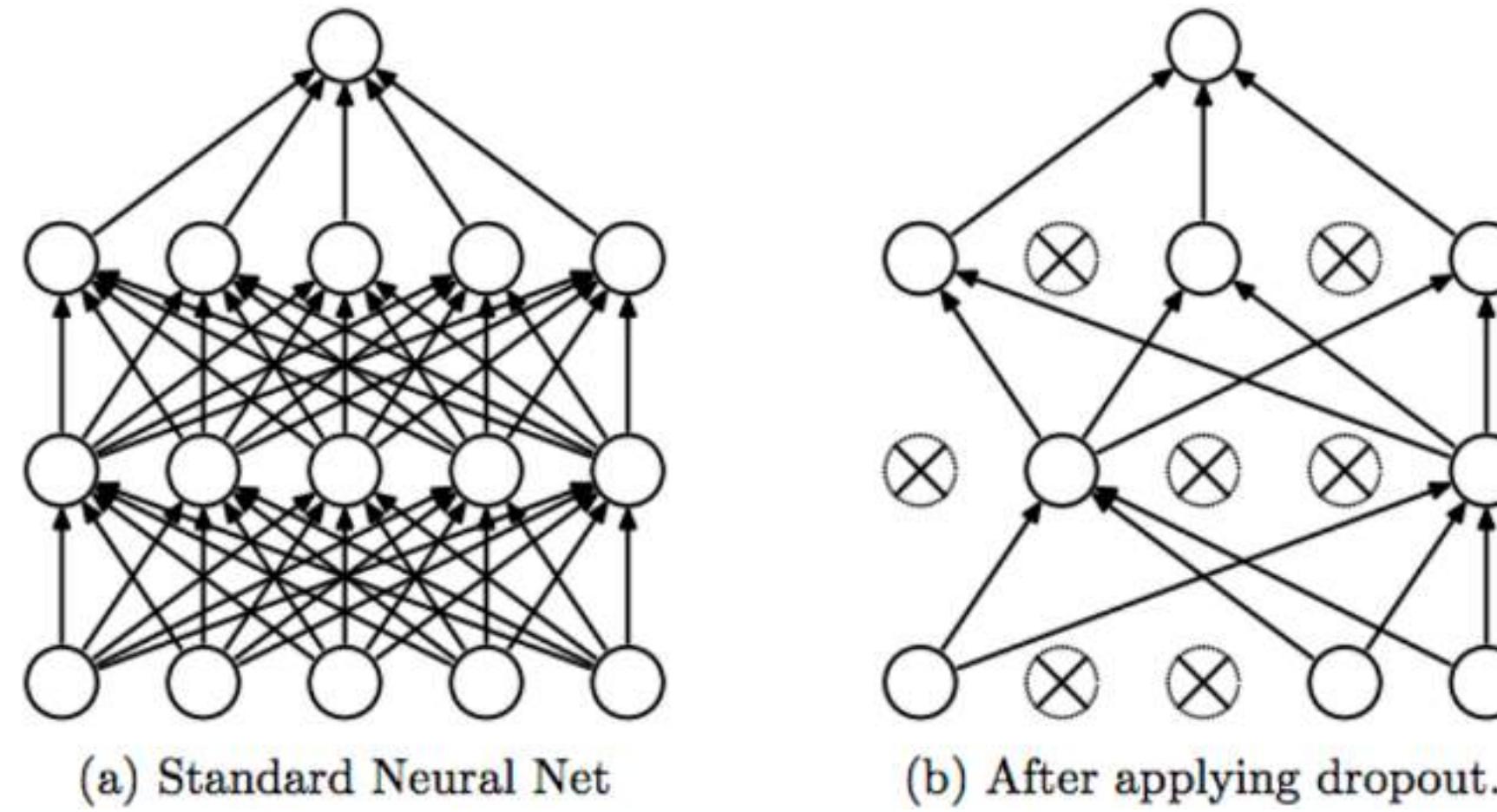
## Dropout

We explore the use of Drop Out as a method reducing overfitting

# Dropout

- Unlike L1 and L2 Normalisation, Dropout doesn't modify the loss function
- It modifies the network itself instead
- It works by randomly dropping out (by setting to zero) a number of nodes (feature maps) of a layer during training

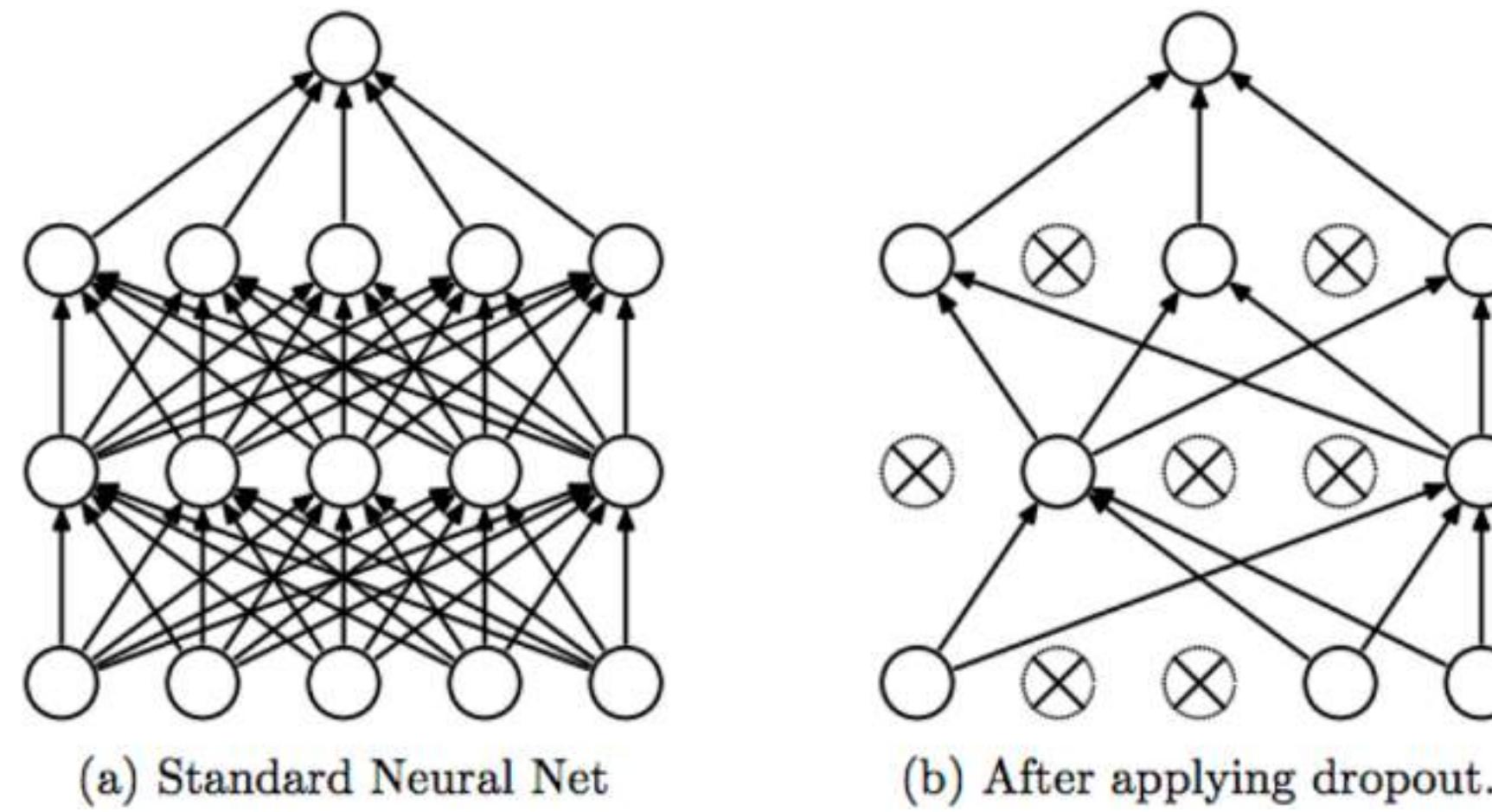
# Dropout Illustrated - Process



Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

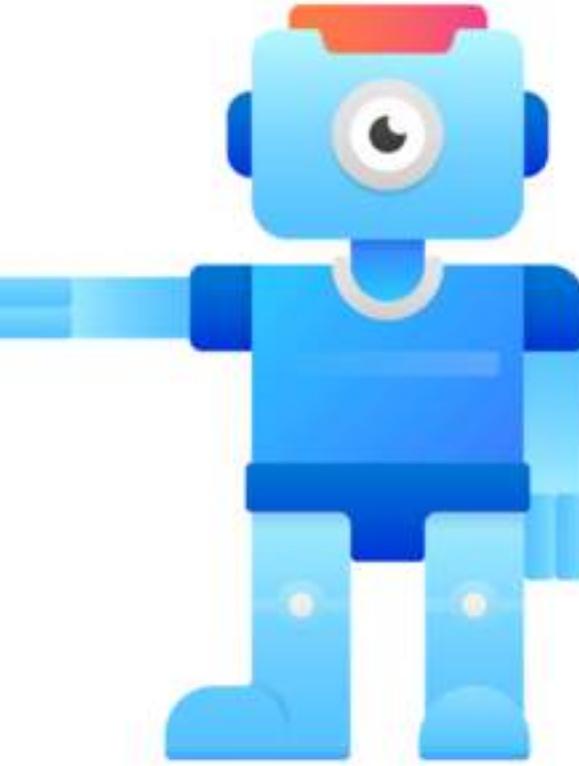
- We start by randomly choosing **half** our nodes to delete/turn off temporarily
- Forward propagate our mini-batch, back propagate to get our gradients for our modified network
- We then restore all nodes and **randomly delete another** set of nodes for the next mini-batch

# Dropout Notes



Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

- The fraction or amount of nodes we drop depends on the **Dropout Rate**
- It forces the network to learn more **robust/reliable** features as it acts like we trained several different networks
- Effectively **doubles** the number of iterations required to converge
- In **Testing**, we use all activations but reduce them a factor  **$p$**  to account for the missing activations during training

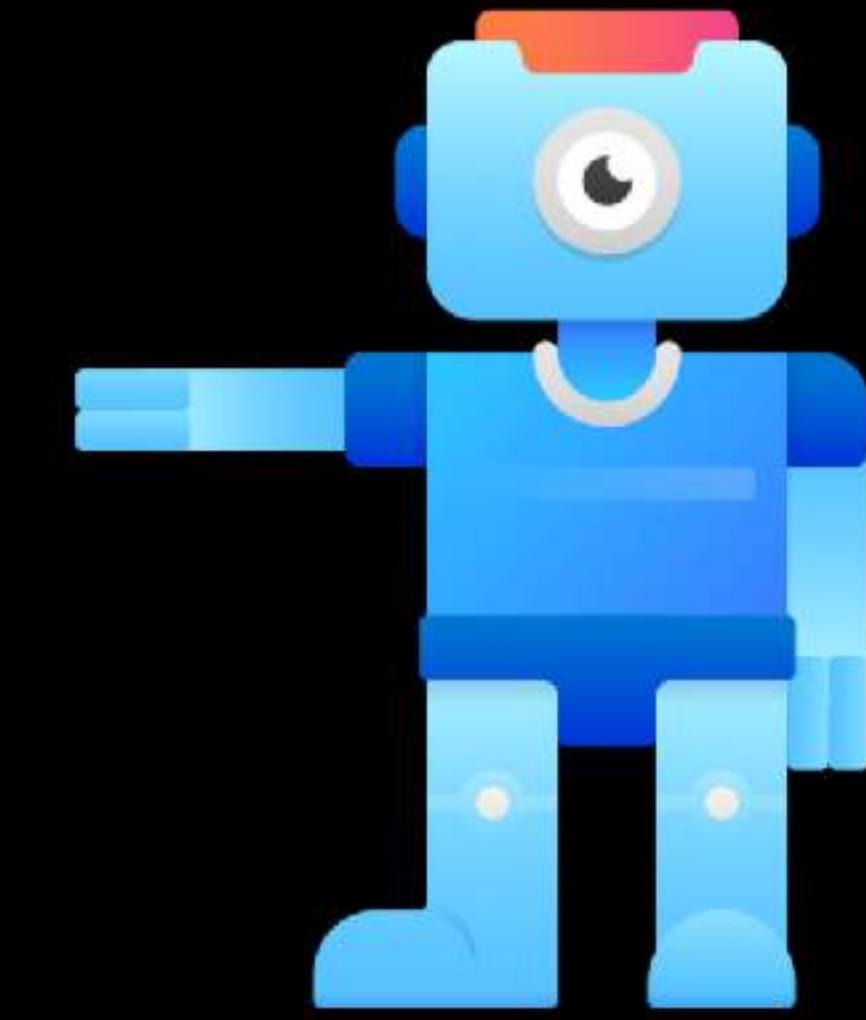


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Data Augmentation**



# MODERN COMPUTER VISION

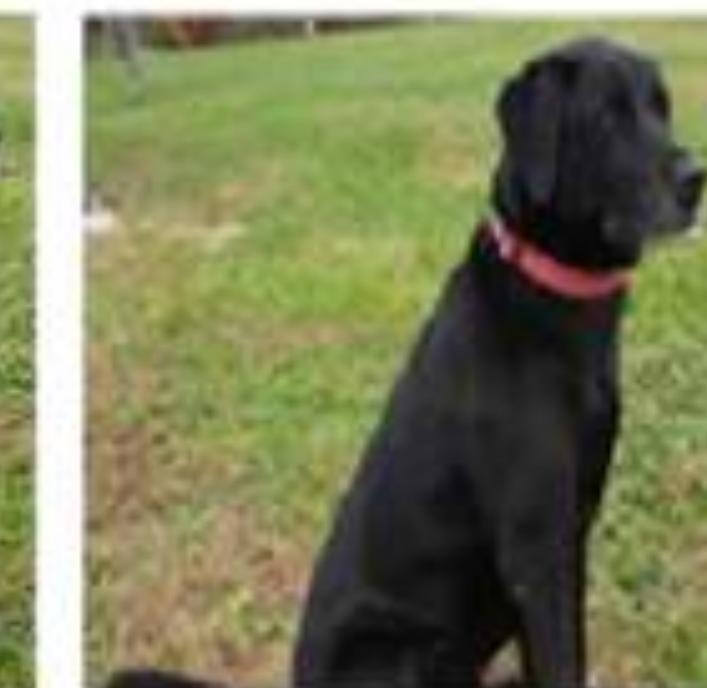
BY RAJEEV RATAN

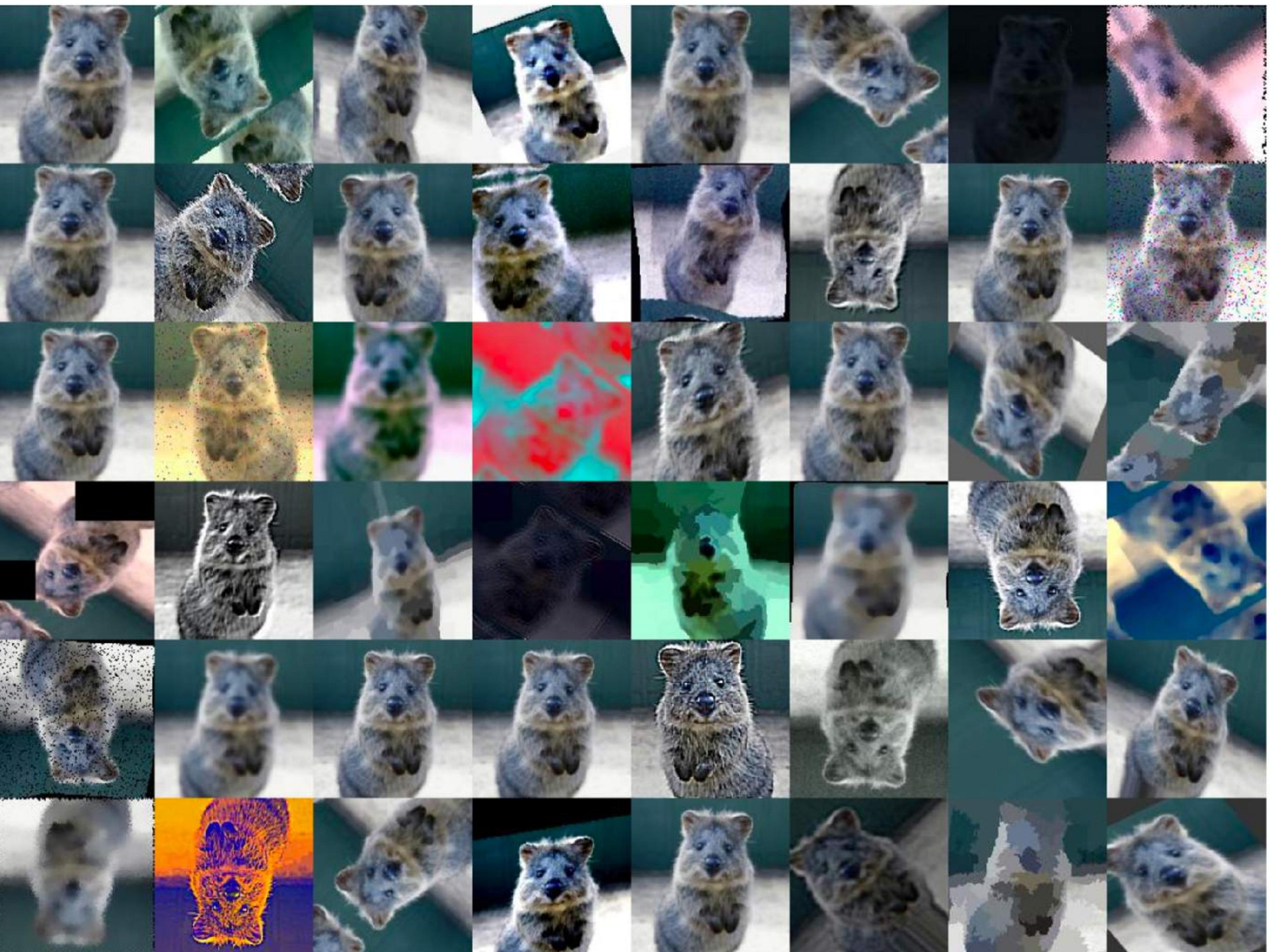
## Data Augmentation

We explore the use of Data Augmentation as a method reducing overfitting

# Data Augmentation

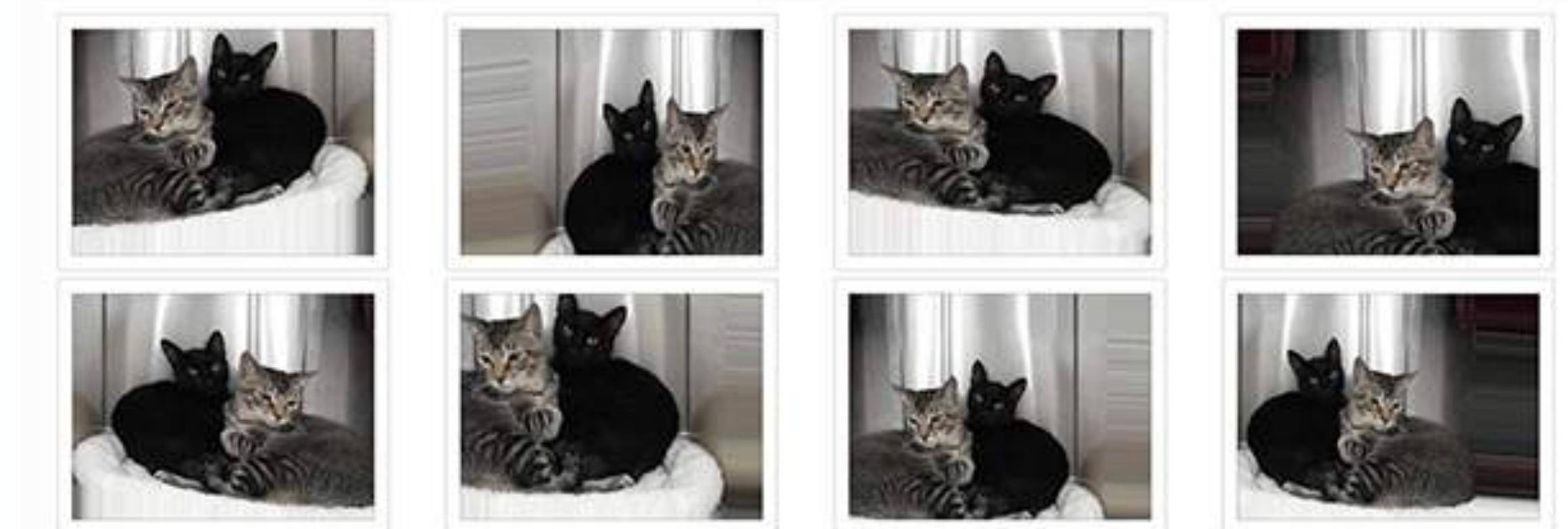
- One of the most effective way of reducing overfitting
- It solves the problem of not having enough data, or enough variation in our dataset
- Image datasets lend themselves easily in applying Data Augmentation





# Data Augmentation Variety

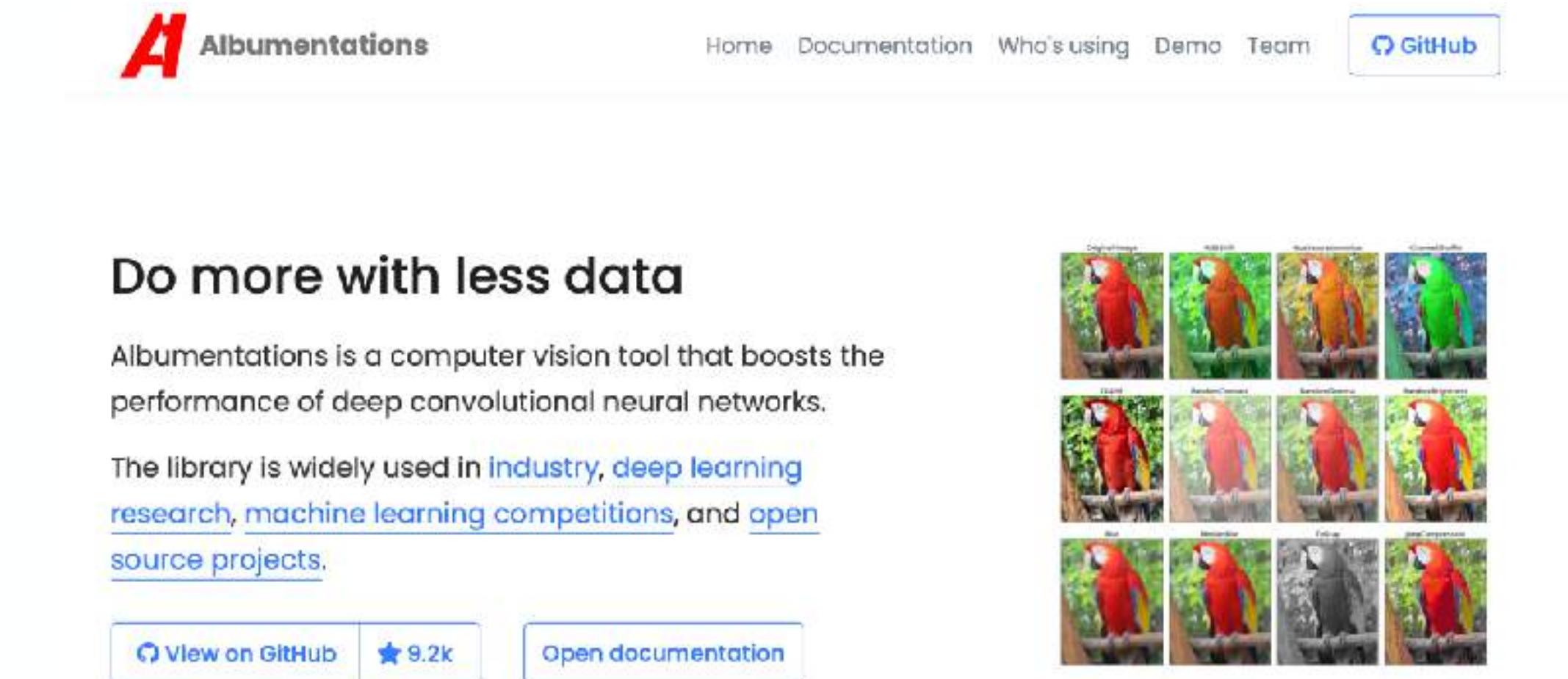
- Both Keras and PyTorch provide Data Augmentation functions that offer many image manipulations such as:
  - Flipping (Horizontally/Vertically)
  - Brightness/Contrast
  - Rotations
  - Zoom
  - Cropping
  - Skew/Shear



<https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>

# More on Data Augmentation

- It's only done on our training dataset
- In most standard implementations, we aren't creating new data, but manipulating our existing input data
- Both Keras and PyTorch provide functions that allow us to implement Data Augmentation.
- Libraries such as Albumentations provide more and new advanced Augmentations



**A** Albumentations

Home Documentation Who's using Demo Team GitHub

**Do more with less data**

Albumentations is a computer vision tool that boosts the performance of deep convolutional neural networks.

The library is widely used in [industry](#), [deep learning research](#), [machine learning competitions](#), and [open source projects](#).

[View on GitHub](#) [9.2k](#) [Open documentation](#)



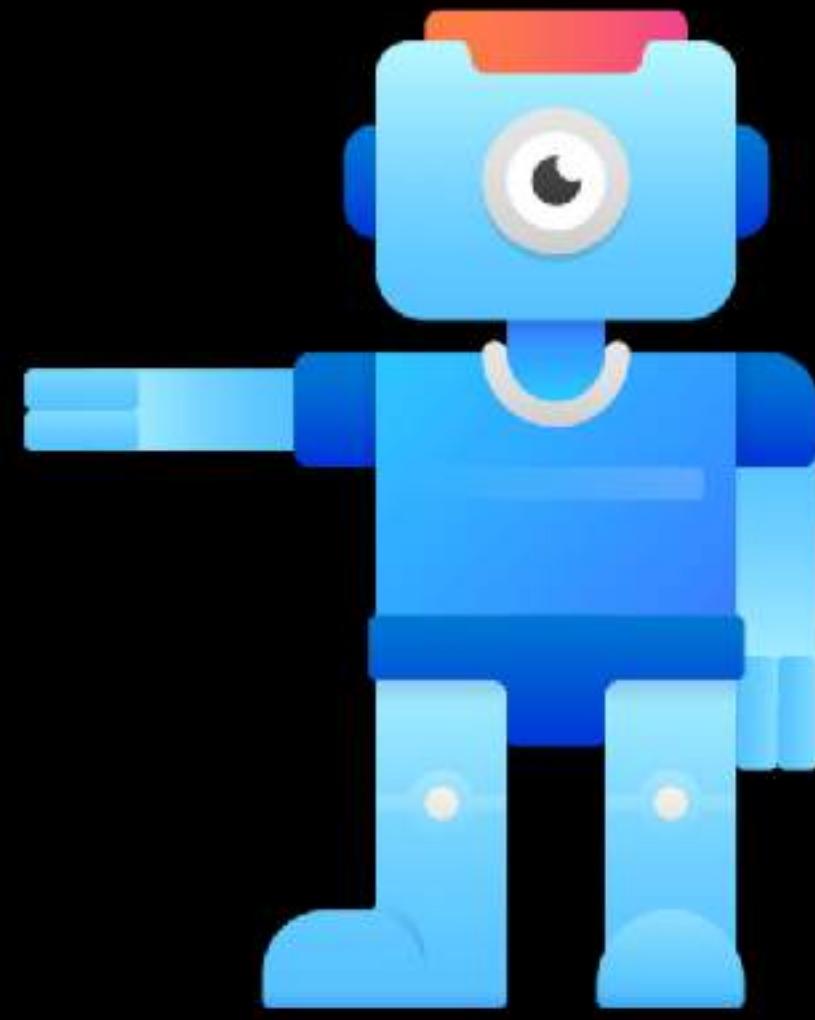


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Early Stopping**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

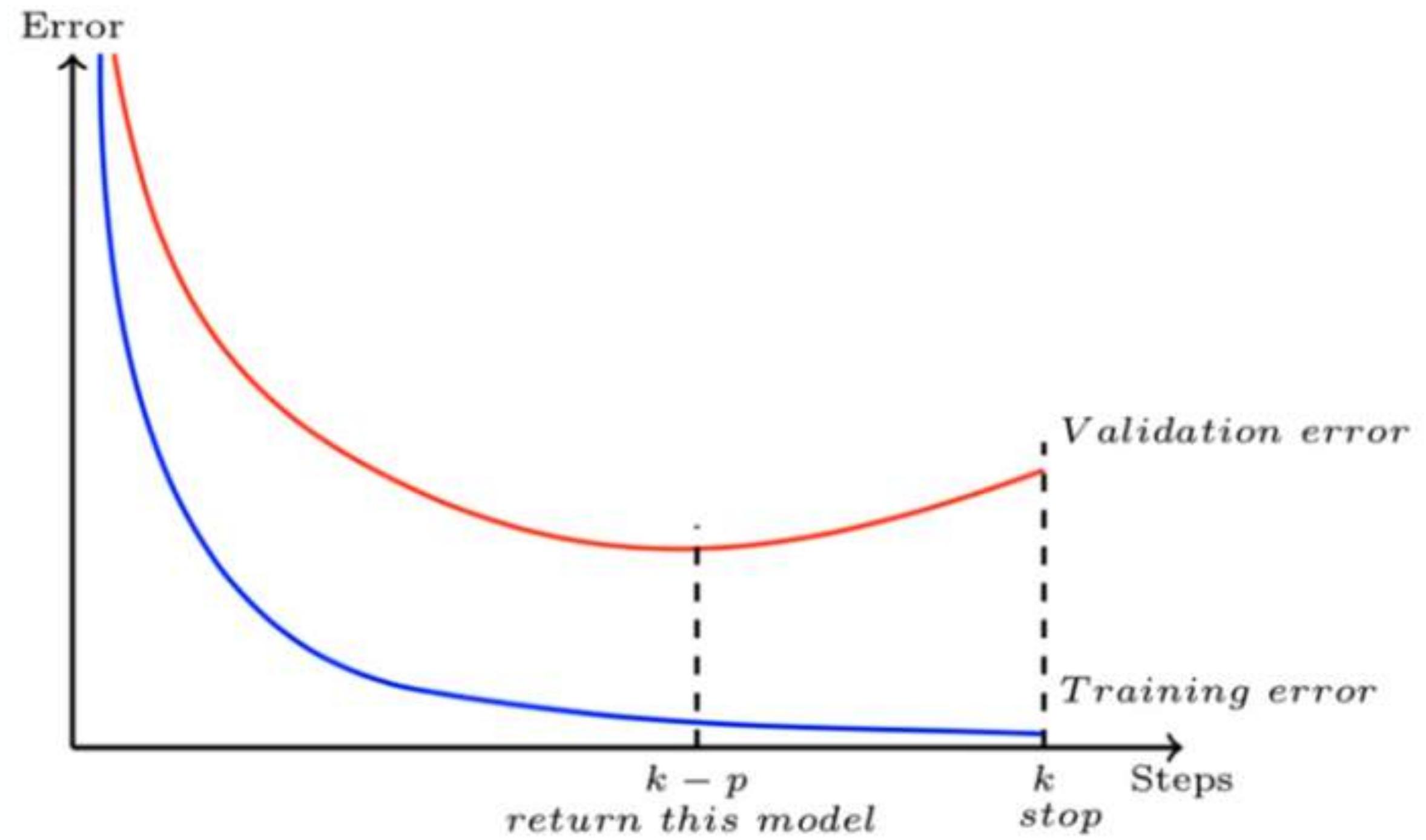
## Early Stopping

We explore the use of Early Stopping as a method reducing overfitting

# The Over Training Problem

- At some point during our training process our validation loss may **stagnate** or **stop decreasing** and sometimes actually start to increase
- At this point, **you should NOT continue training**
- The method by which we detect diminishing returns in training and thus **stop training** our model, is called **Early Stopping**

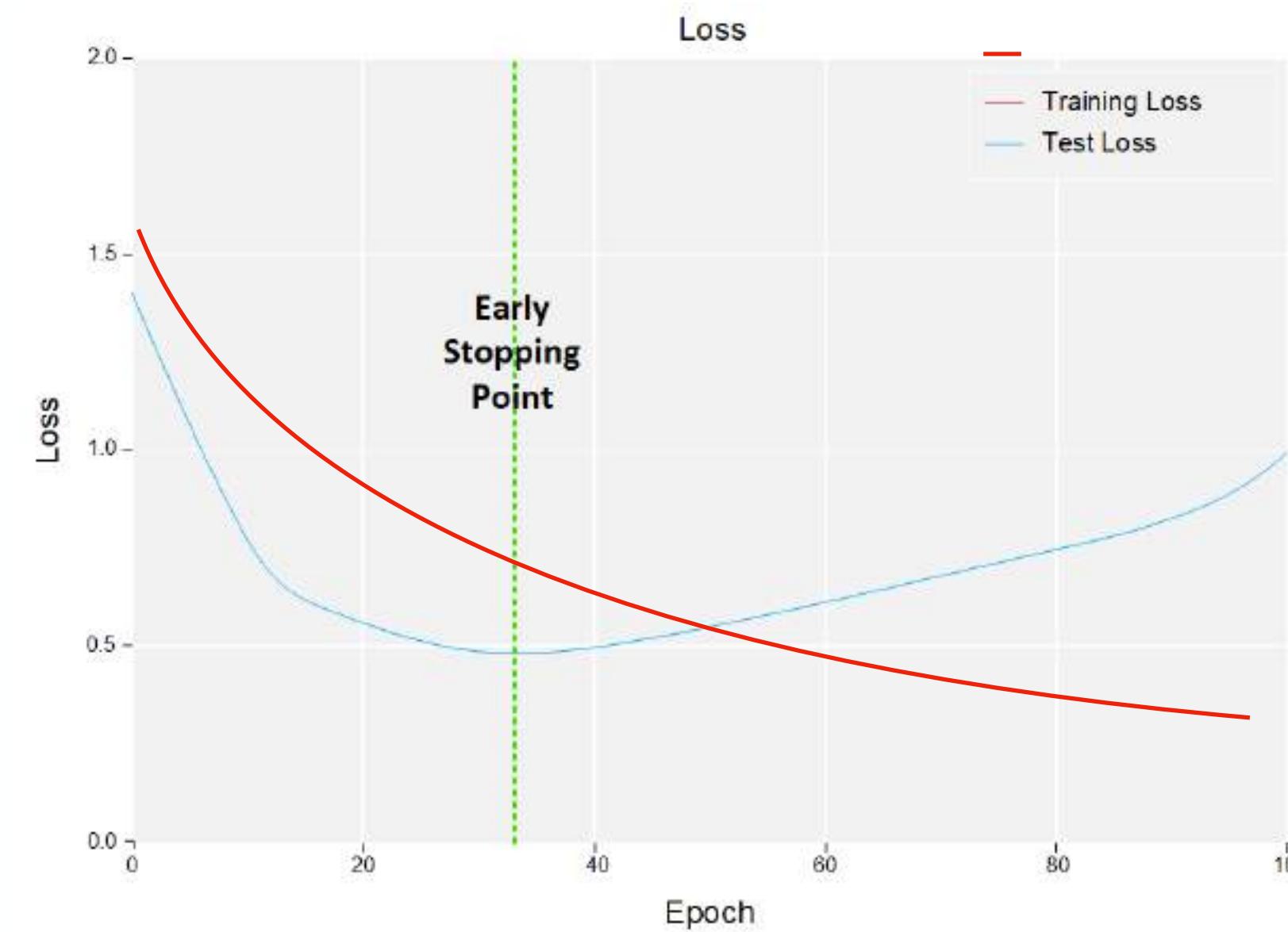
# Early Stopping Illustrated



- Typically, we set some criteria that tells us if our model hasn't reduced its loss after  $x$  amount of Epochs, then stop training.
- Model weights are saved after each epoch so that we can obtain the weights with lowest Test loss

# How Early Stopping Reduces Overfitting

- Early Stopping can be applied manually in PyTorch or by using third party libraries
- Keras supports this feature directly allowing for easy implementation
- We reduce overfitting by ensuring our model doesn't start to learn patterns of noise in our training data.



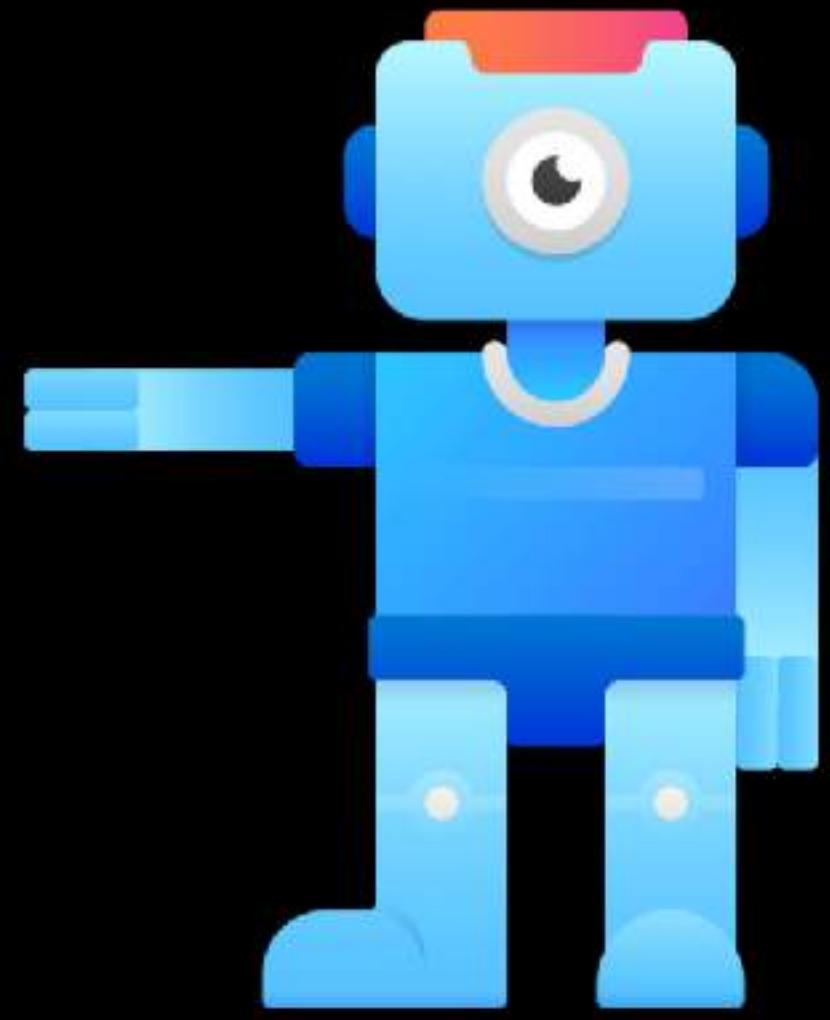


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Batch Normalisation**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

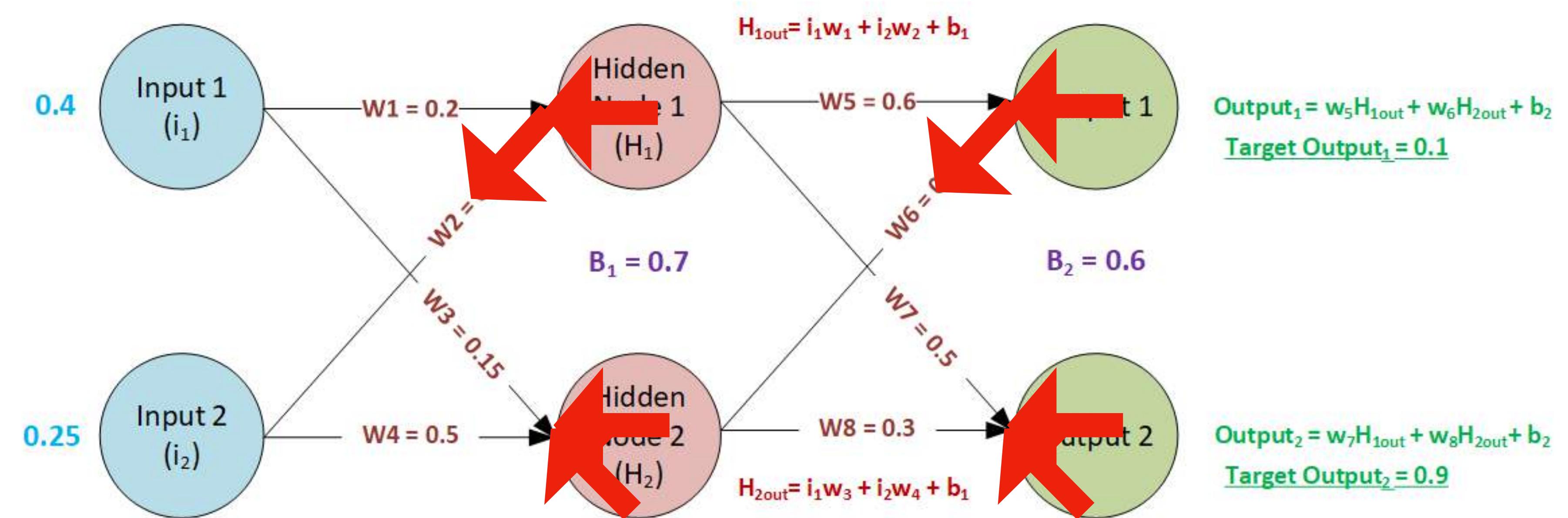
## Batch Normalisation

Another Regularisation Technique that also improves training time

# Deep Learning Problems

- Training deep networks is slow, requires a lot of tweaking for experimentation and lots of data!
- Models are updated layer by layer, backward towards the input.
- And it assumes that the weights it adjusted in the prior layer are fixed.
- *“Very deep models involve the composition of several functions or layers. The gradient tells how to update each parameter, under the assumption that the other layers do not change. In practice, we update all of the layers simultaneously.”* Page 317, Deep Learning, 2016.

# Recall How Back Prop Works



- During training each layer's inputs change as the parameters of the previous layer themselves change
- This results in slower training as it requires smaller learning rates and careful parameter initialisations

# Batch Normalisation

- The Batch Norm technique helps coordinate the update of multiple layers in a model
- It standardises (*rescaling so we have a mean of 0 and standard deviation of 1*) the activations of the prior layer, thus scaling the output of the layer
- It reparametrises the model to make some units always be standardised by definition.
- It reduces internal covariate shift.

The term *internal covariate shift* comes from the paper [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#).

The authors' precise definition is:

We define Internal Covariate Shift as the change in the distribution of network activations due to the change in network parameters during training.

# Implementing BatchNorm

- **Recall** - The output of a Conv layer is 4-dim (or 4-rank) Tensor:
  - **Batch Size** x **Feature Map Height** x **Feature Map Width** x **Channels**
- It calculates the **mean** and **standard deviation** of each input variable to a layer, per mini-batch and uses this to perform the standardisation
- In CNNs filter weights are shared (i.e. the same filter is applied to all parts of the image)
- Therefore, the mean and STD are taken for each filter per mini-batch giving us either 3 means and STDs or 1 (if it's a grayscale image)

# Advice on Using BatchNorm in CNNs

- BatchNorm has been successful in CNNs and in deep models has reduced the time to convergence by half
- In CNNs BatchNorm is best used **between the Conv Layer and the activation function layer (ReLU)**
- When used with **Dropout**, the recommended order is:
  - CONV\_1 -> BatchNorm -> ReLU -> Dropout - CONV\_2

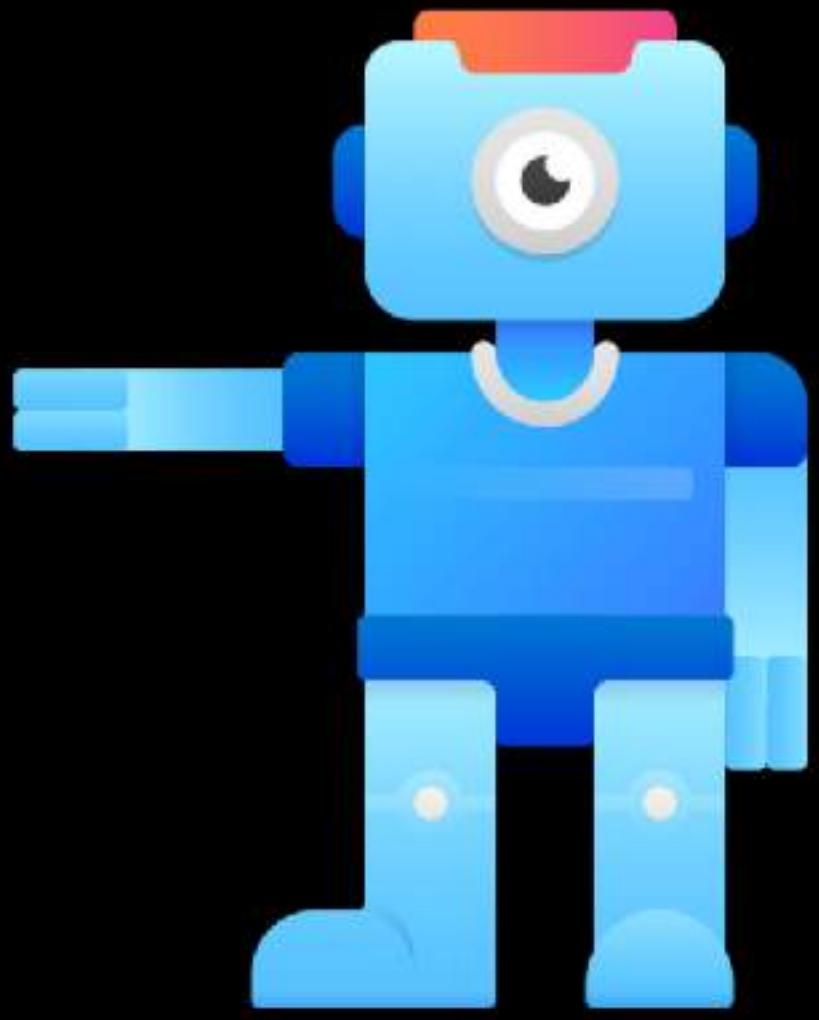


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**When to Use Regularisation**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## When to Use Regularisation

Regularisation isn't always easy to get right, here are some tips!

# Train Without Regularisation First!

- It's good practice to always train your model without regularisation first? Why?
- Sometimes regularisation techniques (or combinations of them) can have detrimental effects on the model performance (e.g. if we used some bad parameter settings)
- Dropout and Batch Norm also increase the convergence time
- It's always good to have a baseline model and introduce different regularisation techniques one by one to assess its impact

# Regularisation Warnings and Tips

- **Dropout** - Don't use it before the final softmax layer
- Regularisation does not offer much help to **smaller less complex** networks
- **Not enough Epochs** - with L2, Dropout and Data Augmentation we do need more epochs to achieve the same performance.
- Data Augmentation and Dropout add additional computations in training and can **slow** it down.
- **Under-fitting** is possible if your L2 weight penalty is too high

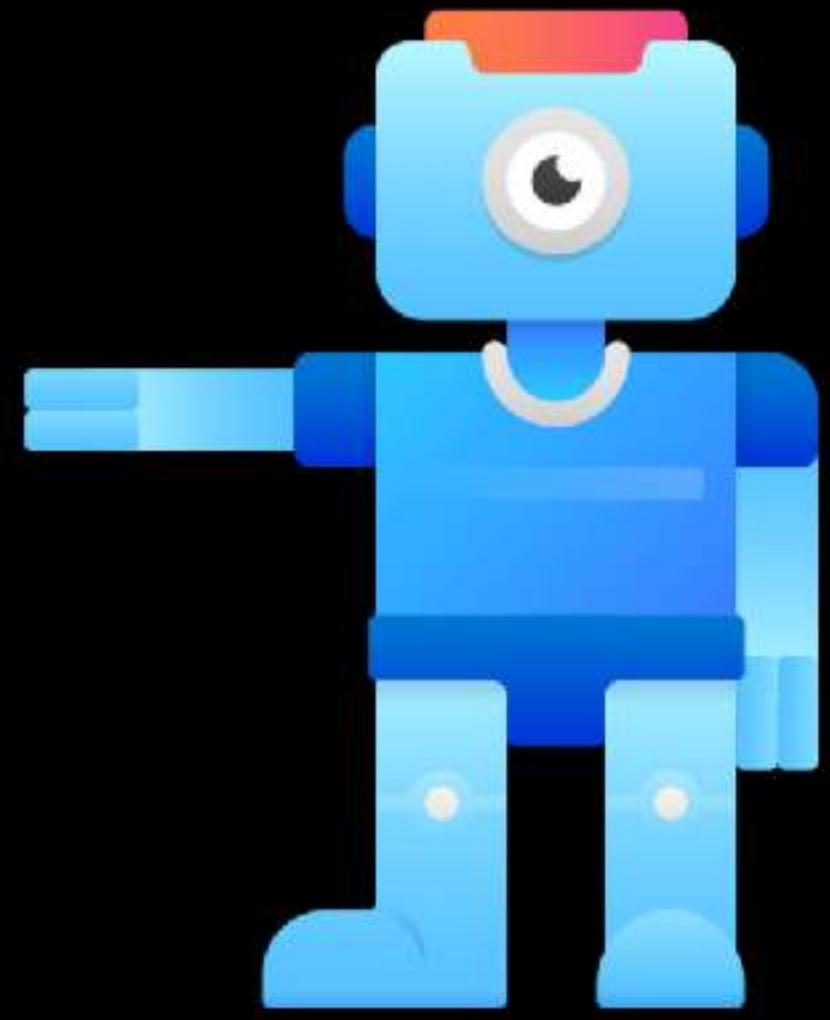


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Implementing Various Regularisation Methods in Keras and PyTorch**



# MODERN COMPUTER VISION

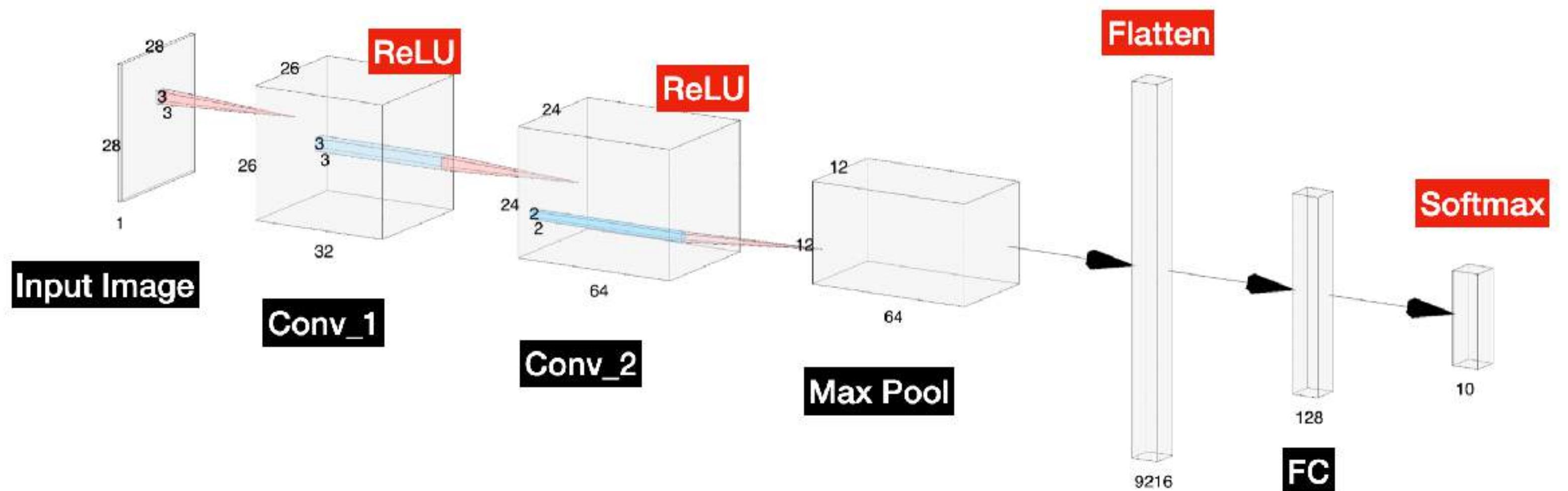
BY RAJEEV RATAN

## Visualising the Filters of Convolution Neural Networks

Understanding what our models learn can help us gain deeper understanding of CNNs

# What do CNN's Learn?

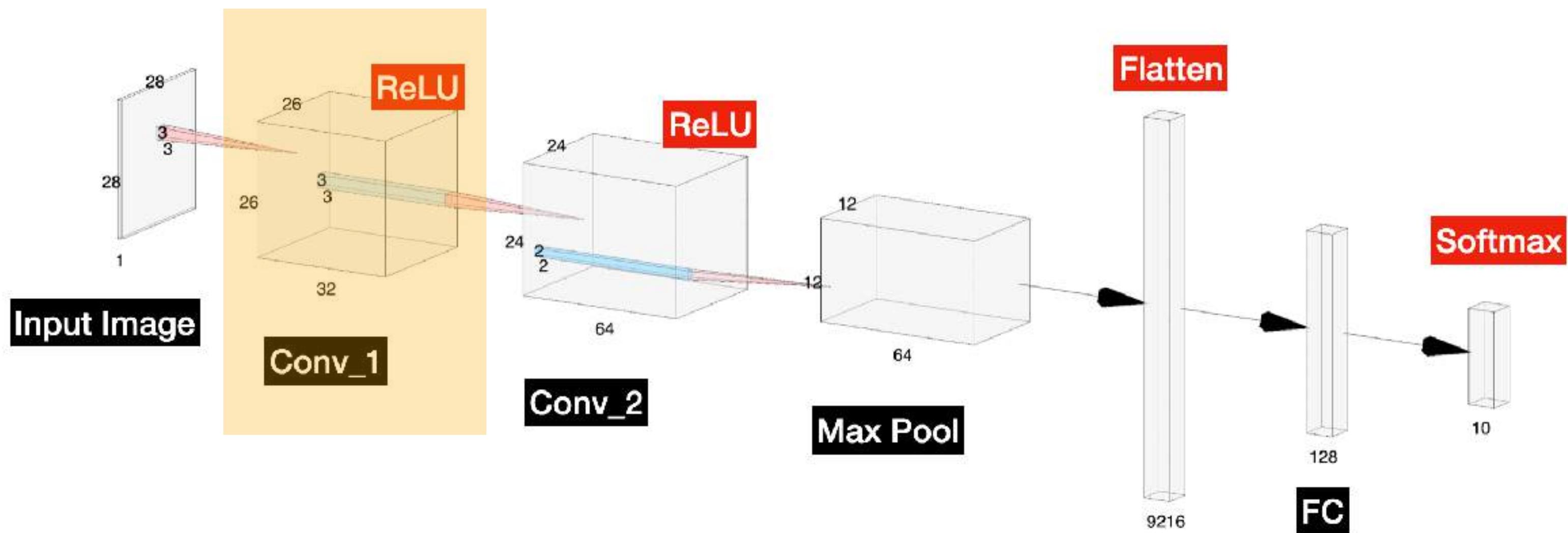
- Learning involves adjusting weights/parameters during training that lead to the lowest loss



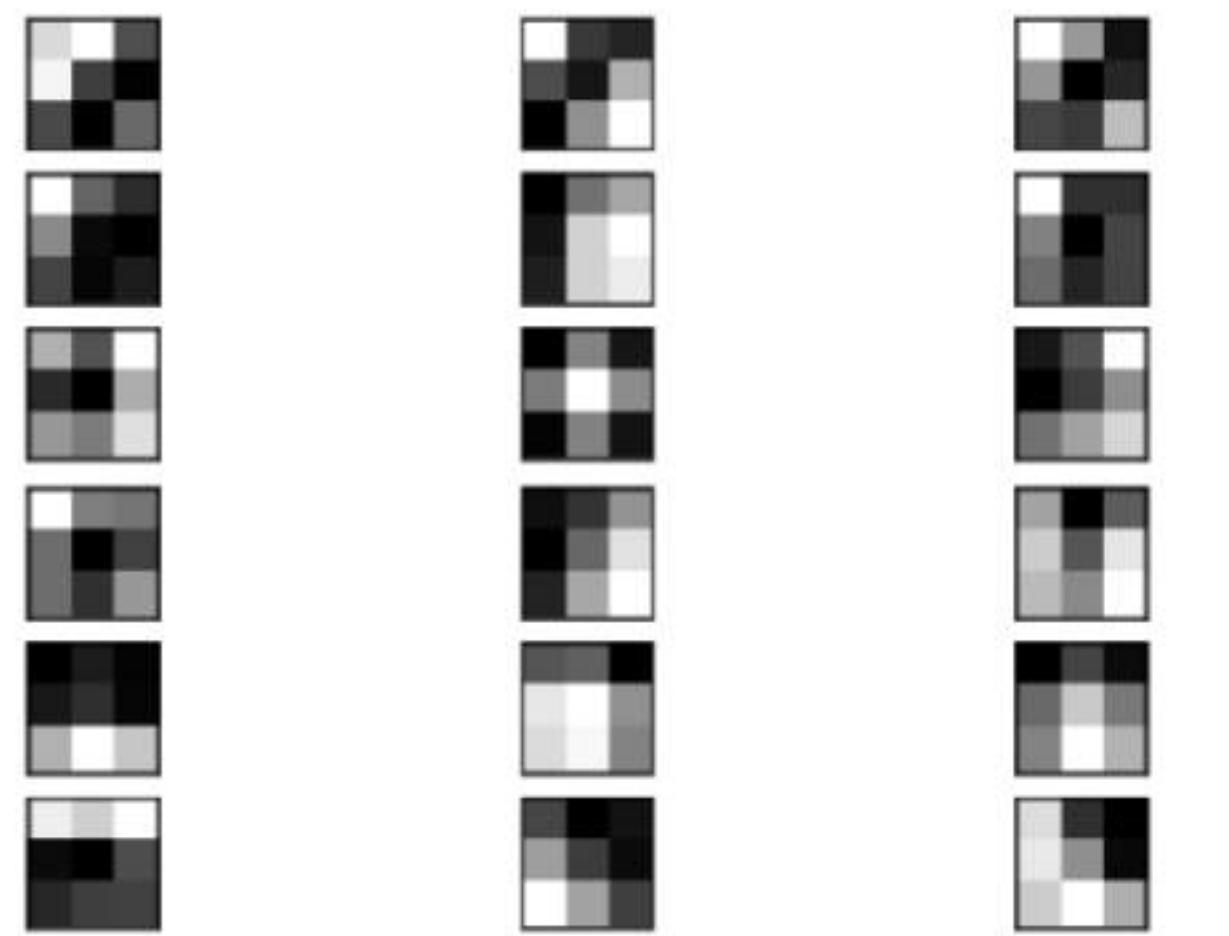
Layer	Parameters
Conv_1 + ReLU	320
Conv_2 + ReLU	18494
Max Pool	0
Flatten	0
FC_1	1,179,776
FC_2 (Output)	1,290
Total	1,199,882

# Exploring a Trained Filter

- In the **Conv\_1** layer we have **32** filters, each of size  **$3 \times 3 \times 1$**  (it would be  $3 \times 3 \times 3$  if using a colour RGB image)
- So if our filter is  $3 \times 3 \times 1$ , that's something can visualise



# What do CNN's Learn?



- If we used matplotlib we can visualise our filters
- **Dark areas correspond to 0** and **white or lighter** areas correspond to **255** (or close to 255)
- **White or lighter** areas in the filter correspond to areas that have **higher weights**

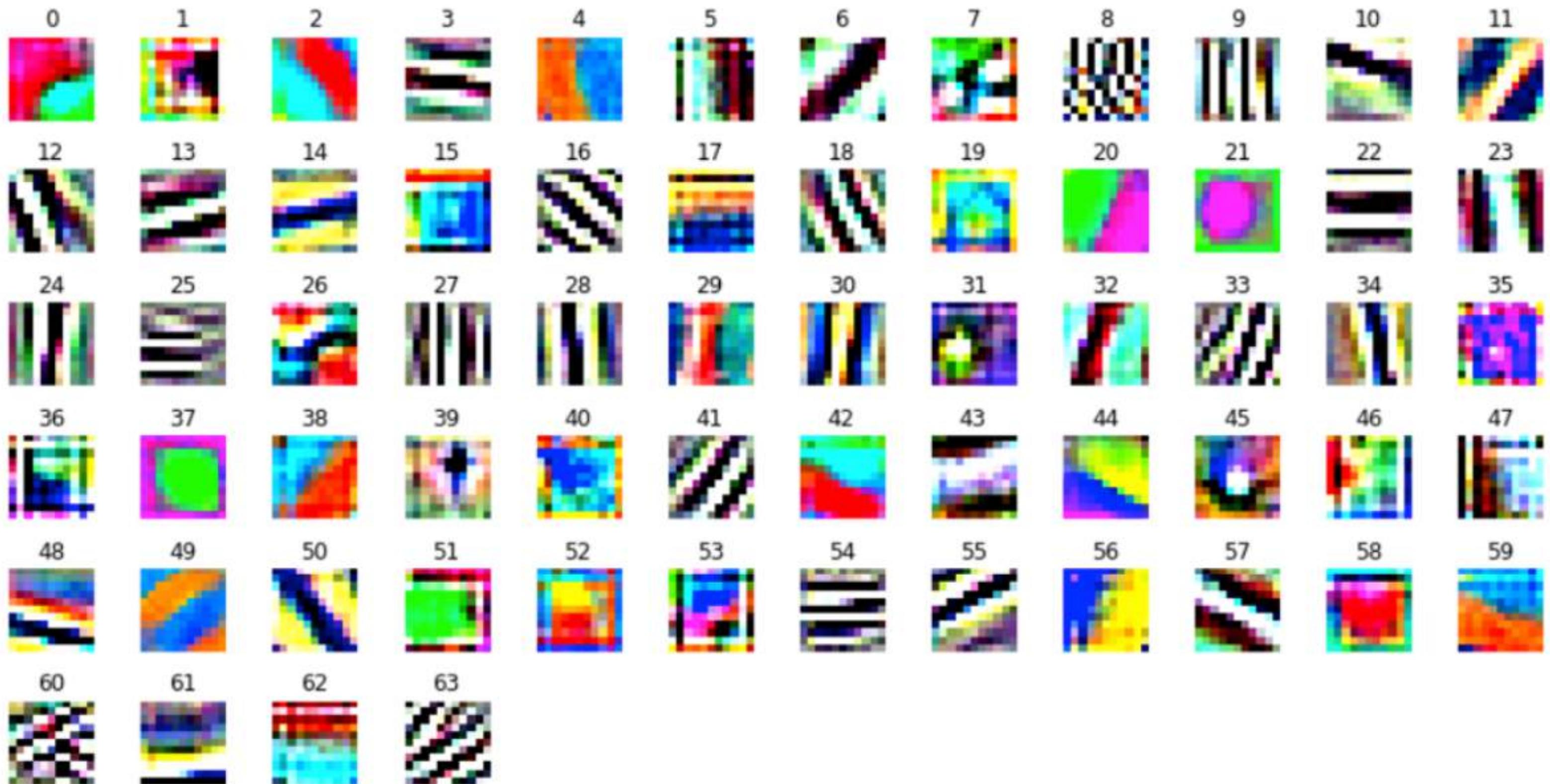
# How do filters work?

$$\begin{array}{|c|c|c|c|c|} \hline
 1 & 0 & 1 & 0 & 1 \\ \hline
 1 & 0 & 0 & 1 & 1 \\ \hline
 0 & 1 & 1 & 0 & 0 \\ \hline
 1 & 0 & 0 & 1 & 0 \\ \hline
 0 & 0 & 1 & 1 & 0 \\ \hline
 \end{array}
 \quad *
 \quad
 \begin{array}{|c|c|c|} \hline
 0 & 1 & 0 \\ \hline
 1 & 0 & -1 \\ \hline
 0 & 1 & 0 \\ \hline
 \end{array}
 \quad =
 \quad
 \begin{array}{|c|c|c|} \hline
 2 & 1 & -1 \\ \hline
 -1 & 1 & 3 \\ \hline
 2 & 1 & 1 \\ \hline
 \end{array}$$

Input Image                          Filter or Kernel                          Output or Feature Map

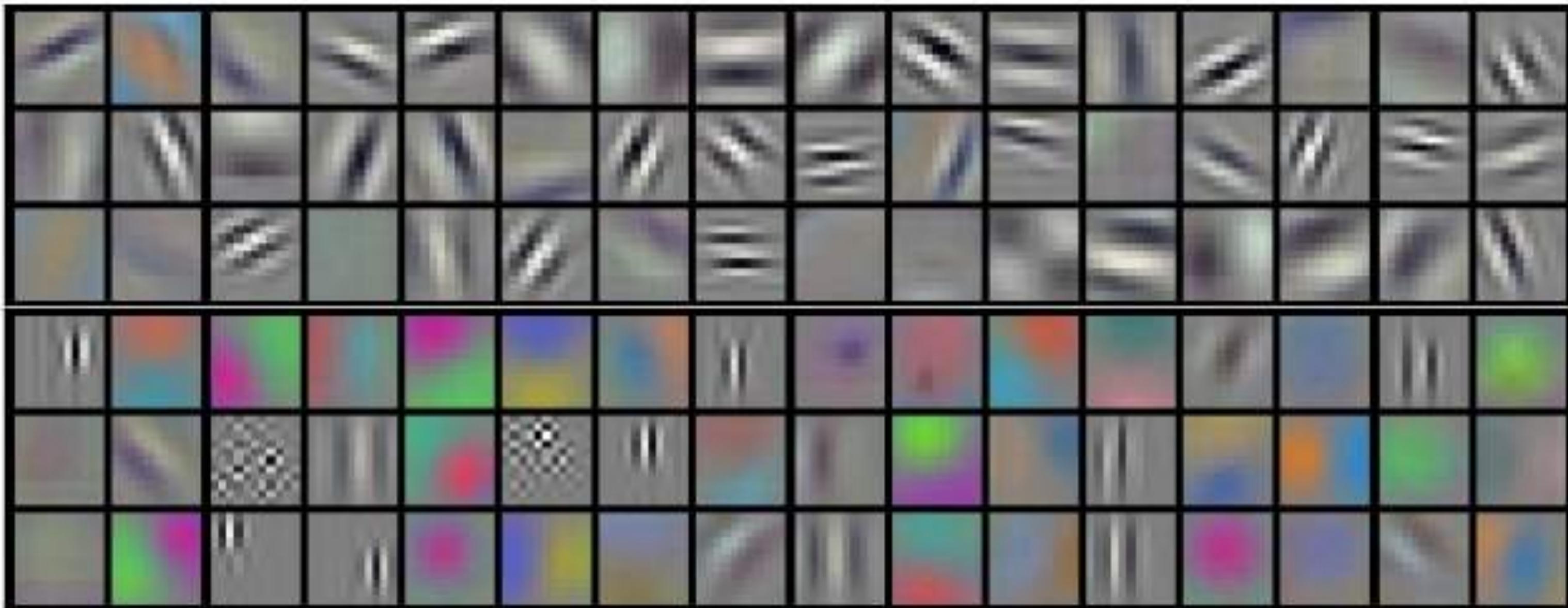
- The convolution operation is the dot product of the **input vector** and **weight vector**
- The dot product between two vectors is **proportional to the angle between vectors**
- The **output is high** when the angle between vectors is  **$0^\circ$**  (vectors are in the same direction)
- Think of our Filter or Kernel as an image, as we **slide that image** across our **input image**, the output is high (or neuron fires) when it **aligns with similar portions of that image**

# Some examples of Filters



Filters from first convolution layer in AlexNet

# Some examples of Filters



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55\*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55\*55 distinct locations in the Conv layer output volume.

# Some examples of Filters

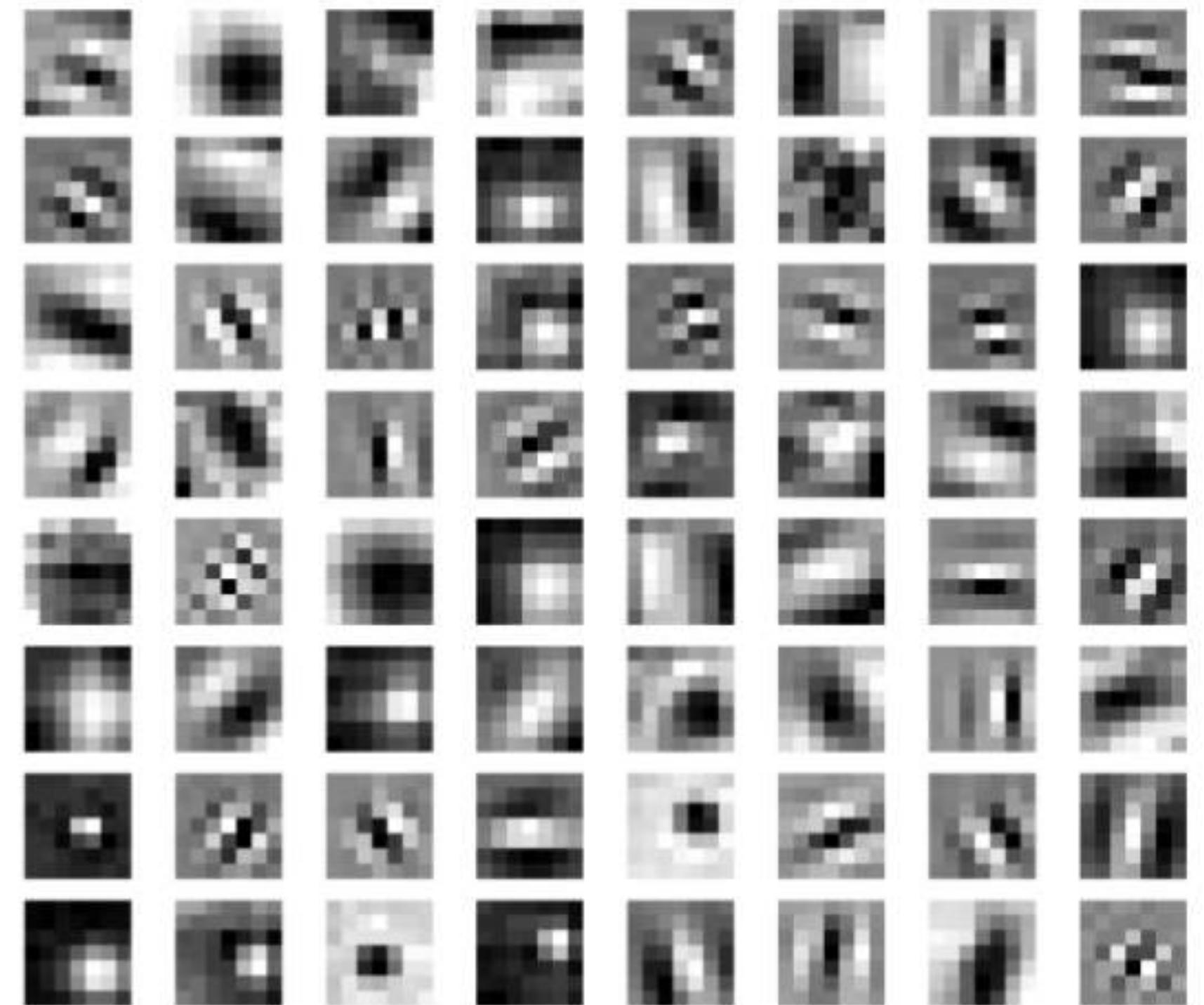
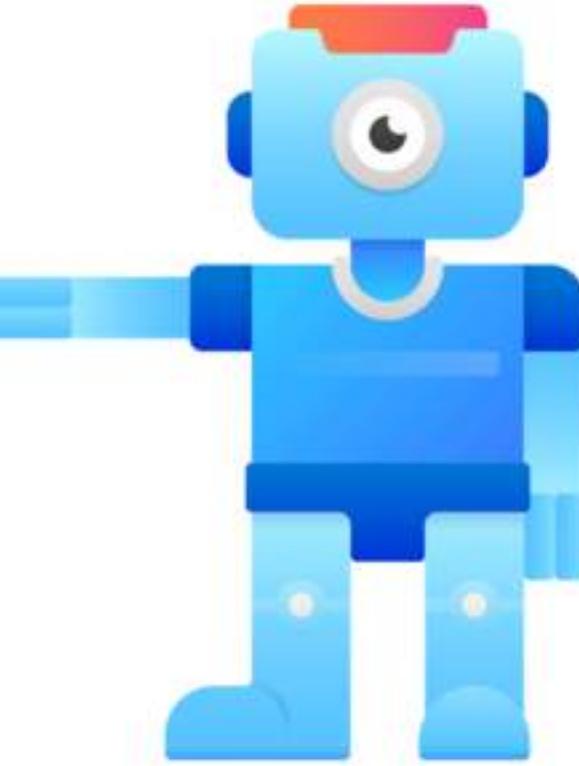


Figure 4. First convolutional layer filter of the ResNet-50 neural network model

# Implementation

- Obtain the weights and bias of a filter (can use `get_weights()` in Keras)
- Normalise weights between 0 and 1
- Use Matplotlib to plot the weight values in 2-D

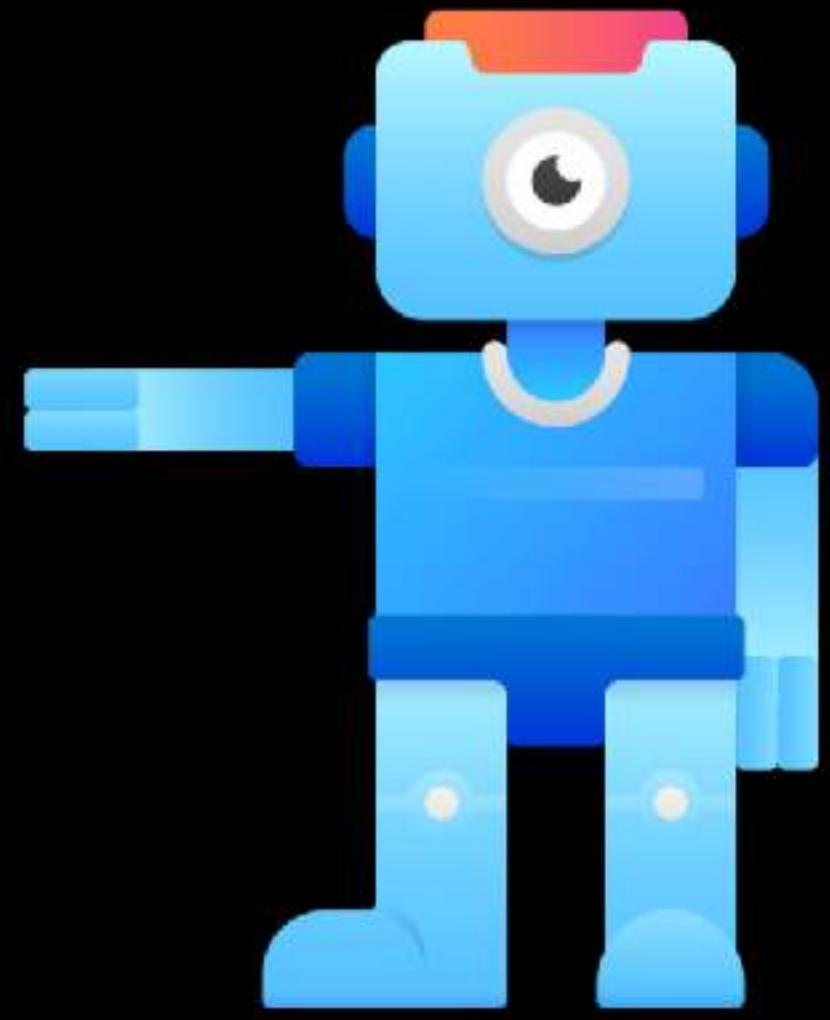


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Filters Activations of Convolution Neural Networks**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Filters Activations of Convolution Neural Networks

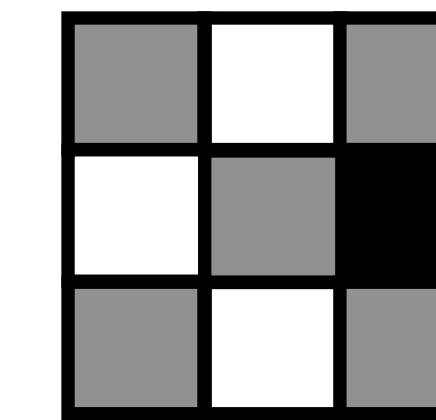
Understanding how our trained filters respond to input data

# Filter Activations

- When an input image is fed into our CNN, filters (some) ‘activate’.
- This is the **output of the ReLU layer**

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input



\*

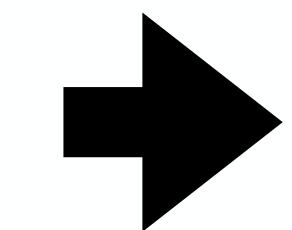
0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

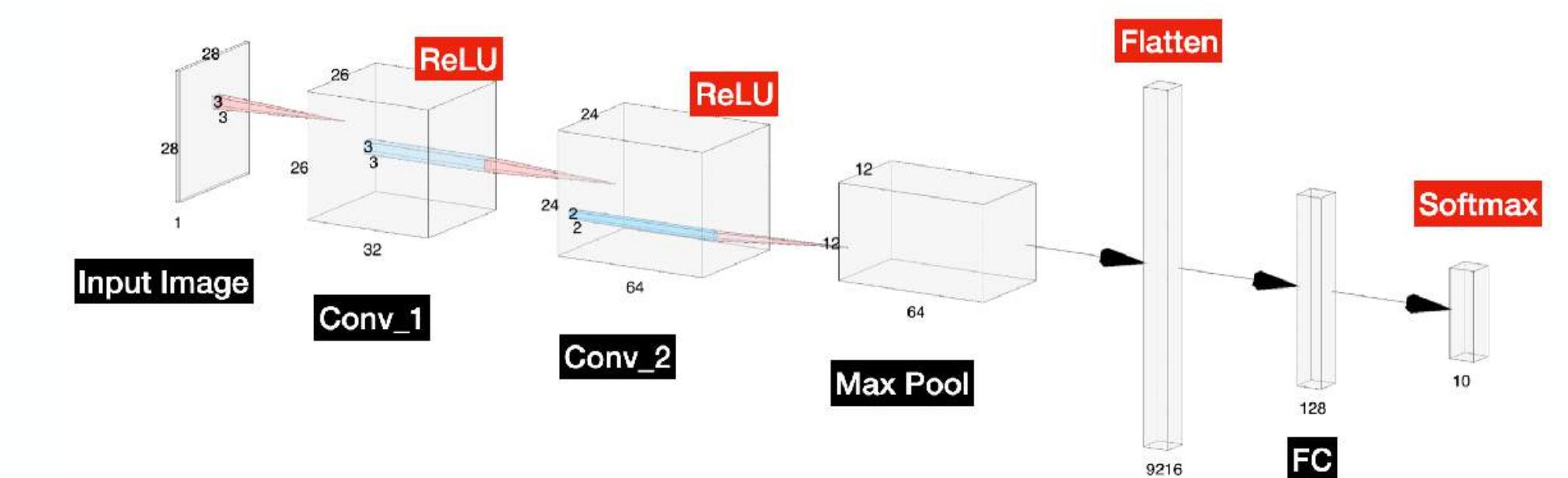
2	1	-1
-1	1	3
2	1	-5

ReLU



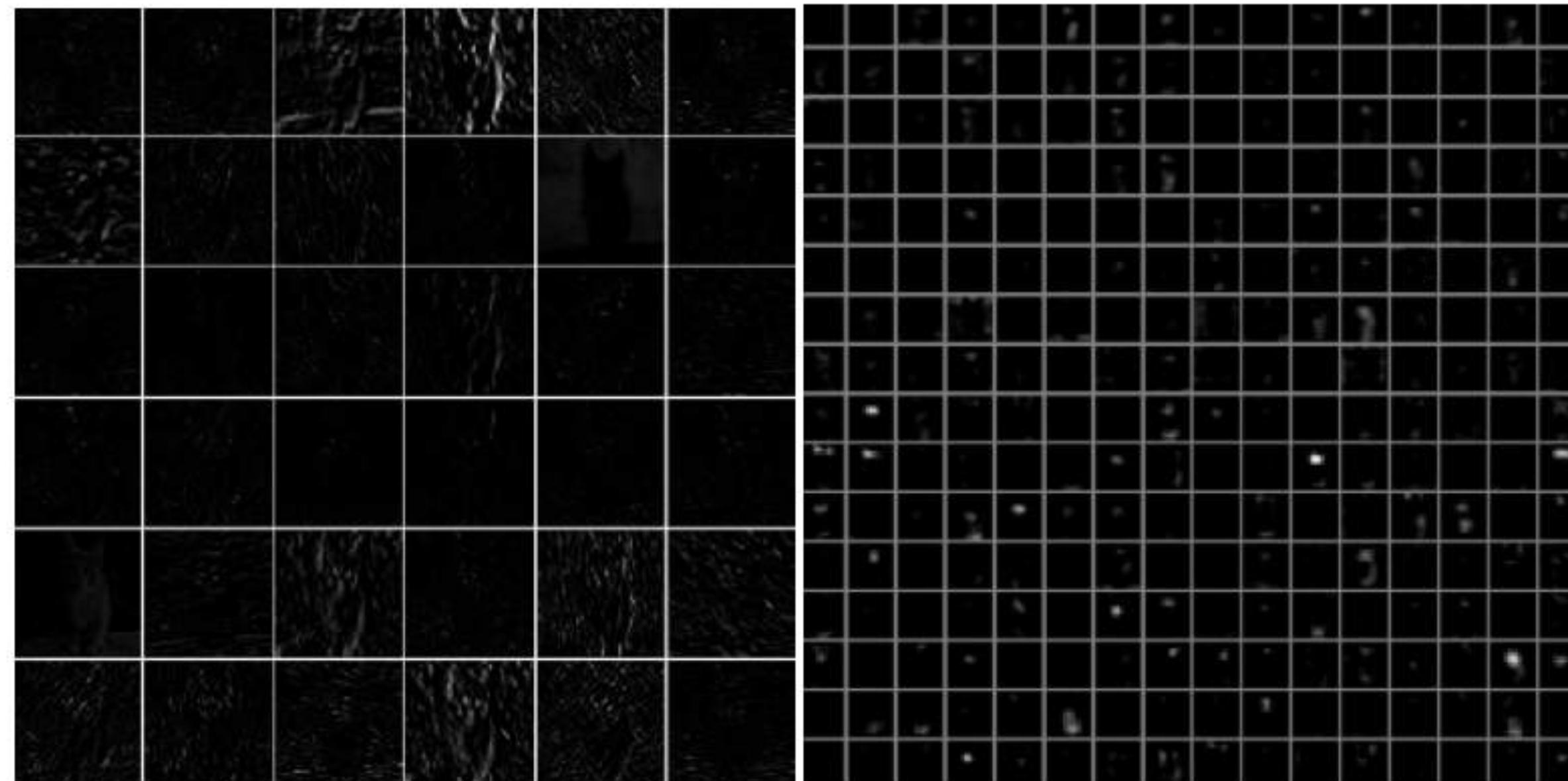
2	1	0
0	1	3
2	1	0

Output or Feature Map



# Visualising Filter Activations

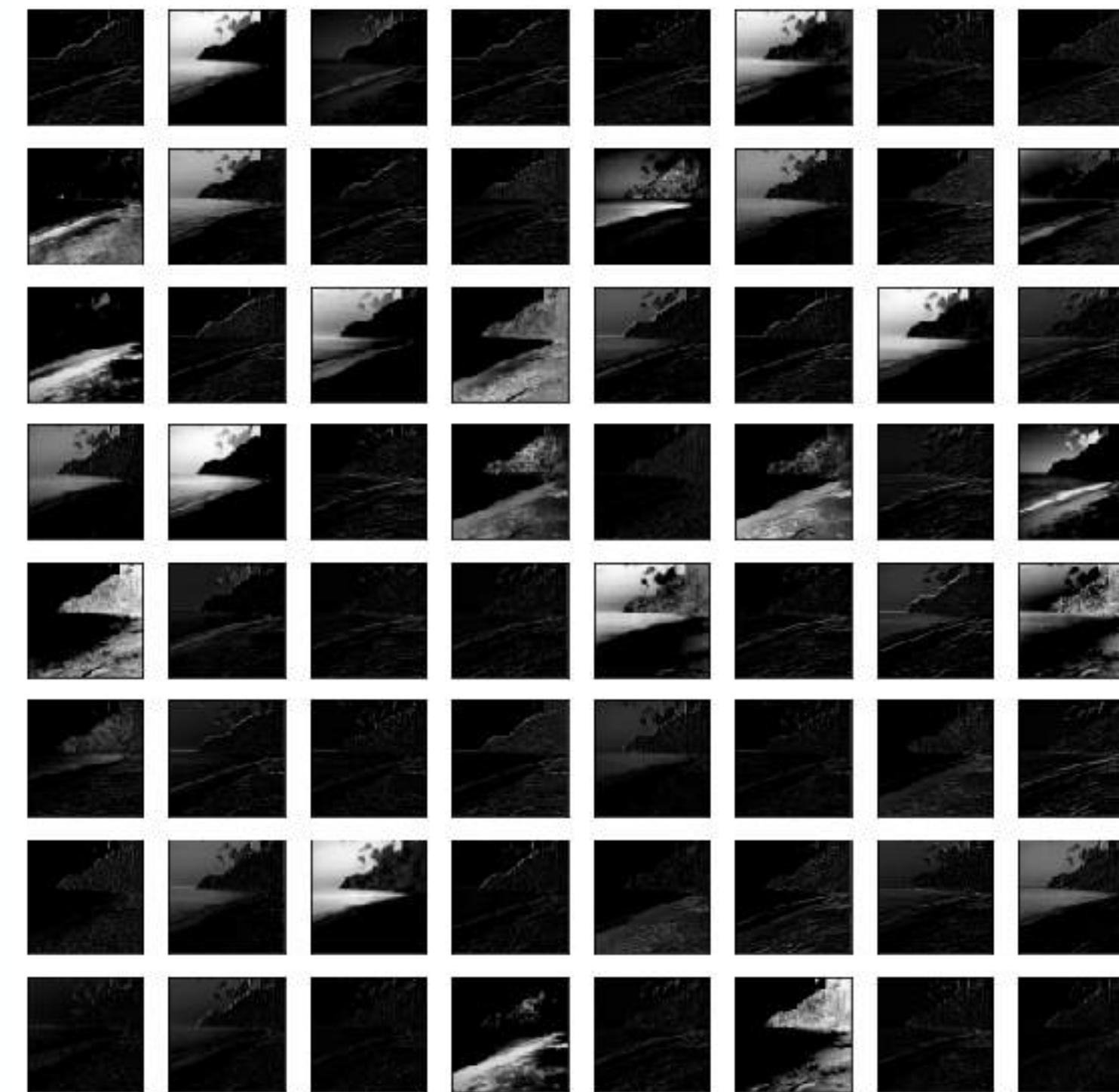
- The outputs (on a trained model) tend appear sparse and localised.
- Often we can spot dead filters (i.e. filters that do turn on for any input)



Typical-looking activations on the first CONV layer (left), and the 5th CONV layer (right) of a trained AlexNet looking at a picture of a cat. Every box shows an activation map corresponding to some filter. Notice that the activations are sparse (most values are zero, in this visualization shown in black) and mostly local.

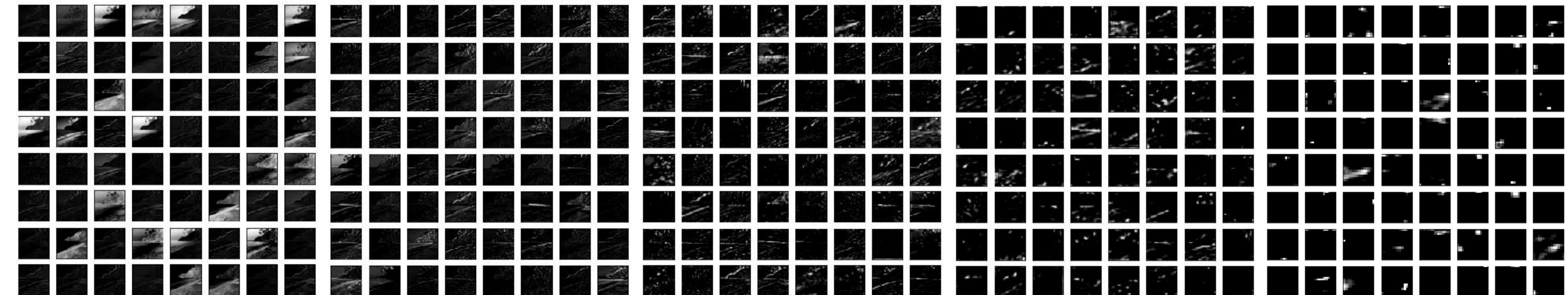
# Visualising Filter Activations vs Filters

- Previously we just looked at the raw weights of a trained CNN Filter
- Here (Filter Activations) we looked at the output generated by applying a Filter to an input image

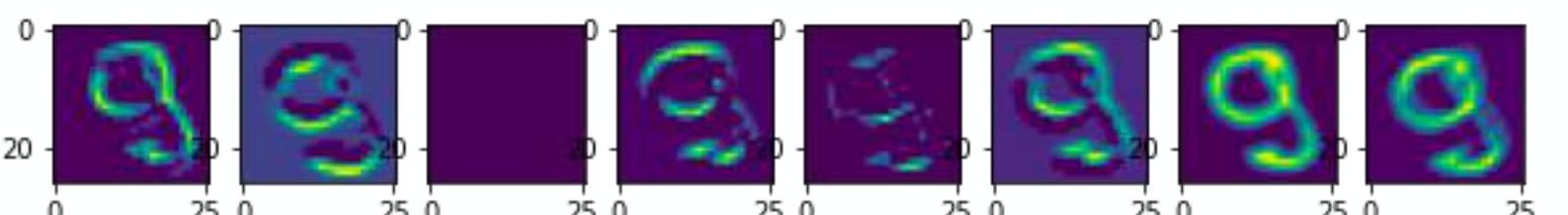
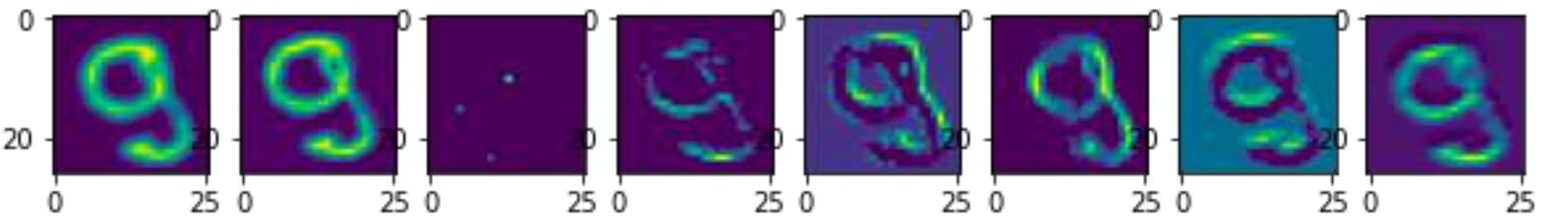
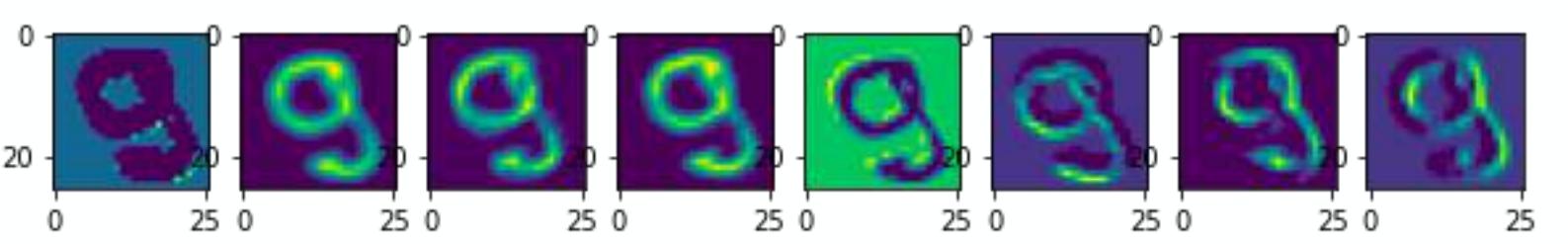
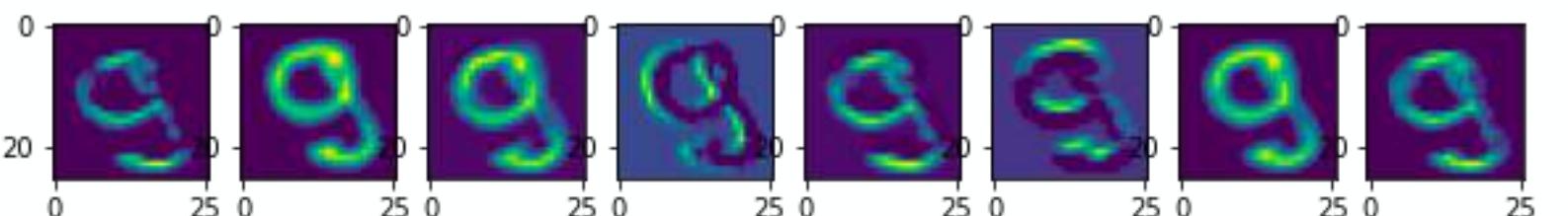
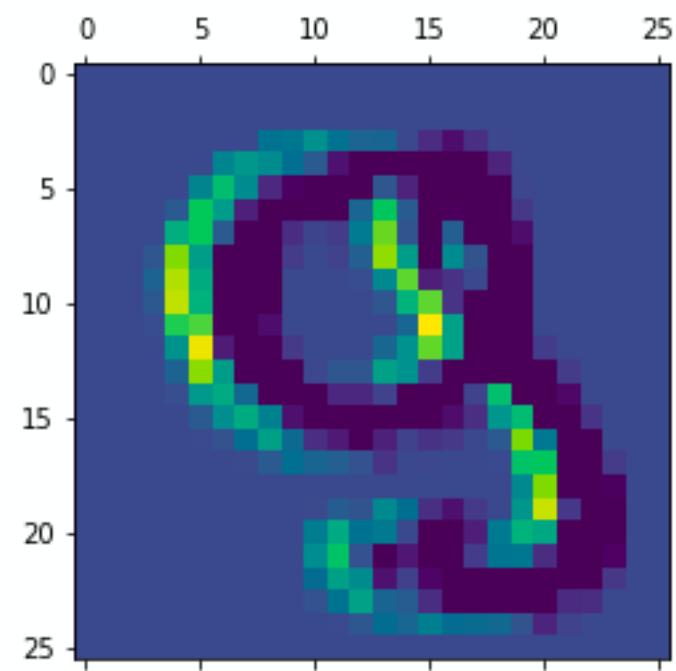


# Visualising Filter Activations vs Filters

- We can see as we progress from the early layers (left) to the deeper layers (right) our feature maps show less detail
- This shows that the ‘image’ dimensions get smaller as the progress through the CNN (most cases) and that the deeper Conv layers are activated by combinations of lower layers.
- That enables them (upper layers) to learn more complex patterns (a combo of lower filters are needed to activate an upper layer)

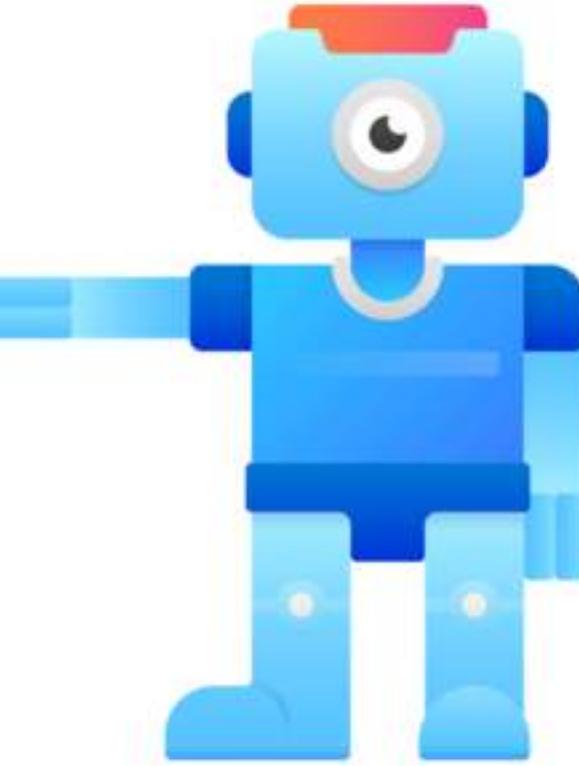


# Example



# Implementation

- Create a new model that takes an image as the input and outputs only the feature maps.
- Load our image and normalise it
- Propagate our input to our new model
- Extract the feature map response we want and use matplotlib to visualise it

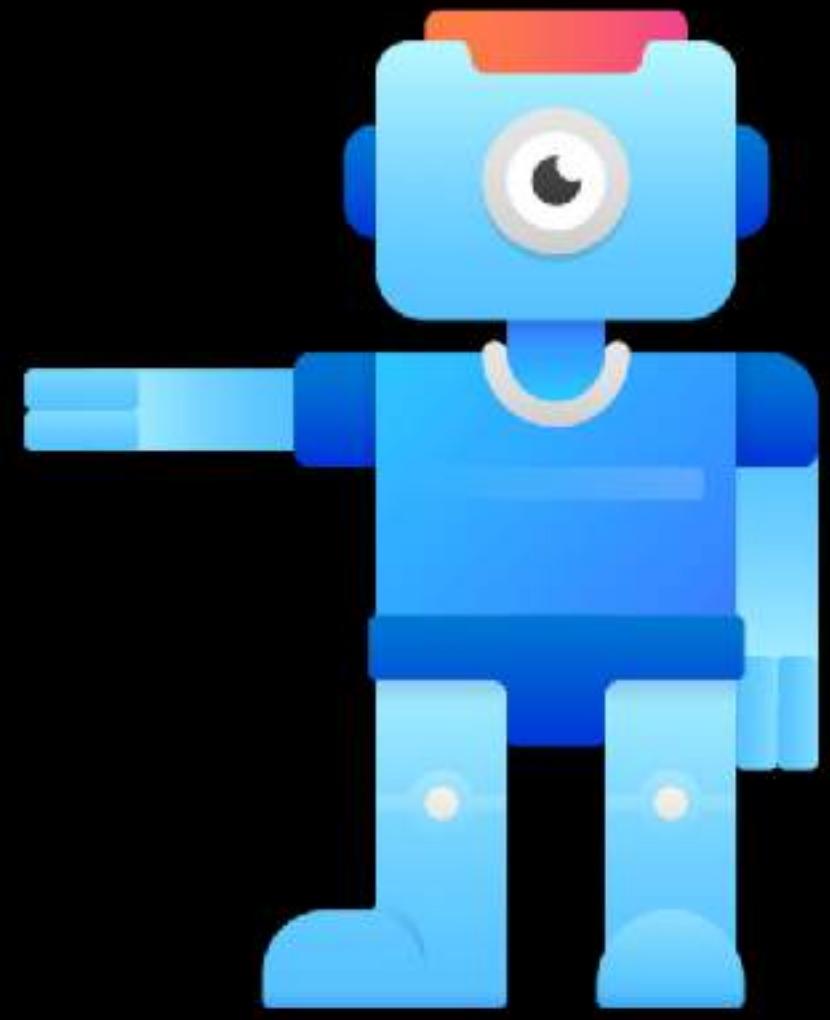


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

## Maximising Filters



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Maximising Filters

Understanding how what makes our filter trigger or fire

# What Inputs Maximise Our Filter?

- To truly understand what our filter is looking for we need to find the input that results in that filter attaining the maximum ReLU output possible.

$$\begin{array}{c}
 \begin{array}{|c|c|c|c|c|} \hline
 1 & 0 & 1 & 0 & 1 \\ \hline
 1 & 0 & 0 & 1 & 1 \\ \hline
 0 & 1 & 1 & 0 & 0 \\ \hline
 1 & 0 & 0 & 1 & 0 \\ \hline
 0 & 0 & 1 & 1 & 0 \\ \hline
 \end{array} & \times & \begin{array}{|c|c|c|} \hline
 0 & 1 & 0 \\ \hline
 1 & 0 & -1 \\ \hline
 0 & 1 & 0 \\ \hline
 \end{array} & = & \begin{array}{|c|c|c|} \hline
 2 & 1 & -1 \\ \hline
 -1 & 1 & 3 \\ \hline
 2 & 1 & -5 \\ \hline
 \end{array} & \xrightarrow{\text{ReLU}} & \begin{array}{|c|c|c|} \hline
 2 & 1 & 0 \\ \hline
 0 & 1 & 3 \\ \hline
 2 & 1 & 0 \\ \hline
 \end{array}
 \end{array}$$

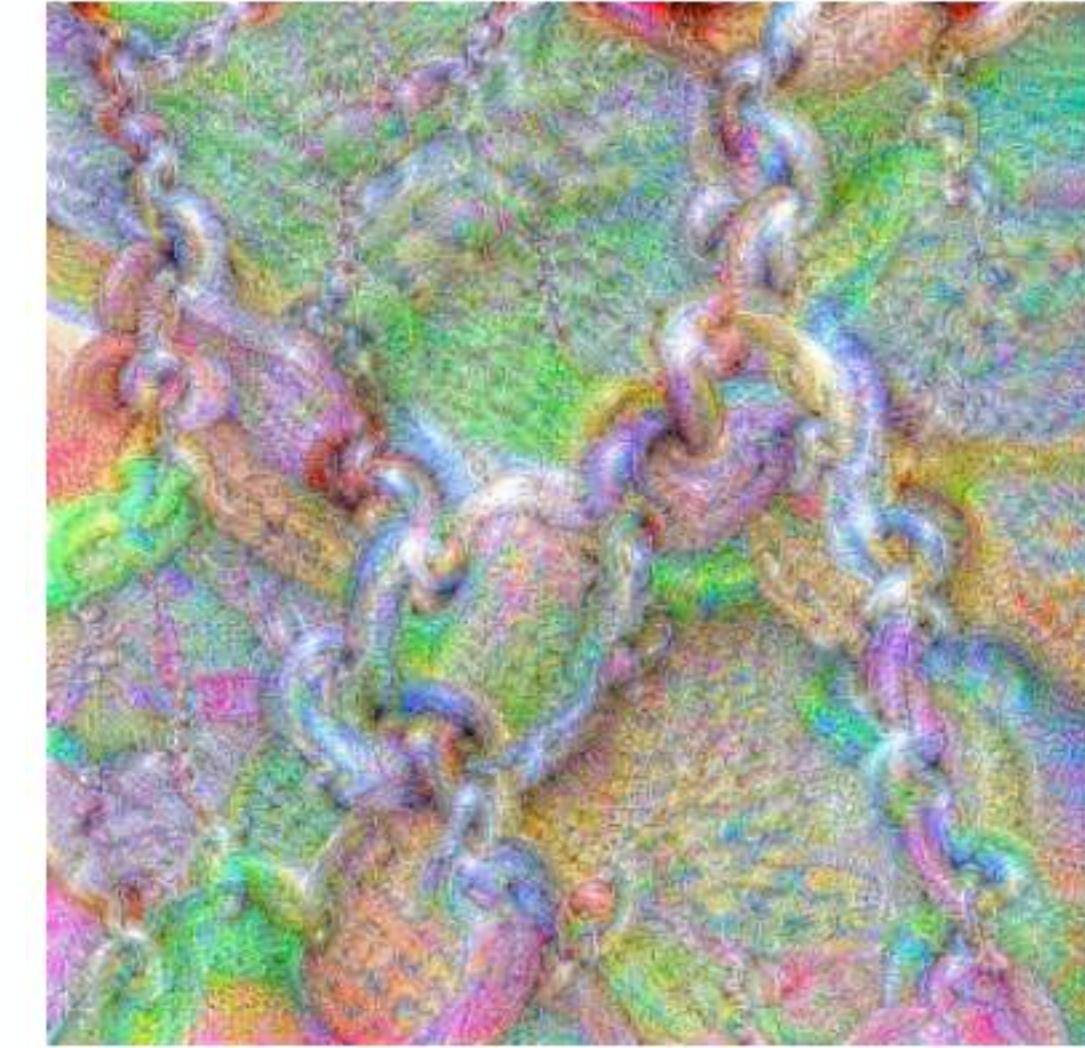
Input                          Filter or Kernel                          Output or Feature Map

# What Inputs Maximise Our Filter?

- Suppose we had a filter that responded maximally to chains?



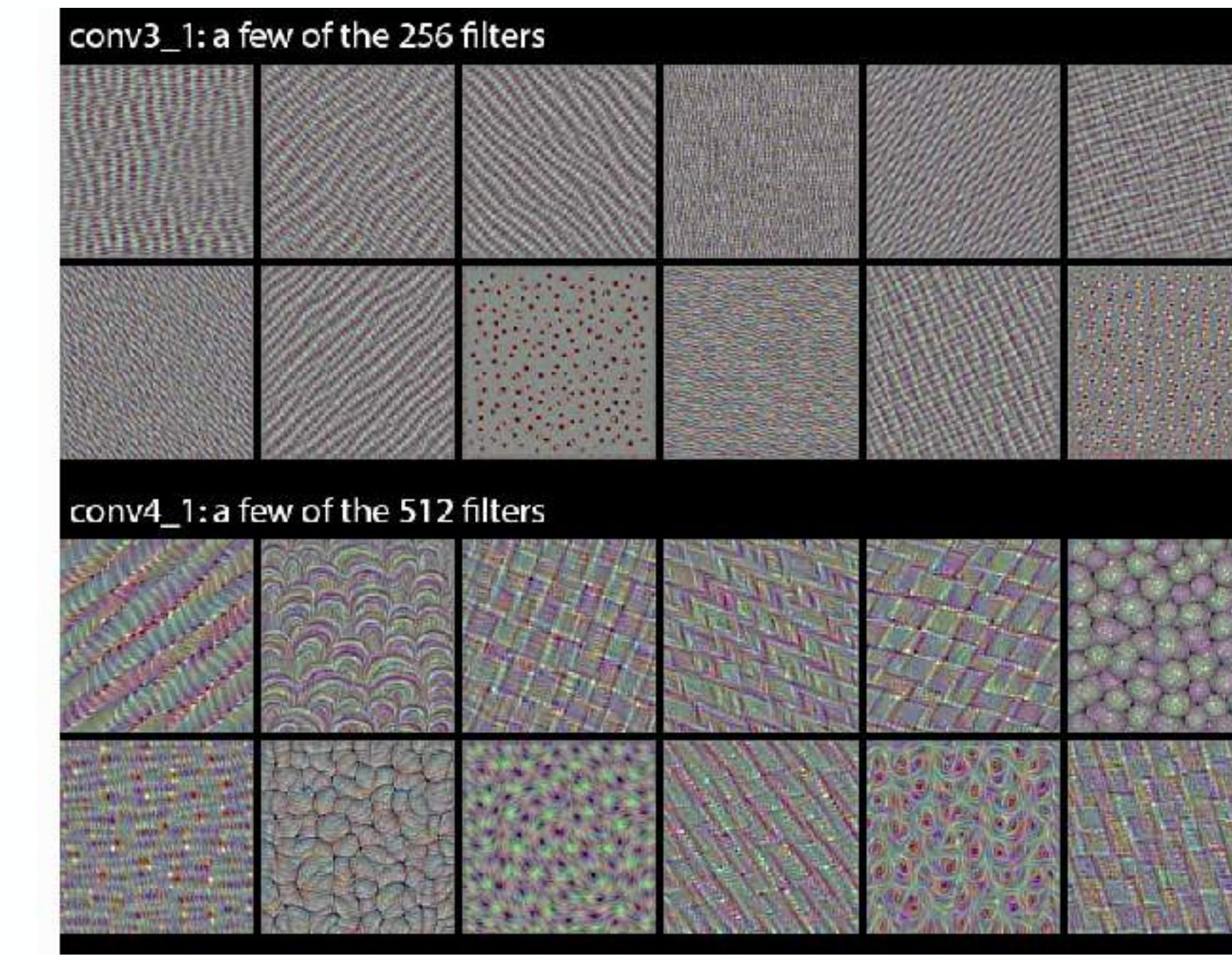
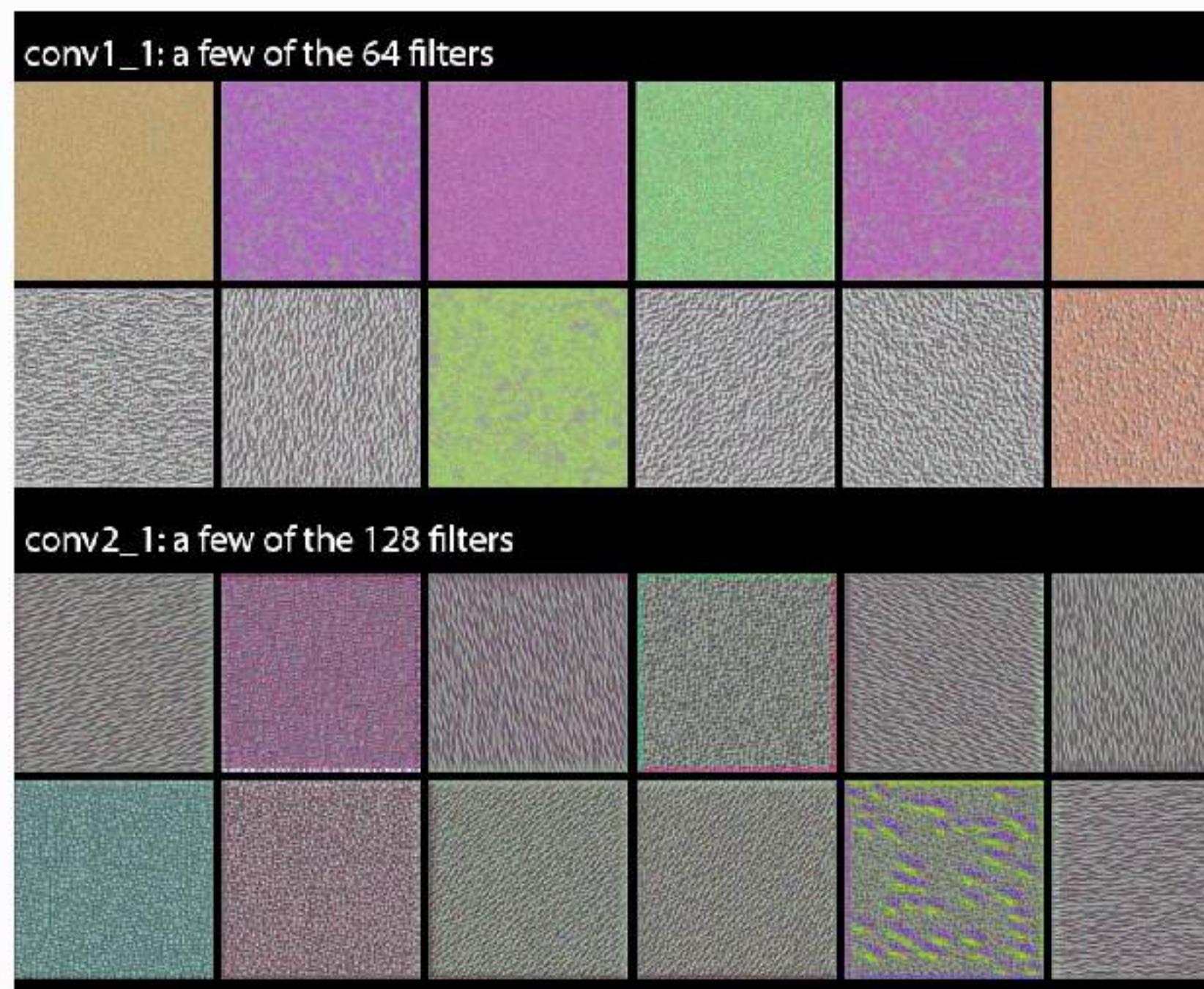
This could be our input image to maximise our filter



In reality it looks more like this when we maximise our filters

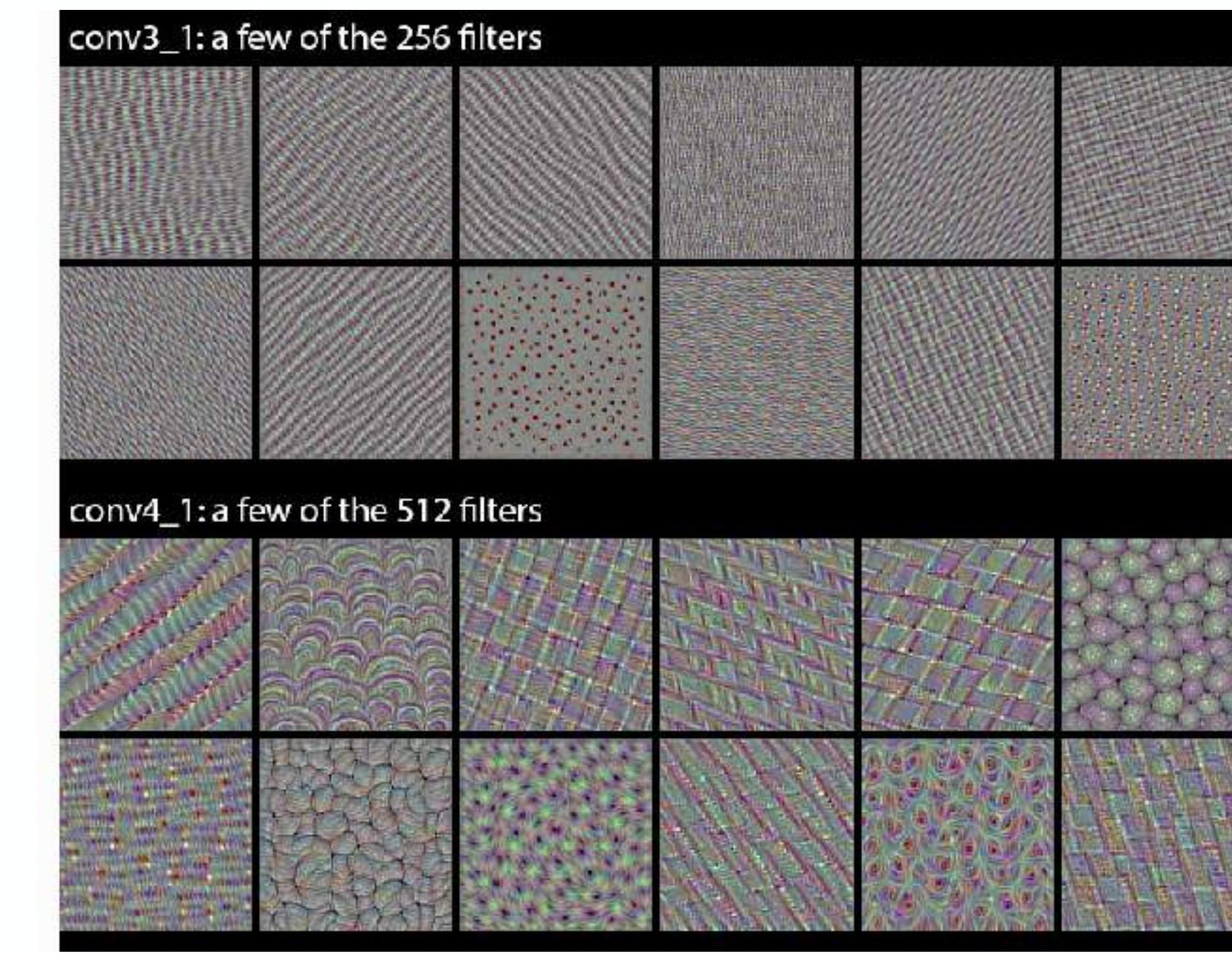
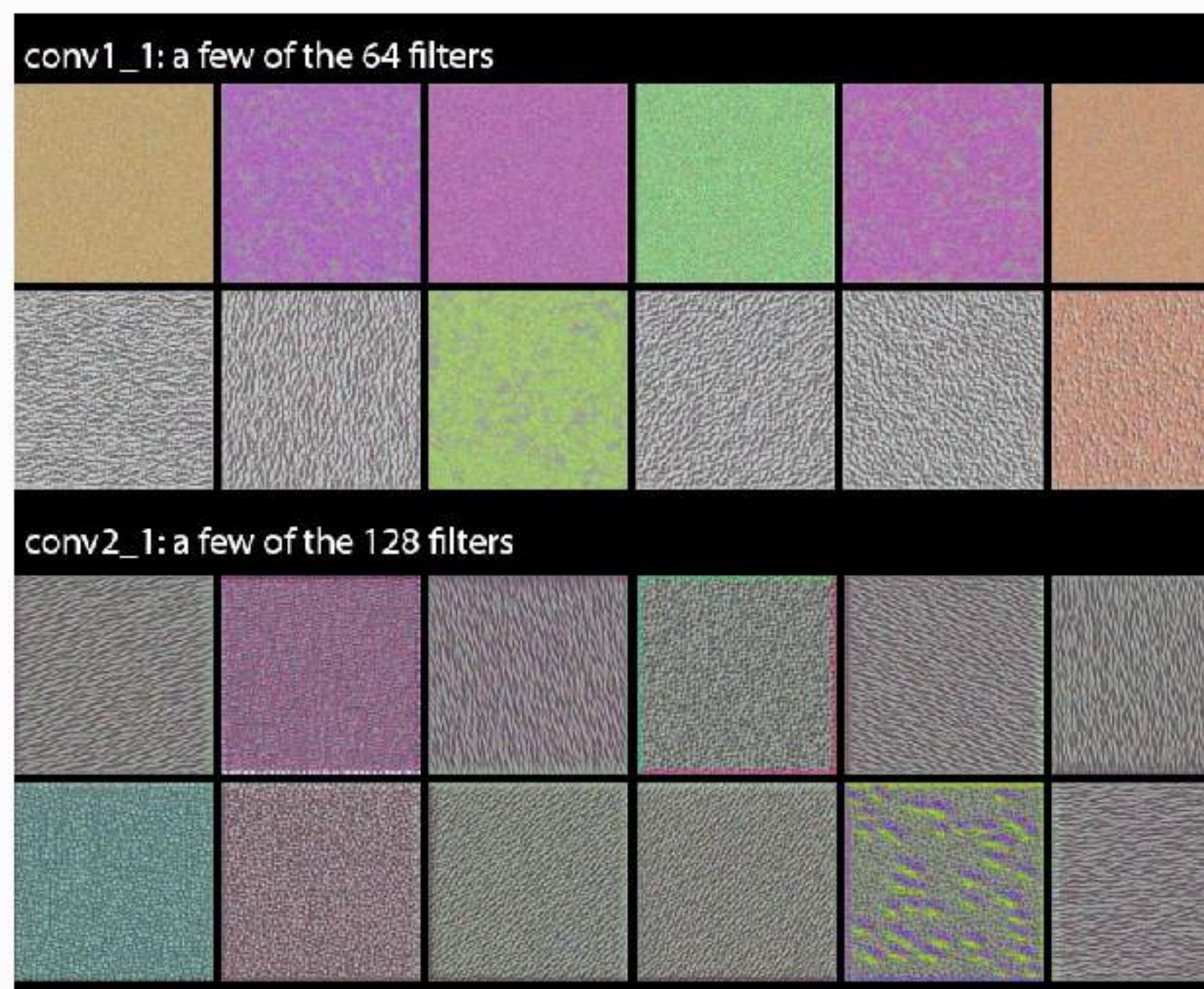
# Understanding Maximising Filters

- Viewing the input that maximises each filter gives us a nice visualisation of the CNN's modular-hierarchical decomposition of it's visual space.
- The first layers basically just encode direction and colour. These direction and colour filters then get combined into basic grid and spot textures. These textures gradually get combined into increasingly complex patterns.



# Understanding Maximising Filters

- Note a lot of these filters are identical but rotated (typically 90 degrees). This provides evidence of the rotation invariance provided by CNNs.



# Implementation

- Load a Trained model
- Define a loss function that seeks to maximise the activation of a specific filter (filter\_index) in a specific layer (layer\_name)
- Small trick involved here - We normalise the gradient of the pixels of the input image, which avoids very small and very large gradients and ensures a smooth gradient ascent process

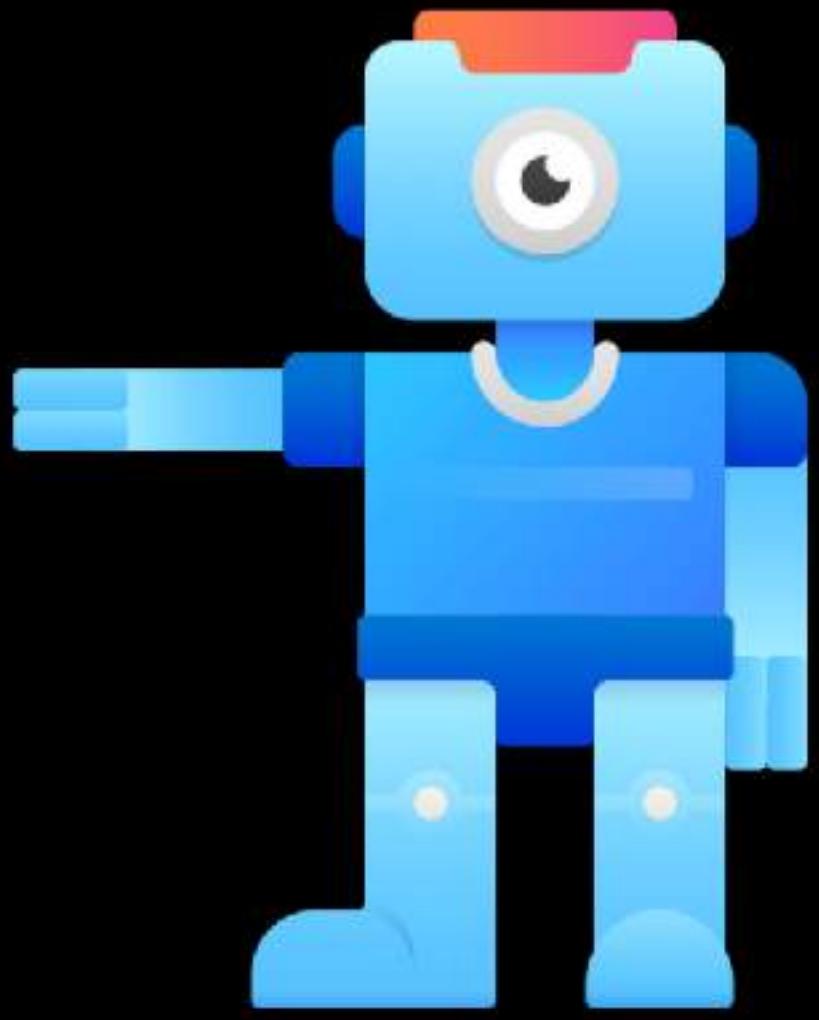


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

## Maximising Class Activations



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Maximising Class Activations

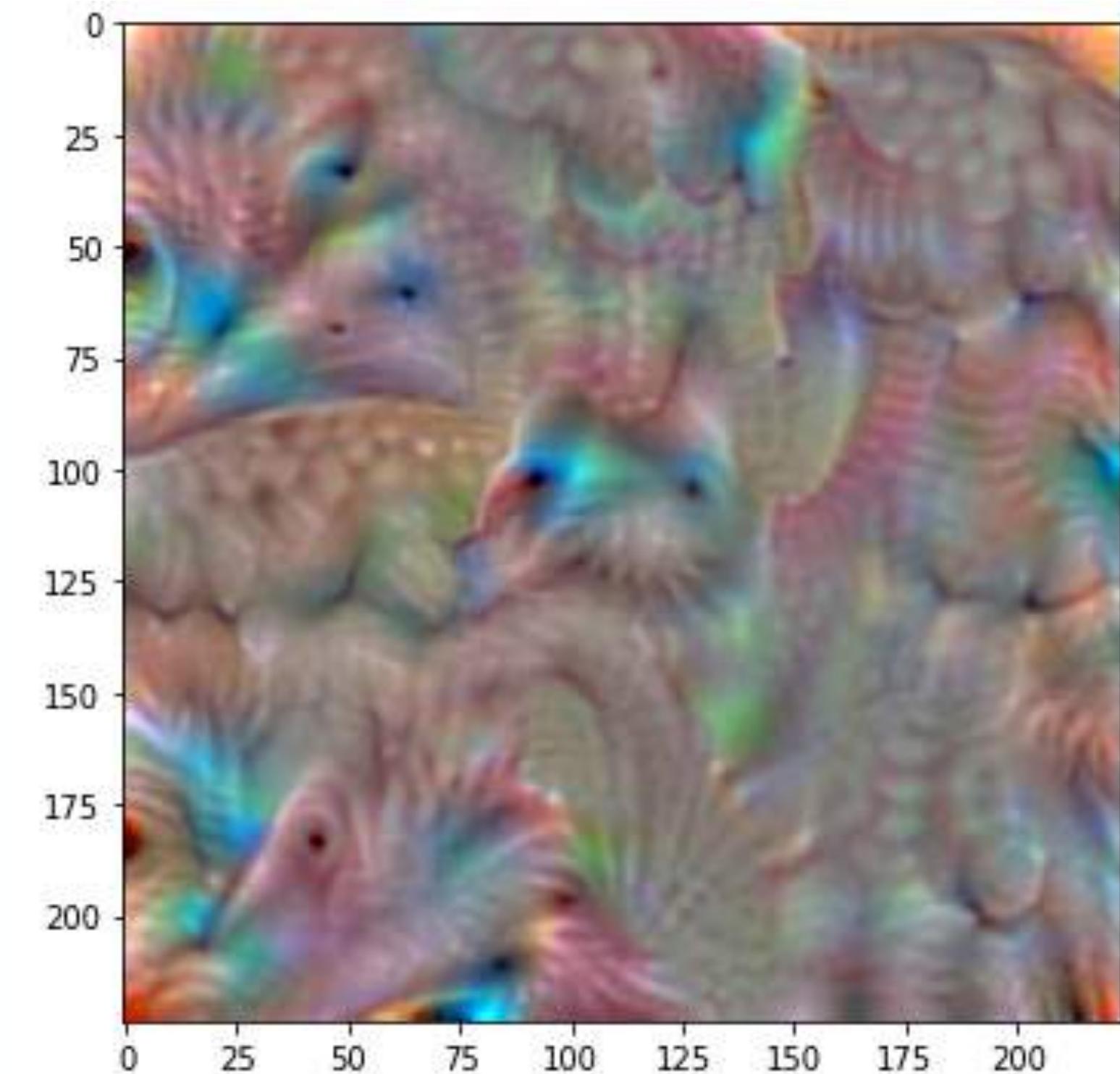
What Input images maximise their class representations?

# Class Maximisation

- So let's say we've trained a CNN to classify cats vs dogs
- But does our CNN know what cat actually looks like?
- What input makes our CNN output at 100% certainty, that it's seeing a cat?

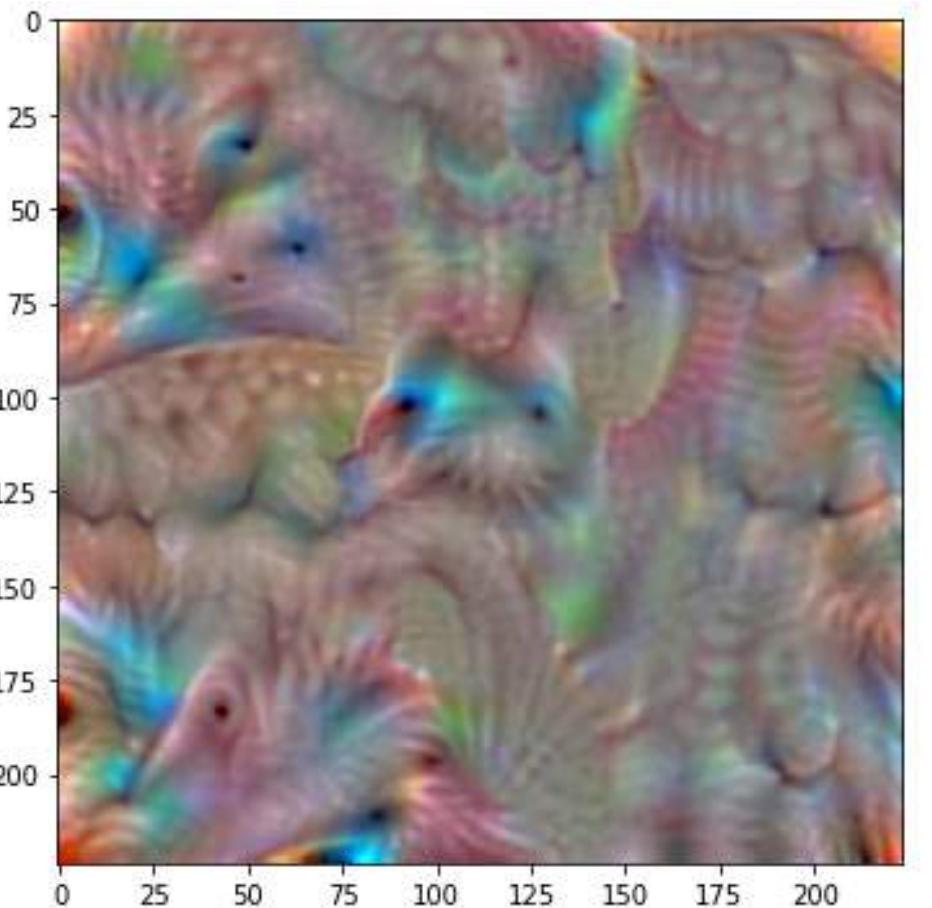
# Class Maximisation

- Apparently this is the image that maximises the class (Bald Eagle) on a CNN (VGG) trained on ImageNet dataset.



# Class Maximisation Takeaways

- CNN's internalise local features (Feathers, beaks, eyes) that bear resemblance to the class it's trained to recognise.
- This shows that CNNs don't learn like we do, technically, they understand a decomposition of the visual input space as a hierarchical-modular network of convolution filters.
- Their internal network is then a probabilistic mapping between combinations of these filters and their class labels.



# Fooling CNNs

- Sometimes small changes in an image can result in widely different classifications

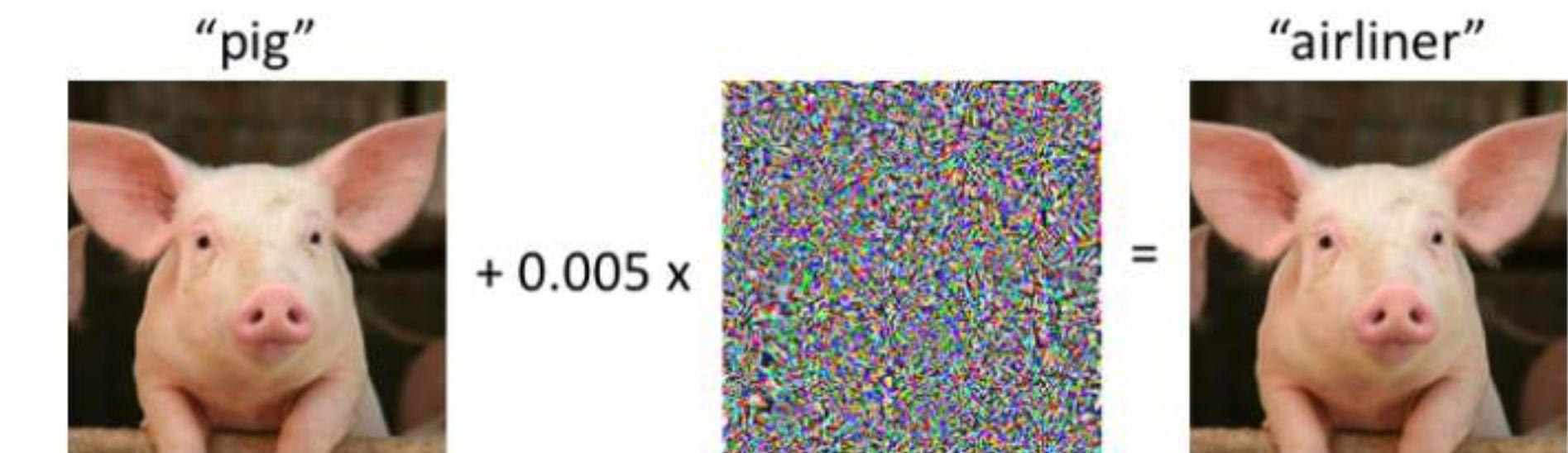
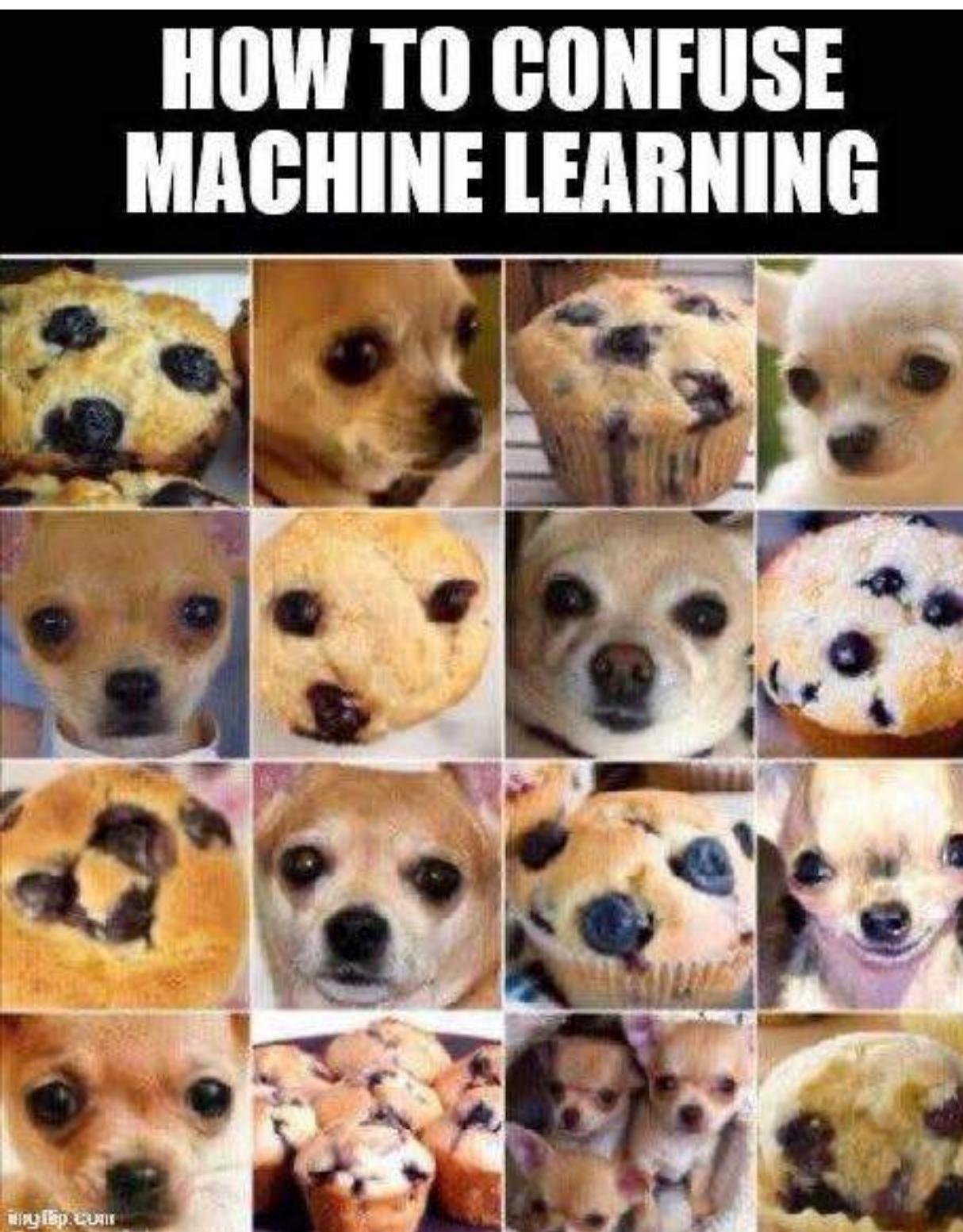


Figure 1: Adversarial example fooling a CNN into detecting a pig as airliner

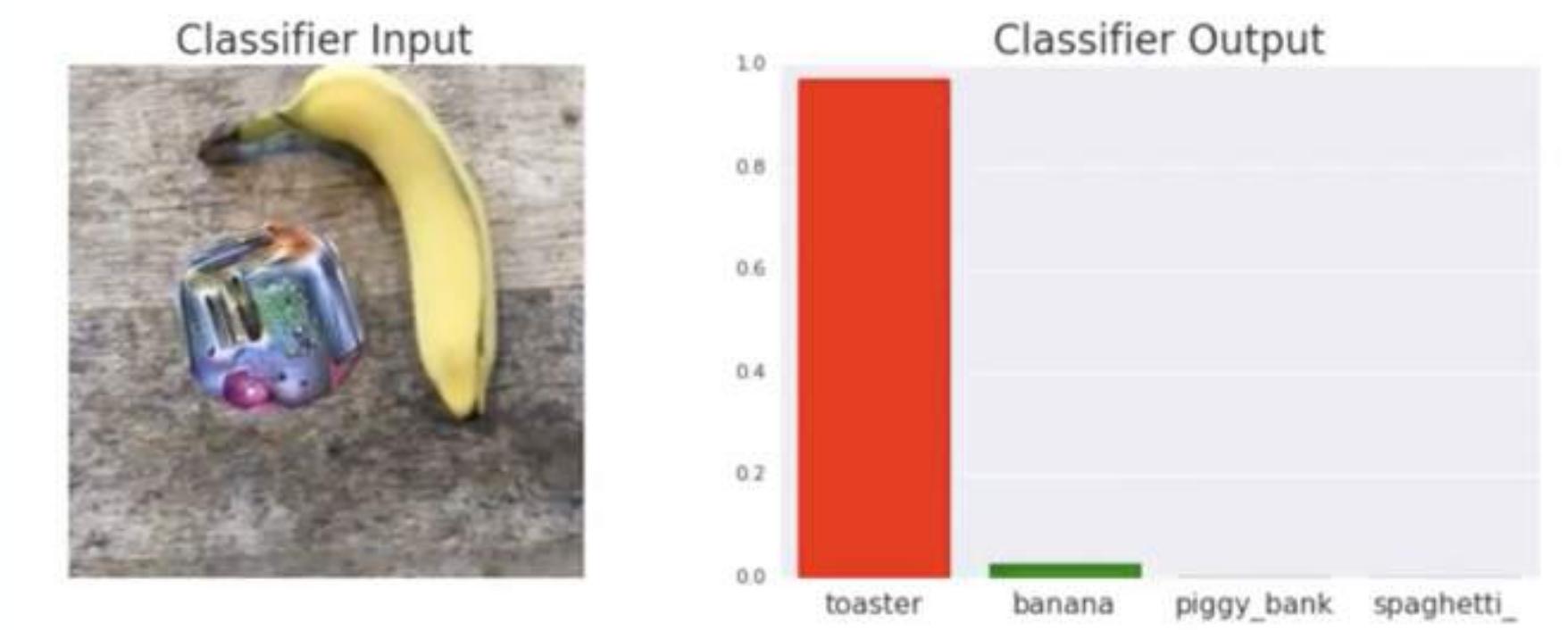
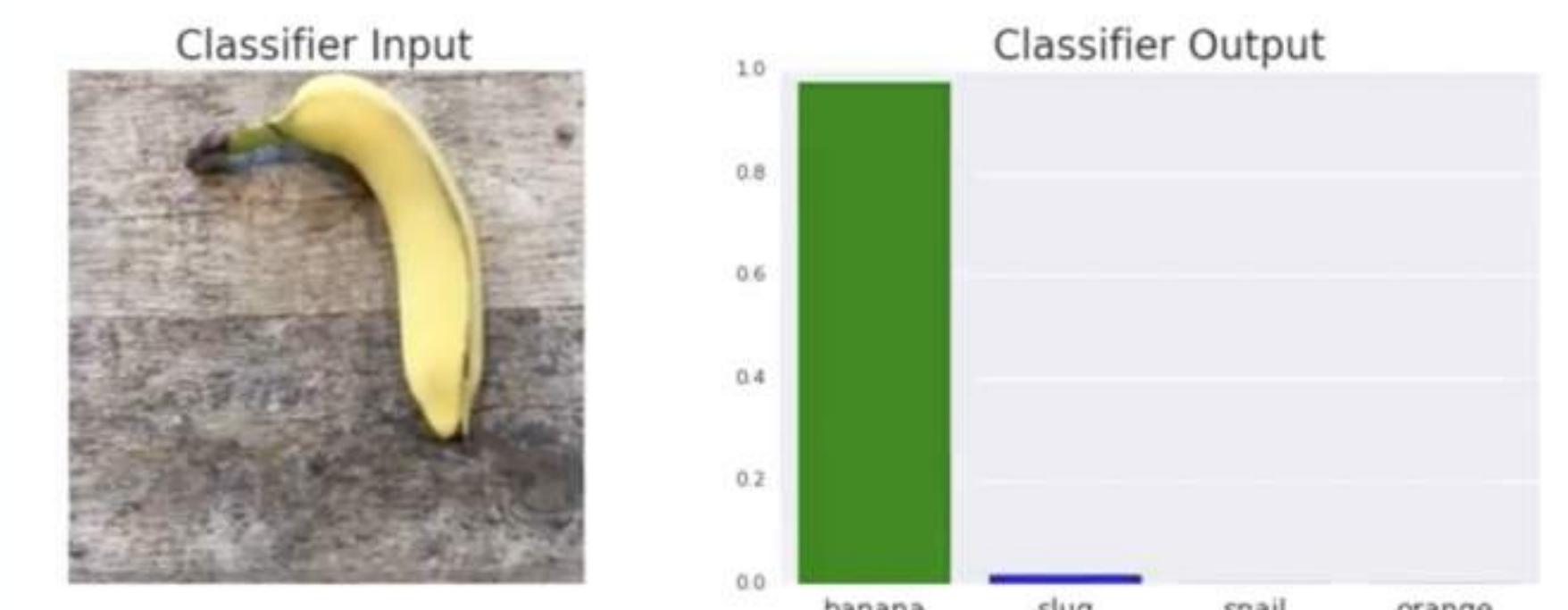


Figure 2 : Adversarial patches making the CNN ignore all other features

# Fooling CNNs

- One Pixel Attacks

AllConv



SHIP  
CAR(99.7%)

NiN



HORSE  
FROG(99.9%)

VGG



DEER  
AIRPLANE(85.3%)



HORSE  
DOG(70.7%)



DOG  
CAT(75.5%)



BIRD  
FROG(86.5%)



CAR  
AIRPLANE(82.4%)



DEER  
DOG(86.4%)



CAT  
BIRD(66.2%)



DEER  
AIRPLANE(49.8%)



BIRD  
FROG(88.8%)



SHIP  
AIRPLANE(88.2%)



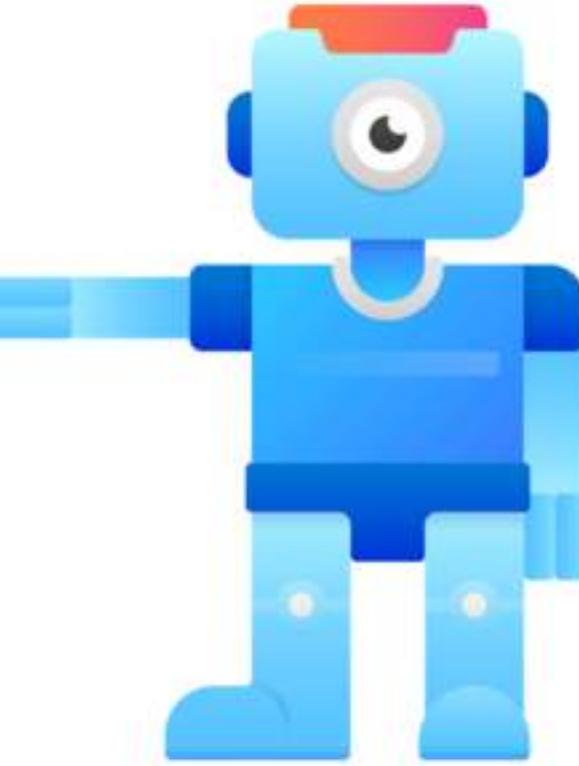
HORSE  
DOG(88.0%)



SHIP  
AIRPLANE(62.7%)



CAT  
DOG(78.2%)

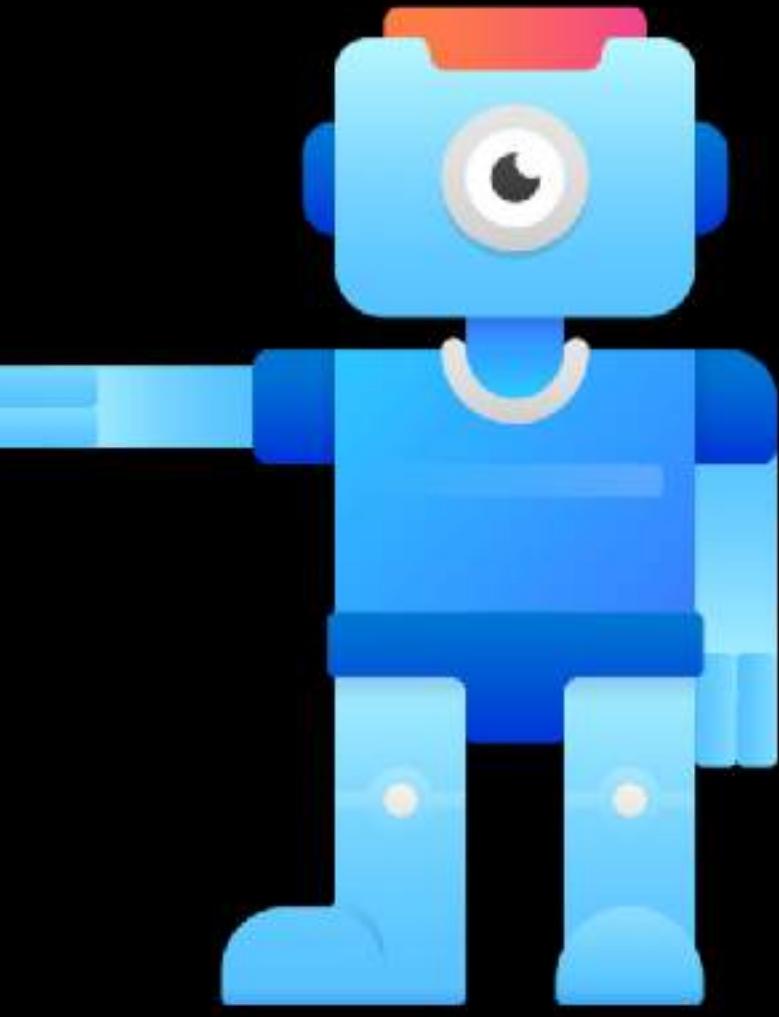


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Grad-CAM: Visual Explanations from Deep Networks  
via Gradient-based Localisation**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localisation

Grad-CAM allows us to see what areas of an image the CNN used to make it's decision

# Grad-CAM

- Grad-CAM is very useful CNN visualisation technique that shows us where the model is looking.



# Grad-CAM - Usefulness

- Grad-CAM is useful as it provides better understanding of our model, showing us what our model used to make its decision.

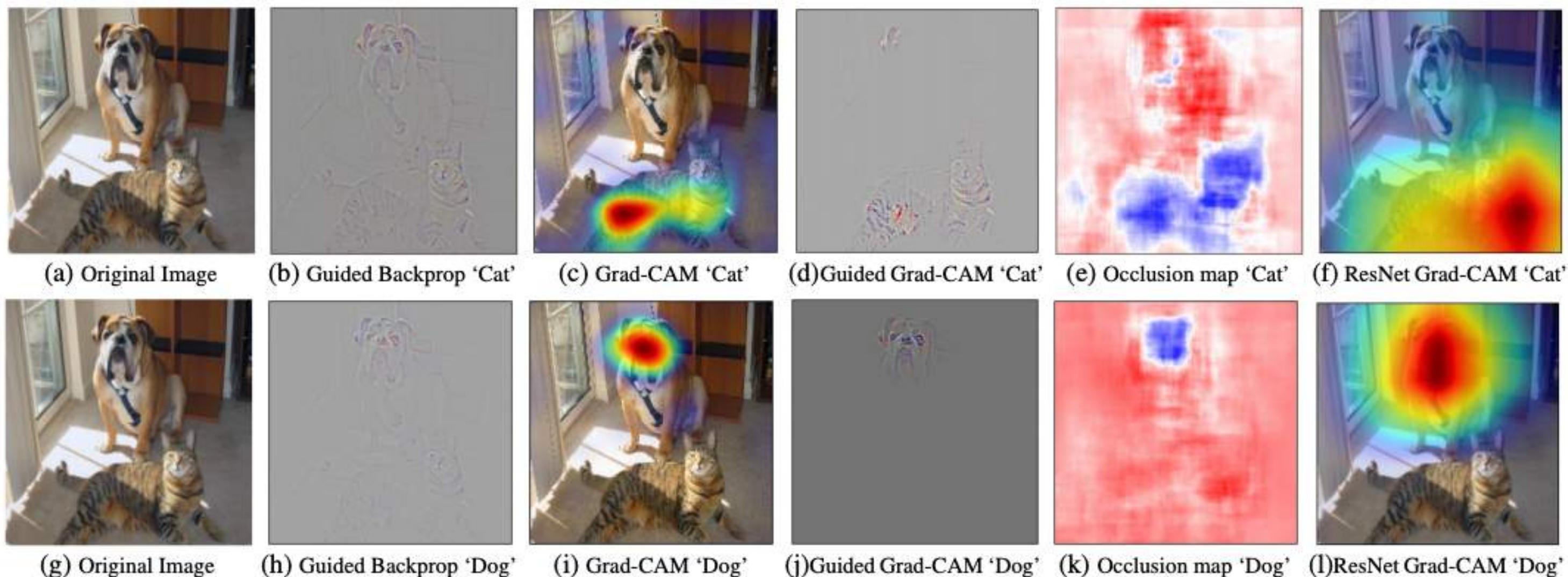
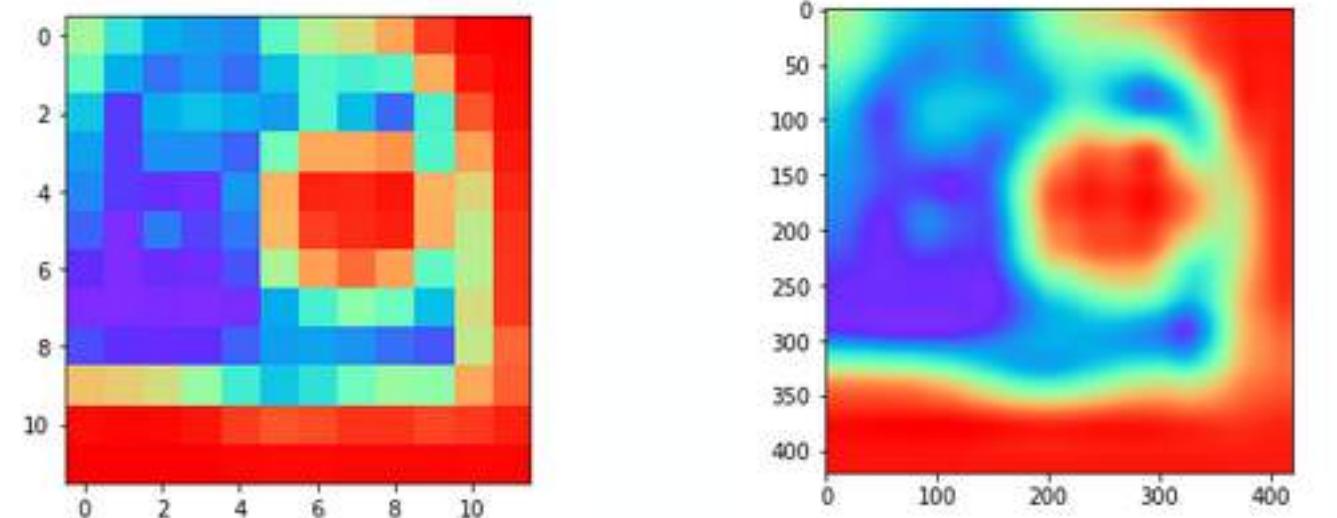
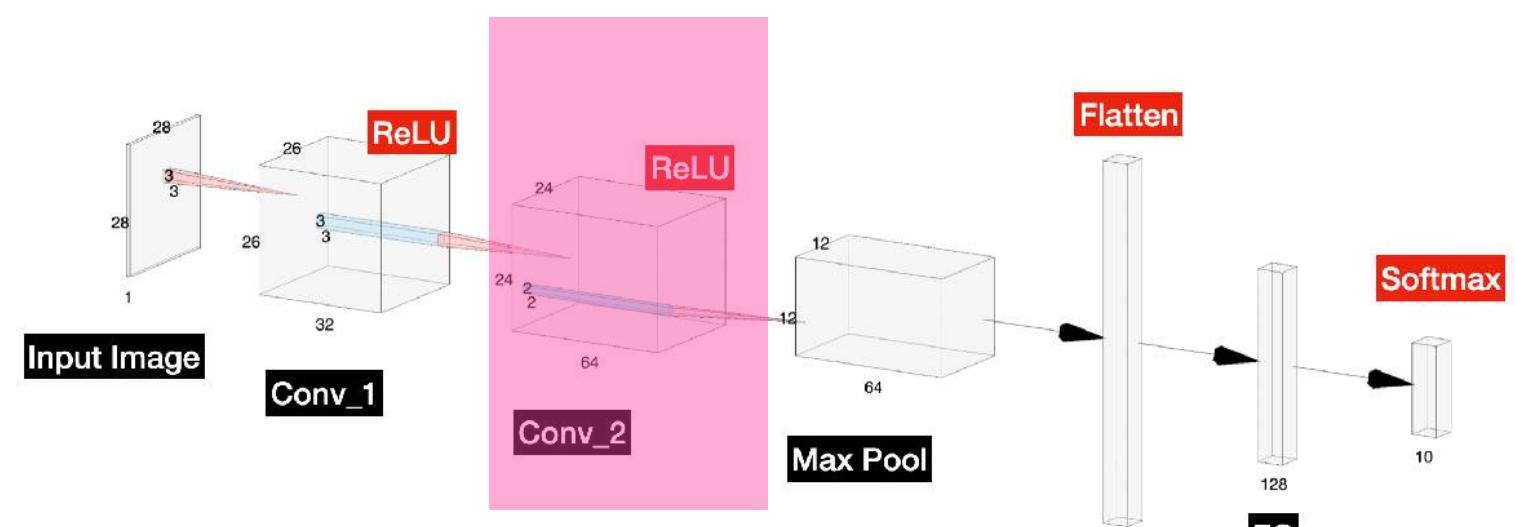


Fig. 1: (a) Original image with a cat and a dog. (b-f) Support for the cat category according to various visualizations for VGG-16 and ResNet. (b) Guided Backpropagation [53]: highlights all contributing features. (c, f) Grad-CAM (Ours): localizes class-discriminative regions, (d) Combining (b) and (c) gives Guided Grad-CAM, which gives high-resolution class-discriminative visualizations. Interestingly, the localizations achieved by our Grad-CAM technique, (c) are very similar to results from occlusion sensitivity (e), while being orders of magnitude cheaper to compute. (f, l) are Grad-CAM visualizations for ResNet-18 layer. Note that in (c, f, i, l), red regions corresponds to high score for class, while in (e, k), blue corresponds to evidence for the class. Figure best viewed in color.

# Grad-CAM - How does it Work?

- Grad-CAM exploits the spatial information that is preserved in Conv Layers.
- It uses the feature maps produced by the last CNN layer
- At this point we can insert some differentiable (so that we can get the gradients) layers after the last Conv filter outputs
- In Grad-CAM, we weight the feature maps using “alpha values” that are calculated based on gradients.
- This is used to make a heat-map, we can create one for each output class
- The heatmap is then upsampled to be overlaid on the image



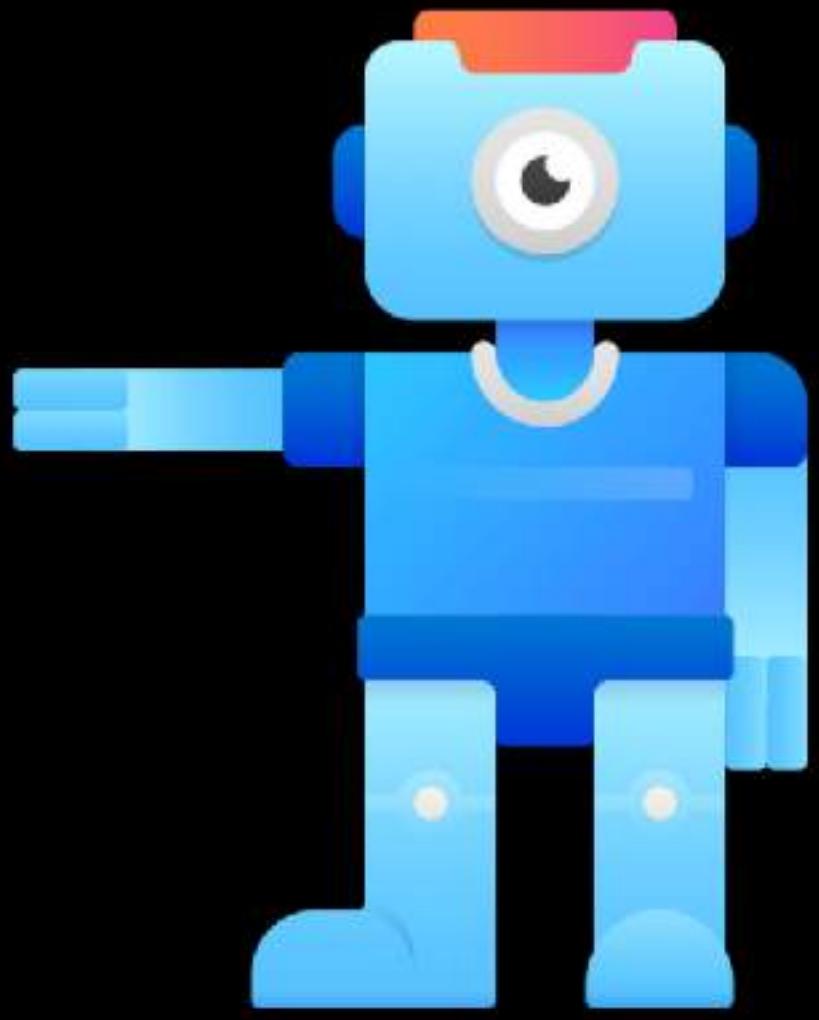


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Basic CNN Design Principles**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Basic CNN Design Principles

Rule of thumb guide to designing basic CNN Architectures

# When Designing a CNN You're Faced with Many Questions

- How many Layers (Conv Filters)?
- How many Filters per layer?
- What size Kernel? Stride?
- Maxpool? Activation Functions?
- How long to train?

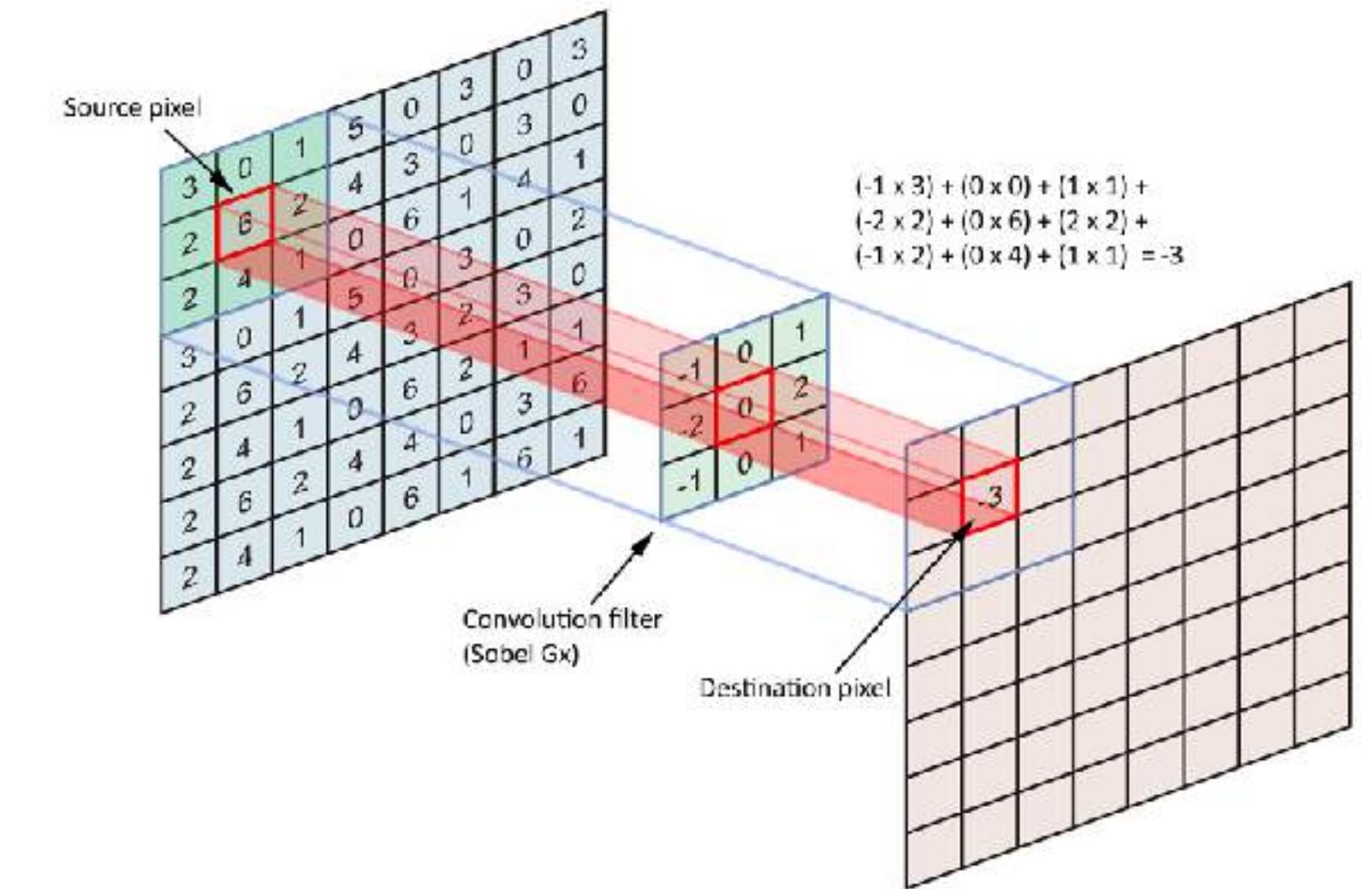
# A Basic Guide to Designing Effective CNNs

## Kernel or Filter Size

- Each filter is a matrix of weights that is convolved with the input image
- Each filter acts like a detector for some feature (edge, blob etc) in the image
- A smaller filters ( $3 \times 3$ ) are better at detecting local features
- Larger filters ( $7 \times 7$  or  $9 \times 9$ ) are better at detecting larger features

### How to decide?

- If you think what distinguishes your images are small localised features then smaller filters are better, if it's larger more global or high-level patterns then larger filters are better.
- A good practice is using smaller filters in earlier layers and gradually increase filter size in deeper layers
- Note, we use odd number sized filters so that we have source pixel (centre).
- <https://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-size>



# A Basic Guide to Designing Effective CNNs

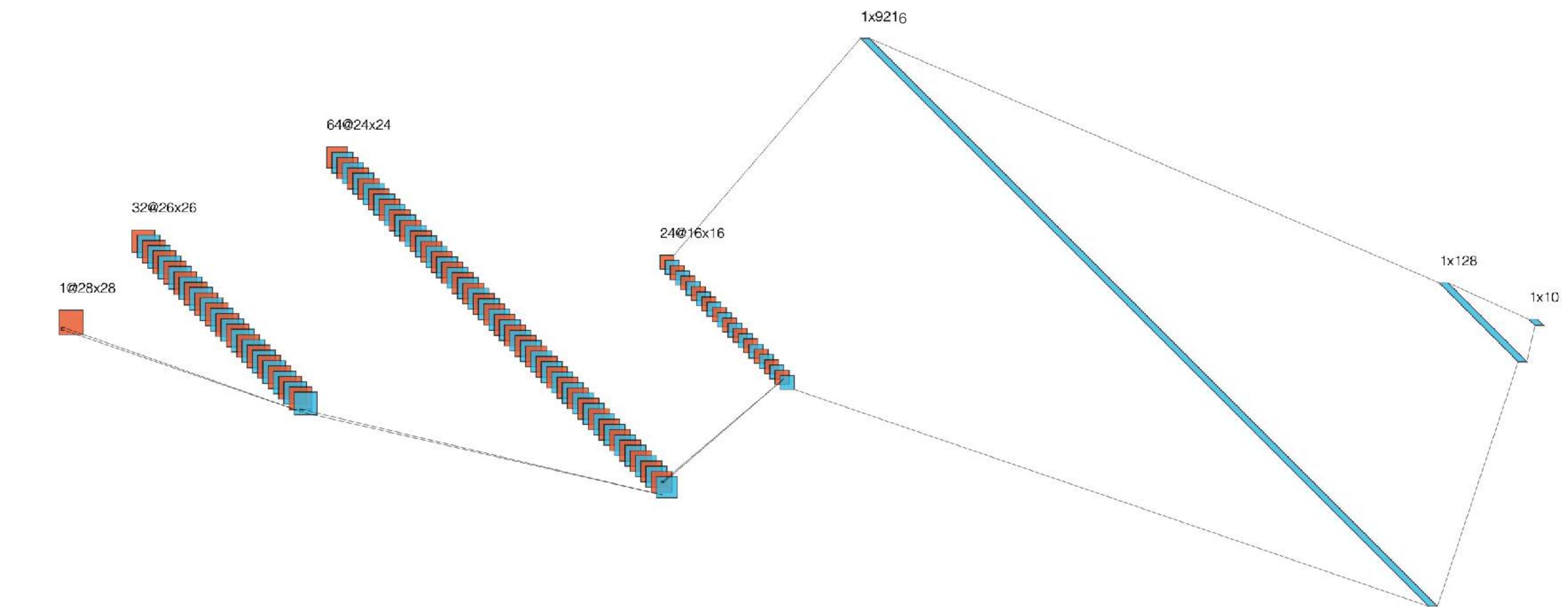
## Number of Kernel/Filters and Number of Conv Layers

- Generally if your task is not too complex 2 or 3 layers would be a good starting point.
  - Complexity is determined by:
    - Number of classes
    - Similarity of classes
    - Variation and deformation in the dataset
- By convention the number of channels grow as we progress through the network (e.g. 32 -> 64 -> 128)
- Note you may see this reversed in some types of networks (like U-Net or some Siamese Network architectures).

# A Basic Guide to Designing Effective CNNs

## Number of Nodes in our Dense Fully Connected Layers

- The number of hidden neurons should be **between the size of the input layer and the size of the output layer.**
- The number of hidden neurons should be **2/3 the size of the input layer, plus the size of the output layer.**
- The number of hidden neurons should be **less than twice the size of the input layer.**



- According to Steffen B Petersen · Aalborg University
- In order to secure the ability of the network to generalise the number of nodes has to be kept as low as possible. If you have a large excess of nodes, your network becomes a memory bank that can recall the training set to perfection, but does not perform well on samples that were not part of the training set.

# A Basic Guide to Designing Effective CNNs

## Padding

- It's generally good practice to use padding (zero padding) so that we retain information at the borders and don't downsample our image too much as we propagate through the network

# A Basic Guide to Designing Effective CNNs

## Stride

- Normally we generally use Stride of 1, any greater would reduce the input size too much so generally we stick to 1

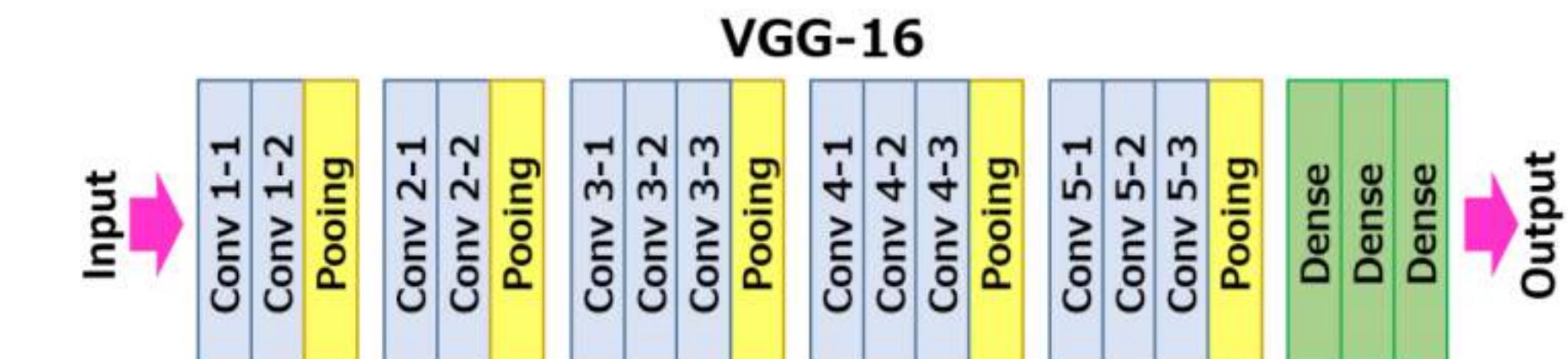
# A Basic Guide to Designing Effective CNNs

## Pooling and Number of Channels

- Normally we use Max Pooling, other pooling methods aren't as well suited to most CNNs
- Number of Channels choice is simple in theory, 3 for colour (RGB) or 1 for grayscale.
- However, the choice of whether to change your input image to grayscale depends on whether you think colour adds key information in distinguishing your class

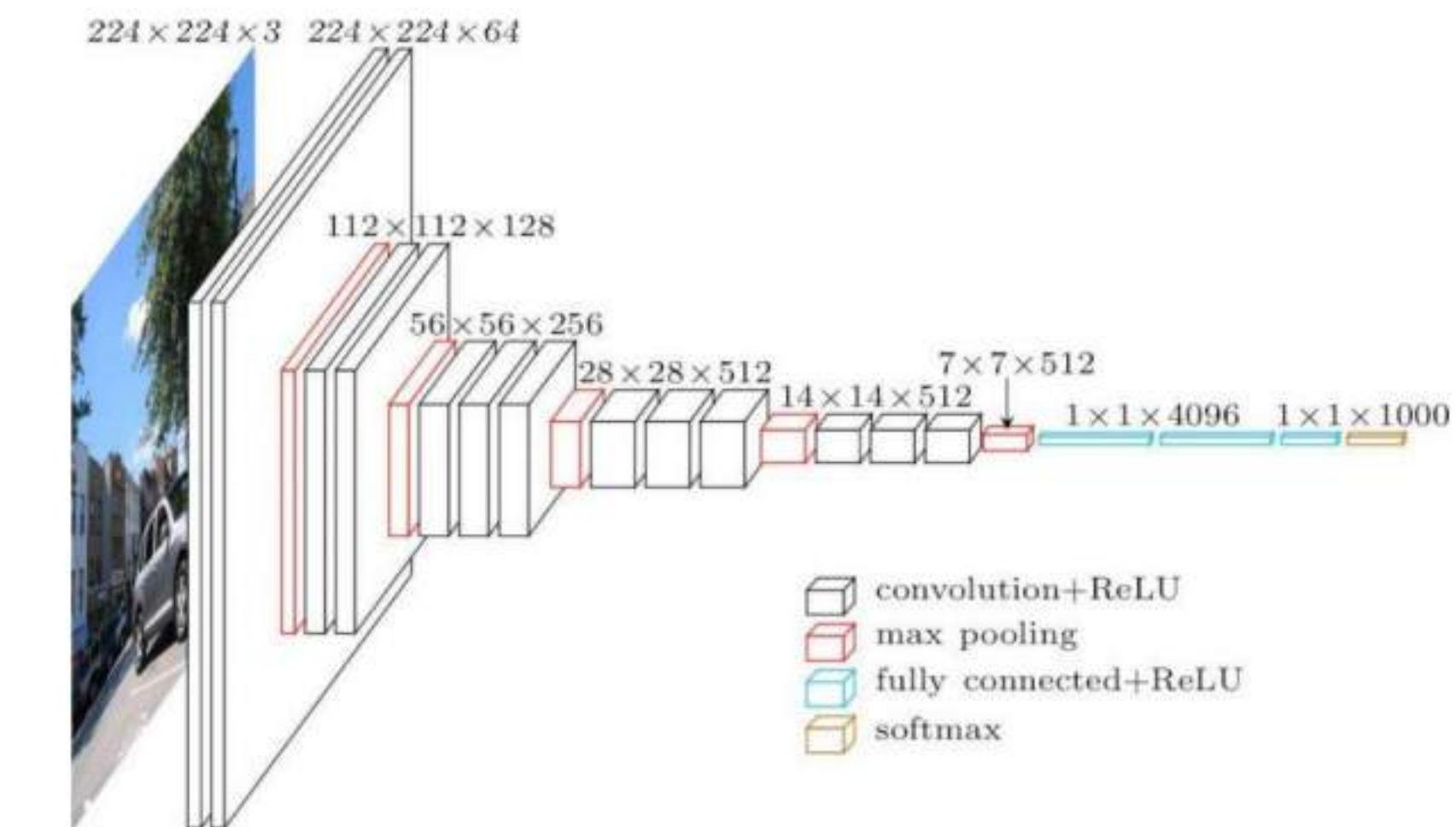
# Classic CNN Architectures

- LeNET
- AlexNet
- VGG16



**The Architecture**

The architecture depicted below is VGG16.



# Modern CNN Architectures

- ResNet
- InceptionV3
- Xception
- MobileNet
- SqueezeNet
- DenseNet
- EfficientNet
- GoogleNet

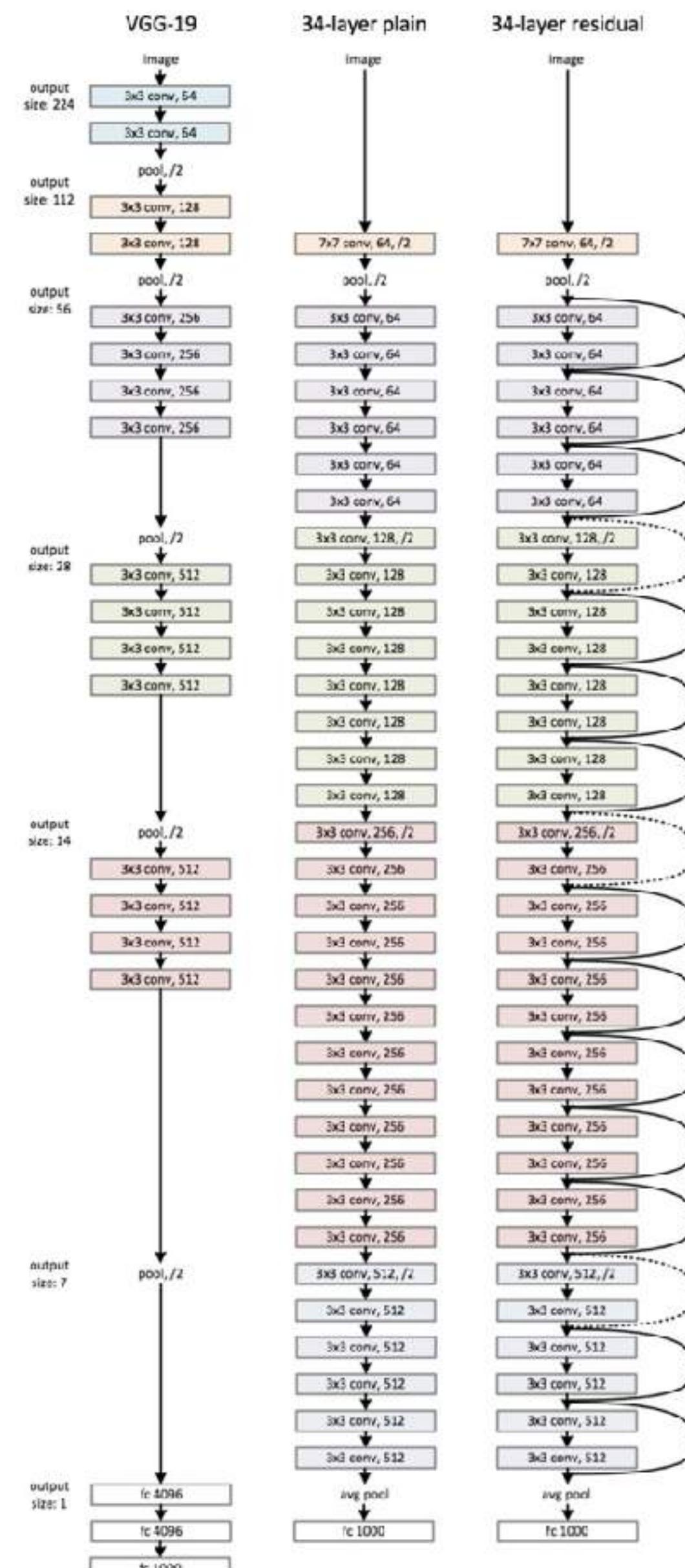


Figure 3. Example network architectures for ImageNet. **Left:** the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle:** a plain network with 34 parameter layers (3.6 billion FLOPs). **Right:** a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. Table 1 shows more details and other variants.



# MODERN COMPUTER VISION

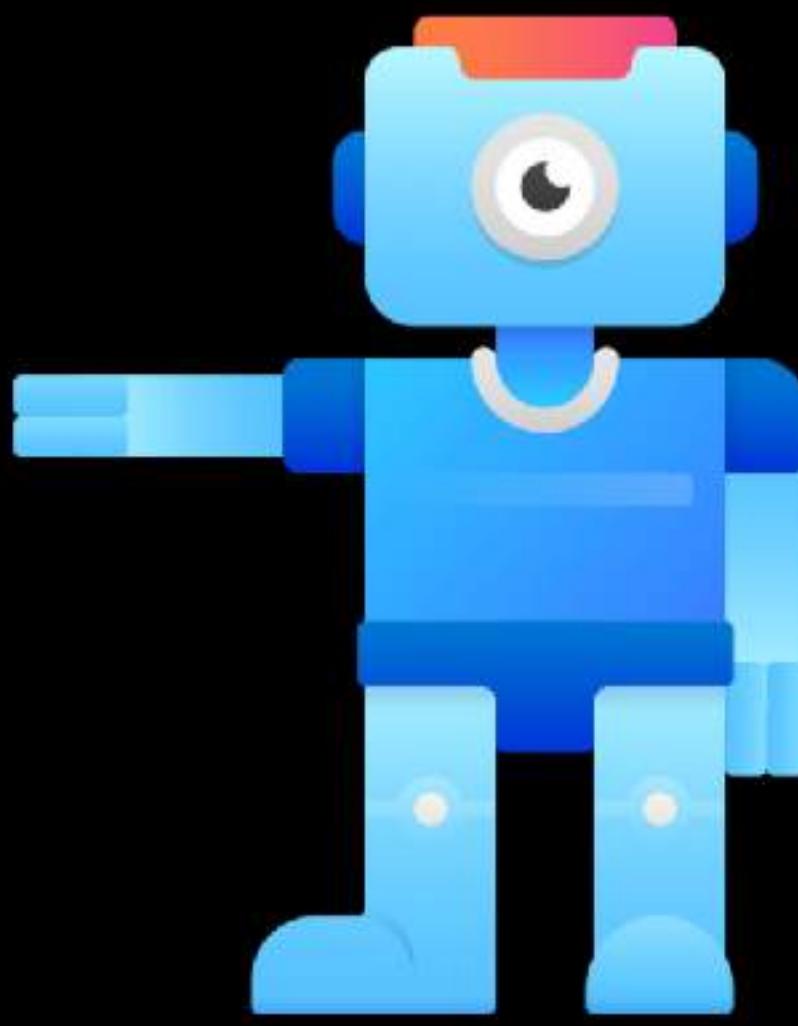
BY RAJEEV RATAN

# Next...

A look at the Evolution of CNNs

# CNN History

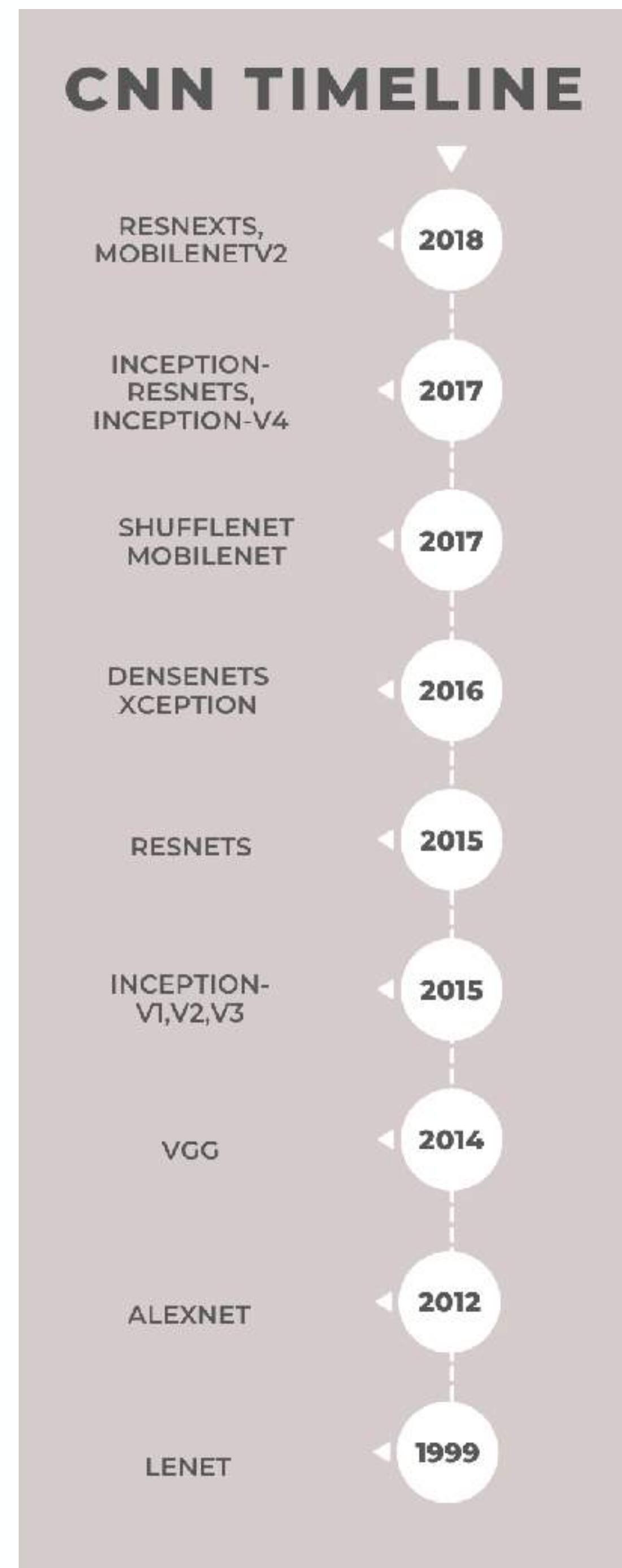
A brief outline of the evolution of CNN Research



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# CNN History





# MODERN COMPUTER VISION

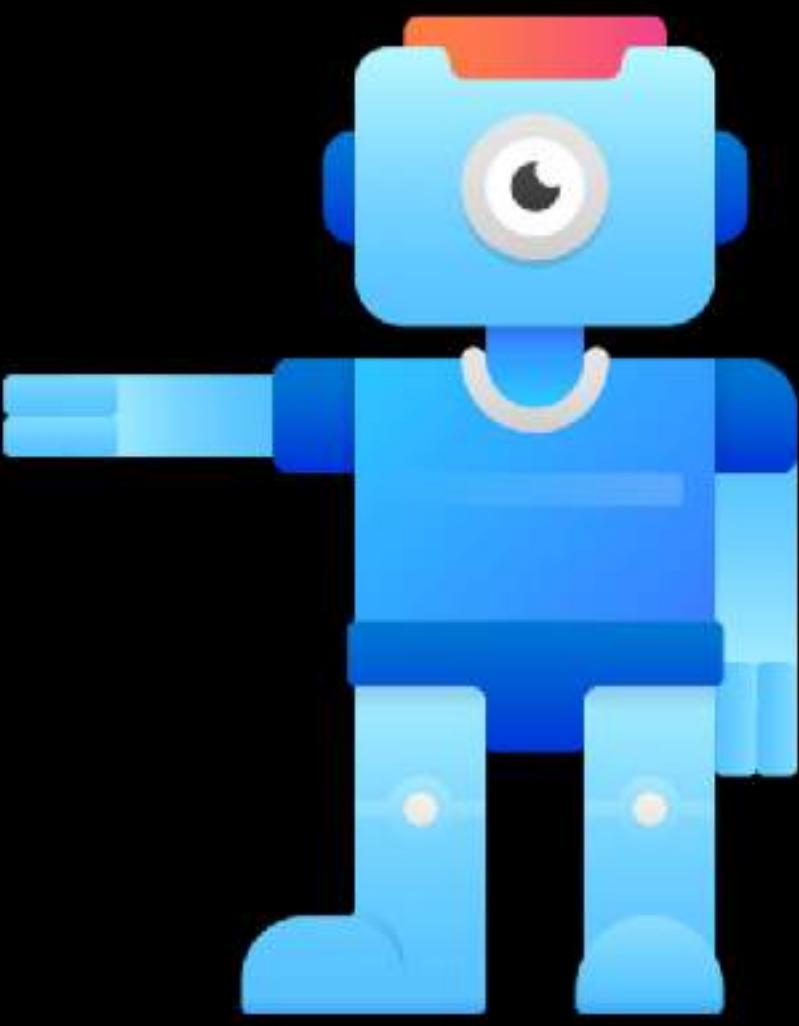
BY RAJEEV RATAN

# Next...

A look at LeNet

# LeNet

## An overview of the LeNet CNN



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# LeNet

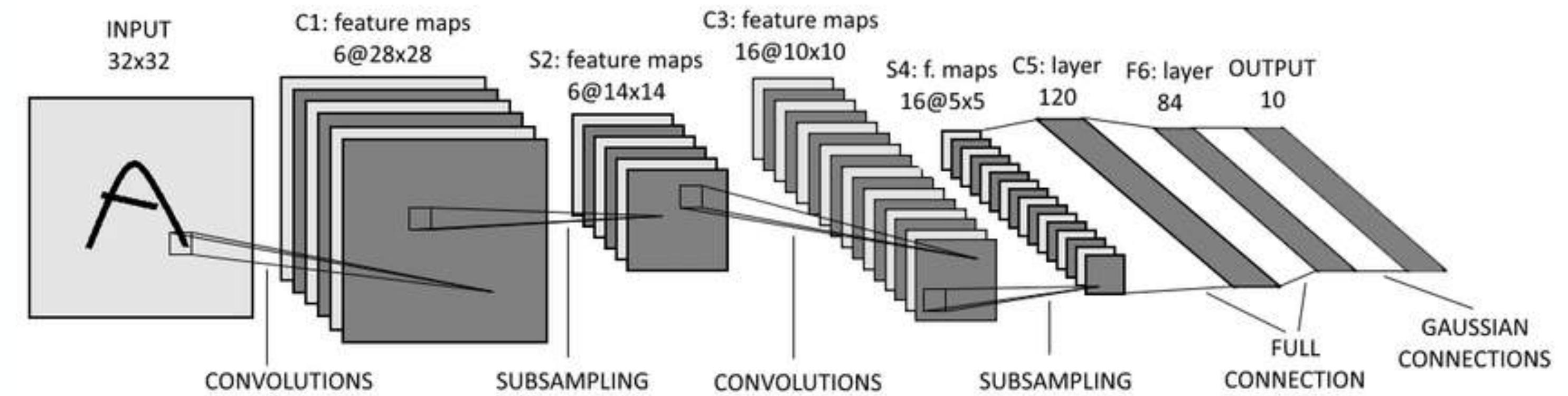
- LeNet was introduced in 1995 by LeCun
- It was developed for handwritten digit recognition for US zip codes (MNIST)

## Zipcode Example

65473      60198      68544  
70065    70117    19032    96720  
27260      61828      19559  
74136      19137      63101  
20878      60521      38002  
48640-2398    20907    14868

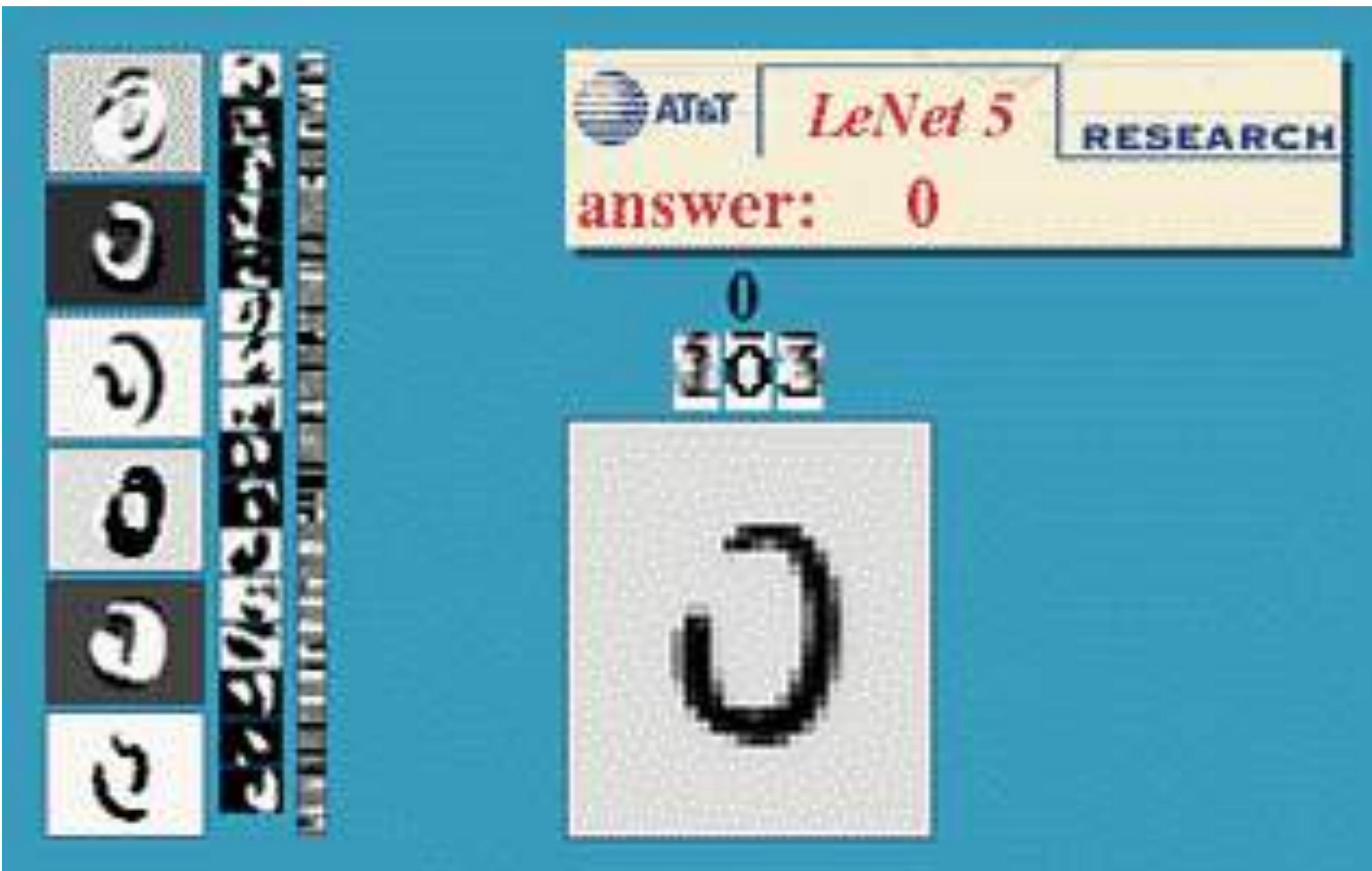
Examples of handwritten postal codes  
drawn from a database available from the US Postal service

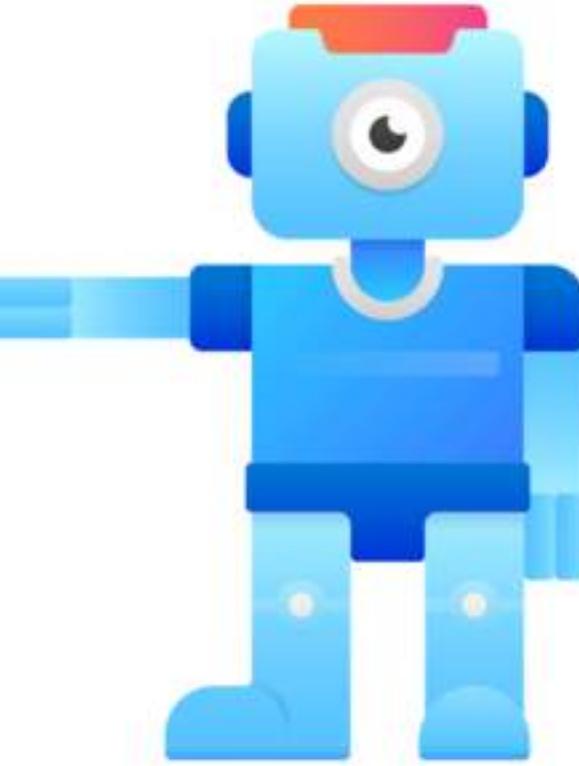
# LeNet Architecture



- Two convolution layers with Max Pool (subsampling of 2x2 kernel and stride 2)
- First Conv layer had 6 filters/kernels and size 5x5 (stride 1, padding 0)
- Second Conv layer had 16 filters/kernels of size 5x5 (stride 1, padding 0)
- After the 2nd Max Pool, we flatten the 5x5x16 layer into 400 neurons and connect it to the first FC layer of 120 neurons, then another FC layer of 84 neurons before reaching the final output layer (10 classes)
- LeNet can achieve 99.3% Accuracy on MNIST

# LeNet Animation





# MODERN COMPUTER VISION

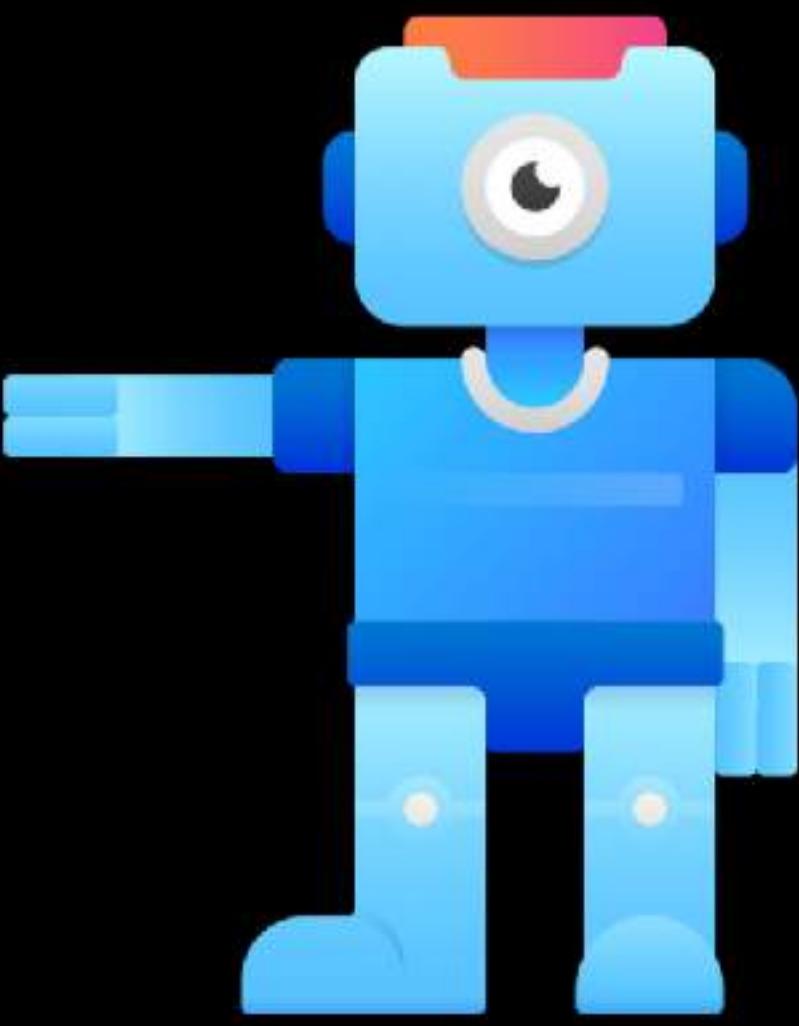
BY RAJEEV RATAN

# Next...

A look at AlexNet

# AlexNet

## An overview of the AlexNet CNN



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# AlexNet

- AlexNet was introduced in 2012 by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton from the University of Toronto.
- It was the ILSVRX winner in 2012.
- It contained 8 layers with the first five being Convolutional Layers and the last 3 being FC layers.
- It has over 60 Million parameters and was trained on two GPUs for over a week!

## IMAGENET Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)

*Held in conjunction with PASCAL Visual Object Classes Challenge 2012 (VOC2012)*

[Back to Main page](#)

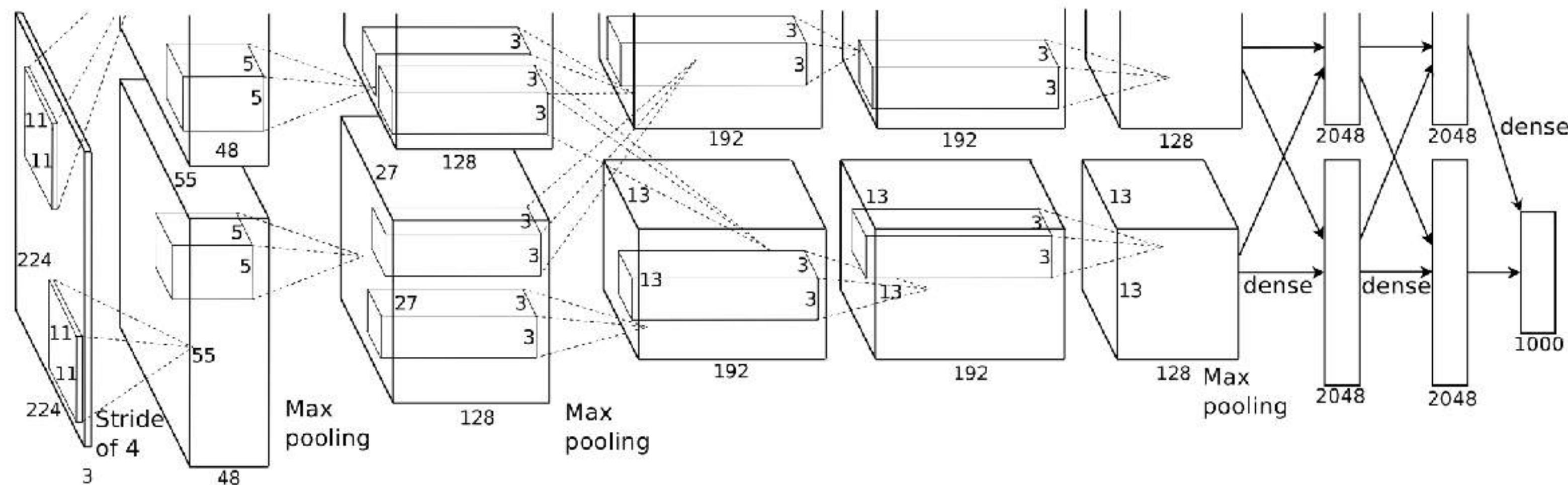
### All results

- [Task 1 \(classification\)](#)
- [Task 2 \(localization\)](#)
- [Task 3 \(fine-grained classification\)](#)
- [Team information and abstracts](#)

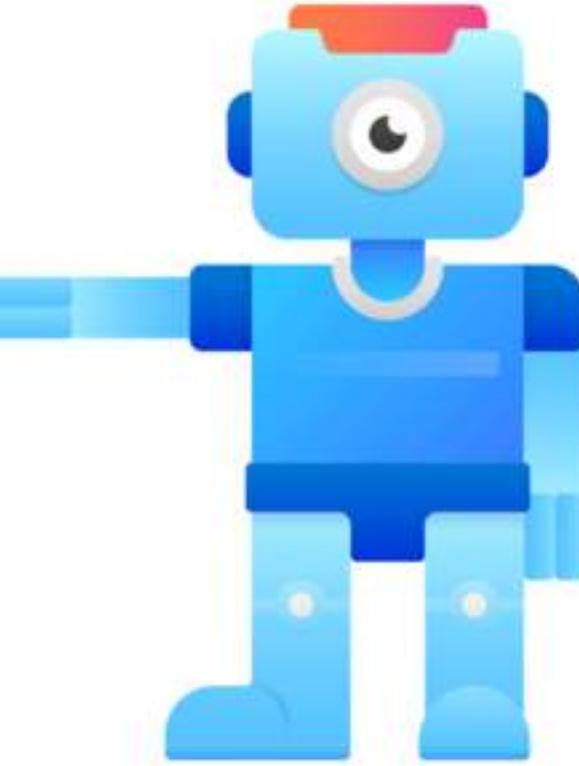
#### Task 1

Team name	Filename	Error (5 guesses)	Description
SuperVision	test-preds-141-146.2009-131-137-145-146.2011-145f.	0.15315	Using extra training data from ImageNet Fall 2011 release
SuperVision	test-preds-131-137-145-135-145f.txt	0.16422	Using only supplied training data
ISI	pred_FVs_wLACs_weighted.txt	0.26172	Weighted sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and CSIFT+FV, respectively.
ISI	pred_FVs_weighted.txt	0.26602	Weighted sum of scores from classifiers using each FV.
ISI	pred_FVs_summed.txt	0.26646	Naive sum of scores from classifiers using each FV.
ISI	pred_FVs_wLACs_summed.txt	0.26952	Naive sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and CSIFT+FV, respectively.

# AlexNet Architecture



**Figure 2:** An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

A look at VGGNet

# VGGNet

## An overview of the VGGNet CNN



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# VGGNet

- VGGNet was first introduced in 2014 by Oxford University Researchers Karen Simonyan and Andrew Zisserman.
- It achieved 92.7% top-5 Accuracy in ImageNet (1000 classes).
- VGG16 has 13 Conv Layers with 3 FC Layers
- VGG19 has 16 Conv Layers with 3 FC Layers

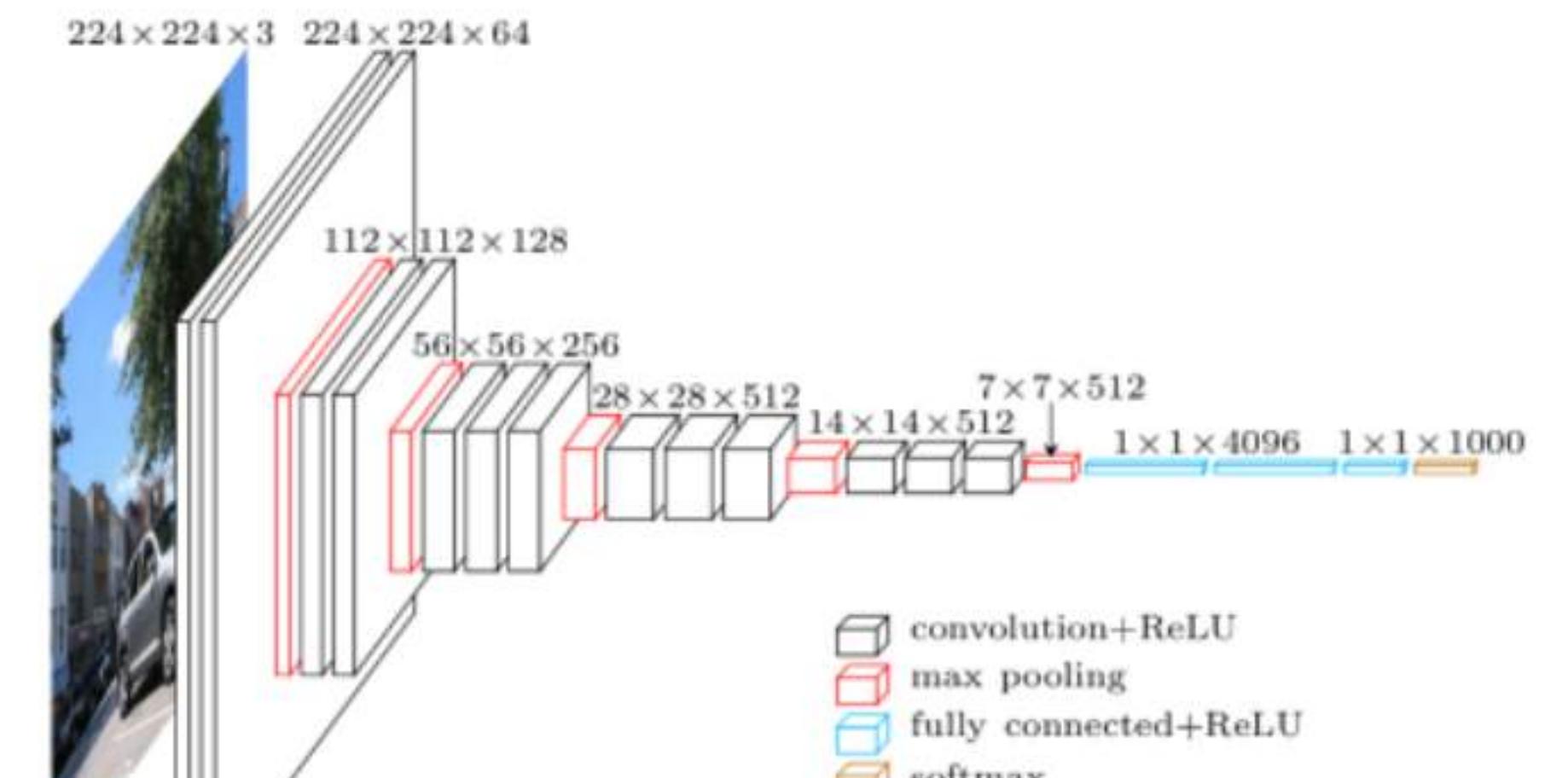
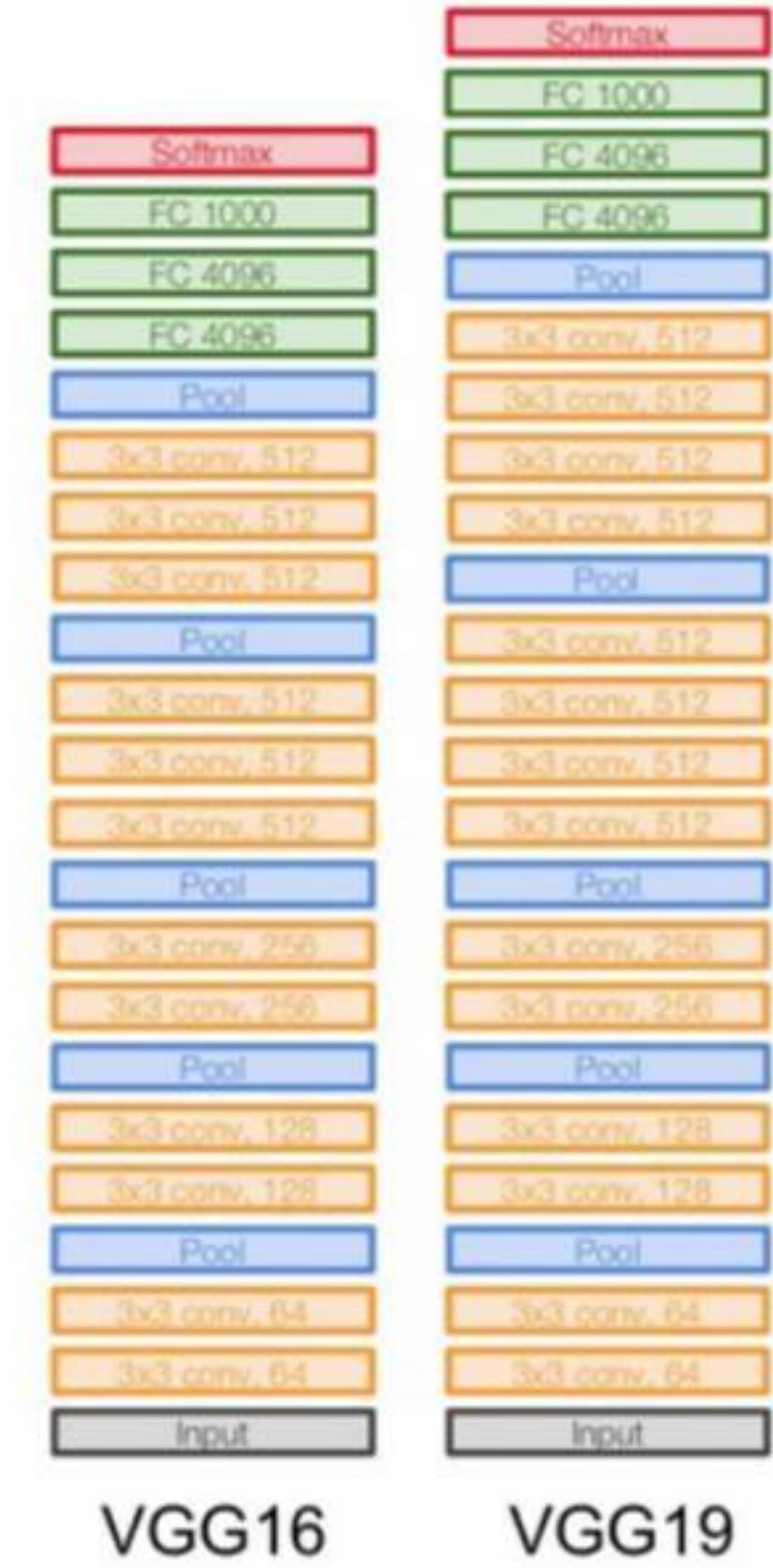


FIG. 2 - MACROARCHITECTURE OF VGG16<sup>[5]</sup>

<https://www.cs.toronto.edu/~frossard/post/vgg16/>

# VGGNet

- VGG follows what we will now call a ‘Classical CNN’ approach where we have sequences of Conv Layers followed by a pooling layer.
- This is then repeated with increasing number of Feature Maps/Filters until we connect it to multiple FC Layers which then outputs our class probabilities using a softmax layer.



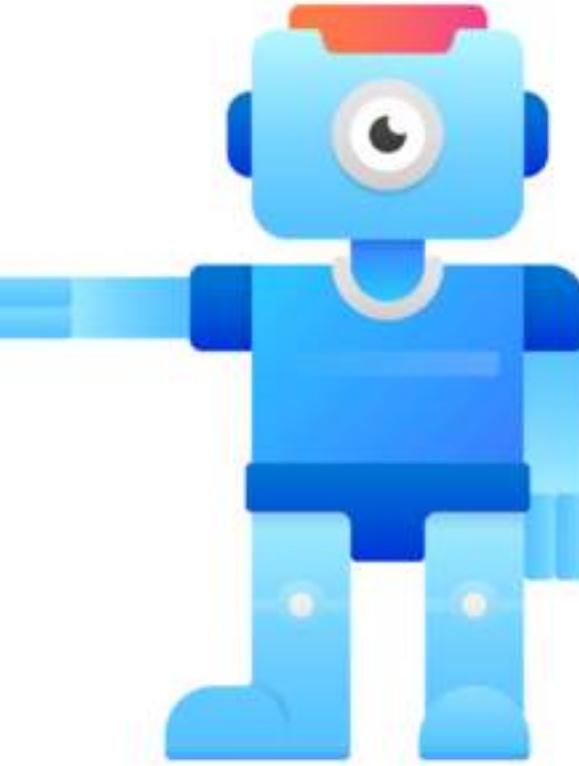
# VGGNet - Parameter Heavy

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as “conv<receptive field size>-<number of channels>”. The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
<b>LRN</b>		<b>conv3-64</b>	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
<b>conv3-128</b>		conv3-128	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
<b>conv3-256</b>		<b>conv3-256</b>	<b>conv3-256</b>	<b>conv3-256</b>	<b>conv3-256</b>
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
<b>conv3-512</b>		<b>conv3-512</b>	<b>conv3-512</b>	<b>conv3-512</b>	<b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144



# MODERN COMPUTER VISION

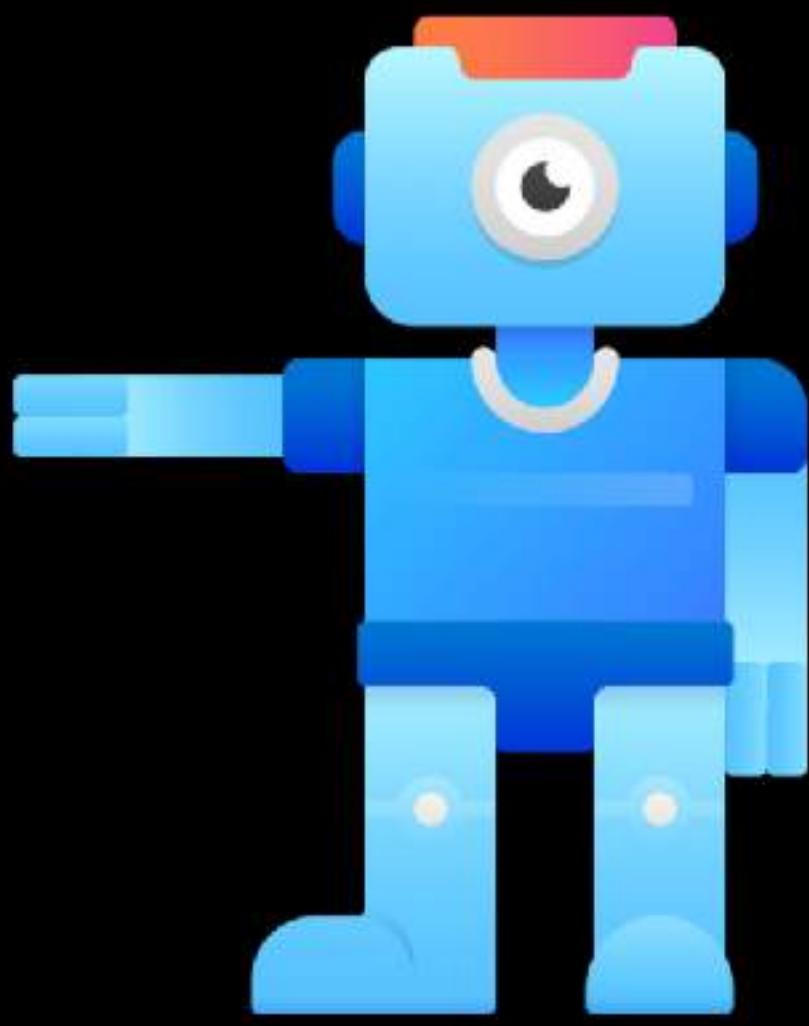
BY RAJEEV RATAN

# Next...

A look at ResNets

# ResNets

An overview of the ResNet or Residual Neural Networks

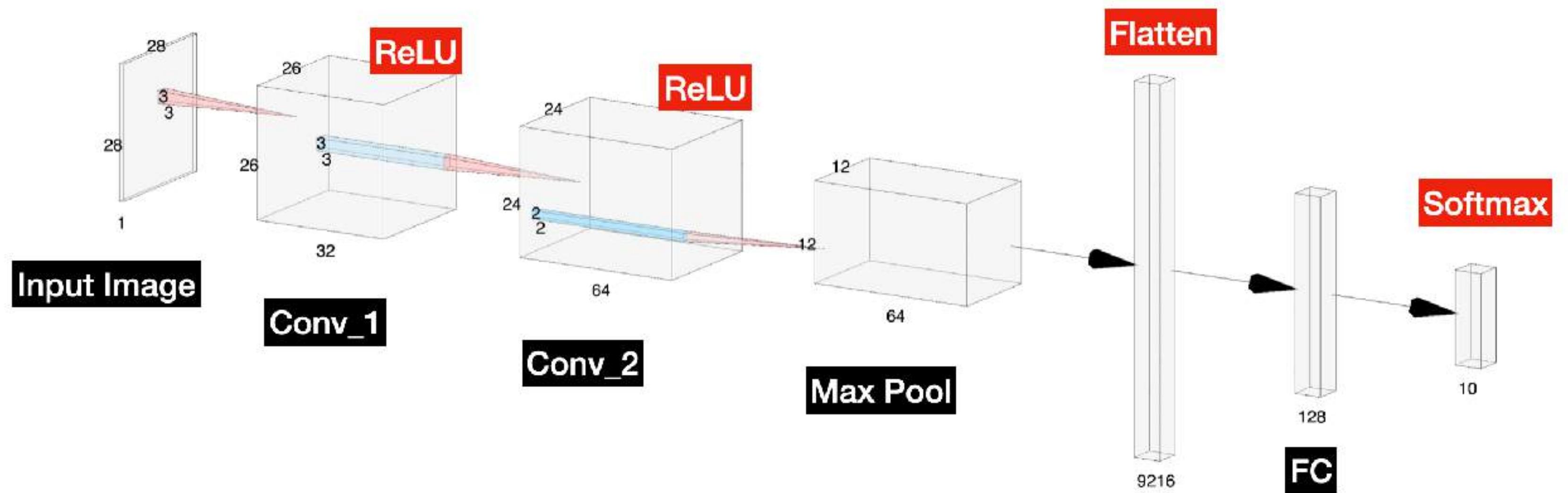


# MODERN COMPUTER VISION

BY RAJEEV RATAN

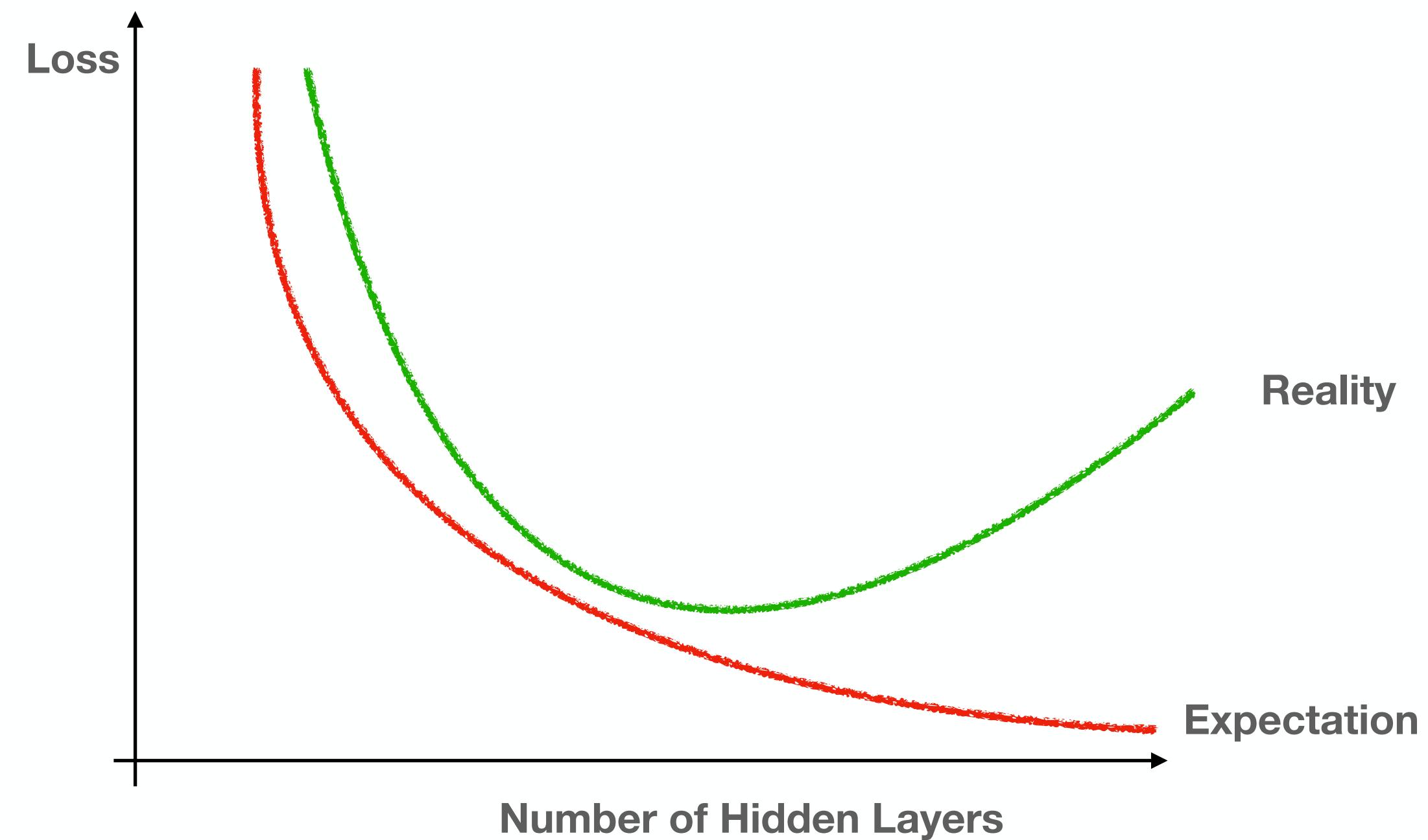
# Plain or Classical CNNs

Look like this...



- A linear sequential sequence of linear operations
- Problems?

# Classical CNNs Give Worse Performance if too Deep

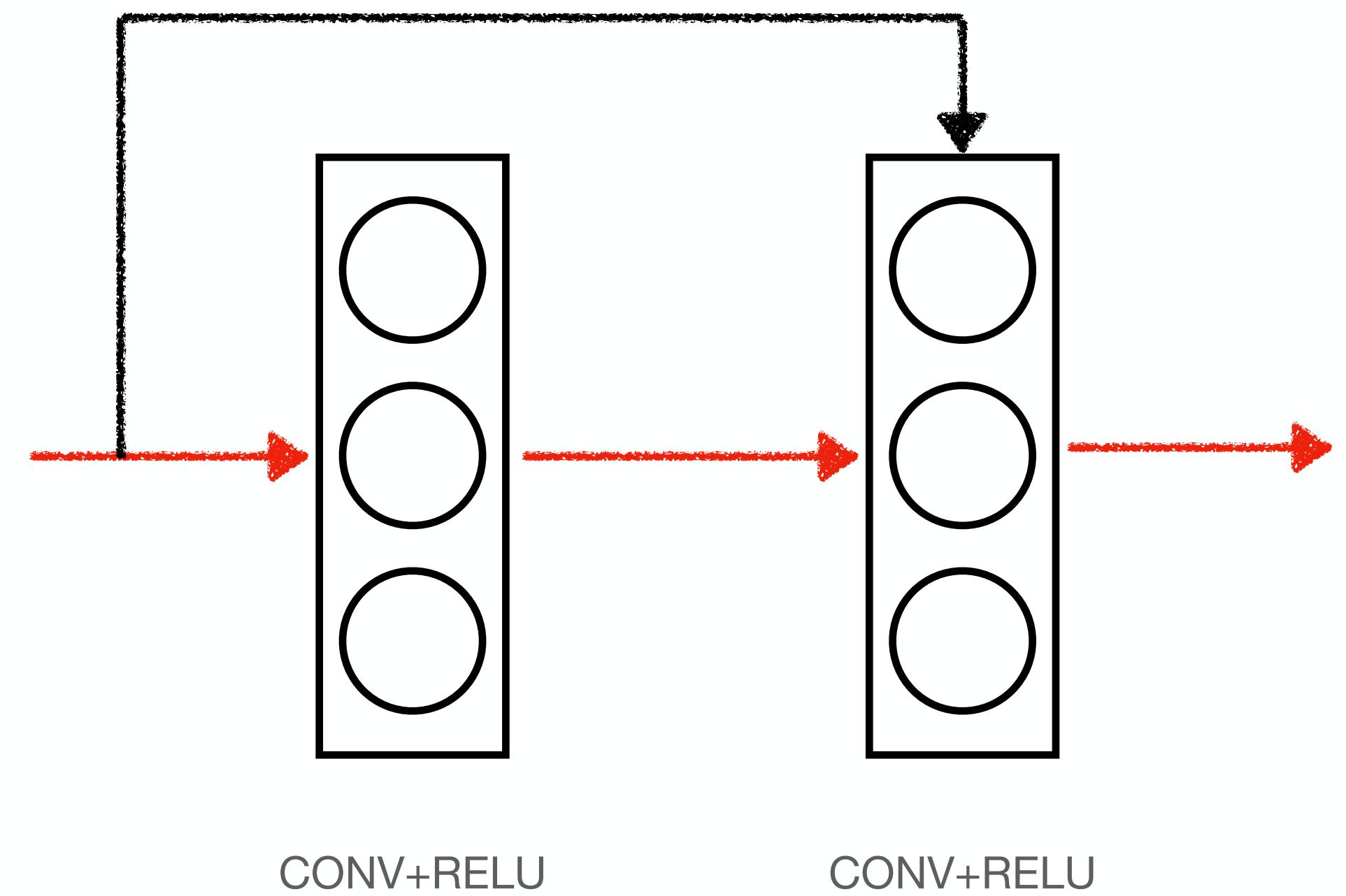


- In reality, a classical CNN architecture has lots of problems training if layers are very deep
- Often performance gets worse when number layers are used

# Very Deep CNNs Experience Exploding or Vanishing Gradients

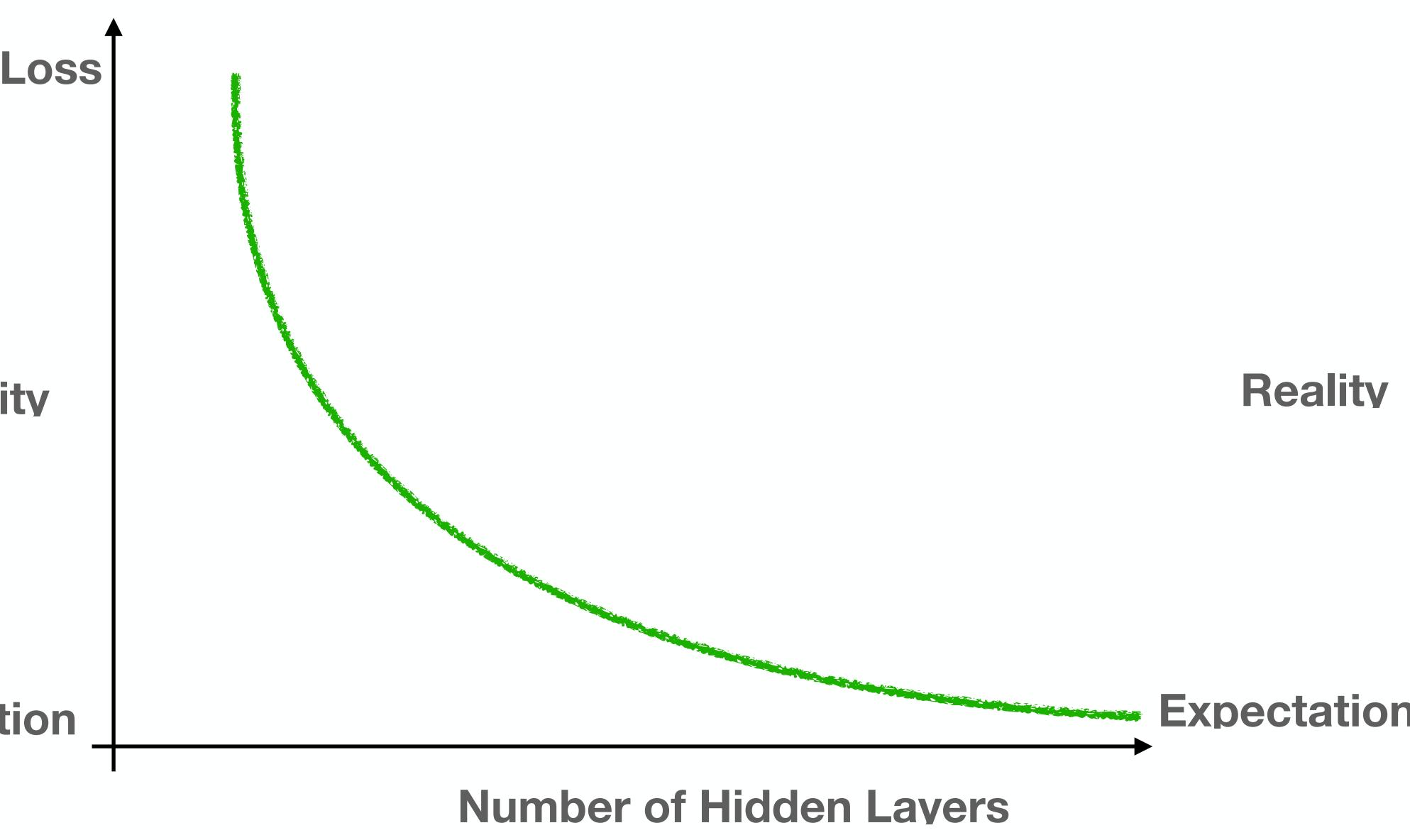
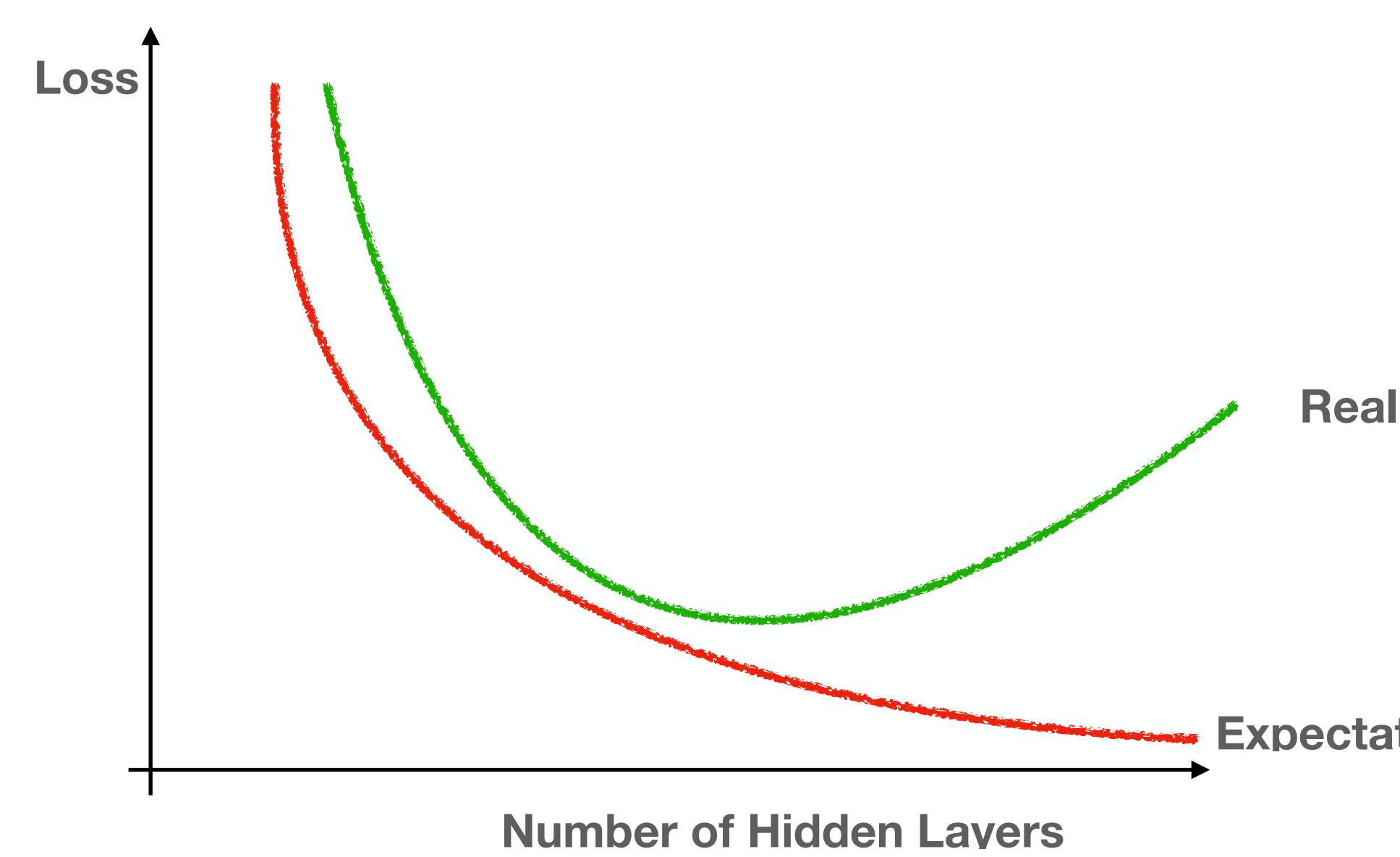
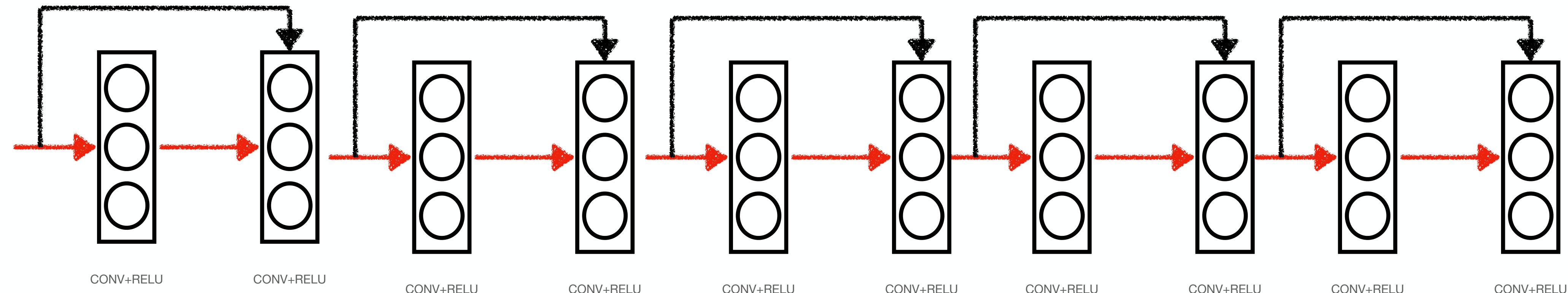
- In deep networks with  $N$  layers,  $N$  derivatives must be multiplied together to perform our gradient updates
- If a derivative is large, gradients increase exponentially or “**explode**”
- Likewise, if derivatives are small, they decrease exponentially or “**vanish**”

# ResNet of Residual Networks Help Solve This



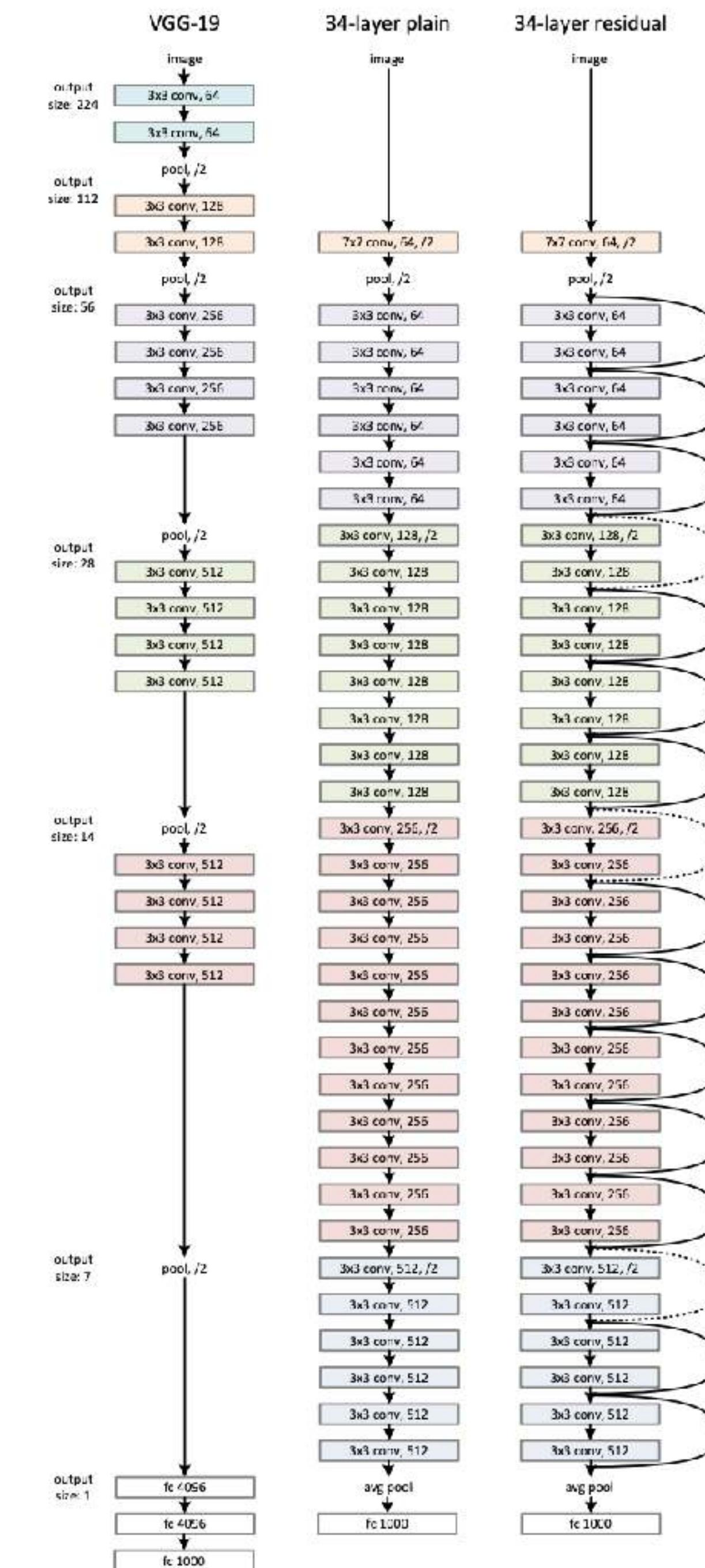
- We ‘Short Circuit’ the network by connecting the input to the previous layer to the layer ahead.

# ResNet of Residual Networks Help Solve This



# ResNets

- ResNets were introduced in 2015 by Microsoft Researchers in paper titled, Deep Residual Learning for Image Recognition from He et al.



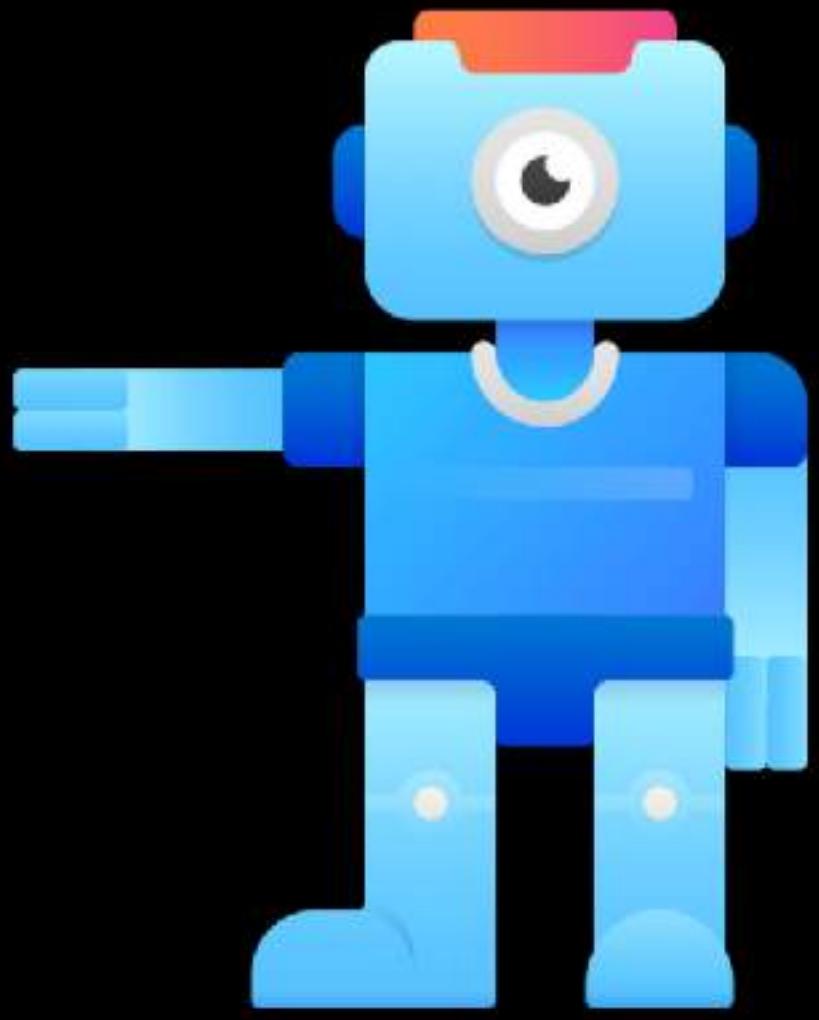


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Why to ResNets Work?**

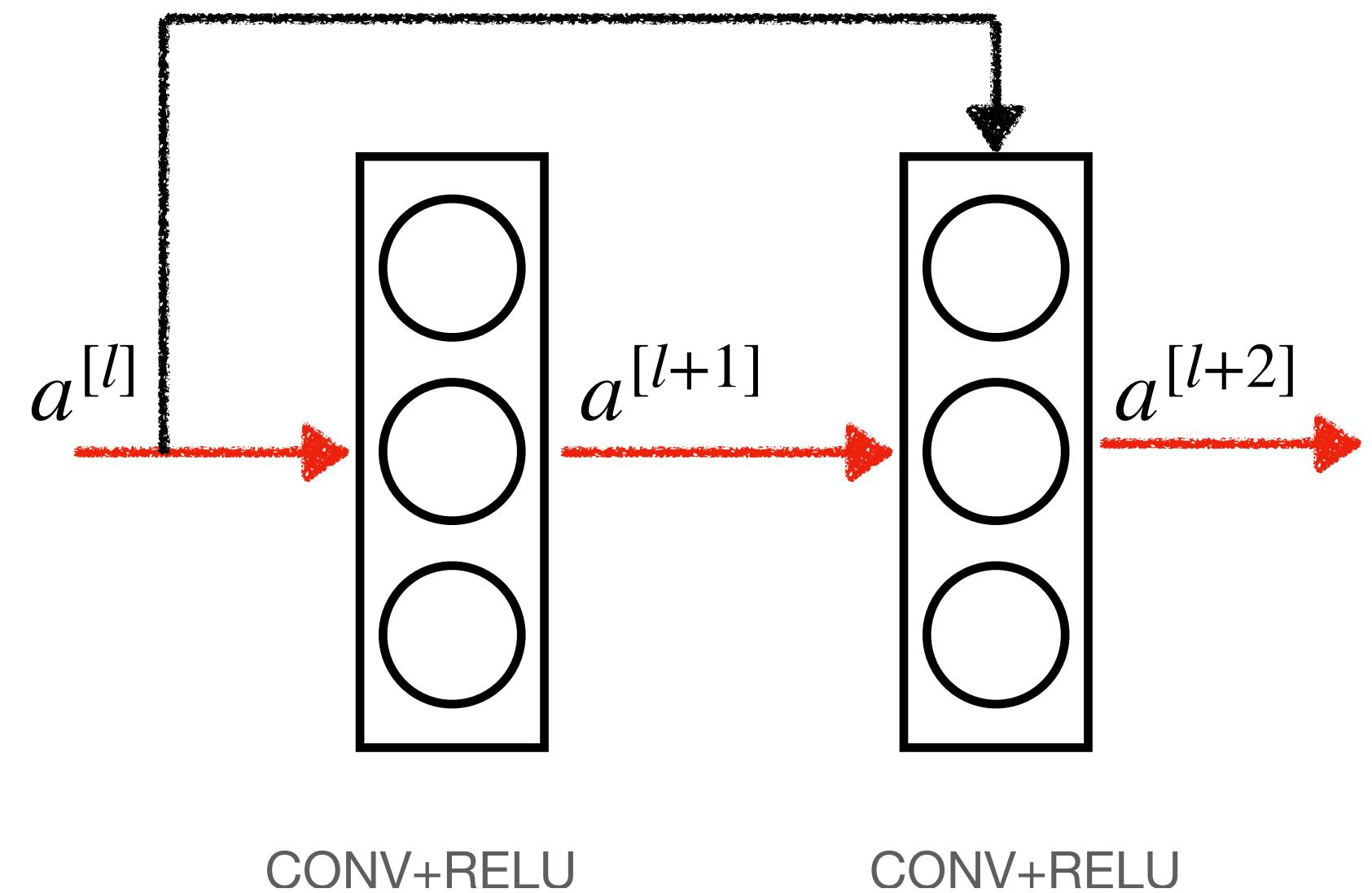


# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Why Do ResNets Work? How ResNets Solve the Vanishing Gradient Problem

# ResNet Mathematics



$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}$$

First Linear Operation

$$a^{[l+1]} = g(z^{[l+1]})$$

After ReLU operation

$$z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]}$$

Second Linear Operation

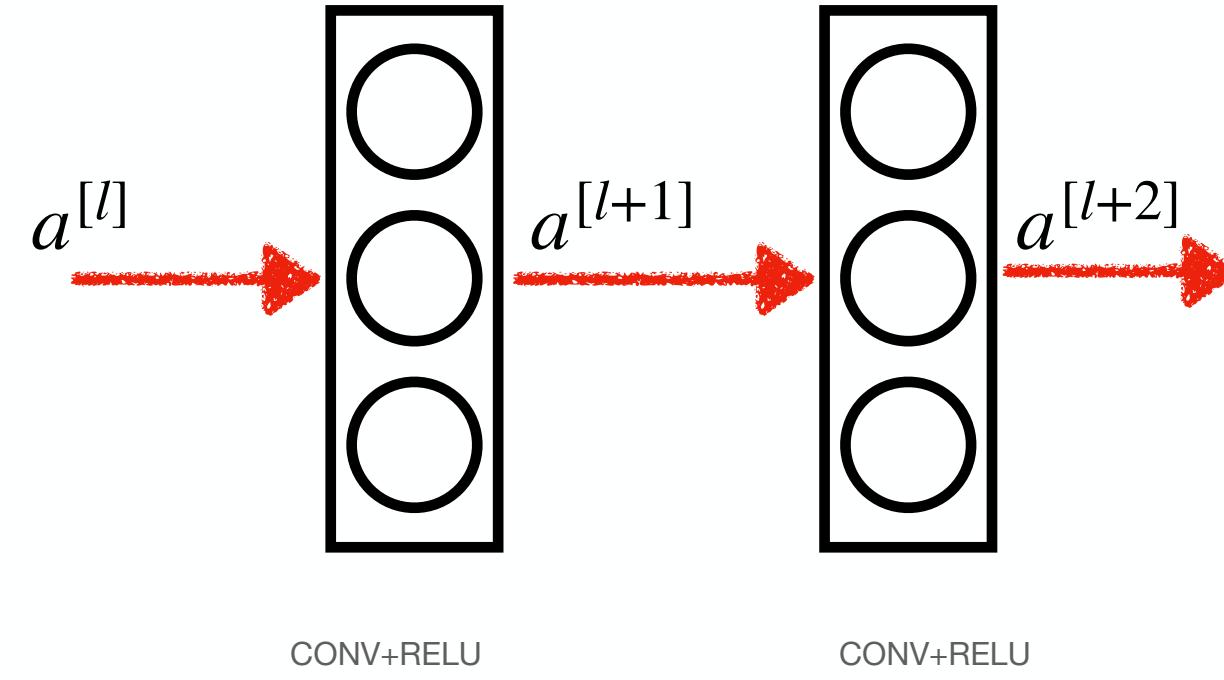
$$a^{[l+2]} = g(z^{[l+2]})$$

Output without Short Circuit

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

Output **with** Short Circuit

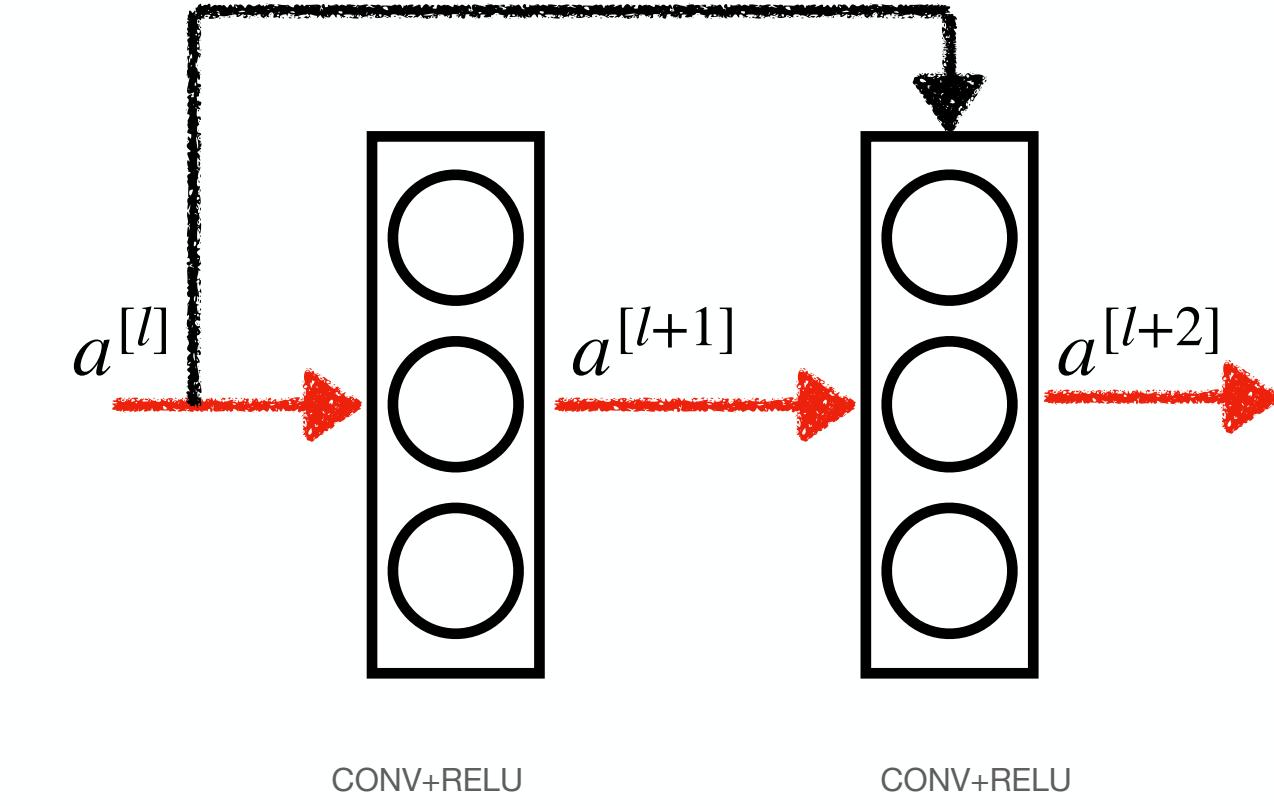
# ResNet Mathematics



Output **without** Short Circuit

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

$$a^{[l+2]} = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]})$$



Output **with** Short Circuit

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

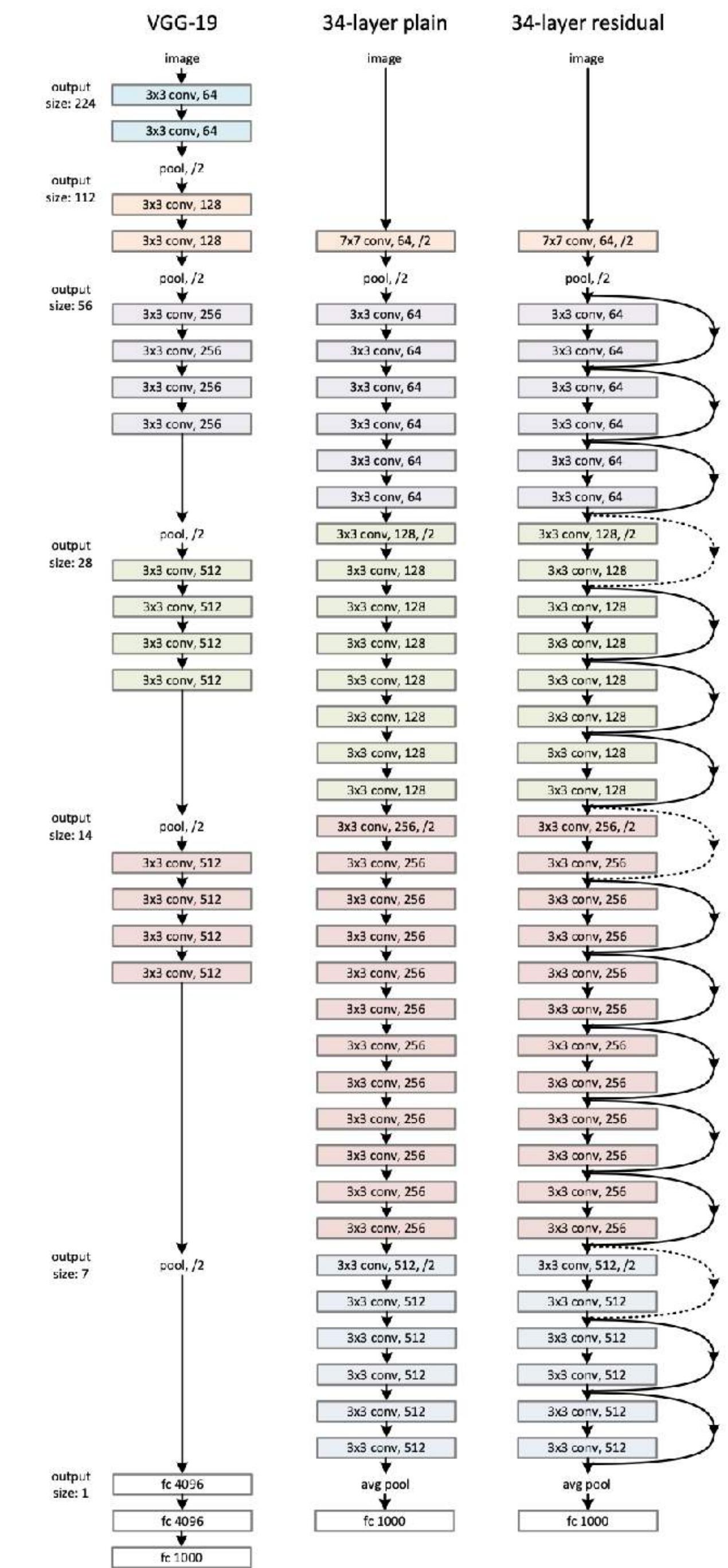
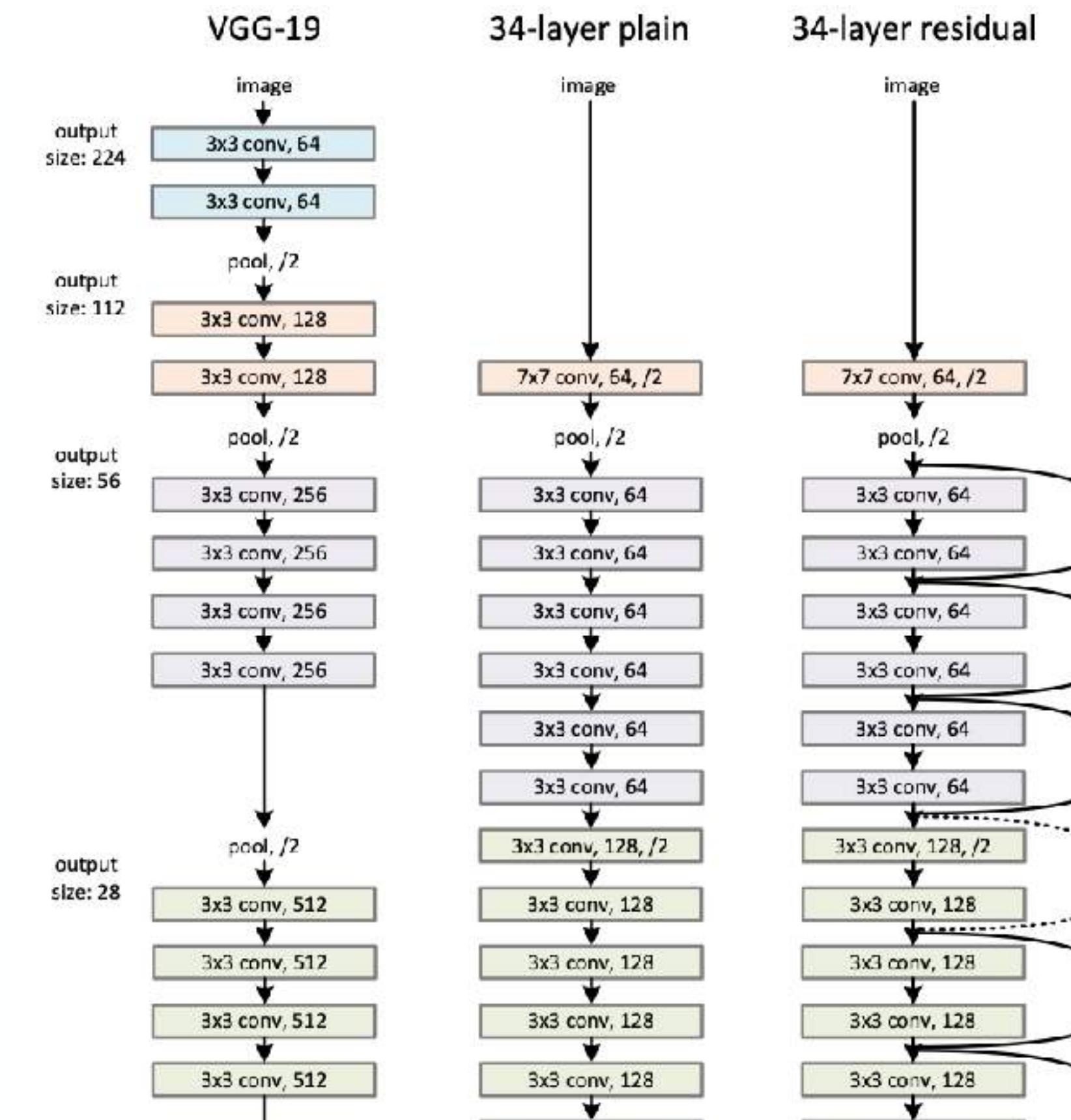
$$a^{[l+2]} = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

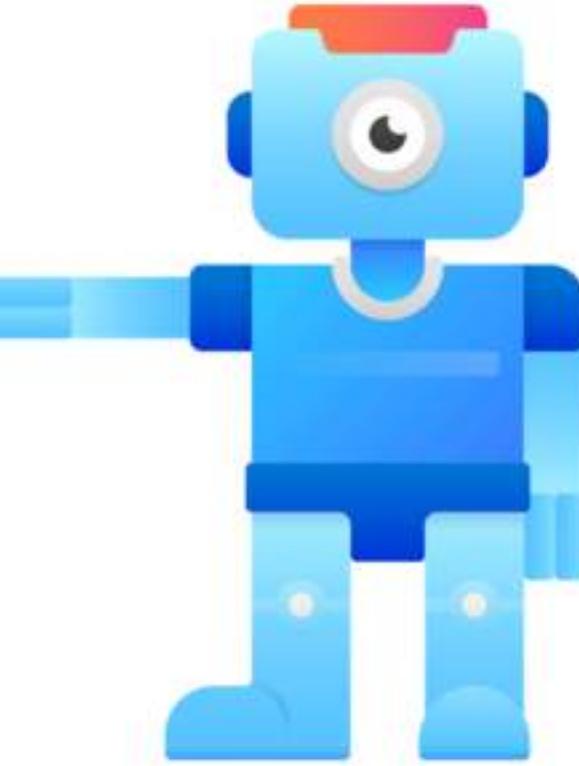
- If  $W$  or  $b$  is small or near 0,  $a^{[l+2]} = g(a^{[l]})$

- This means it allows the our Network to learn the Identity function, thus solving the **vanishing gradient** problem

# ResNet34 and ResNet50 Architectures

- ResNet34 and ResNet50 features several consecutive Conv layers of 3x3 with feature maps (64, 128, 256, 512) with bypasses every 2 Convolutions.
- Their output dimensions remain constant (padding same, stride =1)
- ResNet was built by several stacked residual units and developed with many different numbers of layers: 18, 34, 50, 101, 152, and 1202.



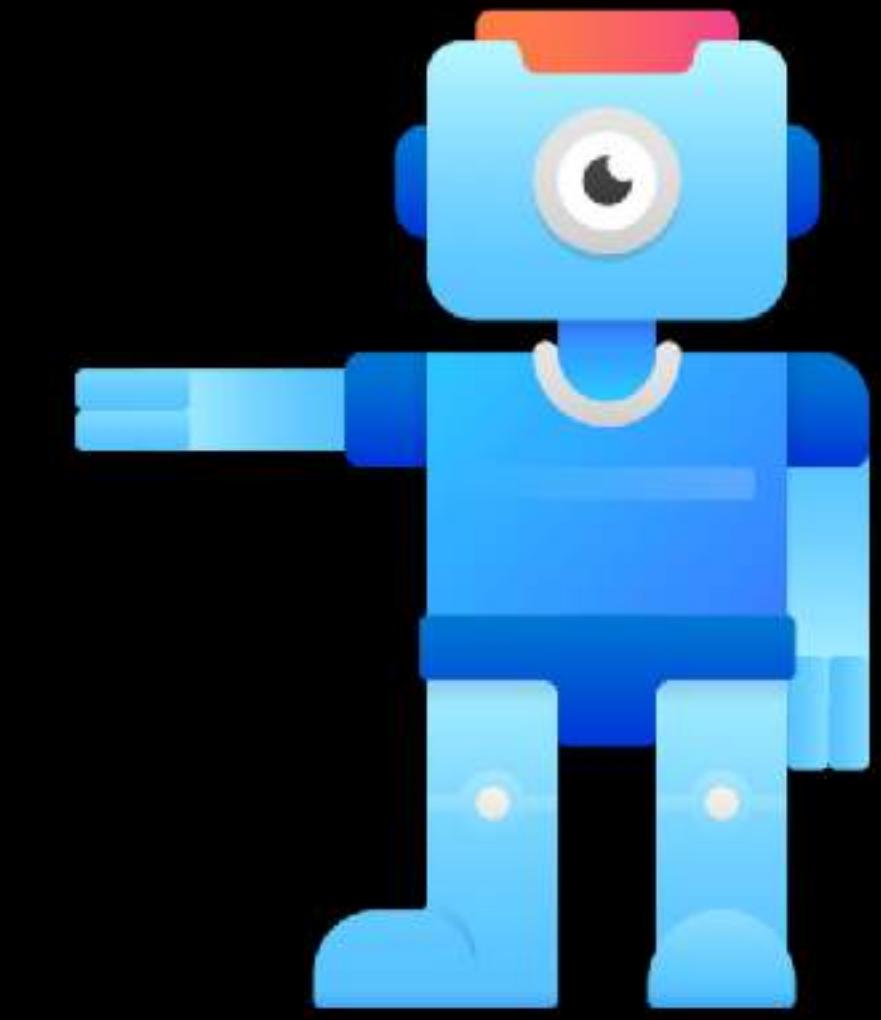


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Take a look at MobileNet**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## MobileNet

MobileNet a fast interference CNN module

# MobileNet

- MobileNet was a CNN architecture developed by Google in 2017.
  - <https://arxiv.org/pdf/1704.04861.pdf>
- MobileNet was designed to be an efficient lightweight CNN that could be used in embedded devices and mobile phones.
- Good CNNs are large (weights are 100mb+)
- Relatively slow on inference (i.e. forward propagation)

# MobileNet Use Cases

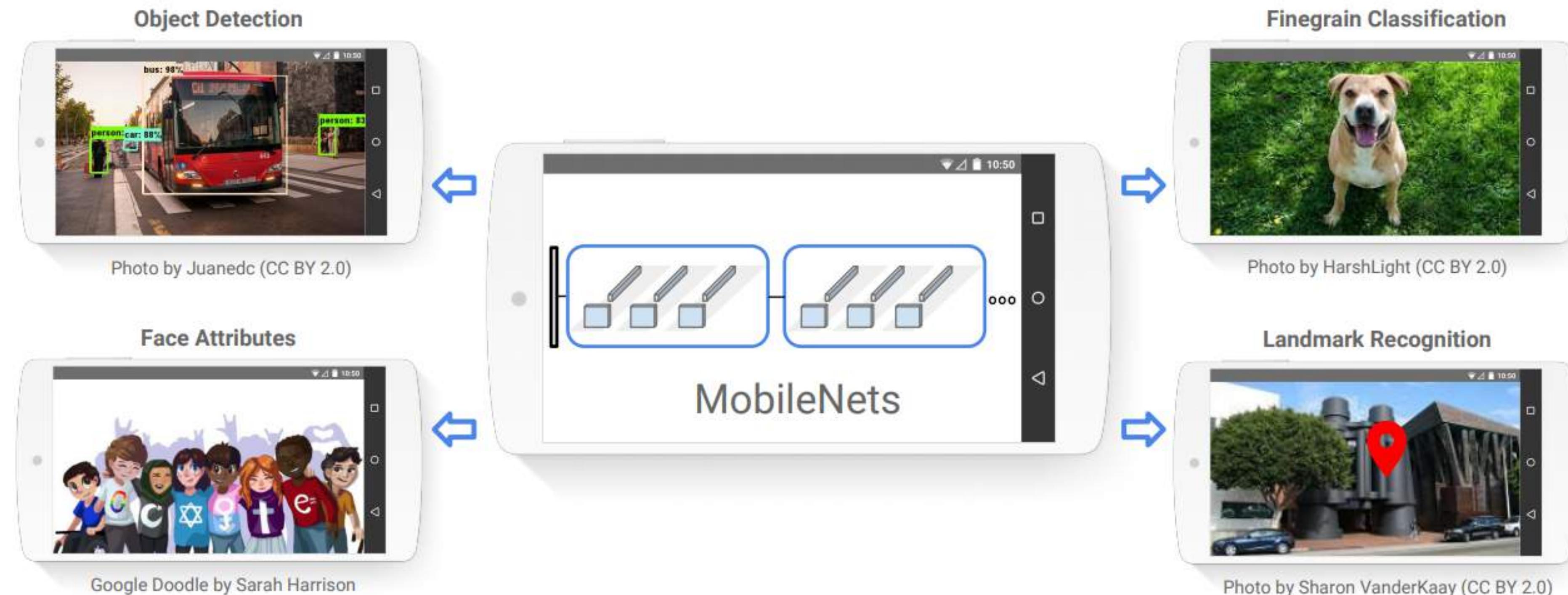
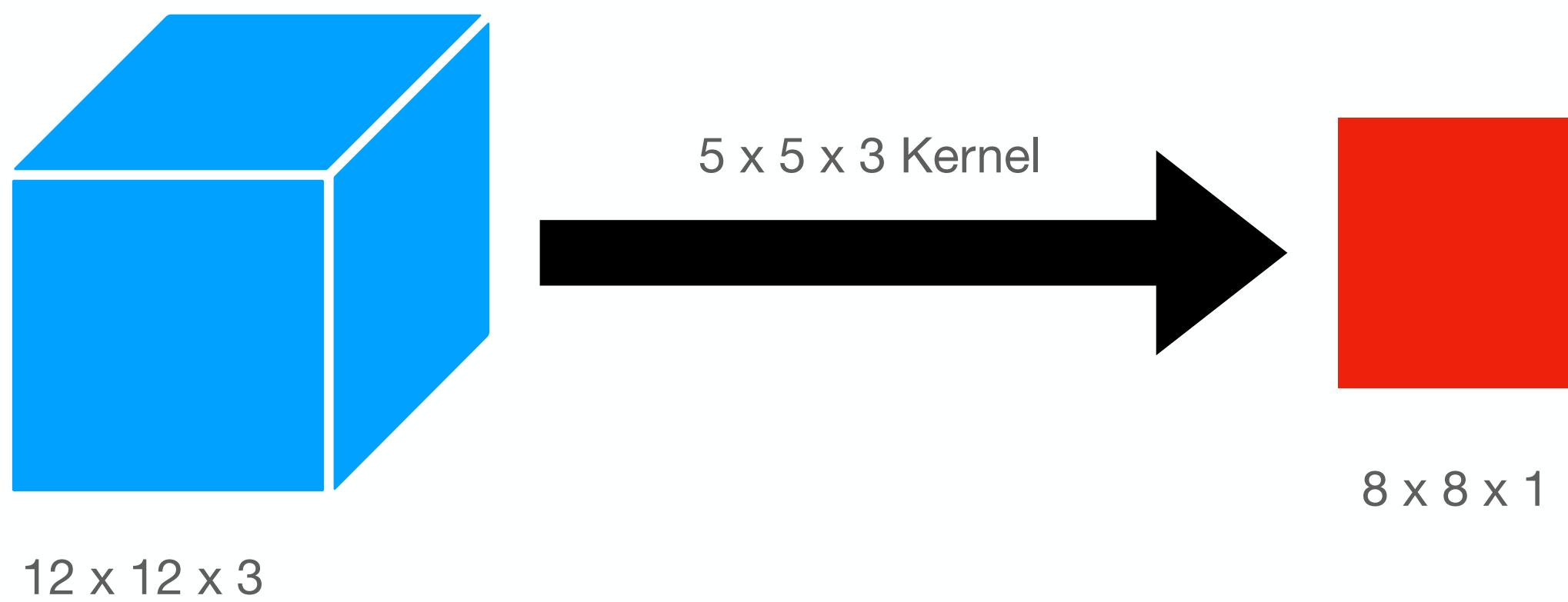


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

# Making CNNs for Mobile/Embedded Devices

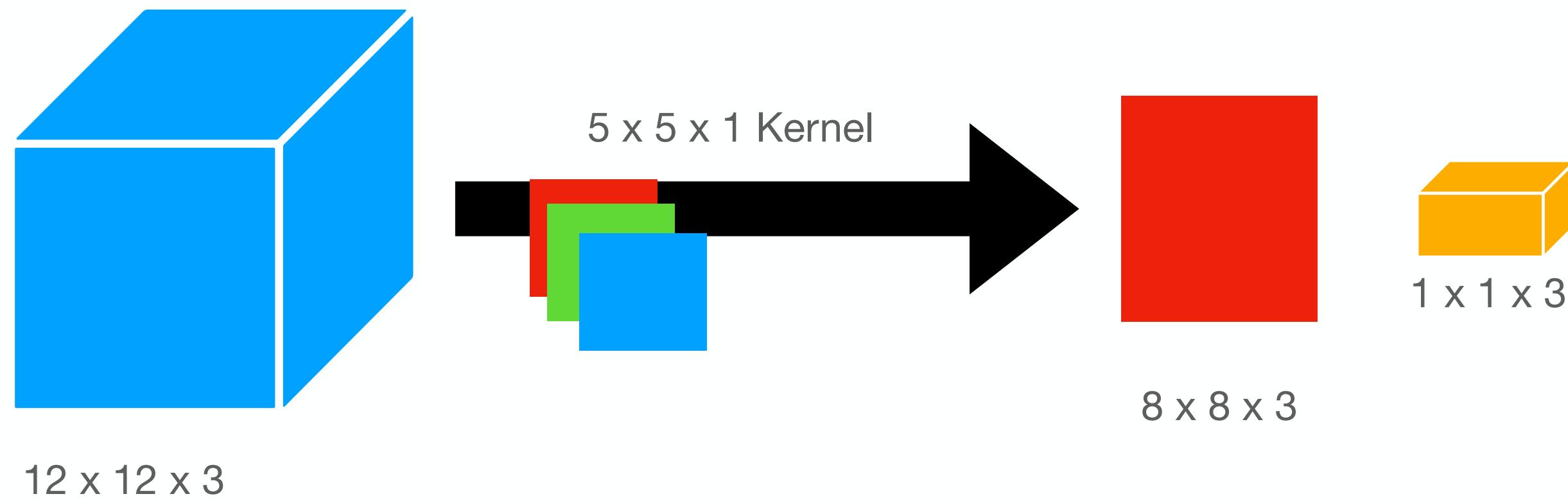
- Mobile or embedded systems typically have low computational power as they are made to be cheap and power efficient.
- Using CNNs on these devices required:
  - Training smaller models
  - Compressing existing models via methods such as Pruning, Distillation or low bit networks.
- MobileNet achieved the goal of being a good mobile phone model by using:
  - Depthwise Separable Convolutions
  - Two Hyper-Parameters

# Convolution Operations in Regular CNNs



- Input image of  $12 \times 12 \times 3$  convolved with a kernel of  $5 \times 5 \times 3$  produces a feature map of size  $8 \times 8 \times 1$
- Given a stride of 1, we have to perform  $5 * 5 * 3 * 64$  operations
- If we had 128 filters this ends up being
- $75 * 64 * 128 = 614,400$

# Depth Wise Convolutions



$$5 \times 5 \times 3 * 64 = 75 * 64 = 4,800$$

$$3 \times 64 \times 128 = 24,576$$

$$4800 + 24,576 = 29,376 \text{ Operations}$$

- We use 3 Filters of  $5 \times 5 \times 1$  and multiply each with a single channel from our input image ( $12 \times 12 \times 1$ )
- This gives us a  $8 \times 8 \times 3$  output
- **Pointwise** Convolutions are then used to get the same output shape
  - We then multiply by our output by a  $1 \times 1 \times 3$  layer
  - This is multiplied 64 times, this now gives us a  $8 \times 8 \times 1$  output
  - This is roughly 20X less Operations

# Two Hyper Parameters

- MobileNet also features two hyper parameters that effectively reduce the size of the model.
  - **Width Multiplier** - This thins the model at each layer
  - **Resolution Multiplier** - Reduces the input image size and thus reduces the internal representation of every subsequent layer.

**Table 8. MobileNet Comparison to Popular Models**

Model	ImageNet	Million
	Accuracy	Parameters
1.0 MobileNet-224	70.6%	4.2
GoogleNet	69.8%	6.8
VGG 16	71.5%	138

**Table 9. Smaller MobileNet Comparison to Popular Models**

Model	ImageNet	Million
	Accuracy	Parameters
0.50 MobileNet-160	60.2%	1.32
SqueezeNet	57.5%	1.25
AlexNet	57.2%	60

**Table 10. MobileNet for Stanford Dogs**

Model	Top-1	Million
	Accuracy	Parameters
Inception V3 [18]	84%	23.2
1.0 MobileNet-224	83.3%	3.3
0.75 MobileNet-224	81.9%	1.9
1.0 MobileNet-192	81.9%	3.3
0.75 MobileNet-192	80.5%	1.9

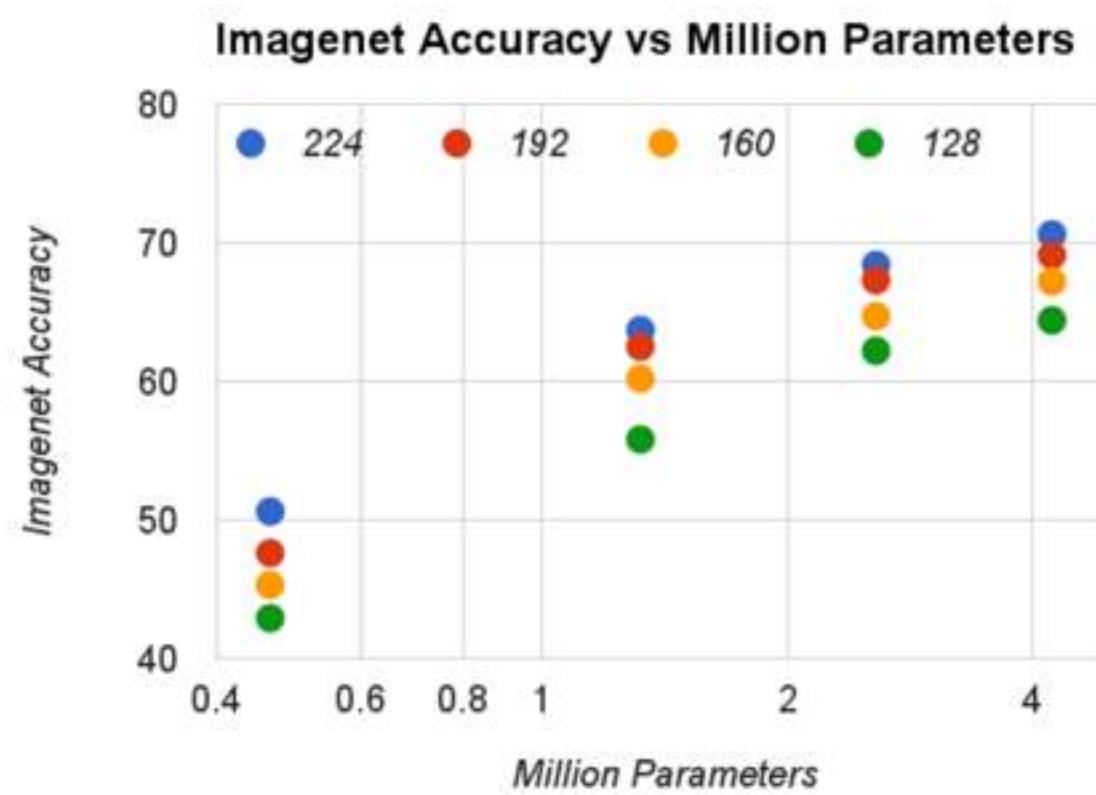


Figure 5. This figure shows the trade off between the number of parameters and accuracy on the ImageNet benchmark. The colors encode input resolutions. The number of parameters do not vary based on the input resolution.



# MODERN COMPUTER VISION

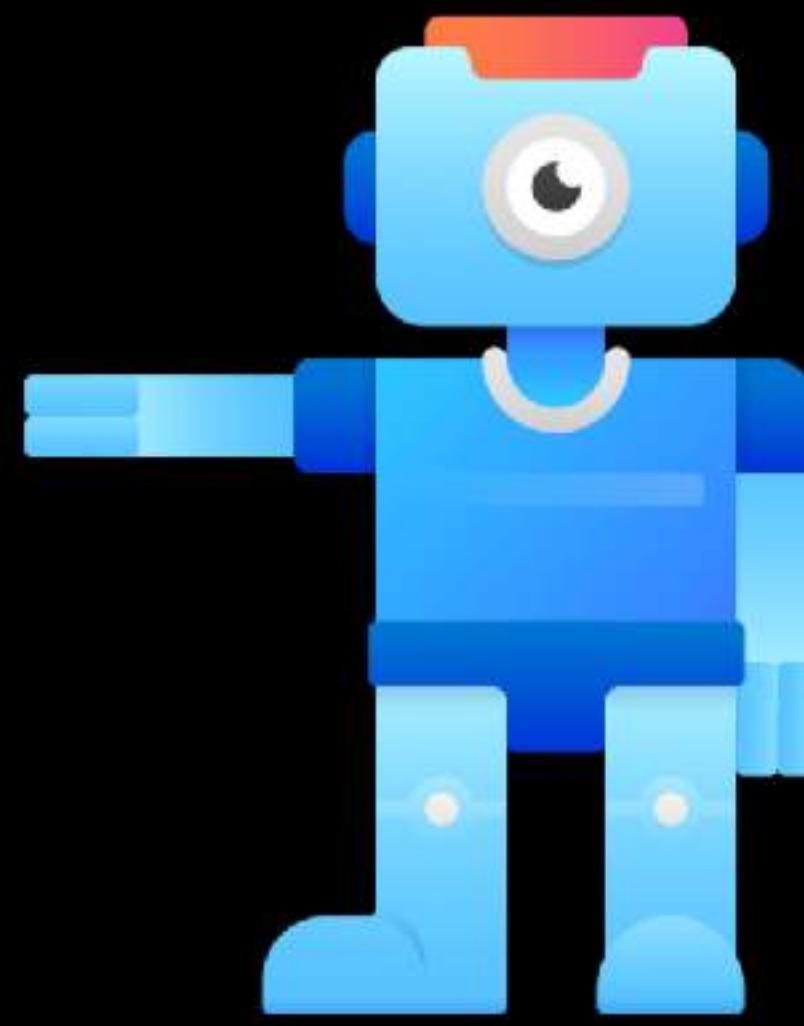
BY RAJEEV RATAN

# Next...

Take a look at Inception Network

# Inception Network

We explore the novel Inception Network or GoogleLeNet Architecture



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Inception Network Motivation

- As you've seen there's a lot of parameter tweaking involved in CNNs
- Filter Sizes, stride, depth, padding, FC layers etc.
- Inception aimed to solve the Filter size selection problem.

# Inception Network

- The Inception V1 (aka GoogleLeNet) Network was introduced by Google (Szegedy et al) in 2014
- It achieved state of the art performance in the ImageNet (ILSVRC14) Challenge

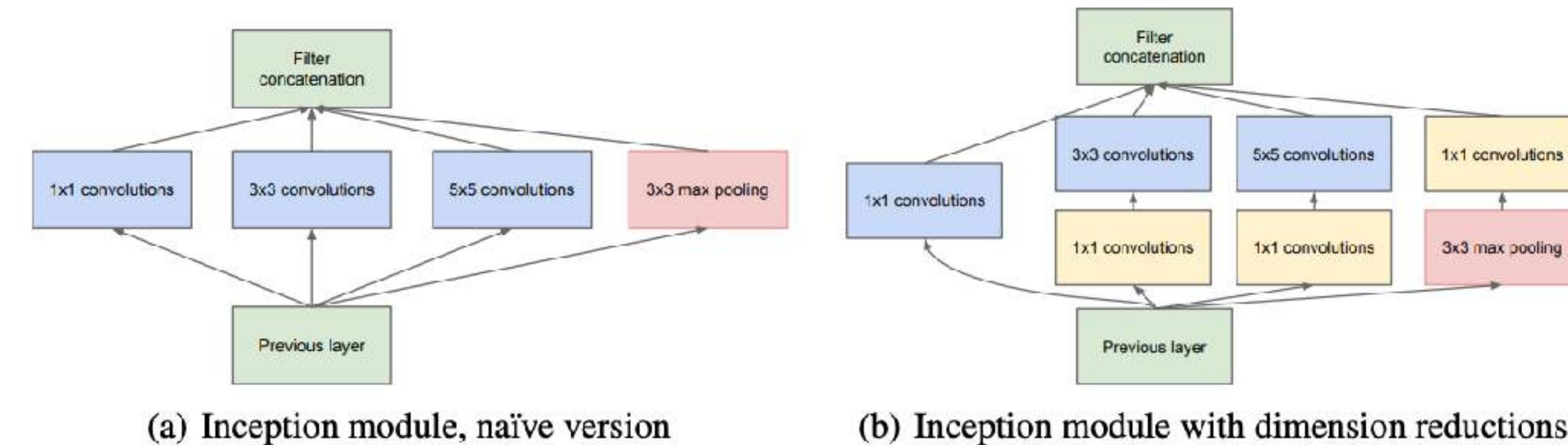
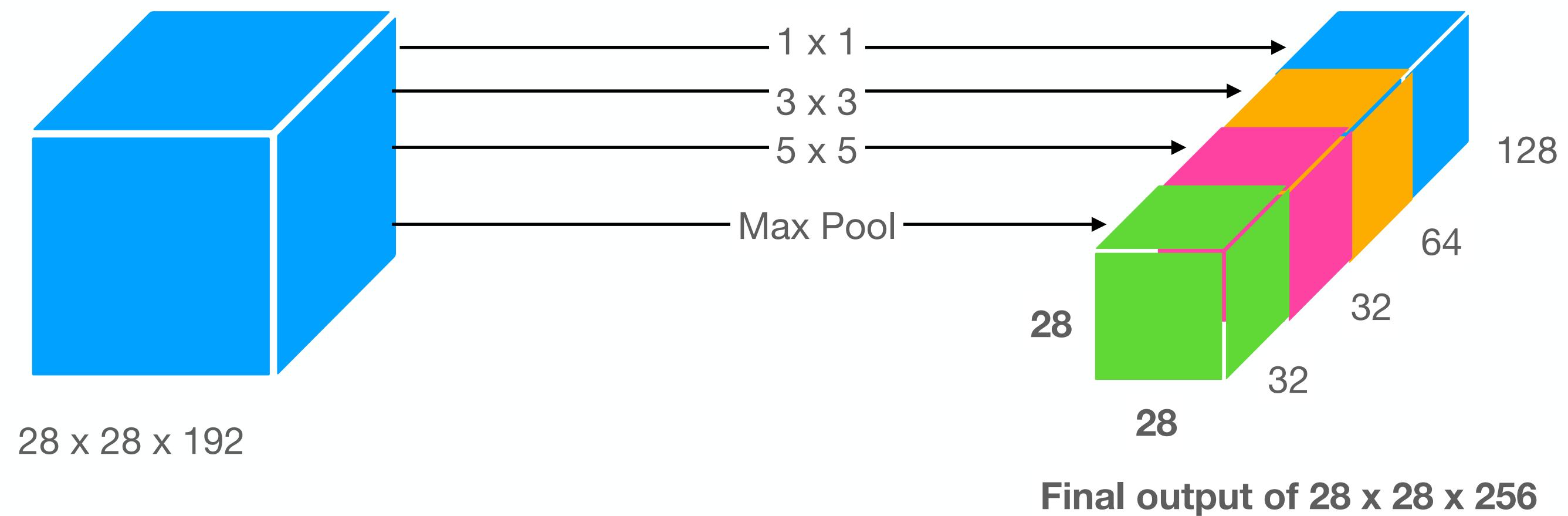


Figure 2: Inception module

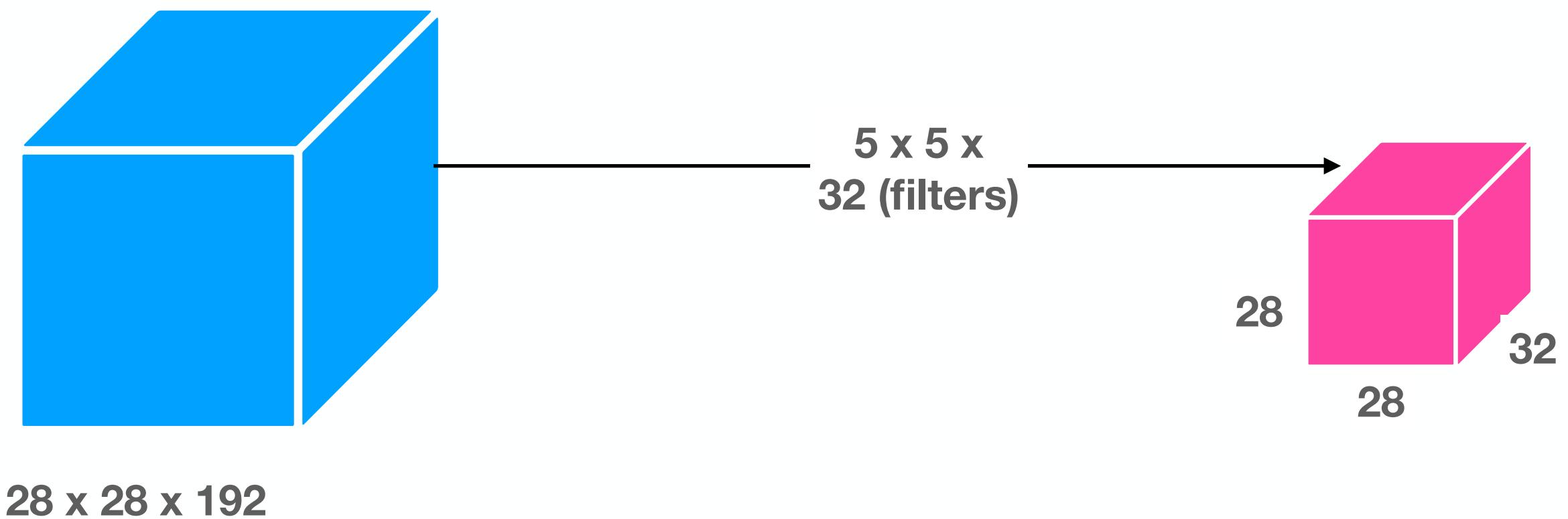
# Inception Network

- The Inception Network allows us to use several different sized Conv filters



- Use ‘same’ padding and stride of 1 to maintain dimension size consistency
- We can now do all filter sizes and even a max pool and just stack them together
- This allows the network to learn a combination of high-level and low-level features

# Inception Network Problem - Computation

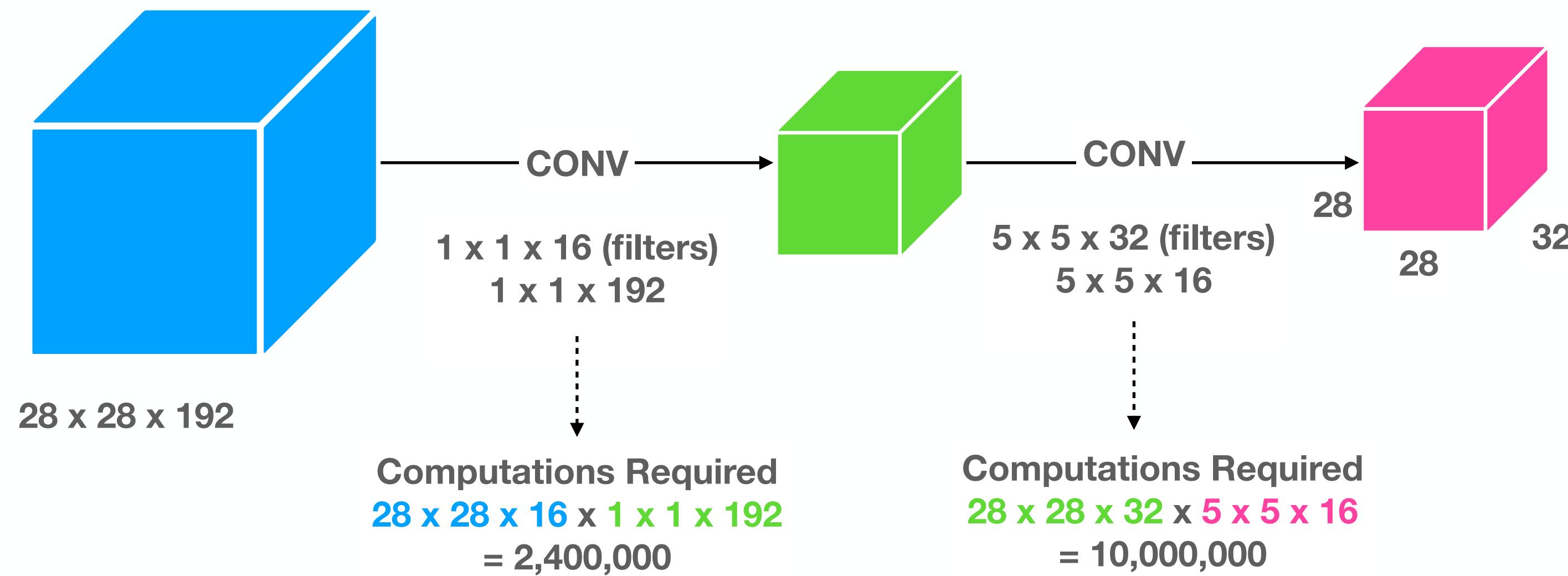


Computations Required  
 $28 \times 28 \times 32 \times 5 \times 5 \times 192$   
 $= 120,000,000$

- We'll use  $1 \times 1$  Convolutions to reduce the computation cost

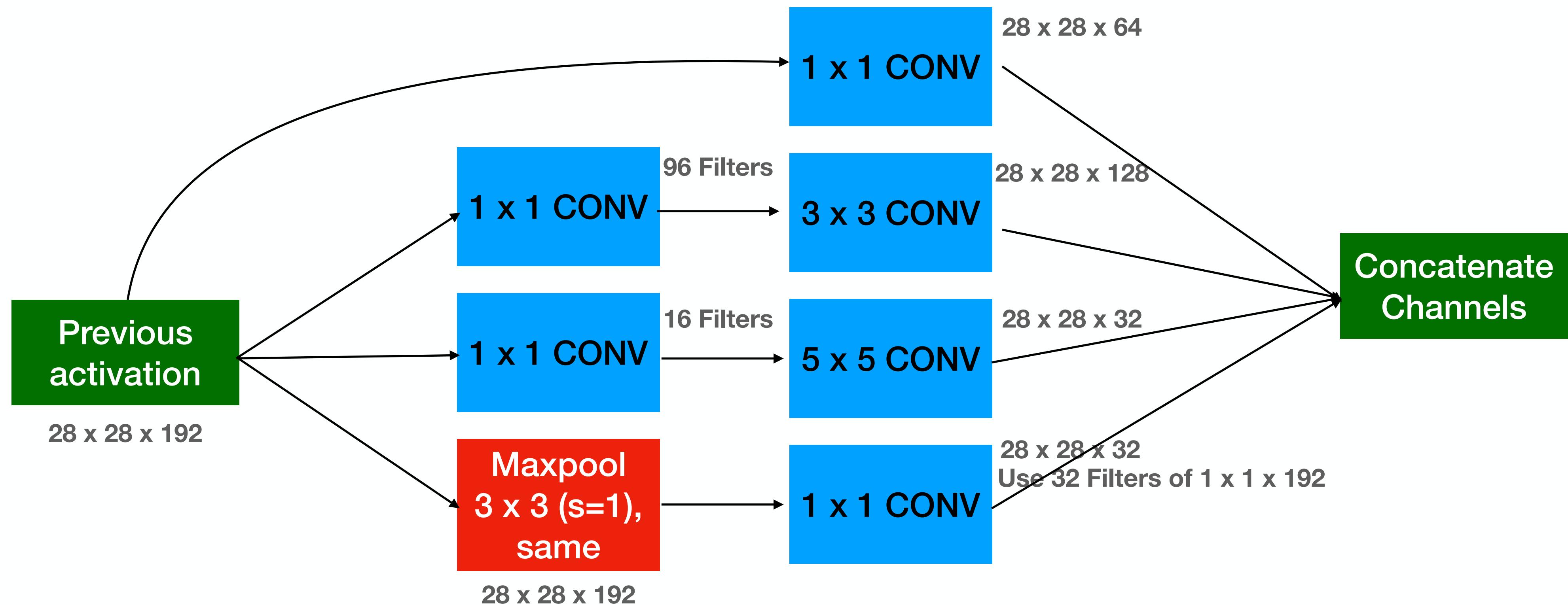
# Intermediate Volumes using $1 \times 1$ Convolution

## Using a Bottleneck Layer

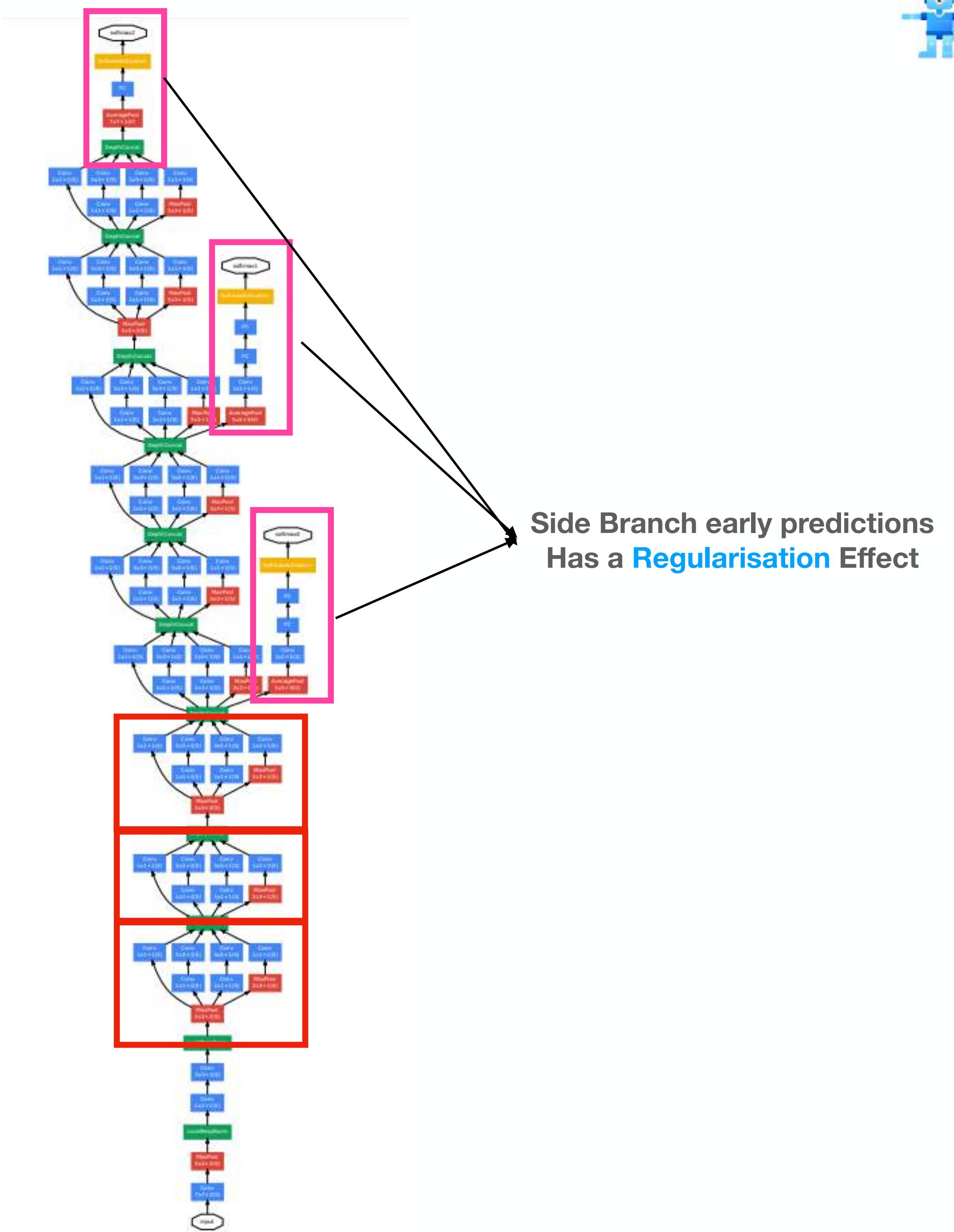


- We shrink the representation and then increase the size after
- This saves us 10X computation costs as it's now only  $2.4M + 10M = 12.4M$  vs **120M** previously

# Inception Block



# Inception Design



# History of Inception Model Work

- **InceptionV1** - 2014 - Network in Network
  - <https://arxiv.org/pdf/1312.4400v3.pdf>
- **InceptionV2**- Going Deeper with Convolutions - 2014
  - <https://arxiv.org/pdf/1409.4842v1.pdf>
- **InceptionV3** - Rethinking the inception architecture for computer vision - 2015
  - <https://arxiv.org/pdf/1512.00567v3.pdf>
- **InceptionV4** - v4: Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Szegedy et al. (2016)
  - <https://arxiv.org/pdf/1602.07261v2.pdf>
- Good History here - <https://nicolovaligi.com/history-inception-deep-learning-architecture.html>

# Fun Fact - Why is Called Inception?



- Meme was actually cited in the paper where the authors made reference to needing CNNs to be deeper, the Inception Network allows us to use deeper networks quite effectively.



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

Let's take a look at SqueezeNet

**SqueezeNet**  
A smaller CNN that maintains good accuracy



# MODERN COMPUTER VISION

BY RAJEEV RATAN

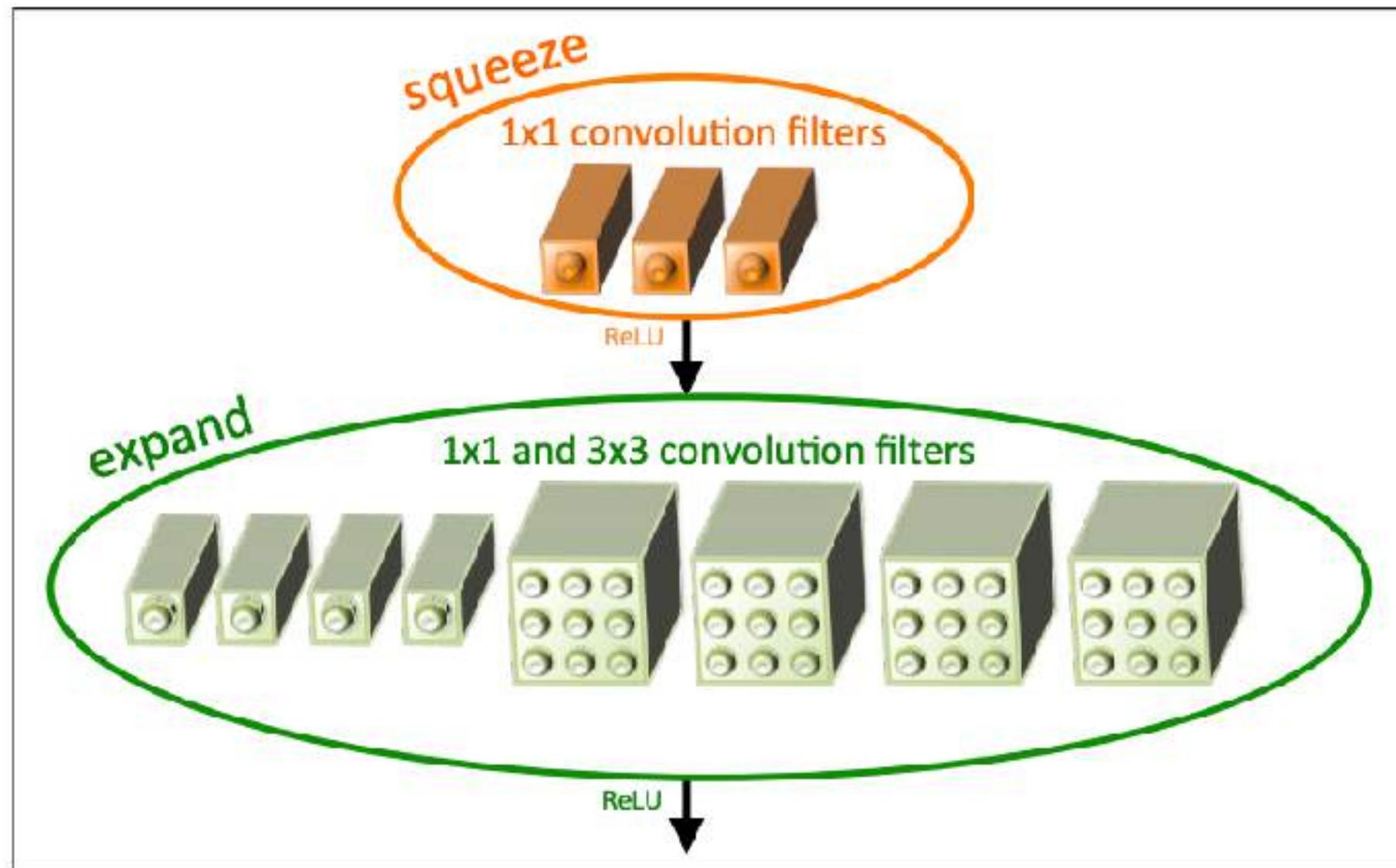
# SqueezeNet

- Introduced in 2016 by researchers at the Berkeley, University of California, Stanford and DeepScale.
- Aim was to make a highly accurate by small CNN. Why?
  - Less communication across servers during distributed training
  - Smaller size allows it to be used on embedded systems like FPGAs
  - Faster to update models via the cloud (less bandwidth)
- It had 50x less parameters than AlexNet and performed 3X faster

# SqueezeNet Key Takeaways

- **Replace 3x3 Filters with of 1x1** - has 9X fewer parameters than a 3x3 filter
- **Decrease the number of input channels to 3x3 filters** - Remember the quantity of parameters in a layer is (input channels \* number of filters \* 3 \* 3).
- **Downsample later in the network so that convolution layers have larger activation maps**

# Fire Module - Squeeze and Expand Layers



<https://arxiv.org/pdf/1602.07360.pdf>

- The Fire Module has a squeeze convolution layer (1x1 filters) that feed into an expand layer (mix of 1x1 and 3x3 convolution filters)

# Fire Model - SqueezeNet's Squeeze and Expand Layers

The full SqueezeNet Architecture consists of a standalone Conv Layer followed by **8 fire** modules.

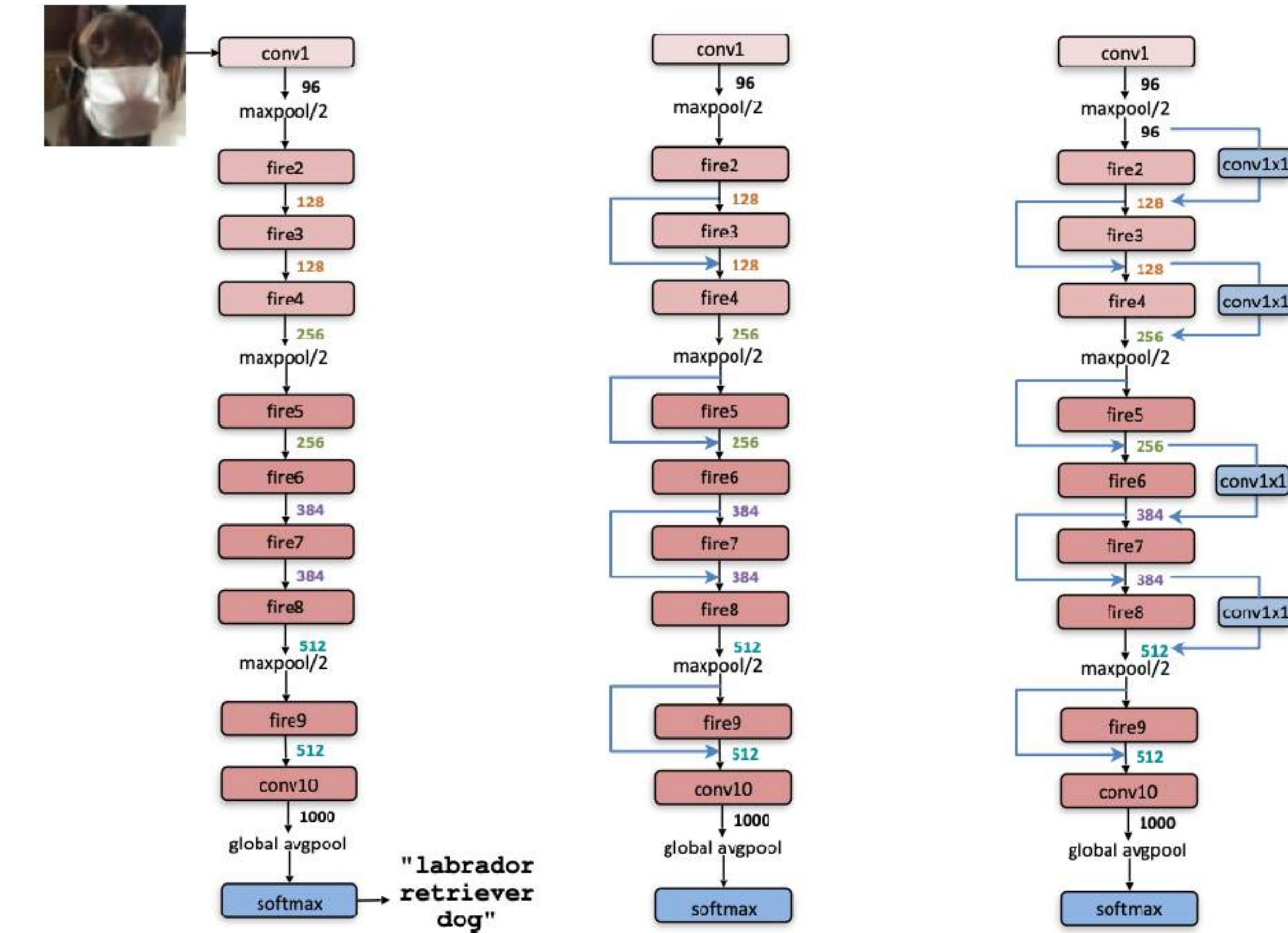
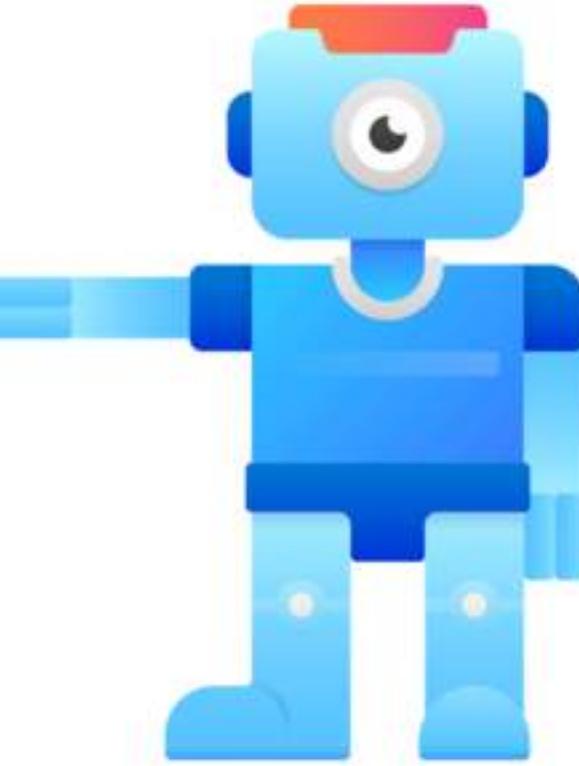


Figure 2: Macroarchitectural view of our SqueezeNet architecture. Left: SqueezeNet (Section 3.3); Middle: SqueezeNet with simple bypass (Section 6); Right: SqueezeNet with complex bypass (Section 6).

# SqueezeNet's Performance

Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	<b>50x</b>	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	<b>363x</b>	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	<b>510x</b>	57.5%	80.3%

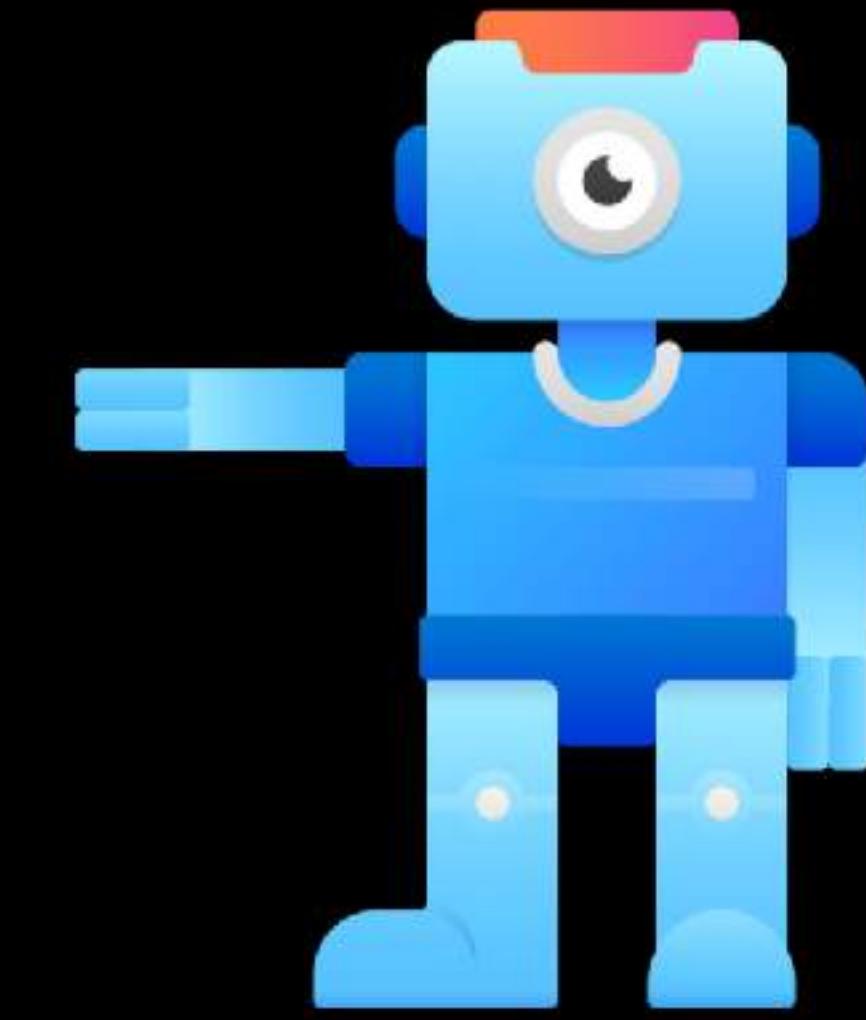


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Let's take a look at Goolge's EfficientNet**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## EfficientNet

Improving Accuracy and Efficiency through AutoML and Model Scaling

# EfficientNet

- Introduced in 2019 by researchers at Google (*Mingxing Tan and Quoc V. Le*)
- Motivation behind EfficientNet:
  - CNNs are typically designed at a fixed resource cost and then scaled up (e.g. ResNet-18 to ResNet-200)
  - Scaling works by either increasing **depth** (number of layers) or **width** (number of filters)
  - This is often tedious to experiment with, requires manual tuning and results in suboptimal results
- What if there was a more principled method to scale up CNNs?

# EfficientNet's Scaling Method **Compound Coefficient**

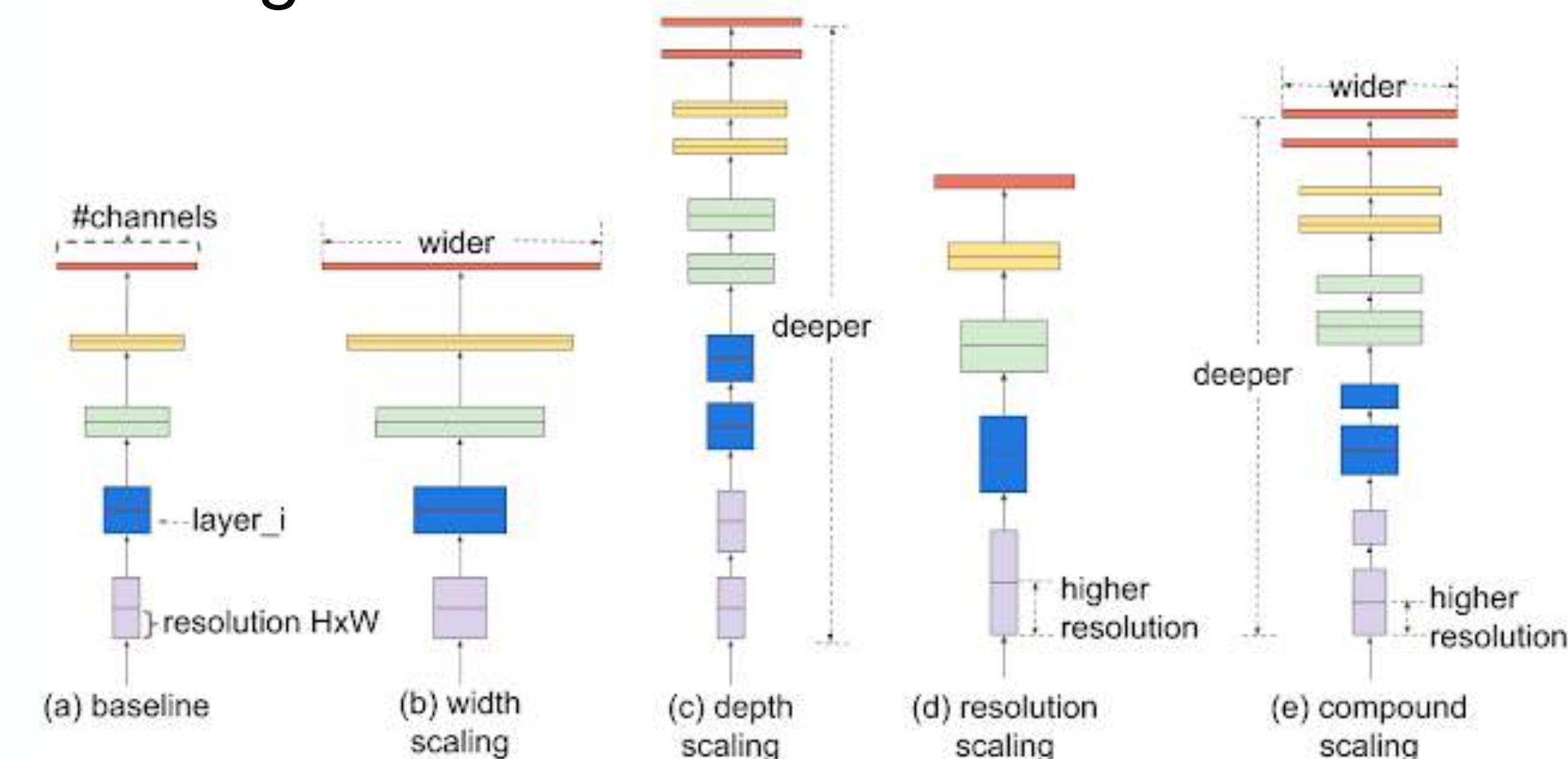
- This method uniformly scales each dimension (width, depth, resolution) with a fixed set of scaling coefficients
- It utilizes Google's new AutoML
- The EfficientNet family of models are able to surpass state-of-the-art accuracy with 10X better efficiency

# Scaling

- Researchers systematically studied the effects of scaling up different dimensions.
- It was found that balancing scaling in all dimensions resulted in the best overall performance

# Grid Search

- **Grid Search** was used to find the relationship between different scaling dimensions of the baseline network under a fixed resource constraint (e.g. 2X more FLOPS).
- This determines the most **appropriate** scaling coefficient for each of the dimensions
- The coefficients are then **applied to scale up** the baseline network to the desired target model size or computational budget



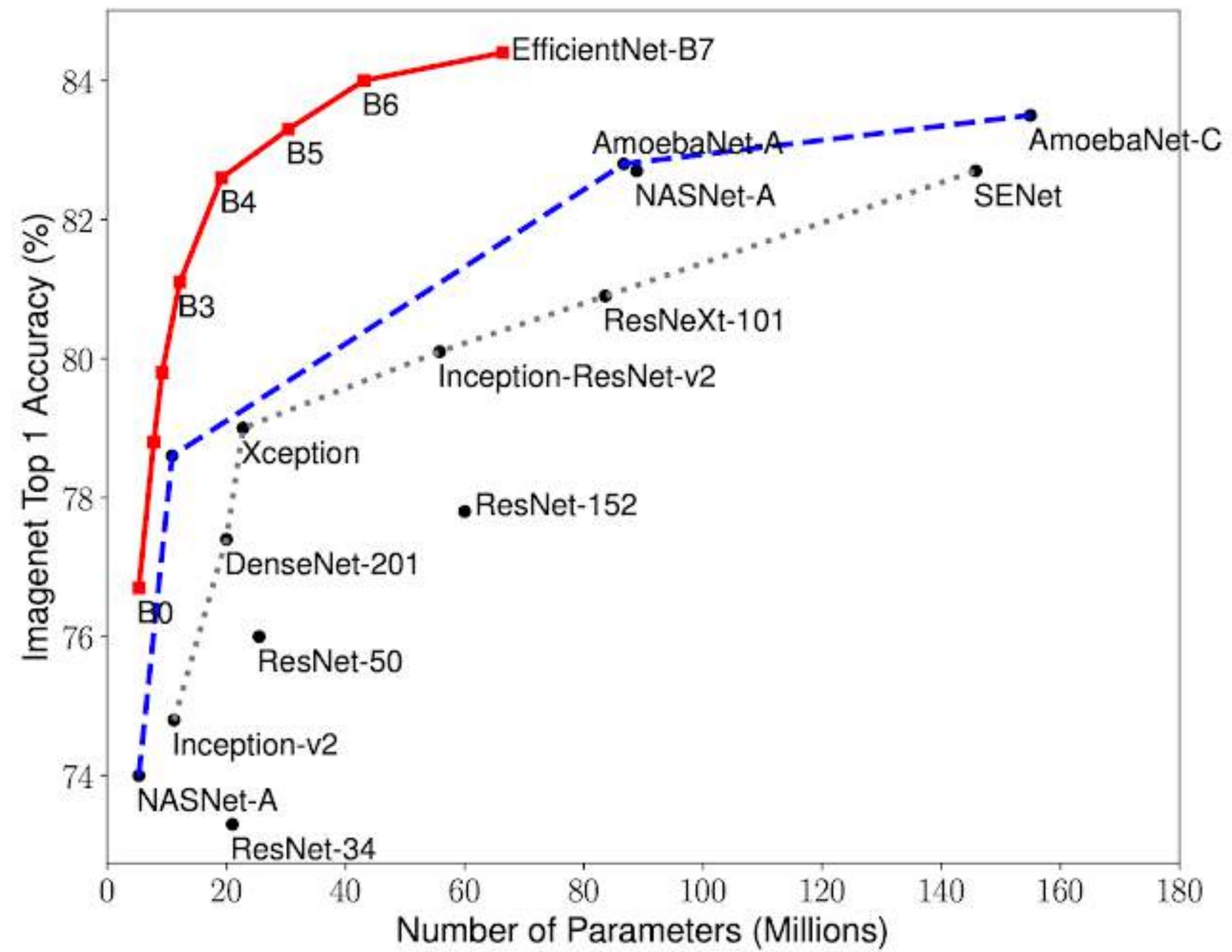
# EfficientNet Architecture

- The compound scaling methodology can be **applied to any CNN** (e.g. MobileNet attained 1.4% boost in accuracy, ResNet a 0.7%)
- The effectiveness of model scaling **depends heavily on the baseline network.**
- Researchers developed a new baseline network by performing a **neural architecture search** using AutoML's MNAS framework which optimises both accuracy and efficiency.
- Their new architecture using mobile **inverted bottleneck convolution** similar to MobileNetV2, but is slightly larger due to the increased FLOP budgets

# EfficientNet Architecture



# EfficientNet Performance



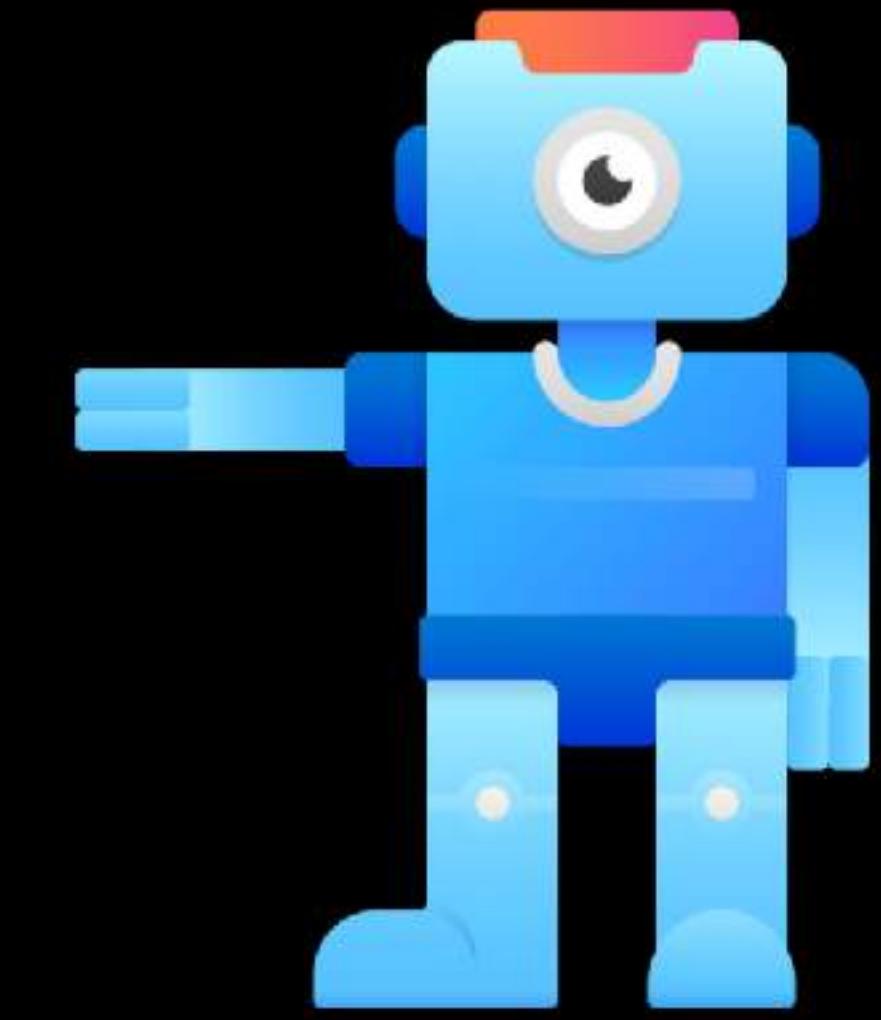


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Let's take a look at DenseNets**



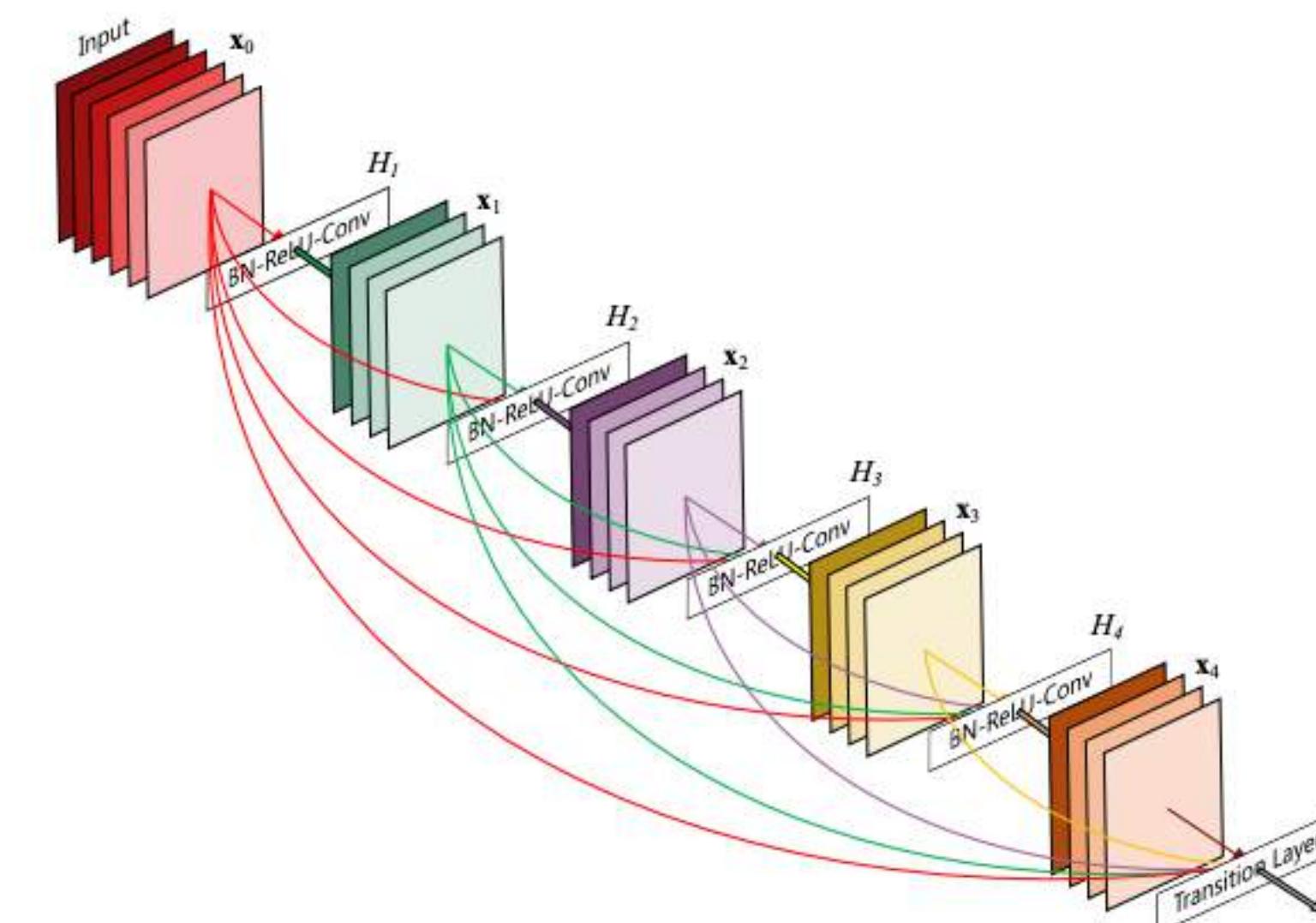
# MODERN COMPUTER VISION

BY RAJEEV RATAN

**DenseNets**  
**ResNets but better!**

# Densely Connected Convolutional Neural Networks

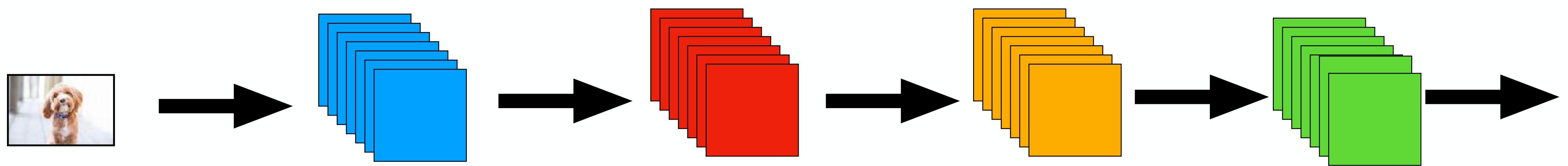
- Introduced in 2016 jointly by Cornwell University, Tsinghua University and Facebook AI Research, DenseNet won Best Paper Award at 2017 CVPR conference.
- It was able to attain higher accuracy than ResNet with fewer parameters.



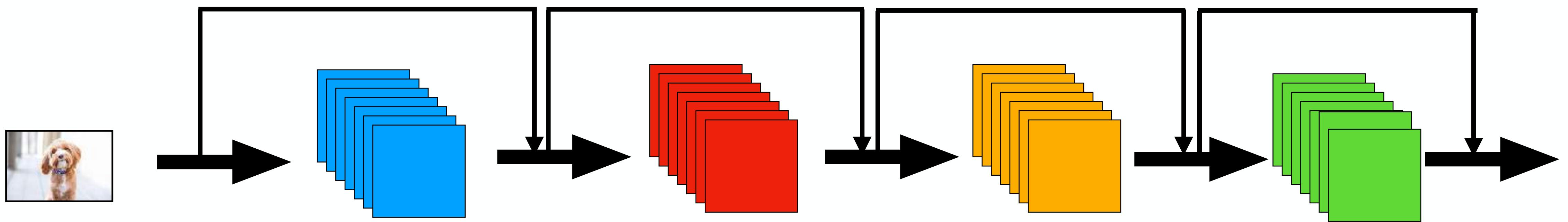
# Motivation

- Training Deep CNNs is problematic due to **vanishing gradients**
- This is because the path for deep networks become so long gradients go to zero before completing the path (vanish)
- DenseNets solve this by using an ingenious concept of ‘collective knowledge’ where each layer receives info from all previous layers

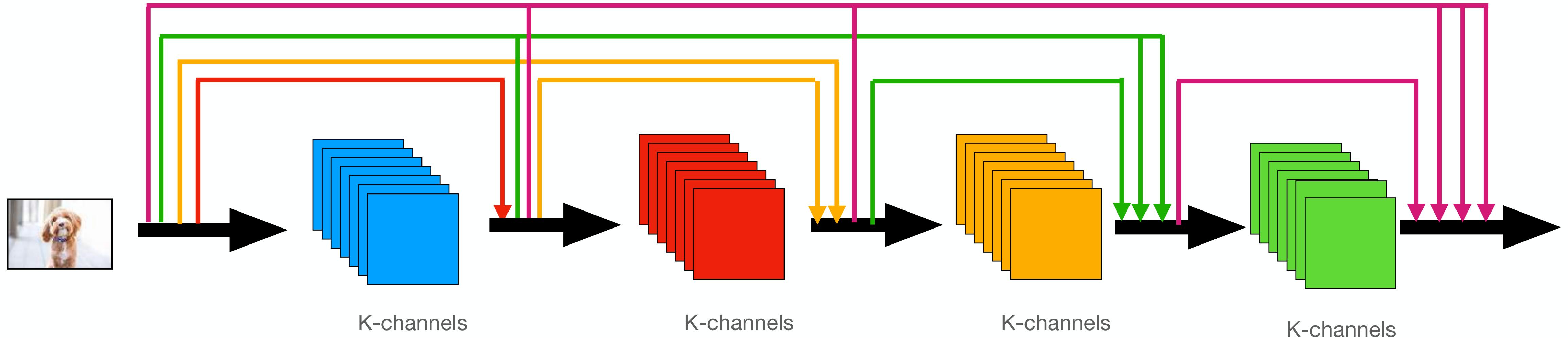
# Classical Convolution Architecture



# ResNet Convolution Architecture



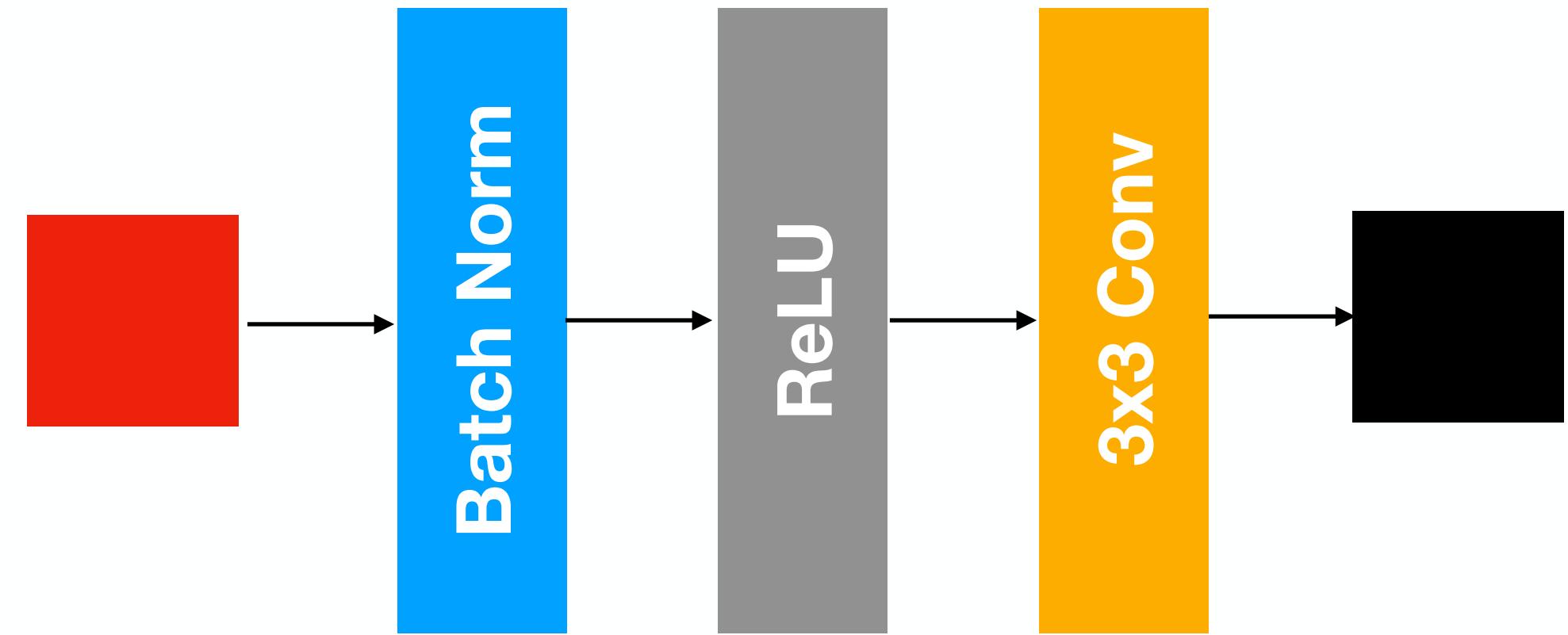
# DenseNet Architecture



- Each layer receives info from all previous layers
- Growth Rate  $k$  is the additional number of channels for each layer

# DenseNet Component

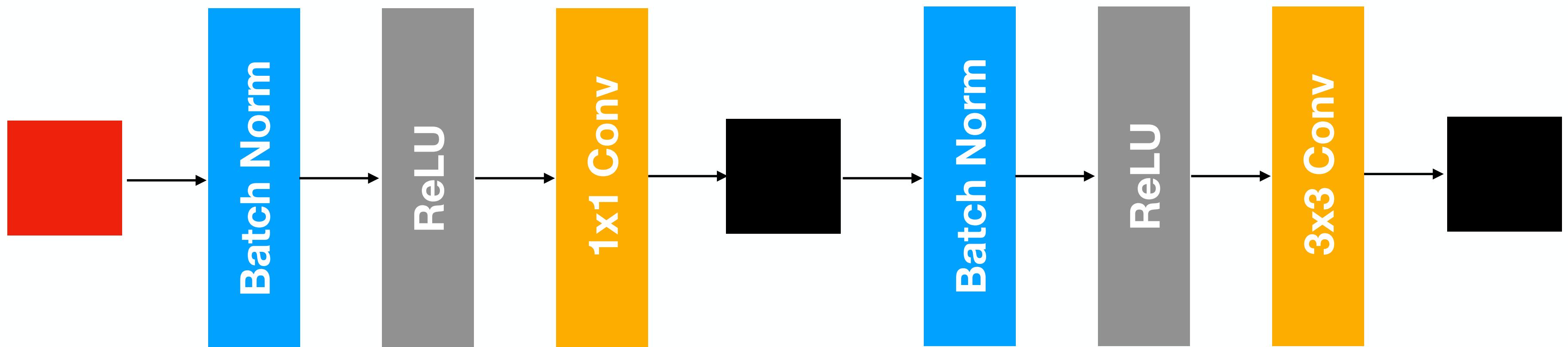
## Basic DenseNet Composition Layer



- Basic DenseNet Composition Layer contains Pre-Activation Batch Norm, ReLU and then a 3x3 Conv Layer

# DenseNet Component

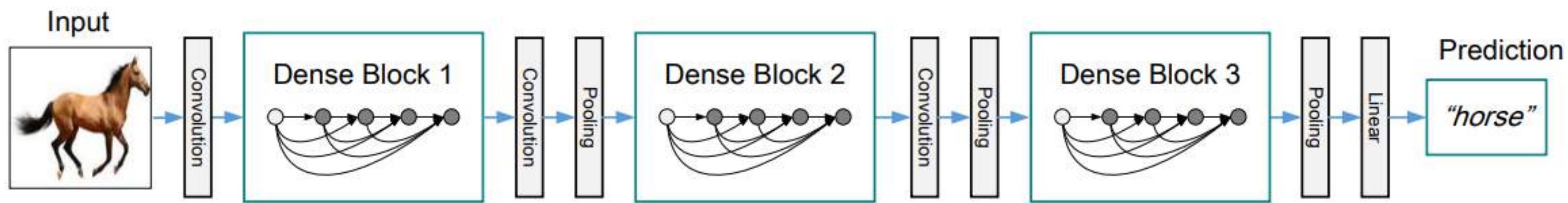
## DenseNet-B (Bottleneck Layers)



- BN-ReLU **1x1 Conv** is done before BN-ReLU **3x3 Layer**

# DenseNet Component

## Multiple Dense Blocks with Transition Layers



**Figure 2:** A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

- 1x1 Conv is followed by 2x2 Average Pooling are used as ‘Transition Layers’ between two contiguous dense blocks

# DenseNet Component

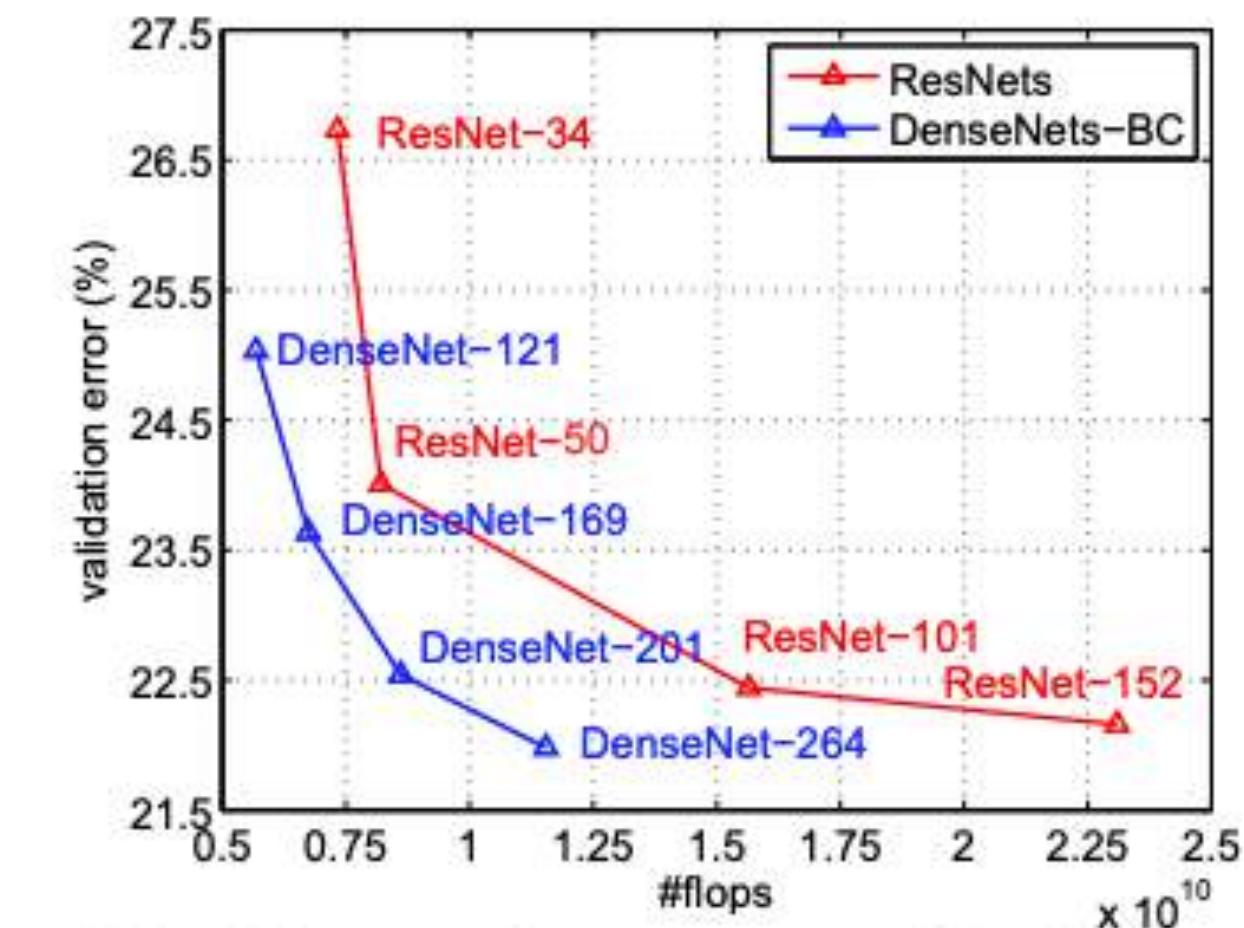
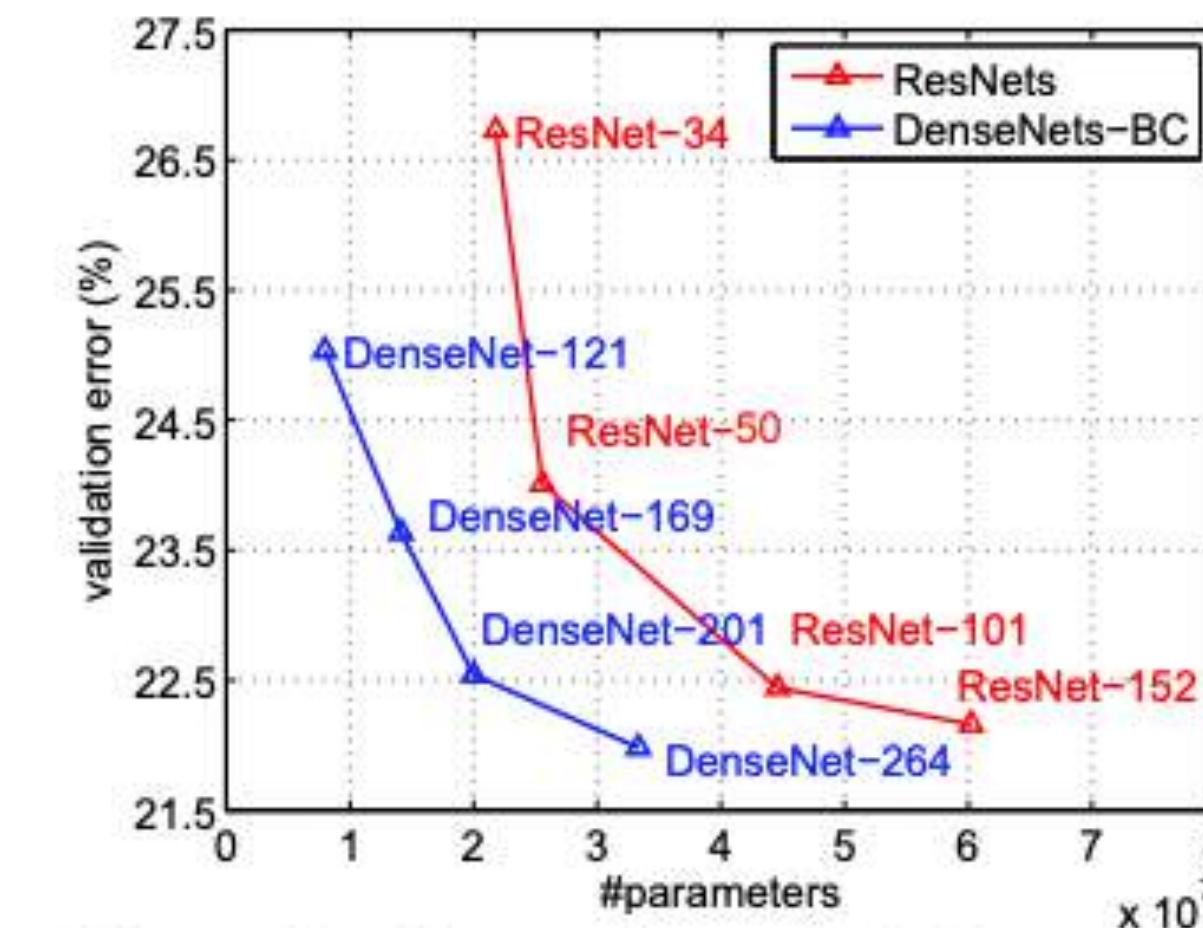
Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	$112 \times 112$		$7 \times 7$ conv, stride 2		
Pooling	$56 \times 56$		$3 \times 3$ max pool, stride 2		
Dense Block (1)	$56 \times 56$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	$56 \times 56$		$1 \times 1$ conv		
	$28 \times 28$		$2 \times 2$ average pool, stride 2		
Dense Block (2)	$28 \times 28$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	$28 \times 28$		$1 \times 1$ conv		
	$14 \times 14$		$2 \times 2$ average pool, stride 2		
Dense Block (3)	$14 \times 14$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	$14 \times 14$		$1 \times 1$ conv		
	$7 \times 7$		$2 \times 2$ average pool, stride 2		
Dense Block (4)	$7 \times 7$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	$1 \times 1$		$7 \times 7$ global average pool		
			1000D fully-connected, softmax		

**Table 1:** DenseNet architectures for ImageNet. The growth rate for all the networks is  $k = 32$ . Note that each “conv” layer shown in the table corresponds the sequence BN-ReLU-Conv.

# DenseNet Performance

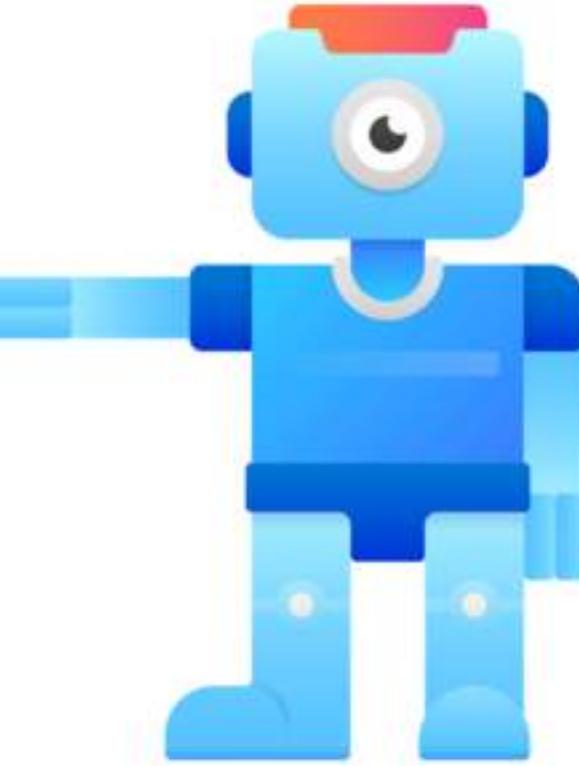
Model	top-1	top-5
DenseNet-121	25.02 / 23.61	7.71 / 6.66
DenseNet-169	23.80 / 22.08	6.85 / 5.92
DenseNet-201	22.58 / 21.46	6.34 / 5.54
DenseNet-264	22.15 / 20.80	6.12 / 5.29

**Table 3:** The top-1 and top-5 error rates on the ImageNet validation set, with single-crop / 10-crop testing.



**Figure 3:** Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

- DenseNet-B - DenseNets with a bottleneck layer
- DenseNet-BC - Bottleneck + Compression (C) Factor  $\theta$  (Controls the feature map reduction)



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**ImageNet**

# Examine Models

## PyTorch

2. [AlexNet](#)
3. [VGG](#)
4. [ResNet](#)
5. [SqueezeNet](#)
6. [DenseNet](#)
7. [Inception v3](#)
8. [GoogLeNet](#)
9. [ShuffleNet v2](#)
10. [MobileNet v2](#)
11. [ResNeXt](#)
12. [Wide ResNet](#)
13. [MNASNet](#)

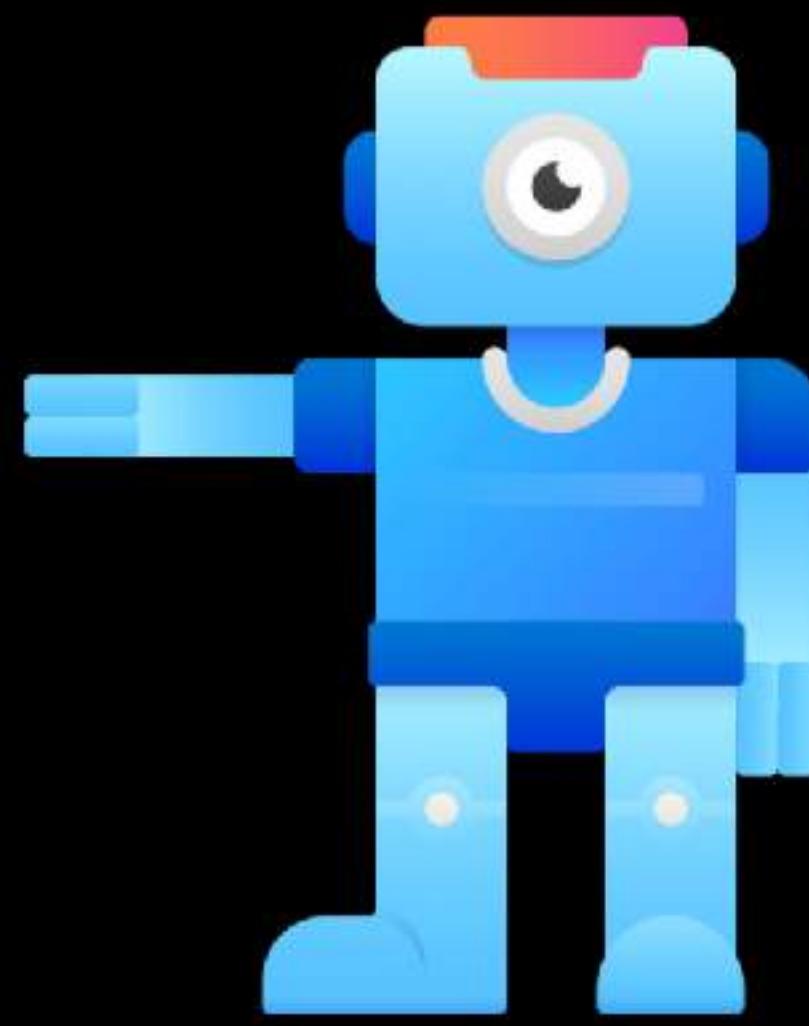
## Keras

1. [Xception](#)
2. [VGG16 19](#)
3. [ResNet50](#)
4. [ResNet152V2](#)
5. [InceptionV3](#)
6. [MobileNetV2](#)
7. [DenseNet](#)
8. [NASNetMobile](#)
9. [NASNetLarge](#)
10. [EfficientNet](#)

## 1.Pretrained Models

- 2.ImageNet
- 3.HDF5
- 4.Rank-1 and Rank-5
- 5.Feature Extractors
- 6.Transfer Learning Theory
- 7.Transfer Learning Keras
- 8.Transfer Learning PyTorch

**ImageNet**  
Why ImageNet is so important to the Computer Vision World



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# ImageNet

- Currently the world's largest dataset of labeled images

<https://www.image-net.org/>



ImageNet is an image database organized according to the [WordNet](#) hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. The project has been [instrumental](#) in advancing computer vision and deep learning research. The data is available for free to researchers for non-commercial use.

Mar 11 2021. ImageNet website update.

# ImageNet - ILSVR

- The most highly-used subset of ImageNet is the [ImageNet Large Scale Visual Recognition Challenge \(ILSVRC\)](#) 2012-2017 image classification and localisation dataset.
- The dataset spans 1000 object classes and contains 1,281,167 training images, 50,000 validation images and 100,000 test images. This subset is available on [Kaggle](#).
- It is the most common benchmark used when evaluating new exotic CNN Models.

# ImageNet in Research

## IM<sup>A</sup>GENET on Google Scholar

**4,386**  
Citations

**2,847**  
Citations

Imagenet: A large-scale hierarchical image database  
J Deng, W Dong, R Socher, LJ Li, K Li... - Computer Vision and ..., 2009 - ieeexplore.ieee.org  
Abstract: The explosion of image data on the Internet has the potential to foster more sophisticated and robust models and algorithms to index, retrieve, organize and interact with images and multimedia data. But exactly how such data can be harnessed and organized  
Cited by 4386 Related articles All 30 versions Cite Save

Imagenet large scale visual recognition challenge  
O Russakovsky, J Deng, H Su, J Krause... - International Journal of ..., 2015 - Springer  
Abstract The ImageNet Large Scale Visual Recognition Challenge is a benchmark in object category classification and detection on hundreds of object categories and millions of images. The challenge has been run annually from 2010 to present, attracting participation  
Cited by 2847 Related articles All 17 versions Cite Save

**...and many more.**

[https://www.image-net.org/static\\_files/files/imagenet\\_ilsvrc2017\\_v1.0.pdf](https://www.image-net.org/static_files/files/imagenet_ilsvrc2017_v1.0.pdf)

# ImageNet in Research

## IMAGENET on Google Scholar

**4,386**  
Citations

**2,847**  
Citations

Imagenet: A large-scale hierarchical image database  
J Deng, W Dong, R Socher, LJ Li, K Li... - Computer Vision and ..., 2009 - ieeexplore.ieee.org  
Abstract: The explosion of image data on the Internet has the potential to foster more sophisticated and robust models and algorithms to index, retrieve, organize and interact with images and multimedia data. But exactly how such data can be harnessed and organized  
Cited by 4386 Related articles All 30 versions Cite Save

Imagenet large scale visual recognition challenge  
O Russakovsky, J Deng, H Su, J Krause... - International Journal of ..., 2015 - Springer  
Abstract The ImageNet Large Scale Visual Recognition Challenge is a benchmark in object category classification and detection on hundreds of object categories and millions of images. The challenge has been run annually from 2010 to present, attracting participation  
Cited by 2847 Related articles All 17 versions Cite Save

**...and many more.**

[https://www.image-net.org/static\\_files/files/imagenet\\_ilsvrc2017\\_v1.0.pdf](https://www.image-net.org/static_files/files/imagenet_ilsvrc2017_v1.0.pdf)

# ImageNet

## Hardly the First Image Dataset



**Segmentation (2001)**  
D. Martin, C. Fowlkes, D. Tal, J. Malik.



**CMU/VASC Faces (1998)**  
H. Rowley, S. Baluja, T. Kanade



**FERET Faces (1998)**  
P. Phillips, H. Wechsler, J. Huang, P. Rauss



**COIL Objects (1996)**  
S. Nene, S. Nayar, H. Murase



**MNIST digits (1998-10)**  
Y. LeCun & C. Cortes



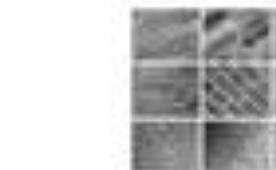
**KTH human action (2004)**  
I. Laptev & B. Caputo



**Sign Language (2008)**  
P. Buehler, M. Everingham, A. Zisserman



**UIUC Cars (2004)**  
S. Agarwal, A. Awan, D. Roth



**3D Textures (2005)**  
S. Lazebnik, C. Schmid, J. Ponce



**CuRRET Textures (1999)**  
K. Dana B. Van Ginneken S. Nayar J. Koenderink



**CAVIAR Tracking (2005)**  
R. Fisher, J. Santos-Victor J. Crowley



**Middlebury Stereo (2002)**  
D. Scharstein R. Szeliski



**CalTech 101/256 (2005)**  
Fei-Fei et al, 2004  
Griffin et al, 2007



**LabelMe (2005)**  
Russell et al, 2005



**ESP (2006)**  
Ahn et al, 2006



**MSRC (2006)**  
Shotton et al. 2006



**PASCAL (2007)**  
Everingham et al, 2009

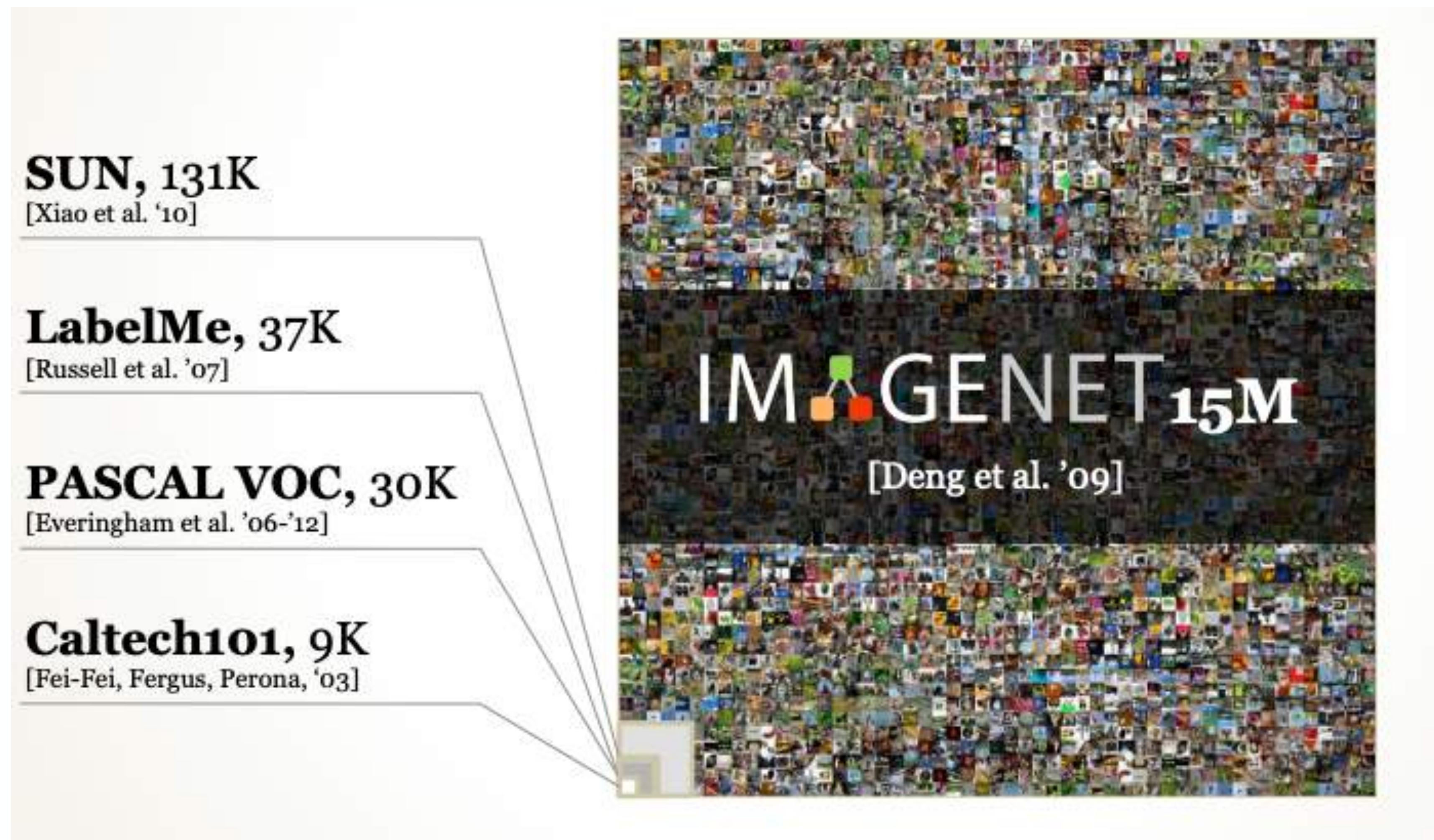


**Lotus Hill (2007)**  
Yao et al, 2007



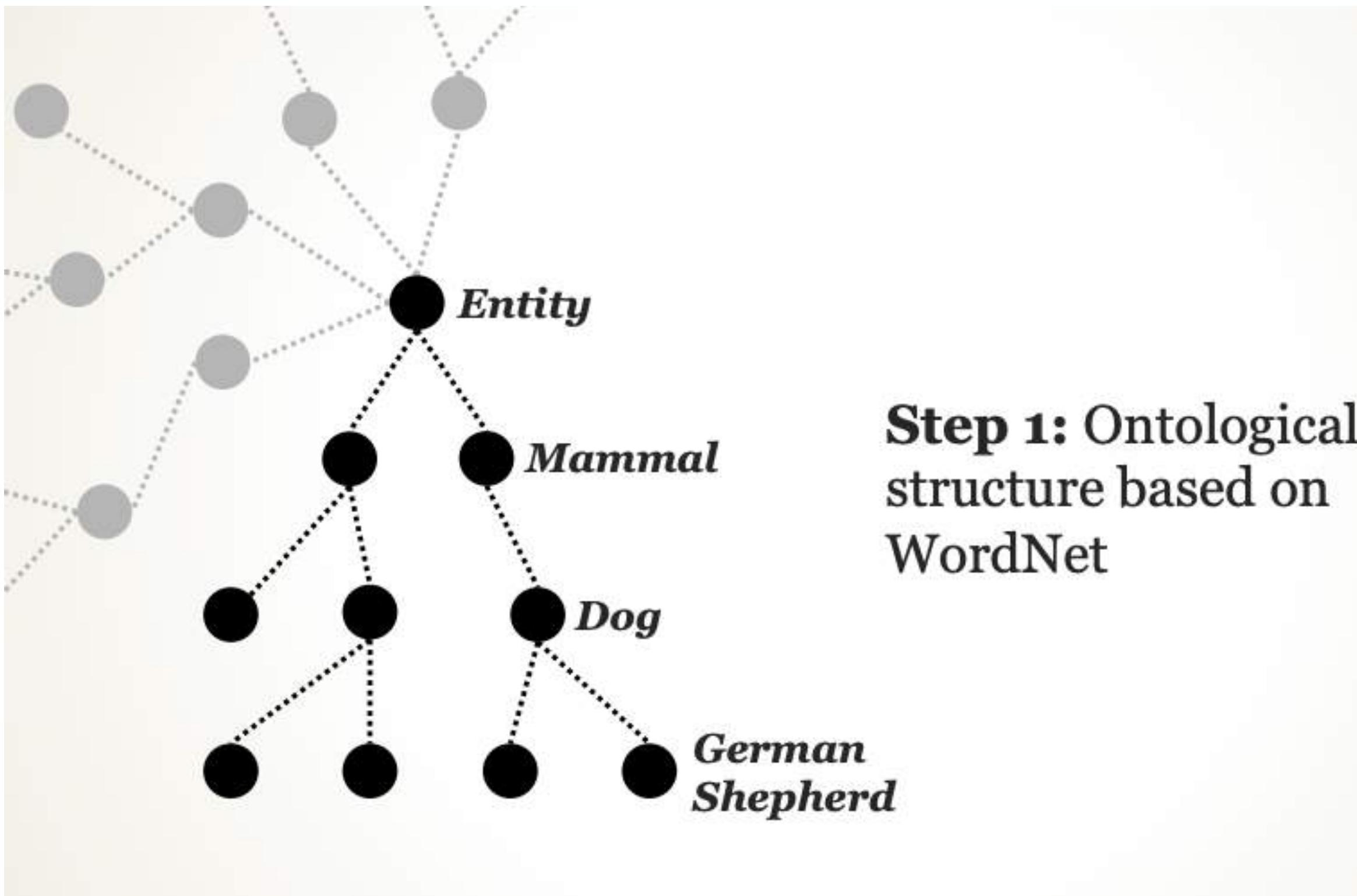
**TinyImage (2008)**  
Torralba et al. 2008

# ImageNet - What makes it so good? It's Size



[https://www.image-net.org/static\\_files/files/imagenet\\_ilsvrc2017\\_v1.0.pdf](https://www.image-net.org/static_files/files/imagenet_ilsvrc2017_v1.0.pdf)

# ImageNet - WordNet?



**Step 1:** Ontological structure based on WordNet

**IMAGENET** SEARCH

14,197,122 images, 21841 synsets indexed

Not logged in. [Login](#) | [Signup](#)

**Bird**  
Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings

2126 pictures 92.85% Popularity Percentile Wordnet IDs

**Treemap Visualization** **Images of the Synset** **Downloads**

ImageNet 2011 Fall Release (32326)

- + plant, flora, plant life (4486)
- geological formation, formation (17)
- natural object (1112)
- sport, athletics (176)
- artifact, artefact (10504)
- fungus (308)
- person, individual, someone, some animal, animate being, beast, brute
- invertebrate (766)
- homeotherm, homiotherm, hor
- work animal (4)
- darter (0)
- survivor (0)
- range animal (0)
- creepy-crawly (0)
- domestic animal, domesticated
- molter, moulter (0)
- varmint, varment (0)
- mutant (0)
- critter (0)
- game (47)
- young, offspring (45)
- poikilotherm, ectotherm (0)
- herbivore (0)
- peeper (0)
- pest (1)
- female (4)
- insectivore (0)
- pet (0)

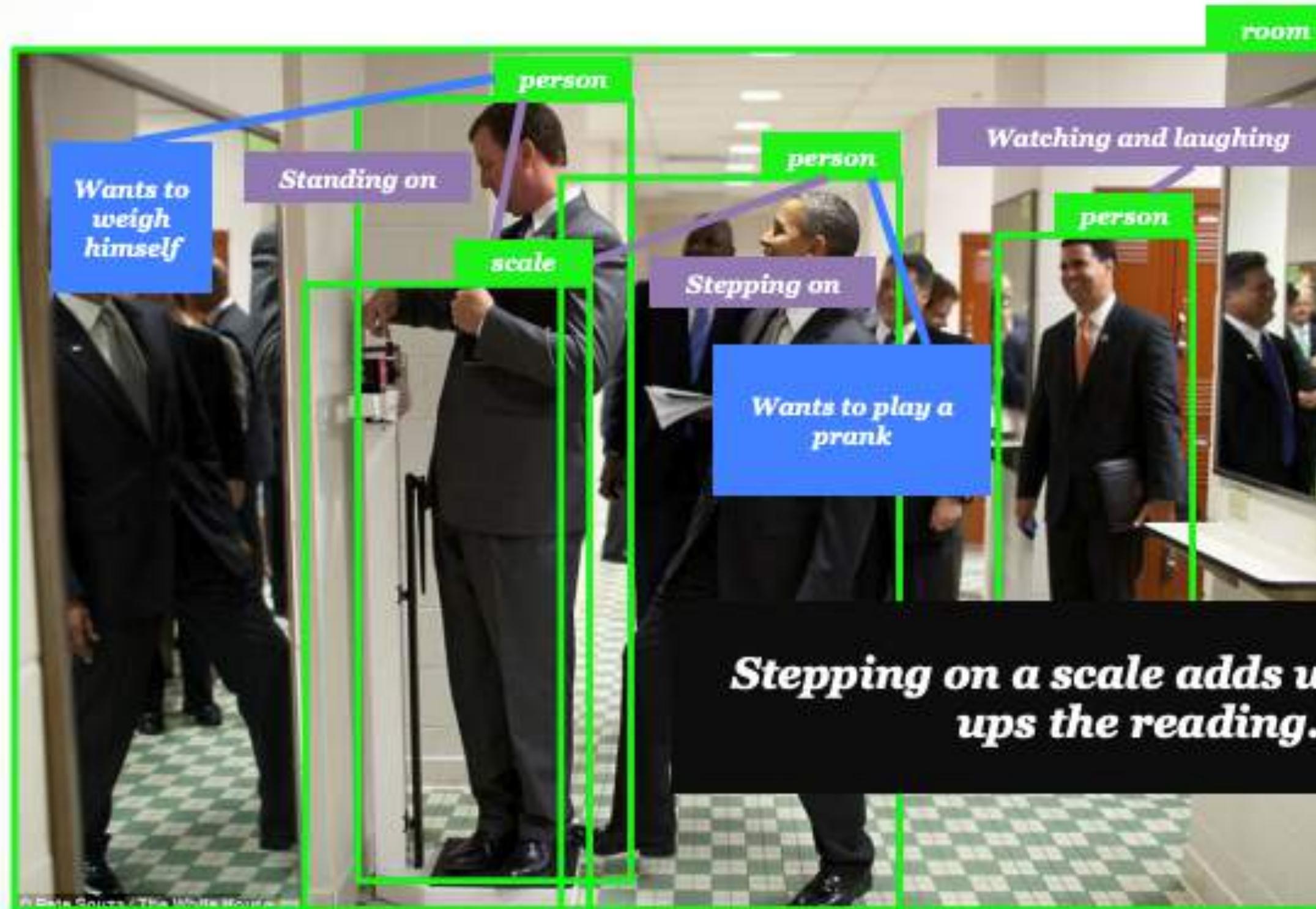
Aquatic	Bird	Gallinaceous
Archaeornis	Nonpasserine	Night
Bird	Trogon	Carinate
Twitterer	Caprimulgidae	
Passerine		
Dickeybird		
Apodiform		
Archaeopteryx		
Hen		
Cuculiform		
Nester		
Cock		
Ratite		
Parrot		

© 2010 Stanford Vision Lab, Stanford University, Princeton University support@image-net.org Copyright infringement

[https://www.image-net.org/static\\_files/files/imagenet\\_ilsvrc2017\\_v1.0.pdf](https://www.image-net.org/static_files/files/imagenet_ilsvrc2017_v1.0.pdf)

# ImageNet - WordNet - Aims to Give Rise to...

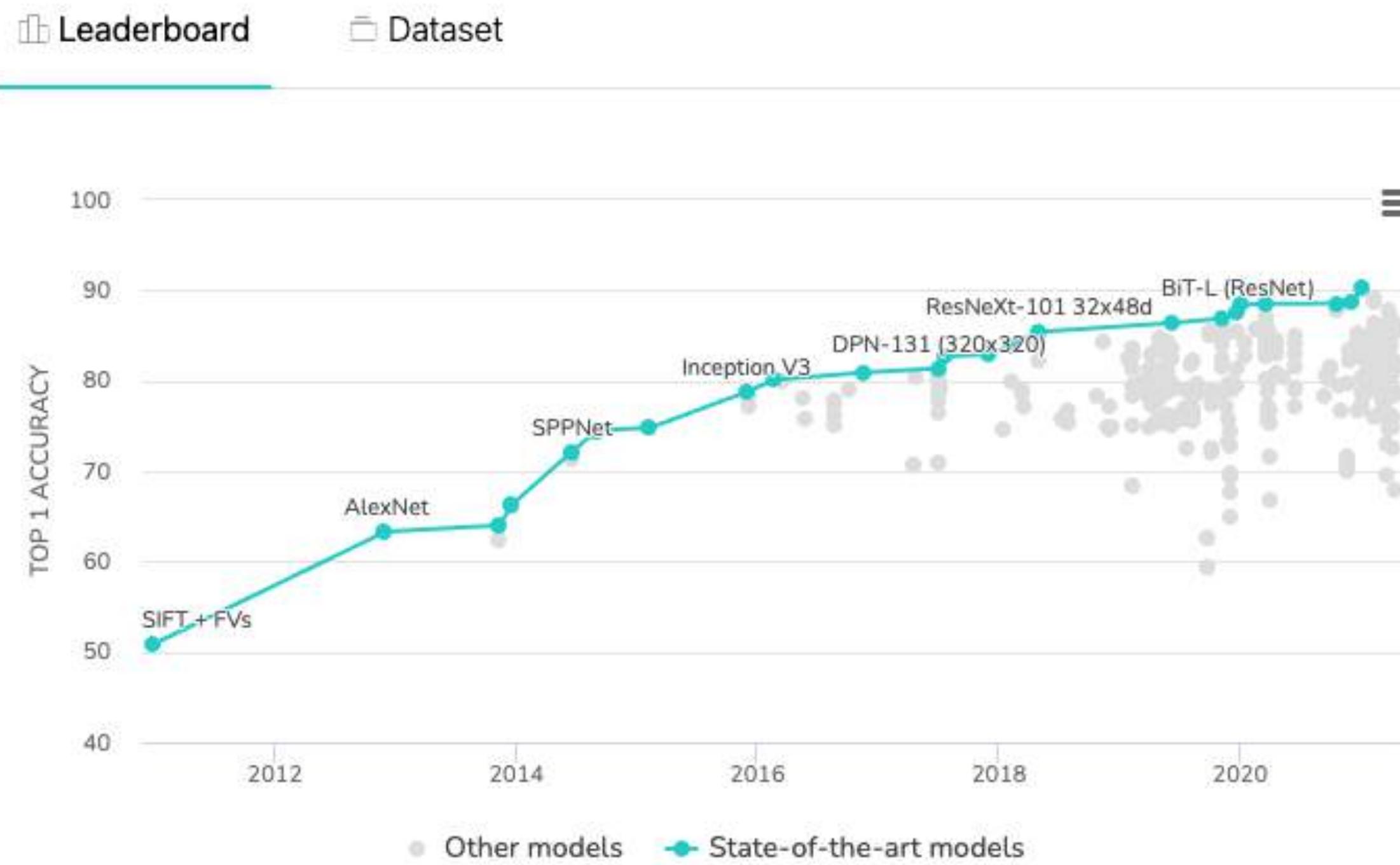
...to human-level understanding.



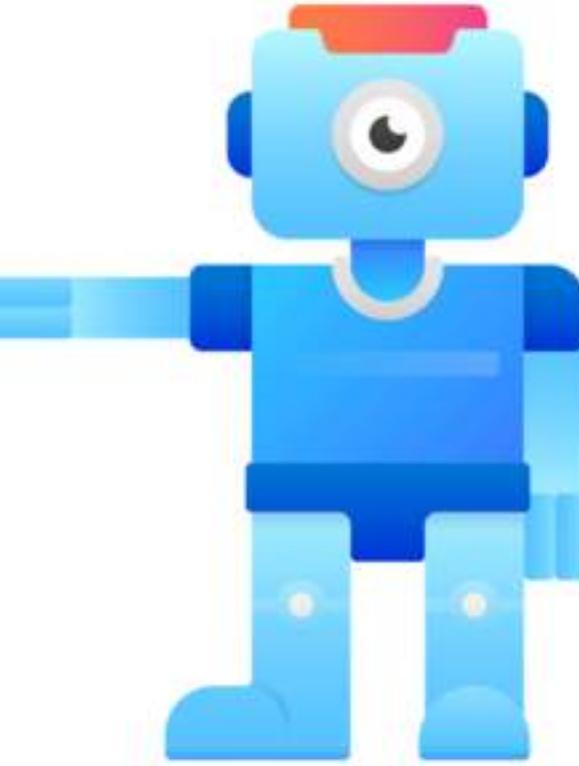
[https://www.image-net.org/static\\_files/files/imagenet\\_ilsvrc2017\\_v1.0.pdf](https://www.image-net.org/static_files/files/imagenet_ilsvrc2017_v1.0.pdf)

# ImageNet - Current State of Art Performance

## Image Classification on ImageNet



RANK	MODEL	TOP 1 ACCURACY	TOP 5 ACCURACY	NUMBER OF PARAMS	EXTRA TRAINING DATA	PAPER
1	Meta Pseudo Labels (EfficientNet-L2)	90.2%	98.8%	480M	✓	Meta Pseudo Labels
2	Meta Pseudo Labels (EfficientNet-B6-Wide)	90%	98.7%	390M	✓	Meta Pseudo Labels
3	NFNet-F4+	89.2%		527M	✓	High-Performance Large-Scale Image Recognition Without Normalization
4	ALIGN (EfficientNet-L2)	88.64%	98.67%	480M	✓	Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision
5	EfficientNet-L2-475 (SAM)	88.61%		480M	✓	Sharpness-Aware Minimization for Efficiently Improving Generalization

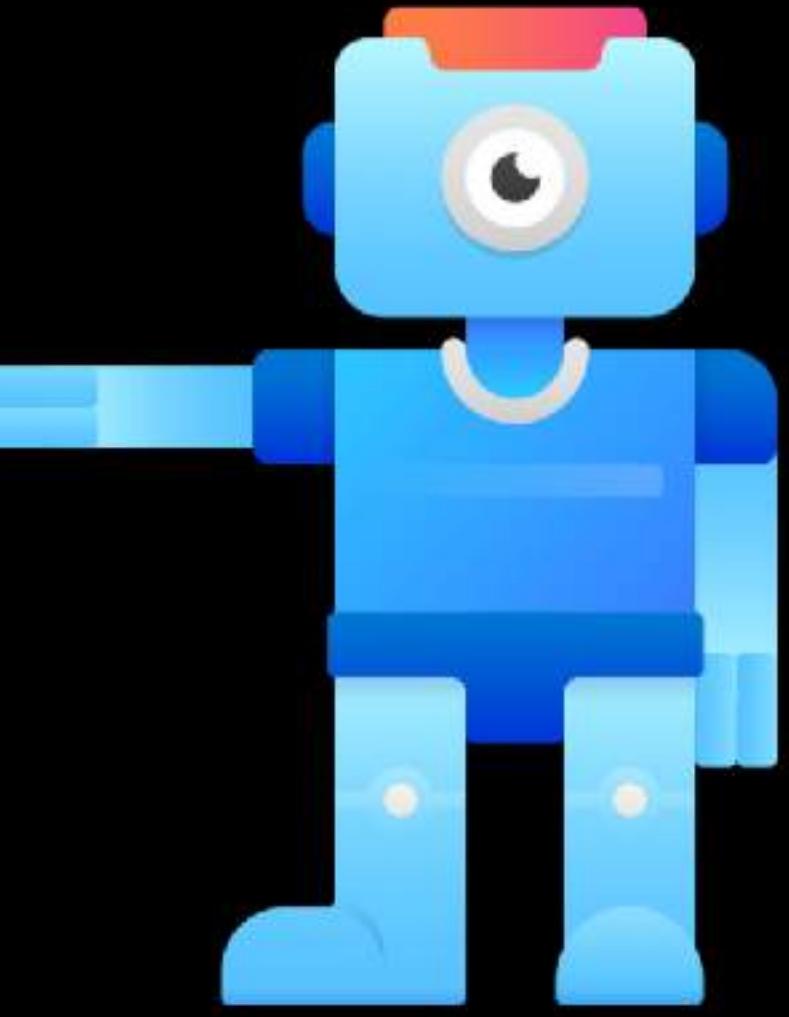


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Top-1 and Top-5 Accuracy**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Top-1 and Top-5 Accuracy (aka Rank-1, Rank-5)

How we benchmark large multi-class classification models

# Rank-N

- Rank-N Accuracy is a way to measure a classifier's accuracy with a bit more leeway.
- Imagine our classifier returns this output:



Class Name	Probability
<b>Shetland sheepdog</b>	<b>0.44</b>
<b>collie</b>	<b>0.31</b>
<b>chow</b>	<b>0.1</b>
<b>wire-haired fox terrie</b>	<b>0.09</b>
<b>lion</b>	<b>0.06</b>

- It'll return '**Shetland sheepdog**' as the predicted class (due to highest probability). However, that would be incorrect.
- The correct class '**collie**' is actually the second most probable, which means the classifier is still doing quite well, but is not reflected if we look only at the top predicted class.

# Rank-N or Top-N Accuracy

- Rank-N Accuracy considers the top N classes with the highest probabilities.

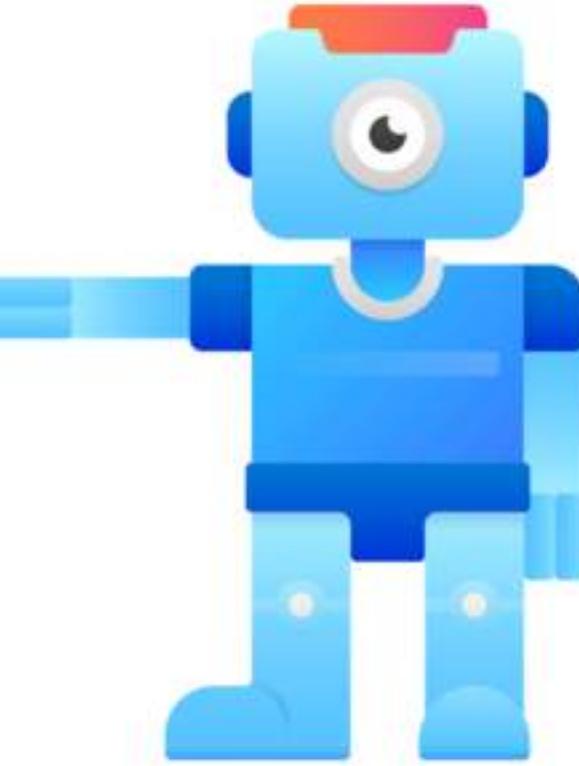


Class Name	Probability
Shetland sheepdog	0.44
collie	0.31
chow	0.1
wire-haired fox terrie	0.09
lion	0.06

- For example, Rank-5, will consider any of the top 5 most likely classes for the predicted label.

# Assessing Classifier Performance

RANK	MODEL	TOP 1 ACCURACY ↑	TOP 5 ACCURACY	NUMBER OF PARAMS	EXTRA TRAINING DATA	PAPER
1	Meta Pseudo Labels (EfficientNet-L2)	90.2%	98.8%	480M	✓	Meta Pseudo Labels
2	Meta Pseudo Labels (EfficientNet-B6-Wide)	90%	98.7%	390M	✓	Meta Pseudo Labels
3	NFNet-F4+	89.2%		527M	✓	High-Performance Large-Scale Image Recognition Without Normalization
4	ALIGN (EfficientNet-L2)	88.64%	98.67%	480M	✓	Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision
5	EfficientNet-L2-475 (SAM)	88.61%		480M	✓	Sharpness-Aware Minimization for Efficiently Improving Generalization

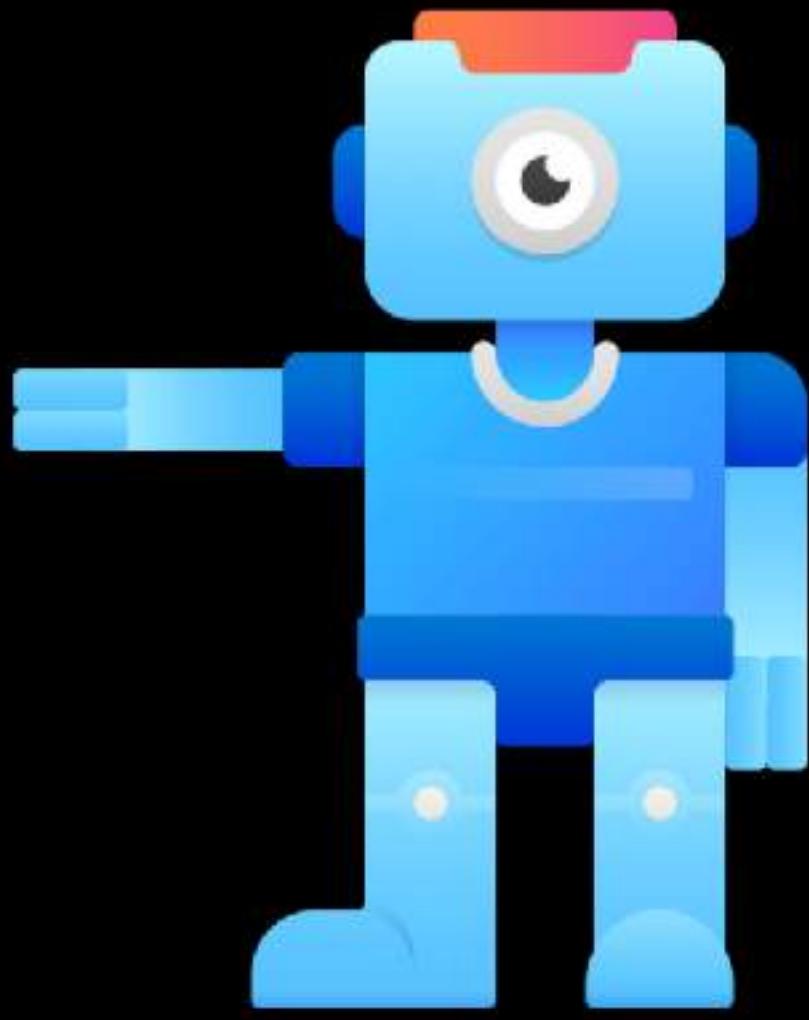


MODERN  
COMPUTER  
VISION

BY RAJEEV RATAN

# Next...

**Callbacks - Early Stopping, Check Pointing, LR Schedule and more**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Callbacks - Early Stopping, Check Pointing, LR Schedule and more

Using CallBack methods in Keras and PyTorch

# Callbacks

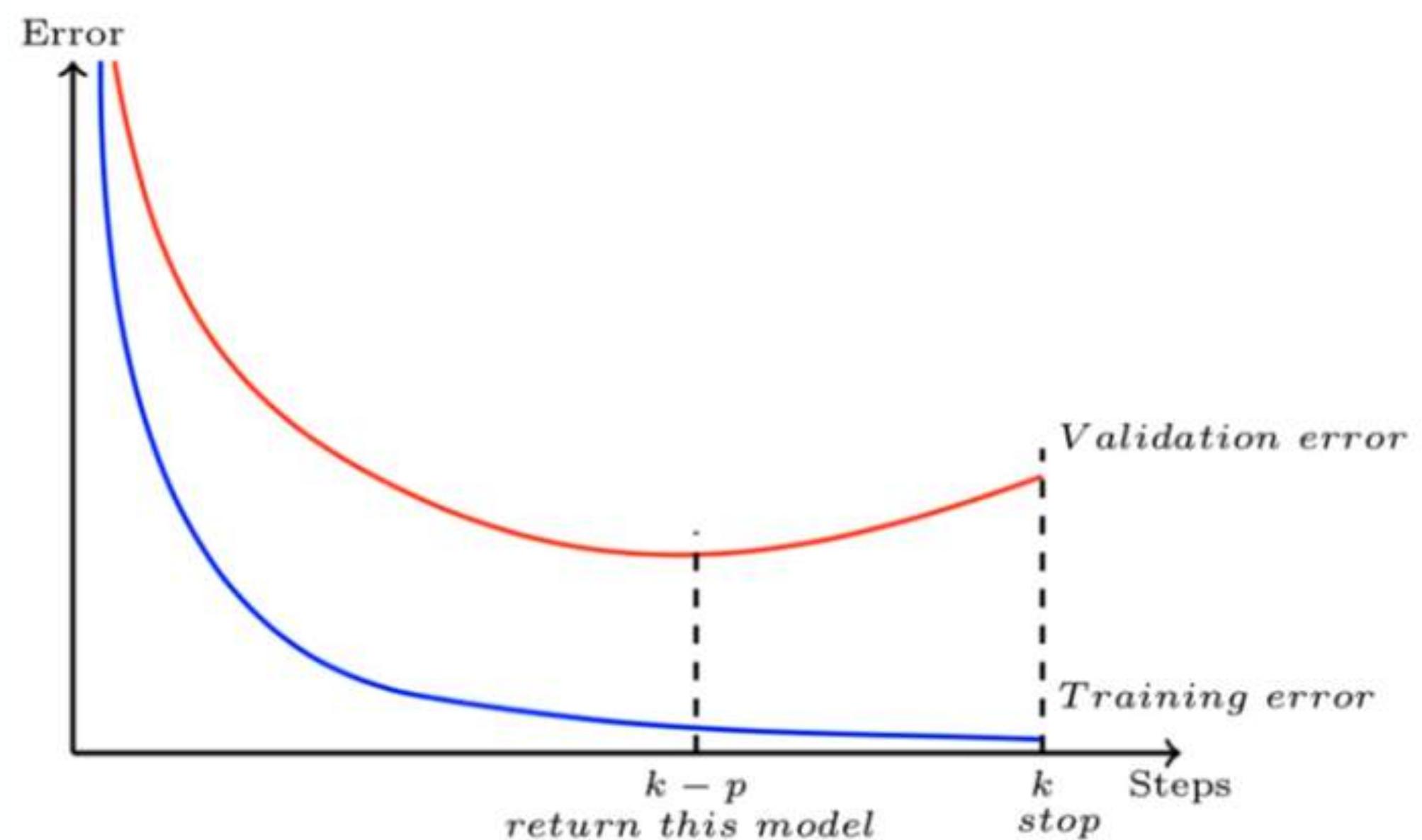
- **Callbacks** are used to perform different actions at different stages during training.
- **Why is this useful?**
  - What if we wanted to save each model after each epoch?
  - What if we set our model to train for 100 epochs but after 30, it started to overfit? Wouldn't we want some way stop training?
  - What if we wanted to log our information somewhere so that it can analysed later on?

# Callbacks are the solution

- Callbacks can be used to perform:
  - Early Stopping
  - Model Checkpointing
  - Learning Rate Scheduler
  - Logging
  - Remote Monitoring
  - Custom Functions

# Early Stopping - A Solution for Overfitting

- During our training process our validation loss may **stagnate** or **stop decreasing** and sometimes actually start to increase (overfitting)
- We can use Callbacks to implement **Early Stopping**



# Model Checkpointing

- During Training we can periodically save the weights after each epoch.
- This allows us to resume training in the event of a crash
- The checkpoint file contains the model's weights or the model itself
- This can usually be configured in many ways

```
tf.keras.callbacks.ModelCheckpoint(  
    filepath,  
    monitor="val_loss",  
    verbose=0,  
    save_best_only=False,  
    save_weights_only=False,  
    mode="auto",  
    save_freq="epoch",  
    options=None,  
    **kwargs  
)
```

# Learning Rate Scheduler

- We can avoid having our loss oscillate around the global minimum by attempting to reduce the Learn Rate by a specified amount.
- If no improvement is seen in our monitored metric (val\_loss typically), we wait a certain number of epochs (patience) then this callback reduces the learning rate by a factor.

```
from keras.callbacks import ReduceLROnPlateau

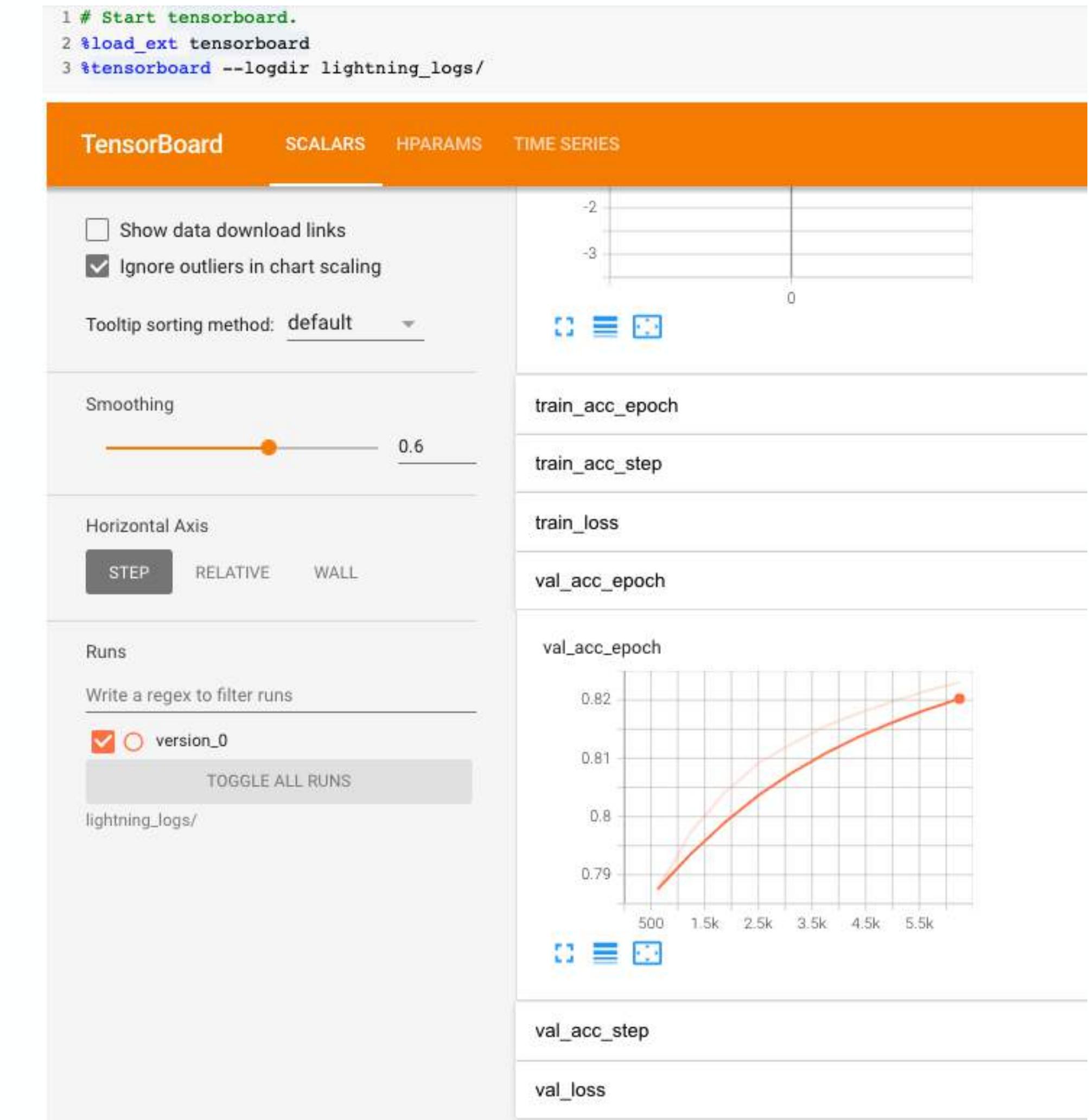
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.2, patience = 3, verbose = 1, min_delta = 0.0001)
```

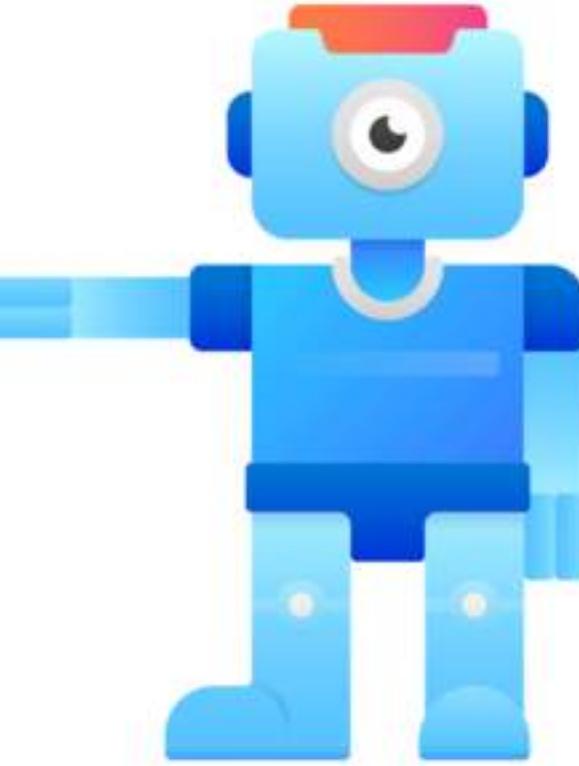
# Logging

- We can automatically log our model's training stats (training and validation loss/accuracy) and view them later using TensorBoard or others.

```
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2),
    tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-{val_loss:.2f}.h5'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs'),
]
model.fit(dataset, epochs=10, callbacks=my_callbacks)
```

```
tf.keras.callbacks.TensorBoard(
    log_dir="logs",
    histogram_freq=0,
    write_graph=True,
    write_images=False,
    update_freq="epoch",
    profile_batch=2,
    embeddings_freq=0,
    embeddings_metadata=None,
    **kwargs
)
```



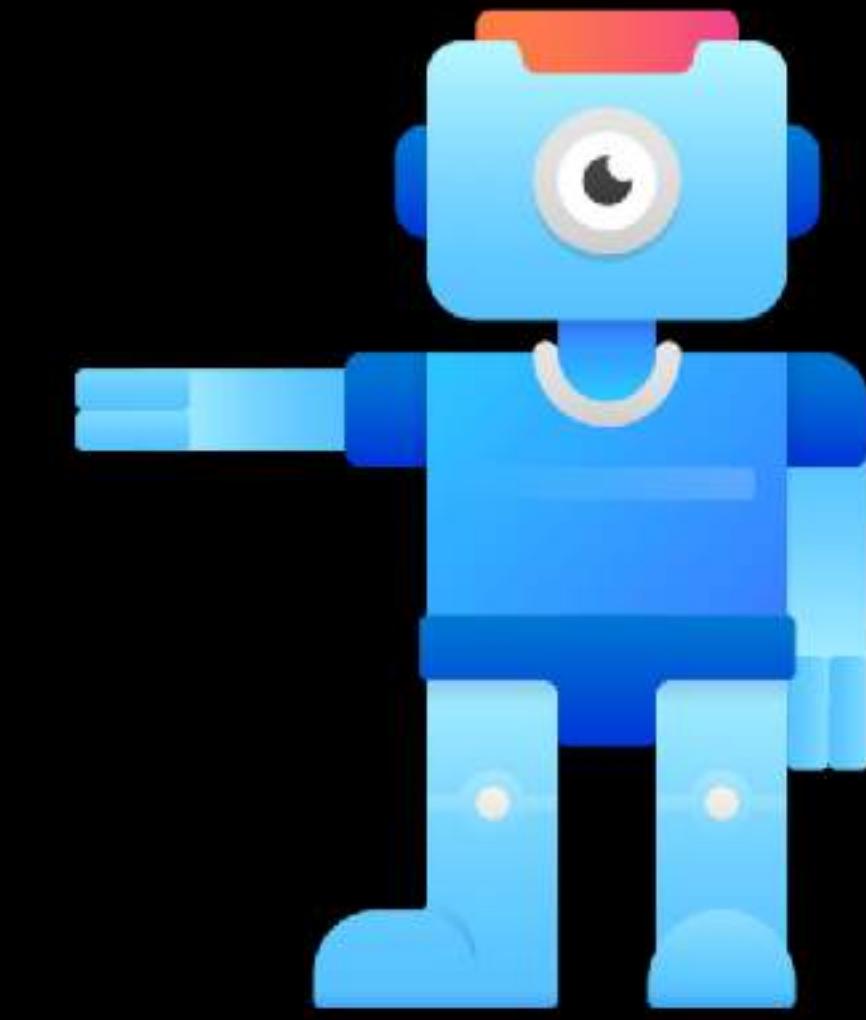


MODERN  
COMPUTER  
VISION

BY RAJEEV RATAN

# Next...

**Callbacks - Early Stopping, Check Pointing, LR Schedule and more**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## PyTorch Lightning

A feature rich easy to use PyTorch High Level Interface Library

# PyTorch Lightning



- **PyTorch Lightning** is an open-source Python library that provides a high-level interface for PyTorch.

## WHAT IS PYTORCH LIGHTNING?

Lightning makes coding complex networks simple.

Spend more time on research, less on engineering. It is fully flexible to fit any use case and built on pure PyTorch so there is no need to learn a new language. A quick refactor will allow you to:

- Run your code on any hardware
- Performance & bottleneck profiler
- Model checkpointing
- 16-bit precision
- Run distributed training
- Logging
- Metrics
- Visualization
- Early stopping
- ... and many more!

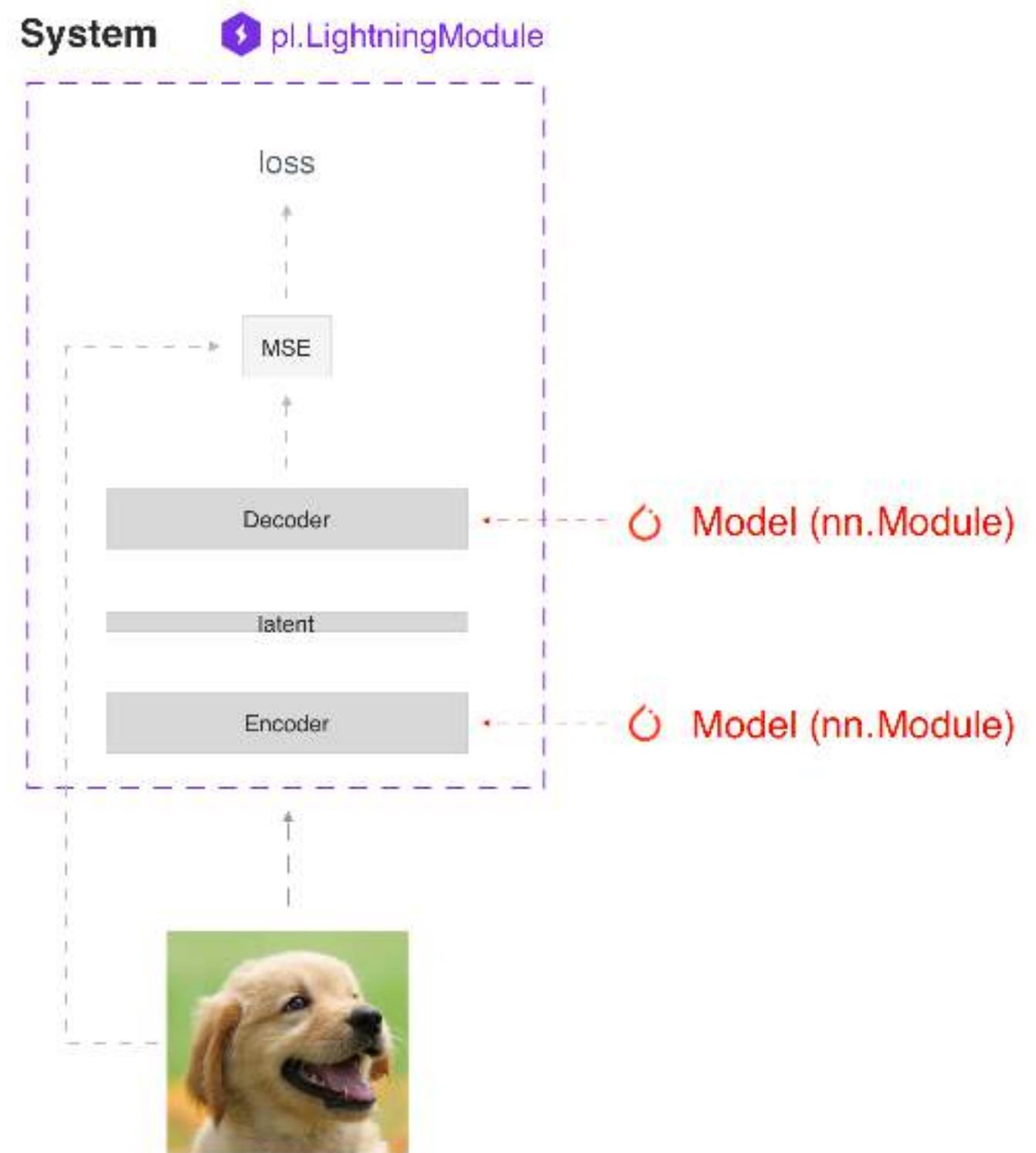
<https://www.pytorchlightning.ai/>

# Why Use Lightning?

- It offers a nice easy to use, modular approach to coding PyTorch Models.
- It was intended for researchers to focus more on science and research and spend less time worrying about how they'd deploy or scale their models during training.
- A model created using Lightning can be trained on multiple-GPUs, TPUs, with Floating Point 16 precision etc.!
- It decouples research code from engineering
- Integrates easily with logging tools such as TensorBoard, MLFlow, Neptune.ai, Comet.ai and Wandb.
- Provides auto-batch selection, learning rate selection and callbacks.

# The Lightning Philosophy

- Lightning proposes we encase all our model's building blocks into a **Lightning Module**.
- It requires basically us reorganising our existing PyTorch Code



# The Lightning Module

- The **Lightning Module** organises our PyTorch code into 5 sections:
  - Computations (init).
  - Train loop (training\_step)
  - Validation loop (validation\_step)
  - Test loop (test\_step)
  - Optimizers (configure\_optimizers)

## Minimal Example

Here are the only required methods.

```
>>> import pytorch_lightning as pl
>>> class LitModel(pl.LightningModule):
...
...
...     def __init__(self):
...         super().__init__()
...         self.l1 = nn.Linear(28 * 28, 10)
...
...
...     def forward(self, x):
...         return torch.relu(self.l1(x.view(x.size(0), -1)))
...
...
...     def training_step(self, batch, batch_idx):
...         x, y = batch
...         y_hat = self(x)
...         loss = F.cross_entropy(y_hat, y)
...         return loss
...
...
...     def configure_optimizers(self):
...         return torch.optim.Adam(self.parameters(), lr=0.02)
```

# The Lightning Module - Computations

PyTorch

```
[ ] import torch
from torch import nn

class MNISTClassifier(nn.Module):

    def __init__(self):
        super(MNISTClassifier, self).__init__()

        # mnist images are (1, 28, 28) (channels, width, height)
        self.layer_1 = torch.nn.Linear(28 * 28, 128)
        self.layer_2 = torch.nn.Linear(128, 256)
        self.layer_3 = torch.nn.Linear(256, 10)

    def forward(self, x):
        batch_size, channels, width, height = x.size()

        # (b, 1, 28, 28) -> (b, 1*28*28)
        x = x.view(batch_size, -1)

        # layer 1
        x = self.layer_1(x)
        x = torch.relu(x)

        # layer 2
        x = self.layer_2(x)
        x = torch.relu(x)

        # layer 3
        x = self.layer_3(x)

        # probability distribution over labels
        x = torch.log_softmax(x, dim=1)

    return x
```

PyTorch Lightning

```
[ ] import torch
from torch import nn
import pytorch_lightning as pl

class LightningMNISTClassifier(pl.LightningModule):

    def __init__(self):
        super(LightningMNISTClassifier, self).__init__()

        # mnist images are (1, 28, 28) (channels, width, height)
        self.layer_1 = torch.nn.Linear(28 * 28, 128)
        self.layer_2 = torch.nn.Linear(128, 256)
        self.layer_3 = torch.nn.Linear(256, 10)

    def forward(self, x):
        batch_size, channels, width, height = x.size()

        # (b, 1, 28, 28) -> (b, 1*28*28)
        x = x.view(batch_size, -1)

        # layer 1
        x = self.layer_1(x)
        x = torch.relu(x)

        # layer 2
        x = self.layer_2(x)
        x = torch.relu(x)

        # layer 3
        x = self.layer_3(x)

        # probability distribution over labels
        x = torch.log_softmax(x, dim=1)

    return x
```

# The Lightning Module - Trainers

PyTorch

```
# -----
# TRAINING LOOP
# -----
num_epochs = 1
for epoch in range(num_epochs):

    # TRAINING LOOP
    for train_batch in mnist_train:
        x, y = train_batch
        logits = pytorch_model(x)
        loss = cross_entropy_loss(logits, y)
        print('train_loss: ', loss.item())

        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

    # VALIDATION LOOP
    with torch.no_grad():
        val_loss = []
        for val_batch in mnist_val:
            x, y = val_batch
            logits = pytorch_model(x)
            val_loss = cross_entropy_loss(logits, y).item()
            val_loss.append(val_loss)

        val_loss = torch.mean(torch.tensor(val_loss))

    print('val_loss:', val_loss.item())
```

PyTorch Lightning

```
class LightningMNISTClassifier(pl.LightningModule):

    def training_step(self, train_batch, batch_idx):
        x, y = train_batch
        logits = self.forward(x)
        loss = self.cross_entropy_loss(logits, y)
        self.log('train_loss', loss)
        return loss

    def validation_step(self, val_batch, batch_idx):
        x, y = val_batch
        logits = self.forward(x)
        loss = self.cross_entropy_loss(logits, y)
        self.log('val_loss', loss)
```

(automatically reduced across epochs)

# The Lightning Module - Optimiser & Loss

PyTorch

```
pytorch_model = MNISTClassifier()
optimizer = torch.optim.Adam(pytorch_model.parameters(), lr=1e-3)
```

PyTorch Lightning

```
class LightningMNISTClassifier(pl.LightningModule):

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=1e-3)
        return optimizer
```

PyTorch

```
from torch.nn import functional as F

def cross_entropy_loss(logits, labels):
    return F.nll_loss(logits, labels)
```

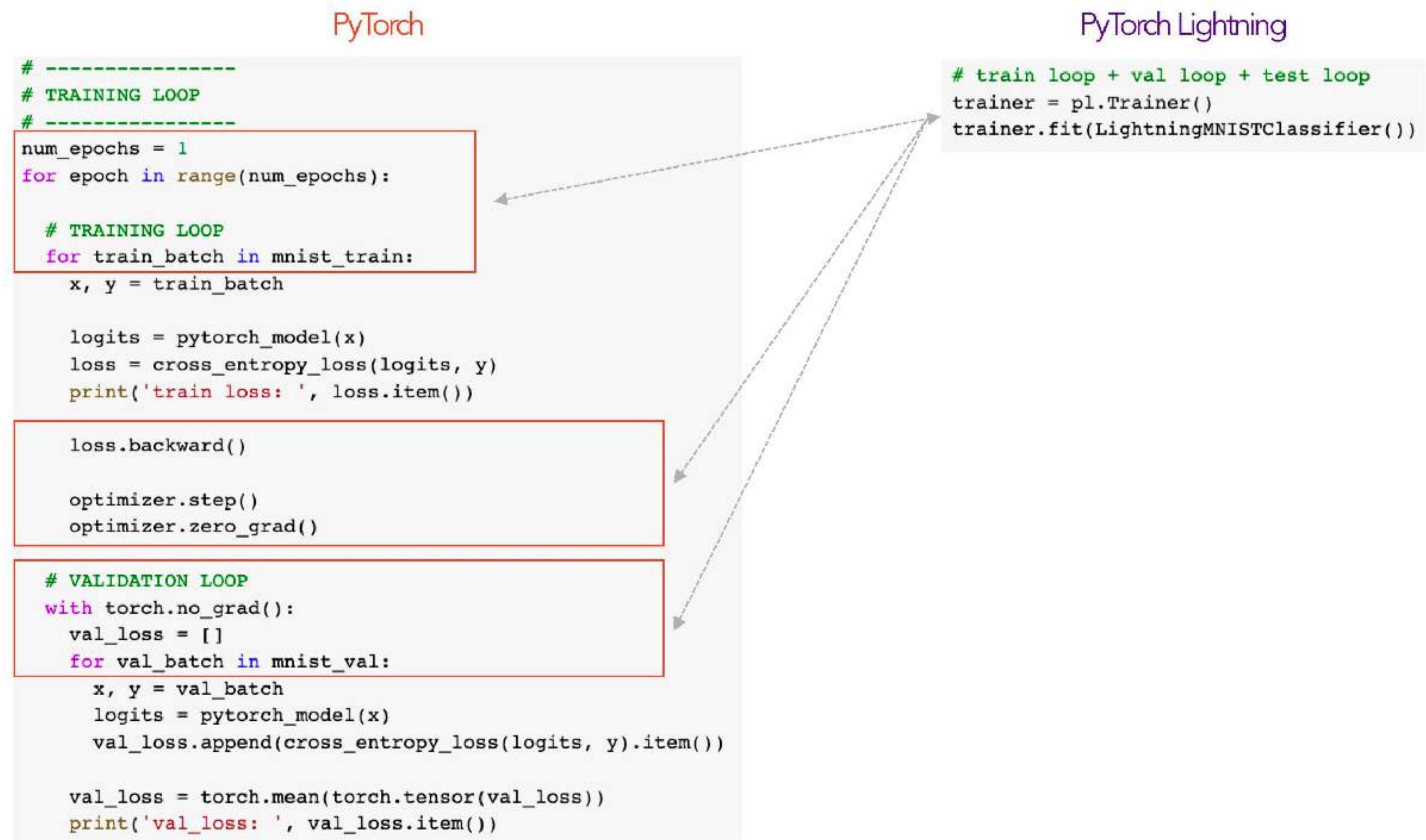
PyTorch Lightning

```
from torch.nn import functional as F

class LightningMNISTClassifier(pl.LightningModule):

    def cross_entropy_loss(self, logits, labels):
        return F.nll_loss(logits, labels)
```

# The Lightning Module - Trainer



# The Lightning Module - Data Modules

PyTorch

```
from torch.utils.data import DataLoader, random_split
from torchvision.datasets import MNIST
import os
from torchvision import datasets, transforms

# -----
# TRANSFORMS
# -----
# prepare transforms standard to MNIST
transform=transforms.Compose([transforms.ToTensor(),
                             transforms.Normalize((0.1307,), (0.3081,))])

# -----
# TRAINING, VAL DATA
# -----
mnist_train = MNIST(os.getcwd(), train=True, download=True)

# train (55,000 images), val split (5,000 images)
mnist_train, mnist_val = random_split(mnist_train, [55000, 5000])

# -----
# TEST DATA
# -----
mnist_test = MNIST(os.getcwd(), train=False, download=True)

# -----
# DATALOADERS
# -----
# The dataloaders handle shuffling, batching, etc...
mnist_train = DataLoader(mnist_train, batch_size=64)
mnist_val = DataLoader(mnist_val, batch_size=64)
mnist_test = DataLoader(mnist_test, batch_size=64)
```

PyTorch Lightning

```
from torch.utils.data import DataLoader, random_split
from torchvision.datasets import MNIST
import os
from torchvision import datasets, transforms

class MNISTDataModule(pl.LightningDataModule):

    def prepare_data(self):
        # prepare transforms standard to MNIST
        MNIST(os.getcwd(), train=True, download=True)
        MNIST(os.getcwd(), train=False, download=True)

    def train_dataloader(self):
        transform=transforms.Compose([transforms.ToTensor(),
                                     transforms.Normalize((0.1307,), (0.3081,))])
        mnist_train = MNIST(os.getcwd(), train=True, download=False,
                            transform=transform)
        self.mnist_train, self.mnist_val = random_split(mnist_train, [55000, 5000])

        mnist_train = DataLoader(mnist_train, batch_size=64)
        return mnist_train

    def val_dataloader(self):
        mnist_val = DataLoader(self.mnist_val, batch_size=64)
        return mnist_val

    def test_dataloader(self):
        transform=transforms.Compose([transforms.ToTensor(),
                                    transforms.Normalize((0.1307,), (0.3081,))])
        mnist_test = MNIST(os.getcwd(), train=False, download=False,
                           transform=transform)
        mnist_test = DataLoader(mnist_test, batch_size=64)
        return mnist_test
```

# Lightning Bolts

## More rapid iteration with Lightning Bolts

A collection of well established, SOTA models and components.

[Visit Bolts](#)

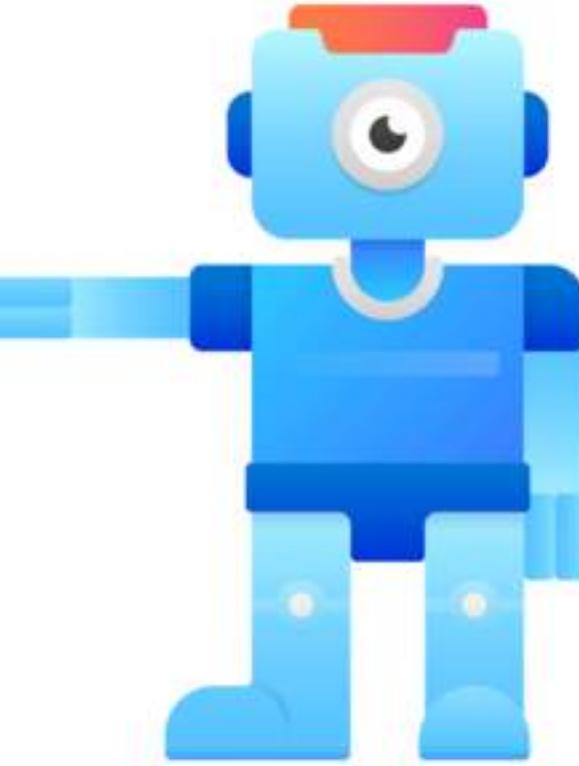
EVEN MORE RAPID ITERATION WITH LIGHTNING

## Lightning Bolts

[PyTorch Lightning Bolts](#) is a community-built deep learning research and production toolbox, featuring a collection of well established and SOTA models and components, pre-trained weights, callbacks, loss functions, data sets, and data modules.

# PyTorch Lightning Exercise

- Convert our Cats vs Dogs PyTorch Code into a Lightning Module
- Auto Batch size finder
- Auto Learning Rate Selector
- Implement Callbacks
  - Early Stopping
  - Model Checkpointing
  - Logging



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**PyTorch Lightning Exercise**

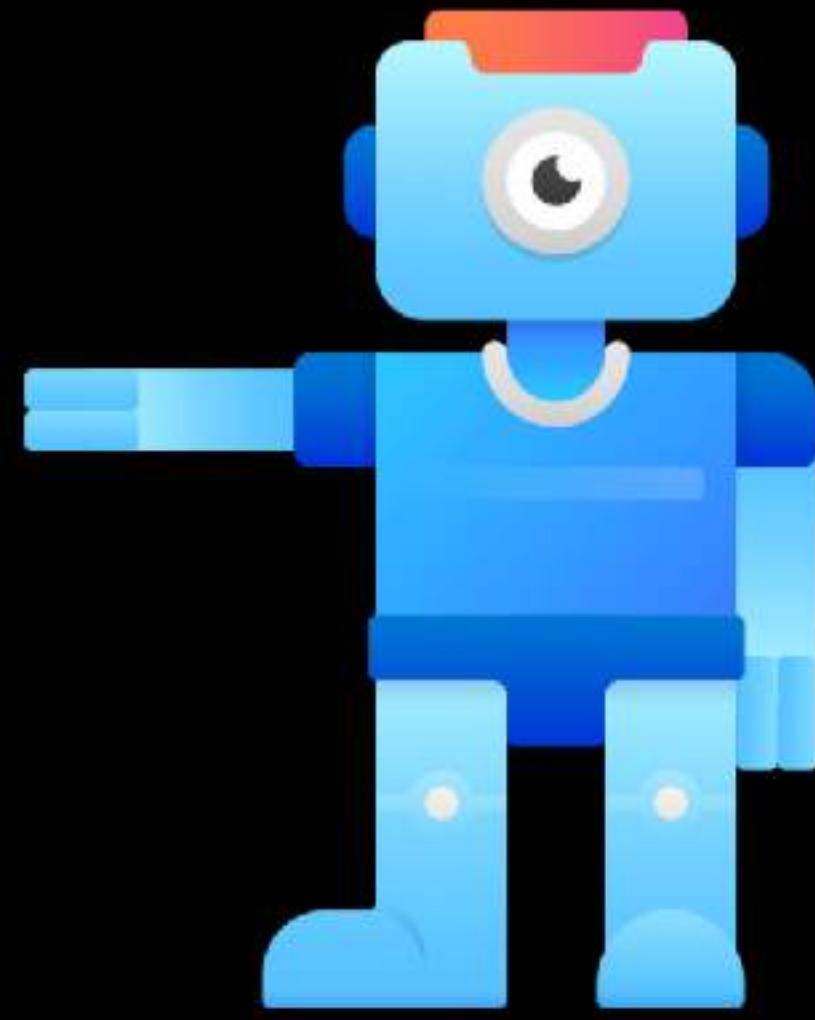


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Transfer Learning**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Transfer Learning

**Why you don't always need to train from scratch**

# Transfer Learning

- In practice, very few people train an entire Convolutional Network from scratch (i.e. random initialization of weights), because it is relatively rare to have a dataset of sufficient size.
- Instead, it is common to **pretrain** a ConvNet on a very large dataset (e.g. **ImageNet**, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an **initialization** or a **fixed feature extractor** for the task of interest. The three major Transfer Learning scenarios look as follows:

<https://cs231n.github.io/transfer-learning/>

# Transfer Learning Rationale

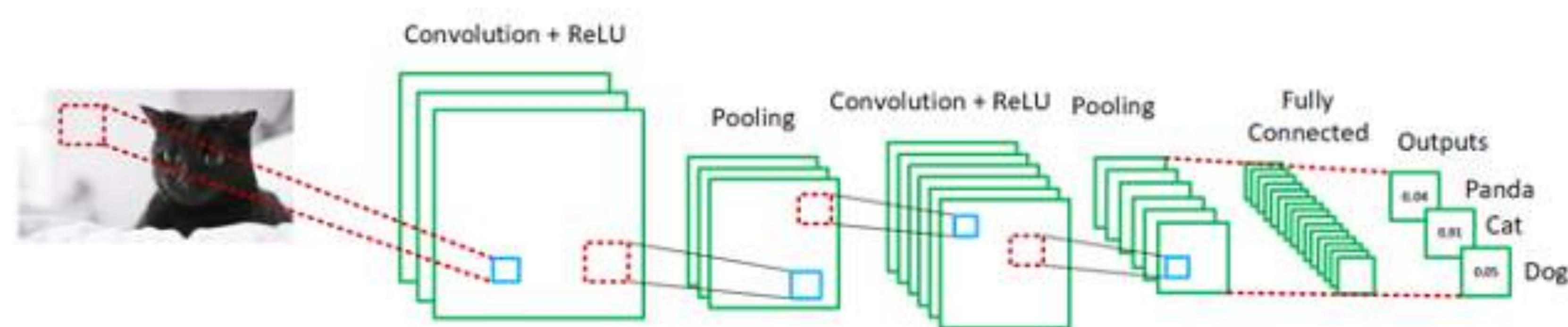
- Training extensive models on vast image datasets like ImageNet can sometimes take weeks.
- A model trained on so much data would have useful embeddings that can be applied to other image domains e.g. edge detectors, pattern and blob detectors
- Example, an ImageNet model trained to detect Tigers, Lions and Horses might be useful to detect other 4-legged mammals
- Transfer Learning is the concept where we utilise trained models in other domains to attain great accuracy as well as faster training times.

# Transfer Learning - 2 Major Types

- Feature Extractor
- Fine Tuning

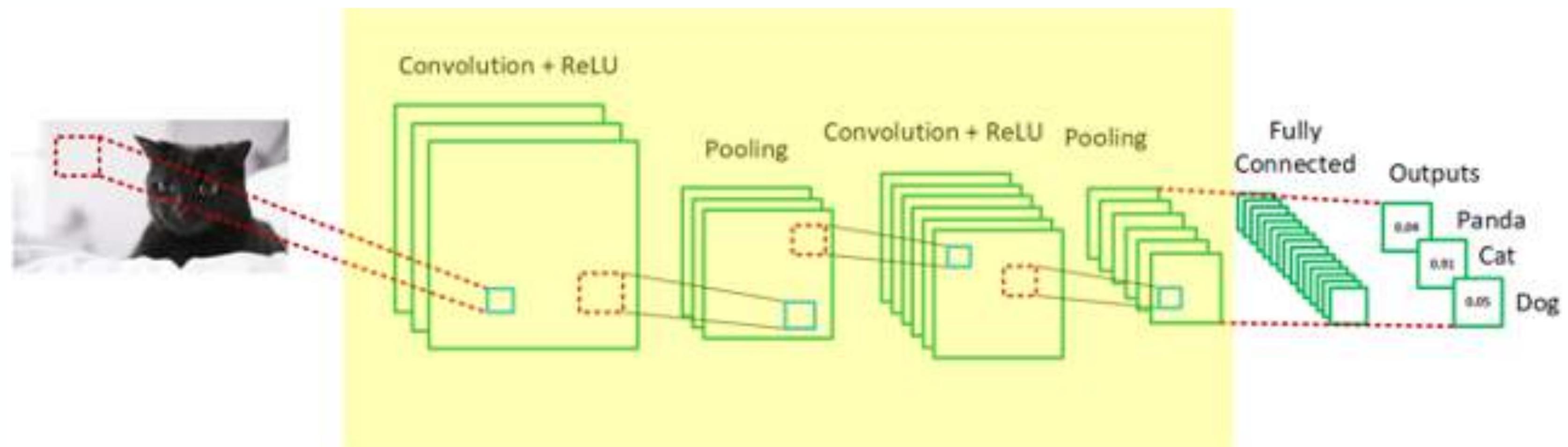
# Transfer Learning - Feature Extractor

## Take a pre-trained Network



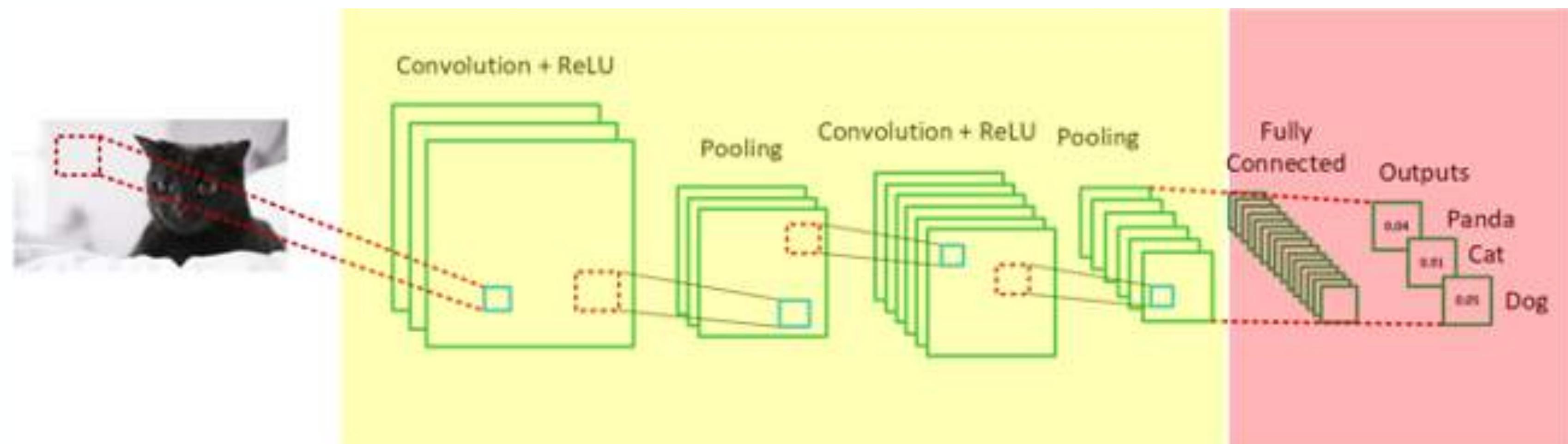
# Transfer Learning - Feature Extractor

Freeze the CONV weights/layers - meaning they remain fixed



# Transfer Learning - Feature Extractor

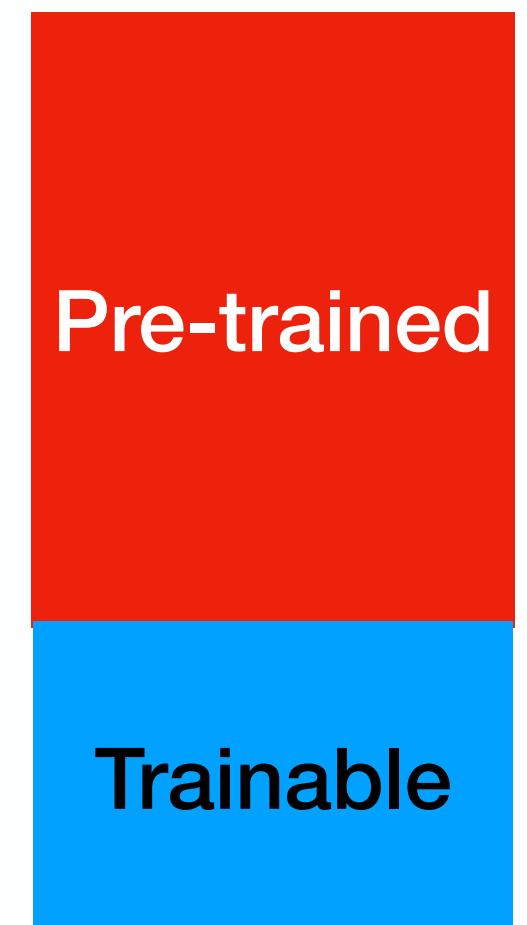
## Replace the Top Layer with your Top Layer and train it



# Transfer Learning - Feature Extractor

The steps involved in Transfer Learning by Feature Extraction

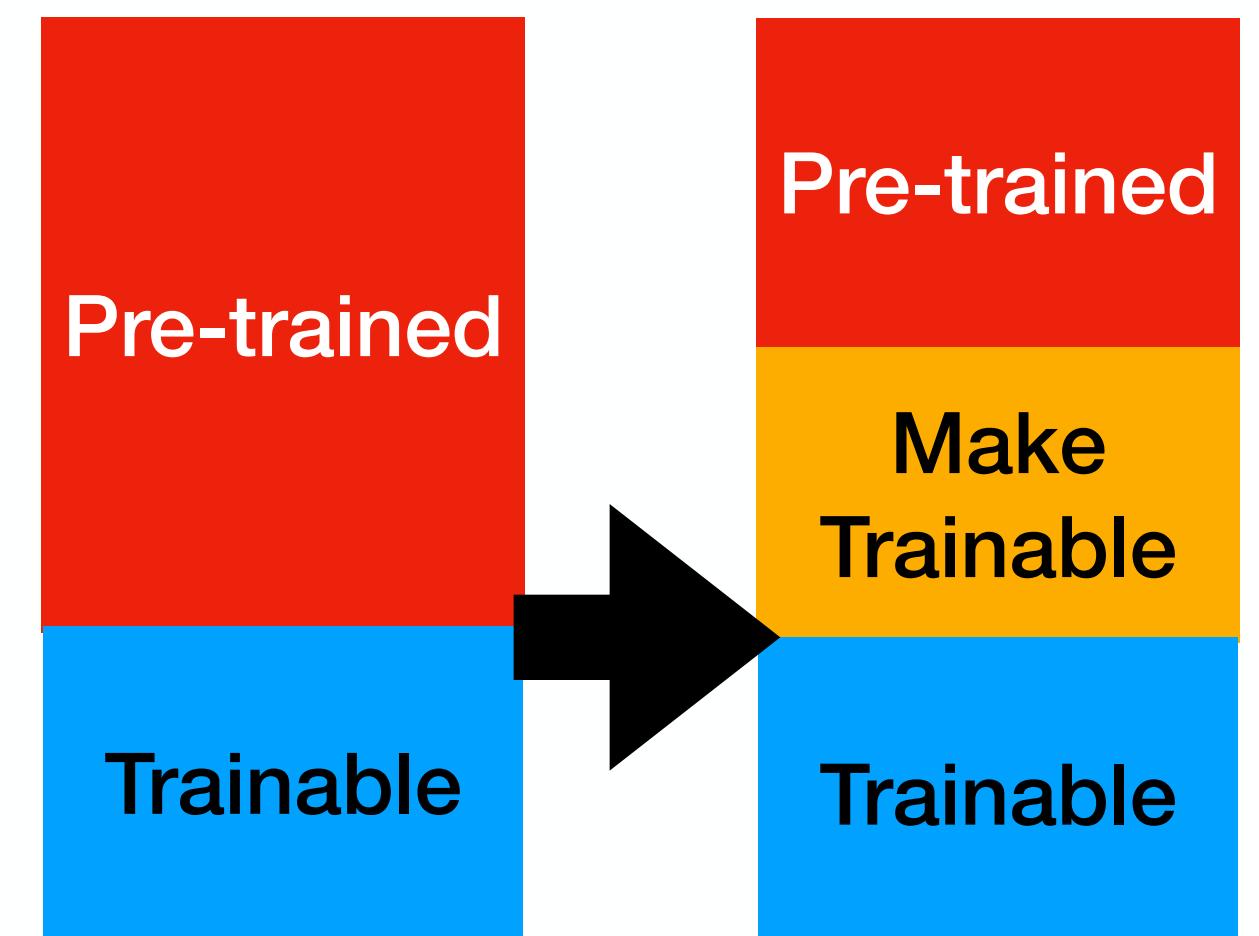
1. **Freeze bottom layers** of a pre-trained network
2. **Replace the top half of the network with your top**, so that it outputs only the number of classes in your dataset
3. Train the model on your new dataset



# Transfer Learning - Fine Tuning

In Fine Tuning, we complete all the steps in Feature Extraction but then we:

1. Unfreeze all or parts of the pre-trained model
  2. Train the model for a few epochs, this is where we '**fine tune**' the weights of the pre-trained model.
- The intuition behind this is earlier features maps in a ConvNet learn generic features, while the later layers learn specifics about the image dataset. By fine tuning we change those specifics from the pre-trained model to the specifics of our dataset.

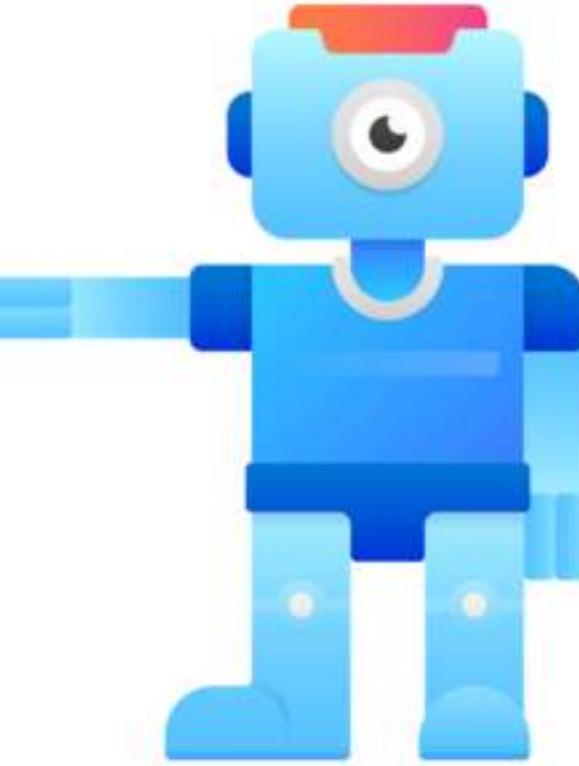


# Transfer Learning - When Do We Use?

- Ideal scenario - New dataset is large and similar to the pre-trained original dataset. Models should not overfit.
- Not ideal but still recommended - New data is large but different.
- If data is small (even if similar) Transfer Learning and Fine Tuning can often overfit on the training data. A useful idea at times is to train a linear classifier on the CNN outputs.

# Transfer Learning Advice

- **Learning Rates** - use very small learning rates for pre-trained models especially when fine tuning. This is because we know the pertained weights are already very good and thus don't want to change them too much
- Due to parameter sharing you can train a pre-trained network on images of different sizes



**MODERN  
COMPUTER  
VISION**

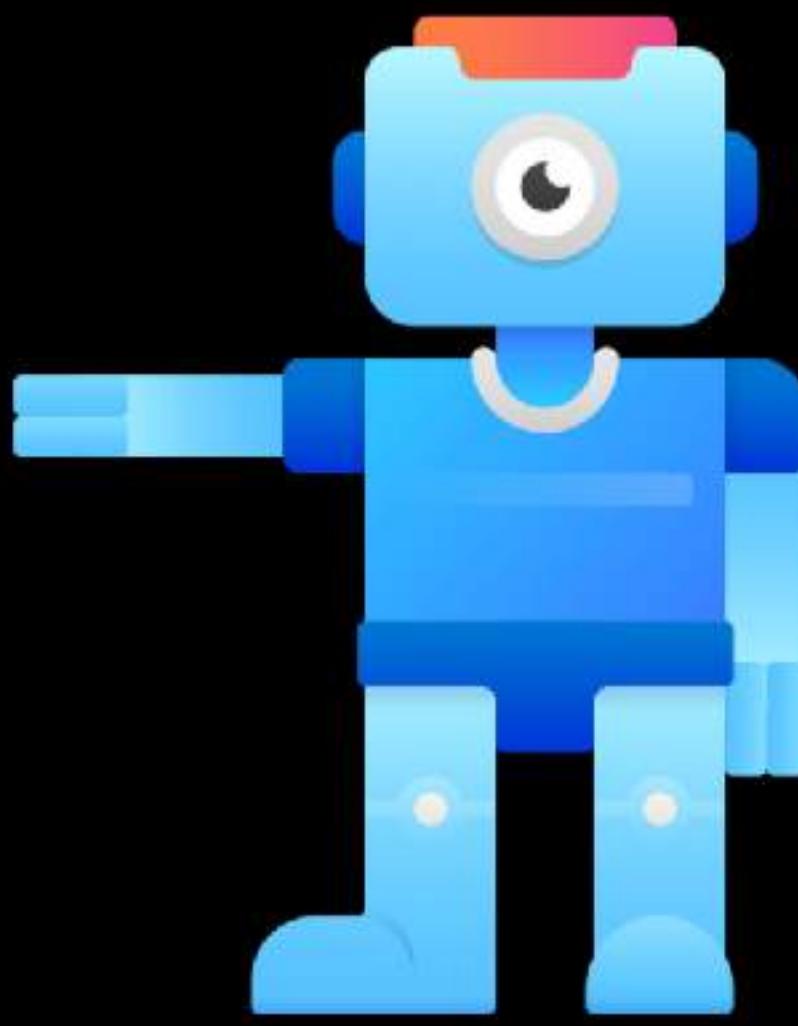
BY RAJEEV RATAN

# Next...

**Transfer Learning and Fine Tuning in Keras and PyTorch**

# Google DeepDream

## How to produce trippy effects in images

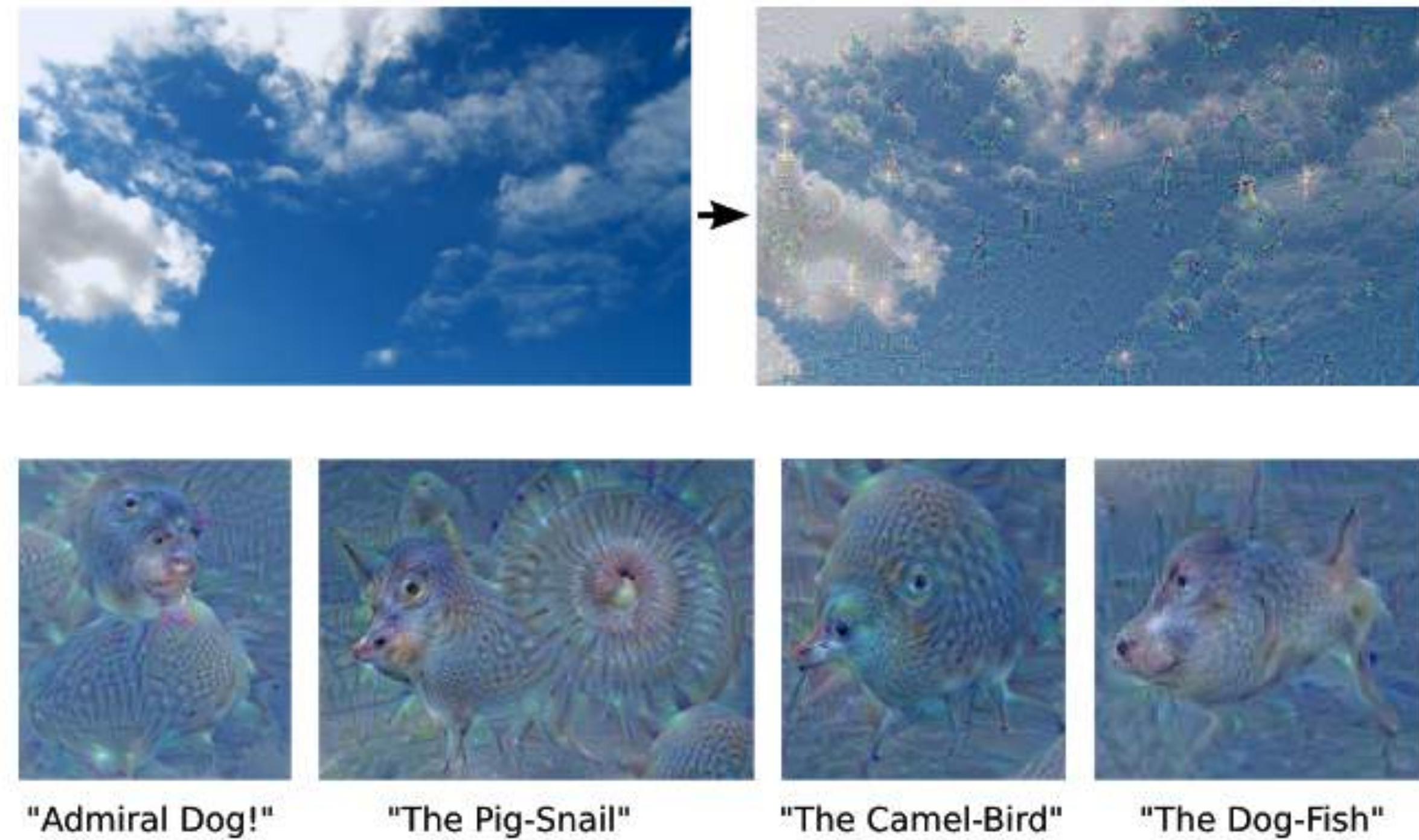


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# What is DeepDream?

- Simply, we take an image and then output ‘dreams’ or hallucinogenic effects on the image
- So what’s the process to do this?



# Trippy isn't it



# DeepDream Process Overview

- The DeepDream algorithm effectively is asking a pre-trained CNN to take a look at an image, identify patterns you recognise and then amplify it.
- It uses **representations** learned by CNNs to produce these hallucinogenic or ‘trippy’ images.
- It was introduced by C. Olah A. Mordvintsev and M. Tyka of Google Research in 2015 in publication titled “DeepDream: A Code Example for Visualizing Neural Networks”



Image Source: <https://deeplearning.net/deepdream/>

# The DeepDream Algorithm - High Level

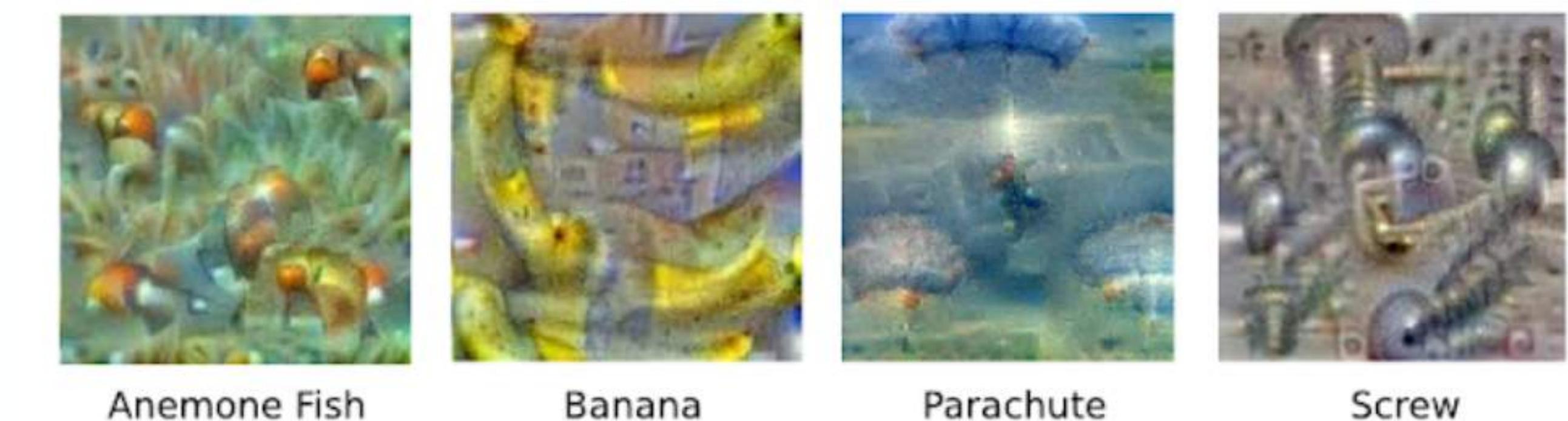
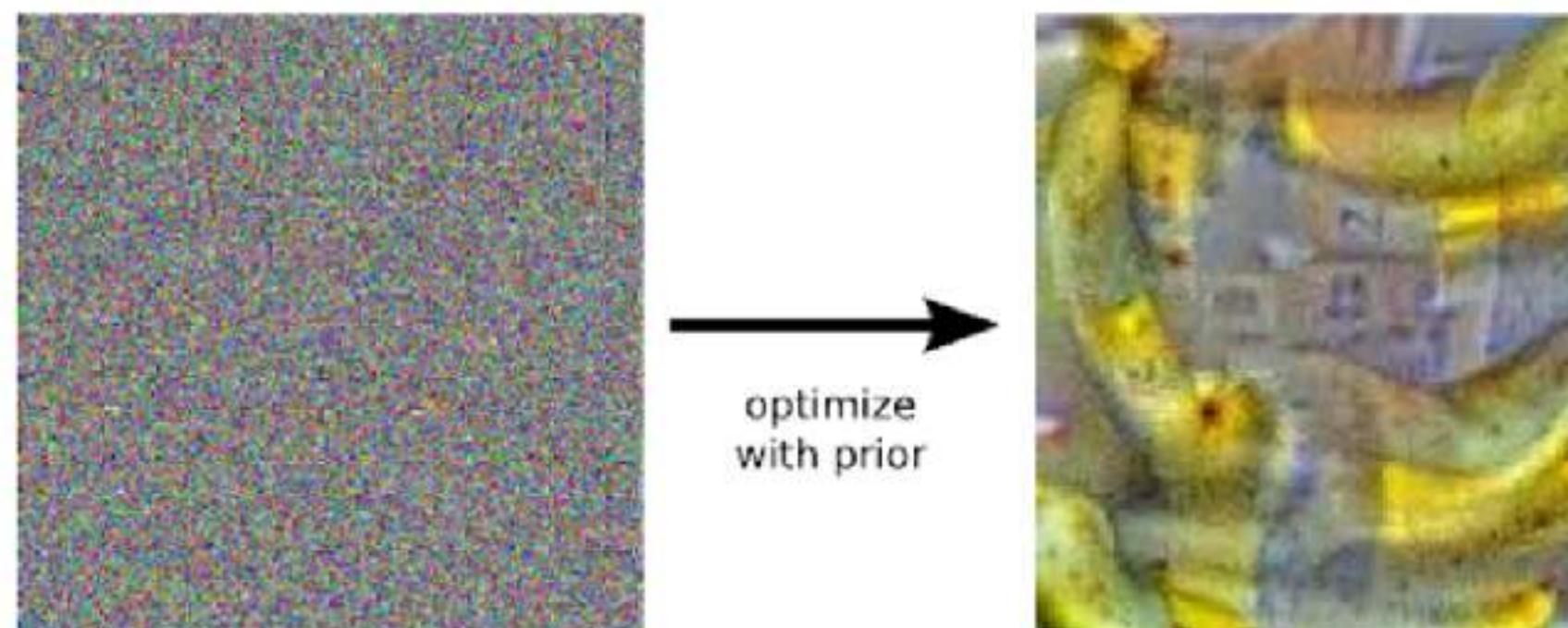
- Remember how we created our filter visualisations?



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55\*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55\*55 distinct locations in the Conv layer output volume.

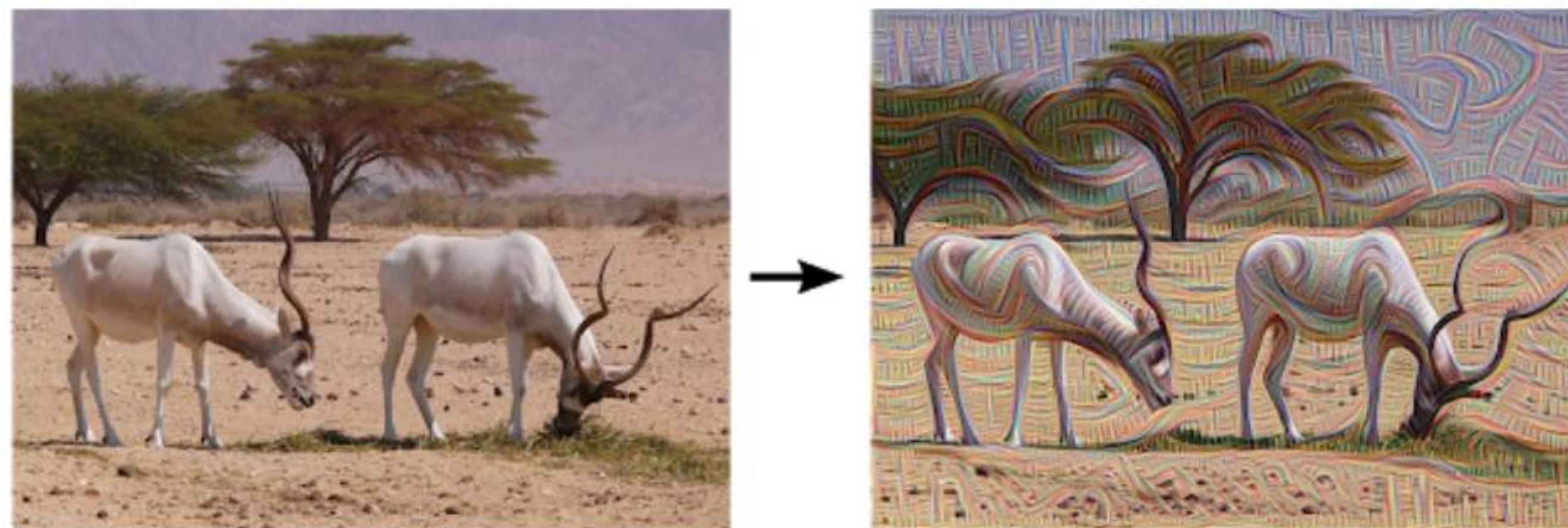
# The DeepDream Algorithm - High Level

- And class maximisation visualisations?



# The DeepDream Algorithm - High Level

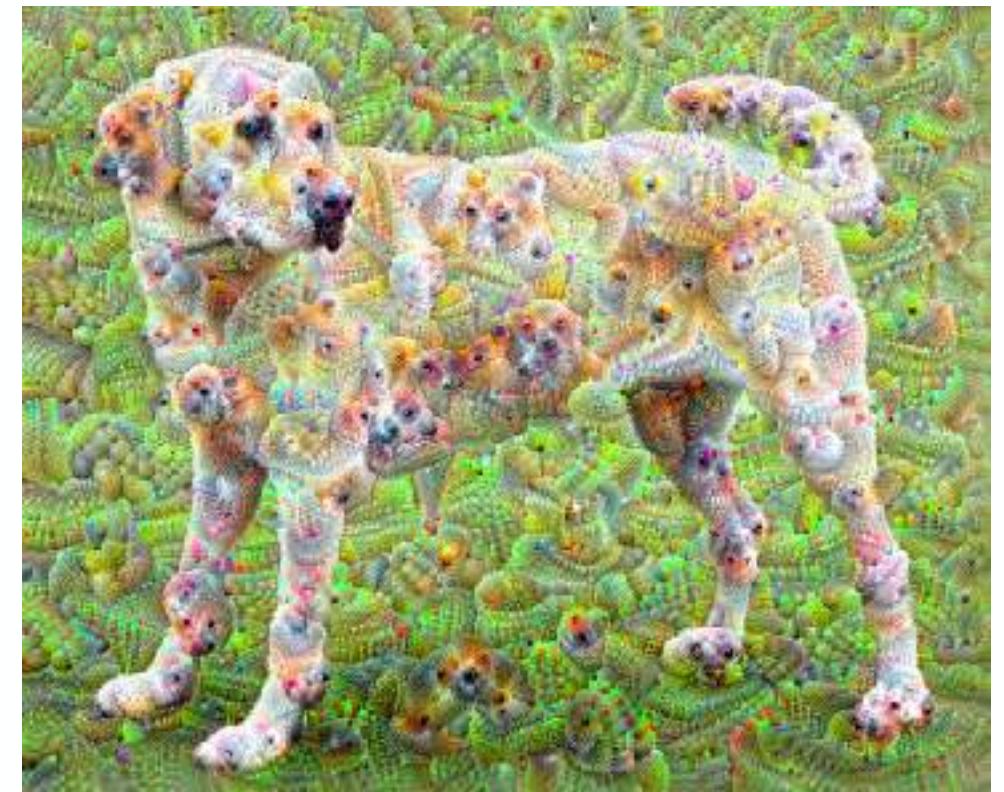
- In Deep Dream, instead of maximising a class output, we let the network decide.
- We feed it an input image and allow the network to analyse the image
- We then choose a layer and ask the network to amplify or enhance whatever it detected.
- Remember each layer deals with different features (high-level, low level) so the complexity of the pattern depends on these features. E.g. low level features look like brushes or strokes.

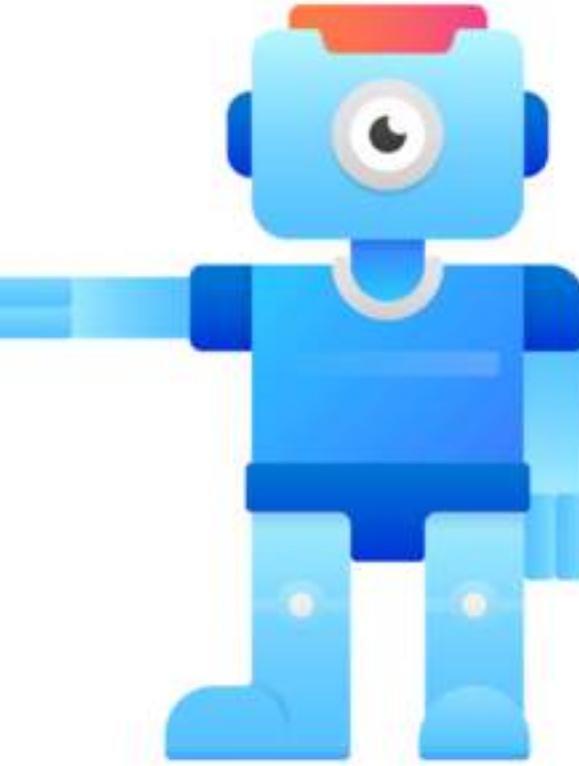


Left: Original photo by [Zachi Evenor](#). Right: processed by Günther Noack, Software Engineer

# The DeepDream Algorithm Low Level

- Get an input image
- Load a pertained model (VGG16, InceptionV3 etc.) to be used as a **feature extractor**
- **Calculate loss** which is the sum of the activations between the chosen layers. We also **normalise** at each layer so contributions from each layer are ‘equal’
- We use **gradient ascent** to do this, unlike gradient descent where we’re trying to find the local minima, gradient ascent tries to **maximise its loss** which allows the network to find less meaningful patters.
- Apply this at different scales (sizes) so that the patterns don’t occur at the same granularity





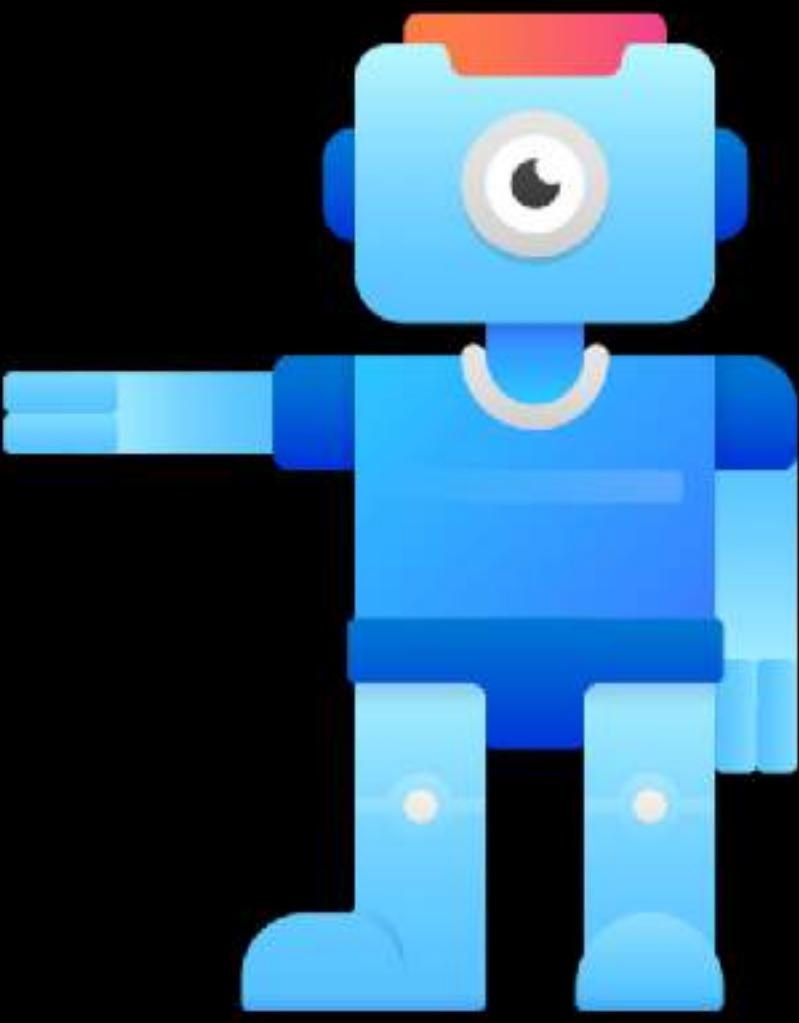
**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Experiment with Google Deep Dream in Keras and PyTorch!**

**Neural Style Transfer**  
Copying Artistic Style onto images

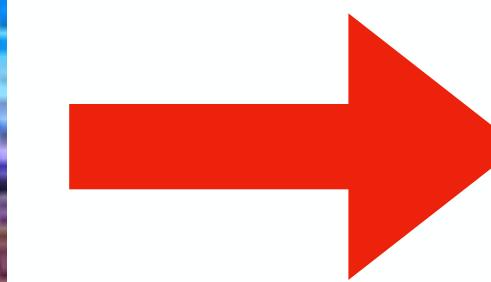
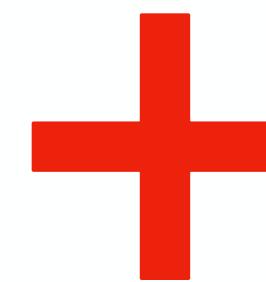


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Neural Style Transfer - AI Art

- Introduced by Leon Gatys et al. in 2015, in their paper titled “[A Neural Algorithm for Artistic Style](#)”, the **Neural Style Transfer** algorithm went viral resulting in an explosion of further work and mobile apps.
- Neural Style Transfer enables the **artistic style** of an image to be applied to another image! It copies the colour patterns, combinations and brush strokes of the original source image and applies it to your input image.

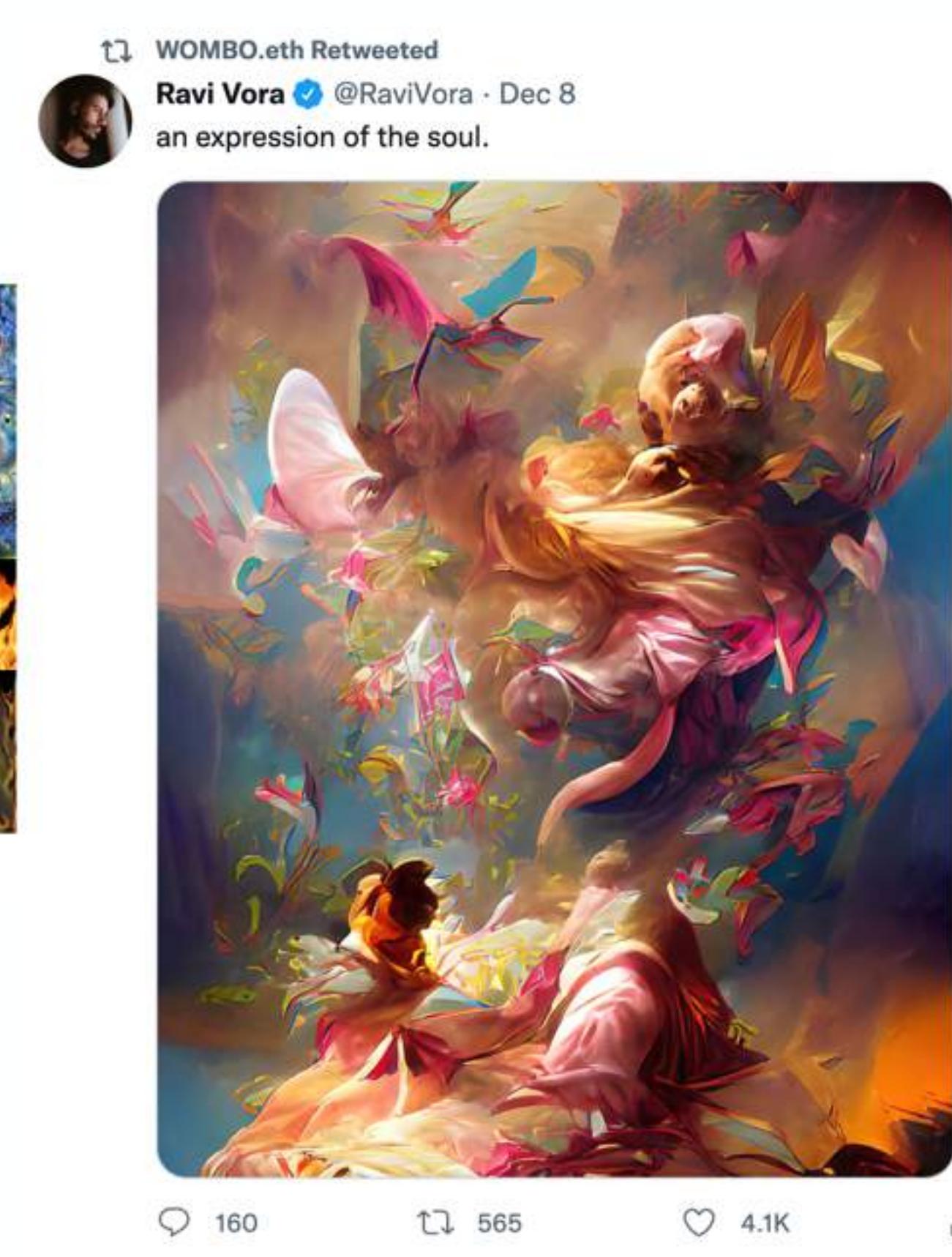


# Amazing Apps like Deep Dream Generator and Womba



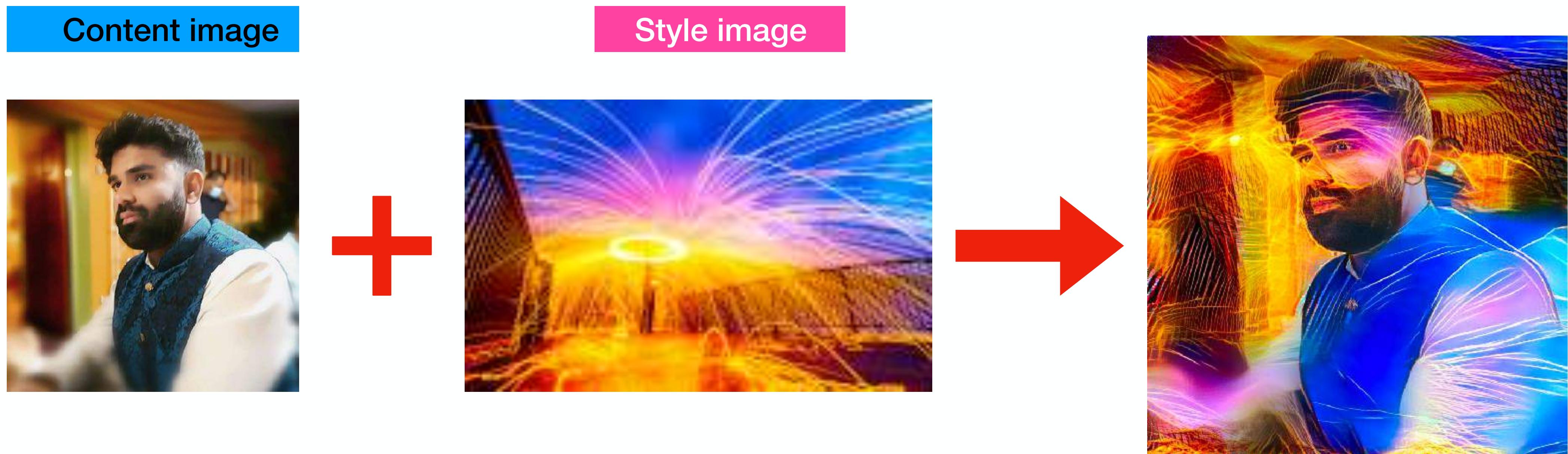
Top: [DeepDreamGenerator.com](https://DeepDreamGenerator.com)

Right: [Womba.ai](https://Womba.ai)



# How does it Work?

- NST uses two images, the **content** image and the **style** image
- We take the **style** of one image and **transfer** it onto the **content** of another image



# Neural Style Transfer using Neural Networks

- We take a pre-trained network (VGG19 on ImageNet)
- Define and combine three loss functions which we will minimise
  - Content Loss
  - Style Loss
  - Total-variation Loss

# Content Loss

$$L_{content} = \sum_l \sum_{i,j} (\alpha C_{i,j}^l - \alpha \tilde{C}_{i,j}^l)^2$$

Content Image   
 Generated Image
  
 ↓                   ↓
   
 Content Weight

- The content loss function measures how **similar the generated image is to the content image**
- It uses Euclidian distance (L2-Norm) to measure the difference between features of the content image and generated image
- We build this loss function by using a pre-trained Network like VGG16
- We then select a higher-level layer that serves as our Content Loss
- We then compute the activation of this layer for both the content and style image
- Then (as you can see above) we take the L2-Norm between these activations

# Style Loss

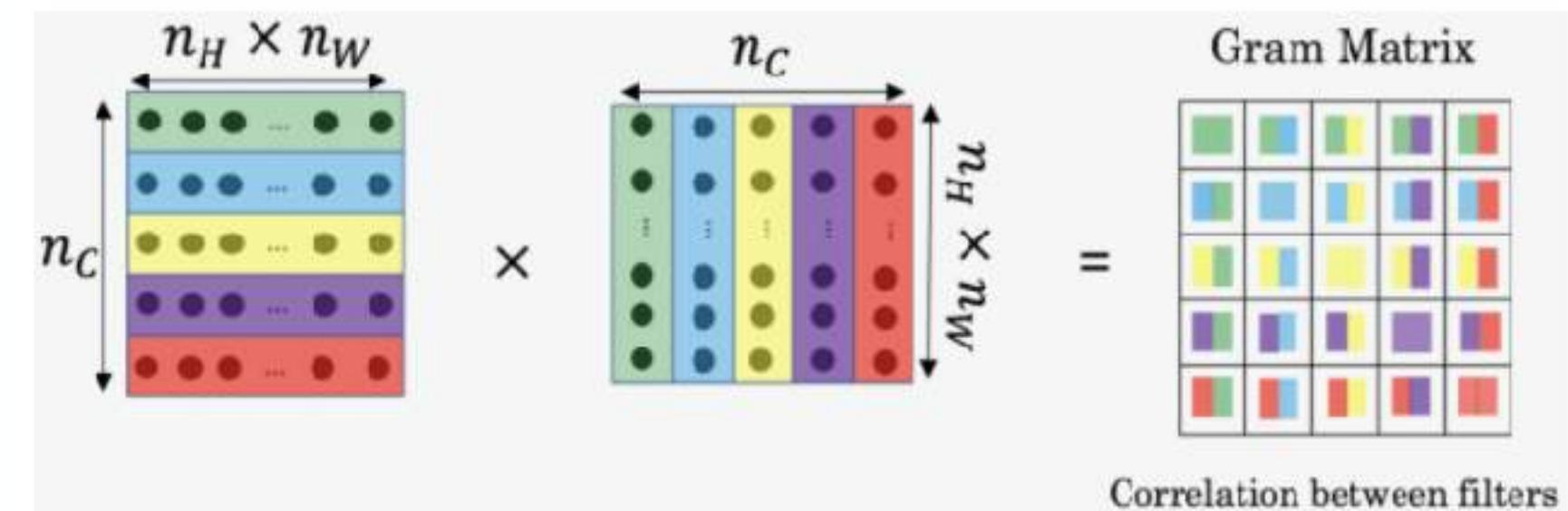
$$L_{style} = \sum_l \sum_{i,j} (\beta G_{i,j}^{s,l} - \beta G_{i,j}^{p,l})^2$$

Style Weight  
 ↗ Gram Matrix for Style Image      ↘ Gram Matrix for Style Image

- Measures how **different** the **generated image** is (in style features) from your **style image**.
- Style loss uses multiple layers (or multi-scale representation), this allows it to capture
  - Low level features/layers such as edges
  - Mid level features such as blobs and corners
  - High level features such as more complex patterns
- The **style** representation is give by the **Gram Matrix**

# Gram Matrix

- For the style loss, unlike the content loss, we don't just find the difference in activations.
- We find the correlation between these activations across different channels of the same layer.



Source: [deeplearning.ai](https://deeplearning.ai)

- Each filter activates upon ‘seeing’ a feature e.g. a cat’s nose.
- Now if another filter in that layer activates upon detecting a cat’s eye this would mean they both activate together when detecting a cat. This means they’re correlated.

# Gram Matrix

- In order to capture the style but not the global arrangement of the content image, we must rely on the correlations between our feature values
- We take the the L2 norm between the gram matrix between layer activations
- We minimise loss between the style of the output image with our original style image, thereby forcing the style of the output image to correlate with the style of the style image.

# Total Variation Loss

- The variation loss was included as it reduces noisy and pixelated looking outputs.
- It allows us to maintain smoothness and spatial continuity.
- $$L_{total}(i, j, k) = \alpha L_{content}(i, k) + \beta L_{style}(j, k)$$

- It operates only on the output image with its goal being to enhance the output image.

# Combining Loss Functions

- The final loss function is the **sum of all previous loss functions** with each component weighted
- This weighting allows us to **tweak** how we want our resultant image i.e. do we want more style or more of the content?
- To **implement** we start iterating (for a preset number) using gradient descent (sometimes using L-BFGS) to minimise our loss functions.



**MODERN  
COMPUTER  
VISION**

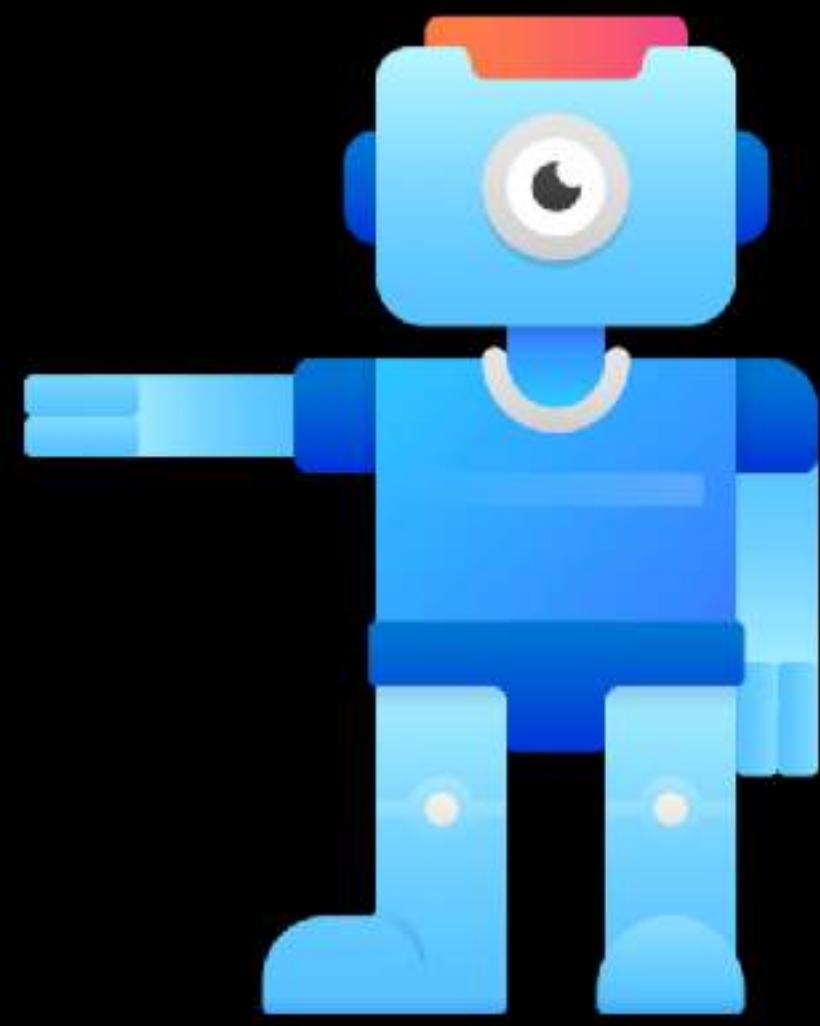
BY RAJEEV RATAN

# Next...

**Implement Neural Style Transfer in Keras and PyTorch**

# Autoencoders

## How Neural Networks can perform Representational Learning



MODERN  
COMPUTER  
VISION

BY RAJEEV RATAN

# Autoencoders

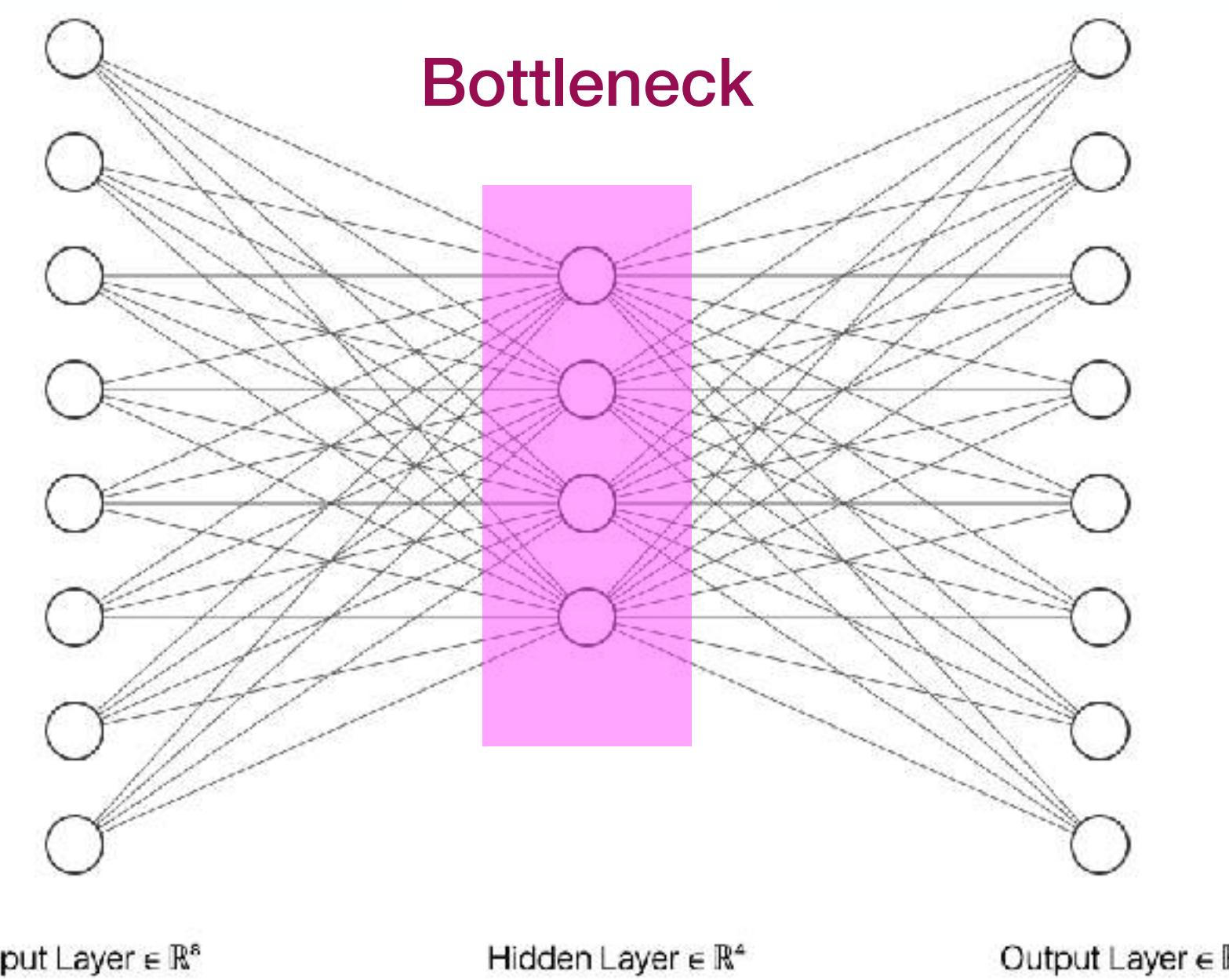
- An **Unsupervised Learning** technique that is used for **representational learning**
- Recall that our **CNN filters act as feature detectors**:
  - high level such as patterns
  - low level such as edges or blobs
- What if we could exploit what a CNN learns about a dataset so that it acts as a method of compression?

# What do Autoencoders do?

- They learn to **compress** data based on their correlations between input features
- Some applications include:
  - Denoising (images, even audio)
  - Image Inpainting
  - Information Retrieval
  - Anomaly Detection

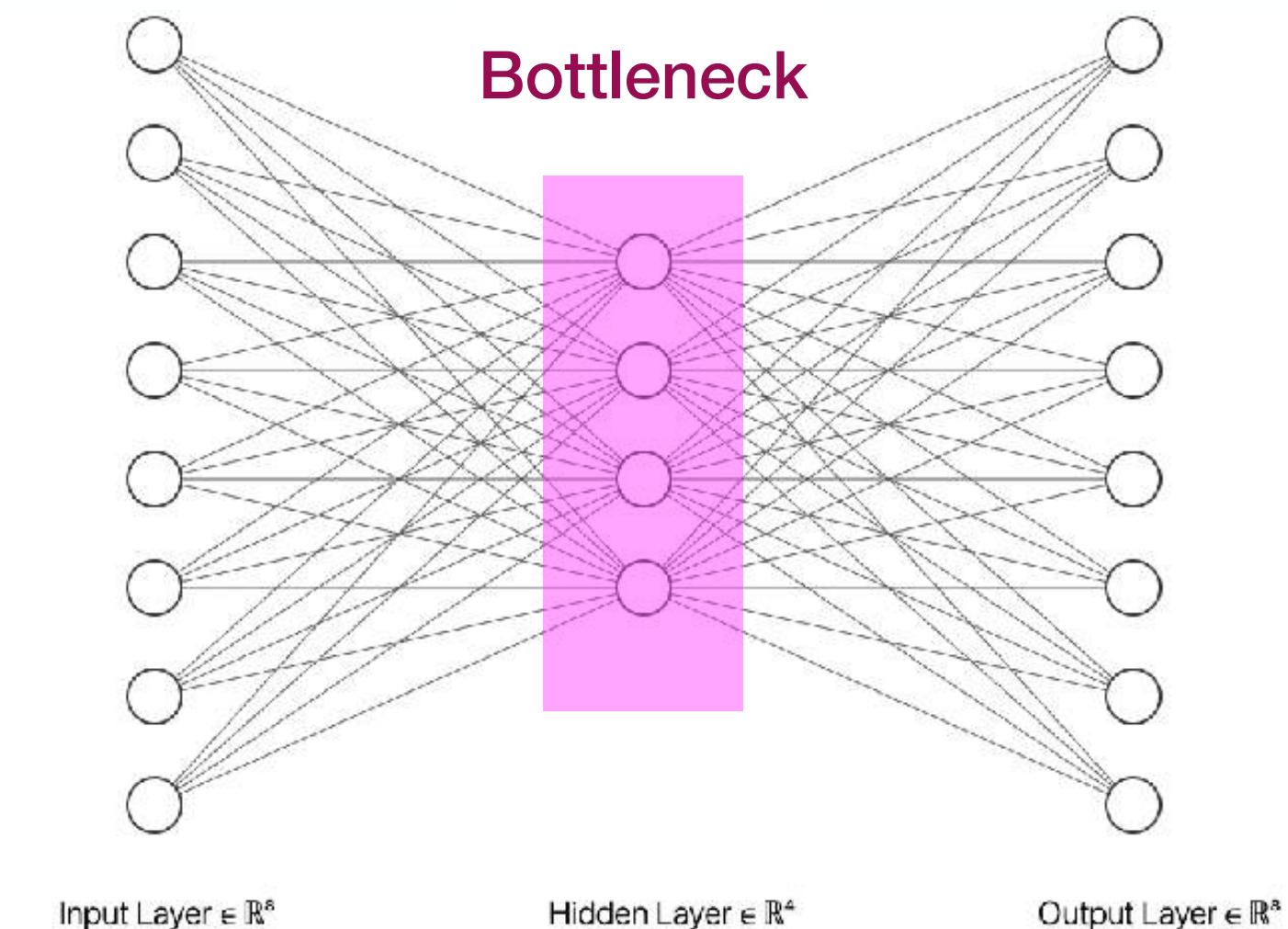
# Autoencoders

- Autoencoders are neural networks that use a **bottleneck** architecture which forces a **compressed** knowledge representation of the input data.
- Autoencoders work very well with data that has **correlated** input features (i.e. not independent).



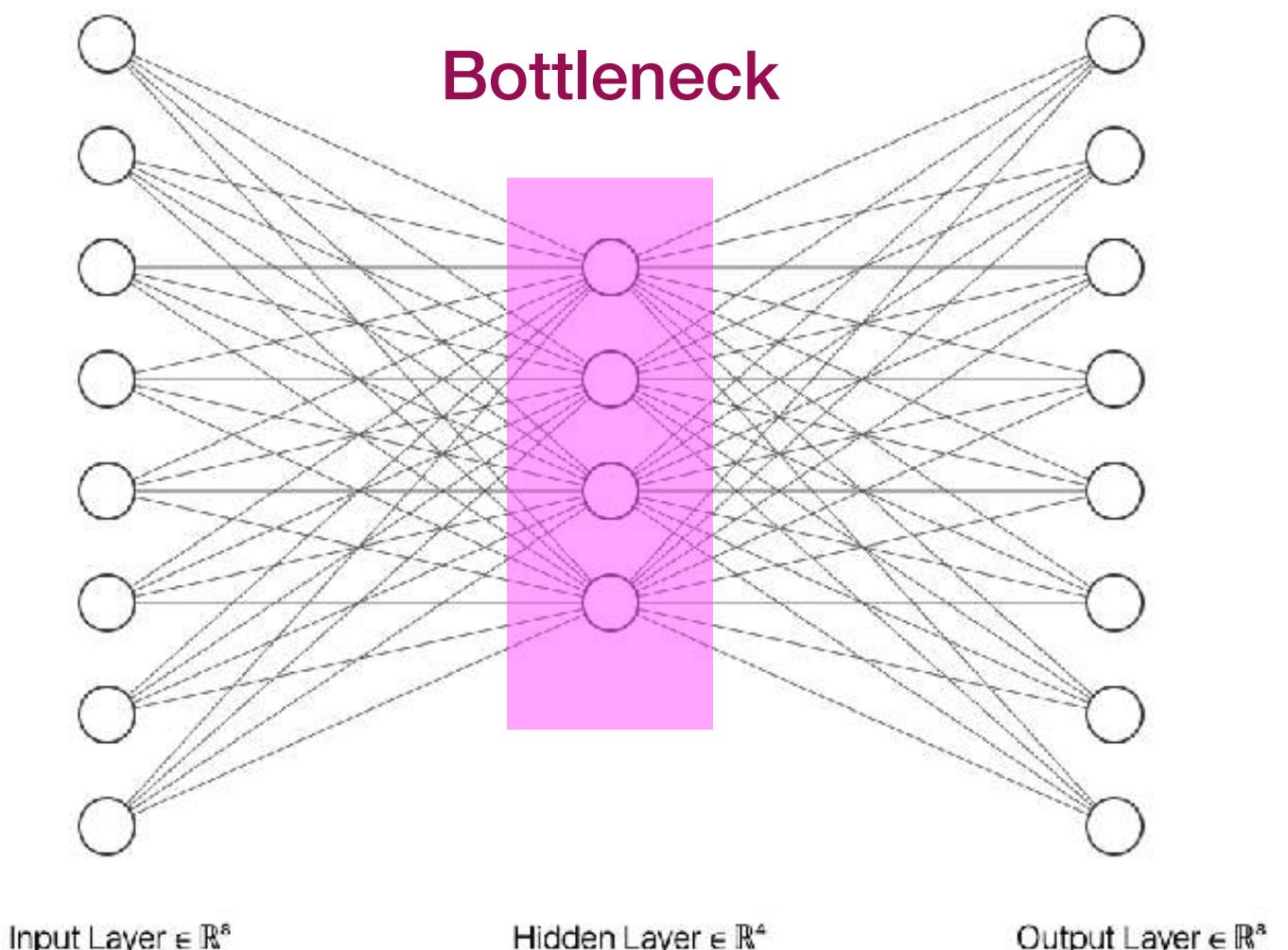
# Bottleneck

- The bottleneck constrains the amount of information that is able to traverse the full network.
- This enables the hidden bottleneck layer(s) to learn a compressed representation of the input data



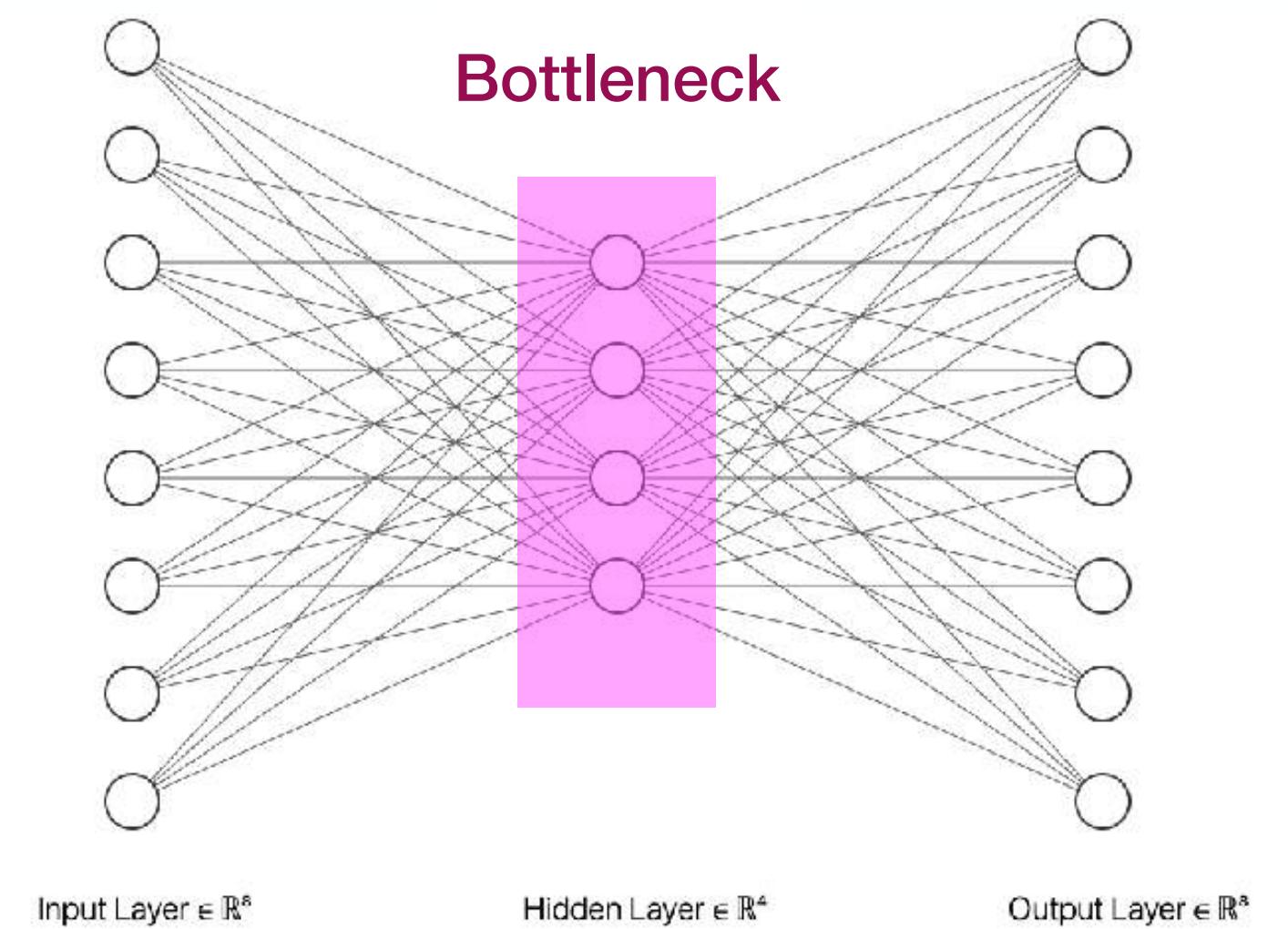
# The Ideal Autoencoder

- The ideal Autoencoder is:
  - Sensitive enough to accurately reconstruct the image
  - Insensitive enough to inputs so that the model doesn't overfit the training data



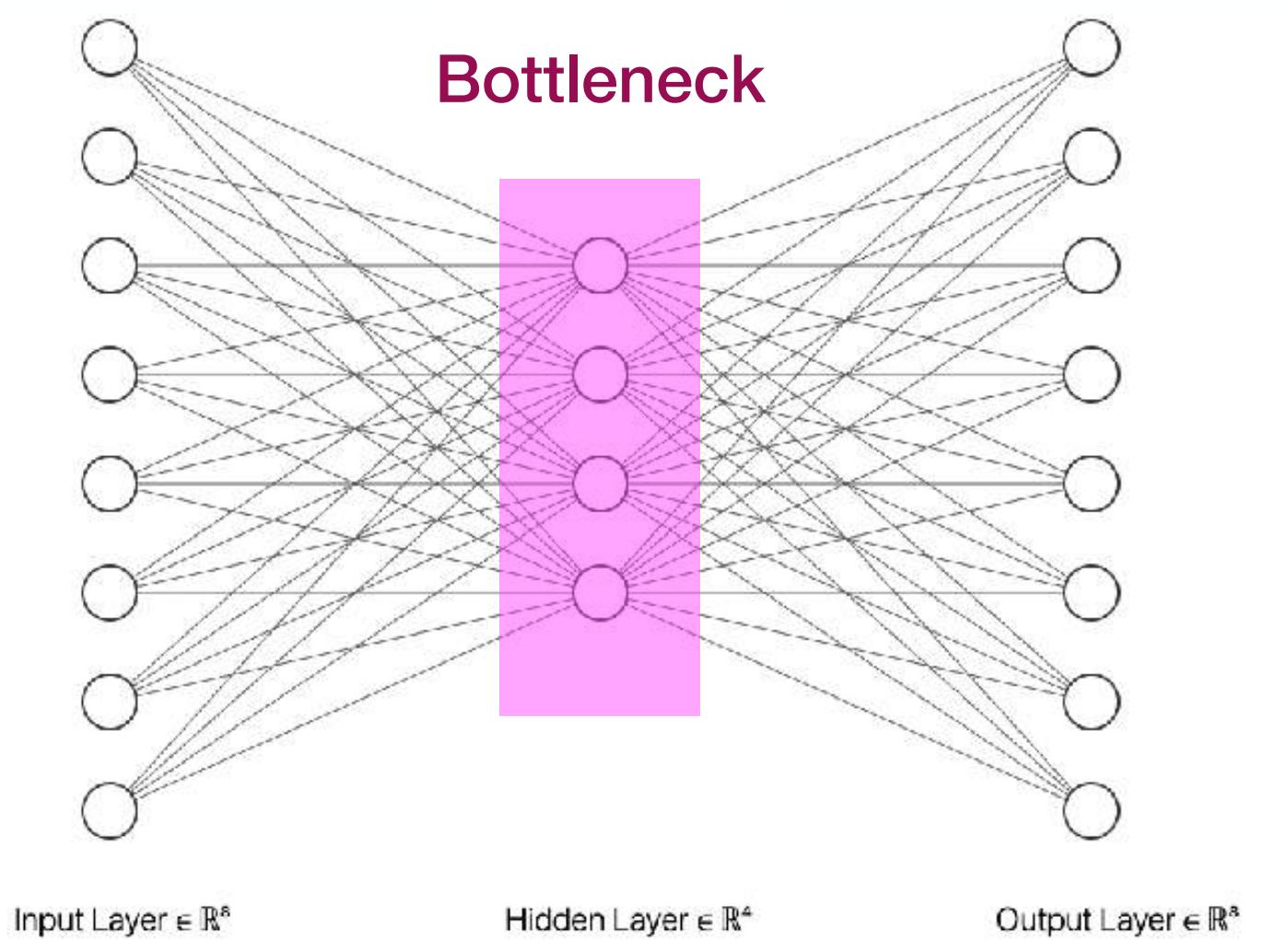
# Autoencoder Architecture

- A simple Autoencoder architecture is one where it bottlenecks/constrains the the number of nodes in the hidden layer(s).
- Notice our input and output match in dimensions, that is because we're reconstructing the input
- Our loss function here penalises reconstruction error
- This allows the model to learn the most important features needed to reconstruct the data/image



# CNN Autoencoder Architecture

- Given that our inputs in Computer Vision applications are images, using Convolution Neural Networks makes total sense
- Using Conv layers provides much better performance



# Training an Autoencoder

- The training process is simple, however there are few differences.
- The target data is the same as the training data
- Likewise for validation, as you're testing how well your encoder-decoder model works
- The loss function can be binary cross entropy or even MSE.

```
autoencoder.fit(x_train, x_train,
                 epochs=50,
                 batch_size=256,
                 shuffle=True,
                 validation_data=(x_test, x_test))
```

# Autoencoder Limitations

- Autoencoders are lossy, meaning the decompressed outputs are degraded compared to the original
- Data-specific meaning that it learns the representation in a specific domain

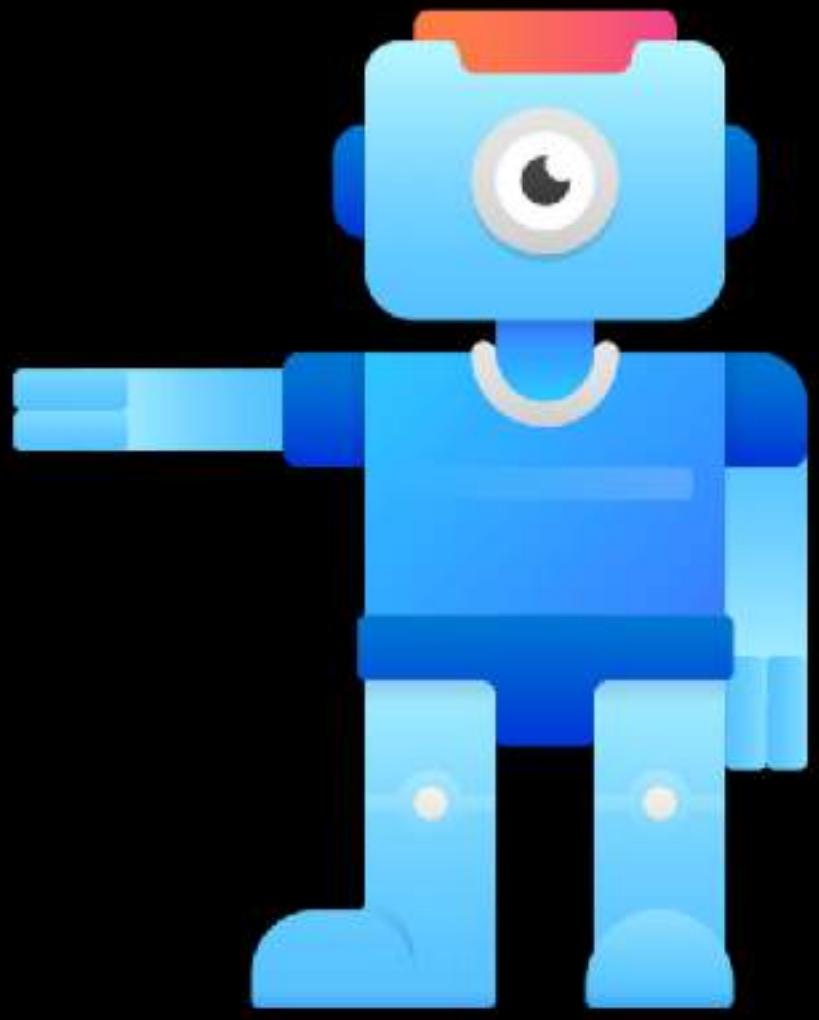


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Autoencoders in Keras and PyTorch**

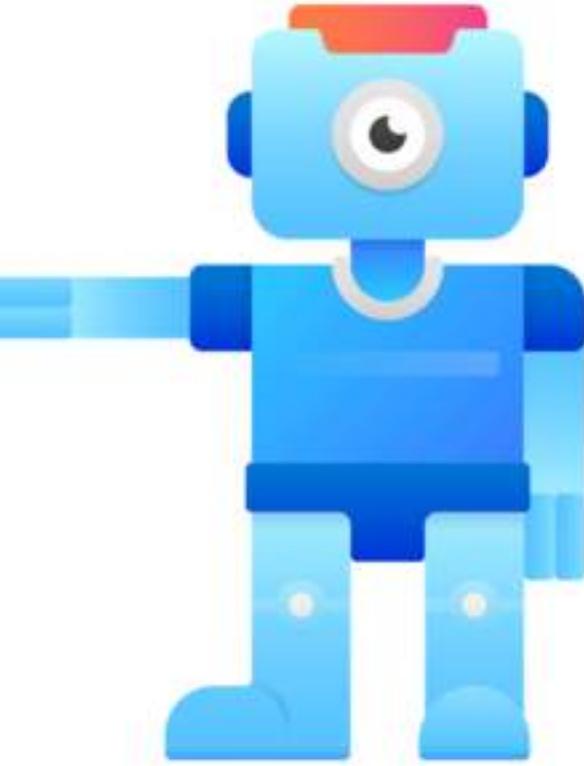


# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Generative Adversarial Networks (GANs)

An Introduction to Generative Adversarial Networks (GANs)

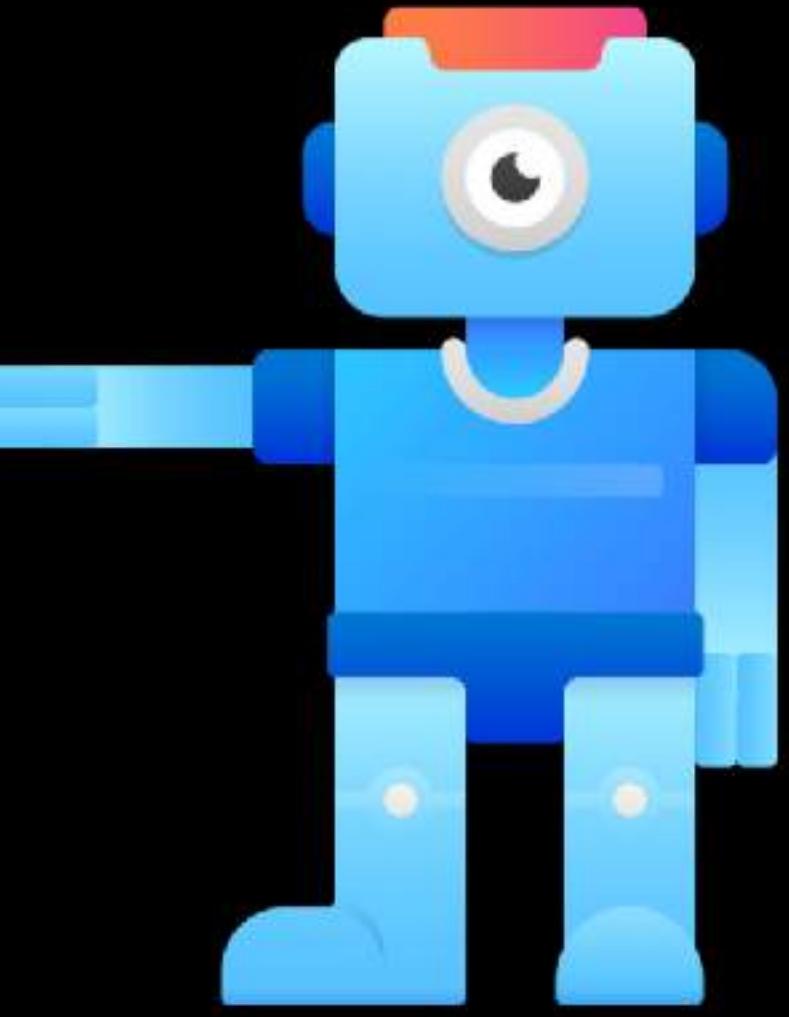


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**How Generative Adversarial Networks (GANs) Work**

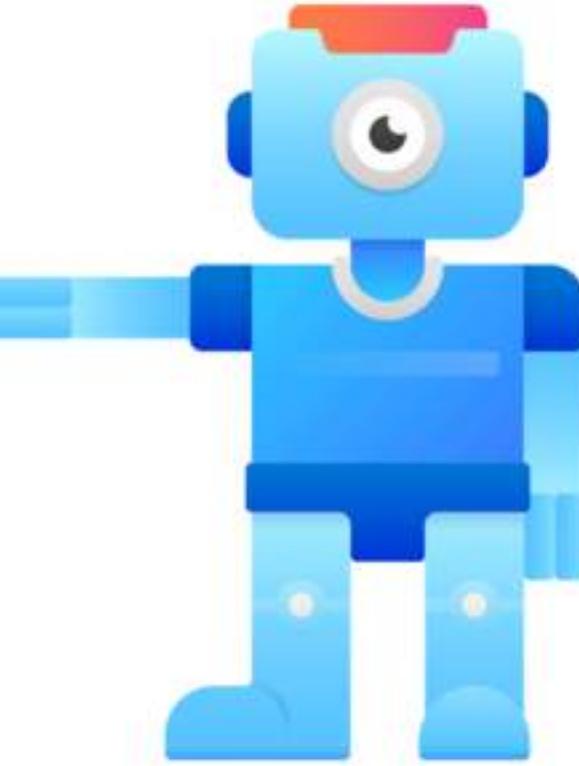


# MODERN COMPUTER VISION

BY RAJEEV RATAN

## How Do GANs Work?

A high-level overview of the basics of GANs

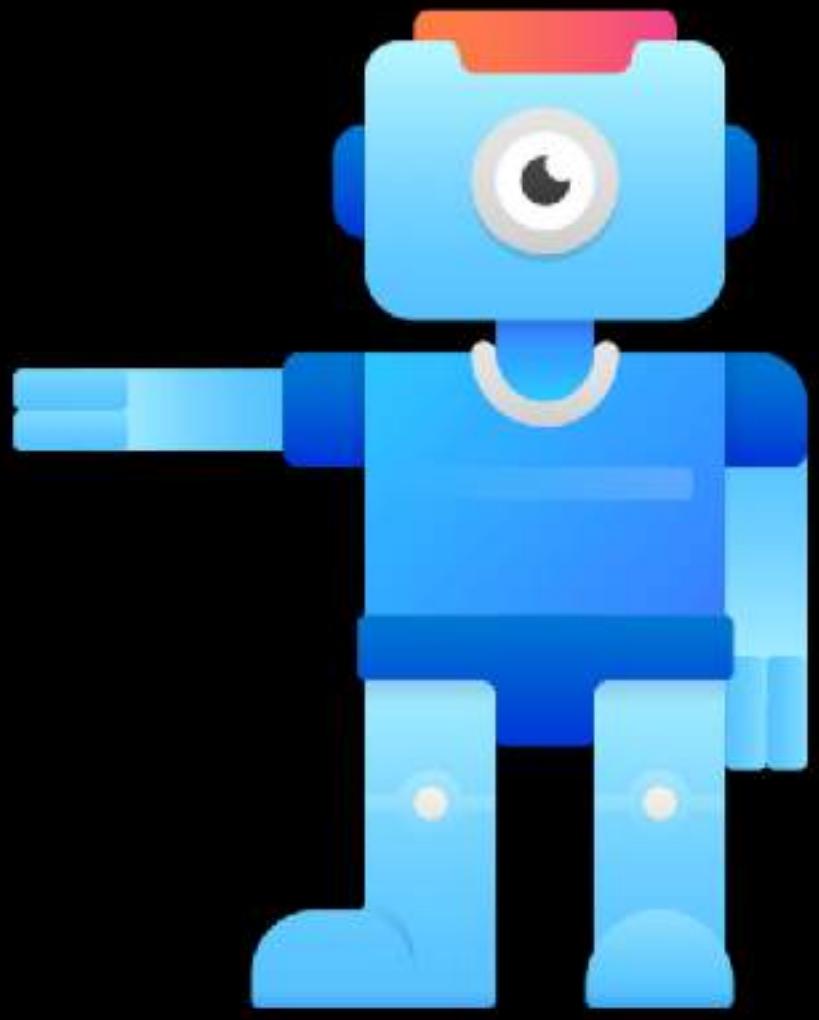


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**A high-level overview of the basics of GANs**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Training Generative Adversarial Networks (GANs)

An Introduction to Generative Adversarial Networks (GANs)

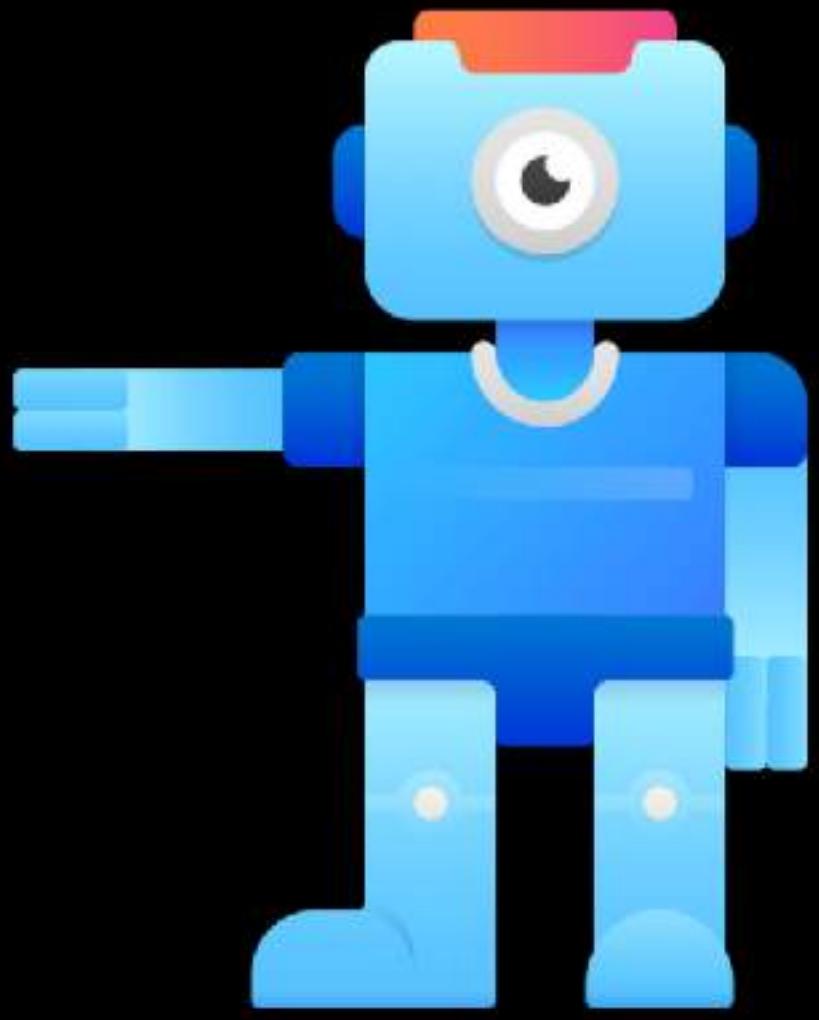


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

Challenges in Training GANs



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Challenges in Training GANs

Things to watch out for when training GANs

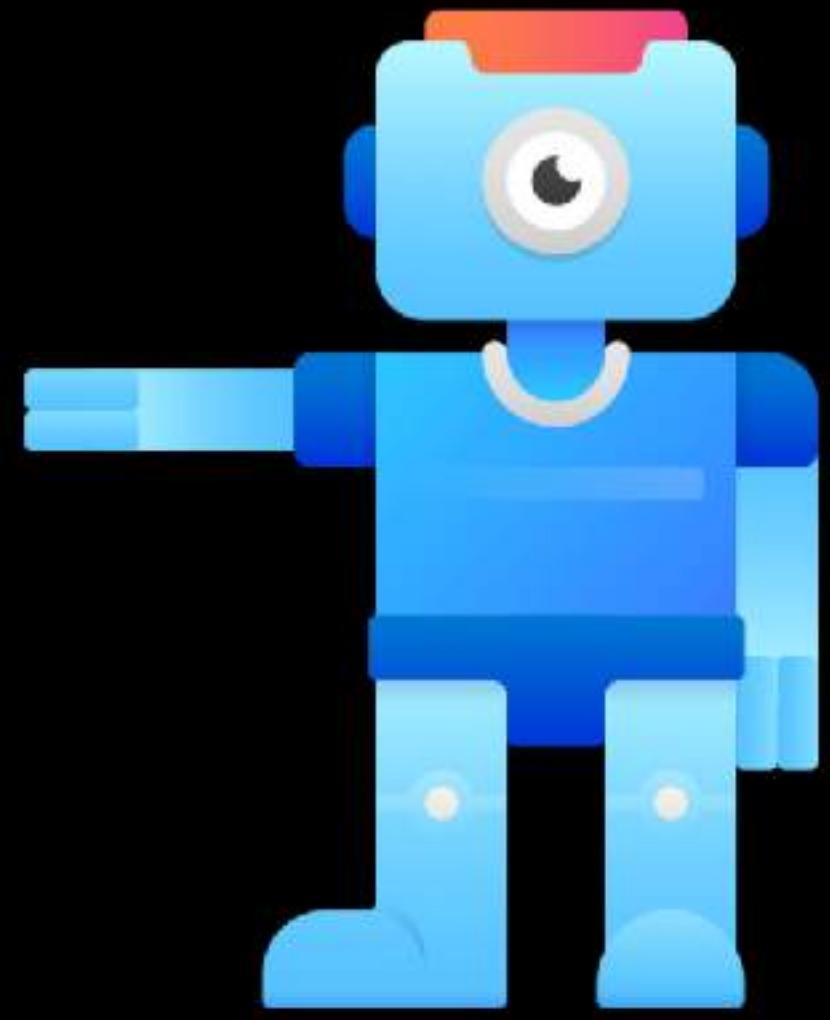


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**GANs in Industry**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## GANs in Industry

Common use cases for GANs in the real world



# MODERN COMPUTER VISION

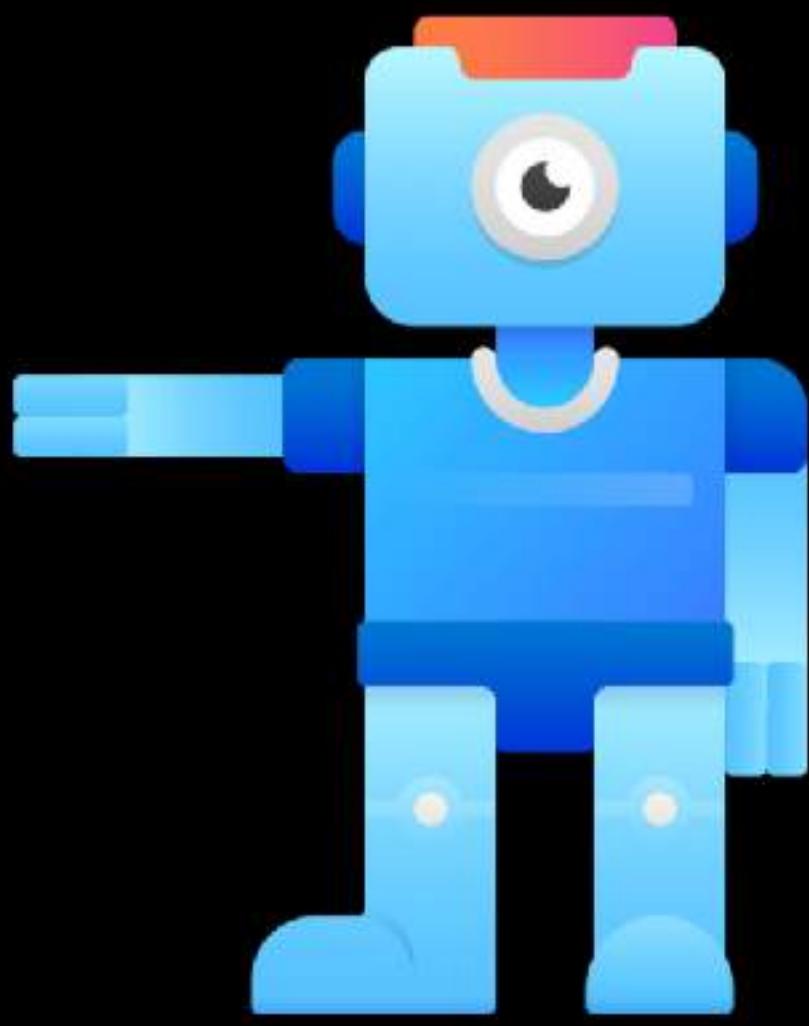
BY RAJEEV RATAN

# Next...

**Siamese Networks**

# Siamese Networks

Siamese Networks for image similarity



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Siamese Networks

- Firstly, what is Siamese?
- The term comes from Siamese twins (conjoined twins) which are twins that unfortunately joined together at birth, sometimes sharing organs.
- Siamese Networks are similarly '**connected**'. They consist of:
  - Two or more identical (in architecture) subnetworks (Neural Networks)
  - The subnetworks share the same parameters & weights
  - Parameter updates are mirrored on both networks (i.e. when one updates, the other one updates as well)



Image Source - [https://commons.wikimedia.org/wiki/File:Chang\\_and\\_Eng\\_the\\_Siamese\\_twins,\\_in\\_a\\_games\\_room.\\_Coloured\\_e\\_Wellcome\\_V0007366.jpg](https://commons.wikimedia.org/wiki/File:Chang_and_Eng_the_Siamese_twins,_in_a_games_room._Coloured_e_Wellcome_V0007366.jpg)

# What are Siamese Networks used for?

- Siamese Networks are very useful for **comparing images** or **image similarity tasks**
- Examples of these are Signature Verification, Face Recognition and Finger Print Matching.
- Siamese Networks give a **simple binary output**, Yes if the images match or No if they do not match.

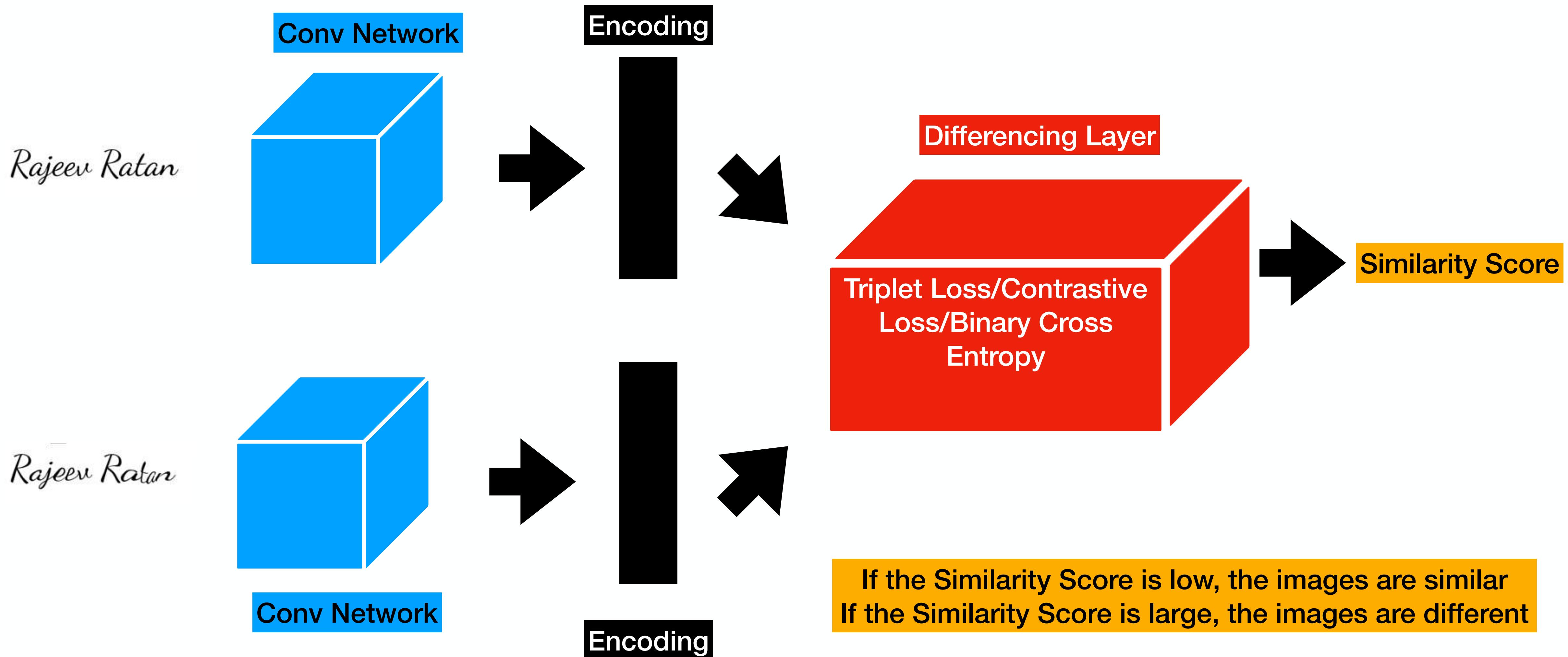
Rajeev Ratan      Rajeev Ratan

**Yes**

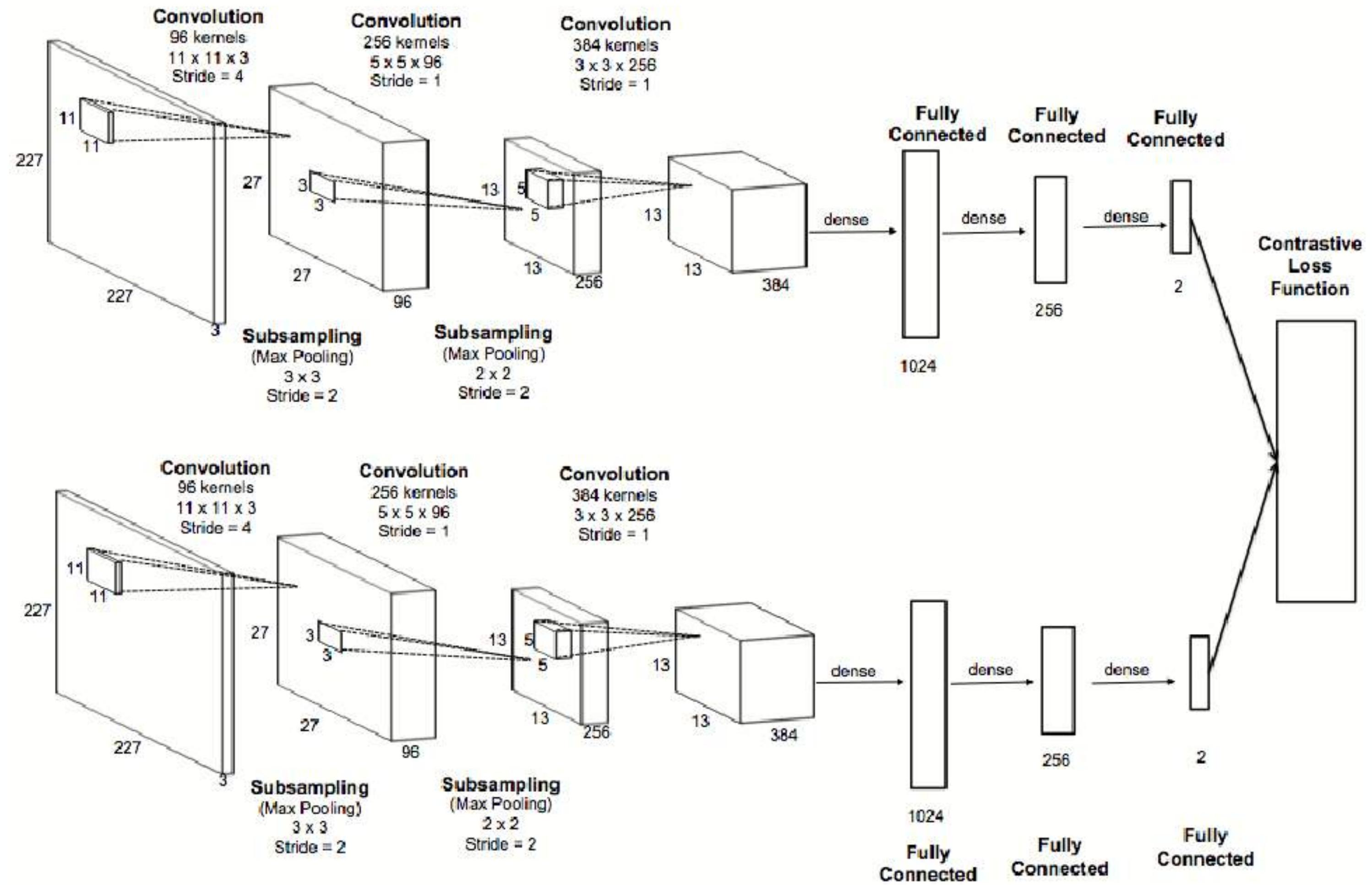
Rajeev Ratan      Rajeev Ratan

**No**

# High Level Diagram of a Siamese Network



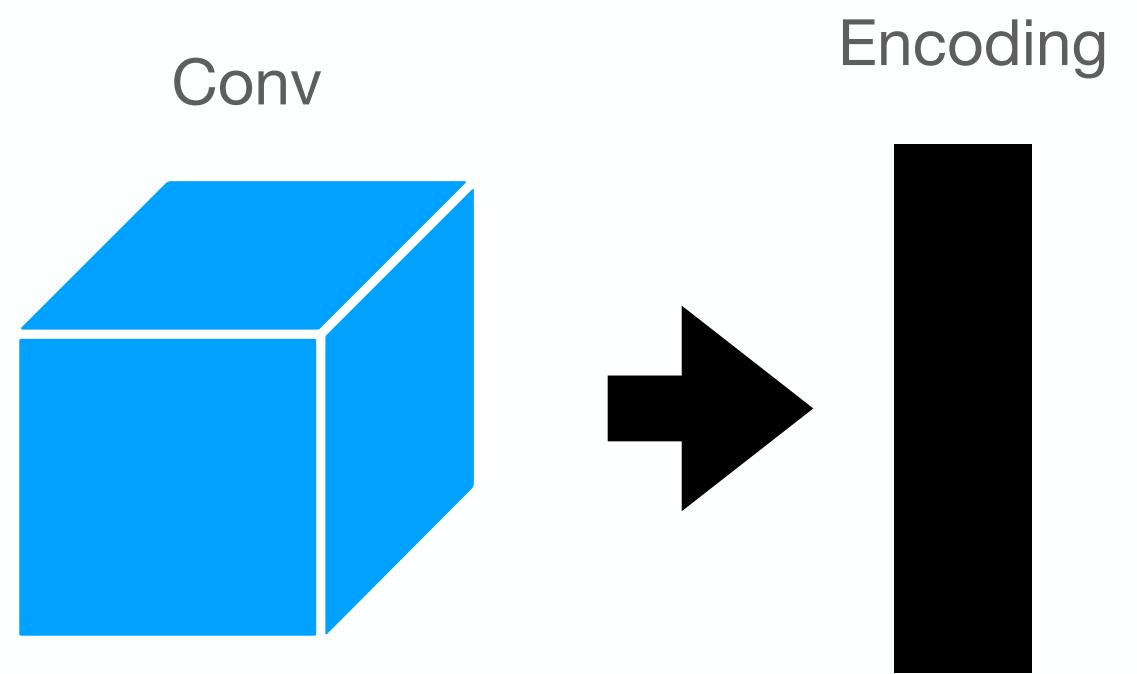
# High Level Diagram of a Siamese Network



Example for a siamese network (source: Rao et al. - <https://www.semanticscholar.org/paper/A-Deep-Siamese-Neural-Network-Learns-the-Similarity-Rao-Wang/3e16932979250e66cd2cb4d8c9a5411e195273be/figure/0>)

# Siamese Network Architecture

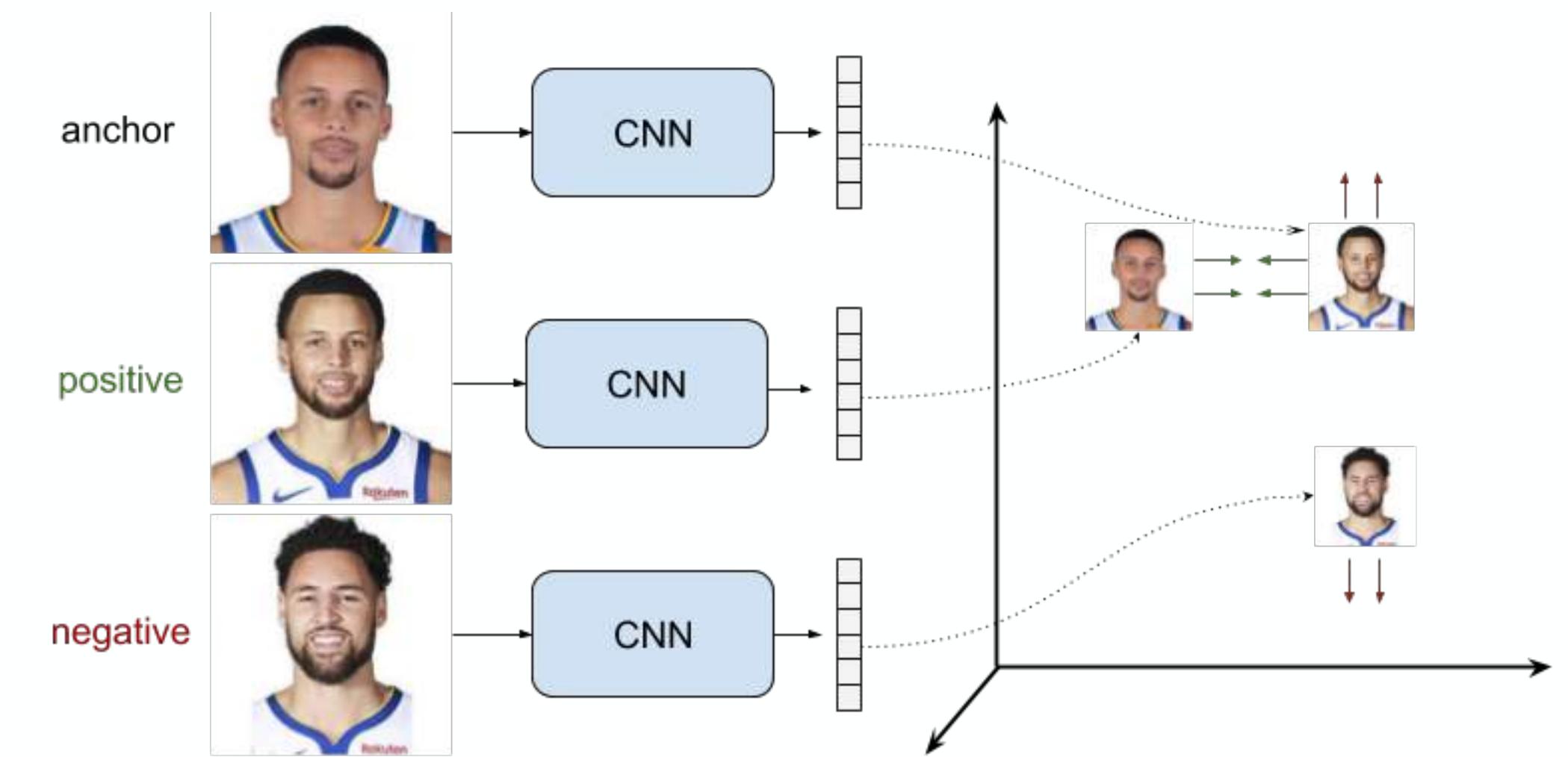
- Notice that the output of Conv Networks in a Siamese Networks are a **flat matrix**.
- This is the **embedding layer or encoding** which can be considered the features extracted from that image.
- The **Differencing Layer** is where we can find the difference using **Euclidian** or **Cosine Difference** between the matrices produced by each network.
- We can then use a **loss function** to assess whether the Siamese Networks made the right decision.



# Siamese Networks Loss Functions

Popular Loss functions used when training Siamese Networks are:

- **Triplet Loss** - where a baseline input is compared to a positive input and a negative input. The distance from the baseline input to the positive input is minimized, and the distance from the baseline input to the negative input is maximized
- Contrastive Loss
- Binary Cross-Entropy

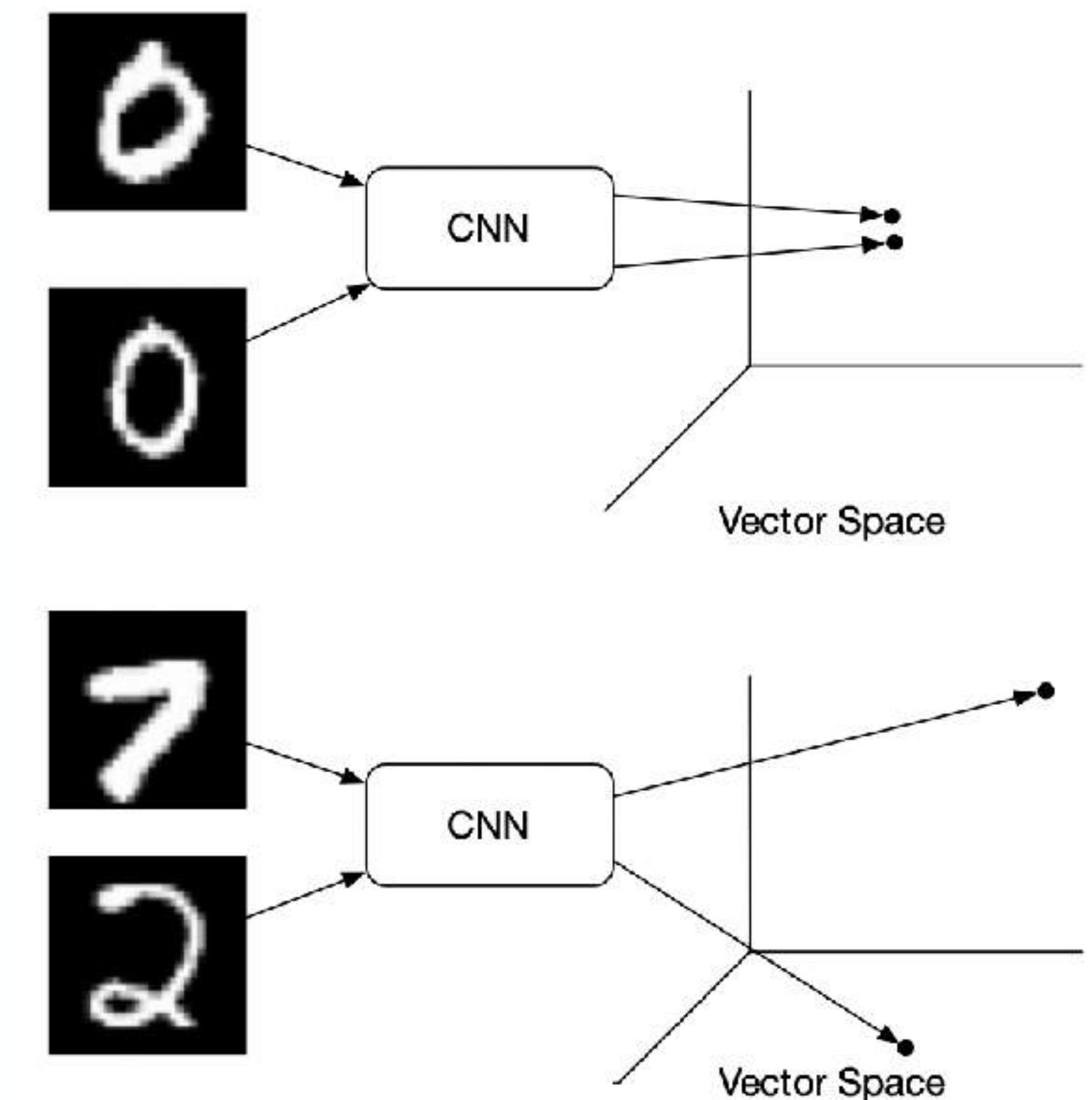


[https://commons.wikimedia.org/wiki/File:Triplet\\_loss.png](https://commons.wikimedia.org/wiki/File:Triplet_loss.png)

# Siamese Networks Loss Functions

Popular Loss functions used when training Siamese Networks are:

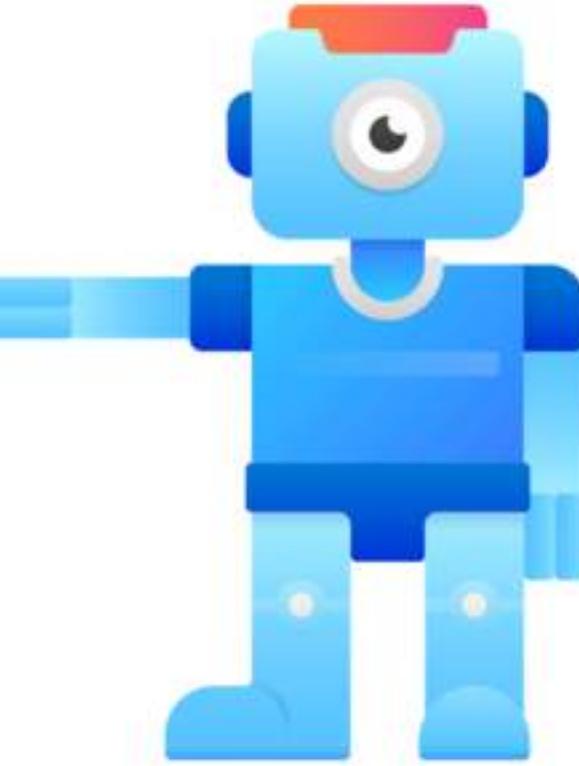
- Triplet Loss
- **Contrastive Loss** - The goal of a siamese networks is to differentiate between pairs of images. Contrastive refers to the fact that these losses are computed contrasting two or more data points representations.
- Binary Cross-Entropy



# Siamese Networks Loss Functions

Popular Loss functions used when training Siamese Networks are:

- Triplet Loss
- Contrastive Loss
- **Binary Cross-Entropy** - given the output of our Siamese Model is binary, i.e. similar or dis-similar, using binary cross-entropy loss function is often the obvious default choice.

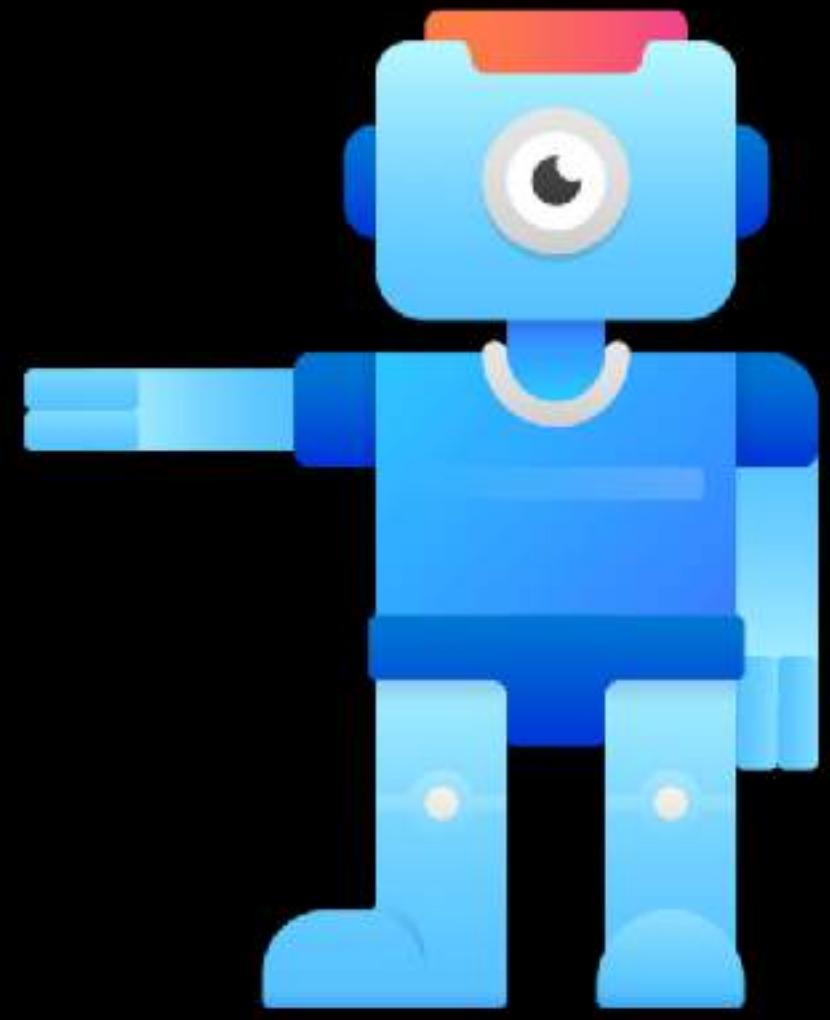


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Training Siamese Networks**



# MODERN COMPUTER VISION

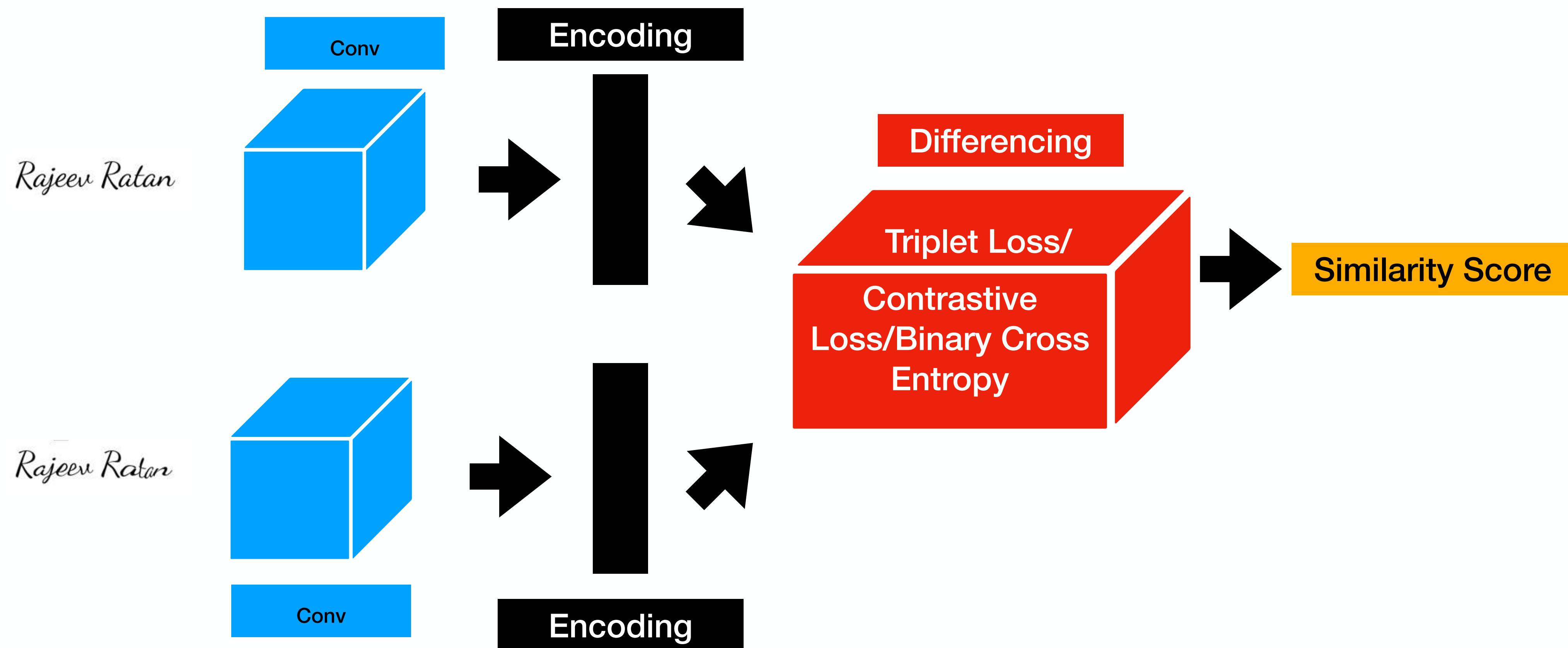
BY RAJEEV RATAN

## Training Siamese Networks

How do we go about Training Siamese Networks

# Siamese Networks

- Remember the goal of a Siamese Network is to classify if the two inputs are the same or different using the Similarity Score.



# Dataset - Image Pairs

- We start the training process by firstly preparing our data by creating **Image Pairs**.
- We create two types of pairs:
  - **Positive** data pair is when both the inputs are the same class
  - **Negative** pair is when the two inputs are difference classes



Positive Pair



Negative Pair

# Building the Network

- We build a CNN that outputs the **feature encoding or embedding** using a fully connected layer.
- **We build the sister** CNN's will have the same architecture, hyperparameters, and weights.
- We then build the **differencing layer to calculate the Euclidian distance** between the output of the two CNN subnetworks encoding.
- The final layer is a **fully-connected layer with a single node** using the sigmoid activation function to output the **Similarity score**.
- Compile the model using one of the loss functions (Contrastive or Triplet Loss work well).

# Ready to Train

- Once we have our image pairs dataset and our models built we can begin training.
- Typically we can use RMSprop or any common Gradient Descent Algorithm.
- CNNs are also typically simple as we only need to create relatively simple embeddings.



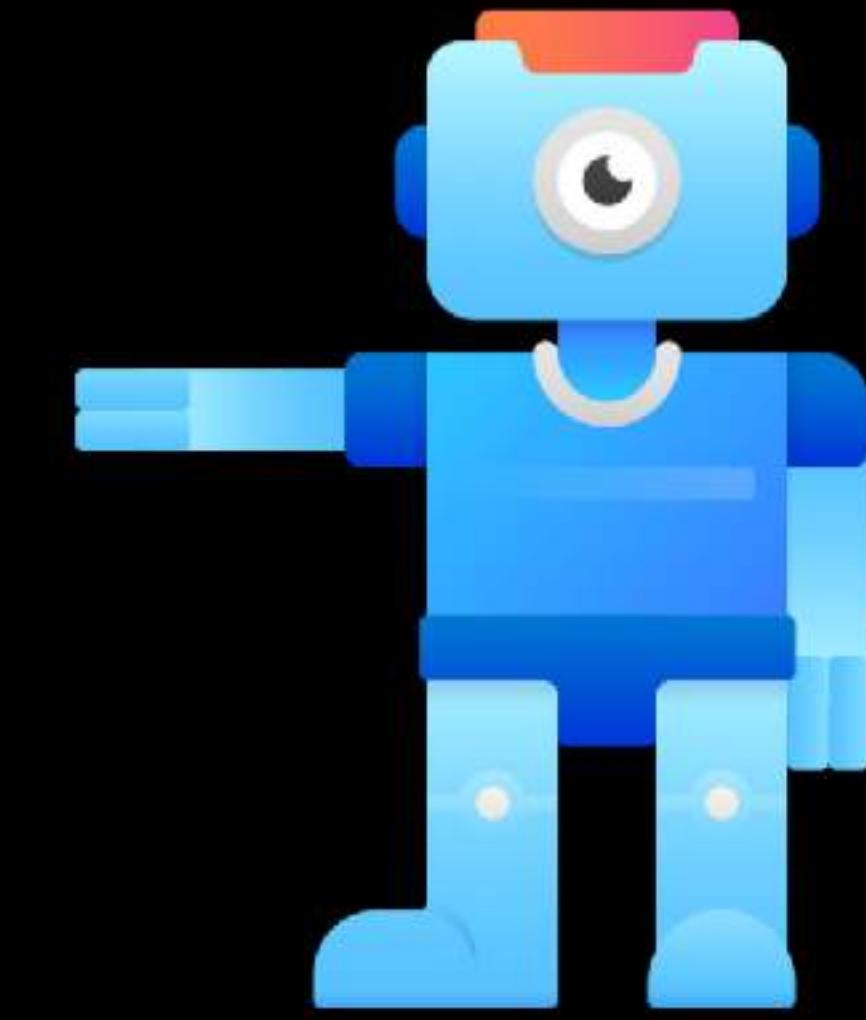


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Siamese Networks in Keras**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Facial Recognition

Overview and an intro to VGGFace and FaceNet

# What is Facial Recognition?

- The ability to automatically attach an individual's identity to a face
- It's a task that human abilities are absolutely brilliant, and even some animals, dogs, crows, sheep can do it!
- Can machines do it?



# Facial Recognition the Early Days

- **OpenCV** has 3 facial recognition libraries, all of which operate similarly.
- They take a dataset of labelled faces, and compute features to represent the images. Their classifiers then utilise these features to classify.
- **Eigenfaces (1987)** - `createEigenFaceRecognizer()`
- **Fisherfaces (1997)** - `createFisherFaceRecognizer()`
- **Local Binary Patterns Histograms (1996)**-  
`createLBPHFaceRecognizer()`

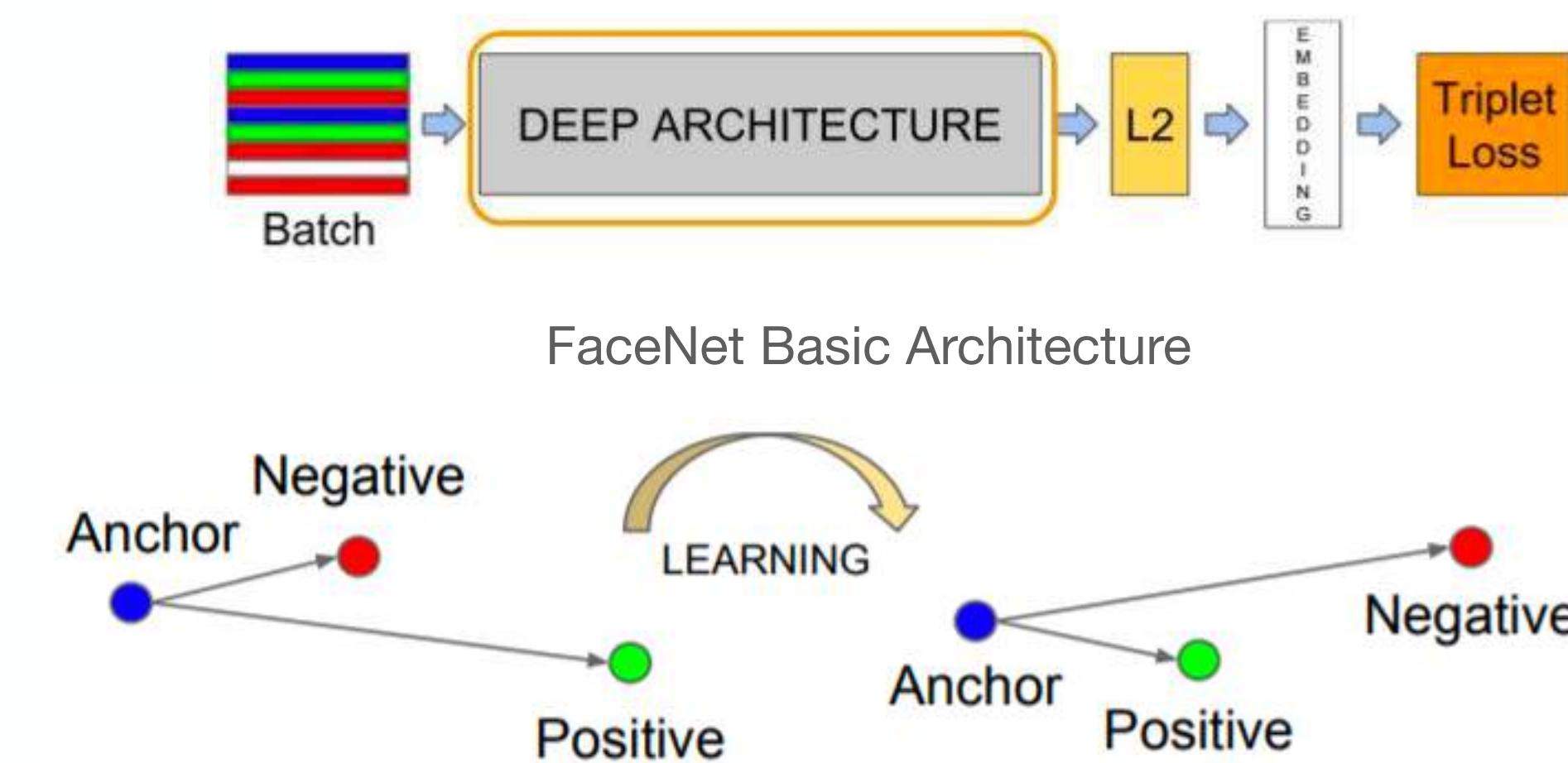
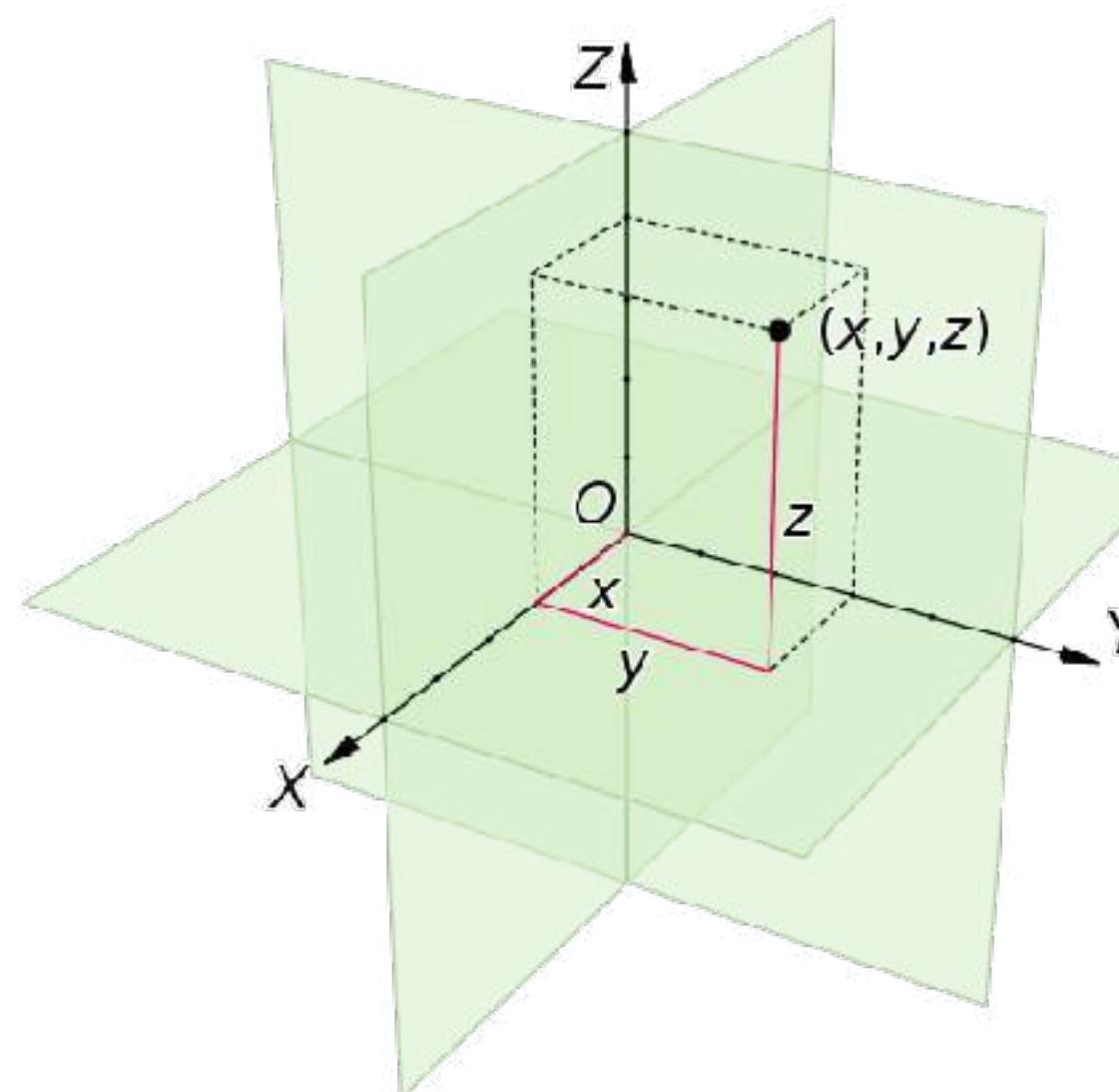


# Facial Recognition using Deep Learning

- Siamese Networks can be used for Facial Recognition.
- Let's take a look at two popular Deep Learning Facial Recognition Networks:
  - VGGFace
  - FaceNet

# A look at FaceNet

- FaceNet was first introduced by Google in 2015
- It transforms a face into a 128 dimension Euclidian space embedding
- Uses the triplet loss function

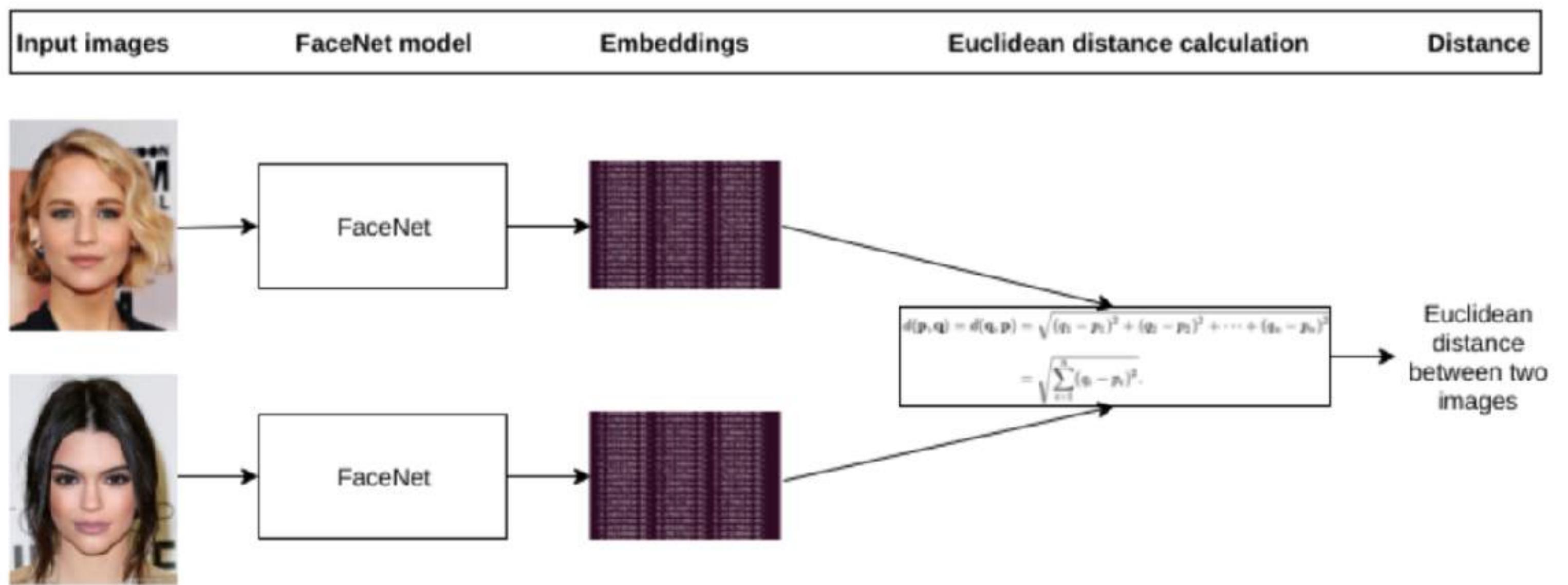


3 Dim in euclidian space

Triplet Loss Training

<https://arxiv.org/pdf/1503.03832.pdf>

# A look at FaceNet



One shot learning using FaceNet

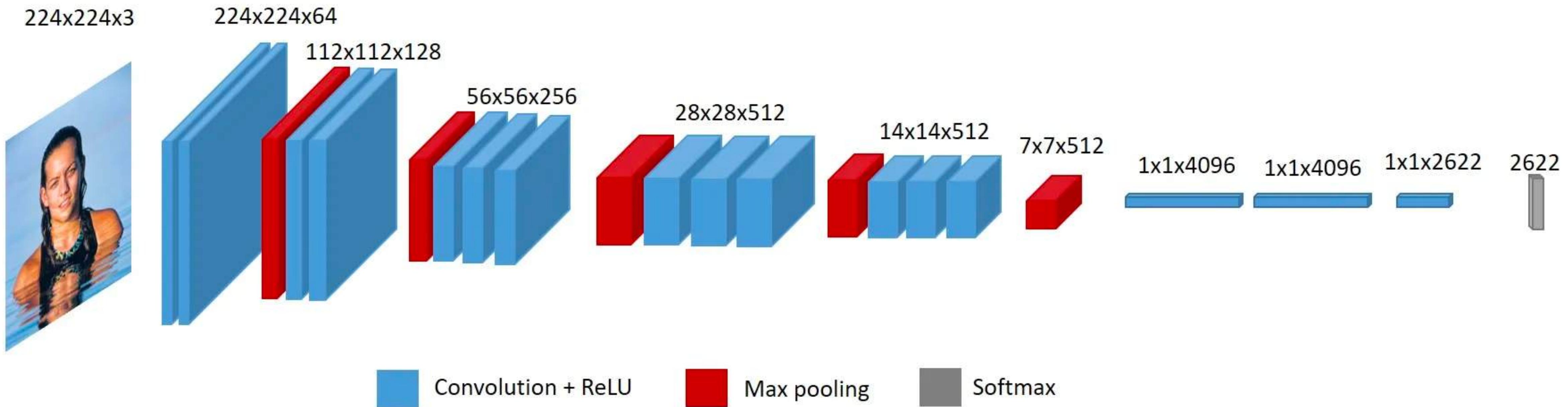
# A look at VGGFace

- VGGFace was introduced by Oxford University Researchers Omkar M. Parkhi, Andrea Vedaldi and Andrew Zisserman in their paper titled ‘Deep Face Recognition) in 2015
- They used Triplet Loss and an embedding vector of 2,622 or 1,024 (depending on the configuration) with input image size being 224 x 224



Figure 1: Example images from our dataset for six identities.

# A look at VGGFace



Dataset	Identities	Images
LFW	5,749	13,233
WDRef [4]	2,995	99,773
CelebFaces [25]	10,177	202,599

Dataset	Identities	Images
Ours	2,622	2.6M
FaceBook [29]	4,030	4.4M
Google [17]	8M	200M

Table 1: **Dataset comparisons:** Our dataset has the largest collection of face images outside industrial datasets by Goole, Facebook, or Baidu, which are not publicly available.

# VGGFace Performance

No.	Method	Images	Networks	Acc.
1	Fisher Vector Faces [21]	-	-	93.10
2	DeepFace [29]	4M	3	97.35
3	Fusion [30]	500M	5	98.37
4	DeepID-2,3		200	99.47
5	FaceNet [17]	200M	1	98.87
6	FaceNet [17] + Alignment	200M	1	99.63
7	Ours	2.6M	1	98.95

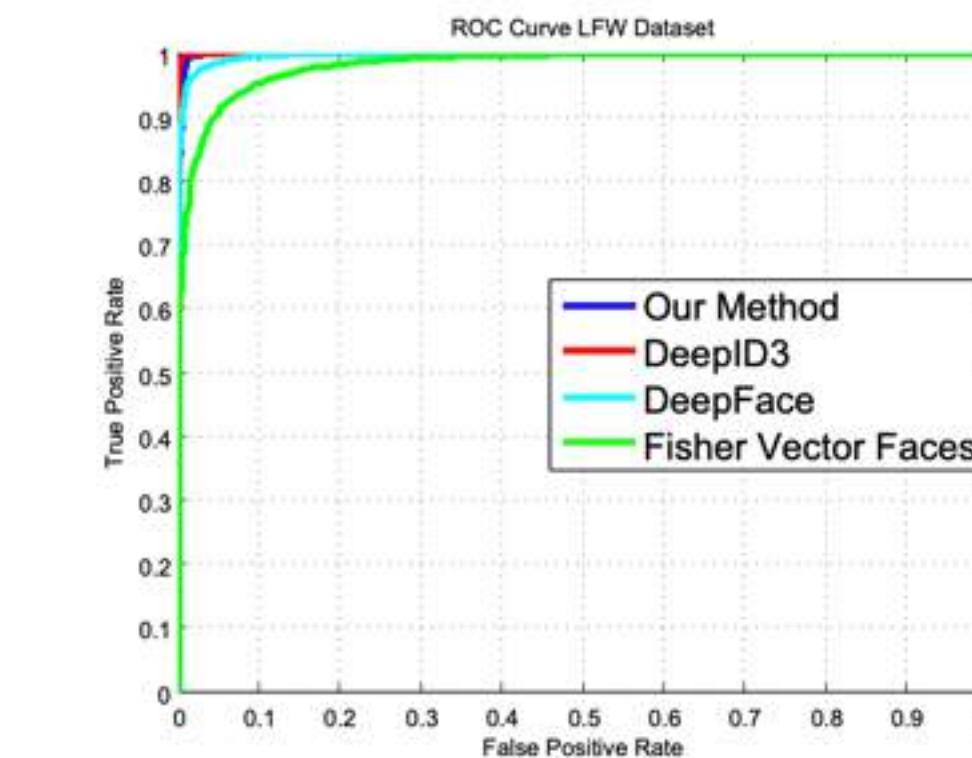


Table 5: **LFW unrestricted setting.** Left: we achieve comparable results to the state of the art whilst requiring less data (than DeepFace and FaceNet) and using a simpler network architecture (than DeepID-2,3). Note, DeepID3 results are for the test set with label errors corrected – which has not been done by any other method. Right: ROC curves.

LFW - Labelled Faces in the Wild

No.	Method	Images	Networks	100%- EER	Acc.
1	Video Fisher Vector Faces [15]	-	-	87.7	83.8
2	DeepFace [29]	4M	1	91.4	91.4
3	DeepID-2,2+,3		200	-	93.2
4	FaceNet [17] + Alignment	200M	1	-	95.1
5	Ours ( $K = 100$ )	2.6M	1	92.8	91.6
6	Ours ( $K = 100$ ) + Embedding learning	2.6M	1	97.4	97.3

Table 6: **Results on the Youtube Faces Dataset, unrestricted setting.** The value of  $K$  indicates the number of faces used to represent each video.



**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Face Similarity, Recognition, VGG Face and FaceNet**

# Object Detection

## The principles of Object Detection

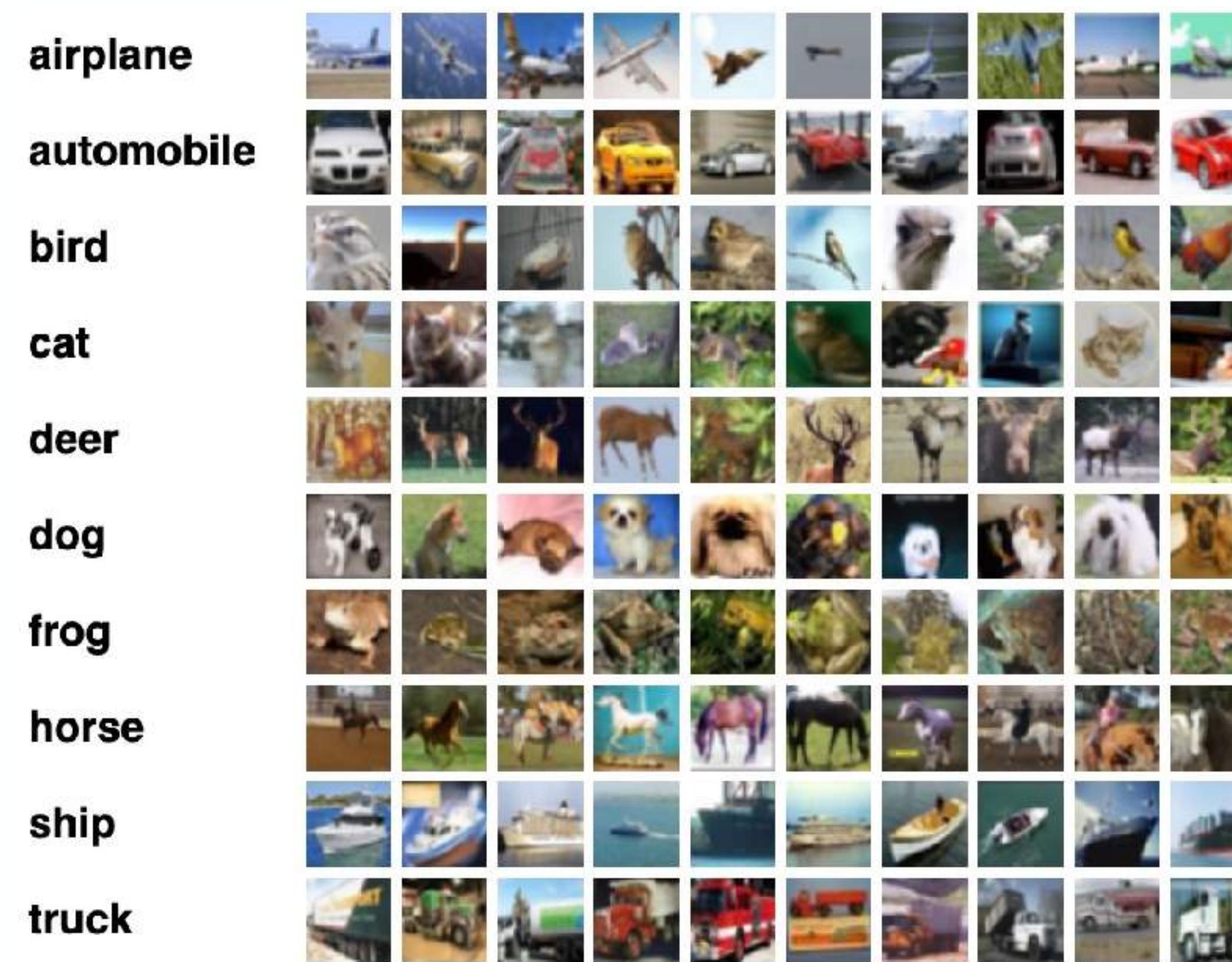


# MODERN COMPUTER VISION

BY RAJEEV RATAN

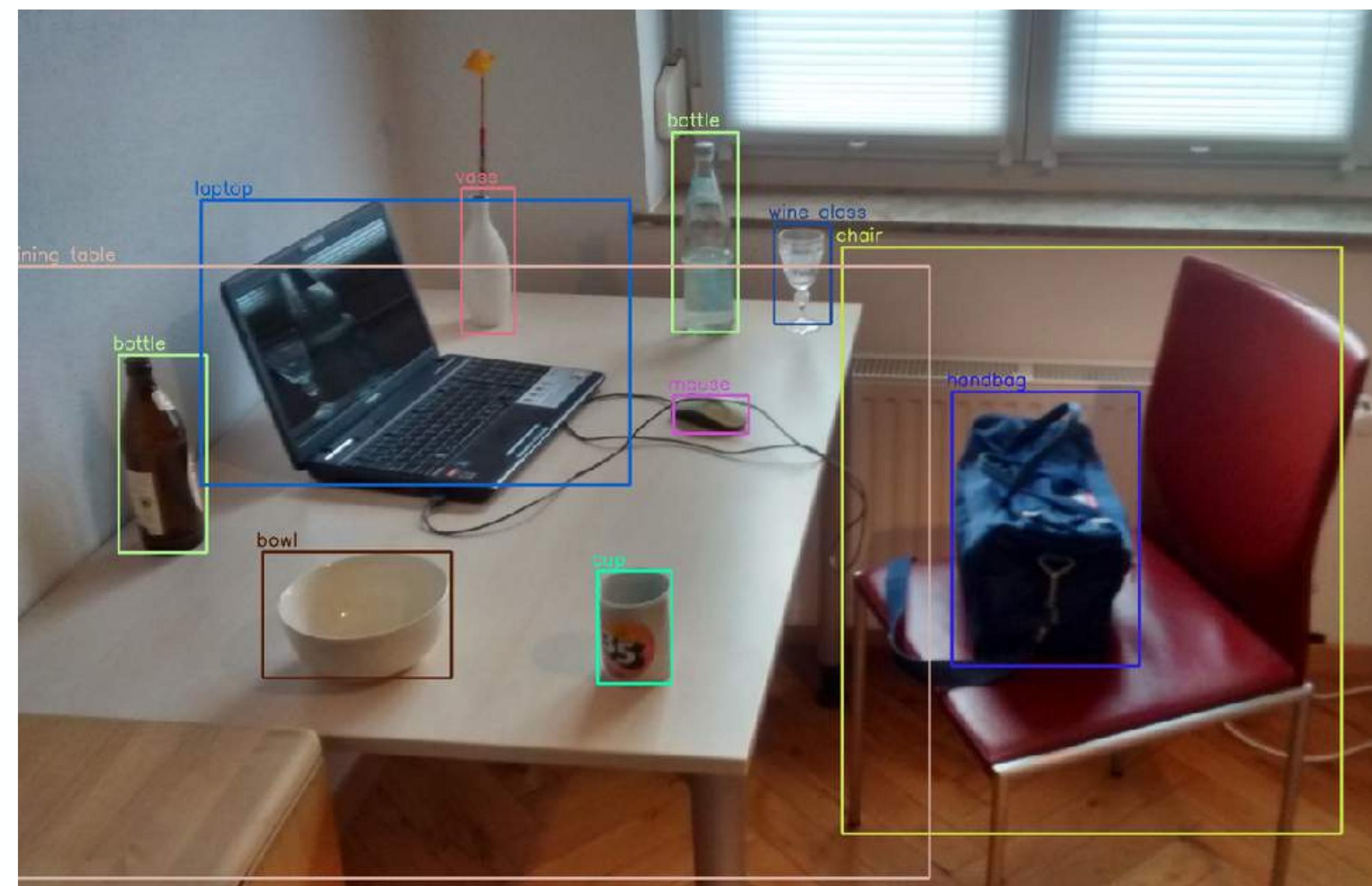
# What is Object Detection?

- So far we've extensively dealt with image classification i.e. determining as a whole, what the image contains



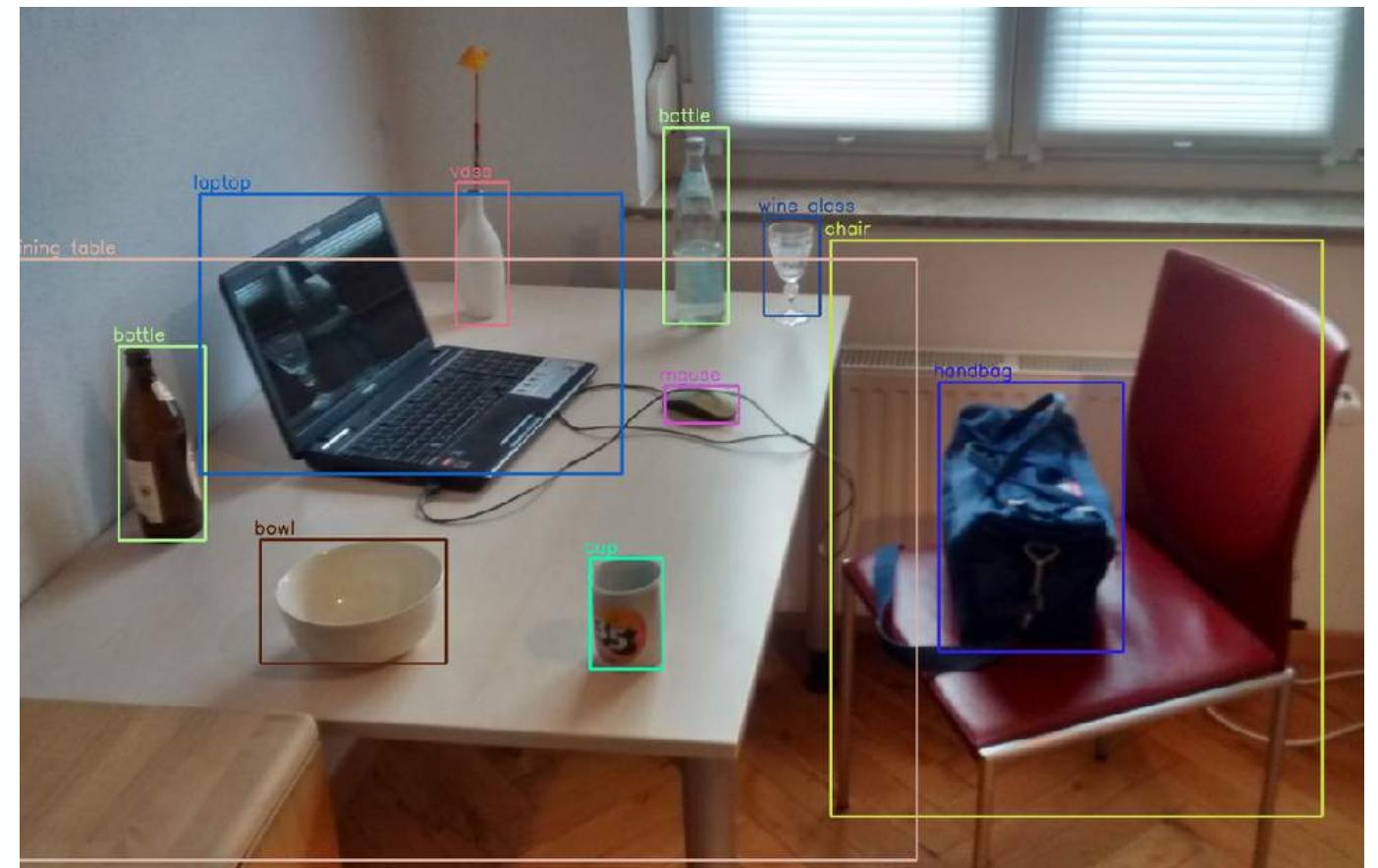
# What is Object Detection?

- But what if we wanted to individually classify the objects in an image?



# Why is Object Detection Hard?

- Object Detection algorithms allows us to do this!
- However, as you may rightfully infer, this presents a much more challenging problem.
- We need to do both:
  1. Determine where objects of interest lie - **Localisation**
  2. Determine what the object within that region is - **Object Classification**



# Object Detectors

- Object Detectors are usually trained on one or more classes and seek to produce a bounding box with object class identified.
- There are many types from non deep learning based such as Haar Cascade Classifiers, Sliding Windows with HoGs
- To Deep Learning Models such as R-CNNs, SSDs, YOLO, Detectron, EfficientDet and RetinaNet
- In Deep Learning methods we have two main types:
  - **Two-Shot** - uses two stages, a region proposal and then classification
  - **Single Shot** - does both localisation and image classification at once

# Formal Wikipedia Definition

- **Object detection** is a computer technology related to [computer vision](#) and [image processing](#) that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos.
- Well-researched domains of object detection include [face detection](#) and [pedestrian detection](#). Object detection has applications in many areas of computer vision, including [image retrieval](#) and [video surveillance](#).

# Object Segmentation

- Related to object detection, but relies on sometimes very different algorithms is **segmentation**.
- Segmentation involves **pixel level classification** of different object classes.



# Classification vs Detection vs Segmentation

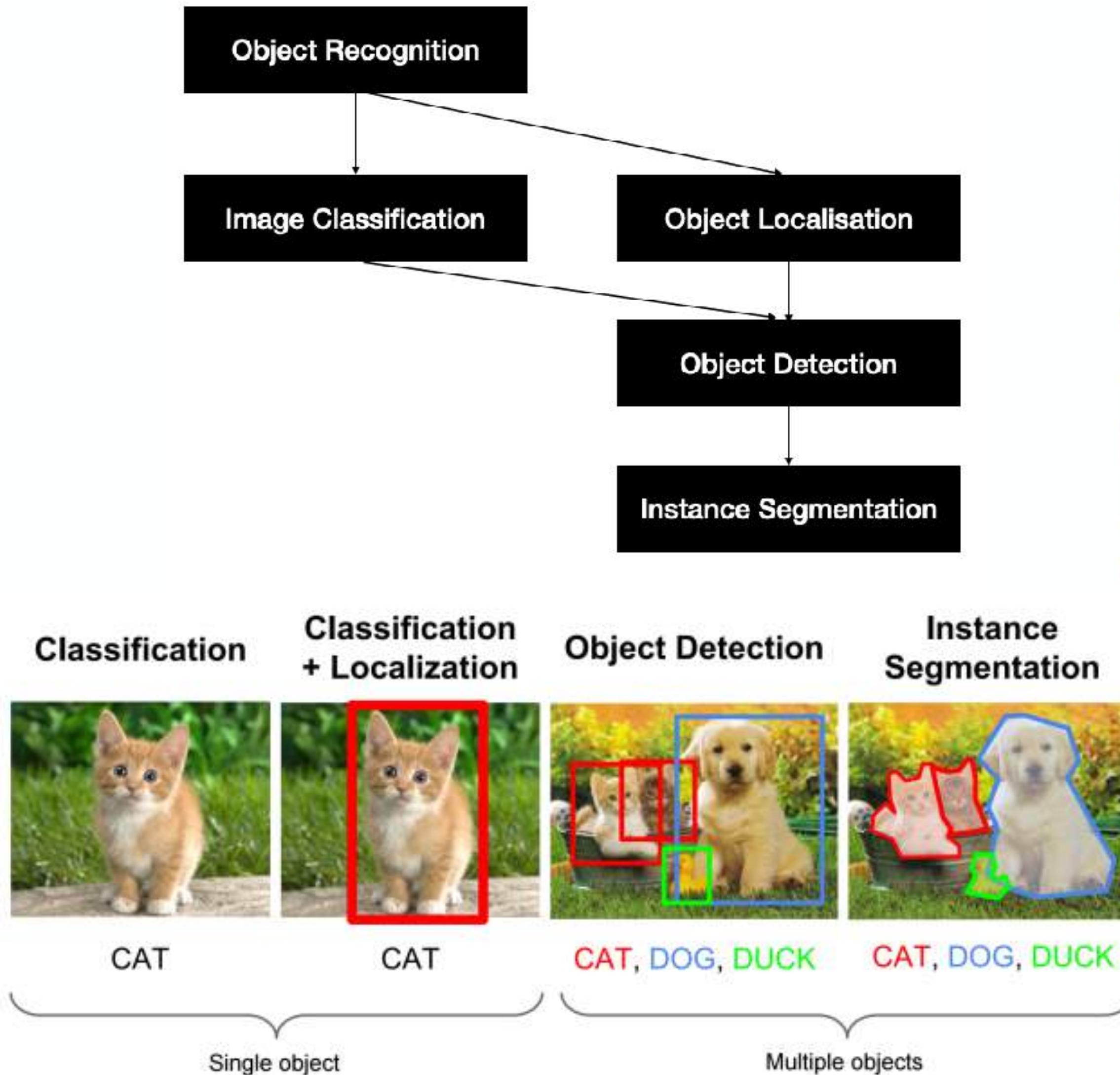
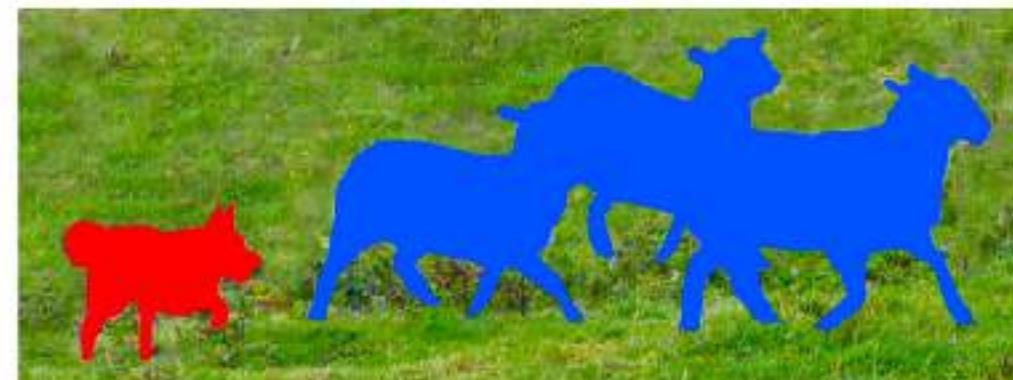
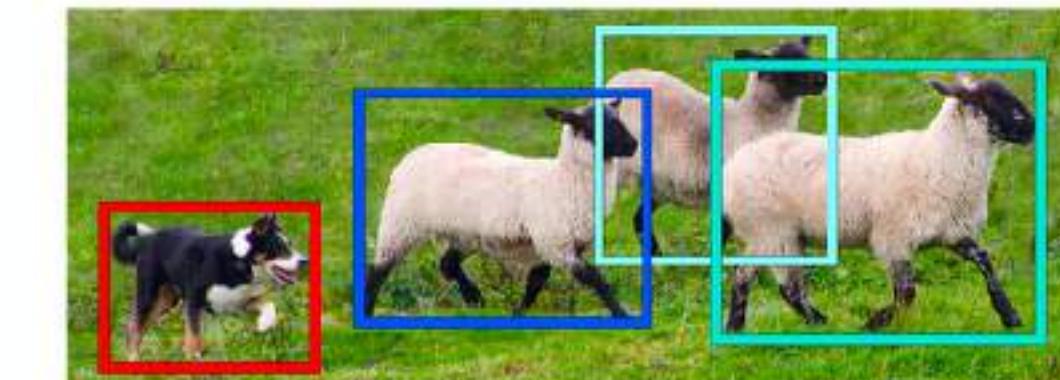


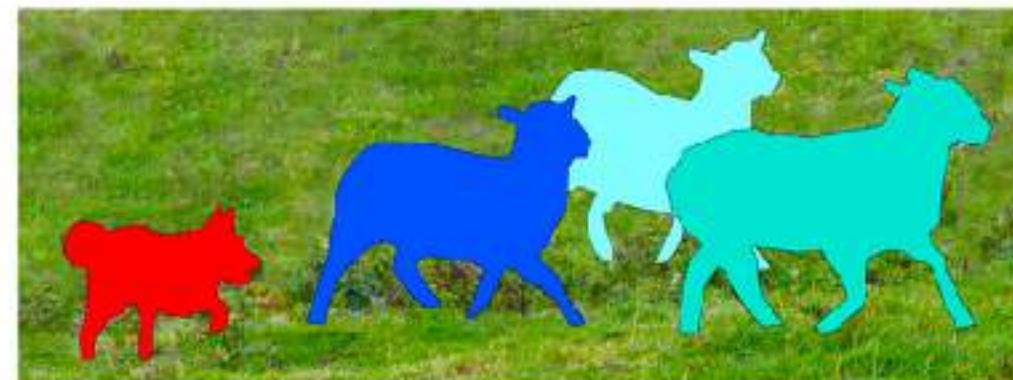
Image Recognition



Semantic Segmentation



Object Detection



Instance Segmentation

# Object Detection Use Cases

Electric Scooter ID

## Gas Leak Detection

Document Digitization

Plant Phenotyping

Flare Stack Monitoring

Resume Parsing

Augmented Reality

## Weed Detection

Microscopy

Bean Counting

Garbage Cleanup

Drone Video Analysis

## Conveyer Belt Debris

Traffic Counter

Pothole Identification

Soccer Player Tracker

Steelyard Throughput

Security Cam Analysis

Self Driving Cars

## Fish Measuring

Remote Tech Support

Tennis Line Tracking

Know Your Customer

Endangered Species Tracking

Inventory Management

## Hard Hat Detection

Pest Identification

OCR Math

Basketball Shot Tracking

Logo Identification

## Satellite Imagery

Traffic Cone Finder

## Airplane Maintenance

Tumor Detection

D&D Dice Counter

Plant Disease Finder

X-Ray Analysis

## Roof Damage Estimator

City Bus Tracking

Board Game Helpers

Dental Cavity Detection

Drought Tracking

Hog Confinements

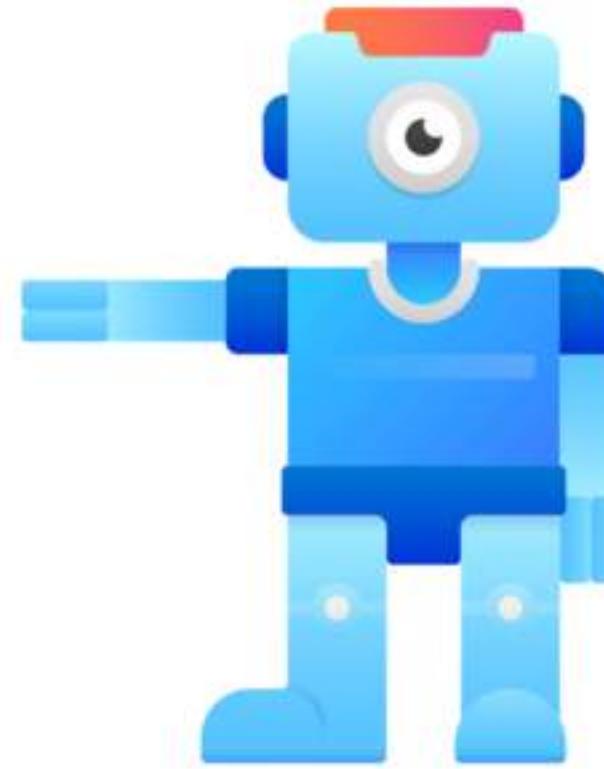
## Sushi Identifier

Oil Storage Estimator

Car Wheel Finder

License Plate Reader

Exercise Counter

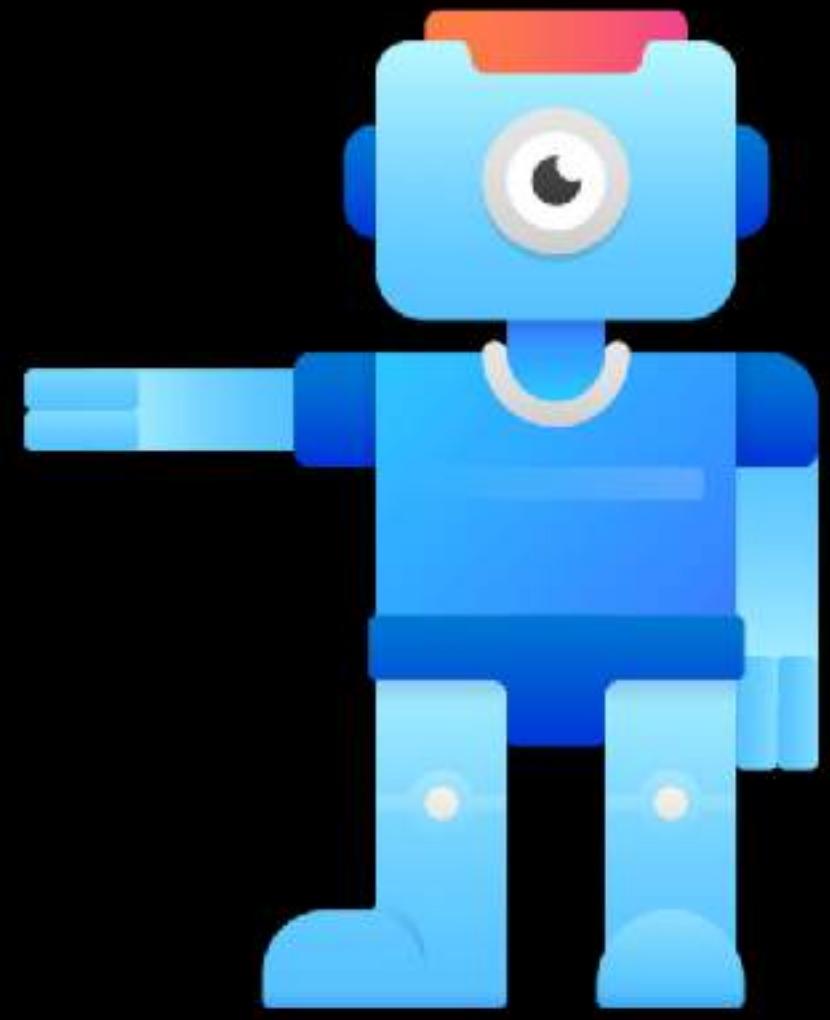


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Early Object Detectors - Haar Cascade Classifiers and Sliding Window with HoGs**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Early Object Detectors - Haar Cascade Classifiers and Sliding Window with HoGs

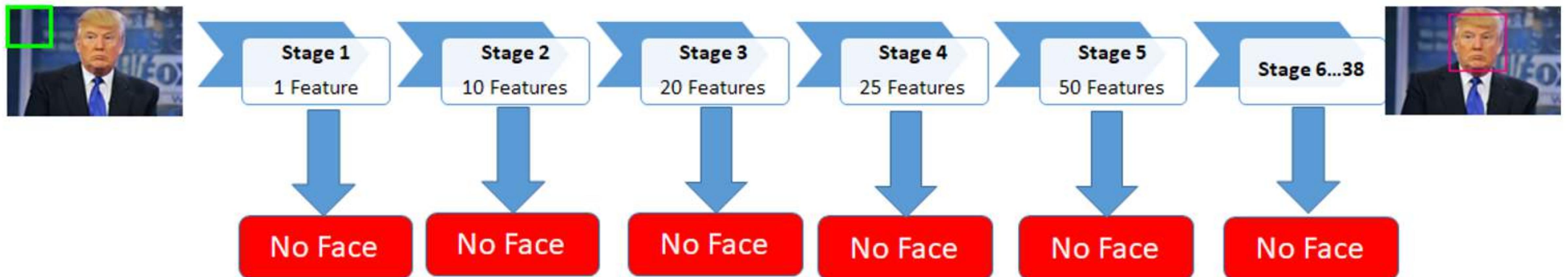
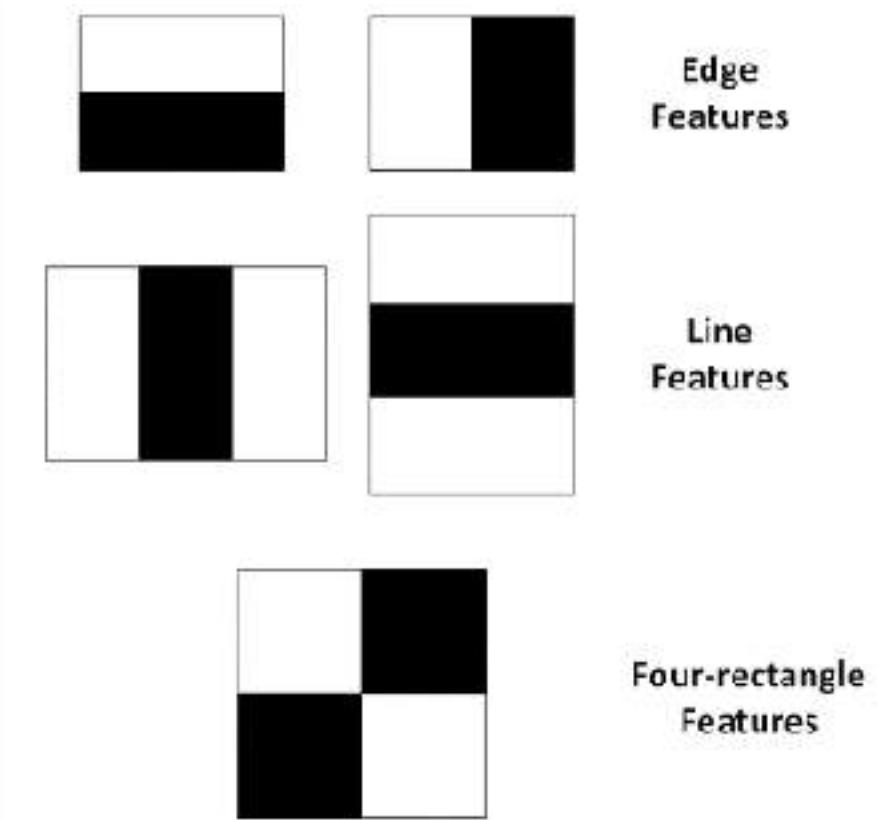
[History and overview of the first Object Detection Algorithms](#)

# Haar Cascade Classifiers

- Object Detection algorithms have been around since the early days of Computer Vision (first good attempts were in the 1990s).
- However, the first really successful implementation of Object Detection was in 2001 with the introduction of **Haar Cascade Classifiers** used by **Viola-Jones** in their **Face Detection** algorithm. It was very fast and easy to use detector, with good enough but not great accuracy.
- It takes Haar features as inputs into a series of cascaded classifiers and uses a series of steps to determine if a face has been detected.
- Haar cascades are very effective but difficult to develop, train and optimise.

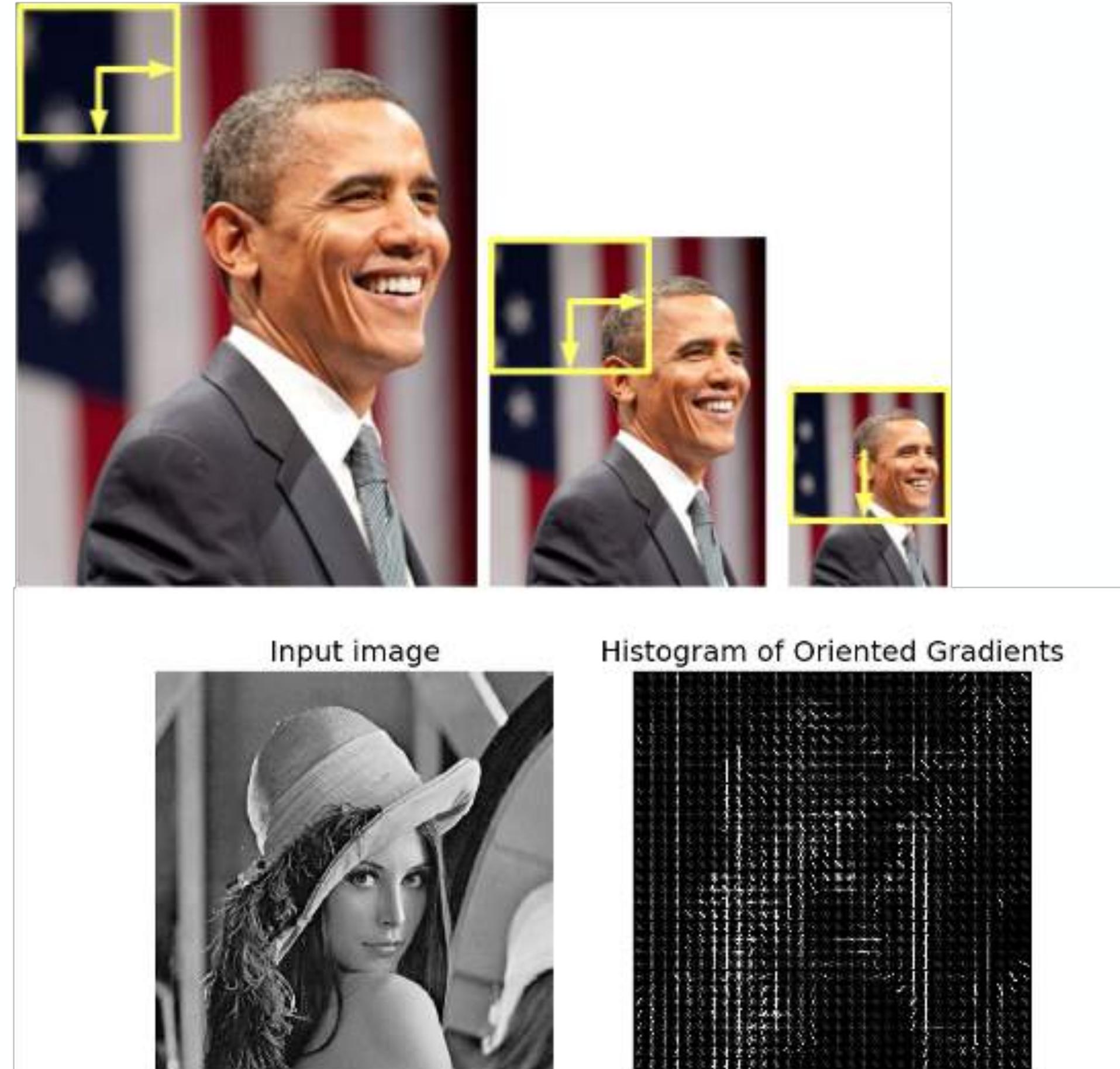
# Haar Cascade Classifiers

- Haar Cascade Classifiers, are trained using positive (with faces) and negative images.
- Features are extracted using rectangular blocks of various shapes. A process called Integral Images was used to speed up the process. They then used Ada Boost to improve performance and ensure good accuracy with reduced features. The Viola Jones application of this method uses 38 stages of various features to detect faces.



# Histogram of Gradients with SVM Sliding Windows

- **Sliding Windows** is a method where we extract segments of our image, piece by piece in the form of a rectangle extractor window.
- In each window we compute the **Histogram of Gradients (HoGs)** and compute how closely it matches the object we are trying to detect.
- **Support Vector Machines (SVMs)** are typically used to classify our HoG features.



# Previous Object Detection Methods are Simply Manual Feature Extraction combined with Sliding Windows

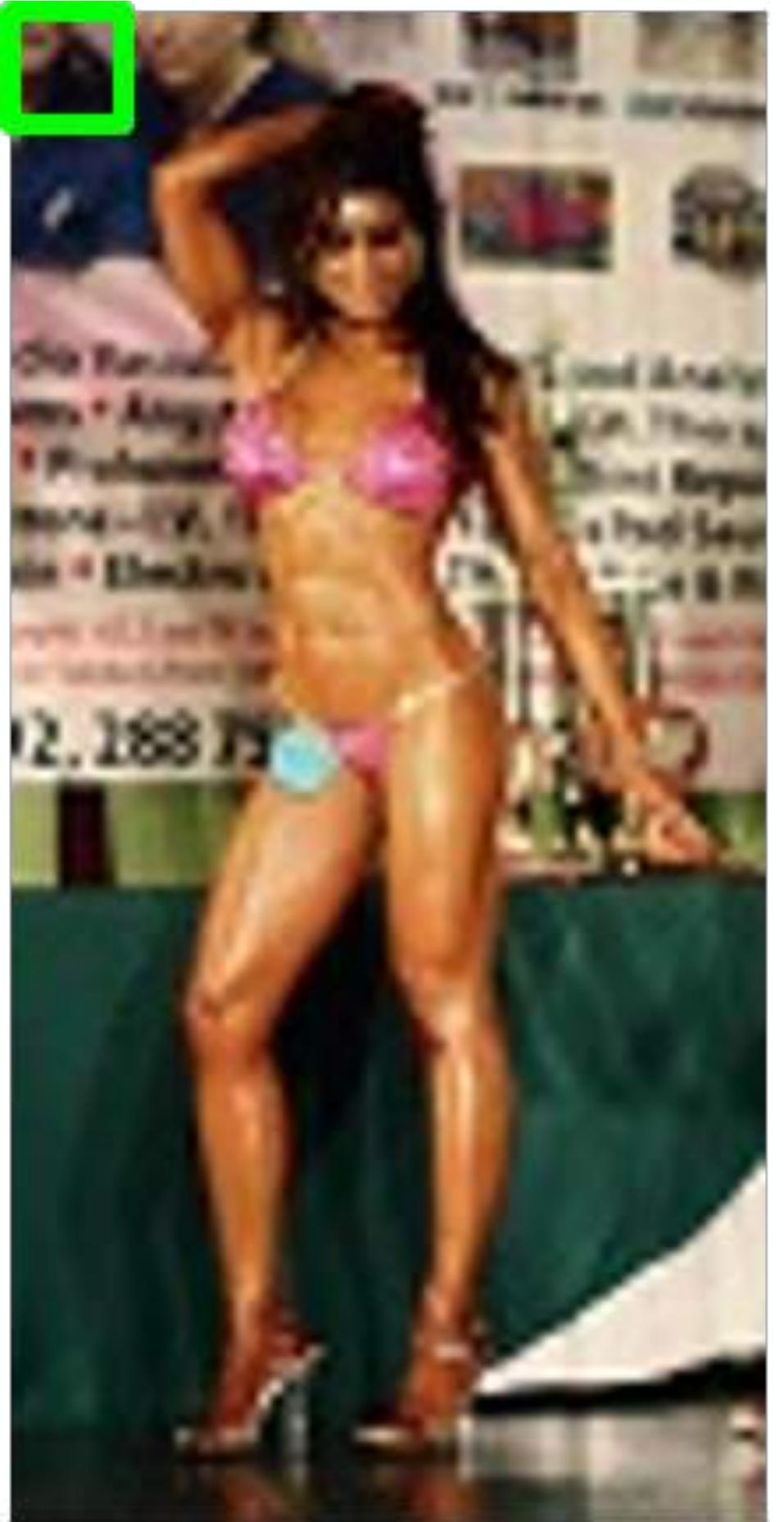
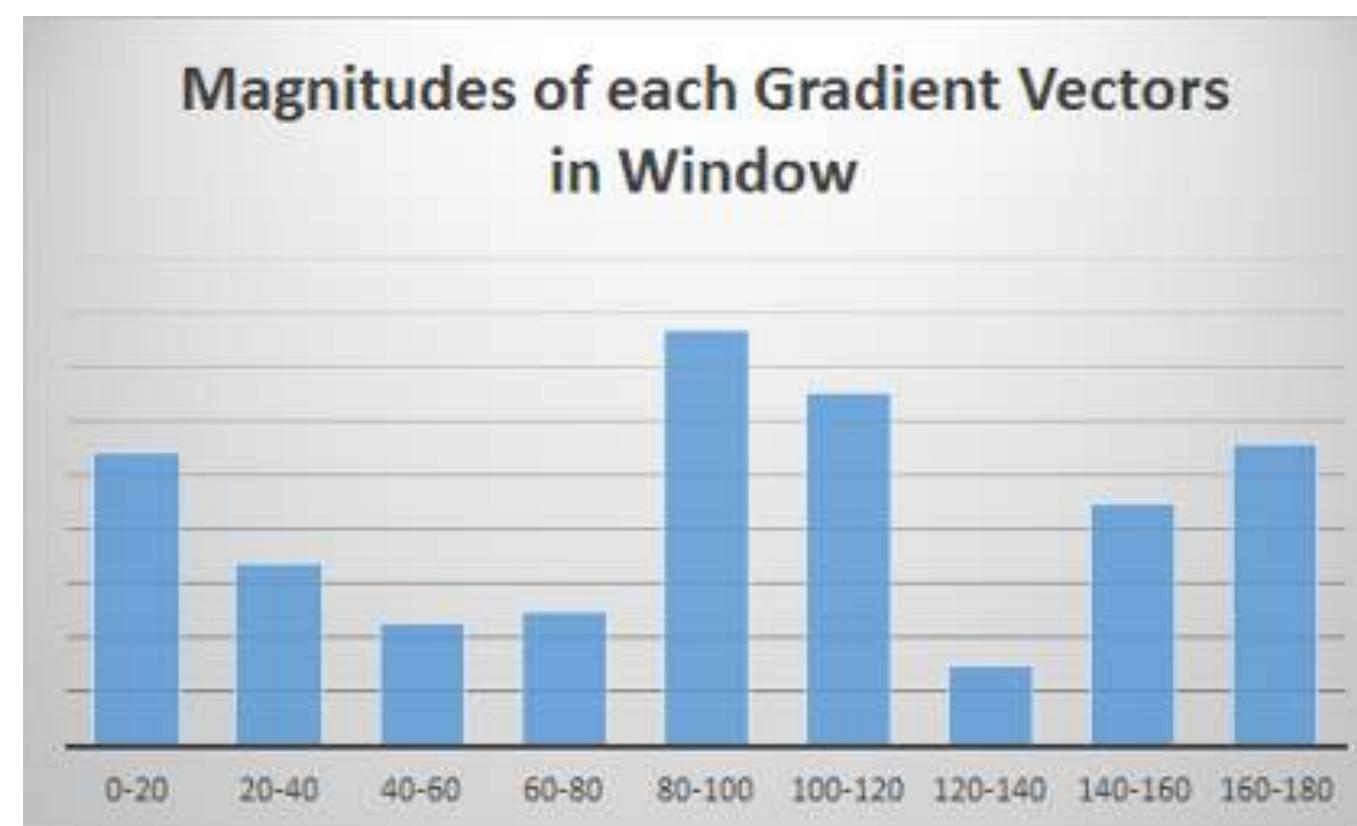
- Previous methods such as Haar Cascade Classifiers and HoG + SVM are basically manual **Feature Extraction Methods** combined with the use of a **Sliding Window**.
- We can eliminate some complexity by using a sliding window method with a CNN. However, the following problems still occur.
- Using a sliding window across an image results in hundreds even thousands of classifications done for a single image.
- Scaling issues, what size window do we use? Is pyramiding robust enough?
- What if the window isn't defined as we set it out to be. Do we use windows of different ratios? You can easily see how this can blow up into millions of classifications needed for a large image.



# How do you Slide?

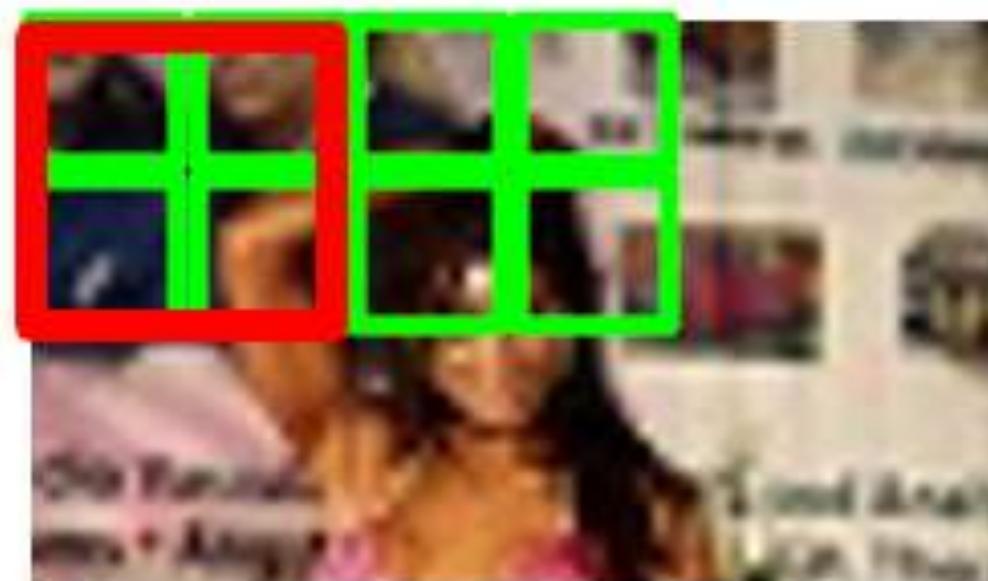
# How do we use HoGs with Sliding Windows?

- Using an  **$8 \times 8$**  pixel detection window or cell (in **green**), we compute the gradient vector or edge orientations at each pixel.
- This generates **64 ( $8 \times 8$ ) gradient vectors** which are then represented as a histogram.
- Each cell is then split into angular bins, where each bin corresponds to a gradient direction (e.g. x, y). In the Dalal and Triggs paper, they used 9 bins  $0-180^\circ$  ( $20^\circ$  each bin).
- This effectively reduces 64 vectors to just 9 values.
- As it stores gradients magnitudes, it's relatively immune to deformations.

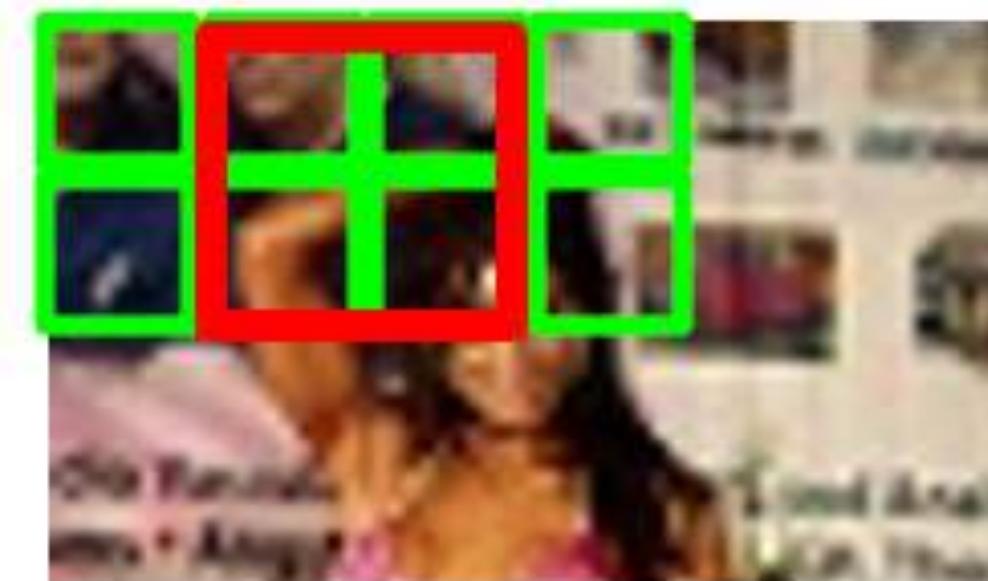


# How do we use HoGs with Sliding Windows?

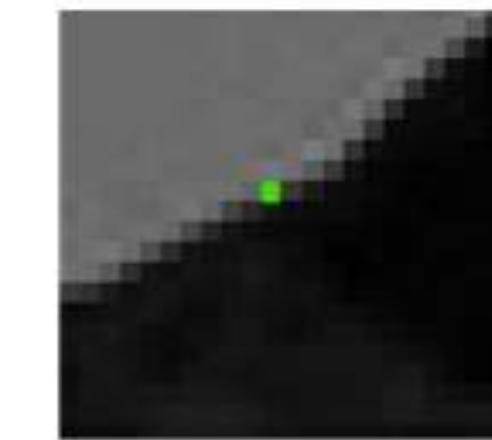
- We then **Normalise** the gradients to ensure **invariance** to **illumination** changes i.e. Brightness and Contrast. E.g. in the images on the right, if we divide the vectors by the gradient magnitudes we get 0.707 for all, this is normalisation.
- Instead of individual window cell normalisation, a method called **Block Normalisation** is used. This takes into account neighbouring blocks so we normalise taking into consideration larger segments of the image.



Block 1

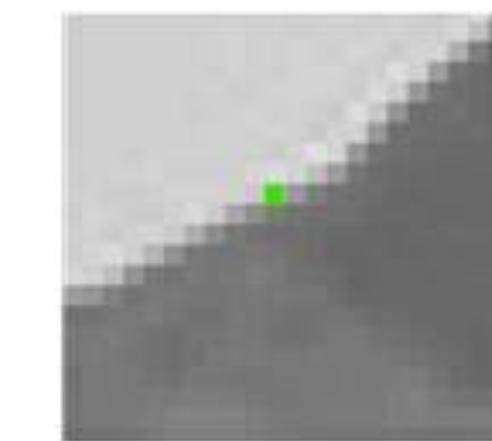


Block 2



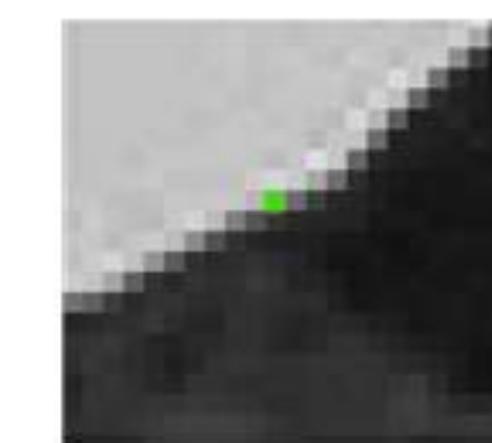
101		
91		51
	61	

$$\Delta_H = 50 \\ \Delta_V = 50 \\ |\Delta| = \sqrt{50^2 + 50^2} = 70.72$$



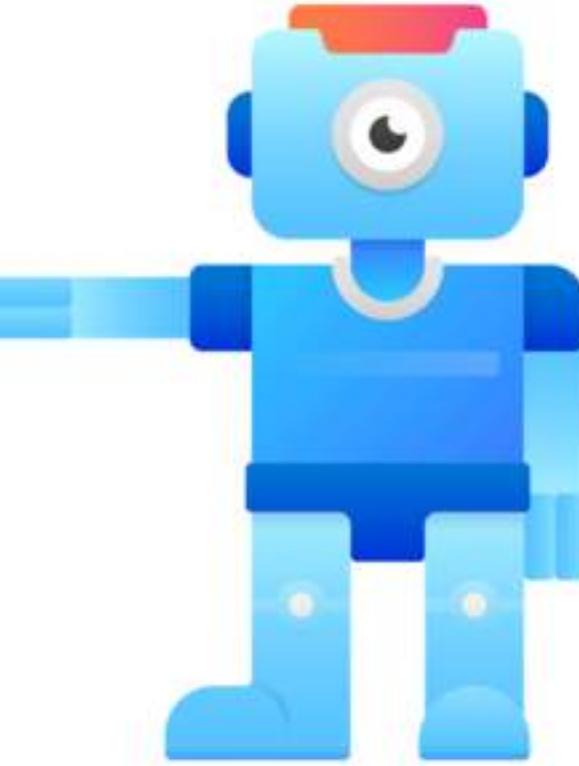
171		
161		121
	121	

$$\Delta_H = 50 \\ \Delta_V = 50 \\ |\Delta| = \sqrt{50^2 + 50^2} = 70.72$$



181		
171		71
	81	

$$\Delta_H = 100 \\ \Delta_V = 100 \\ |\Delta| = \sqrt{100^2 + 100^2} = 141.42$$

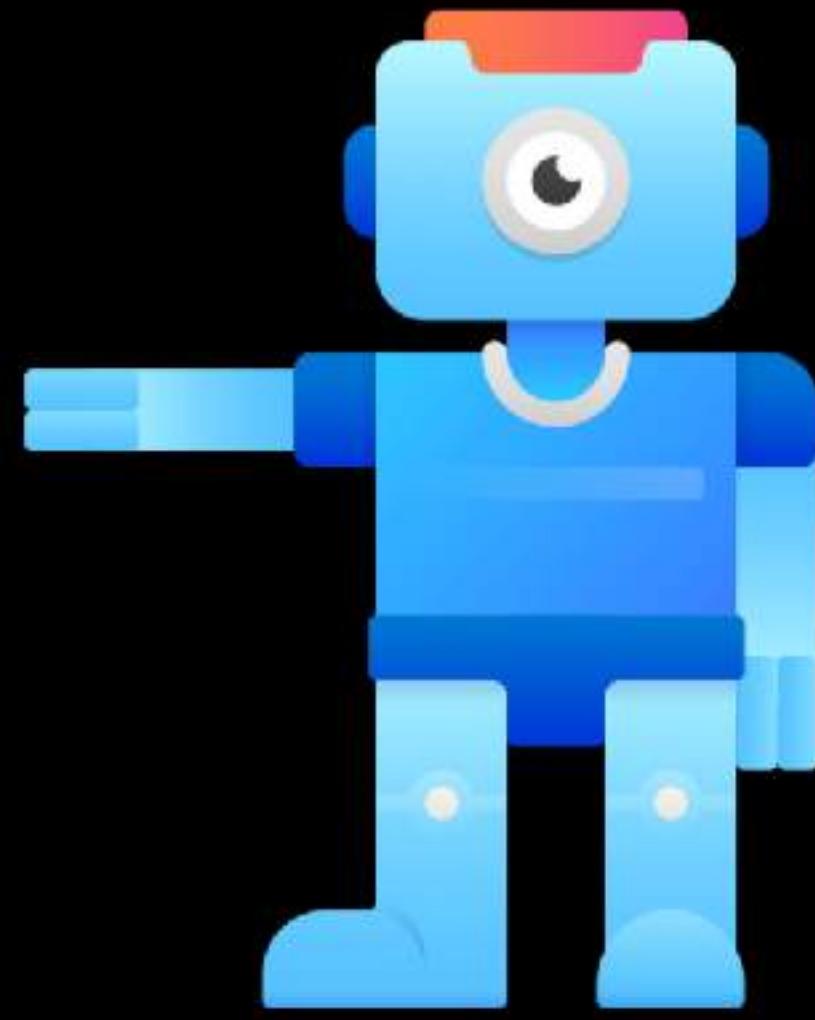


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Intersection Over Union - Assessing Object Detector Performance**



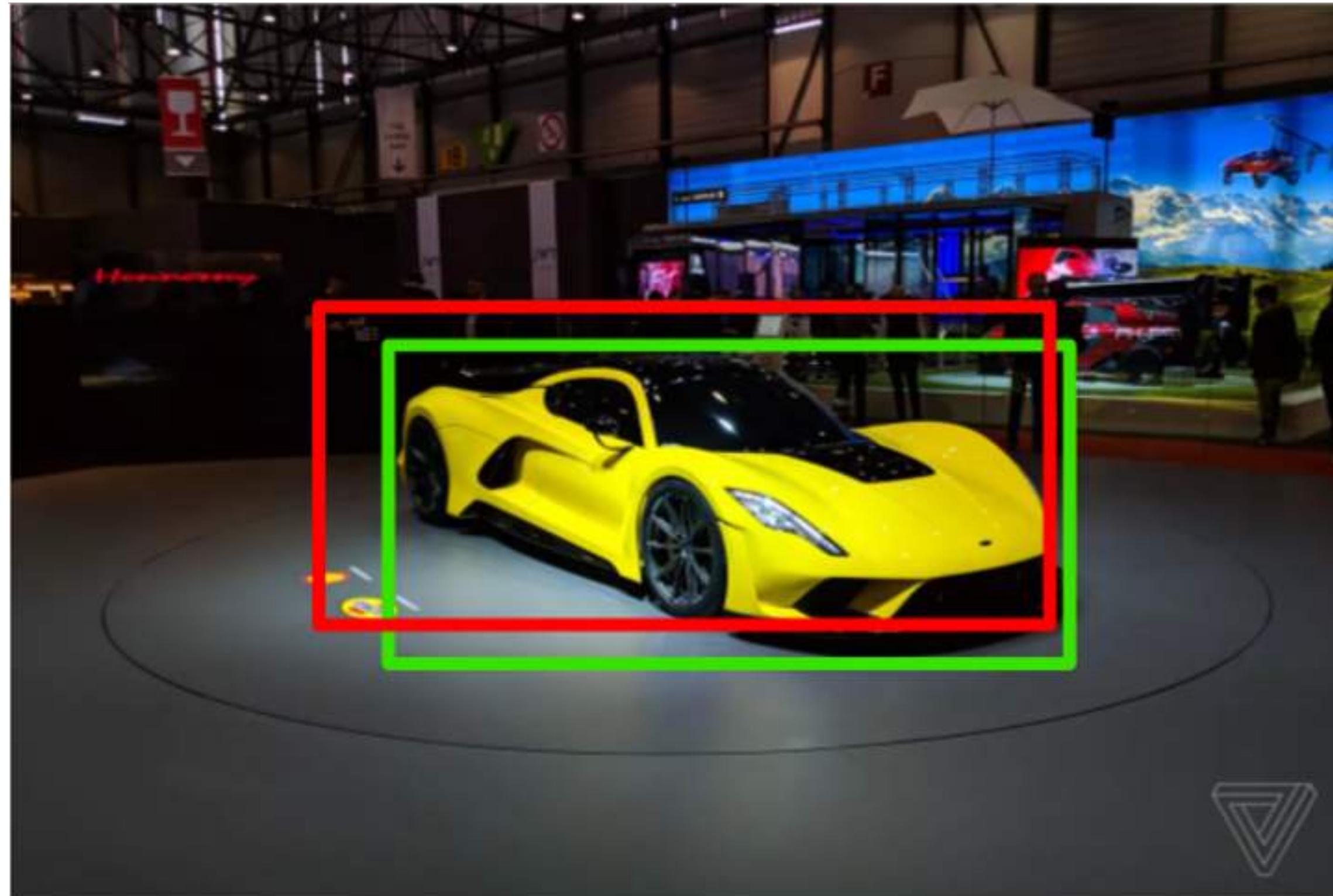
# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Intersection Over Union - Assessing Object Detector Performance

How we use Intersection over Union to assess the Performance of Object Detector Bounding Boxes

# Assessing Bounding Boxes from Object Detectors



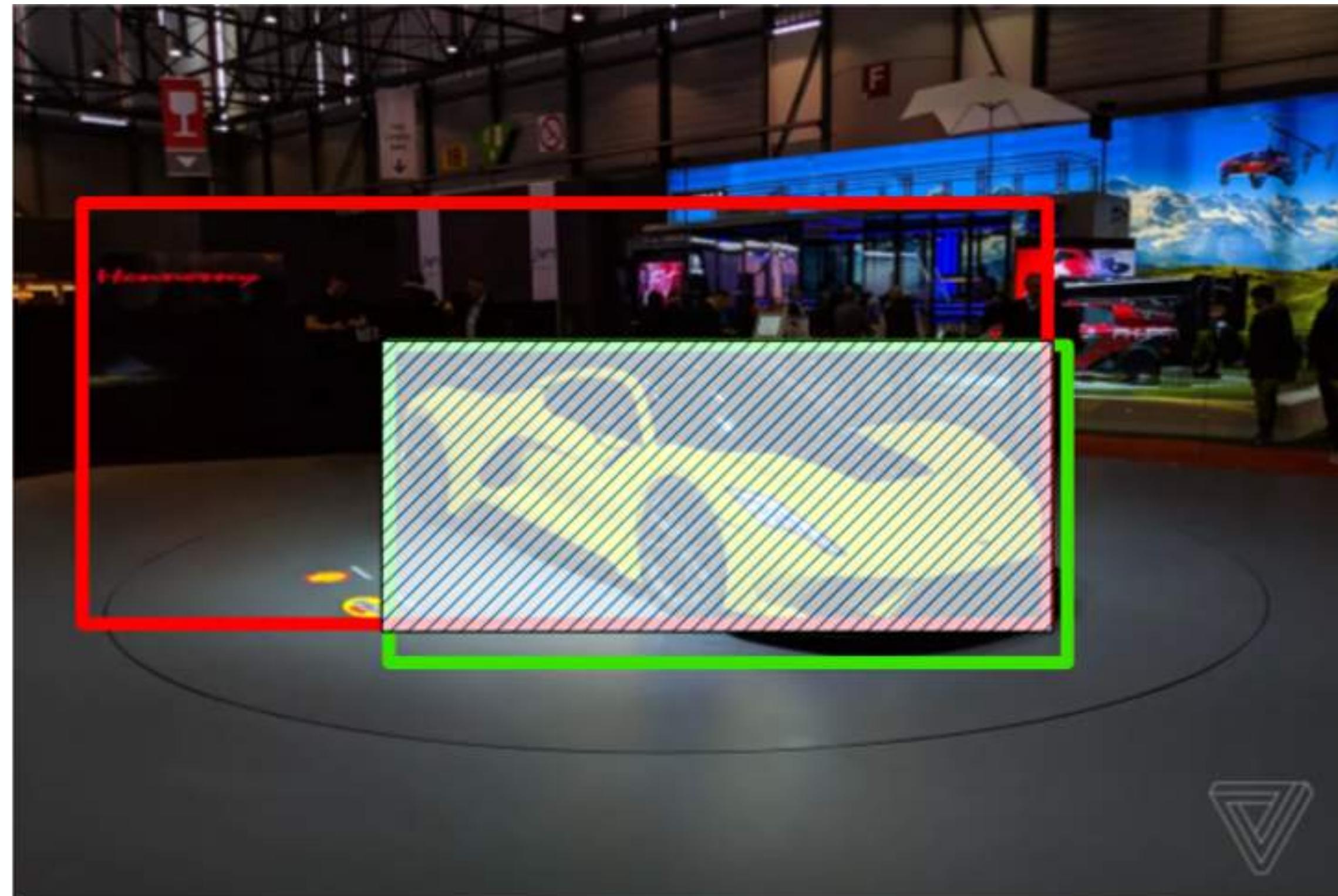
- Let's assume the **green** box is our ground truth
- Our Object Detector proposes the **red** box
- How do we assess the **goodness** of this proposal?

# How much of the correct area is covered by our predicted bounding box?



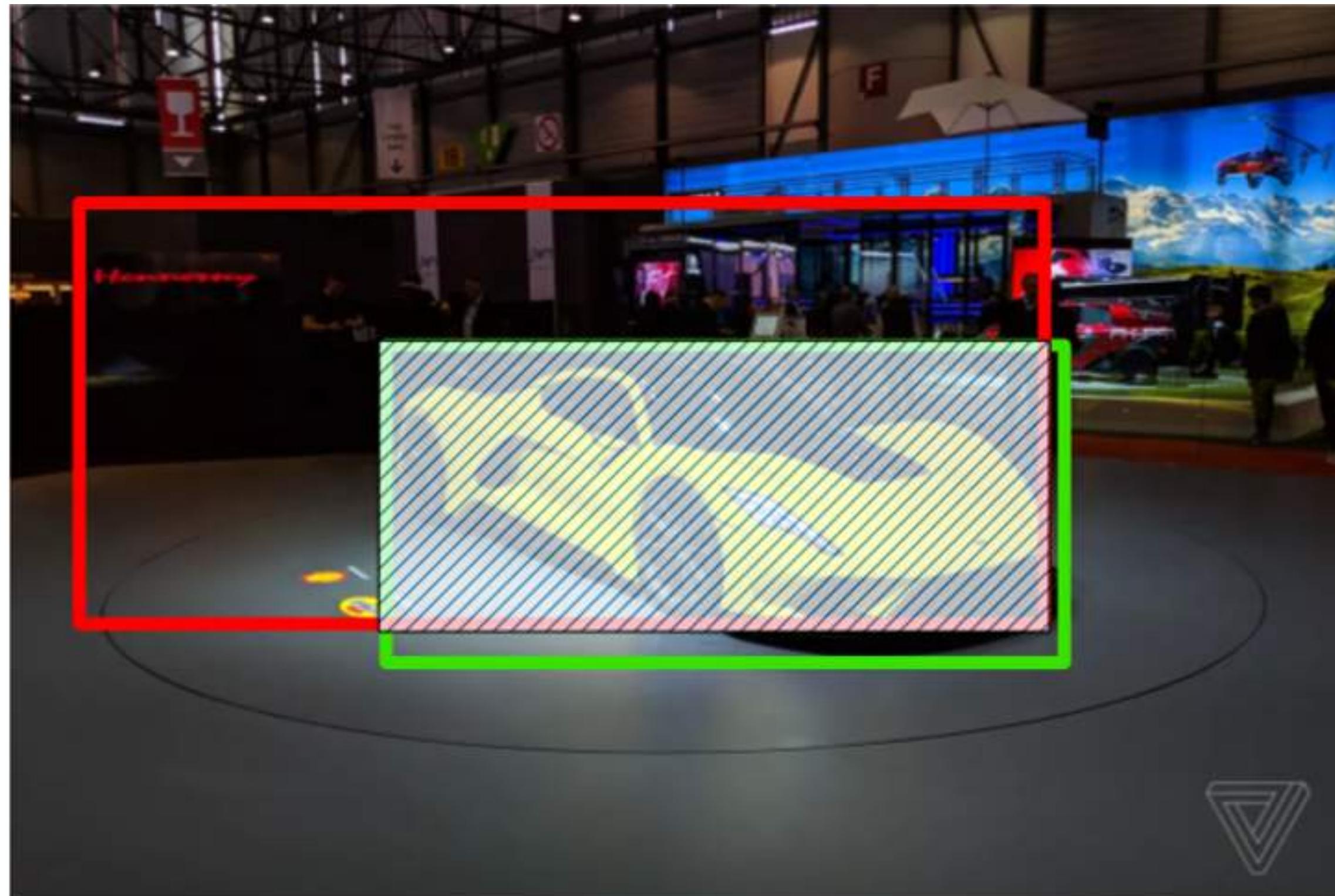
- From a glance it looks like it covers almost 90% of the ground truth, great!
- However, is this a good metric to use?

# How much of the correct area is covered by our predicted bounding box?

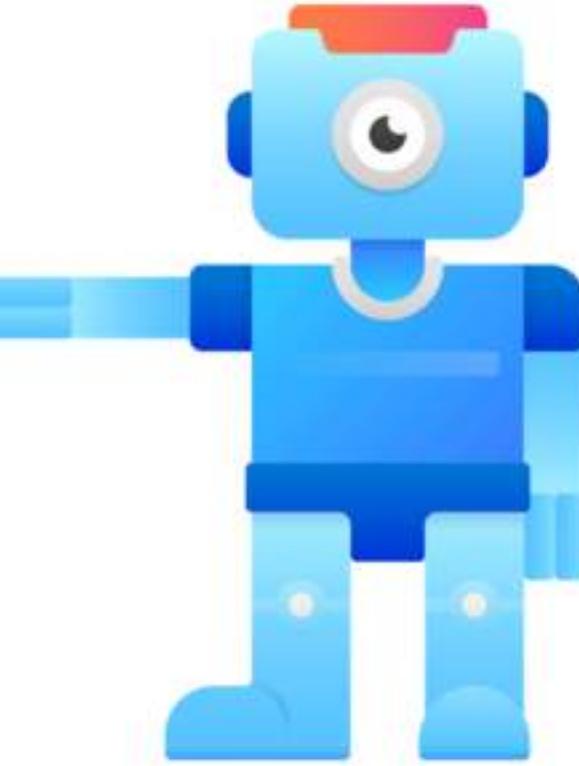


- This box also covers 90% of the green box, but clearly this isn't a good proposal
- We solve this by using the Intersection per Union metric

# How much of the correct area is covered by our predicted bounding box?



- $IoU = \frac{\text{Size of Union}}{\text{Size of Prediction}}$
- IoU considers the ratio of the overlap to the size of the proposed box
- An IoU over 0.5 is considered acceptable
- The higher the IoU the better the bounding box

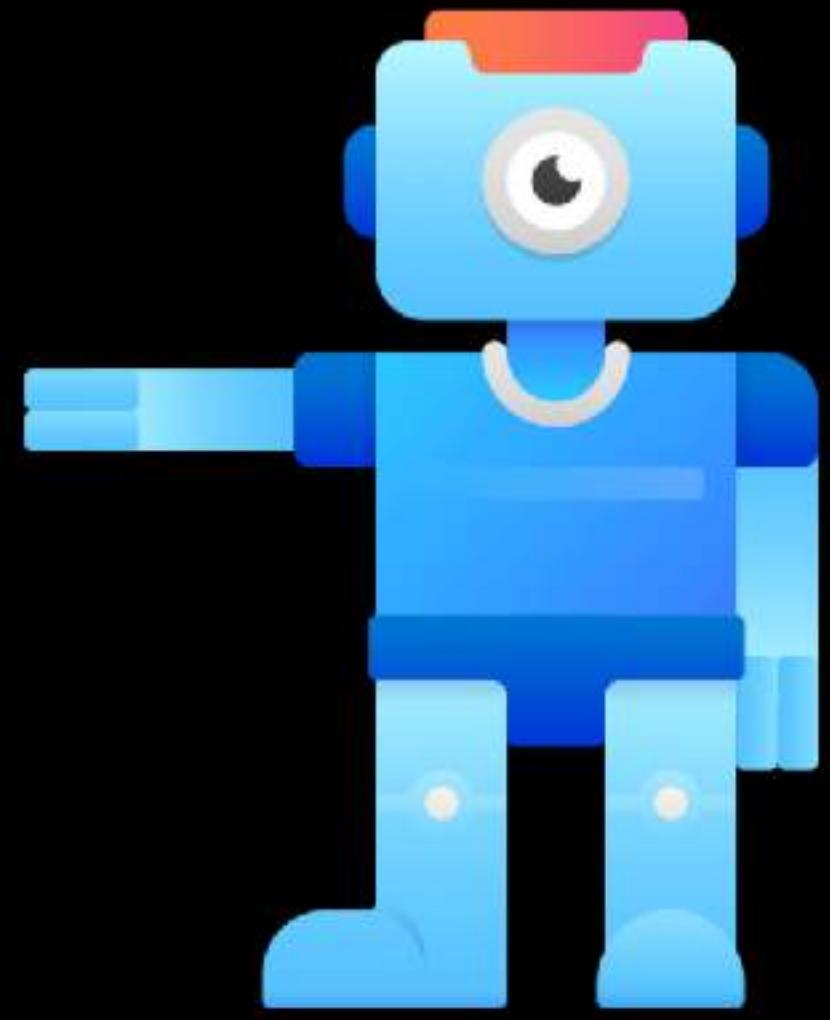


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Mean Average Precision (mAP)**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Mean Average Precision (mAP)

Another Metric used to assess Object Detector Performance

# Mean Average Precision (mAP)

- The best way to compare Object Detection Models on the same dataset
- It's the aggregation of several metrics

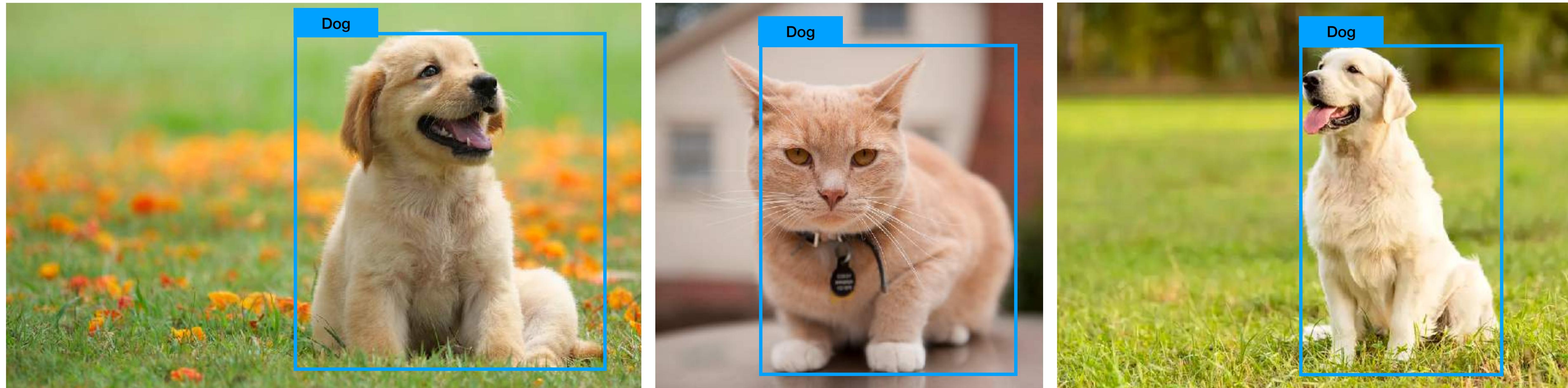
# Mean Average Precision (mAP)

## A brief explanation

- $Precision = \frac{TP}{TP + FP}$ , when your model predicts positive how often is it right.
- $Recall = \frac{TP}{TP + FN}$ , how well your model is at finding all positives
  - **TP** - Got a positive correct
  - **FP** - Predicted positive but was wrong
  - **FN** - Missed a positive

# Mean Average Precision (mAP)

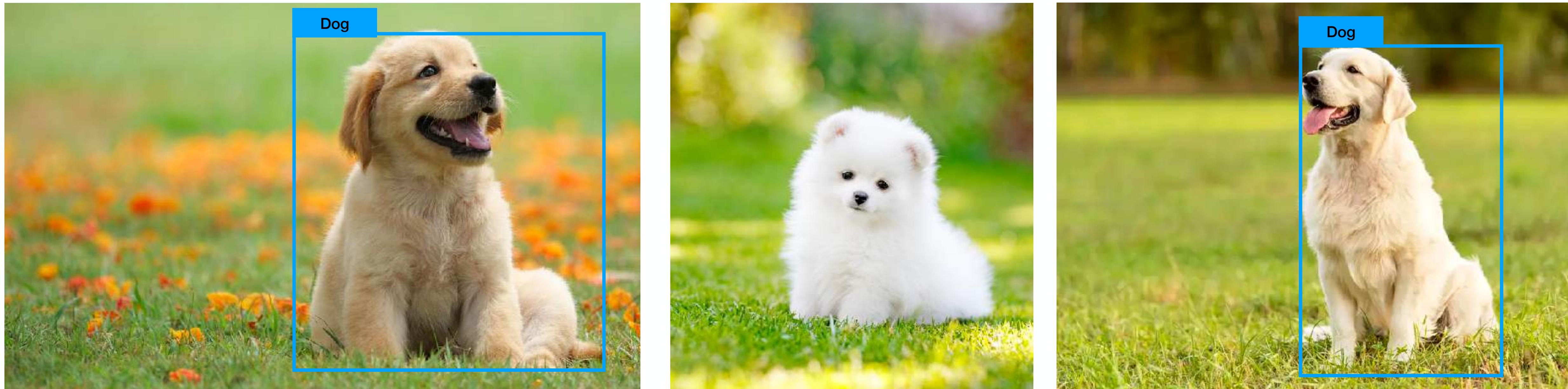
## Precision



- Our model predicted dogs 3 times but only 2 were correct
- **Precision = 2/3**

# Mean Average Precision (mAP)

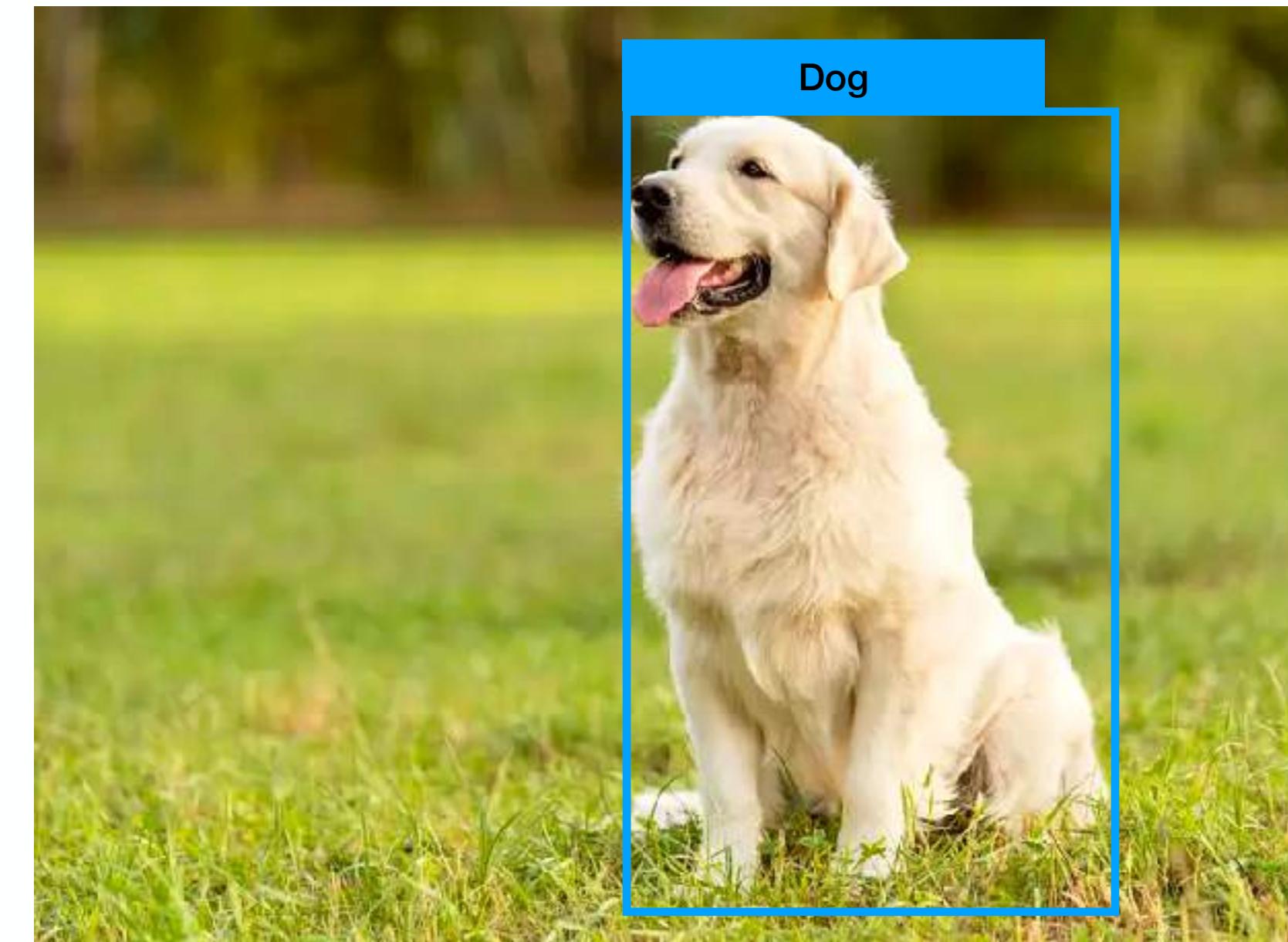
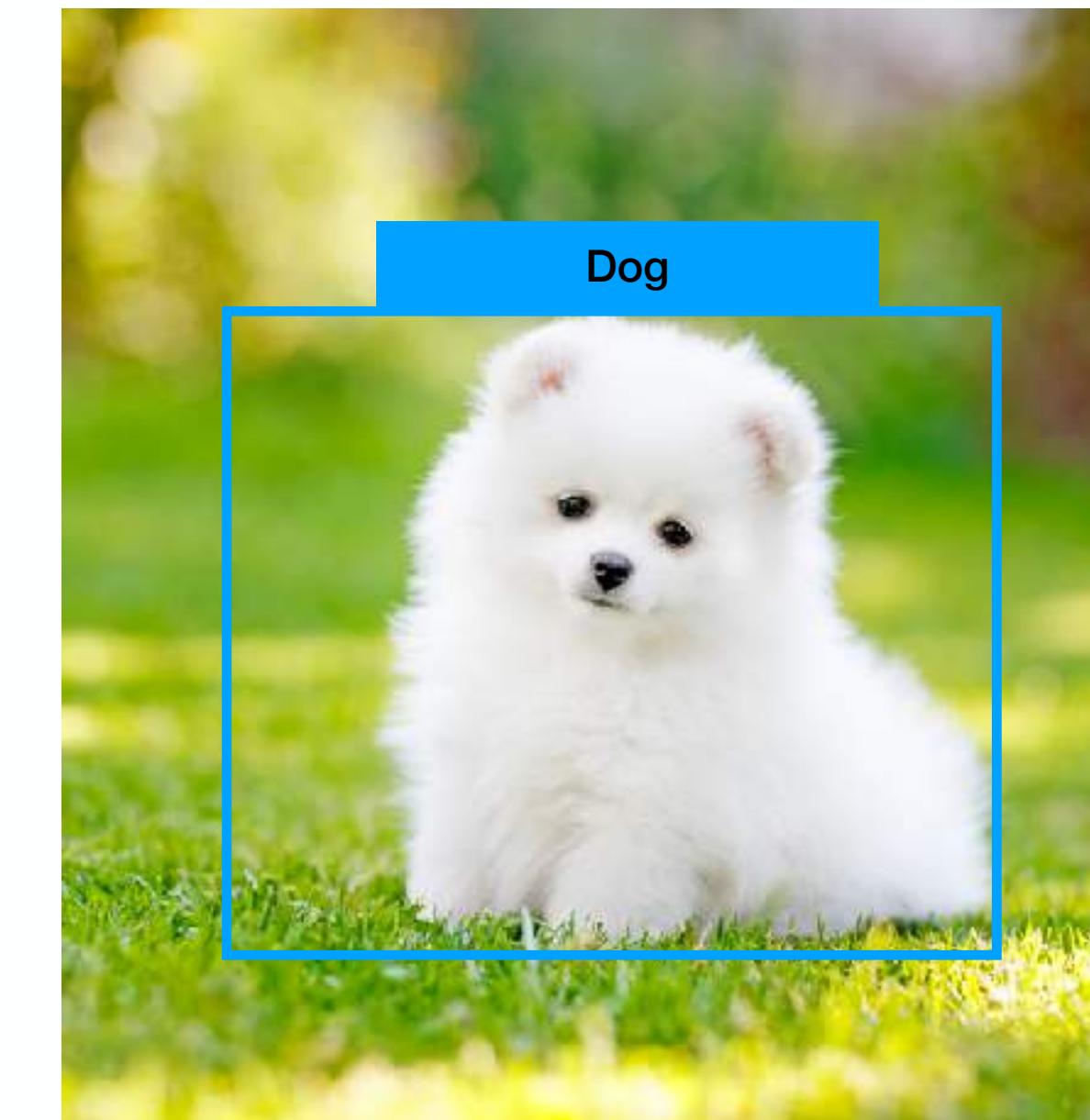
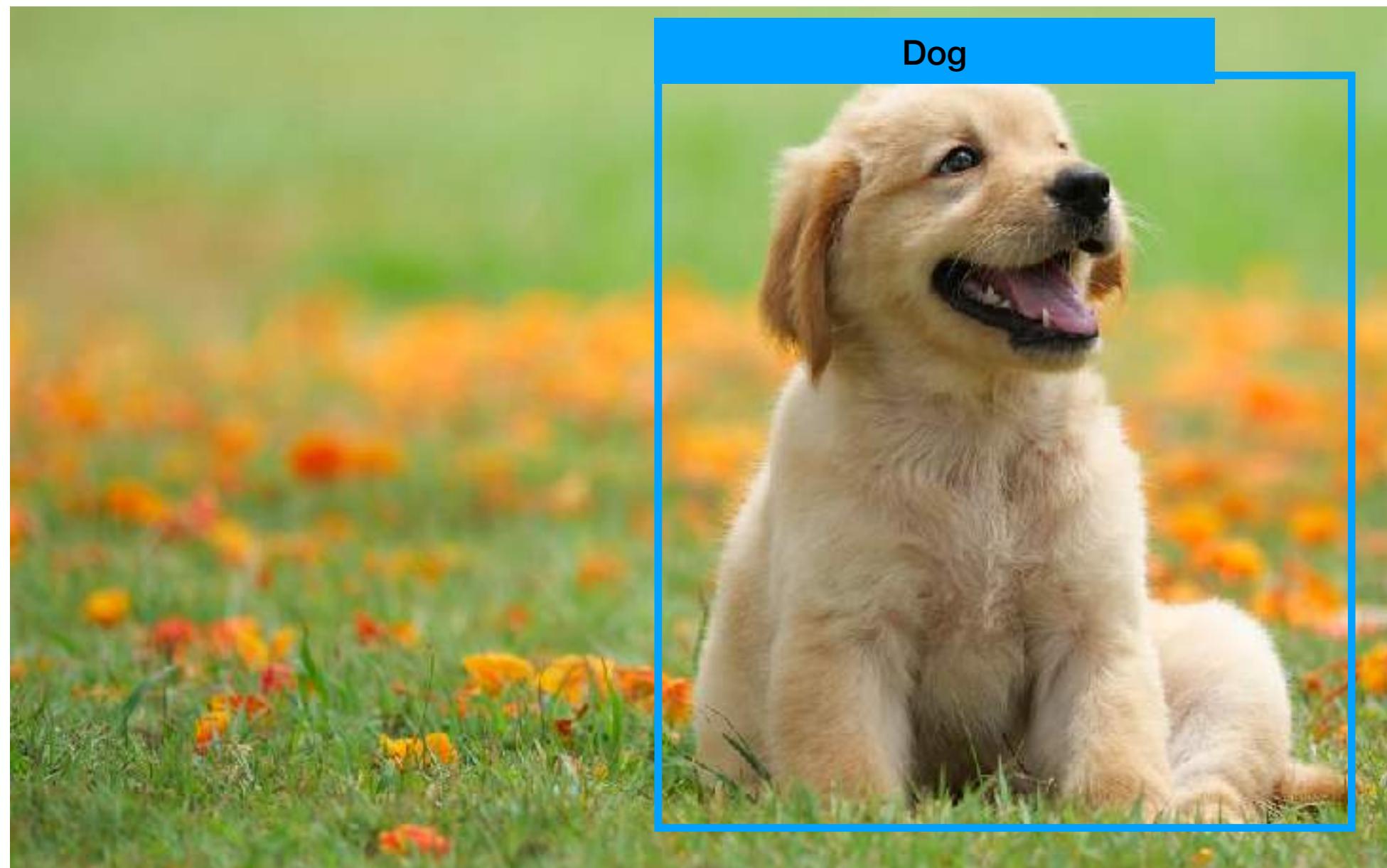
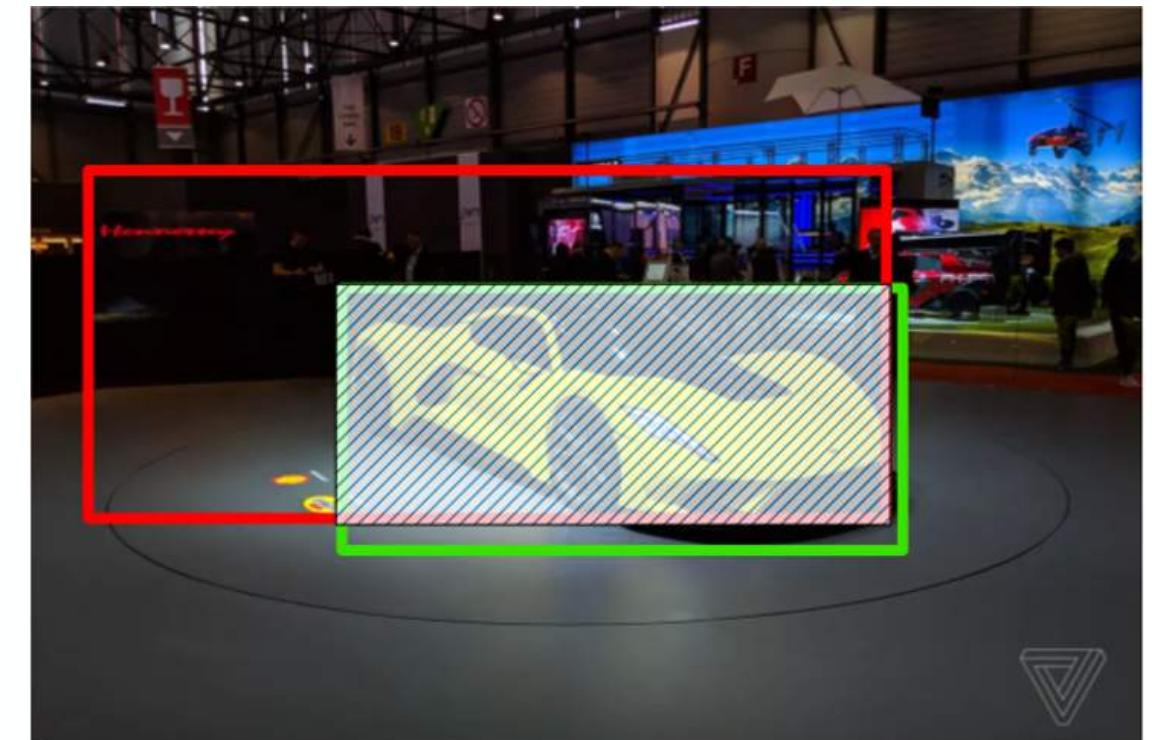
## Recall



- Our model predicted 2 dogs but missed 1
- **Recall = 2/3**

# Mean Average Precision (mAP)

What if we changed our IoU threshold?



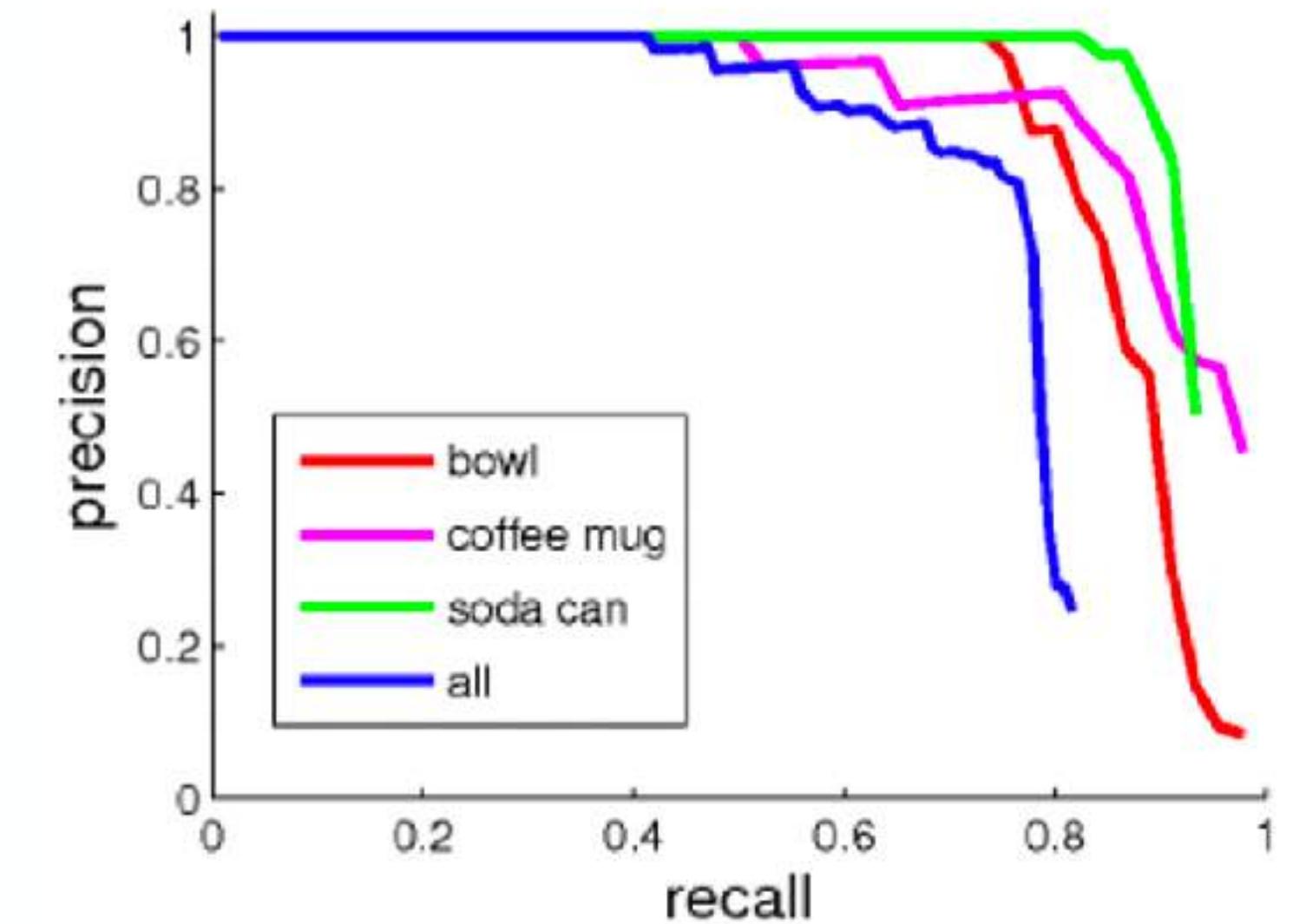
- Before if we used a strict IoU criteria, we wouldn't generate a bounding box for the middle image
- Different IoU thresholds yield different precision and recall values

# Mean Average Precision (mAP)

- **Average Precision (AP)** is the finding the area under the precision-recall (varying the IoU/confidence threshold).
- The **Mean Average Precision or mAP** is the average of all these APs

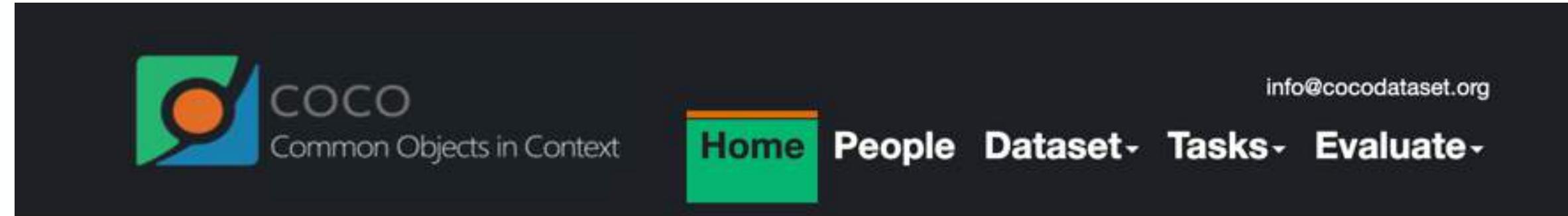
$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k$$

*AP<sub>k</sub> = the AP of class k*  
*n = the number of classes*



[https://www.researchgate.net/figure/Object-detection-results-on-a-video-sequence-Left-Precision-recall-curves-of\\_fig4\\_221069160](https://www.researchgate.net/figure/Object-detection-results-on-a-video-sequence-Left-Precision-recall-curves-of_fig4_221069160)

# The COCO Dataset - Object Detector Benchmark Dataset



## News

- We are pleased to announce the LVIS 2021 Challenge and Workshop to be held at ICCV.
- Please note that there will not be a COCO 2021 Challenge, instead, we encourage people to participate in the LVIS 2021 Challenge.
- We have partnered with the team behind the open-source tool [FiftyOne](#) to make it easier to download, visualize, and evaluate COCO
- [FiftyOne](#) is an open-source tool facilitating visualization and access to COCO data resources and serves as an evaluation tool for model analysis on COCO.

## What is COCO?



COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

- ✓ Object segmentation
- ✓ Recognition in context
- ✓ Superpixel stuff segmentation
- ✓ 330K images (>200K labeled)
- ✓ 1.5 million object instances
- ✓ 80 object categories
- ✓ 91 stuff categories
- ✓ 5 captions per image
- ✓ 250,000 people with keypoints

## Collaborators

Tsung-Yi Lin Google Brain

Genevieve Patterson MSR, Trash TV

Matteo R. Ronchi Caltech

Yin Cui Google

Michael Maire TTI-Chicago

Serge Belongie Cornell Tech

Lubomir Bourdev WaveOne, Inc.

Ross Girshick FAIR

James Hays Georgia Tech

Pietro Perona Caltech

Deva Ramanan CMU

Larry Zitnick FAIR

Piotr Dollár FAIR

## Sponsors



**CVDF**



**Microsoft**



**facebook**

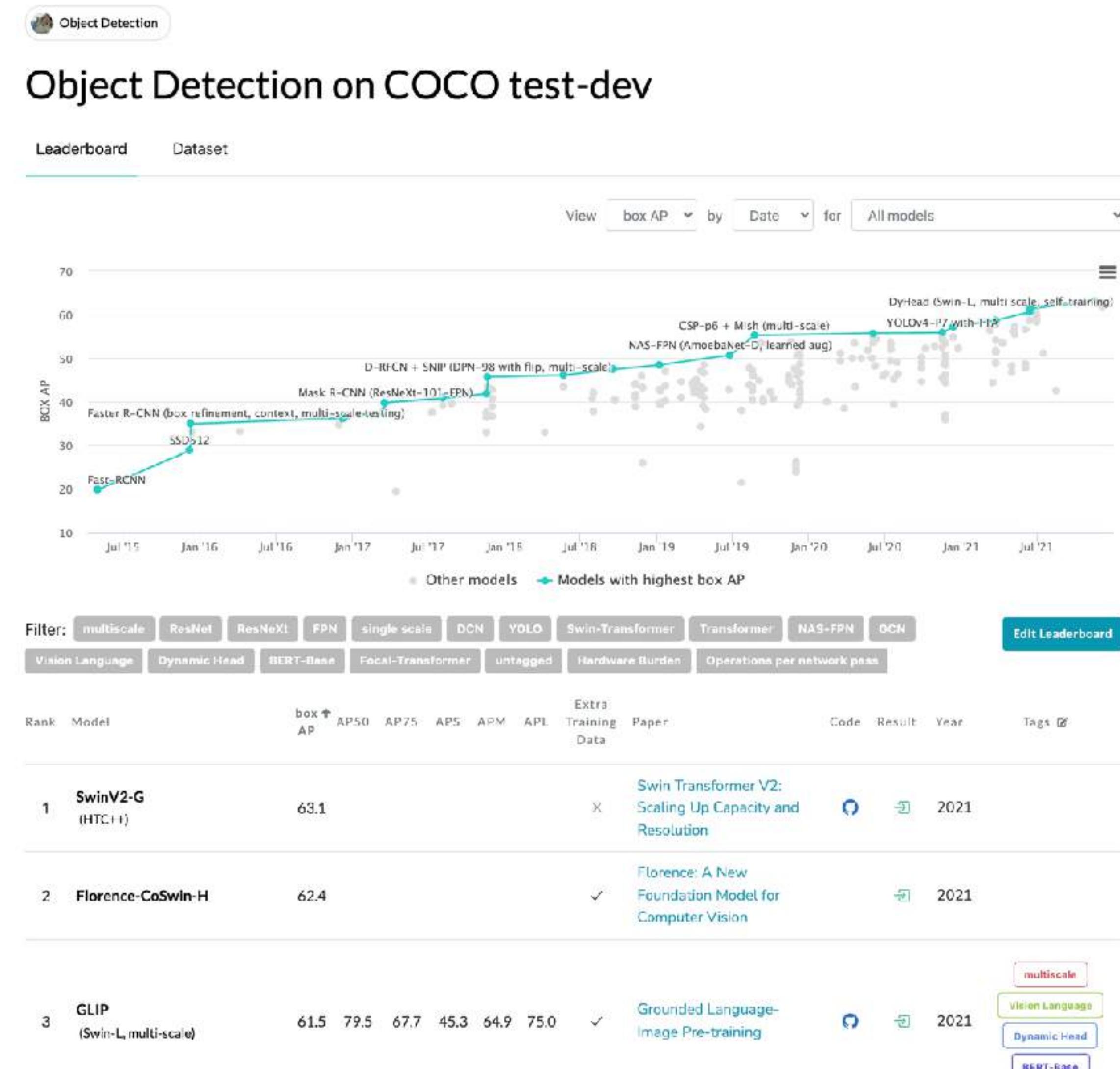


**Mighty Ai**

## Dataset examples



# The COCO Dataset - State of Art Performers



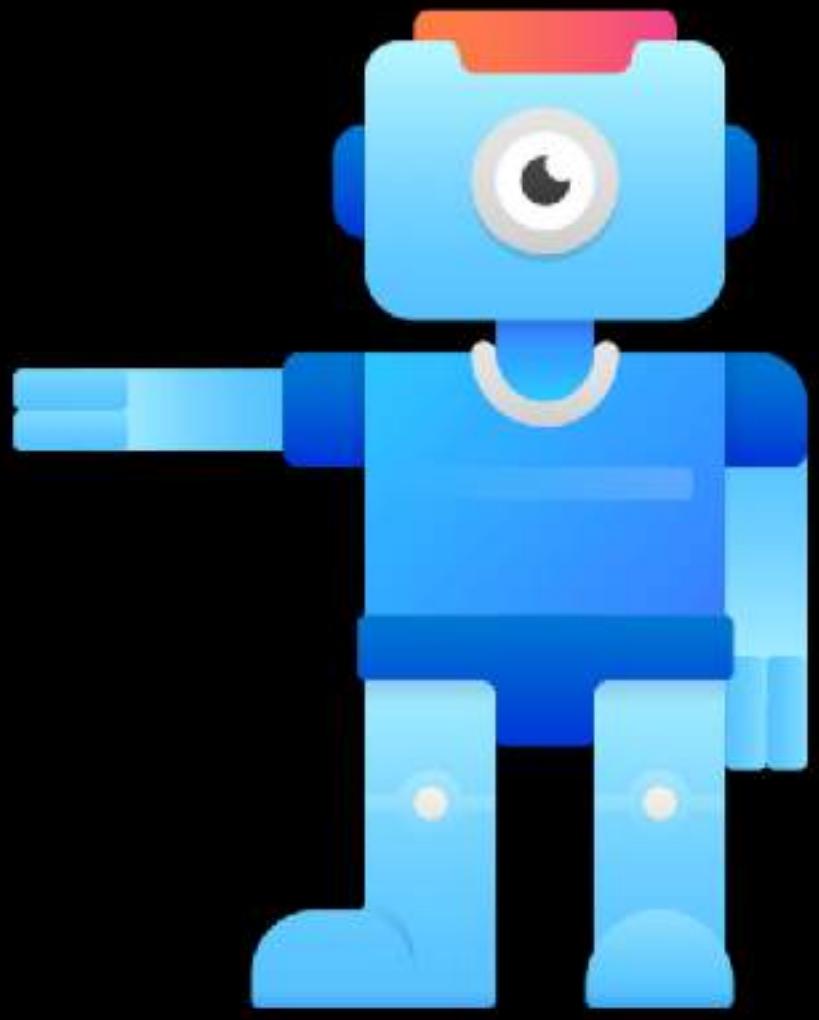


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Non-Maximum Suppression**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

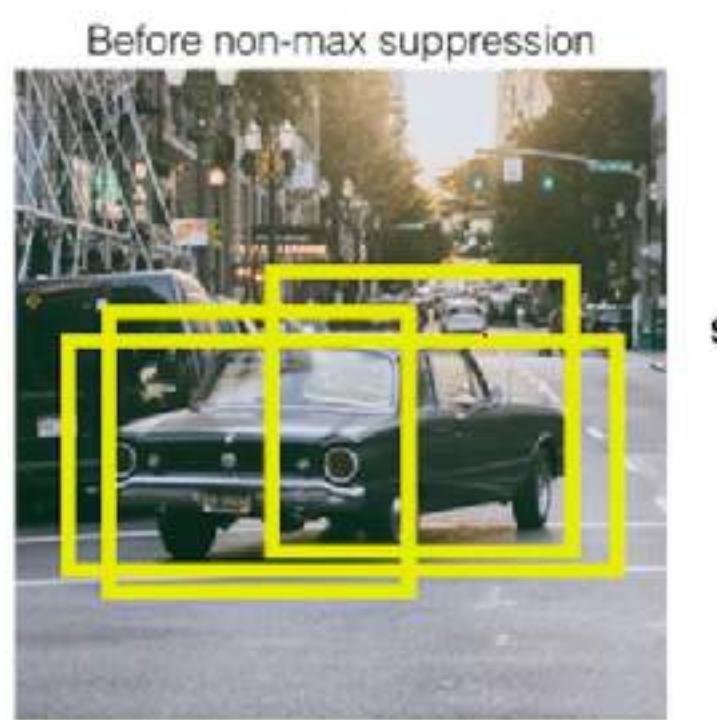
## Non-Maximum Suppression

Clean up bounding box proposals

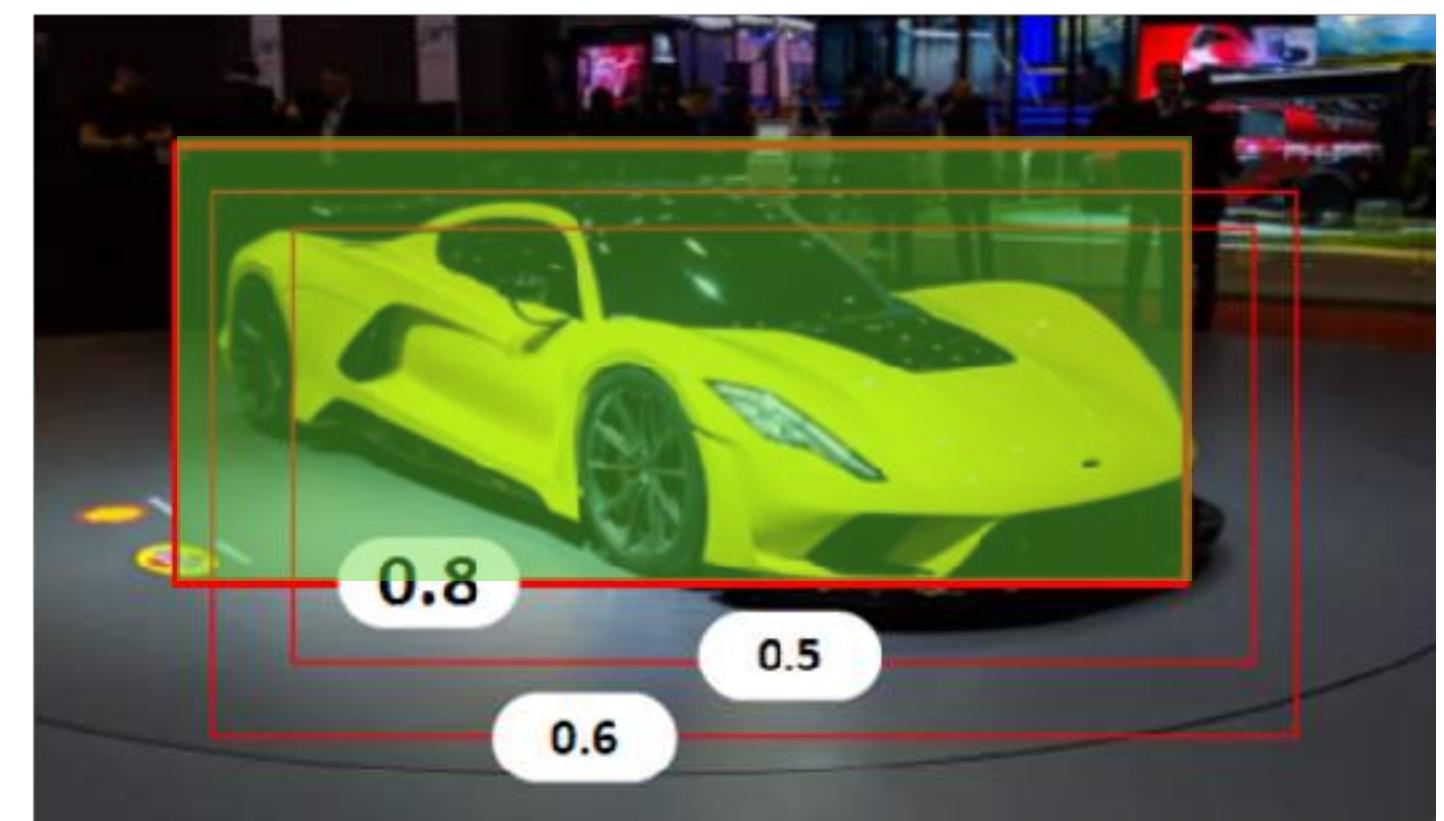
# Non-Maximum Suppression

A technique to clean up overlapping bounding boxes

- **STEP 1** – Get overlapping bounding boxes
- **STEP 2** – Get the maximum probabilities associated with each box (probability of the object belonging a class e.g. a car)
- **STEP 3** – Select the bounding box with the highest confidence:
  - Add it to the **Final Proposal List** (initially empty)
  - Remove it from our **Initial Proposal List**
- **STEP 4** – Calculate the IoU for the boxes in the **Initial Proposal List** with the box in the **Final Proposed List**
- **STEP 5** – If the **IoU exceeds a set threshold** (typically 0.5), meaning the boxes overlap a lot we **reject it**
- **STEP 6** – We do this for all proposals in the **Initial Proposed List**



Non-Max  
Suppression  
→



$$IoU = \frac{\text{Size of Union}}{\text{Size of Prediction}}$$

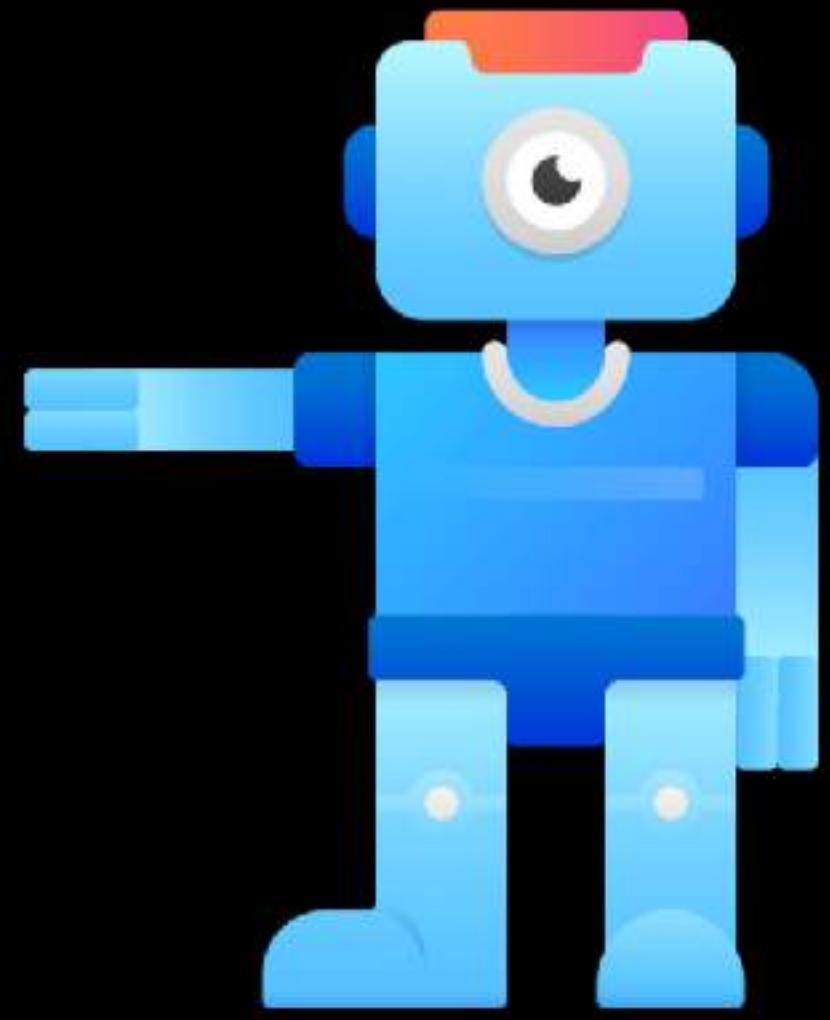


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**R-CNNs, Fast R-CNNs and Faster R-CNNs**



# MODERN COMPUTER VISION

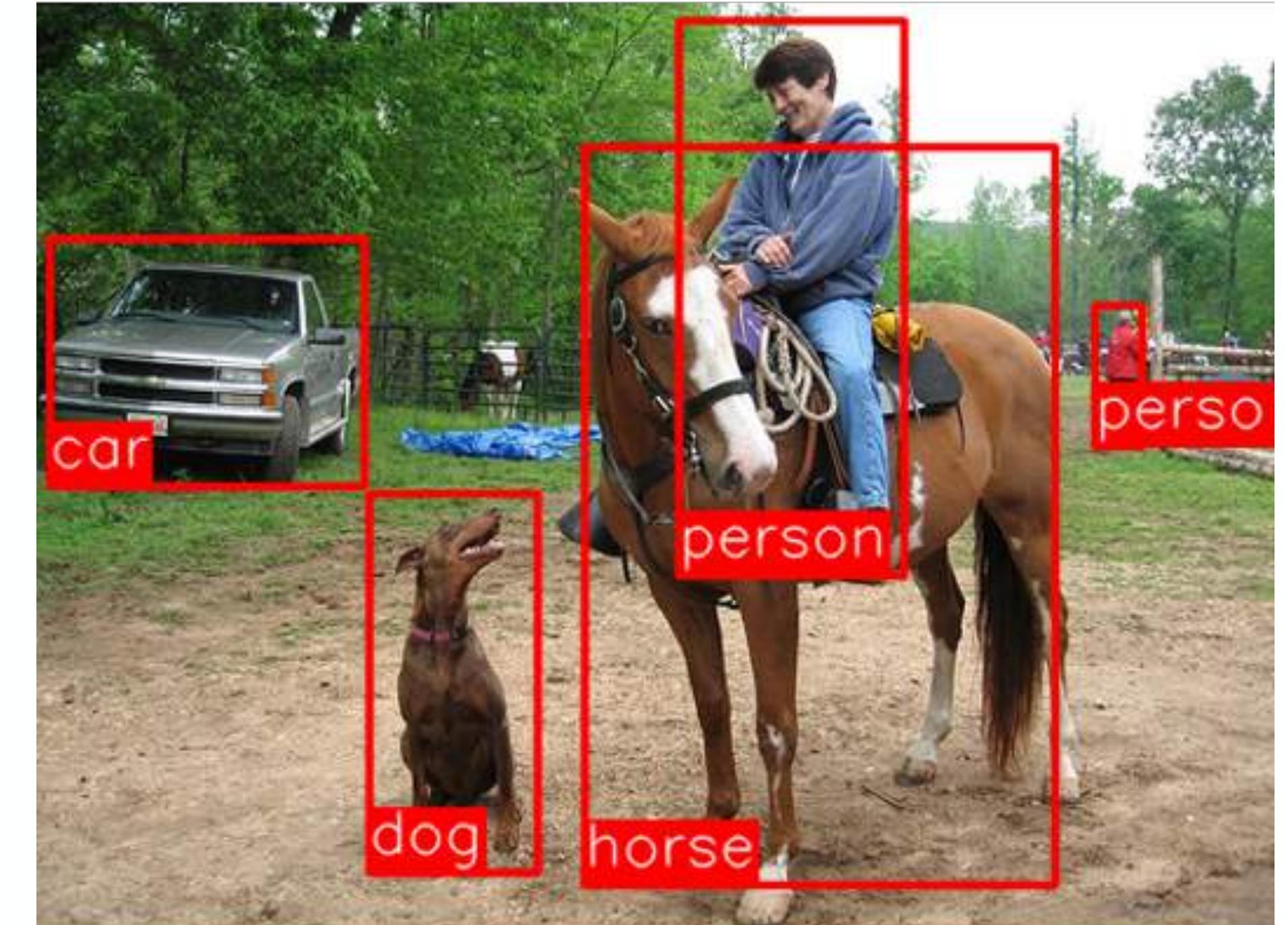
BY RAJEEV RATAN

## R-CNNs, Fast R-CNNs and Faster R-CNNs

Overview of the R-CNNs family of Models

# Regions with CNNs or R-CNNs

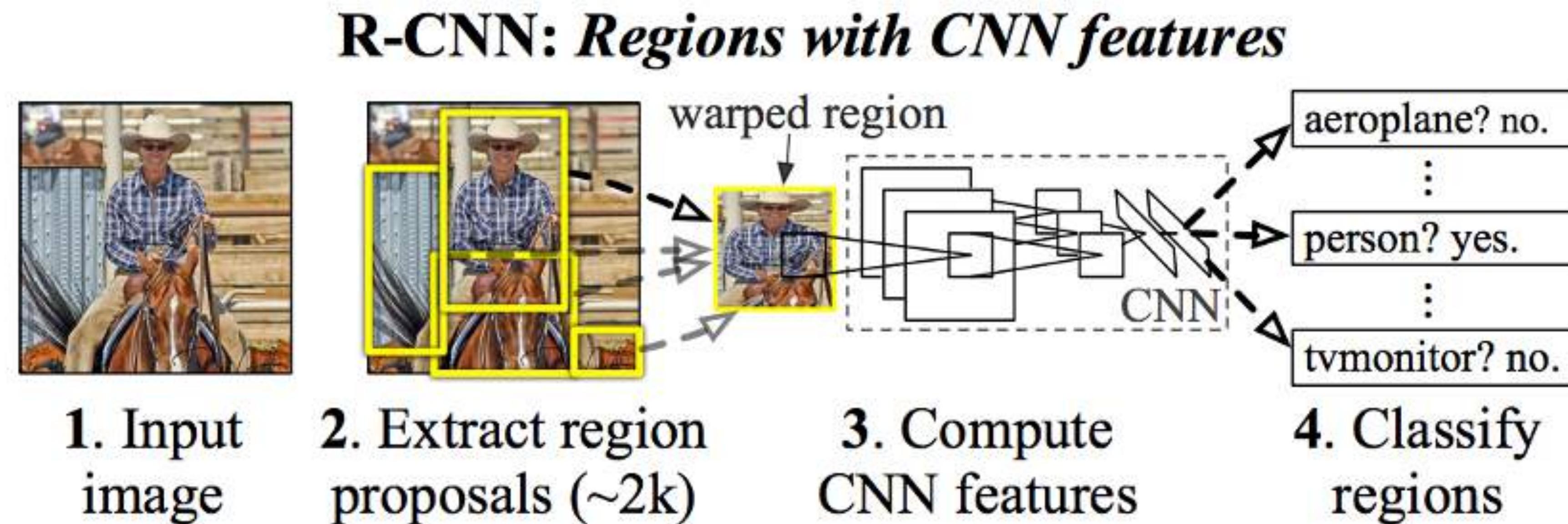
- Not to be confused with Recurrent Neural Networks or RNNs, **R-CNNs** were one of the first **Deep Learning** based **Object Detectors**.
- Introduced in 2014 by researchers at the University College of Berkeley, R-CNNs obtained dramatically higher performance in the PASCAL VOC Challenge (A dataset used to benchmark Object Detectors similar to COCO).



<https://arxiv.org/abs/1311.2524>

# R-CNNs Overview

- R-CNNs attempted to solve the **exhaustive search** previously performed by sliding windows, by proposing bounding boxes and passing these extracted boxes to an image classifier.
- But how does the algorithm determine these bounding box proposals?
- By using the **Selective Search algorithm**



# R-CNNs - Selective Search

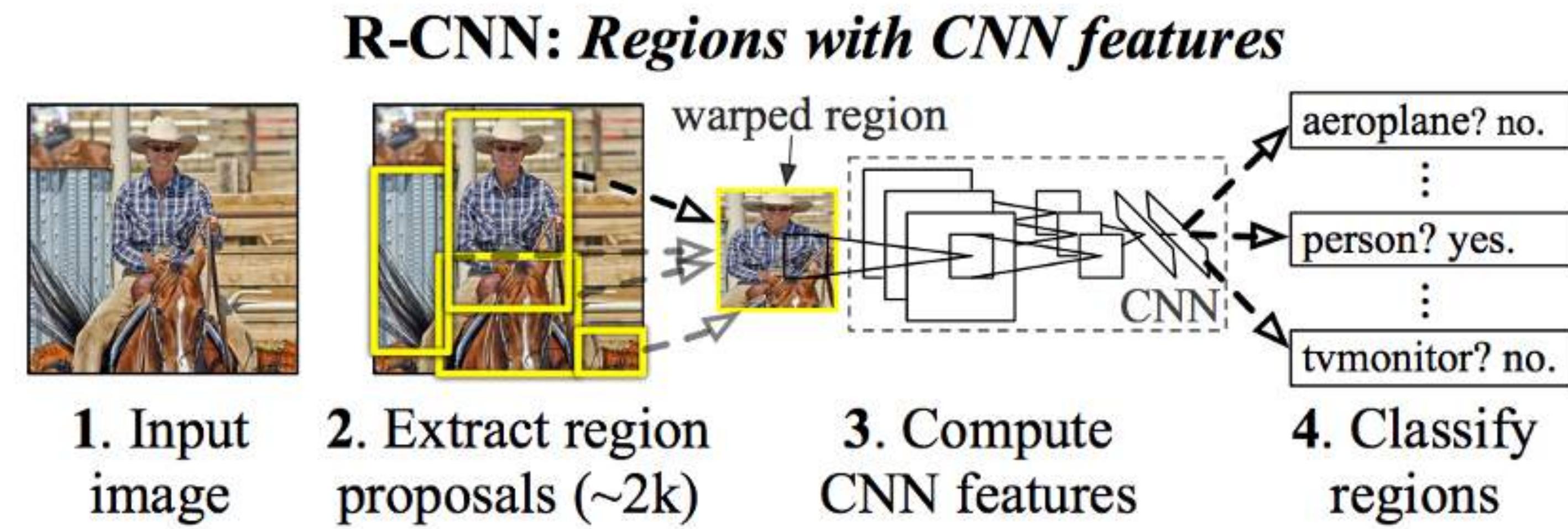
- Selective Search attempts to **segment the image into groups** by combining similar areas such as colours/textures and propose these regions as “interesting” bounding boxes.



Figure 2: Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales. On the right we necessarily find the objects at different scales as the girl is contained by the tv.

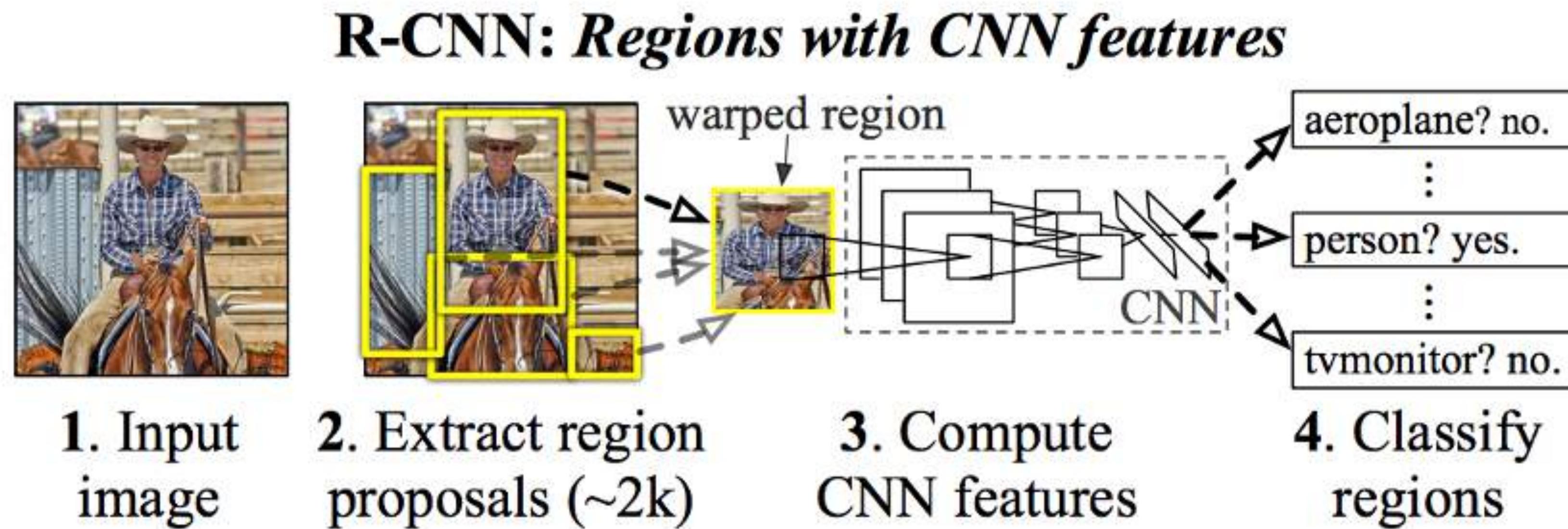
# R-CNNs - Selective Search

- When selective search has identified these **regions/boxes** it passes these extracted images to our CNN (e.g. one trained on ImageNet) for classification.
- We don't use the CNN directly for classification (although we can), we use an SVM to classify the CNN extracted features.



# R-CNNs - Selective Search

- After the first region proposal has been classified, we then use a simple linear regression to generate a tighter bounding box.



# Newer Version - Fast R-CNNs

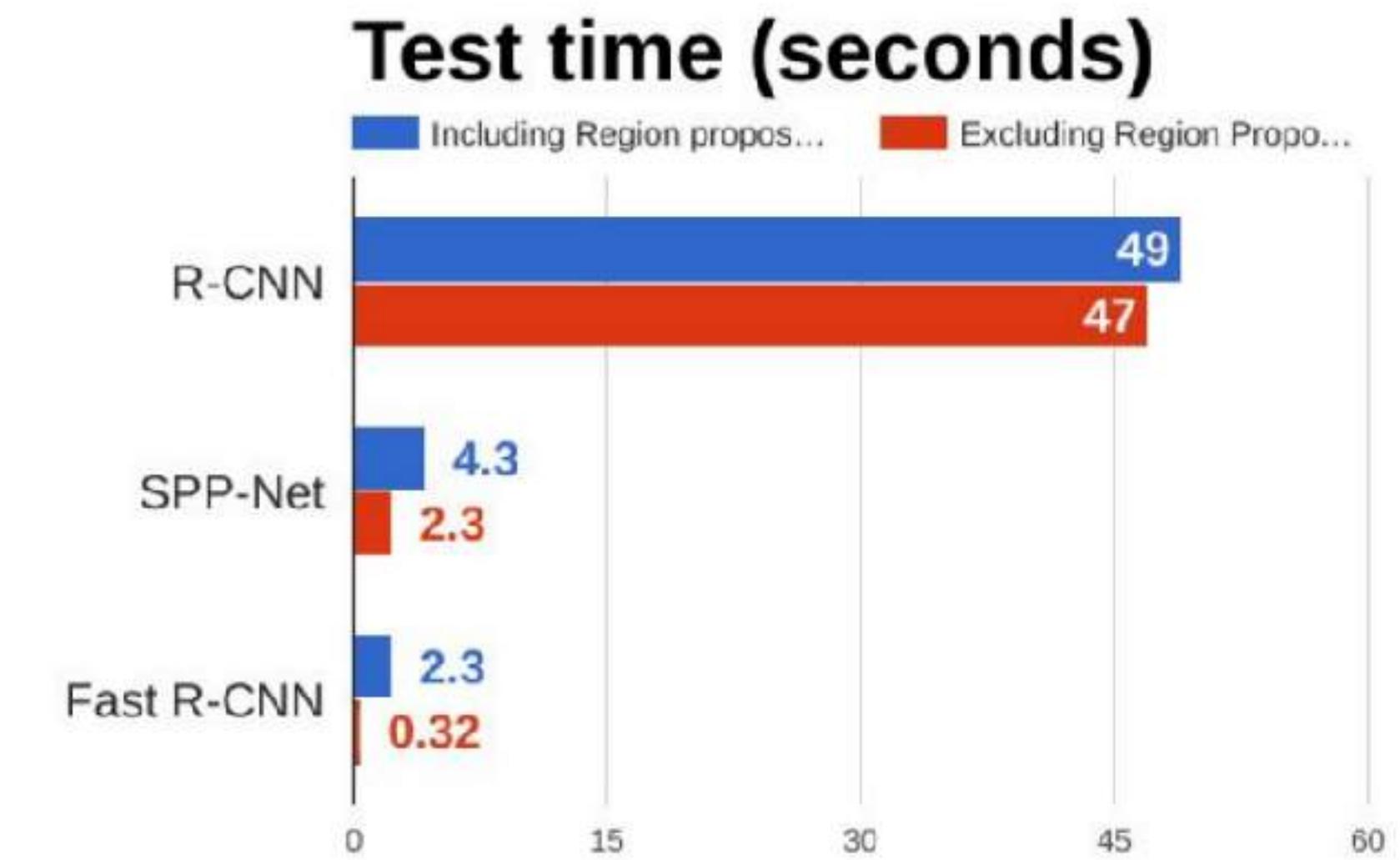
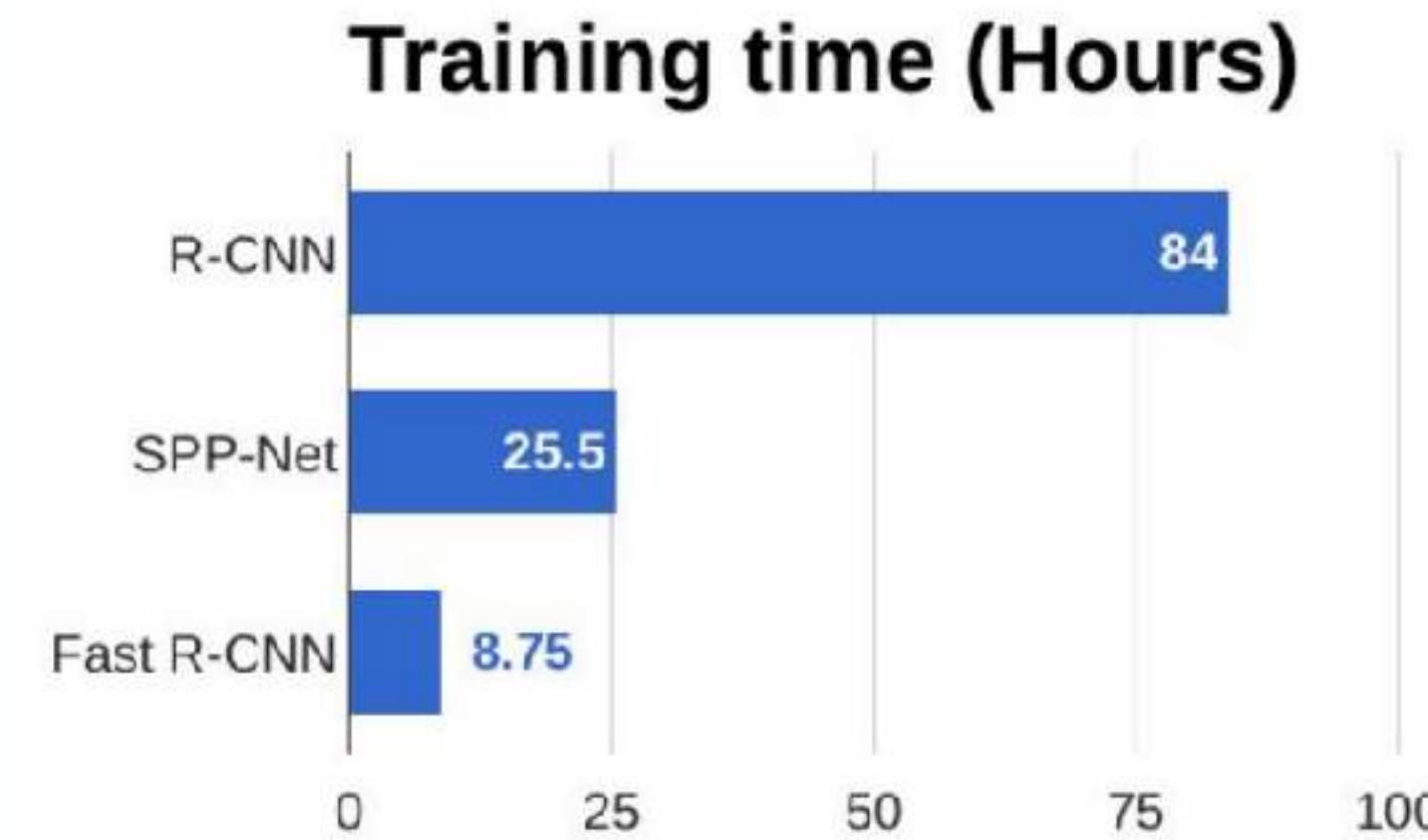
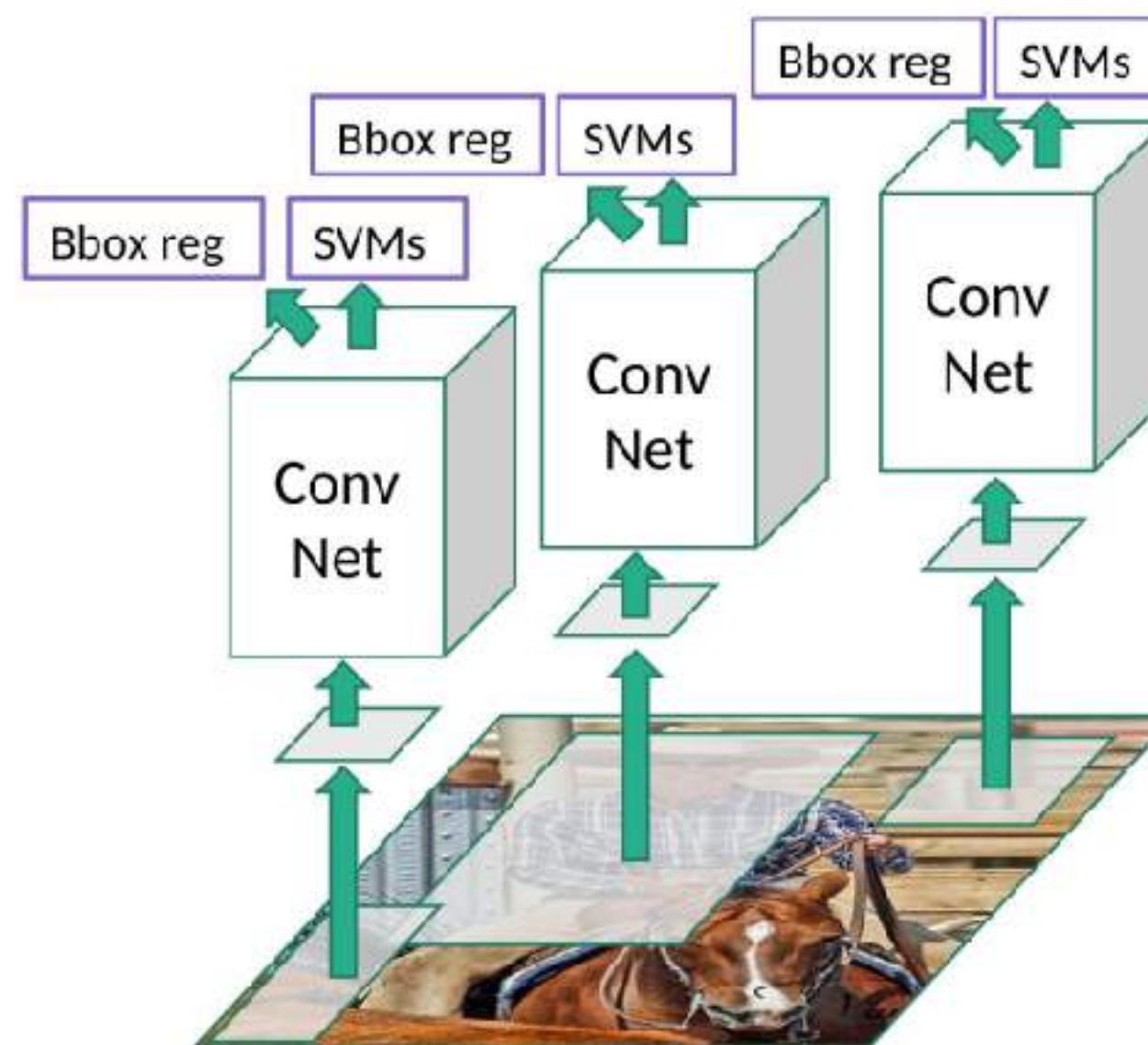
- While state of the art in 2014, R-CNNs were **notoriously slow** as each bounding box had to be classified by our CNN. This made real-time Object Detection almost impossible.
- The reasons for the heavy computation was due to requiring **3 separately trained models:**
  - A Feature Extraction CNN
  - An SVM to predict the final Class
  - A Linear Regression model to tighten the bounding box
- In the following iteration in 2015, **Faster R-CNNs** attempted to solve these problems

# Combining the training of the CNN, Classifier and Bounding Box Regressor into a single Model

- Faster R-CNNs firstly **reduced the number of proposed bounding boxes** by removing the overlap generated. How?
- We run the CNN across the image just once use a technique called **Region of Interest Pooling (RoIPool)**.
- **RoIPool** allows us to share the forward pass of a CNN for the image across its sub regions.
- This works because previously regions are simply extracted from the CNN feature map and then pooled. **Therefore, there is only need to run our CNN once on the image!**

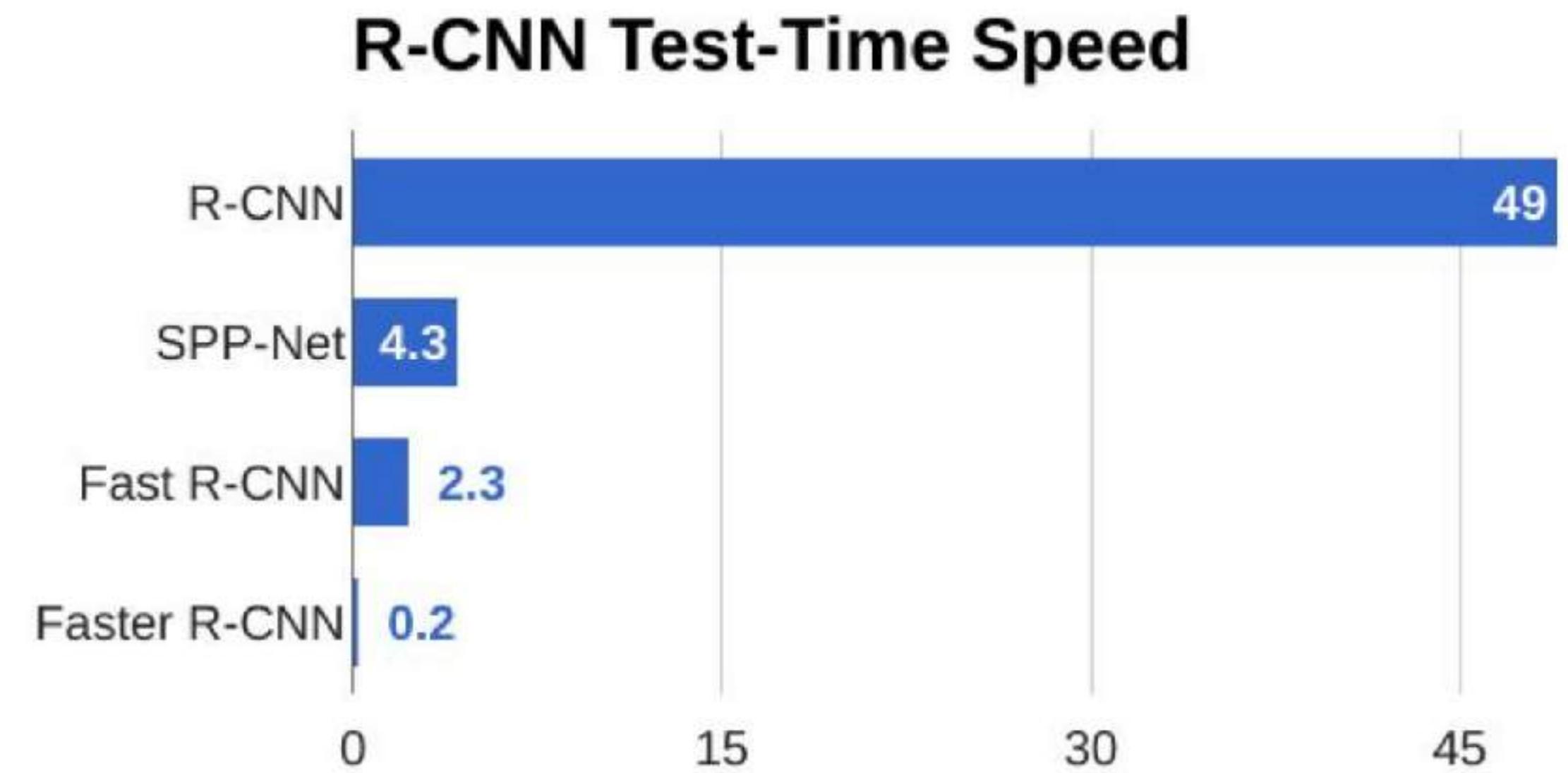
# Fast R-CNNs - Improvements

- **Training time was improved** by combining the training of the CNN, Classifier and Bounding Box Regressor into a single Model
- **SVM Feature Classifier** -> Softmax layer at the top of the CNN
- **Linear Regression** -> Bounding Box output layer (parallel to our Softmax)



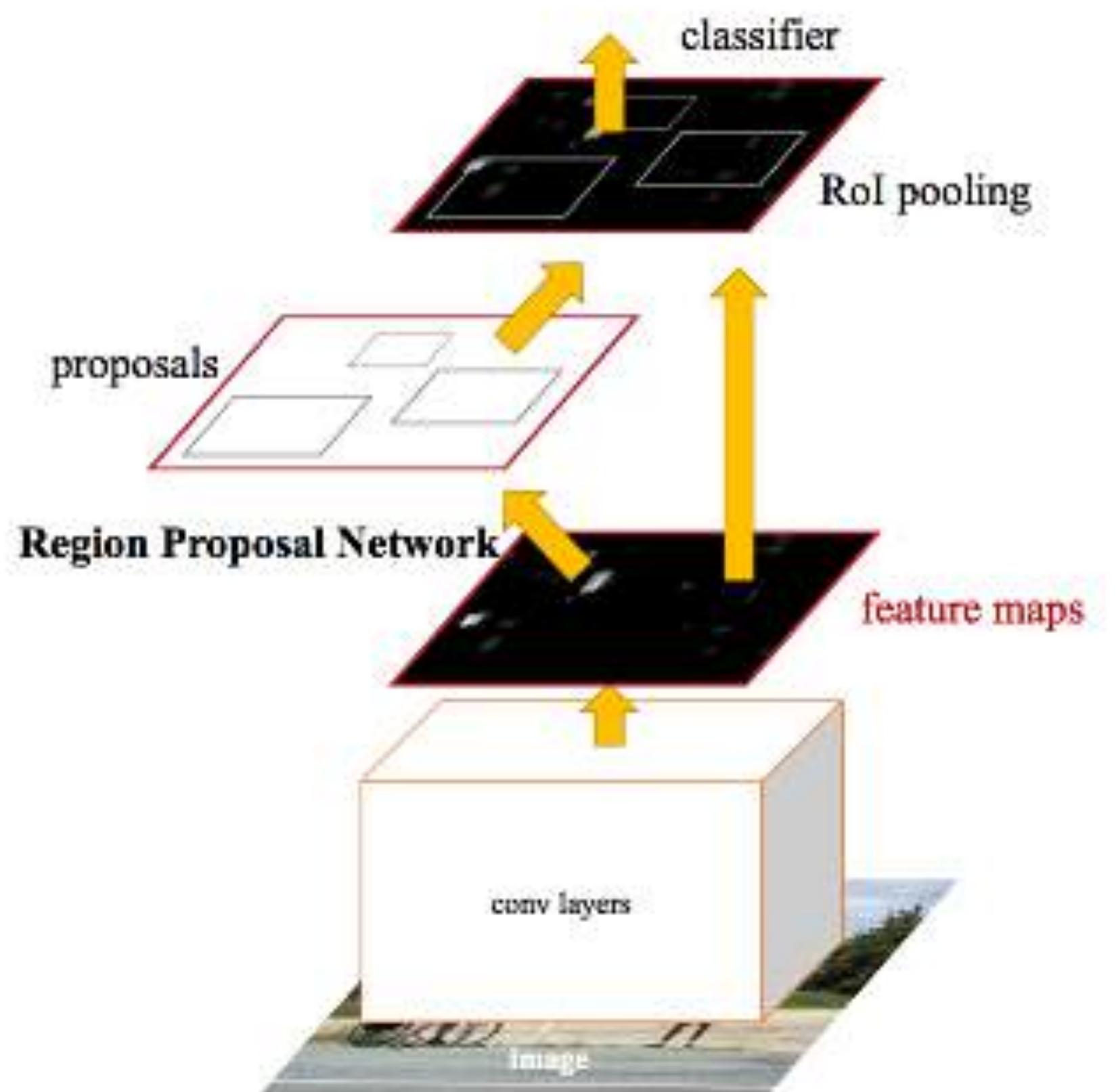
# Faster R-CNNs - 2016 Improvements

- Fast R-CNNs made significant speed increases however, region proposal still remained relatively slow as it still relied on **Selective Search**.
- Fortunately, a Microsoft Research team figured out a how to eliminate this bottleneck.



# Speeding Up Region Proposal

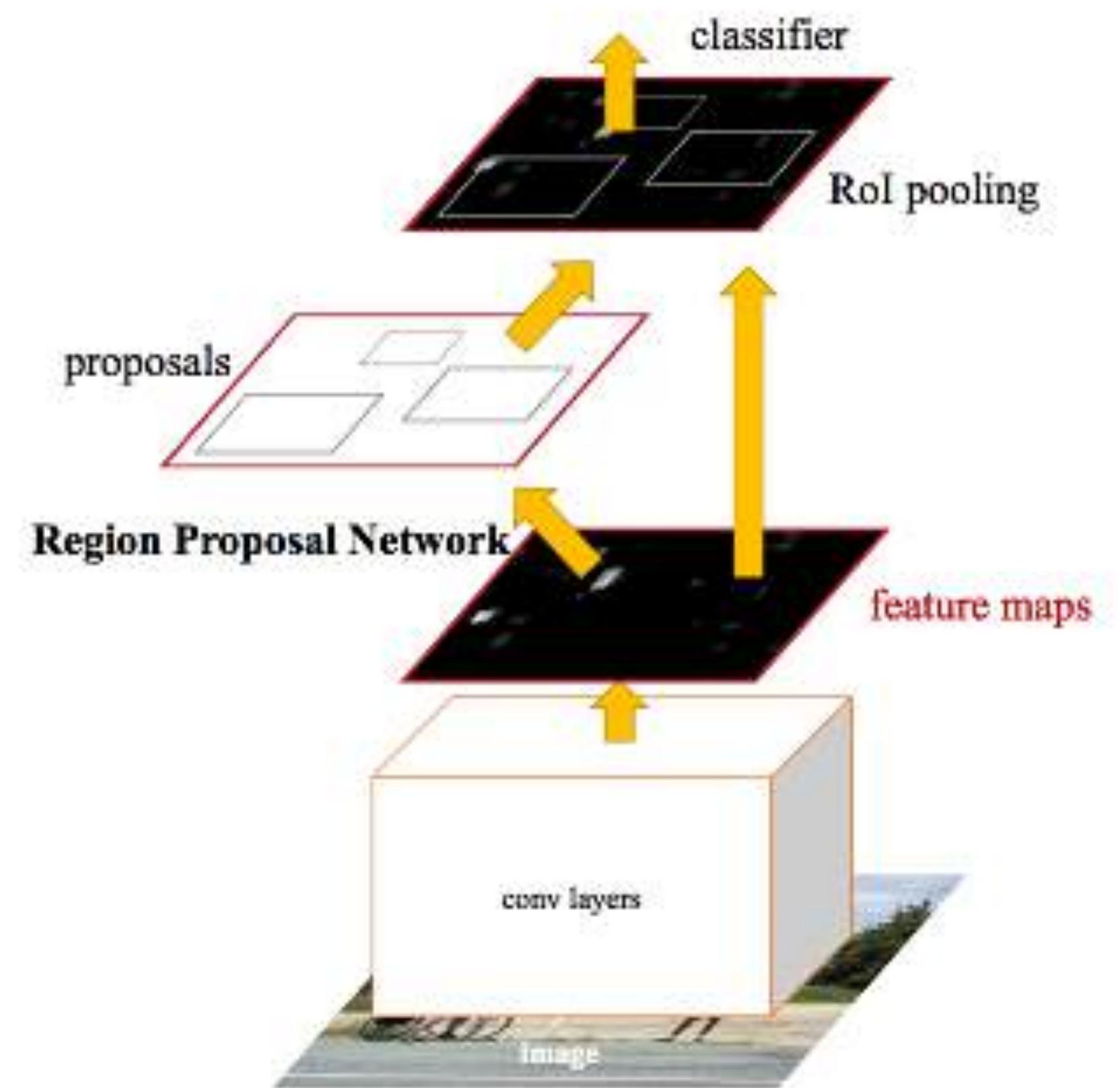
- **Selective Search** relies on features extracted from the image.
- What if we just **reused those features** to do Region Proposal instead?
- That was the insight that made Faster R-CNNs extremely efficient.

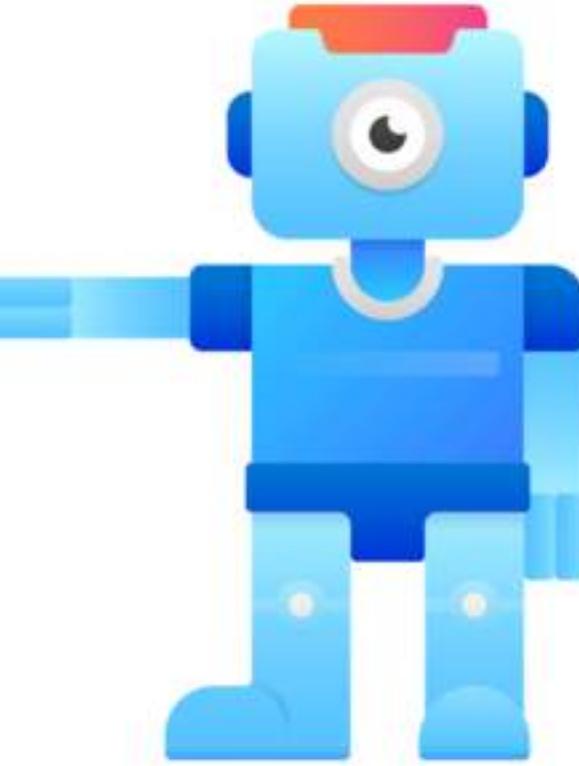


<https://arxiv.org/abs/1506.01497>

# Region Proposal with Faster R-CNNs

- Faster R-CNNs add a fully convolutional network on top of the features of the CNN to create a **Region Proposal Network**.
- The authors of the paper state “The Region Proposal Network **slides a window** over the **features** of the CNN. At each window location, the network outputs a **score** and a **bounding box per anchor** (hence  $4k$  box coordinates where  $k$  is the number of anchors).”
- After each pass of this sliding window, it outputs  $k$  potential bounding boxes and a **score** or **confidence** of how good this box is expected to be.



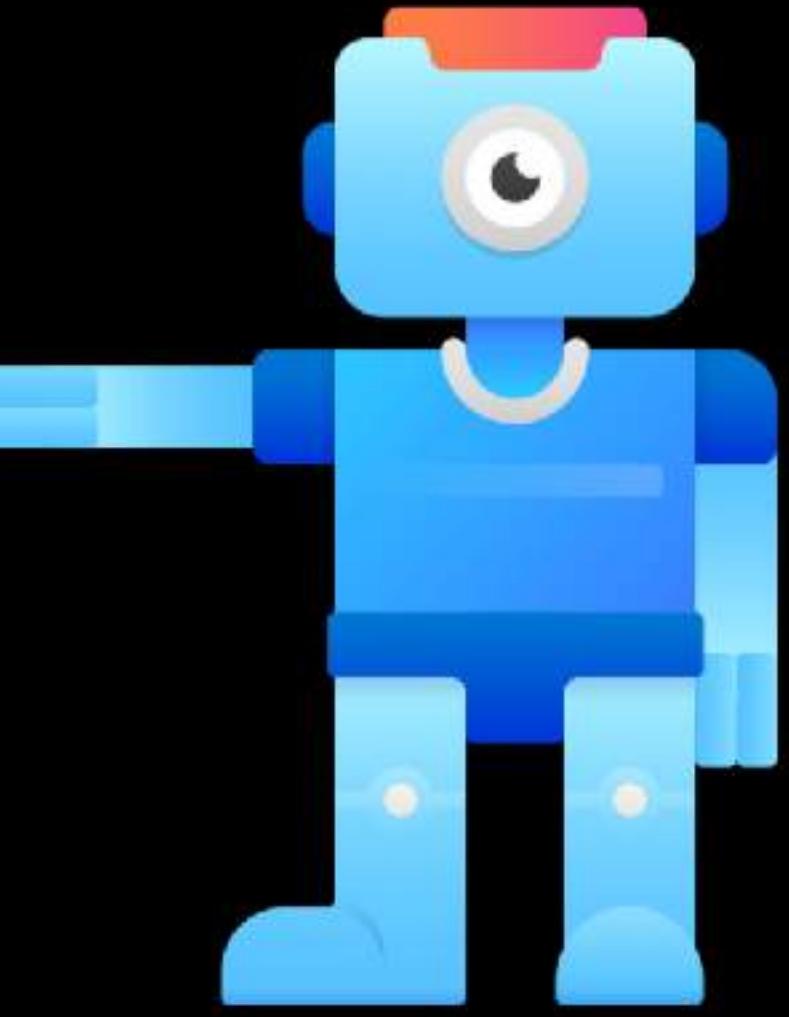


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Introduction to Single Shot Detectors (SSDs)**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## Introduction to Single Shot Detectors (SSDs) [Overview of SSD Object Detectors](#)

# Single Shot Detectors (SSDs)

- We've just discussed the R-CNN family and we've seen how successful they can be.
- However, their weakness is that their inference time performance is still not optimal, running typically at 7 fps at best on powerful hardware.
- SSDs aim to improve this speed by eliminating the need for the Region Proposal Network. Hence the name ‘Single Shot’.
- It is a **One-Stage** Object Detector whereas R-CNNs are **Two-Stage**.

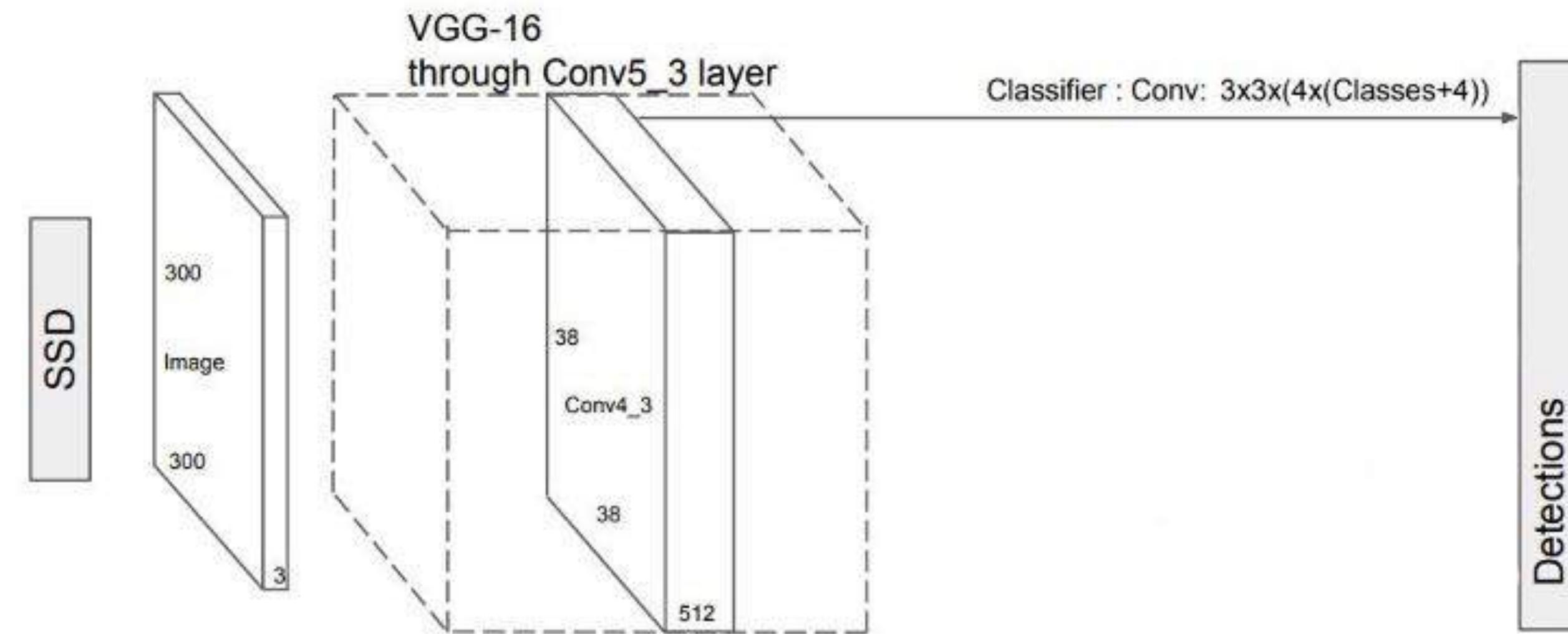
System	VOC2007 test <i>mAP</i>	FPS (Titan X)	Number of Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	~6000	~1000 x 600
YOLO (customized)	63.4	45	98	448 x 448
SSD300* (VGG16)	77.2	46	8732	300 x 300
SSD512* (VGG16)	79.8	19	24564	512 x 512

# How do SSDs Improve Speed?

- SSD's use multi-scale features and default boxes as well as dropping the resolution images to improve speed.
- SSD's do not resample pixels or features for bounding box hypotheses and is as accurate as approaches that do.
- This allows SSD's to achieve real-time speed with almost no drop (sometimes even improved) accuracy.

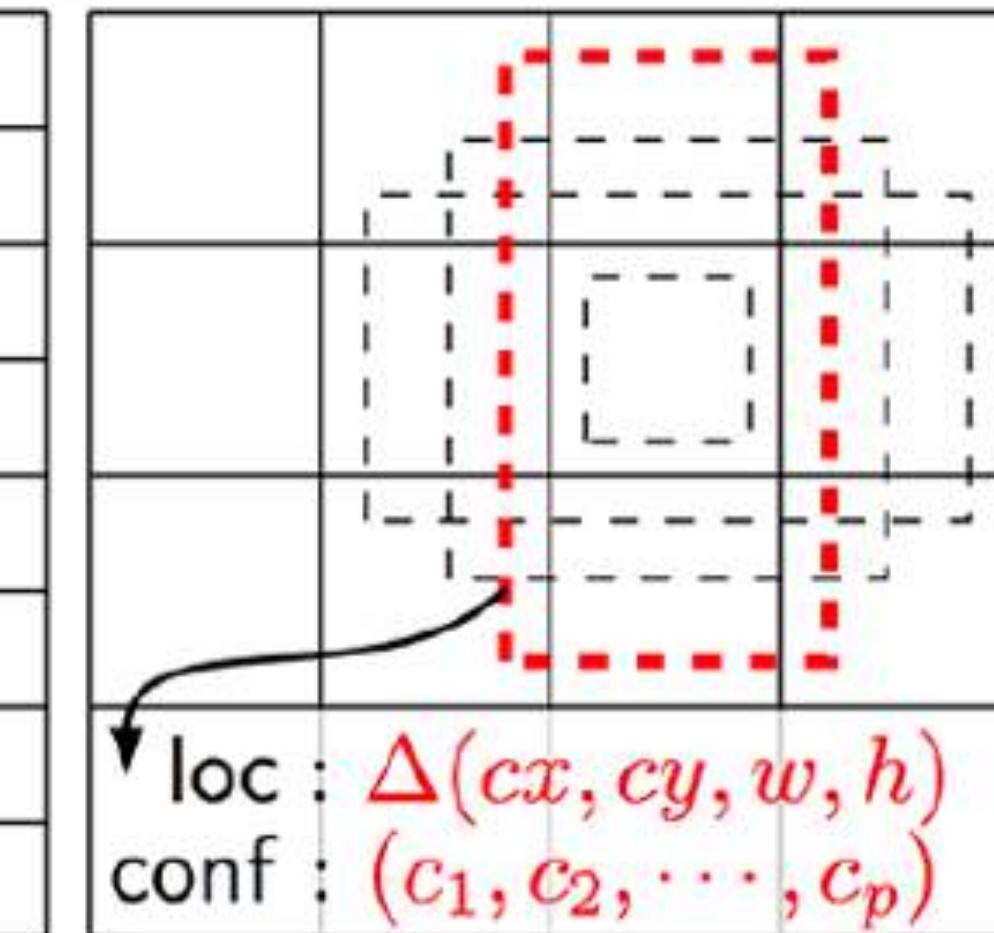
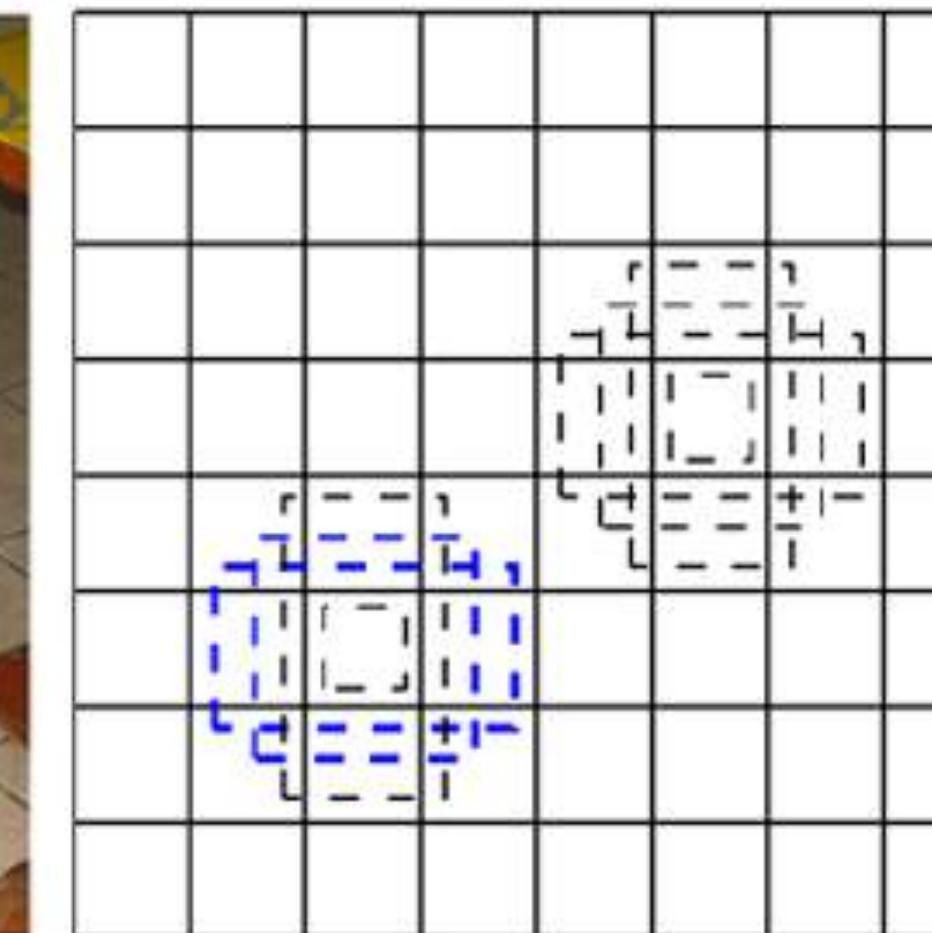
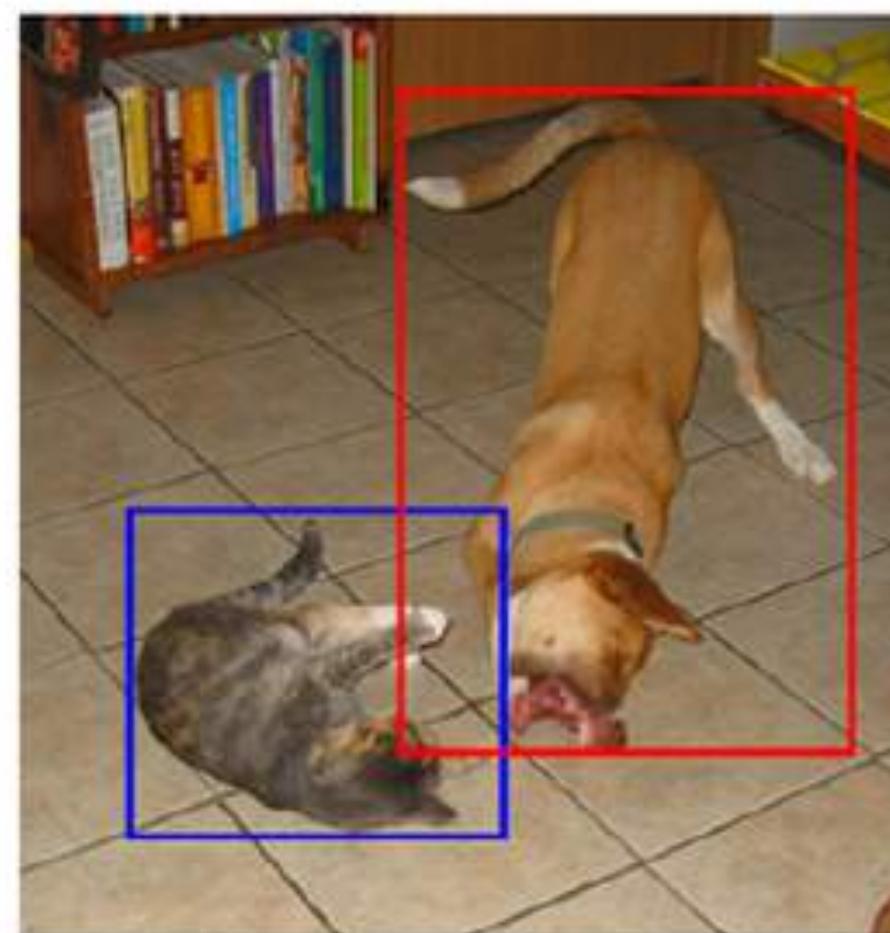
# How do SSDs Improve Speed?

- SSD's are composed of two main parts:
  - **Feature Map Extractor** (VGG16 was used in the published paper but ResNet or DenseNet may provide better results)
  - **Convolution Filter** for Object Detection



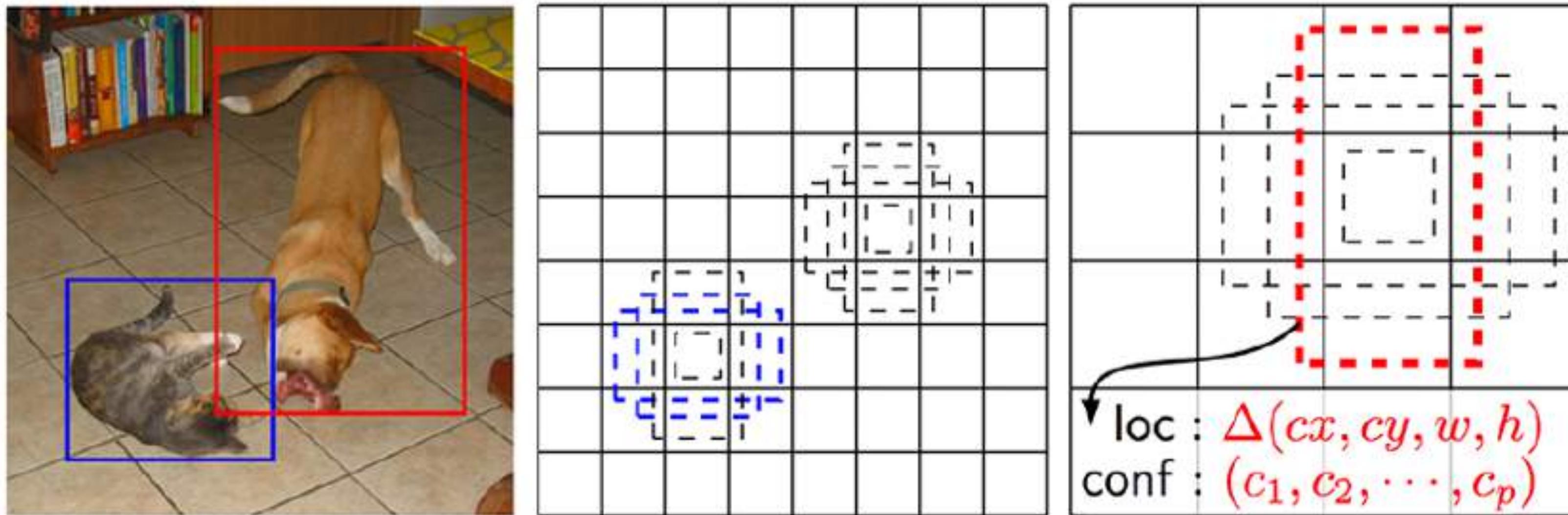
# Discretising Feature Maps

- Using the VGG16's **CONV4\_3** Layer, it makes 4 to 6 (user set) object predictions (shown below) for each cell.
- It predicts the **class scores** and adds one extra for no object being found.
- **Fewer cells allow larger objects** to be detected (e.g. the dog and right rightmost diagram with the red box) and **large number of cells allow more granular detection of smaller objects** (e.g. the cat).



# Using VGG16 to Extract Feature Maps

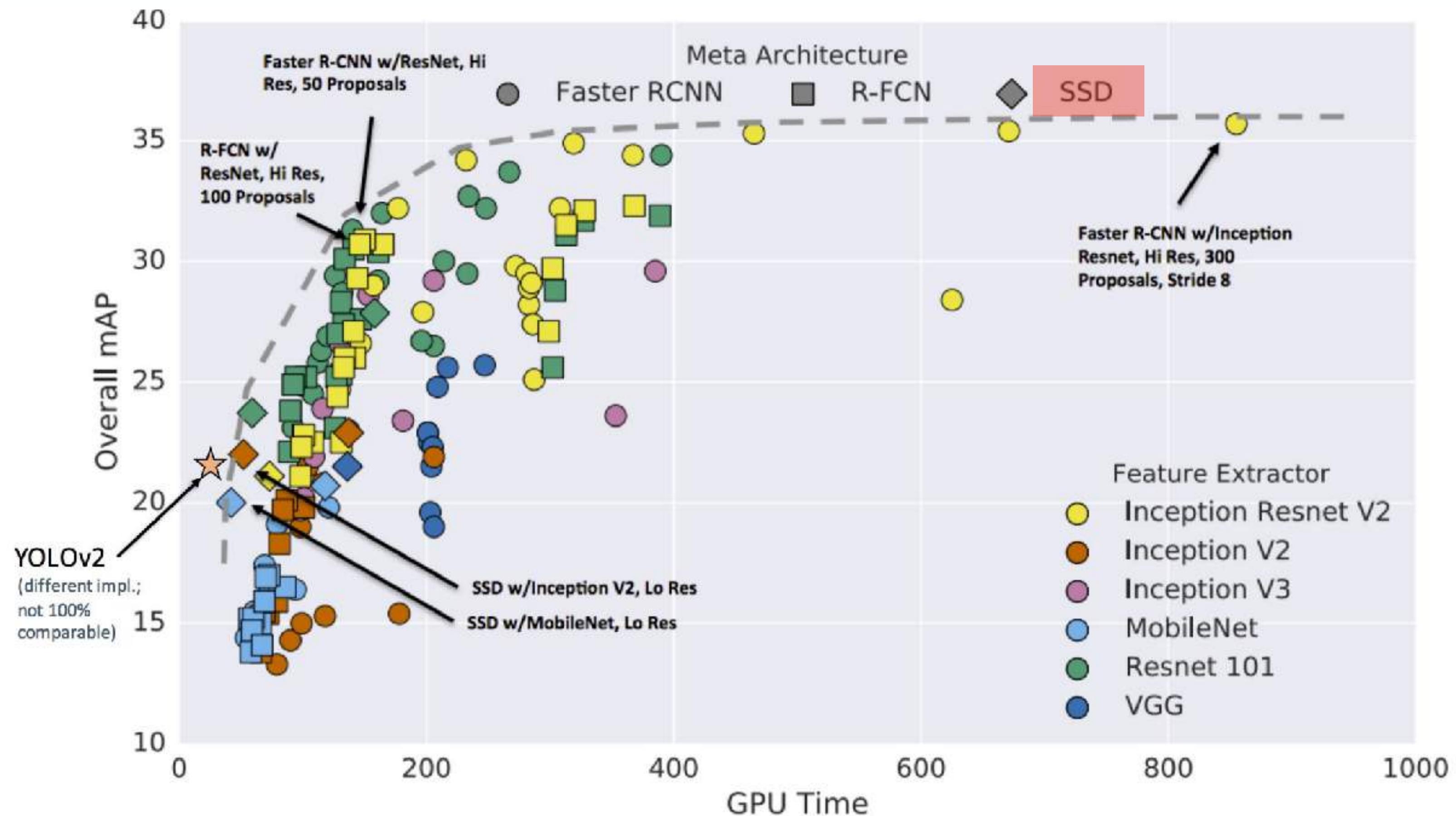
- For each **bounding box** we obtain the **probabilities** of all classes within the region.
- This allows us to produce **overlapping boxes** where multiple objects occur.



# MultiBox & Loss Functions

- Making multiple predictions for boundary boxes and confidence scores is called **MultiBox**.
- Loss Functions used in Training:
  - **Class predictions** - Categorical cross-entropy
  - **Location or localisation loss** - Smooth L1-loss. SSD only penalizes predictions of positive matches and ignores negative matches as we want to get our positives as close to the ground truth as possible.

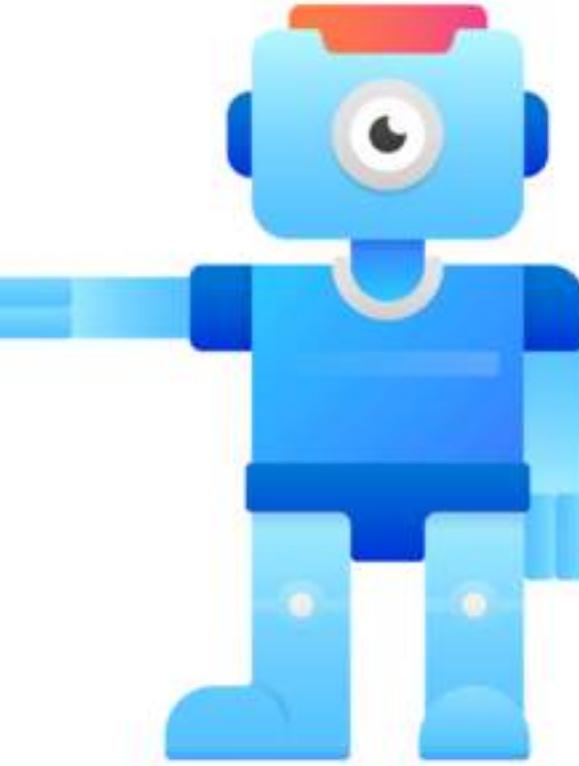
# SSD Performance



Slide from Ross Girshick's [CVPR 2017 Tutorial](#), Original Figure from Huang et al

# Take Aways

- SSDs are faster than Faster R-CNN but less accurate in detecting small objects.
- Accuracy increases if we increase the number of default boxes as well as better designed boxes
- Multi-scale feature maps improve detection at varying scales.
- Faster than YOLO (v2), as accurate as Faster R-CNN
- Predicts categories and box offsets
- Uses small convolutional filters applied to feature maps
- Makes predictions using feature maps of different scales
- For training, requires that ground truth data is assigned to specific outputs in the fixed set of detector outputs
- Slower but more accurate than YOLO
- Faster but less accurate than Faster R-CNN

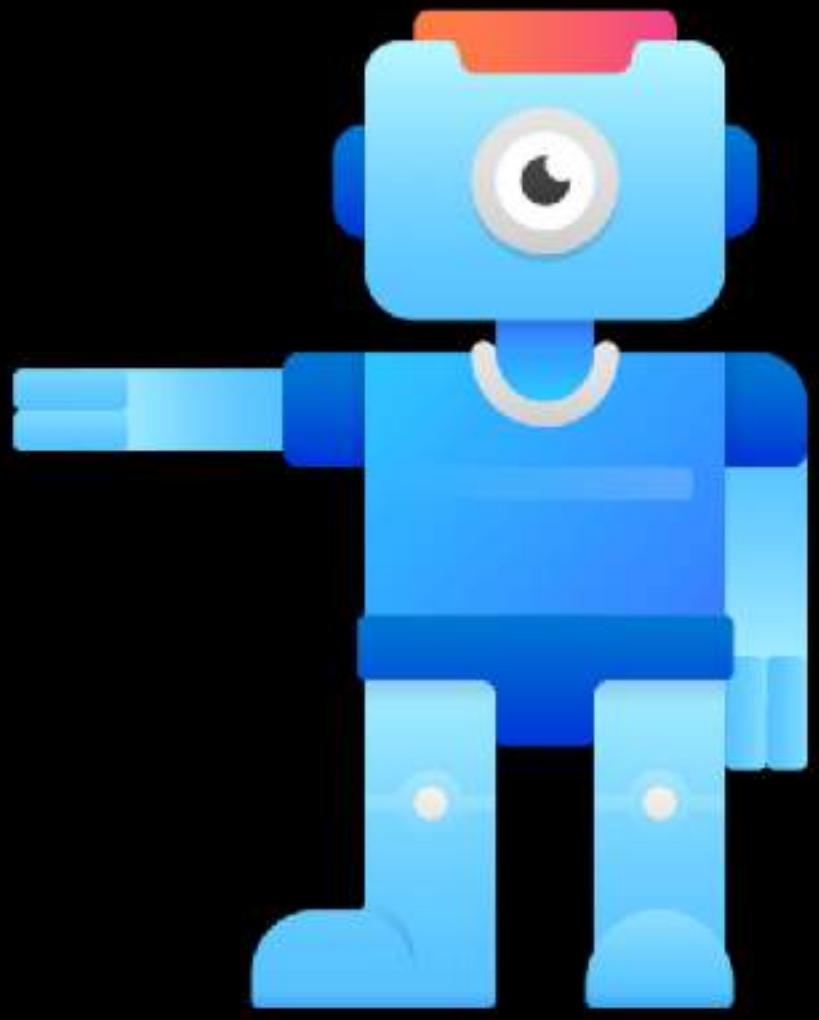


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Introduction to the YOLO Object Detectors**



# MODERN COMPUTER VISION

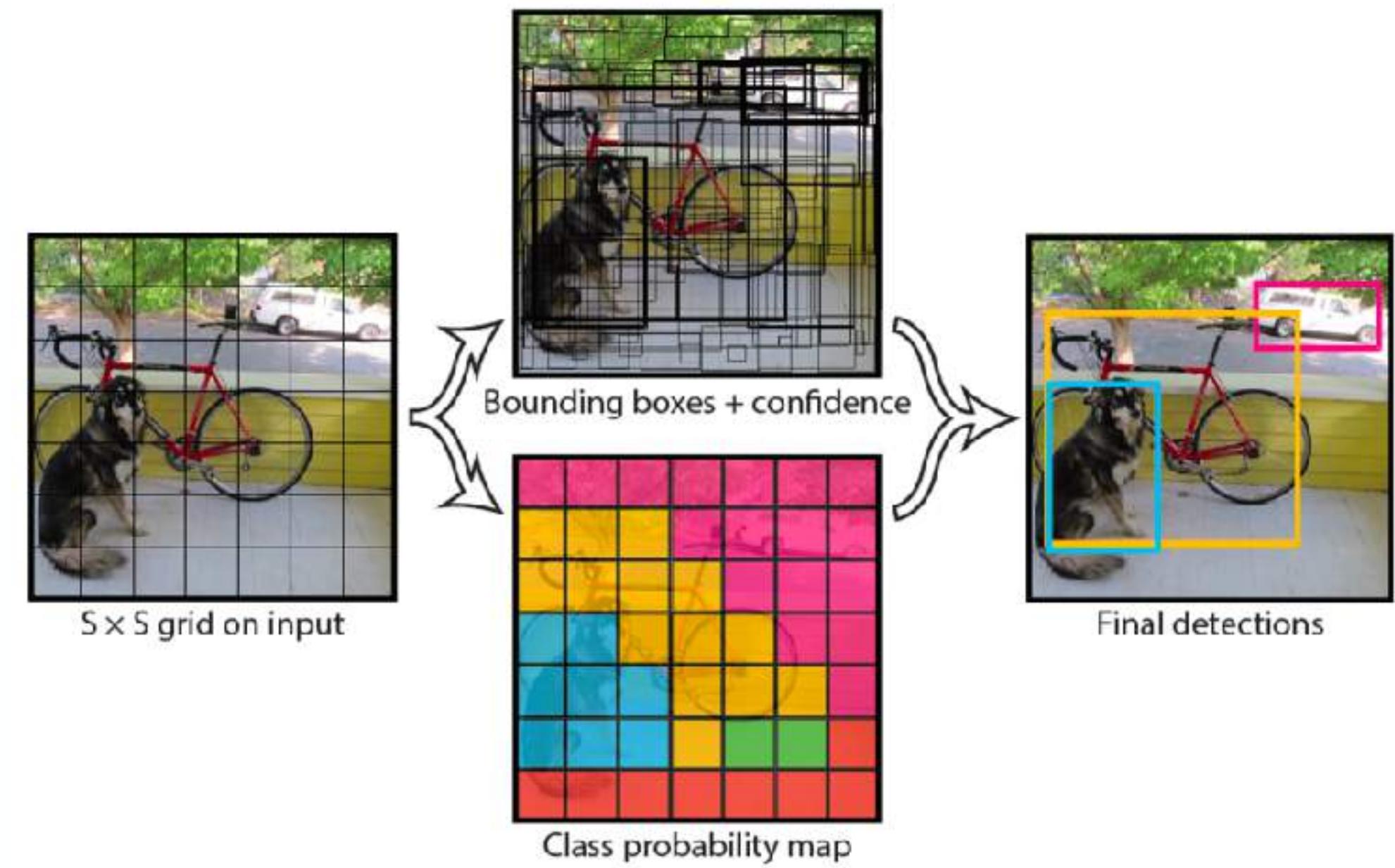
BY RAJEEV RATAN

## Introduction to the YOLO Object Detectors

Overview of the R-CNNs family of Models

# YOLO - You Only Look Once

- Developed in 2015 by brilliant researchers **Joseph Redmon** and **Ali Farad**.
- YOLO is a **single stage** detector strategy (hence the You Only Look Once name) object detector algorithm.
- R-CNNs and Fast/Faster R-CNNs tended to be quite accurate, their speed was a **huge drawback**, getting single digits FPS even on powerful GPUs.
- SSDs improved this dramatically, using their single stage detector technique, however they **were not as accurate as R-CNNs**.
- **YOLO (now on YOLOv4 and v5)** attempted to solve both problems and strike a balance **between speed and accuracy**.



# The many flavours of YOLO

- YOLO has had tremendous success in real world applications and has sprung many different versions or flavours.
- Examples are, TinyYOLO, YOLOv2,v3,v4, v5 and YOLOx scaled-YOLO, YOLO with various backends etc.
- The most popular flavour of YOLO used in industry was YOLOv3 now it's the Ultralytics implementation of YOLOv5.
- Pretrained YOLO models (on COCO) are readily available and easy to use!

```

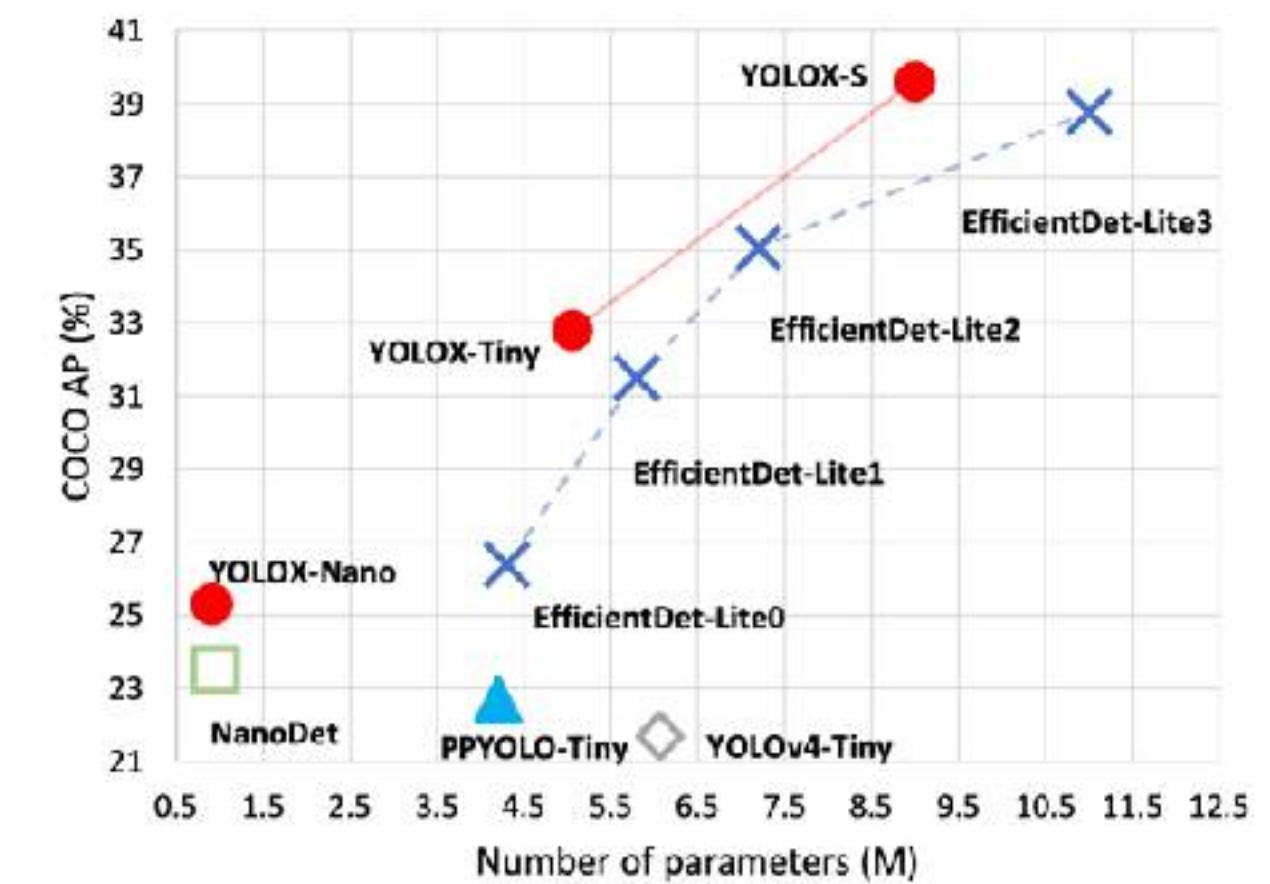
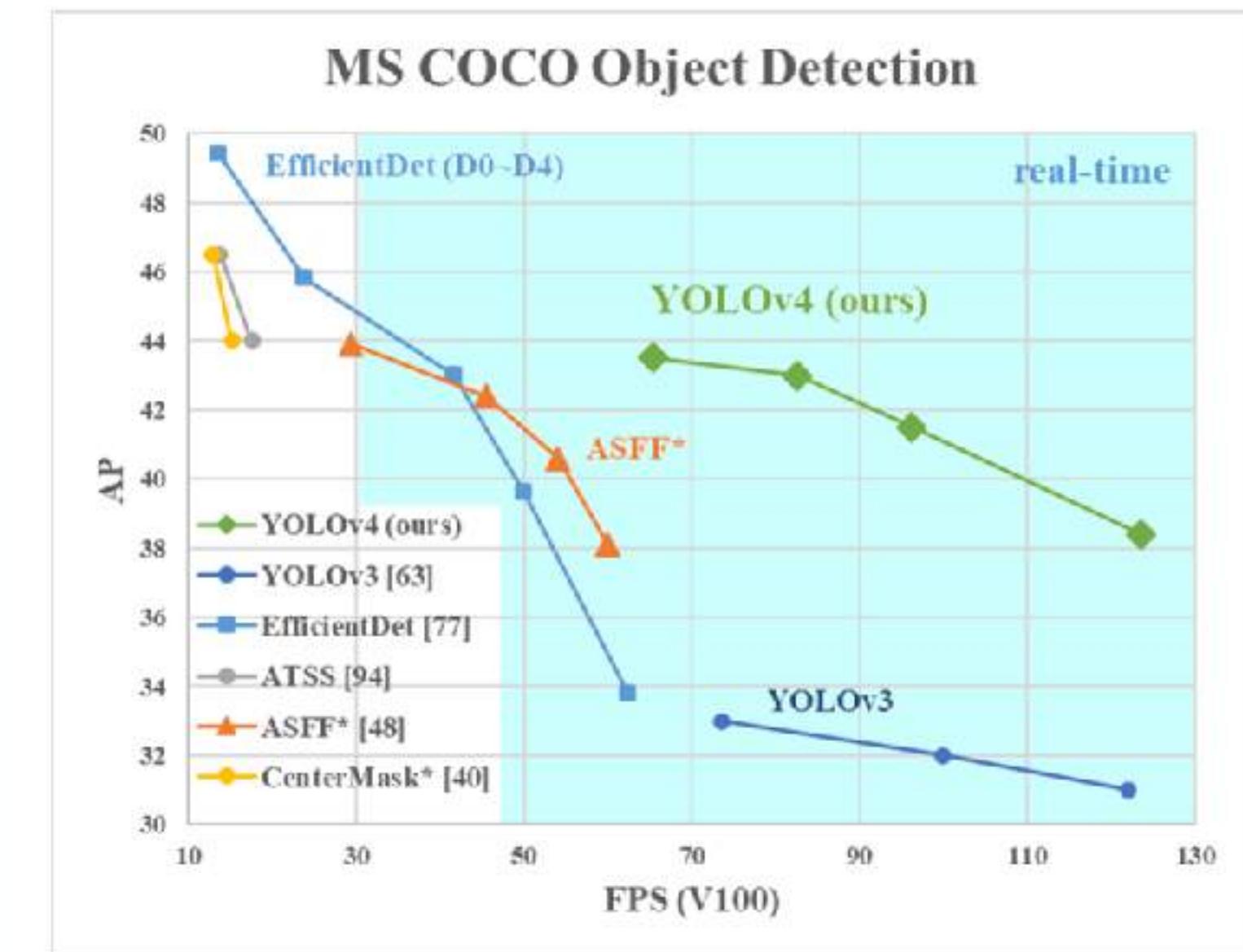
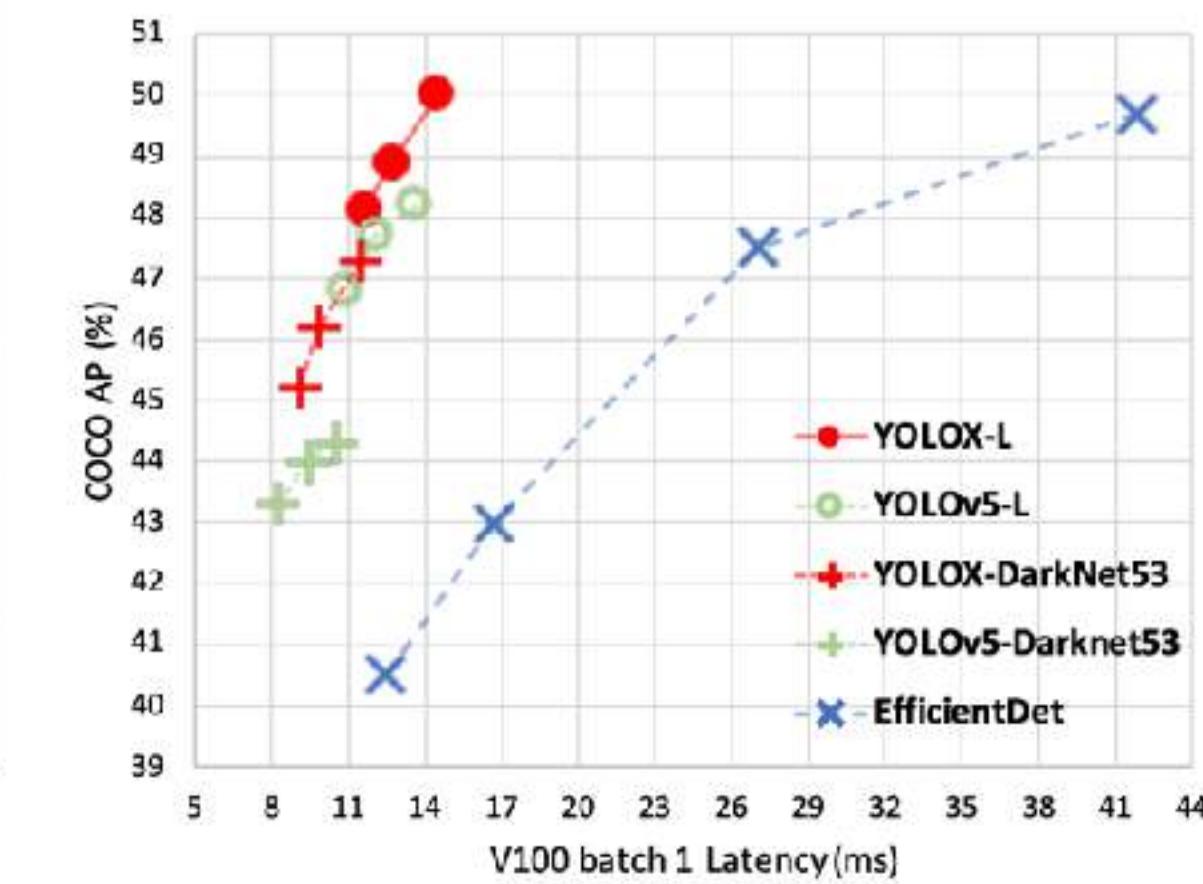
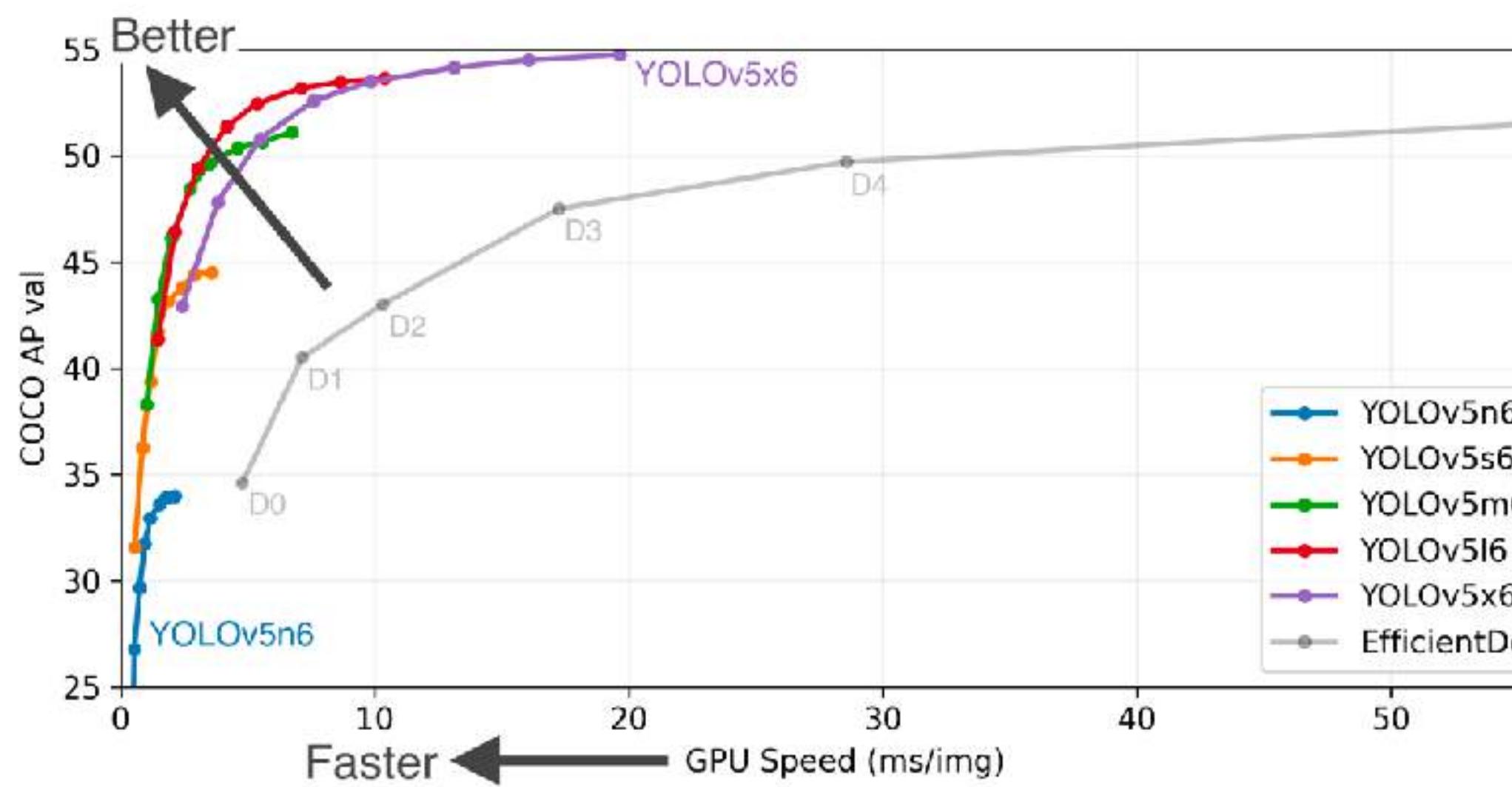
1 person
2 bicycle
3 car
4 motorbike
5 aeroplane
6 bus
7 train
8 truck
9 boat
10 traffic light
11 fire hydrant
12 stop sign
13 parking meter
14 bench
15 bird
16 cat
17 dog
18 horse
19 sheep
20 cow
21 elephant
22 bear
23 zebra
24 giraffe
25 backpack
26 umbrella
27 handbag
28 tie
29 suitcase
30 frisbee
31 skis
32 snowboard
33 sports ball
34 kite
35 baseball bat
36 baseball glove
37 skateboard
38 surfboard
39 tennis racket
40 bottle
41 wine glass
42 cup
43 fork
44 knife
45 spoon
46 bowl
47 banana
48 apple
49 sandwich
50 orange
51 broccoli
52 carrot
53 hot dog
54 pizza
55 donut
56 cake
57 chair
58 sofa
59 pottedplant
60 bed
61 diningtable
62 toilet
63 tvmonitor
64 laptop
65 mouse
66 remote
67 keyboard
68 cell phone
69 microwave
70 oven
71 toaster
72 sink
73 refrigerator
74 book
75 clock
76 vase
77 scissors
78 teddy bear
79 hair drier
80 toothbrush

```



# How Does YOLO Compare to Others?

- Currently the best performing YOLO models are **YOLOv4**, **YOLOv5** and **YOLOX**.



# How Does YOLO Work?

- Prior detection systems **re-purpose classifiers** or localisers to perform detection. They apply the model to an image at **multiple locations and scales**. High scoring **regions** of the image are considered **detections**.
- Instead, YOLO applies a **single neural network to the full image**. This network divides the image into **regions** and predicts bounding boxes and **probabilities for each region**. These bounding boxes are weighted by the predicted probabilities.
- This has several advantages over classifier-based systems.
  - It looks at the whole image at test time so its predictions are informed by global context in the image.
  - It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

# The YOLO Technical Report is Hilarious

- Not all researchers take themselves too seriously and the brilliant minds behind YOLO certainly don't!

## YOLOv3: An Incremental Improvement

Joseph Redmon   Ali Farhadi  
 University of Washington

### Abstract

*We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At  $320 \times 320$  YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP<sub>50</sub> in 51 ms on a Titan X, compared to 57.5 AP<sub>50</sub> in 198 ms by RetinaNet, similar performance but 3.8x faster. As always, all the code is online at <https://pjreddie.com/yolo/>.*

### 1. Introduction

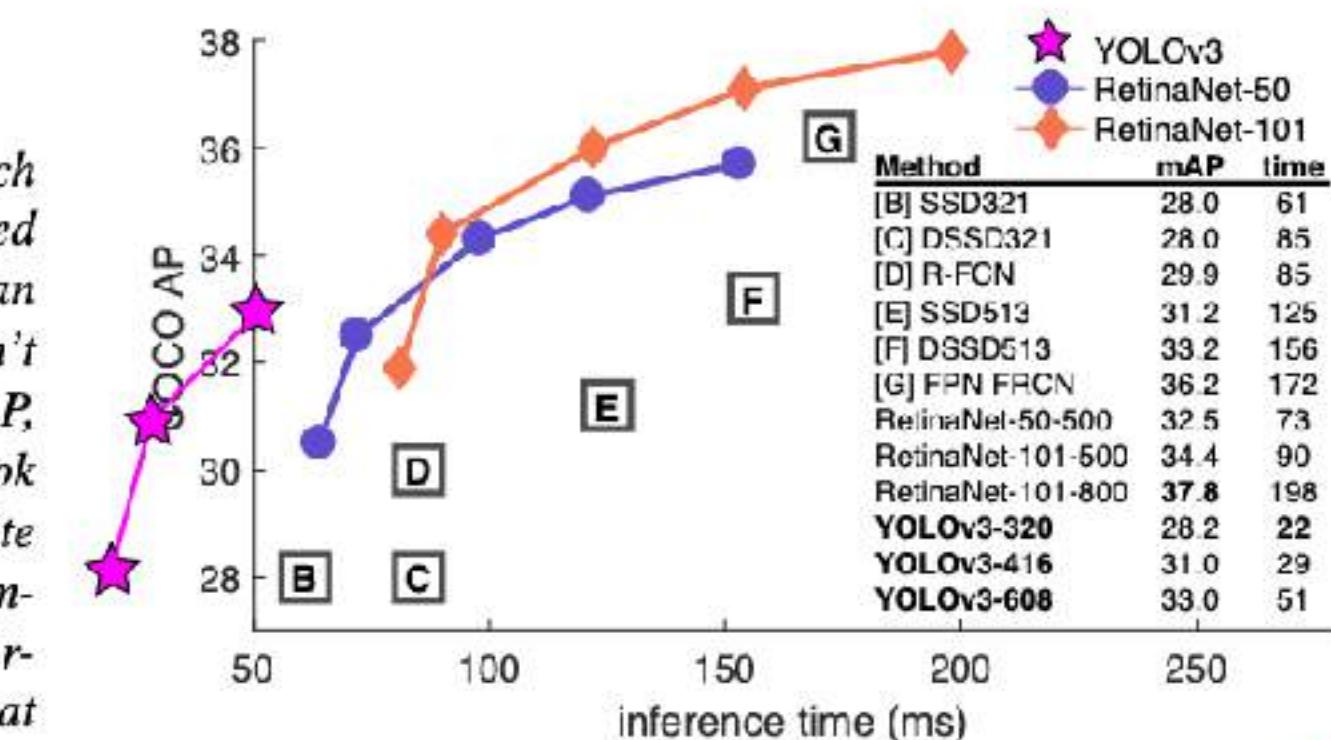


Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

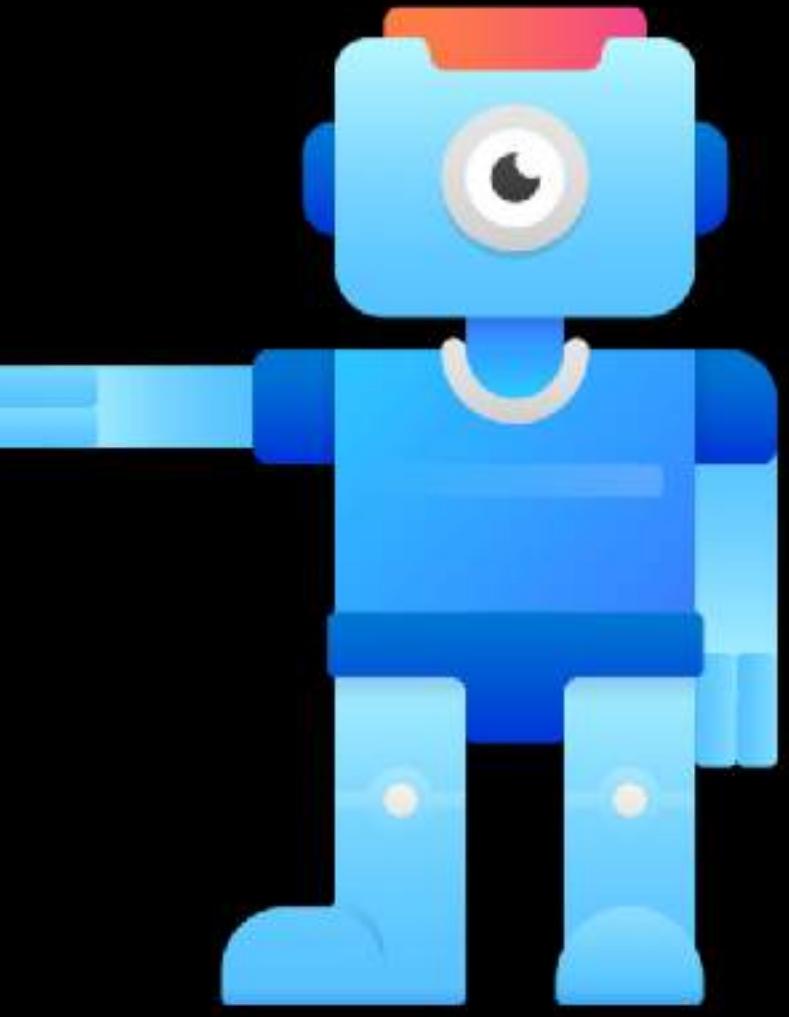


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**How does YOLO work?**



# MODERN COMPUTER VISION

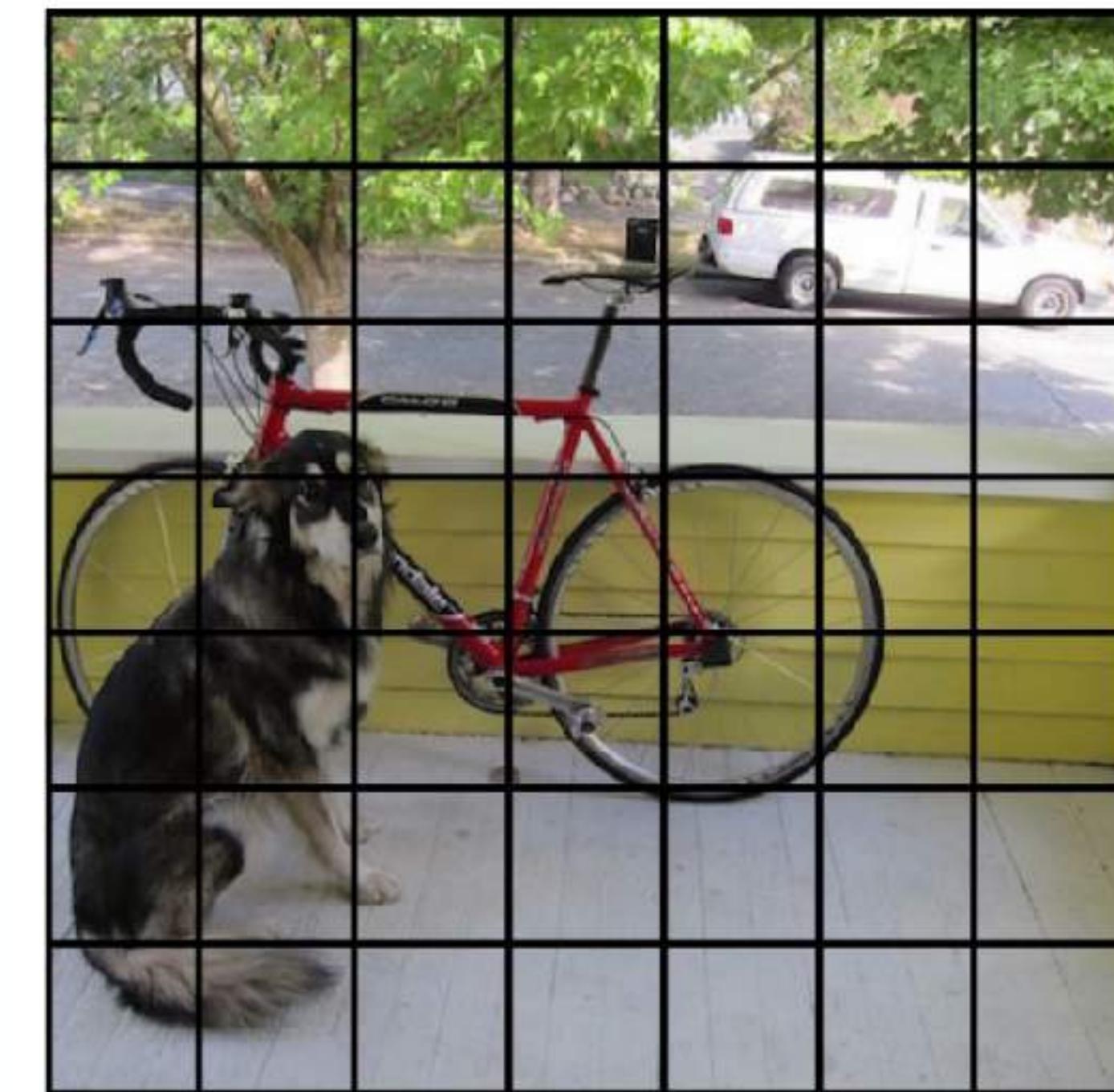
BY RAJEEV RATAN

## How Does YOLO Work?

An overview of how YOLO makes predictions

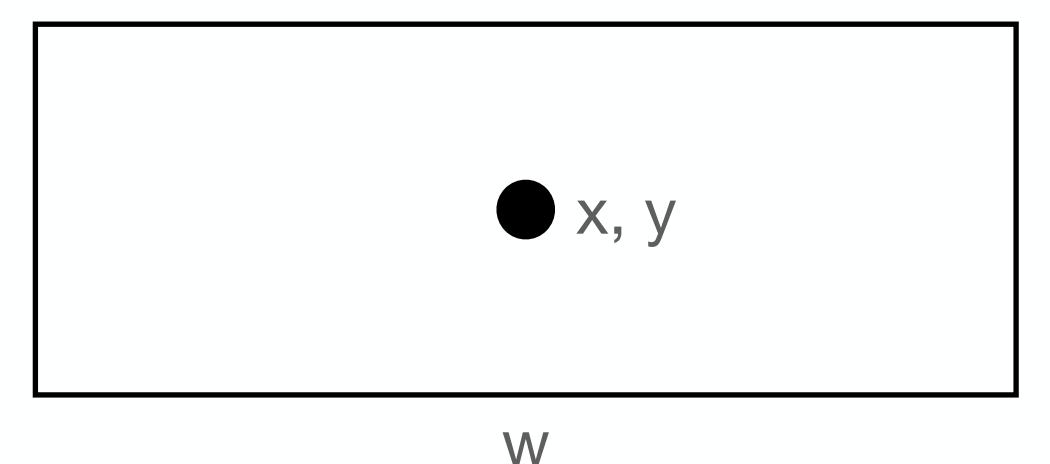
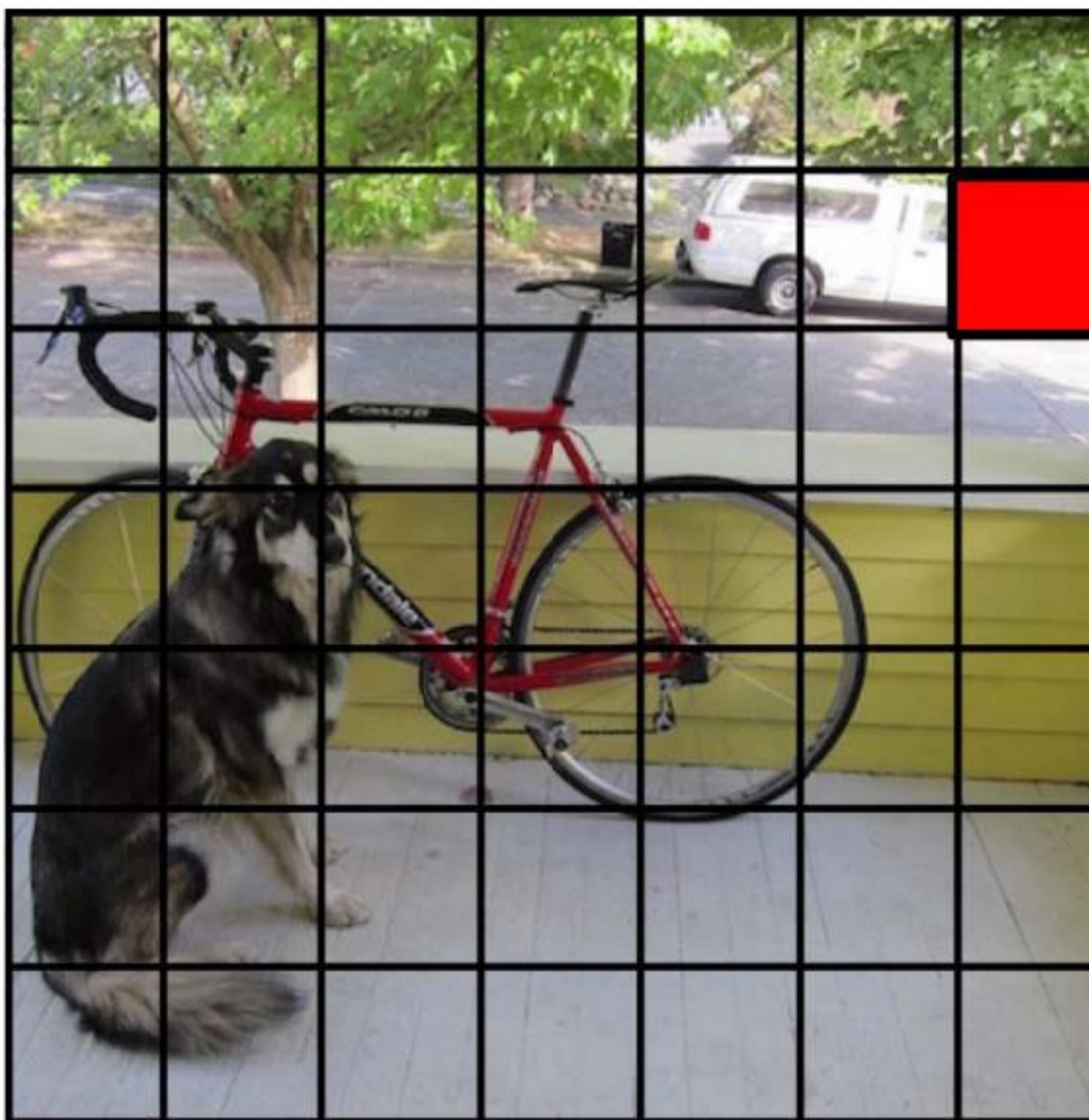
# Split Image into Grid

- Firstly, we split the image into an  $S \times S$  grid, typically a  $7 \times 7$ .



# Each Cell Gives B Boxes

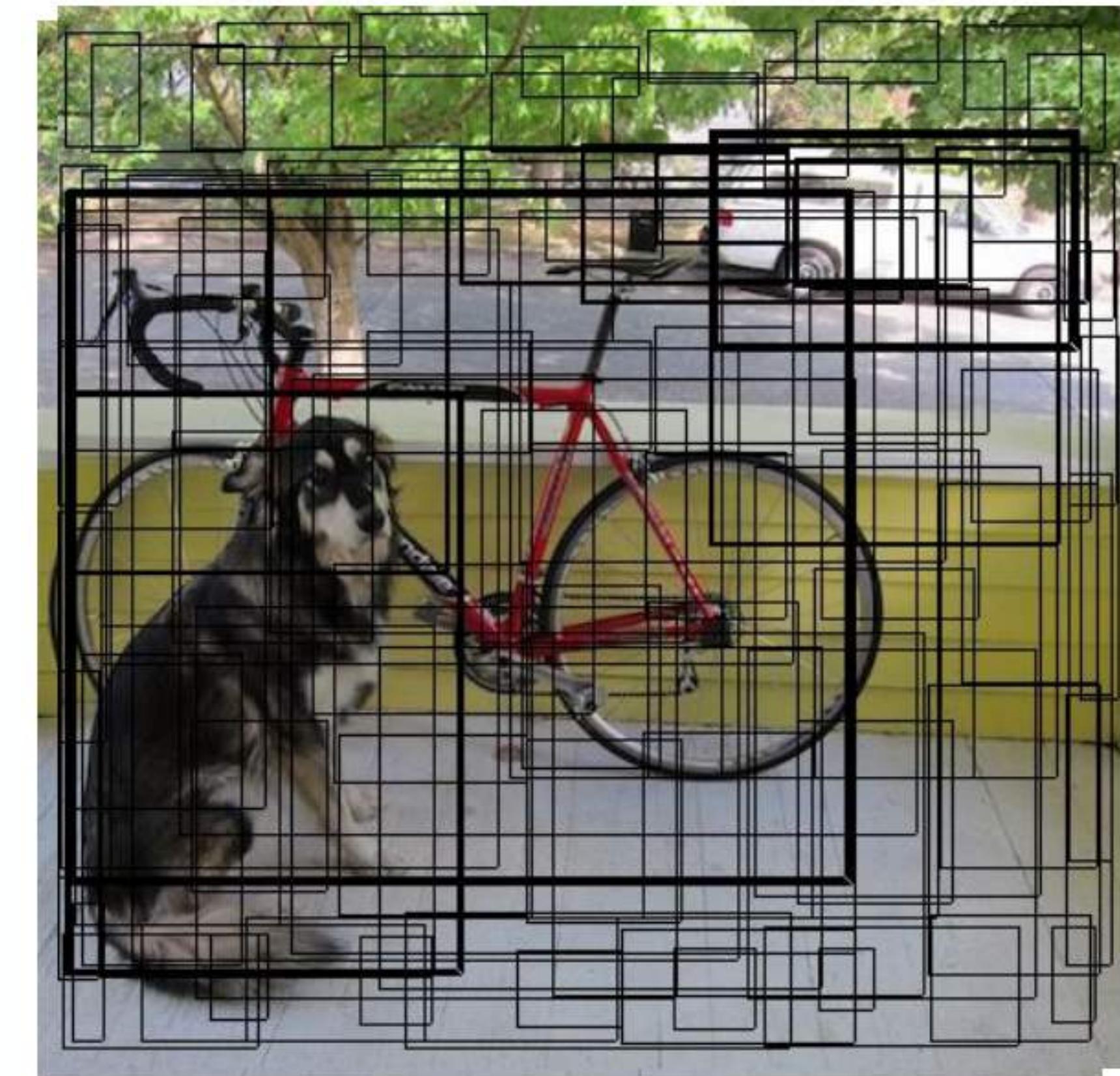
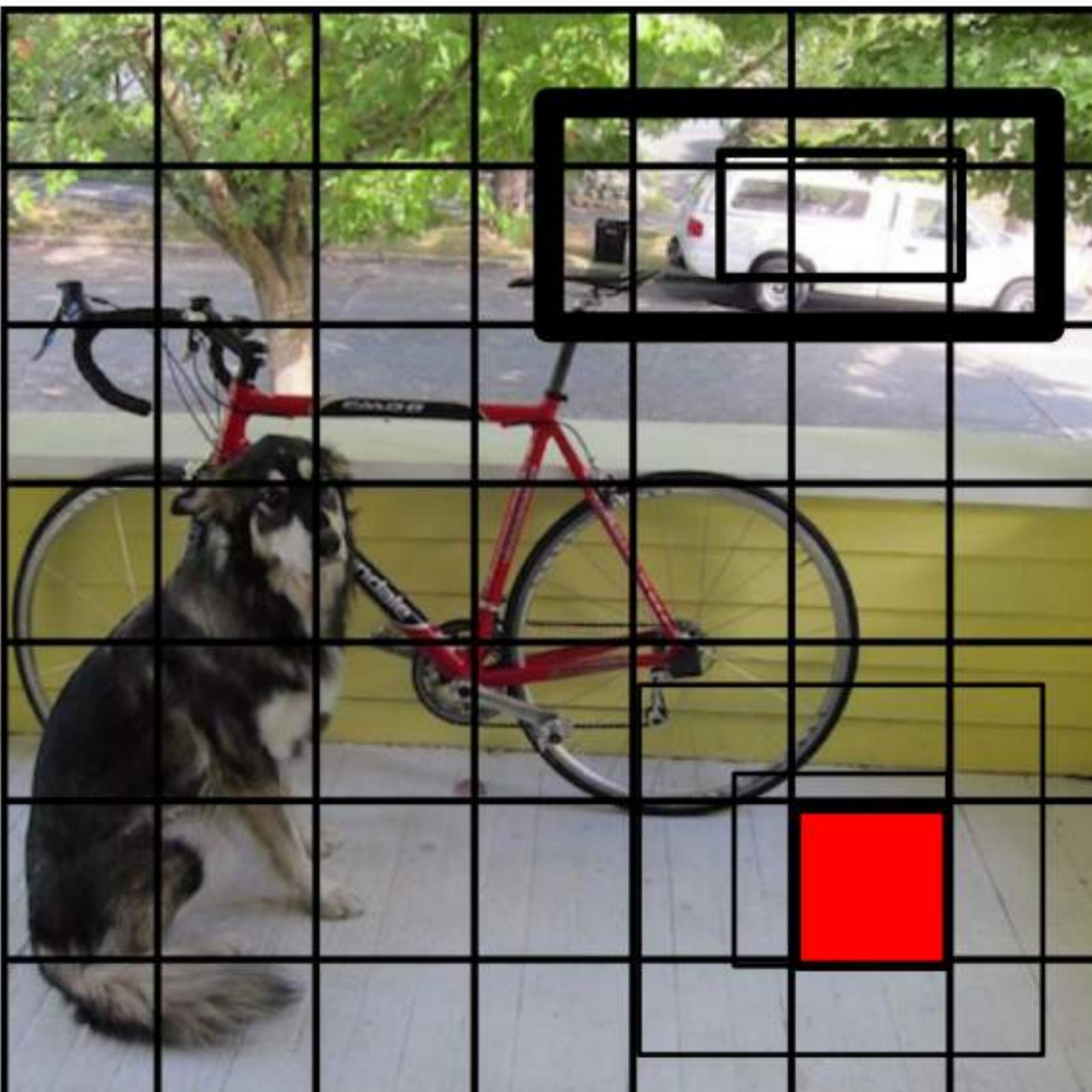
- Next, each cell predicts **B bounding boxes** ( $x, y, w, h$ ) and the confidences of each box having an object i.e the **Probability** that box has an object  $\sim P(\text{Object})$



**B = 2**

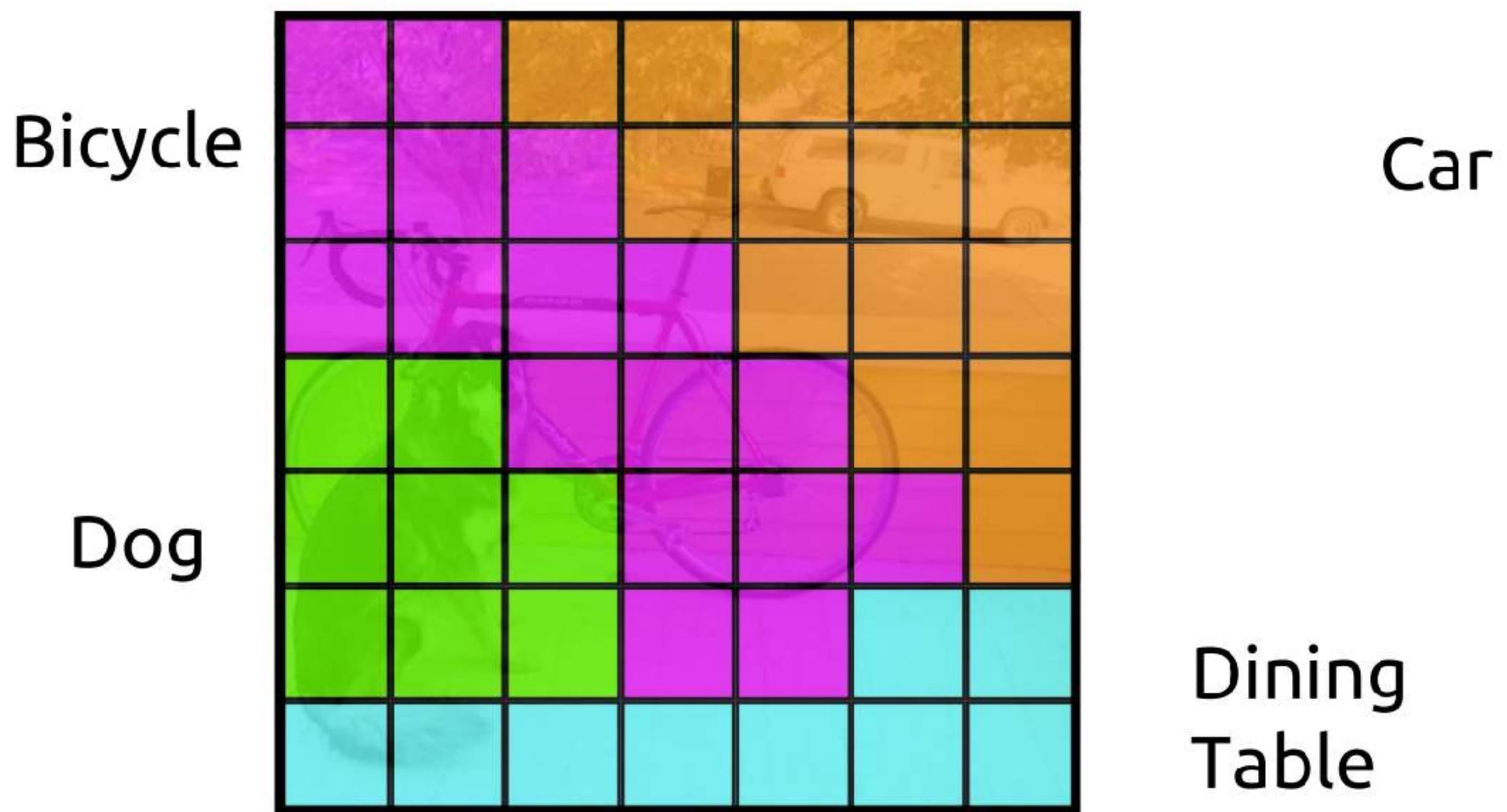
# Generate All Bounding Boxes

- So now for each cell we have B boxes and the associated probabilities of each box having an object.



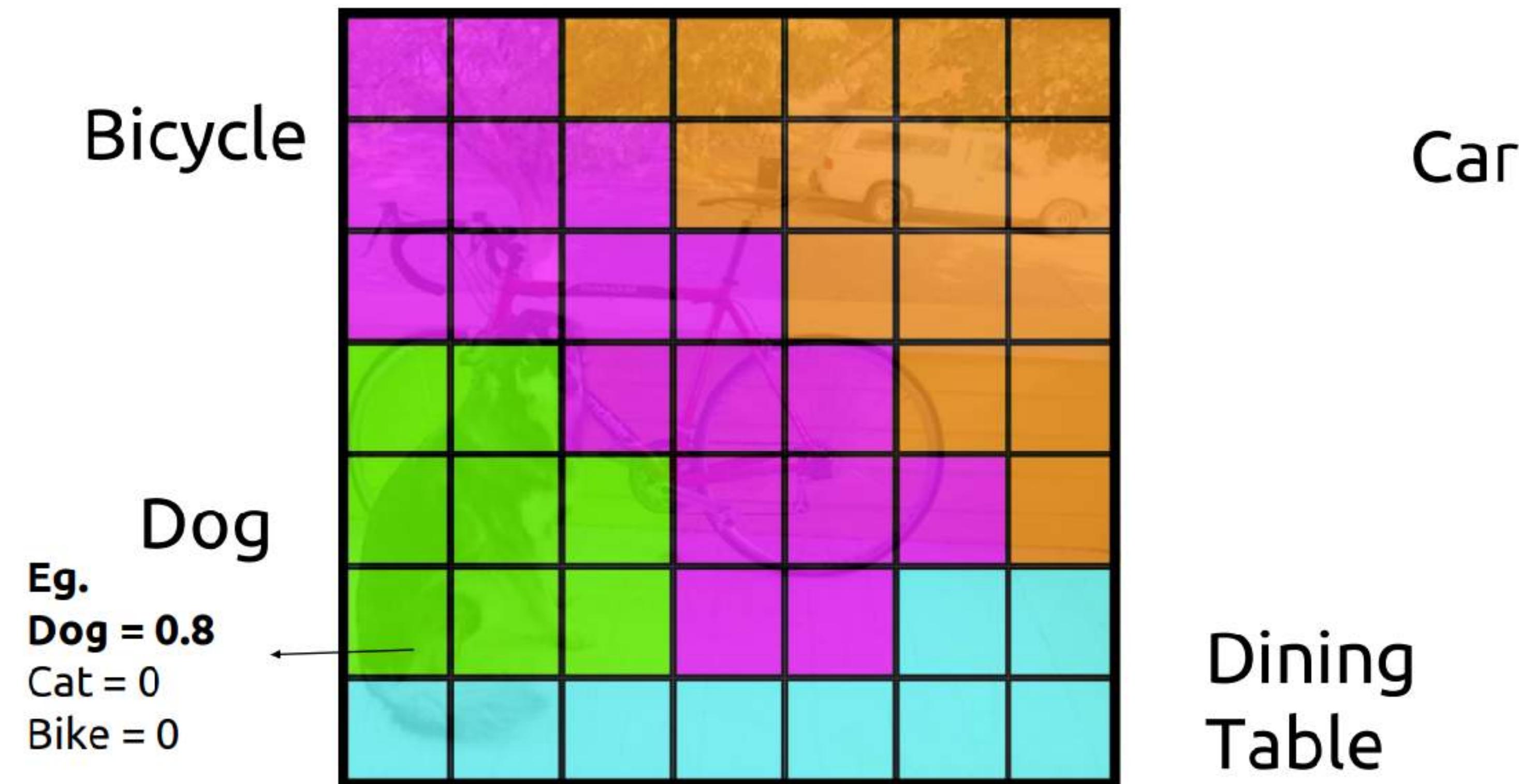
# Class Probability per Cell

- Each cell predicts a class probability.



# Class Probability per Cell

- Each cell provides the probability of the object class e.g.  $P(\text{ Dog} | \text{Object})$



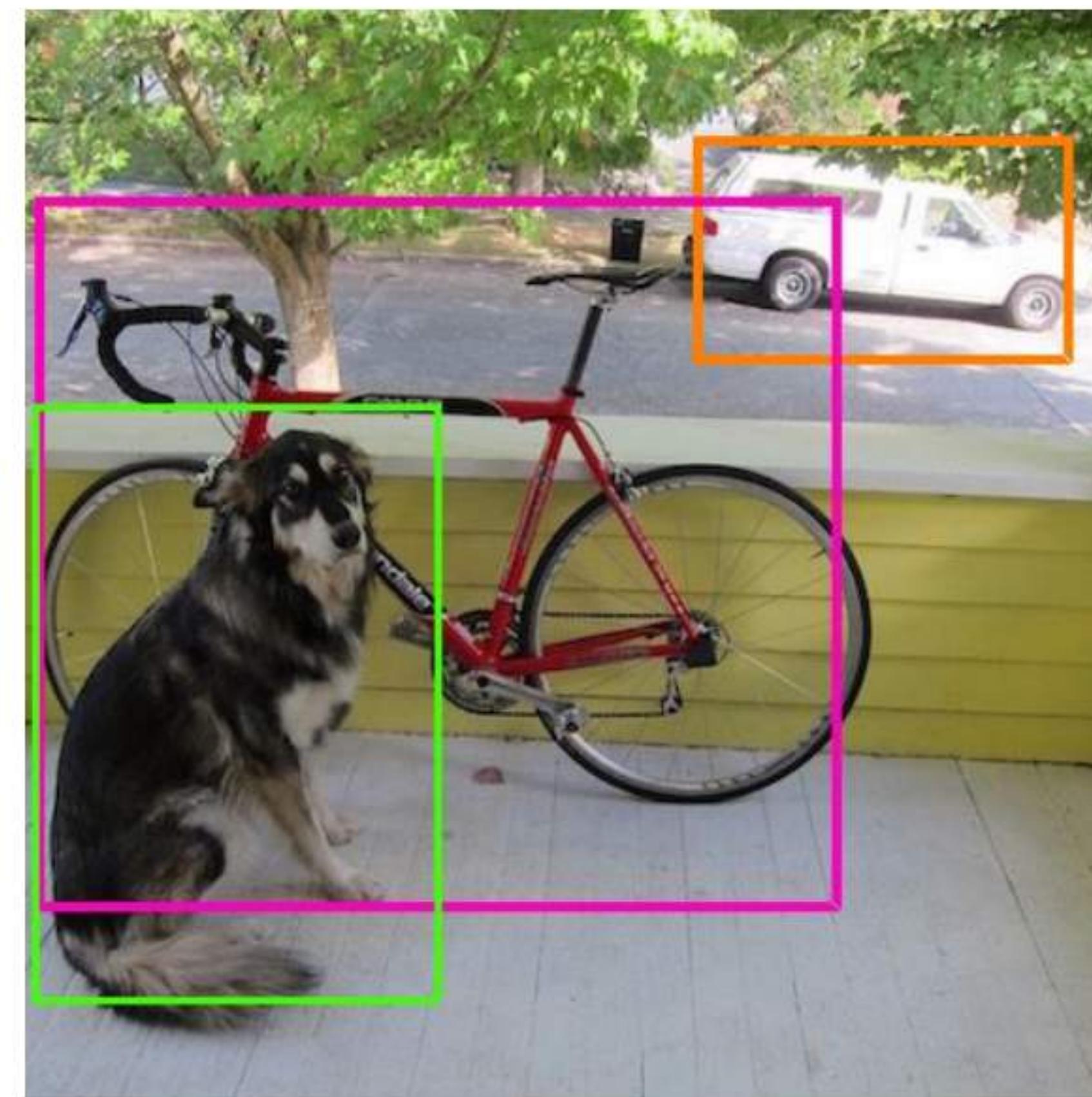
# Combining Box and Class Predictions

- $P(\text{class}) = P(\text{class}|\text{object}) * P(\text{Object})$



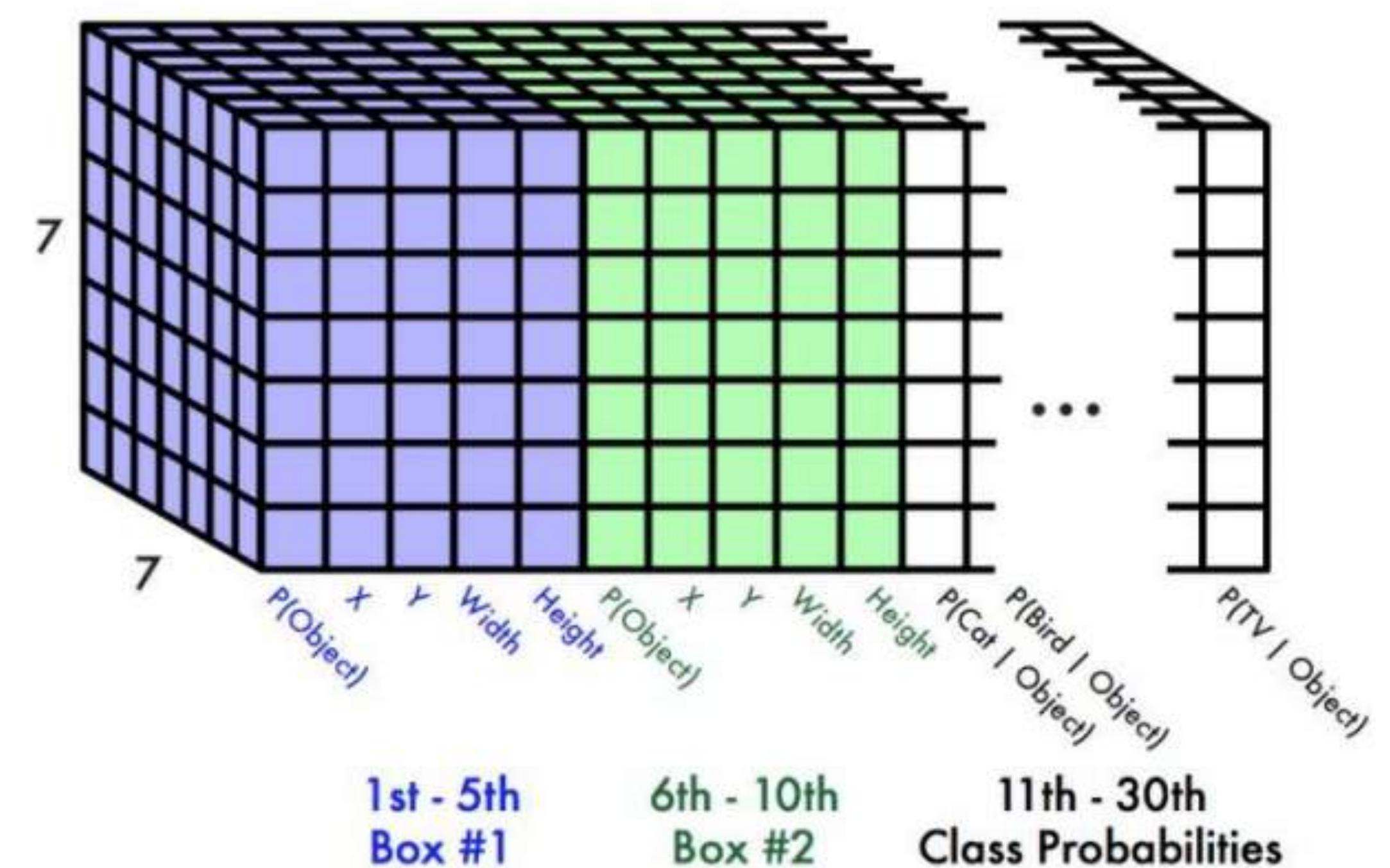
# Apply Non Maximum Suppression (NMS)

- We threshold detections and use Non Maximum Suppression (NMS)

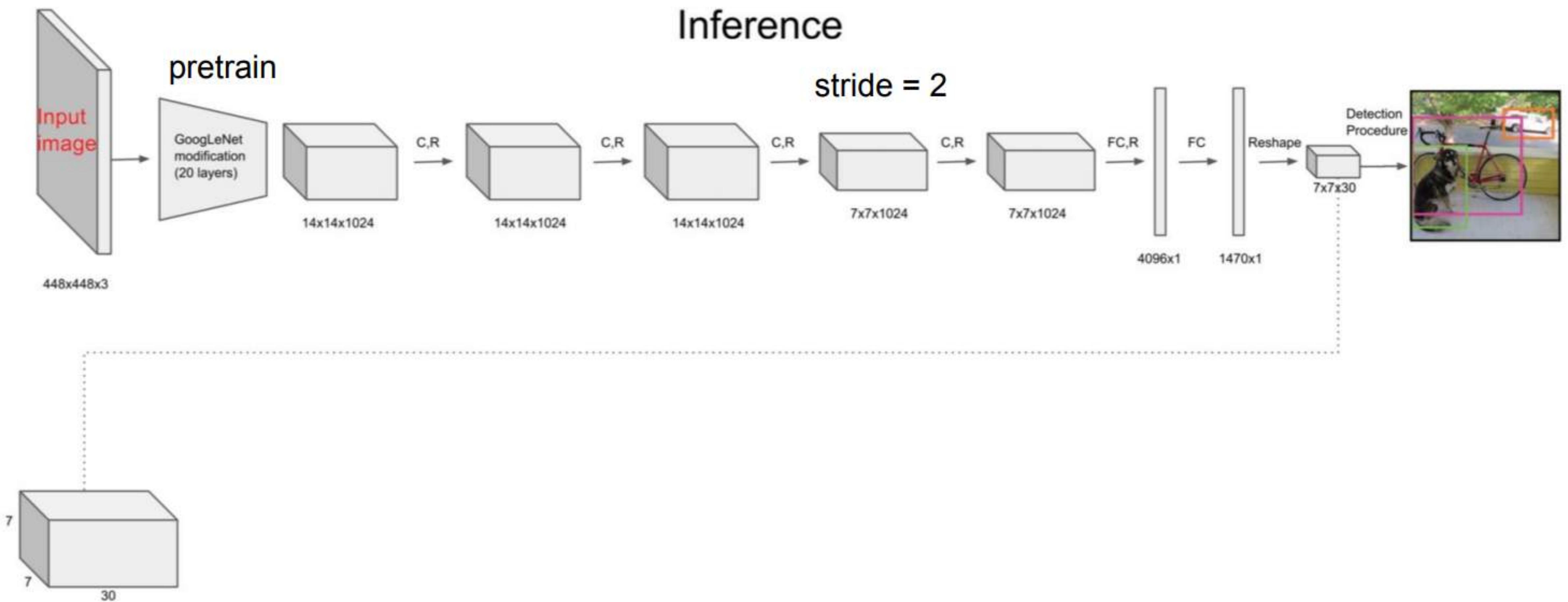


# Output Predictions

- Each cell provides the following predictions:
  - For each bounding box we get:
  - 4 coordinates ( $x, y, w, h$ )
  - 1 confidence value
- A number of class probabilities
- The output forms the dimensions
- $S * S * (B * 5 + C)$



# Inference Overview



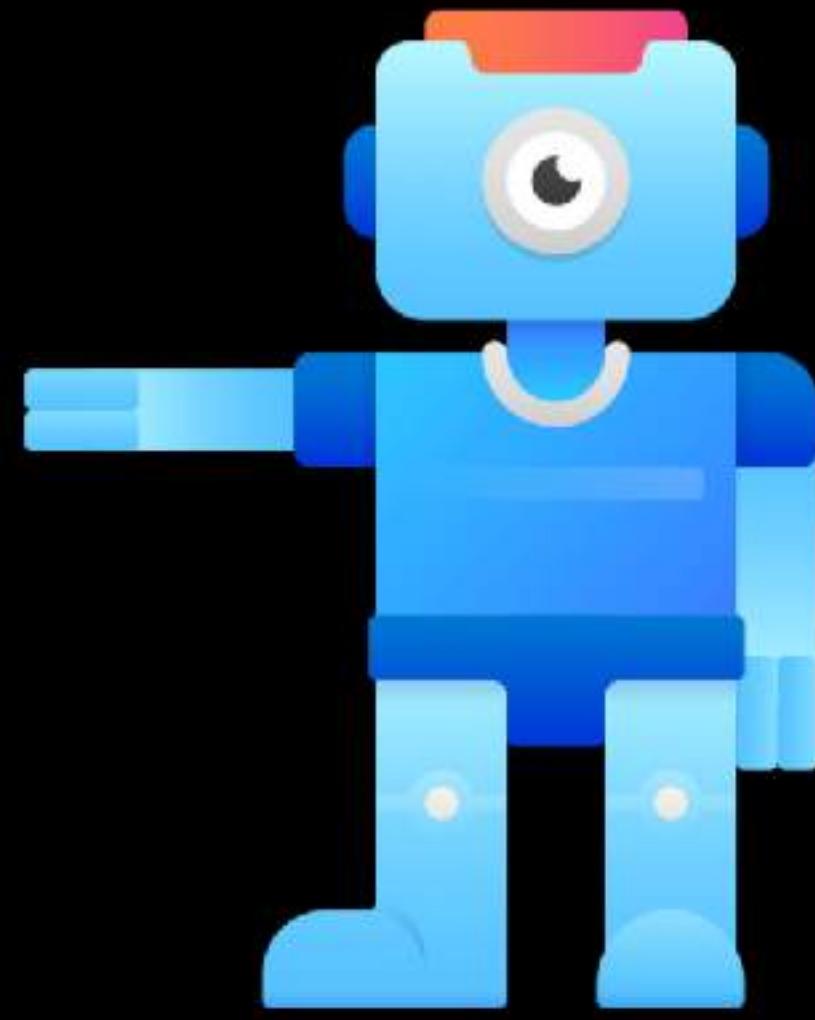


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

Training YOLO



# MODERN COMPUTER VISION

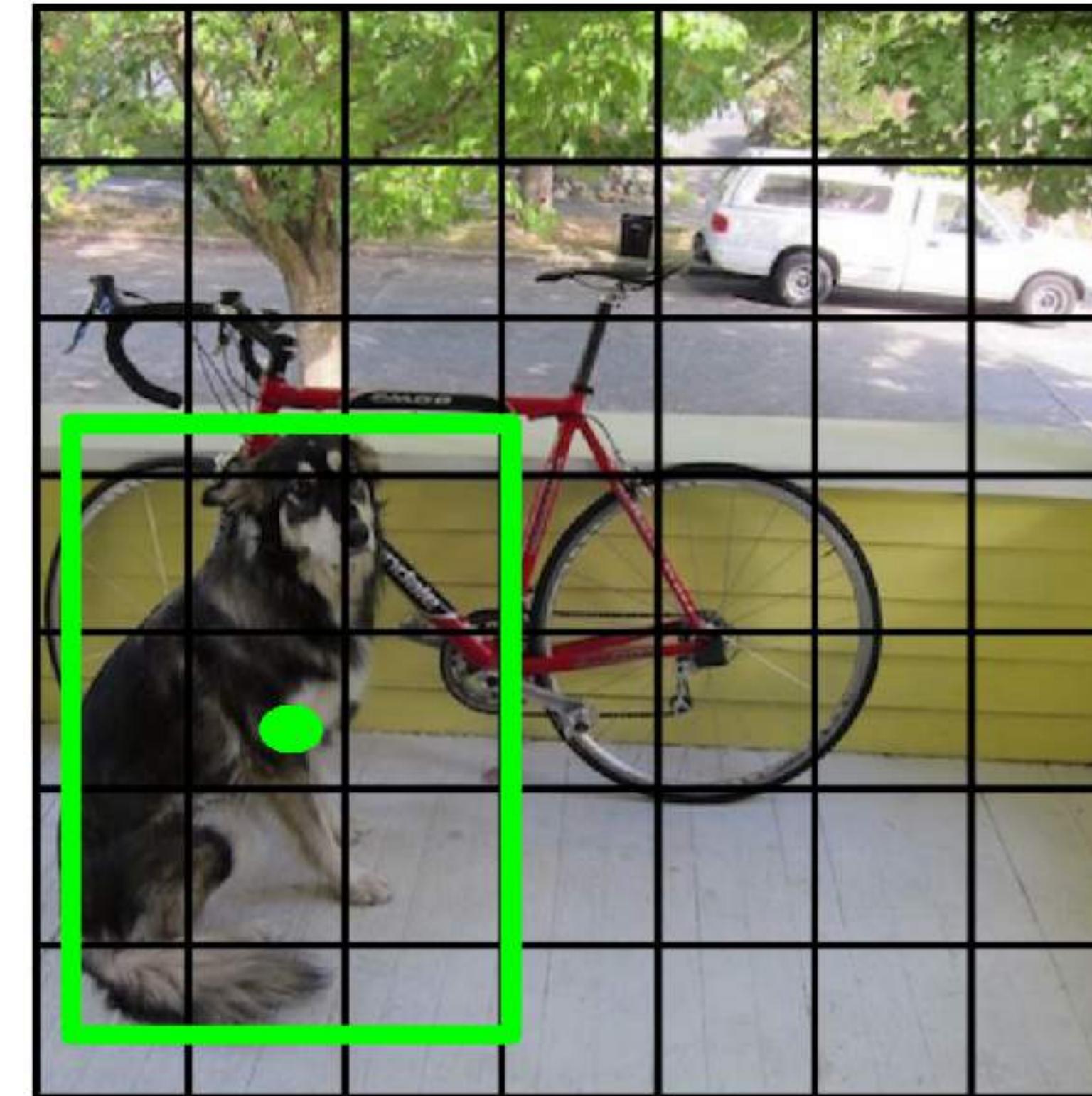
BY RAJEEV RATAN

## Training YOLO

The training process and loss functions involved in training YOLO

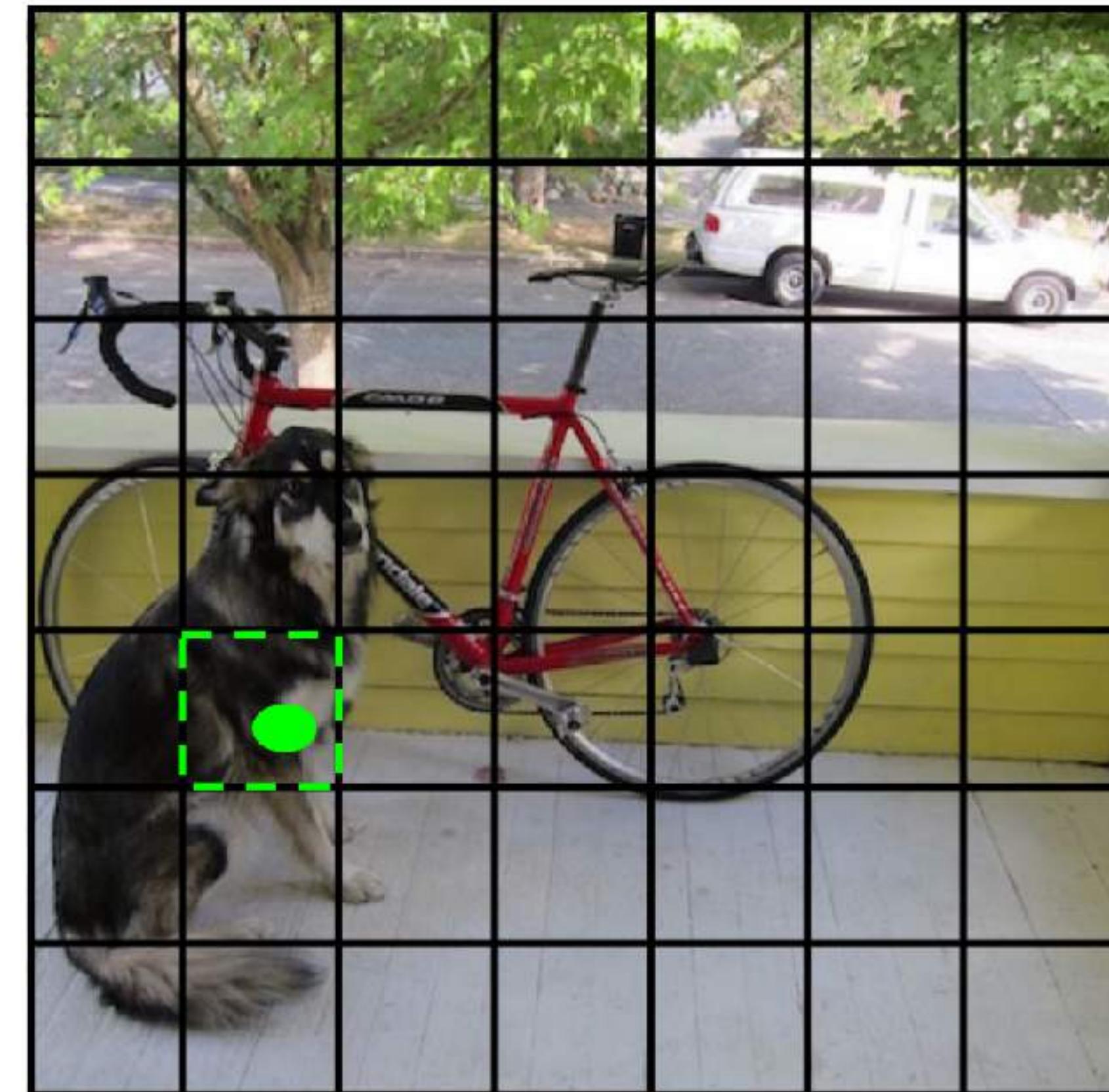
# Let's dive into how the training process works

- During training we have the ground truth labels that we use as our ‘targets’ for our model.



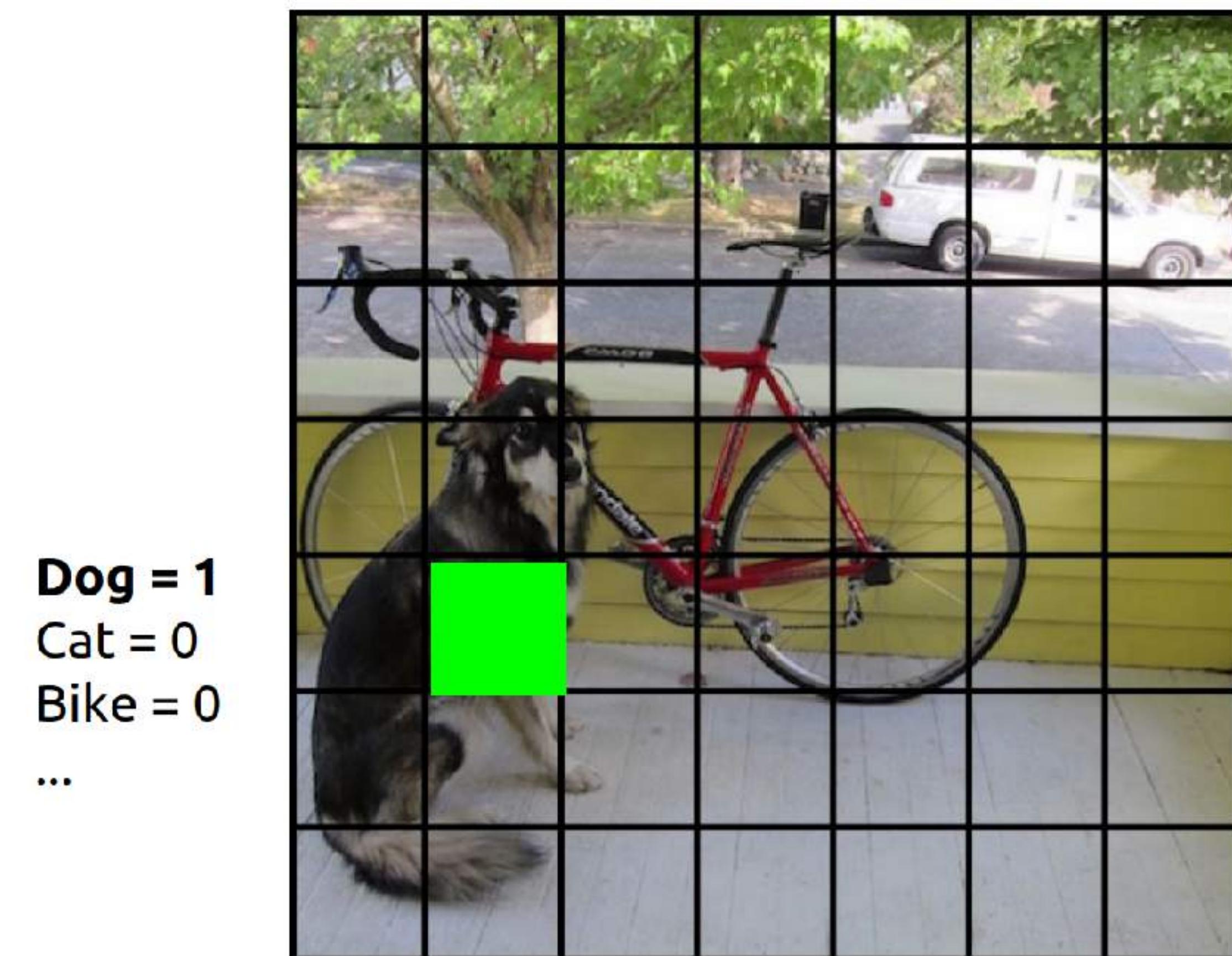
# Proposes B bounding boxes per cell

- During training the model attempts to match the example to the right cell.



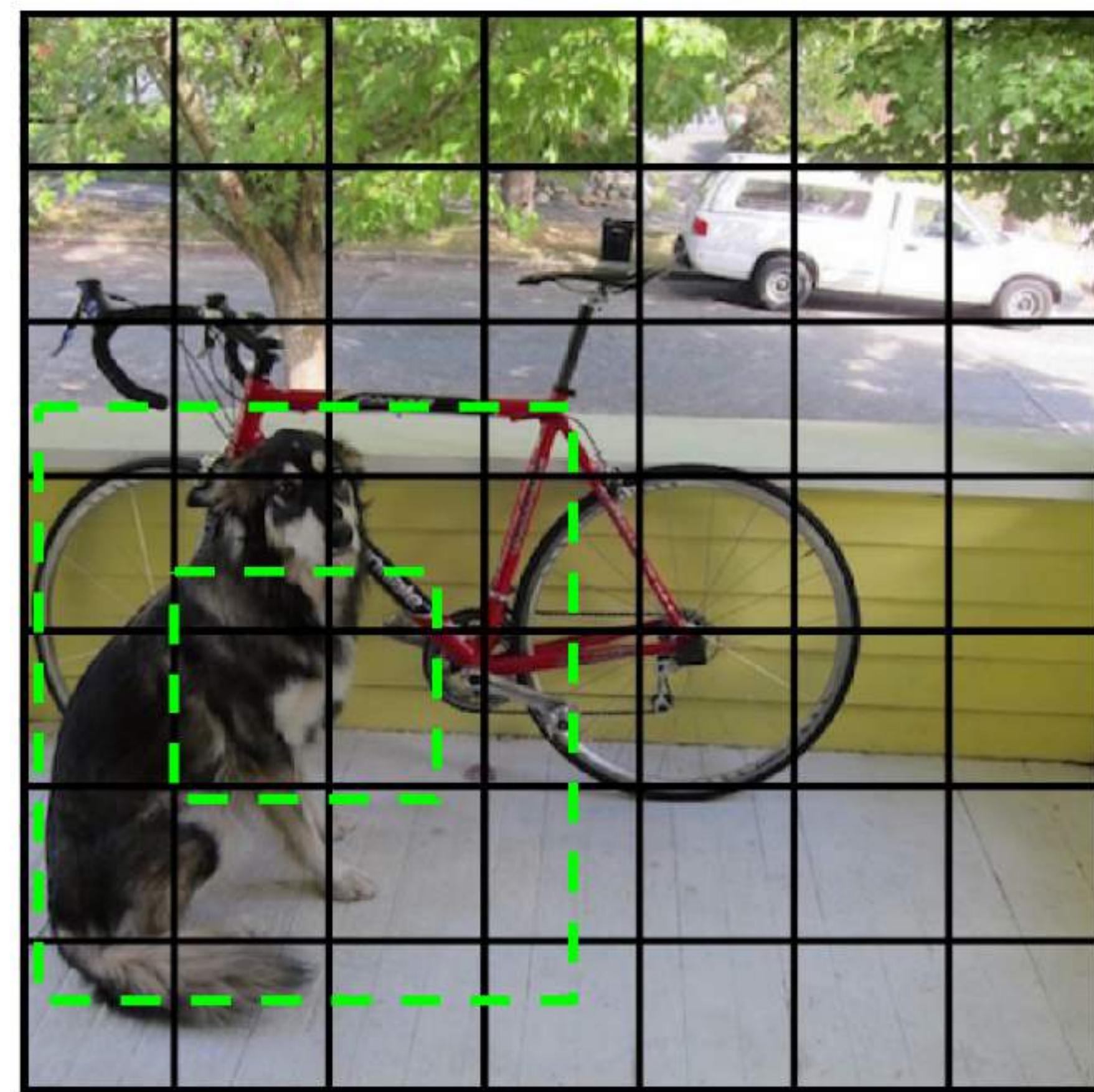
# Class prediction for the cell

- It then adjusts that cell's class prediction



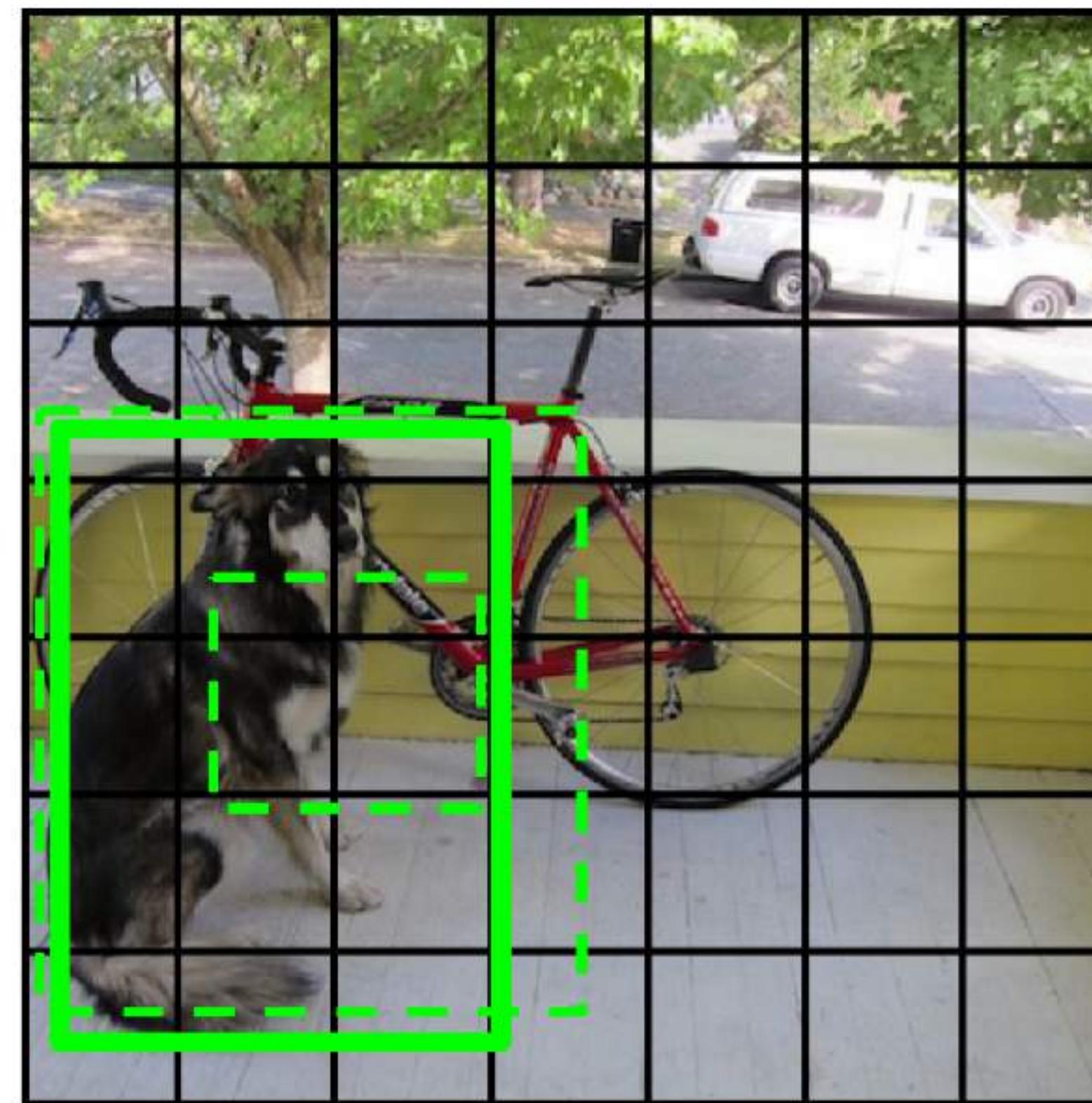
# Predicted bounding boxes

- We then get the 2 bounding box predictions for that cell



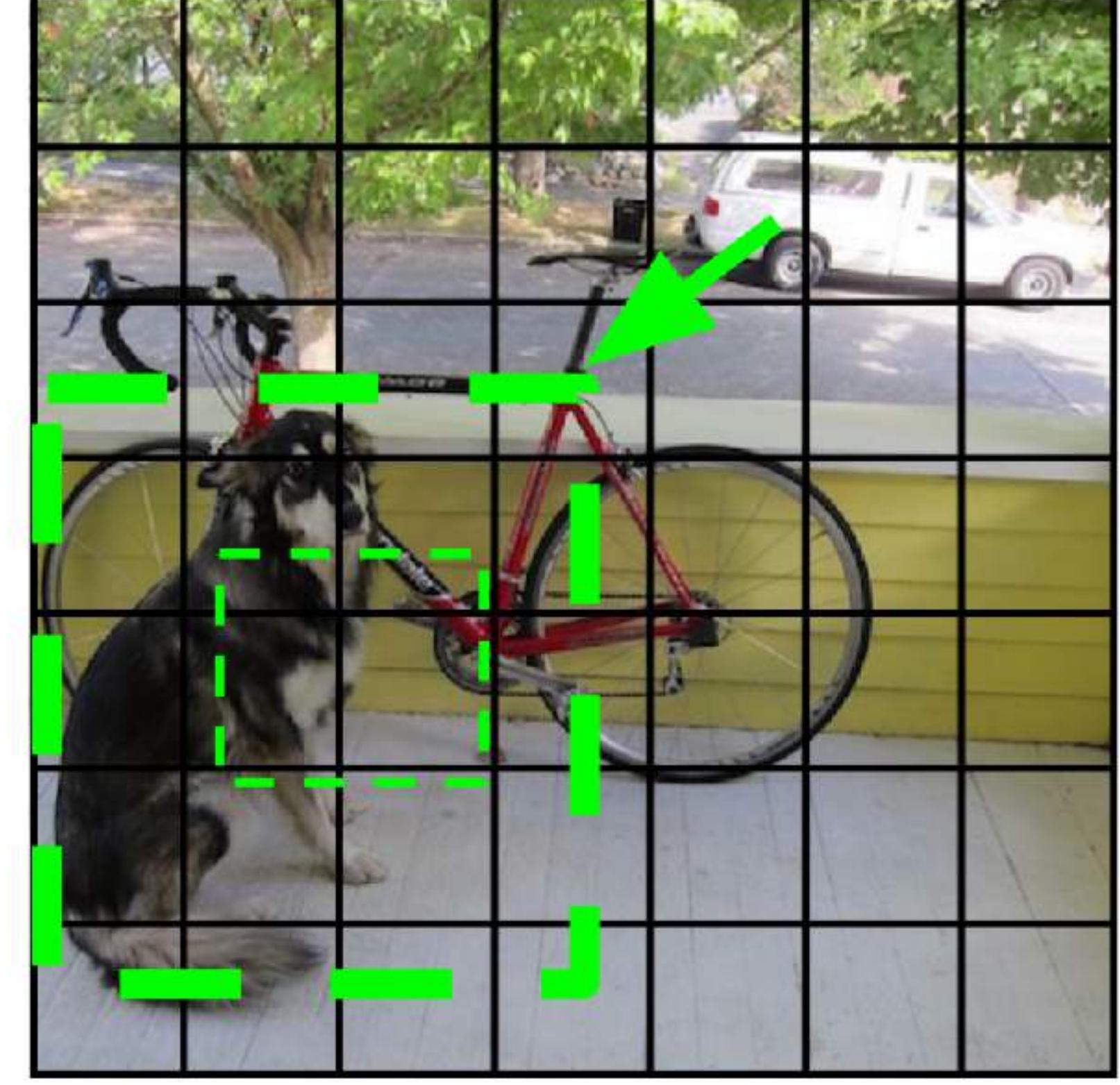
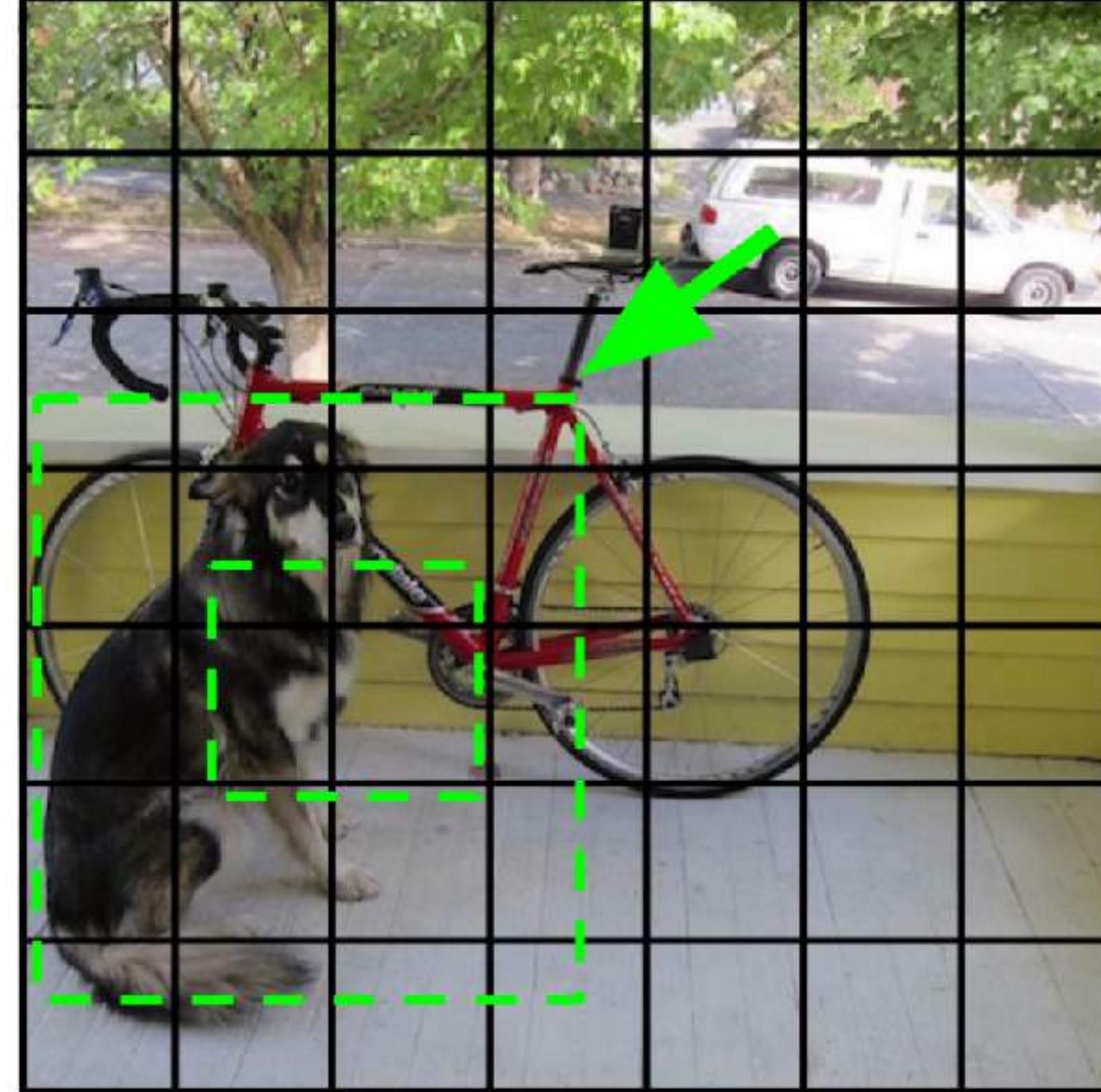
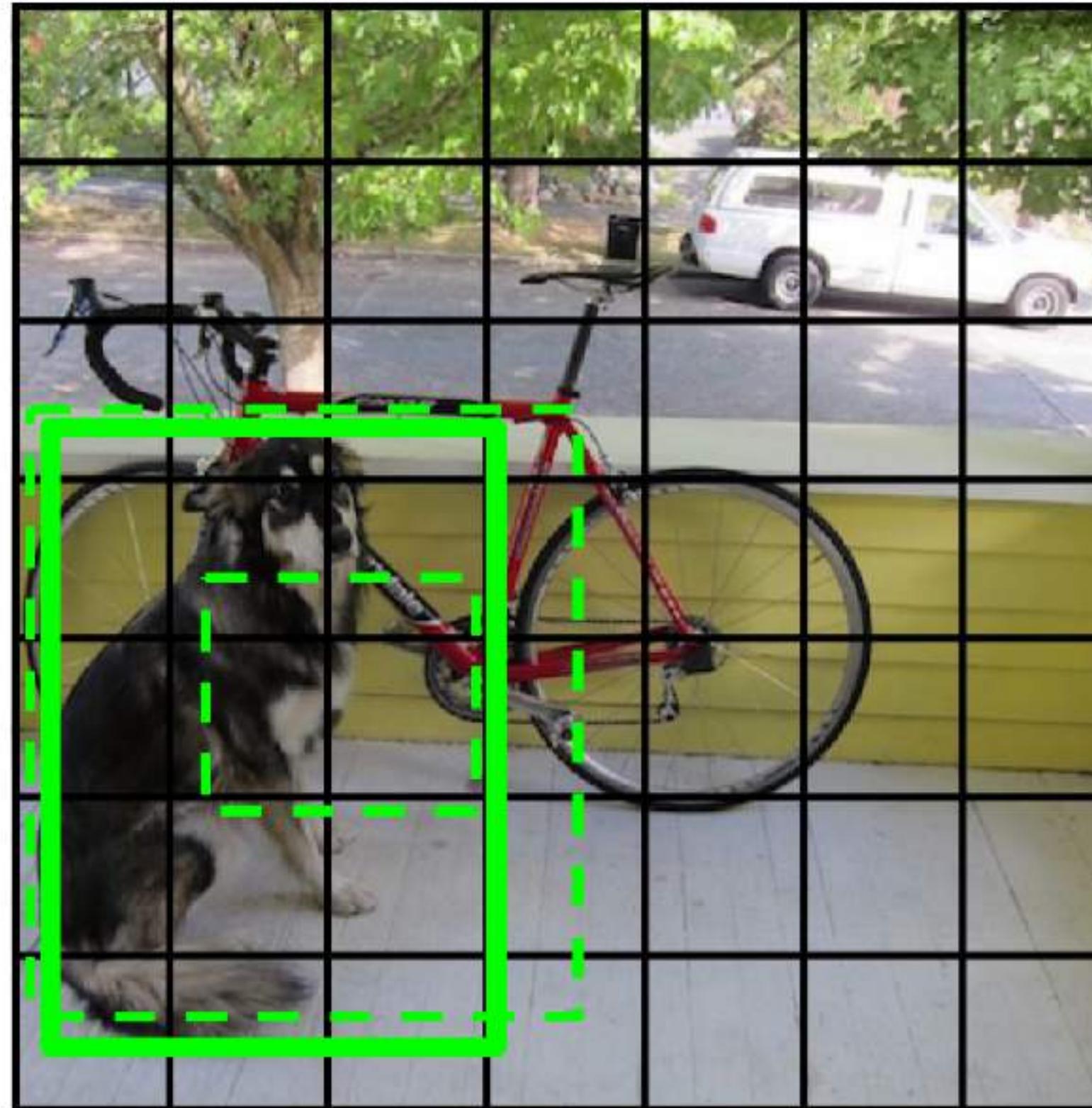
# Predicted bounding boxes

- The best box is then adjusted (i.e. confidence increase) while the less accurate box is decreased in confidence.



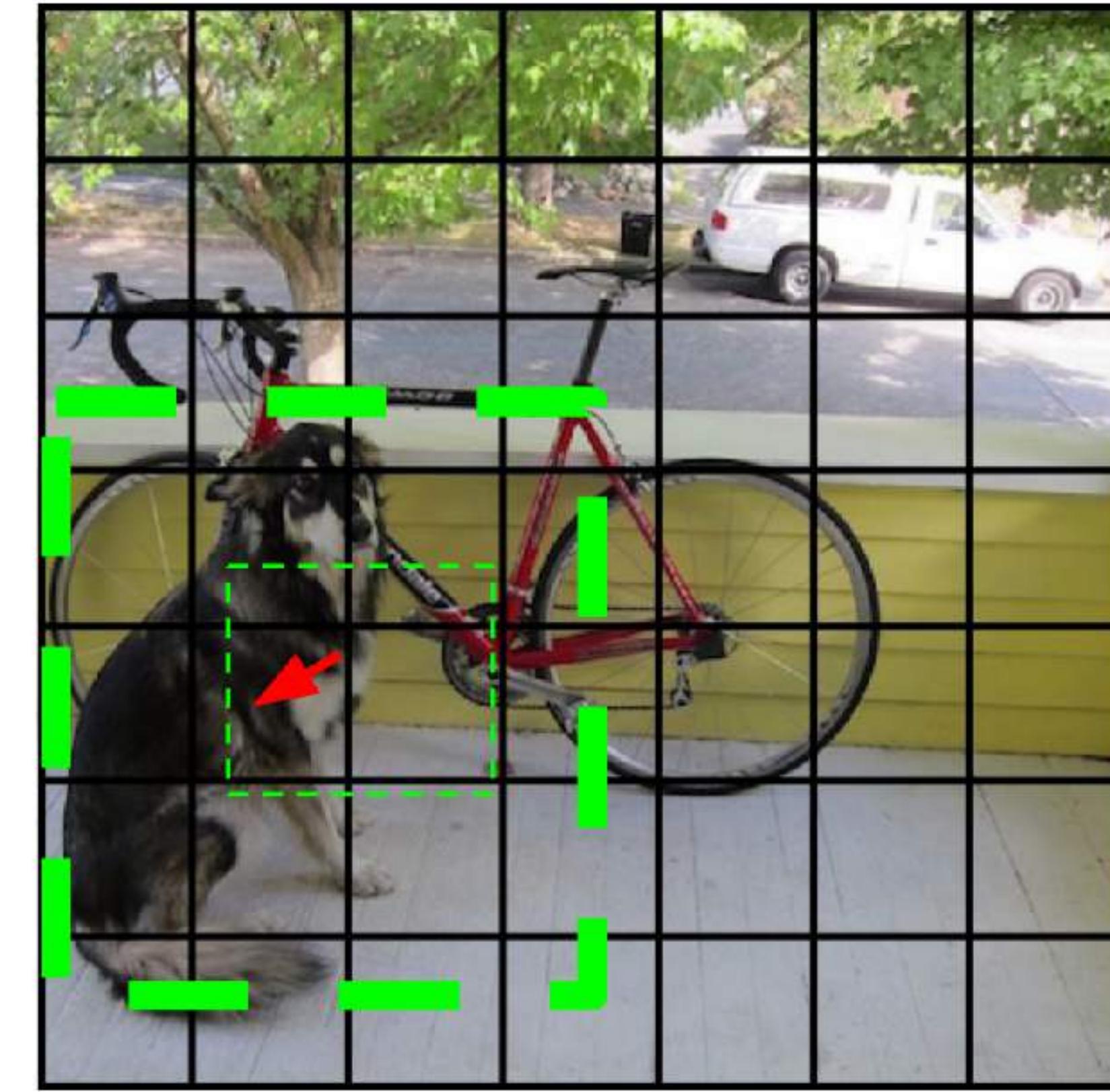
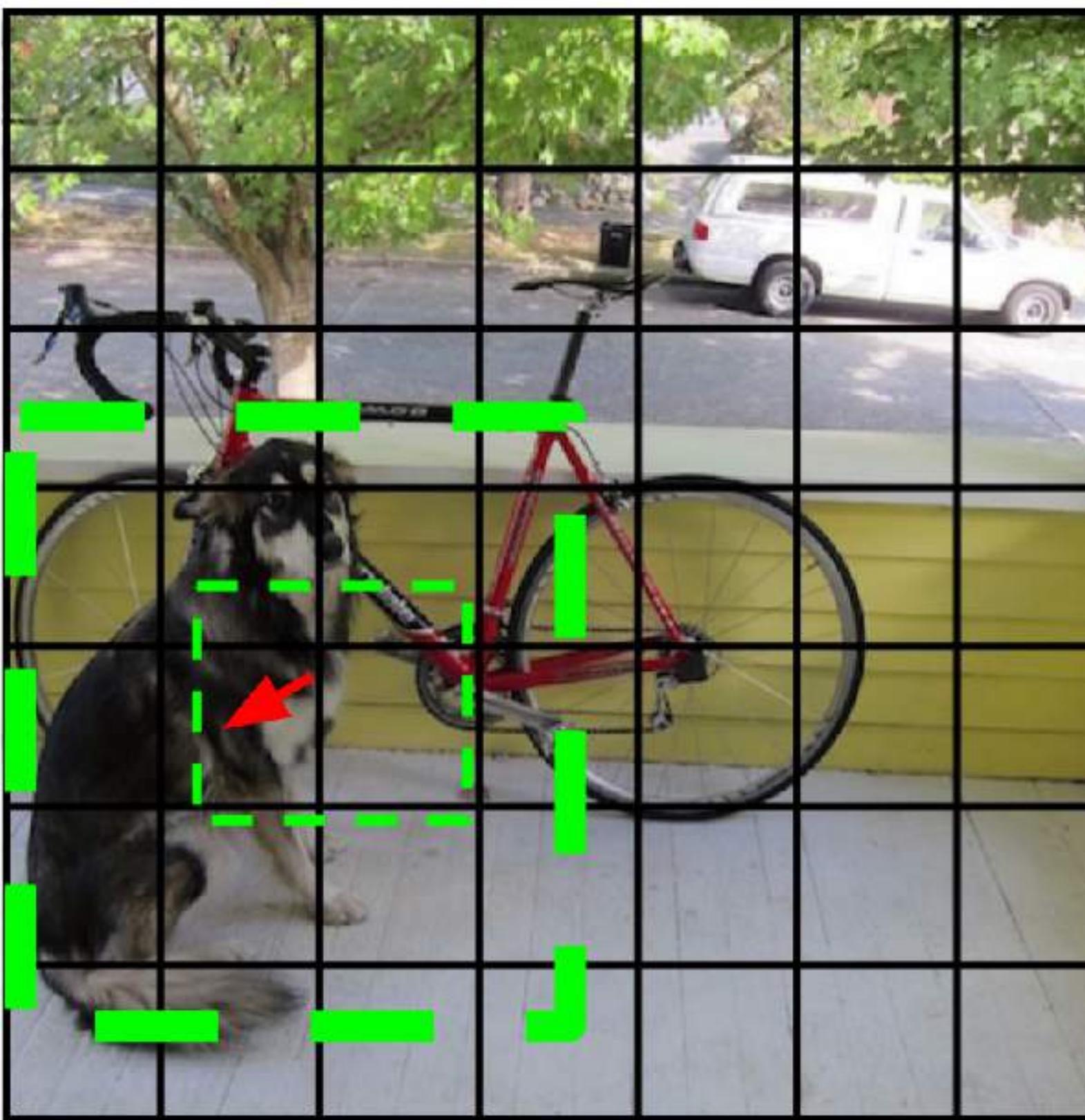
# Predicted bounding boxes

- The best box is then adjusted (i.e. confidence increase) while the less accurate box is decreased in confidence.



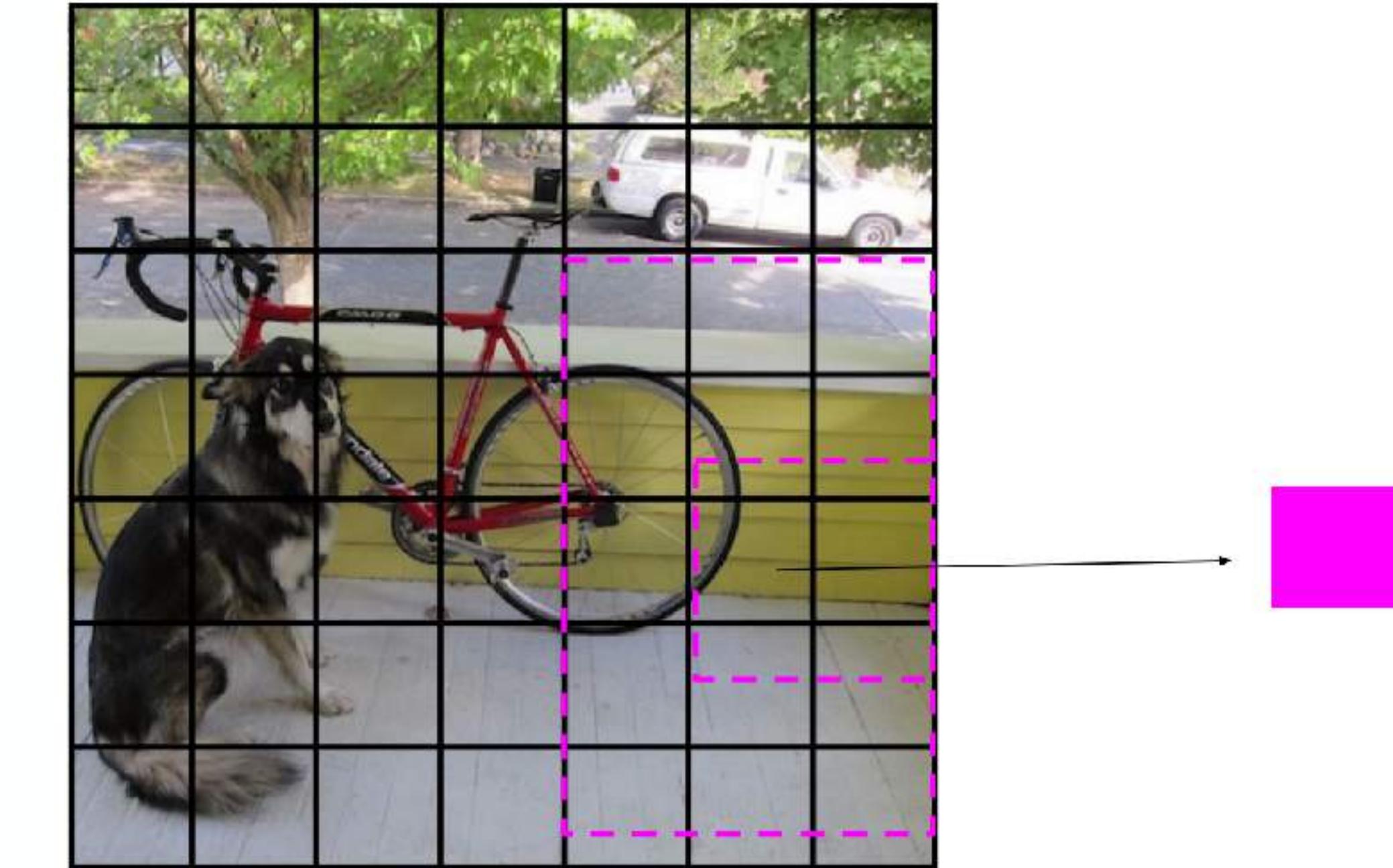
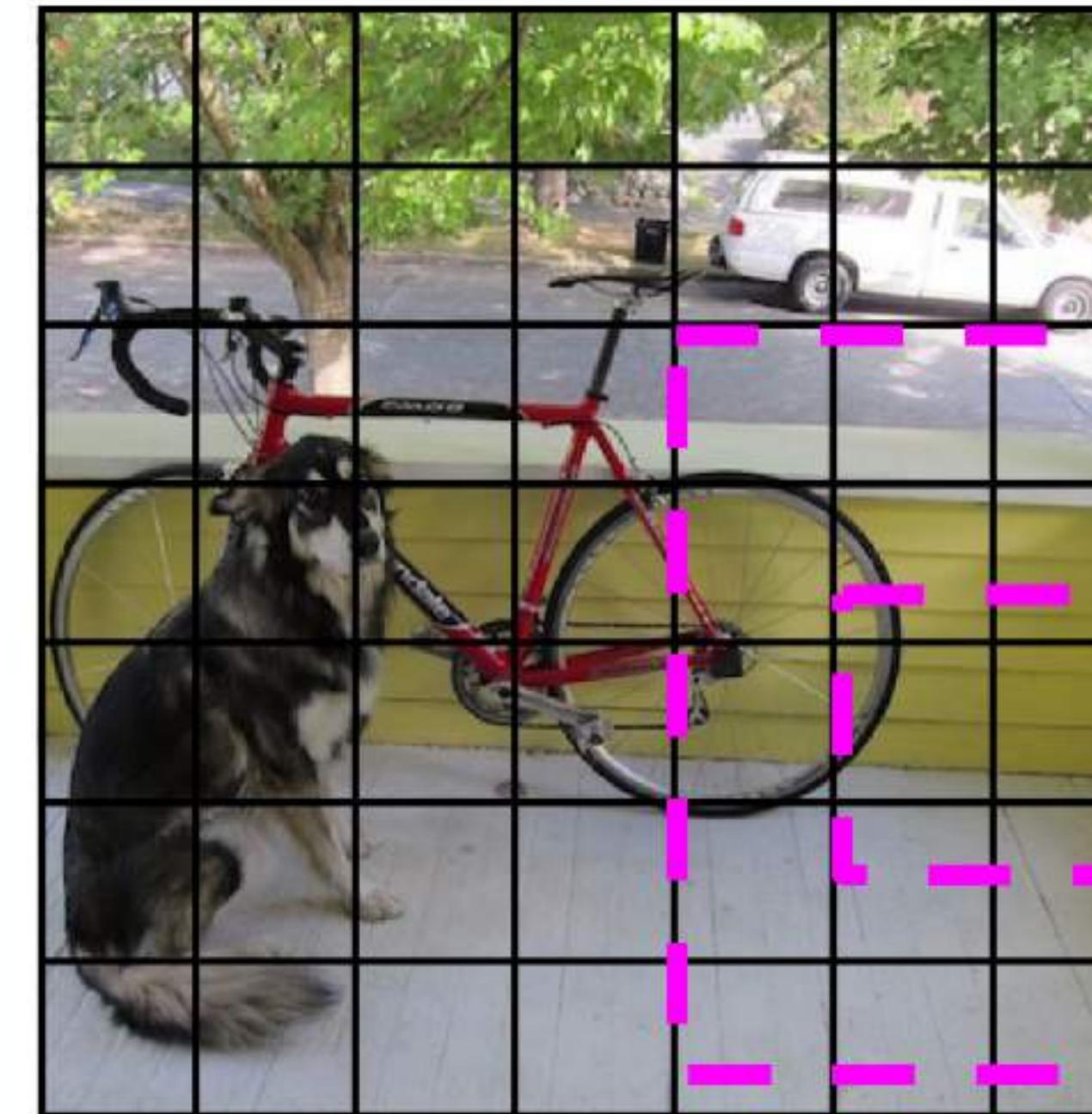
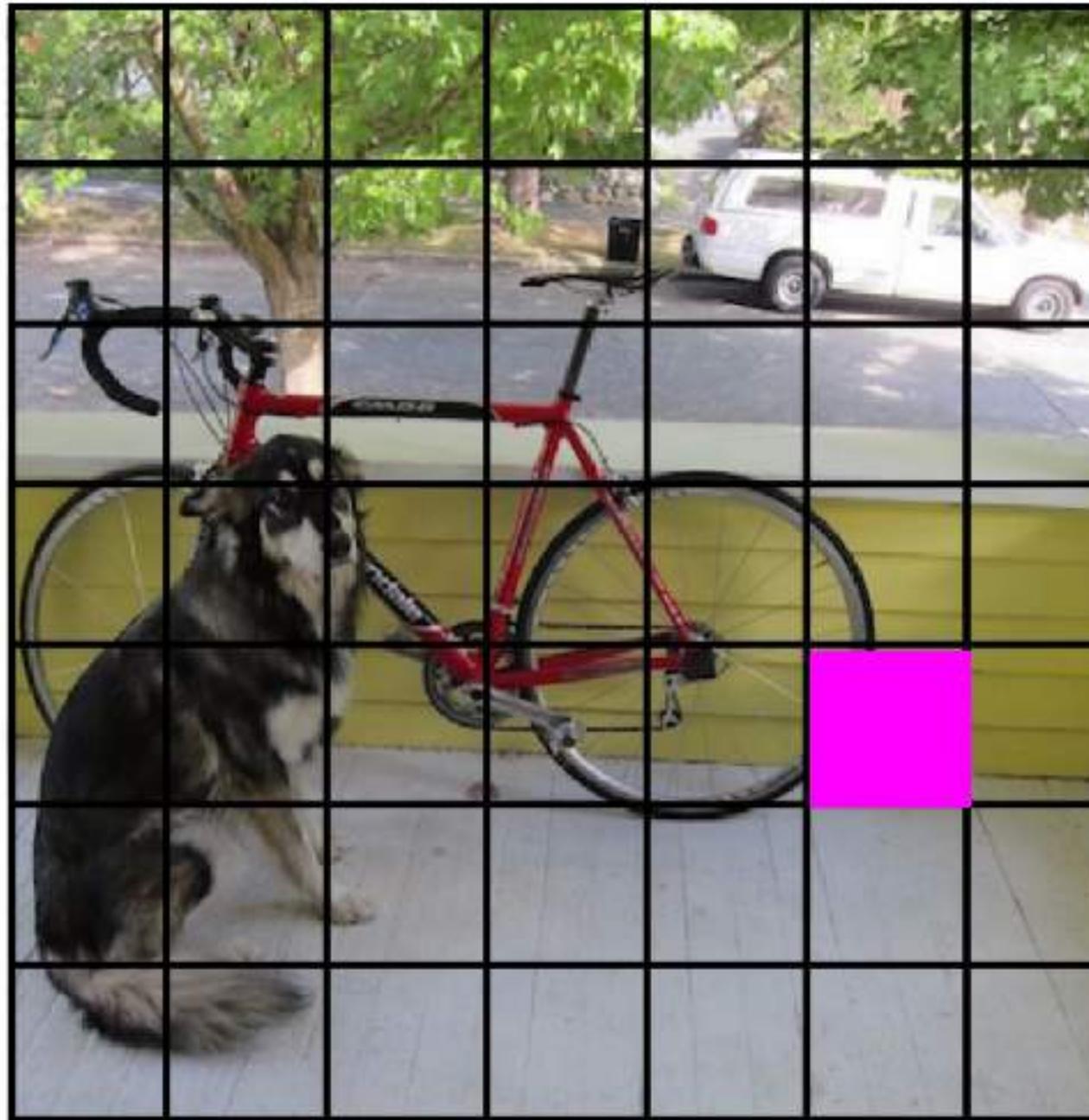
# Decrease the confidence of the other box

- The best box is then adjusted (i.e. confidence increase) while the less accurate box is decreased in confidence.



# Cells with no ground truth detections are also decreased in confidence

- The boxes where no ground truth boxes lie are decreased in confidence
- We don't adjust the class probabilities or coordinates of these boxes



# YOLO's Loss Functions

- **Classification Loss** - if an object is detected, it is the squared error loss of the class conditional probabilities for each class
- **Localisation Loss** - Measures error from the predicted boundary box to the ground truth
- **Confidence Loss** - confidence that an object lies in the box

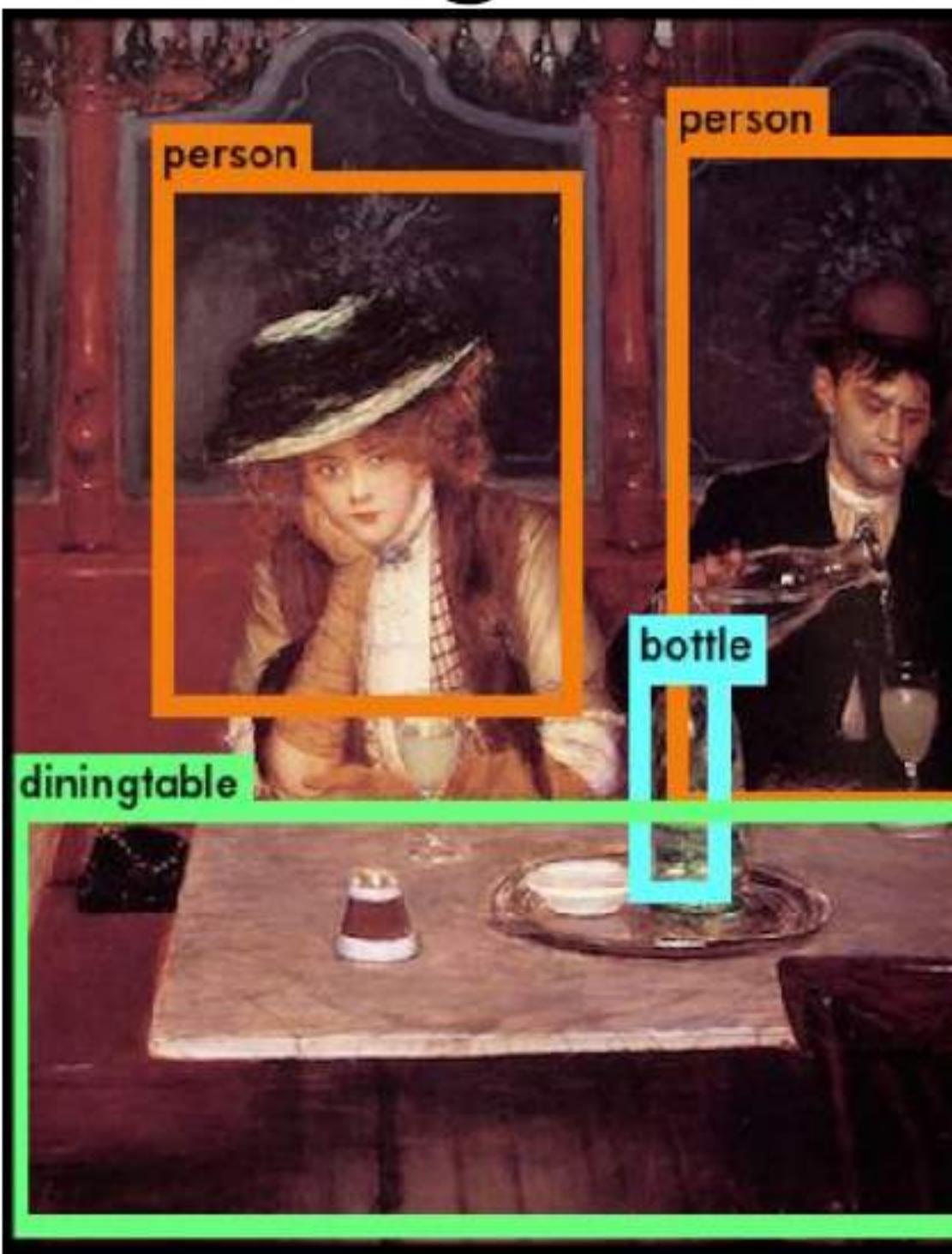
# YOLO's Performance

	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	63.4	45 FPS	22 ms/img

<https://pjreddie.com/publications/>

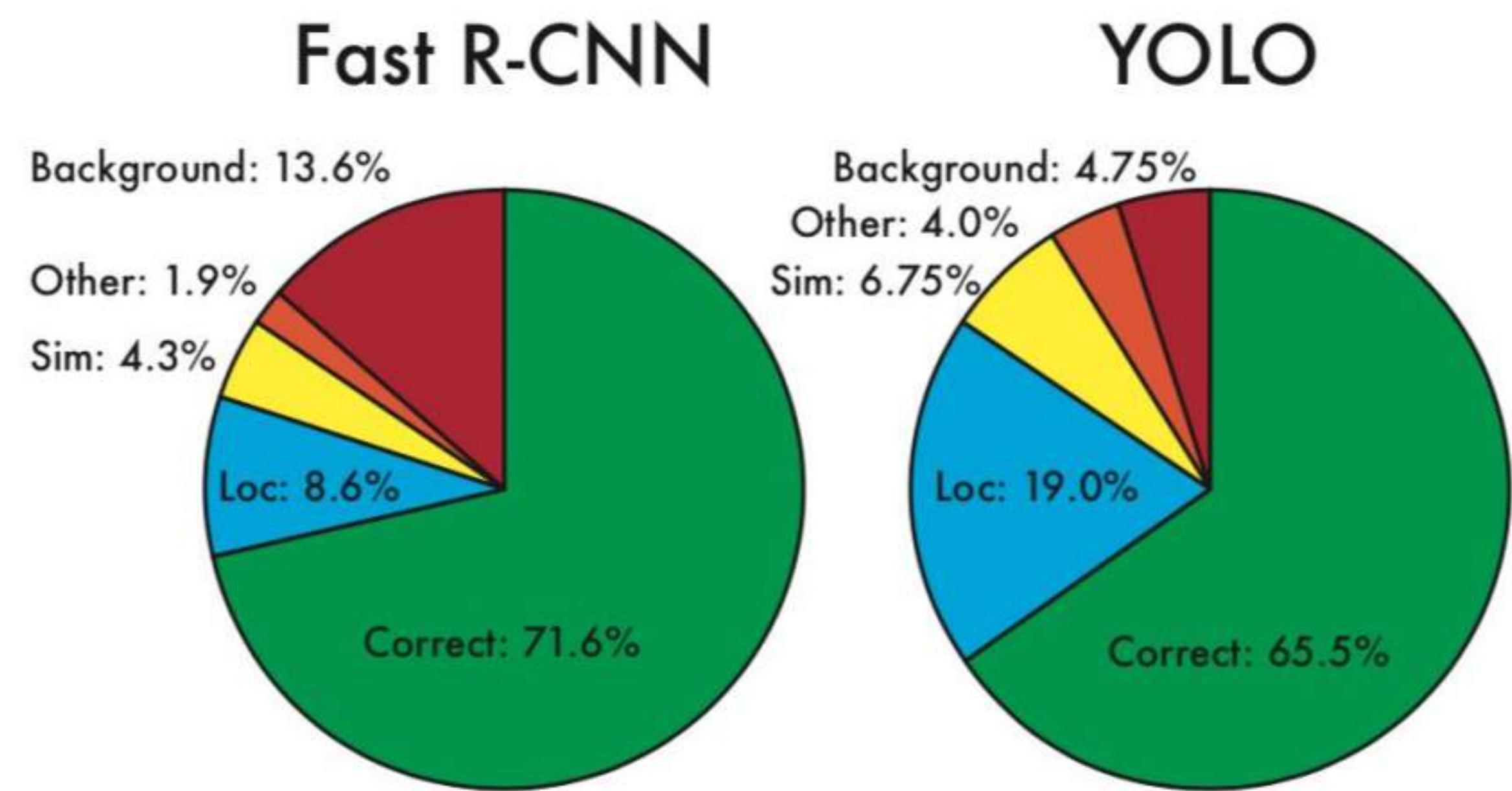
# YOLO's Performance

YOLO generalizes well to new domains (like art)



Ref: <https://pjreddie.com/publications/>

# YOLO vs Fast R-CNN



Loc: Localization Error

Correct class,

$.1 < \text{IOU} < .5$

Background:

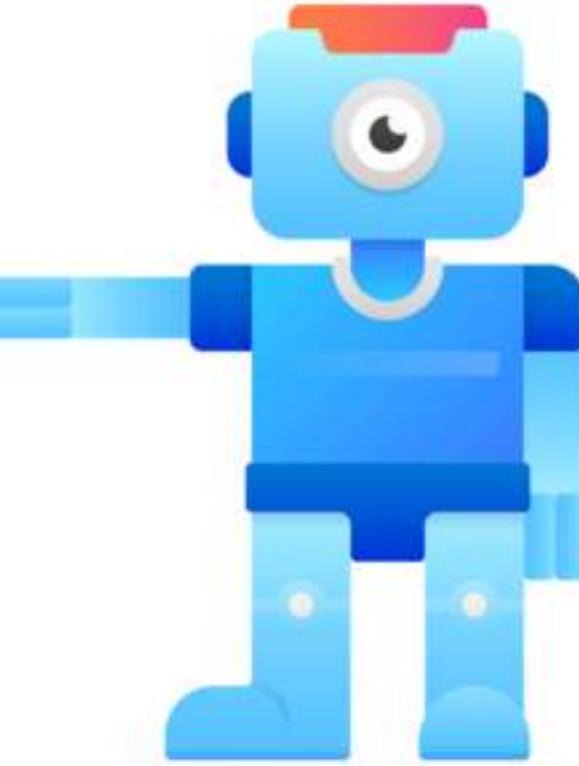
$\text{IOU} < 0.1$

**Figure 4: Error Analysis: Fast R-CNN vs. YOLO** These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

<https://arxiv.org/pdf/1506.02640.pdf>

# YOLO Take Aways

- YOLO is fast!
- YOLOv4, v5 and X are perhaps the best in accuracy (mAP) in 2021
- Provides End2end training
- Low background error
- Performance (mAP) was not as good as slower Fast R-CNNs
- More localisation errors

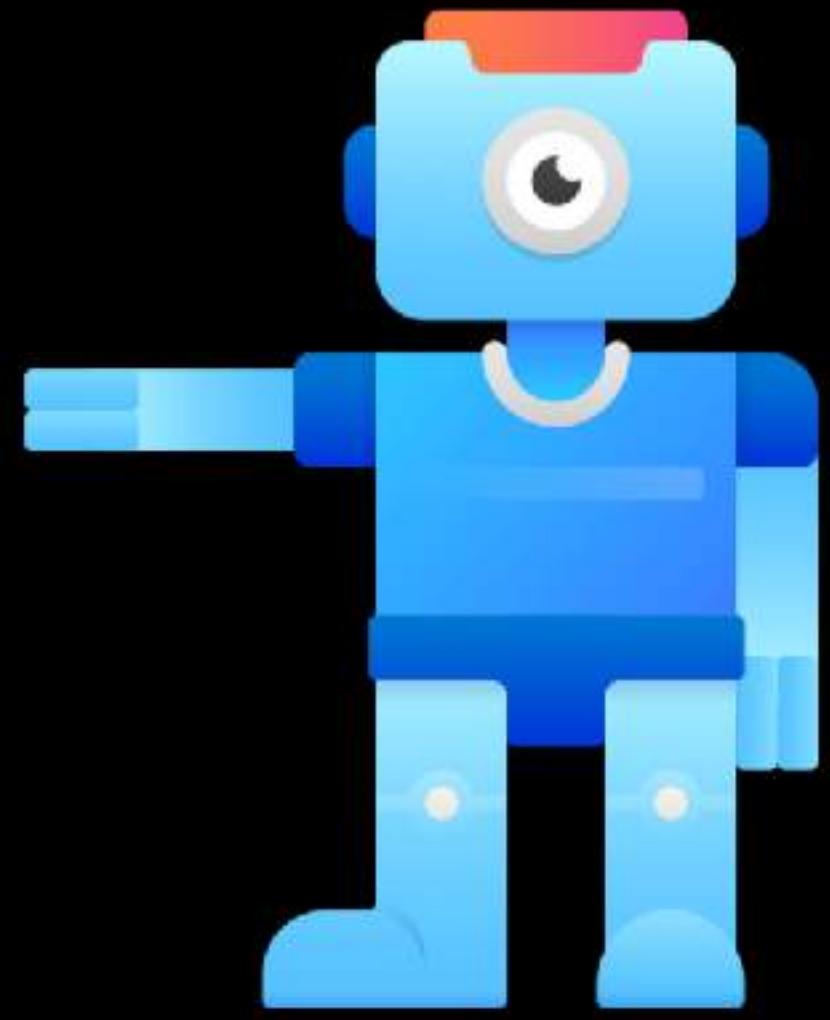


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**YOLO Architecture and Evolution from YOLOv3 to v5**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

## YOLO Architecture and Evolution from YOLOv3 to v5

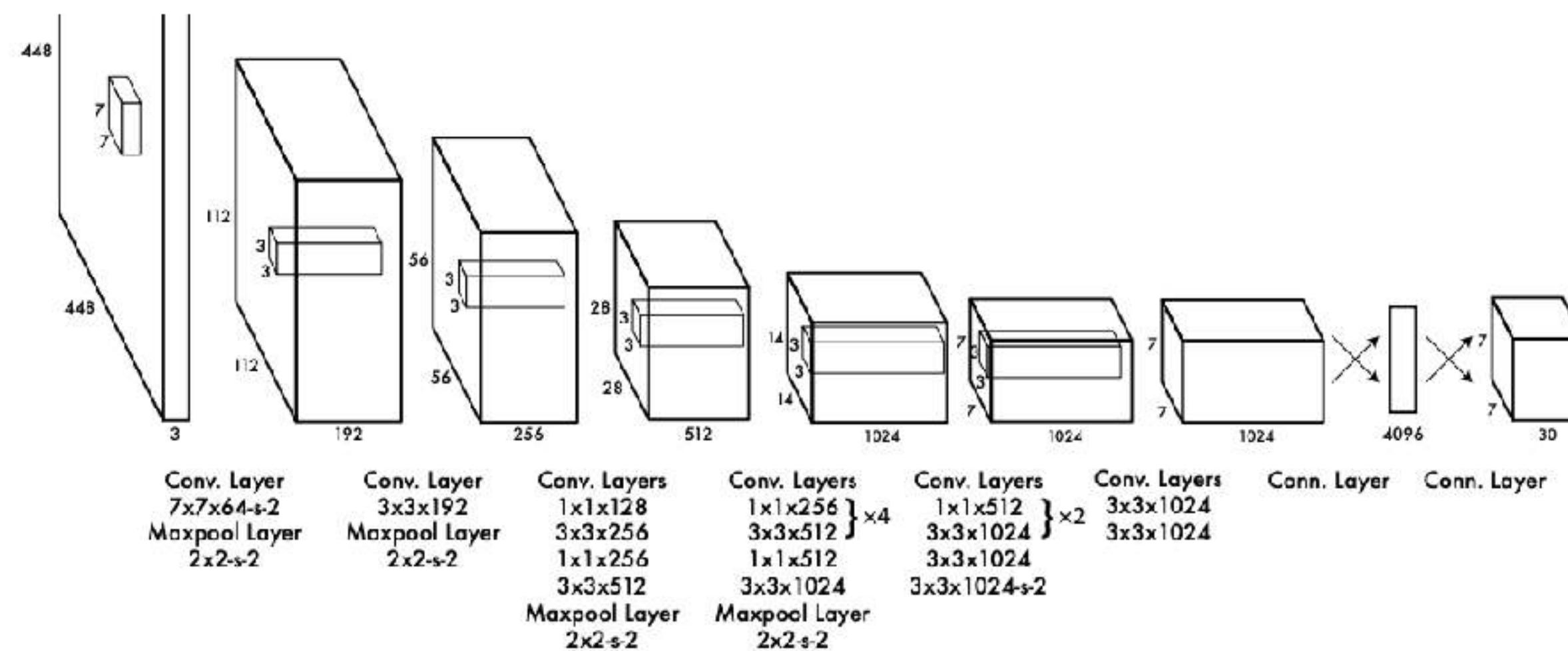
We take a look at YOLO's core Architecture and Evolution from YOLOv3 to v5

# Object Detector Architecture

- Most conventional object detectors are consist of two to three main parts
  - **Backbone** (typically VGG, ResNet, DenseNet or DarkNet) - used for feature extraction
  - **Head** - for loss calculations and predictions during inference
  - **Neck** - introduced only in recent detectors, it is directly leveraged into the backbones for enhancing the richness and semantic representation of the extracted features for objects of difference shapes and sizes

# YOLOv1 and v2

- YOLOv1 was the first showing of YOLO, a single stage detector and was made public in 2015.
- It utilised batch normalisation and leaky ReLU activations

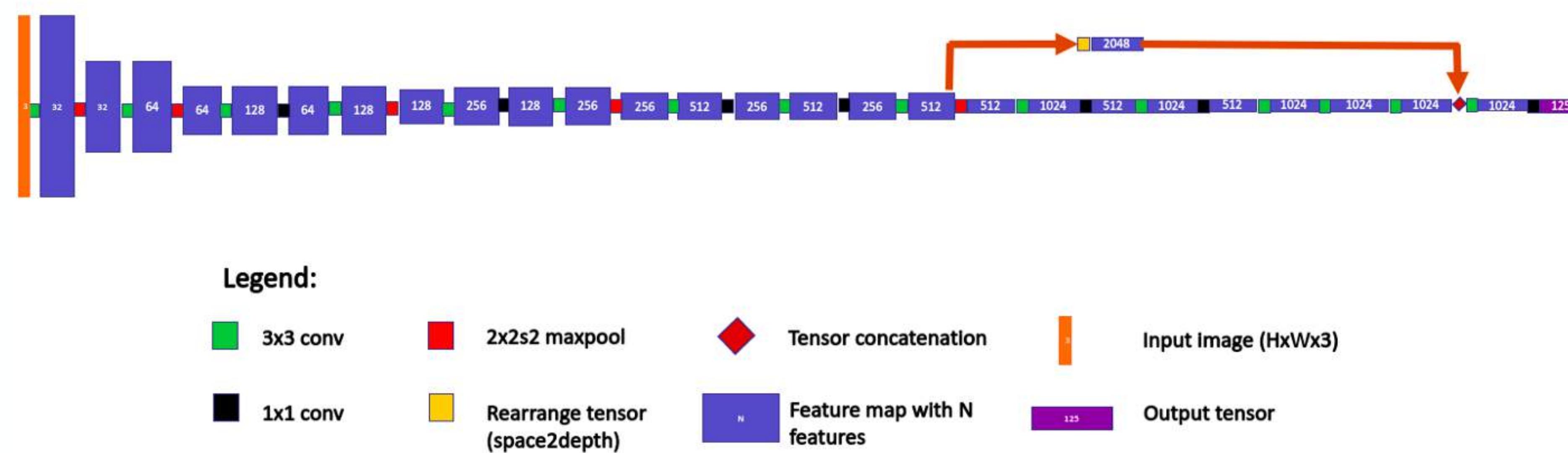


<https://arxiv.org/pdf/1506.02640.pdf>

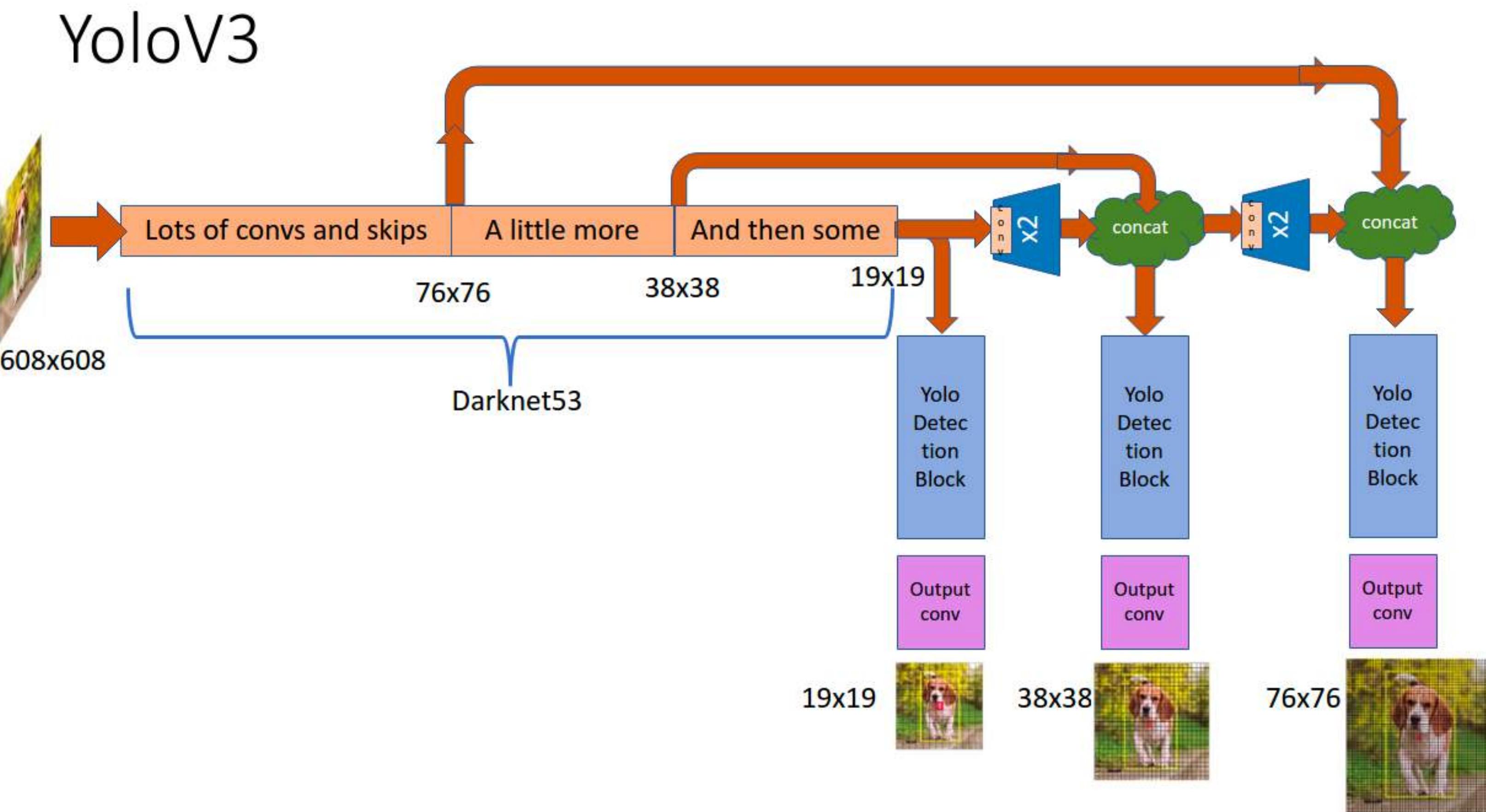
**Figure 3: The Architecture.** Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating  $1 \times 1$  convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution ( $224 \times 224$  input image) and then double the resolution for detection.

# YOLOv2

- In YOLOv2 the authors implemented several changes such as:
  - Removing the FC layer at the end (facilitating resolution independence)
  - A few new versions of YOLOv2 such as TinyYOLOv2 were released

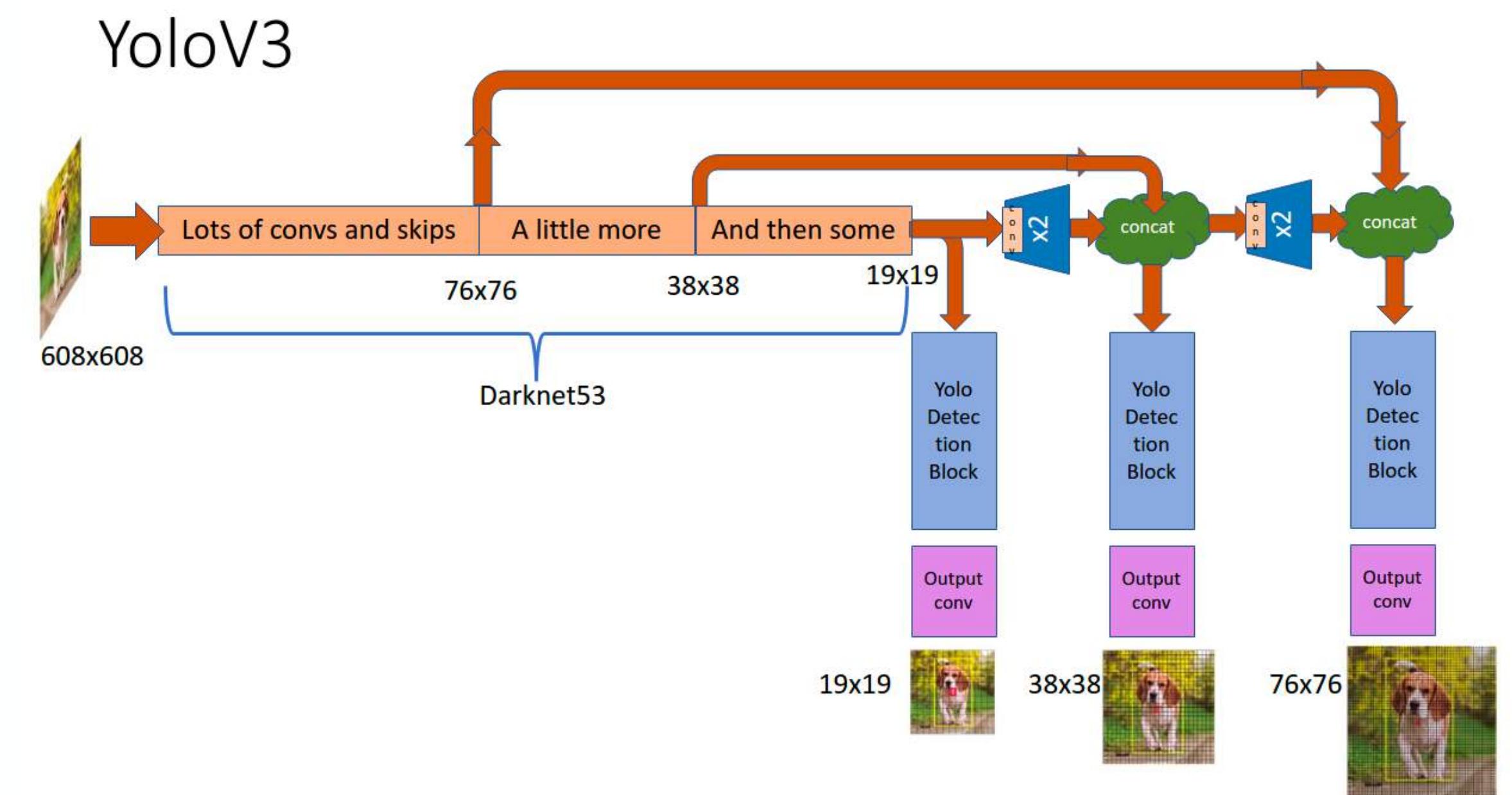


# YOLOv3



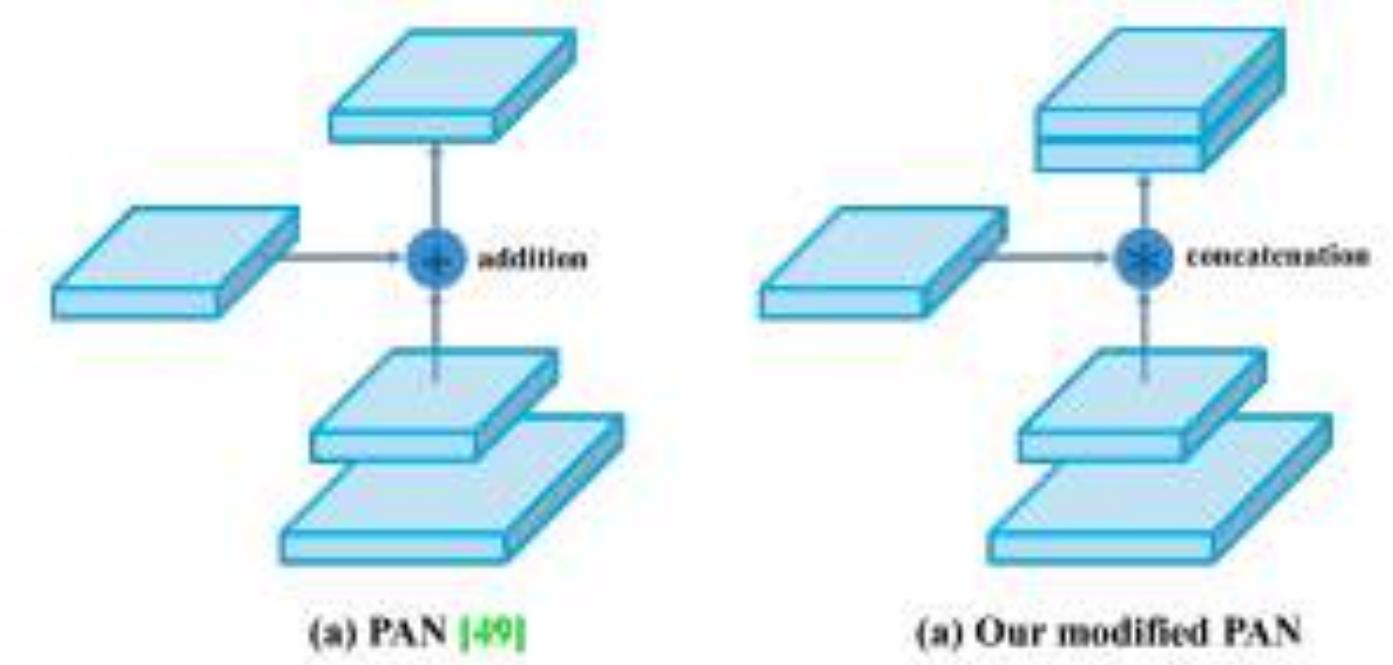
# YOLOv3

- YOLOv3 was inspired by ResNet and Feature Pyramid Networks (FPN)
- The researchers utilised a new feature extractor backend called Darknet-53 which had skip connections like ResNet and 3 prediction heads like FPN.

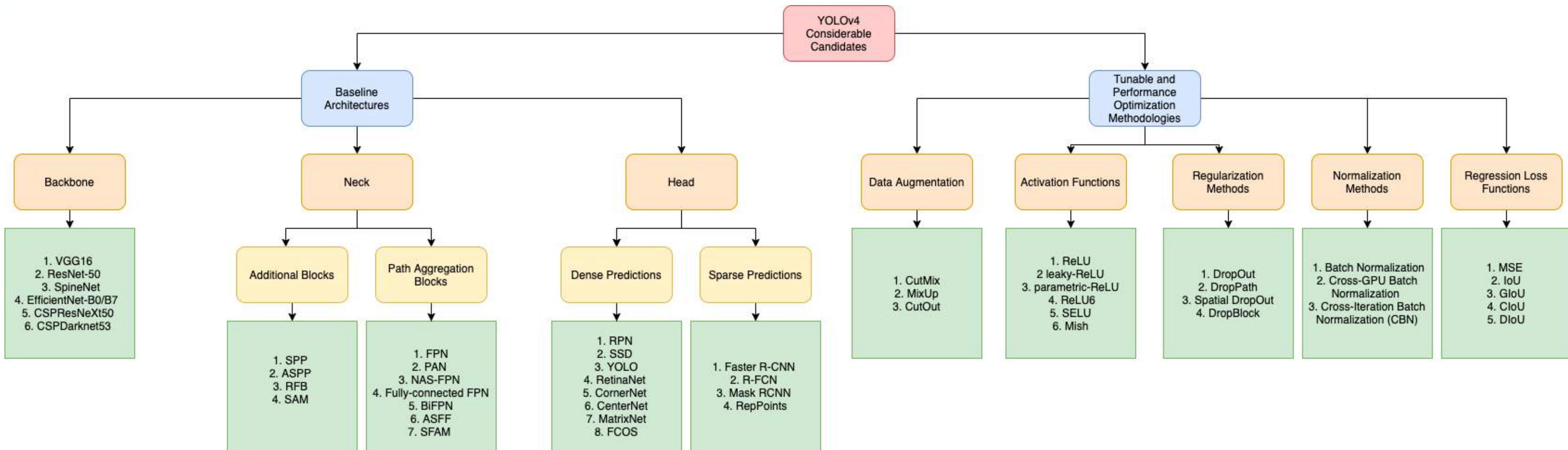


# YOLOv4

- YOLOv4 shortlisted 3 different backbones (Darknet53, ResNet50 and EfficientNet-B3) providing various speed (FPS) and accuracy.
- Darknet53 was selected as the best choice
- YOLOv4 features a modified Path Aggregation Network (PAN)
- It uses Spatial Pyramid Pooling (SPP) tightly coupled with Darknet53, this aided in increasing the receptive field.



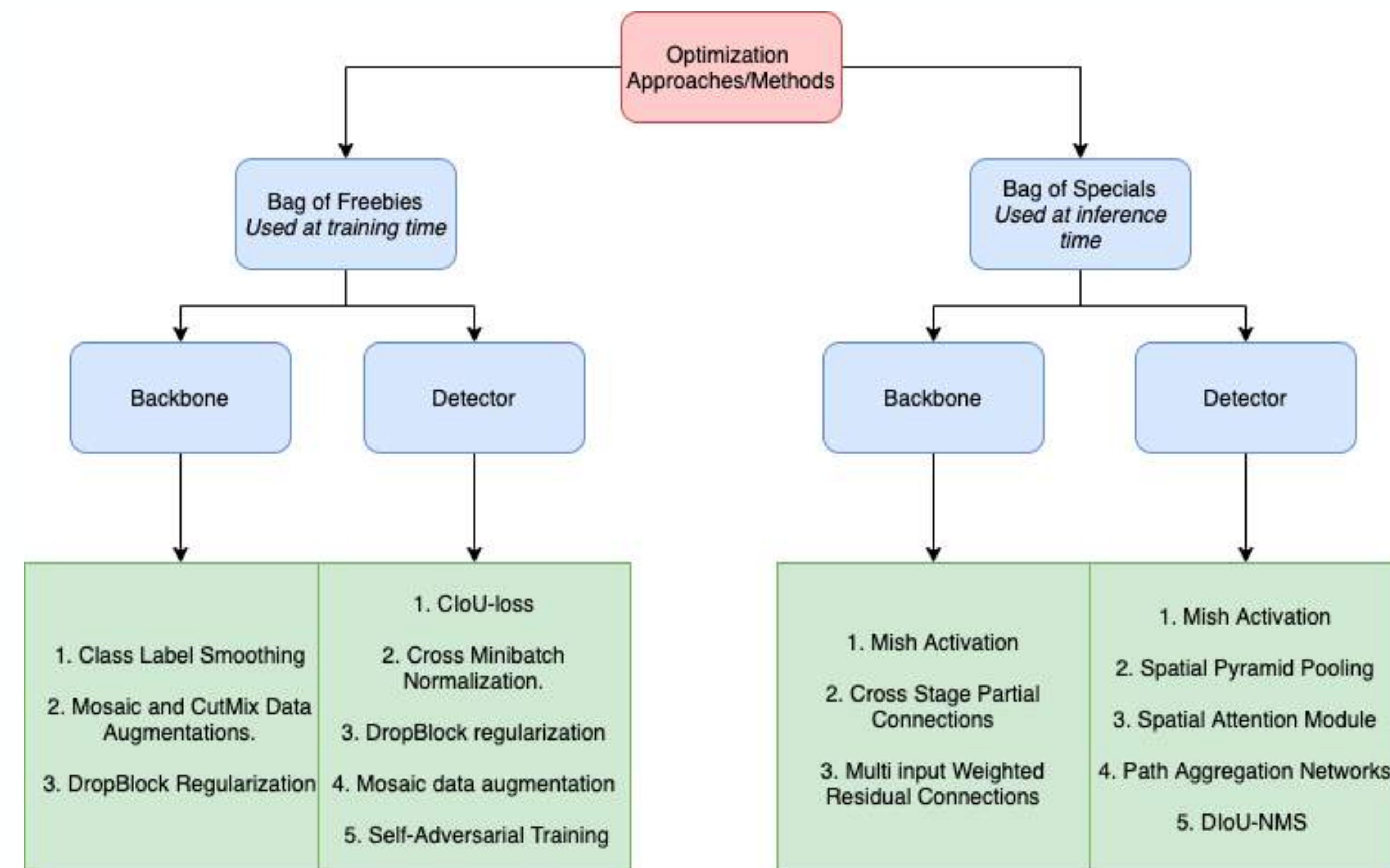
# YOLOv4



Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPResNext50	512x512	425x425	20.6 M	1058 K	31 (15.5 FMA)	62
CSPDarknet53	512x512	725x725	<b>27.6 M</b>	950 K	52 (26.0 FMA)	<b>66</b>
EfficientNet-B3 (ours)	512x512	<b>1311x1311</b>	12.0 M	668 K	11 (5.5 FMA)	26

# YOLOv4 - Performance Optimisations

- Significant Performance Optimisations (Bag of Freebies and Bag of Specials) were introduced.

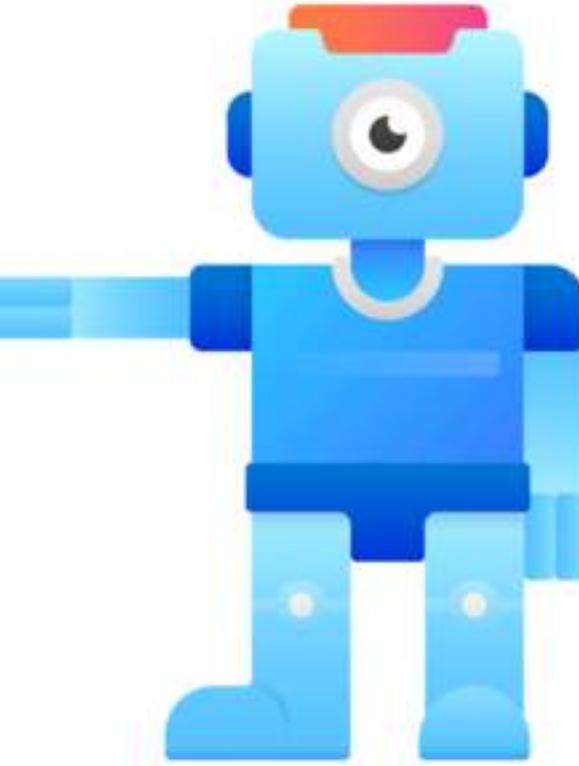


# YOLOv5 - Ultralytics

- A heavily optimised PyTorch version of YOLOv4 that has been open-sourced by Ultralytics who have made the training and deployment of YOLOv5 models quite easy and configurable!



<https://github.com/ultralytics/yolov5>

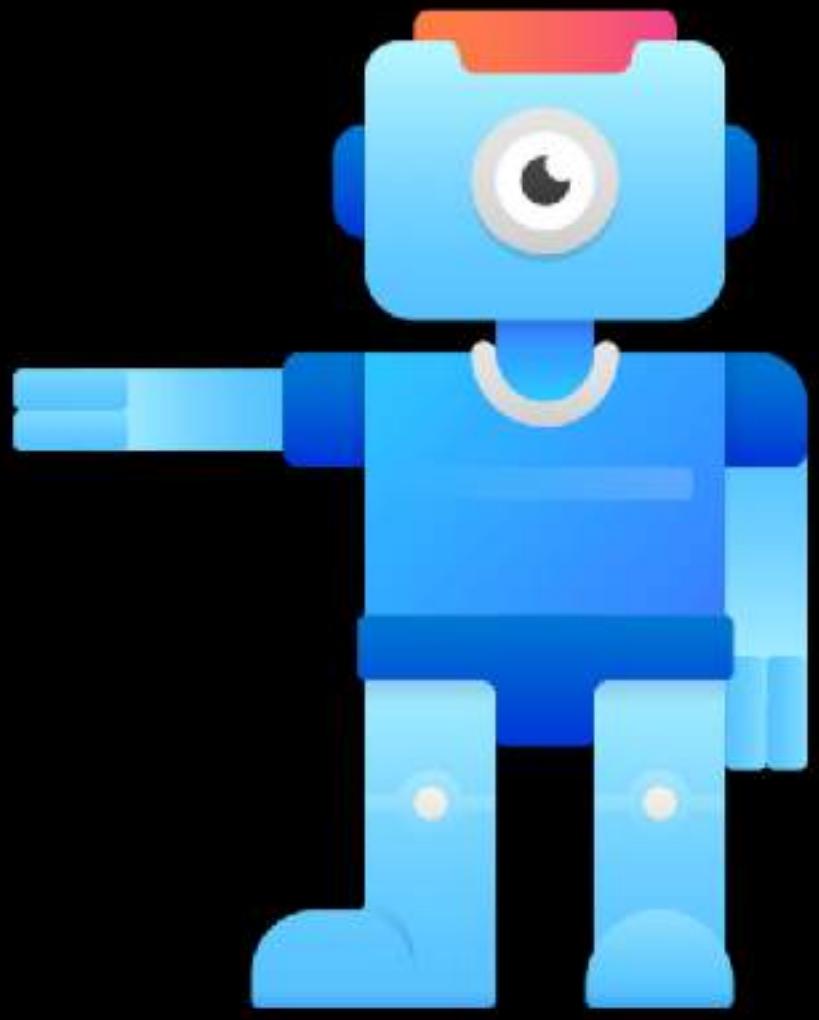


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**EfficientDet - Object Detection**



# MODERN COMPUTER VISION

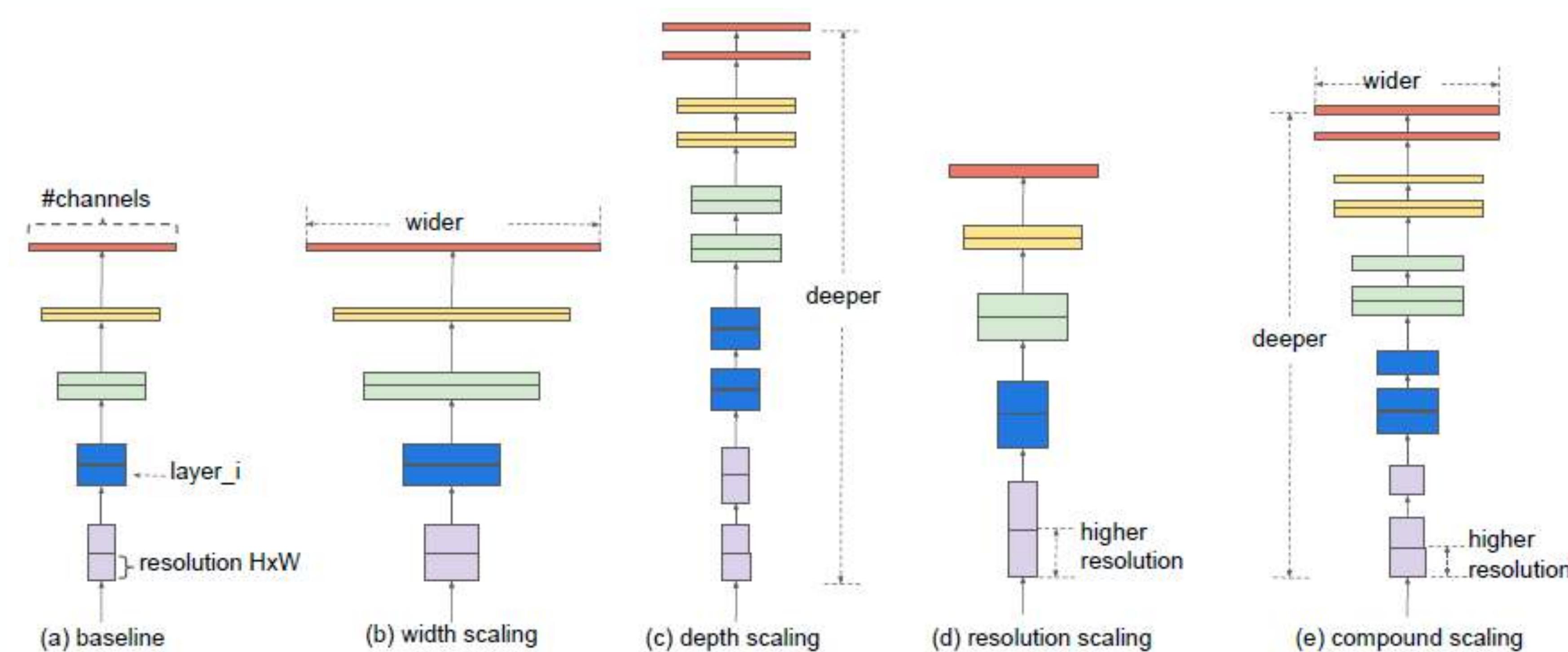
BY RAJEEV RATAN

## EfficientDet - Object Detection

We take a look at EfficientDet for Object Detection

# Recall EfficientNet

- EfficientNet was an effective scaling method for CNNs.



# EfficientDet

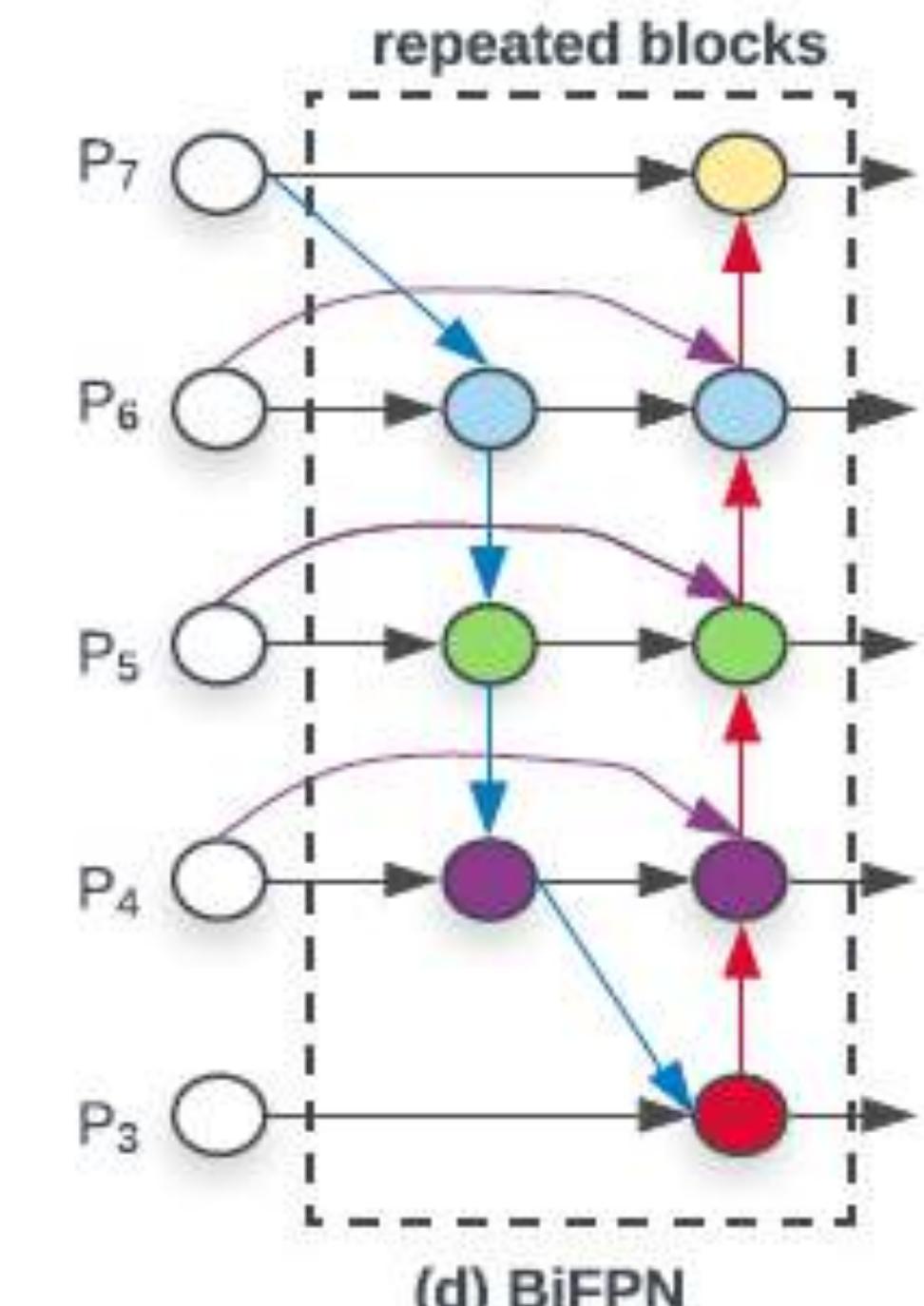
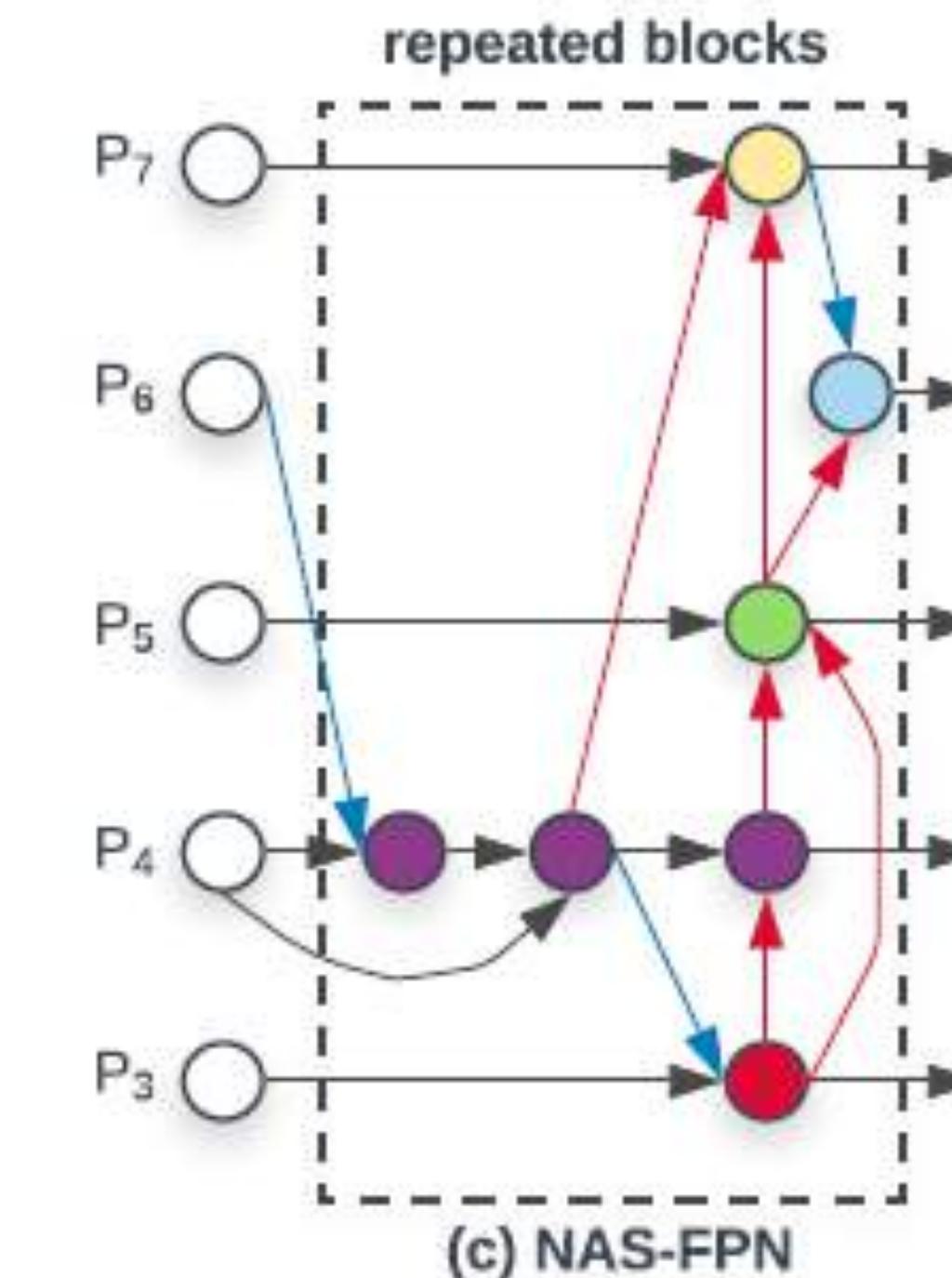
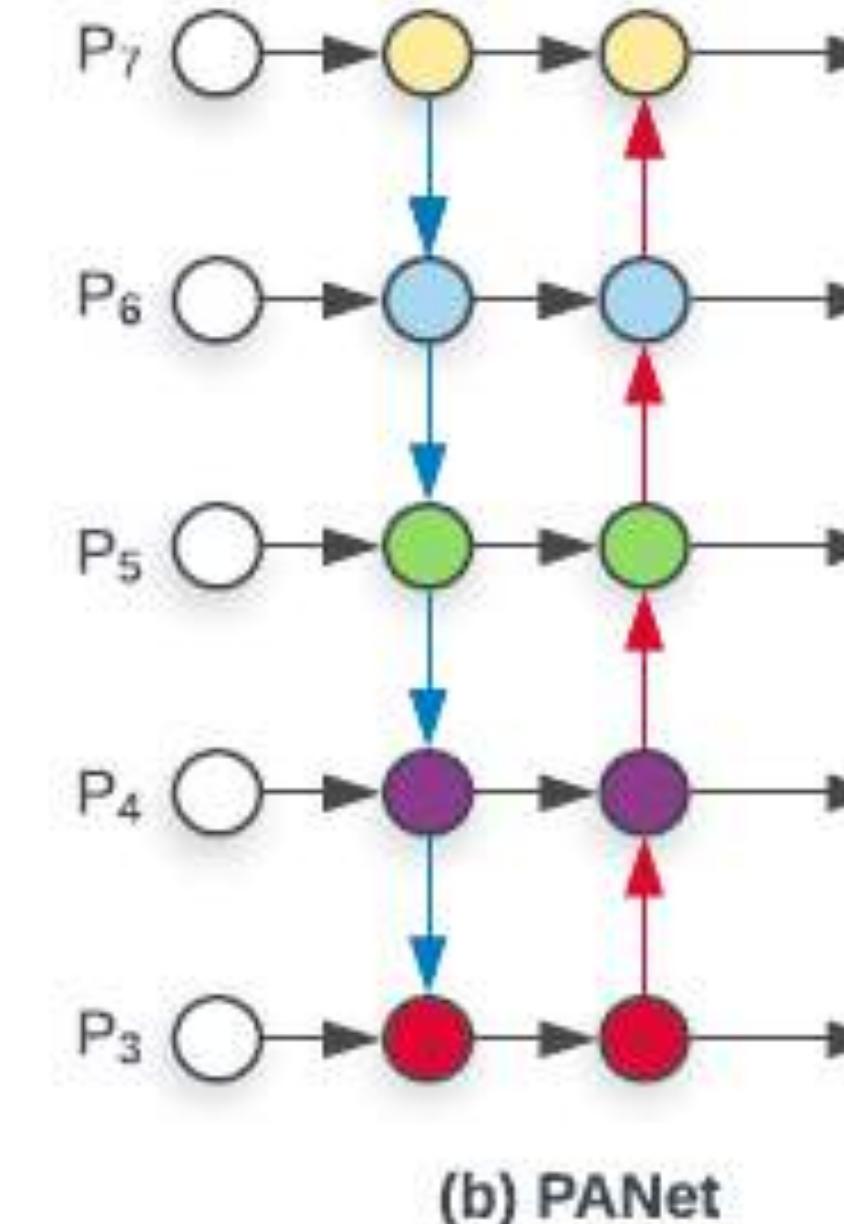
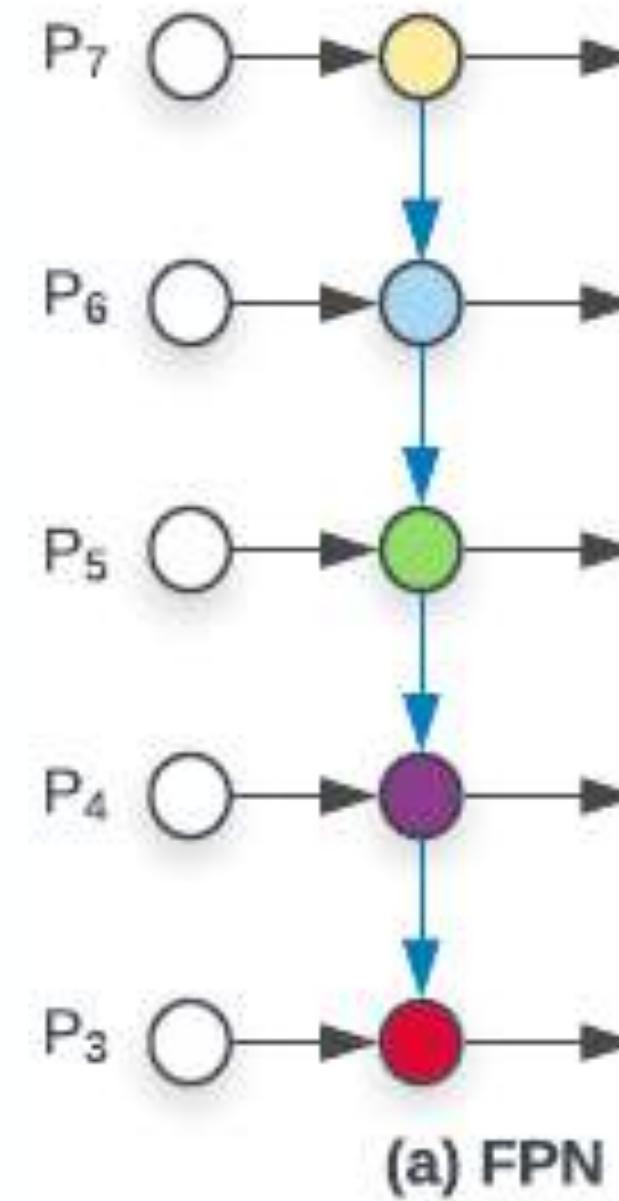
- The researchers (Google) behind EfficientDet wanted to see if it was possible to build a **scalable object detection architecture** with both high **accuracy** and better **efficiency** across a wide variety of **performance constraints**.
- EfficientDet is a **single shot** detector like SSD, RetinaNet and YOLO.
- It utilises philosophies from EfficientNet and made improvements to **model scaling** and **multi scale feature fusion**.

# EfficientDet Solves Two Problems

- The researchers sought to solve two main problems:
  - **Efficient multi-scale feature fusion** - Feature Pyramid Networks (FPN) have become the most popular method for fusing multi-scale features. However, they simply sum them up without any distinction and as we know not all features contribute equally to the output features thus needing a better strategy.
  - **Model Scaling** - most detectors relied on improving the backbone for improving accuracy, however, the researchers noted that scaling up feature network and box/class prediction networks are also important when considering accuracy and efficiency.

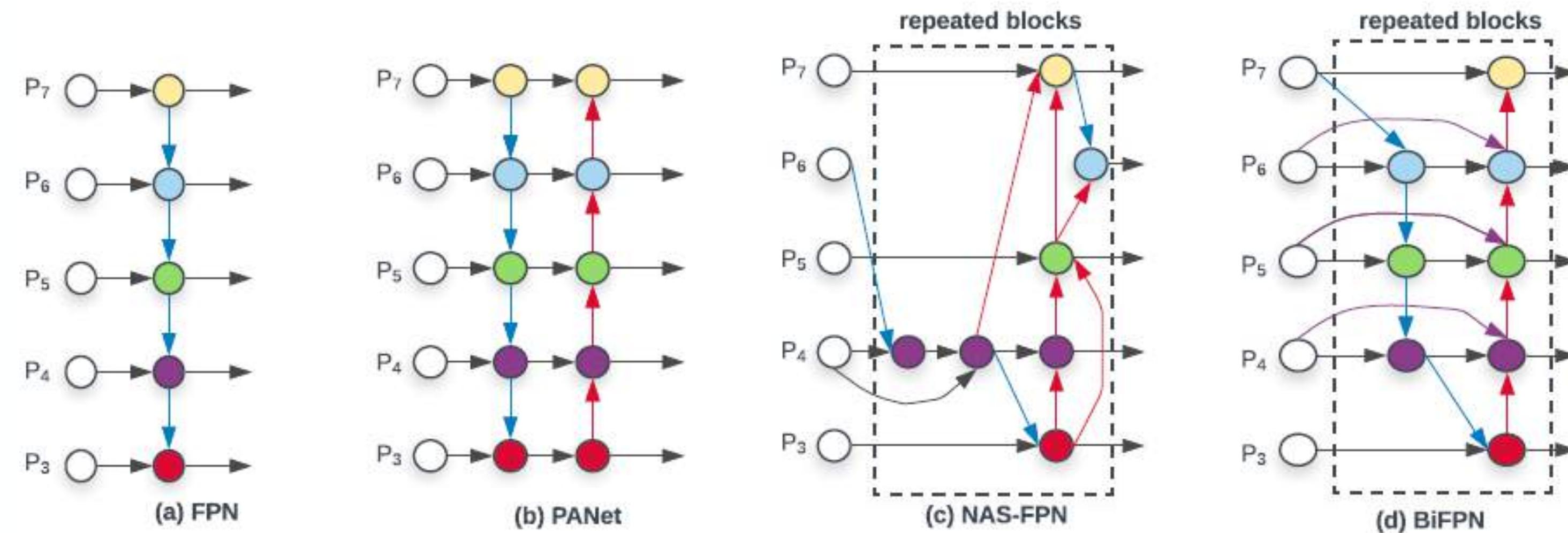
# EfficientDet

- EfficientDet uses a **BiFPN** architecture with **multi-scale feature fusion** which aims to **aggregate features at different resolutions**.

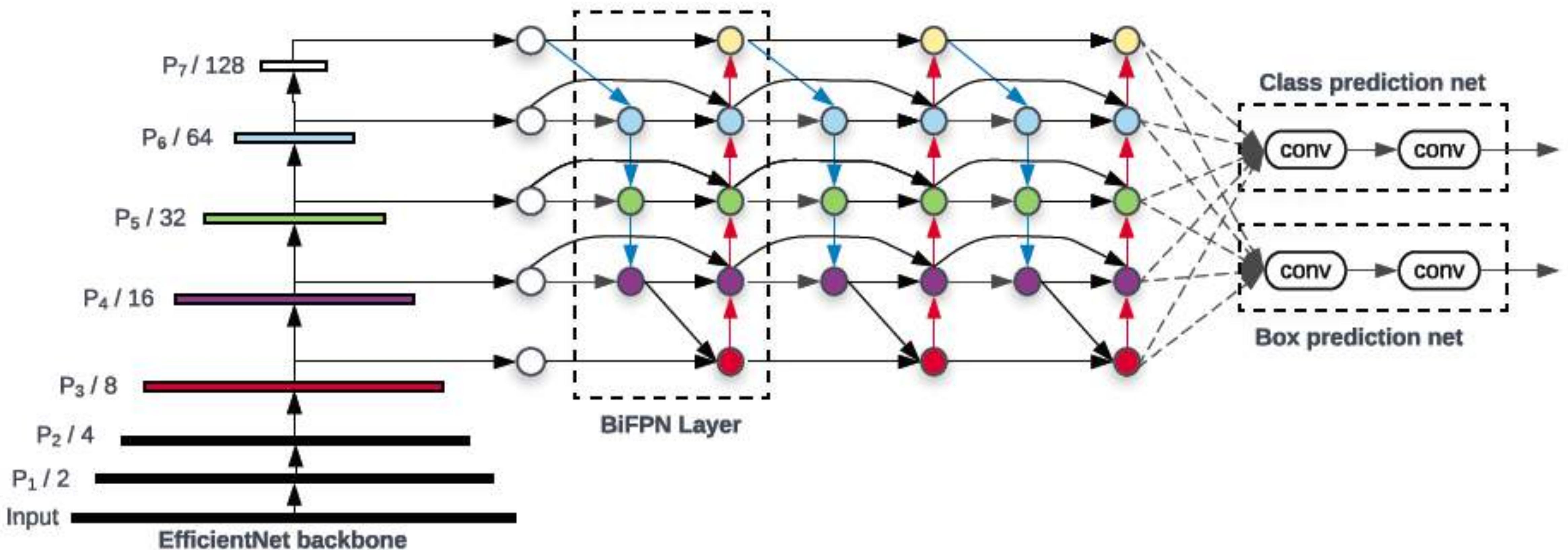


# EfficientDet - Why BiFPN?

- One of the FPN's limitations was that it was limited by the top-down information flow
- PANet adds an extra bottom-up path aggregation network
- NAS-FPN use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block
- However, in EfficientDet they use **BiFPN** which achieved better accuracy and efficiency.



# EfficientDet Architecture

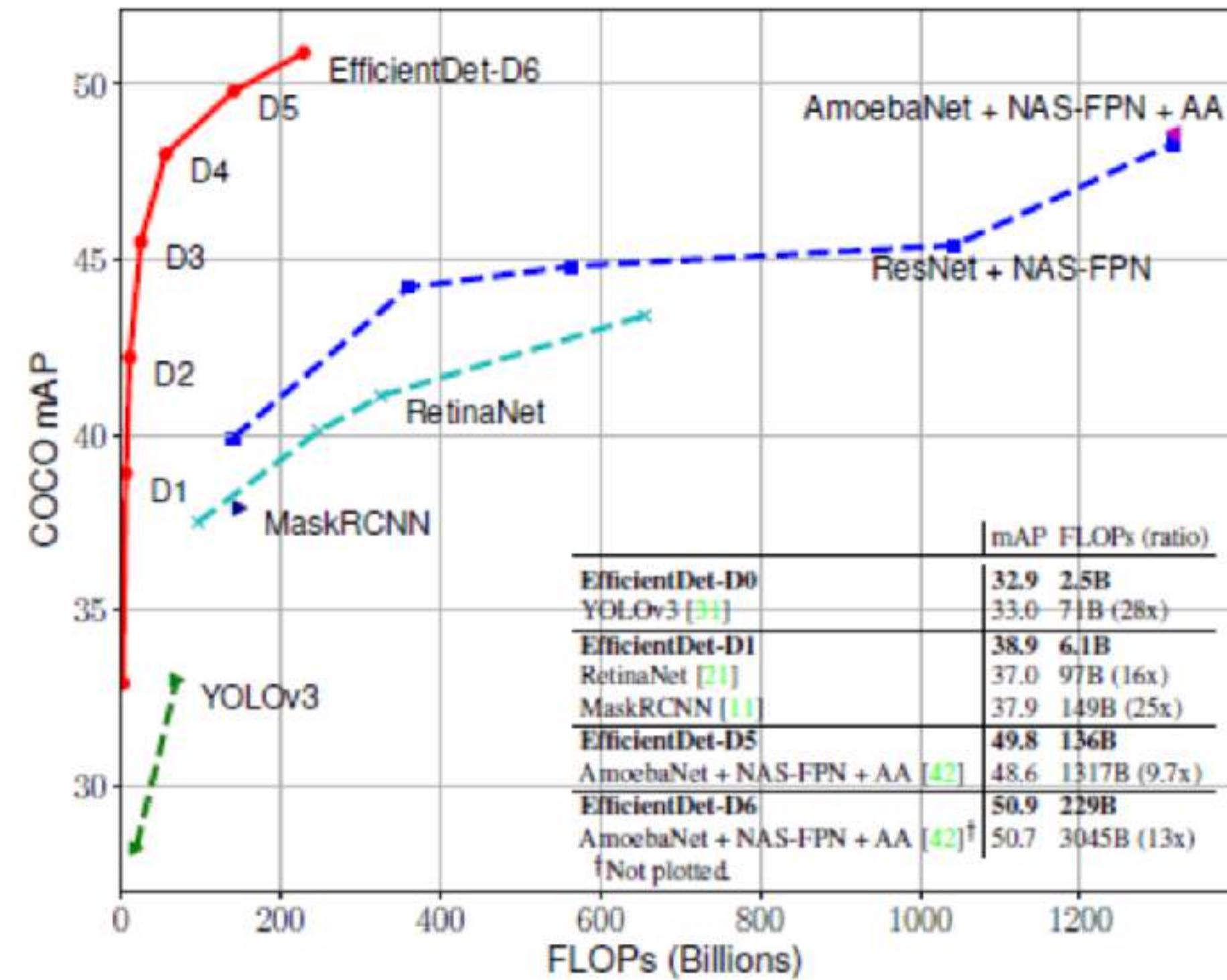


# EfficientDet Architecture

	Input size $R_{input}$	Backbone Network	BiFPN $W_{bifpn}$	Box/class net $D_{bifpn}$	Box/class $D_{class}$
D0 ( $\phi = 0$ )	512	B0	64	2	3
D1 ( $\phi = 1$ )	640	B1	88	3	3
D2 ( $\phi = 2$ )	768	B2	112	4	3
D3 ( $\phi = 3$ )	896	B3	160	5	4
D4 ( $\phi = 4$ )	1024	B4	224	6	4
D5 ( $\phi = 5$ )	1280	B5	288	7	4
D6 ( $\phi = 6$ )	1408	B6	384	8	5
D7	1536	B6	384	8	5

Table 1: **Scaling configs for EfficientDet D0-D7** –  $\phi$  is the compound coefficient that controls all other scaling dimensions; *BiFPN*, *box/class net*, and *input size* are scaled up using equation 1, 2, 3 respectively. D7 has the same settings as D6 except using larger input size.

# EfficientDet Performance



Model FLOPs vs COCO accuracy - All numbers are for single-model single-scale. Our EfficientDet achieves much better accuracy with fewer computations. In particular, EfficientDet-D6 achieves new state-of-the-art 50.9% COCO mAP with 4x fewer parameters and 13x fewer FLOPs than prior models.

Source Credit :<https://arxiv.org/abs/1911.09070>

# EfficientDet Performance

Model	mAP	#Params	Ratio	#FLOPS	Ratio	GPU LAT(ms)	Speedup	CPU LAT(s)	Speedup
<b>EfficientDet-D0</b>	<b>32.4</b>	<b>3.9M</b>	<b>1x</b>	<b>2.5B</b>	<b>1x</b>	<b>16 ±1.6</b>	<b>1x</b>	<b>0.32 ±0.002</b>	<b>1x</b>
YOLOv3 [26]	33.0	-	-	71B	28x	51 <sup>†</sup>	-	-	-
<b>EfficientDet-D1</b>	<b>38.3</b>	<b>6.6M</b>	<b>1x</b>	<b>6B</b>	<b>1x</b>	<b>20 ±1.1</b>	<b>1x</b>	<b>0.74 ±0.003</b>	<b>1x</b>
MaskRCNN [8]	37.9	44.4M	6.7x	149B	25x	92 <sup>†</sup>	-	-	-
RetinaNet-R50 (640) [17]	37.0	34.0M	6.7x	97B	16x	27 ±1.1	1.4x	2.8 ±0.017	3.8x
RetinaNet-R101 (640) [17]	37.9	53.0M	8x	127B	21x	34 ±0.5	1.7x	3.6 ±0.012	4.9x
<b>EfficientDet-D2</b>	<b>41.1</b>	<b>8.1M</b>	<b>1x</b>	<b>11B</b>	<b>1x</b>	<b>24 ±0.5</b>	<b>1x</b>	<b>1.2 ±0.003</b>	<b>1x</b>
RetinaNet-R50 (1024) [17]	40.1	34.0M	4.3x	248B	23x	51 ±0.9	2.0x	7.5 ±0.006	6.3x
RetinaNet-R101 (1024) [17]	41.1	53.0M	6.6x	326B	30x	65 ±0.4	2.7x	9.7 ±0.038	8.1x
NAS-FPN R-50 (640) [5]	39.9	60.3M	7.5x	141B	13x	41 ±0.6	1.7x	4.1 ±0.027	3.4x
<b>EfficientDet-D3</b>	<b>44.3</b>	<b>12.0M</b>	<b>1x</b>	<b>25B</b>	<b>1x</b>	<b>42 ±0.8</b>	<b>1x</b>	<b>2.5 ±0.002</b>	<b>1x</b>
NAS-FPN R-50 (1024) [5]	44.2	60.3M	5.1x	360B	15x	79 ±0.3	1.9x	11 ±0.063	4.4x
NAS-FPN R-50 (1280) [5]	44.8	60.3M	5.1x	563B	23x	119 ±0.9	2.8x	17 ±0.150	6.8x
<b>EfficientDet-D4</b>	<b>46.6</b>	<b>20.7M</b>	<b>1x</b>	<b>55B</b>	<b>1x</b>	<b>74 ±0.5</b>	<b>1x</b>	<b>4.8 ±0.003</b>	<b>1x</b>
NAS-FPN R50 (1280@384)	45.4	104 M	5.1x	1043B	19x	173 ±0.7	2.3x	27 ±0.056	5.6x
<b>EfficientDet-D5 + AA</b>	<b>49.8</b>	<b>33.7M</b>	<b>1x</b>	<b>136B</b>	<b>1x</b>	<b>141 ±2.1</b>	<b>1x</b>	<b>11 ±0.002</b>	<b>1x</b>
AmoebaNet+ NAS-FPN + AA(1280) [37]	48.6	185M	5.5x	1317B	9.7x	259 ±1.2	1.8x	38 ±0.084	3.5x
<b>EfficientDet-D6 + AA</b>	<b>50.6</b>	<b>51.9M</b>	<b>1x</b>	<b>227B</b>	<b>1x</b>	<b>190 ±1.1</b>	<b>1x</b>	<b>16 ±0.003</b>	<b>1x</b>
AmoebaNet+ NAS-FPN + AA(1536) [37]	50.7	209M	4.0x	3045B	13x	608 ±1.4	3.2x	83 ±0.092	5.2x
<b>EfficientDet-D7 + AA</b>	<b>51.0</b>	<b>51.9M</b>	<b>1x</b>	<b>326B</b>	<b>1x</b>	<b>262 ±2.2</b>	<b>1x</b>	<b>24 ±0.003</b>	<b>1x</b>

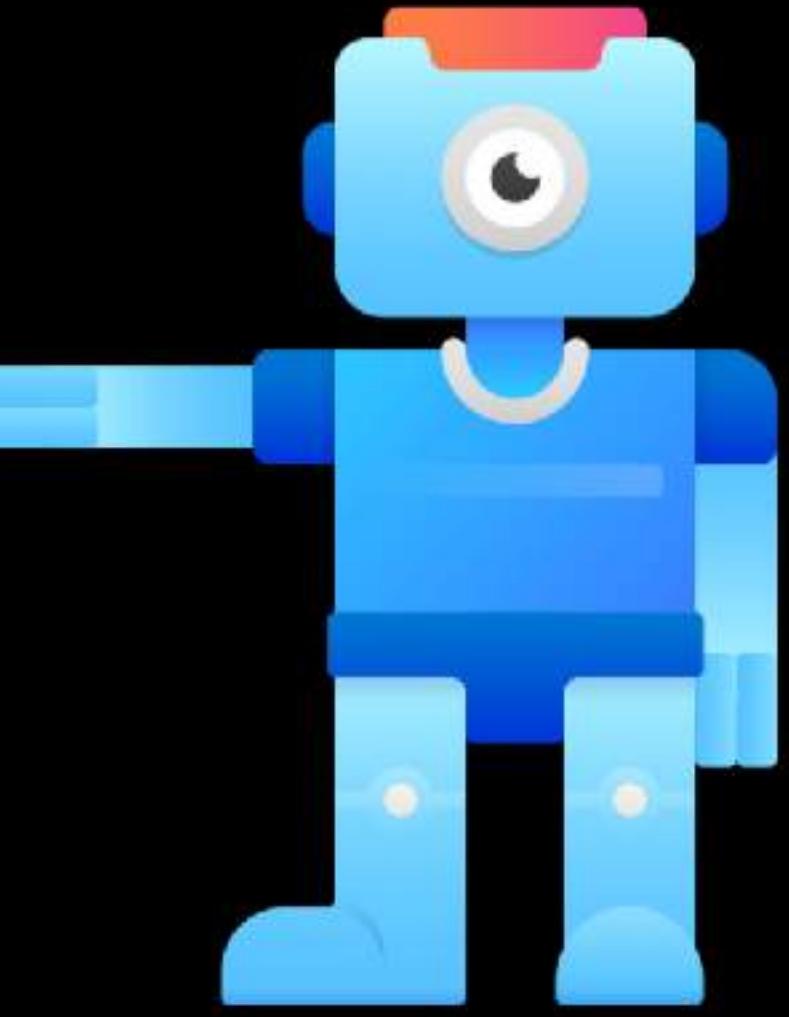


**MODERN  
COMPUTER  
VISION**

BY RAJEEV RATAN

# Next...

**Detectron2 - Object Detection**



# MODERN COMPUTER VISION

BY RAJEEV RATAN

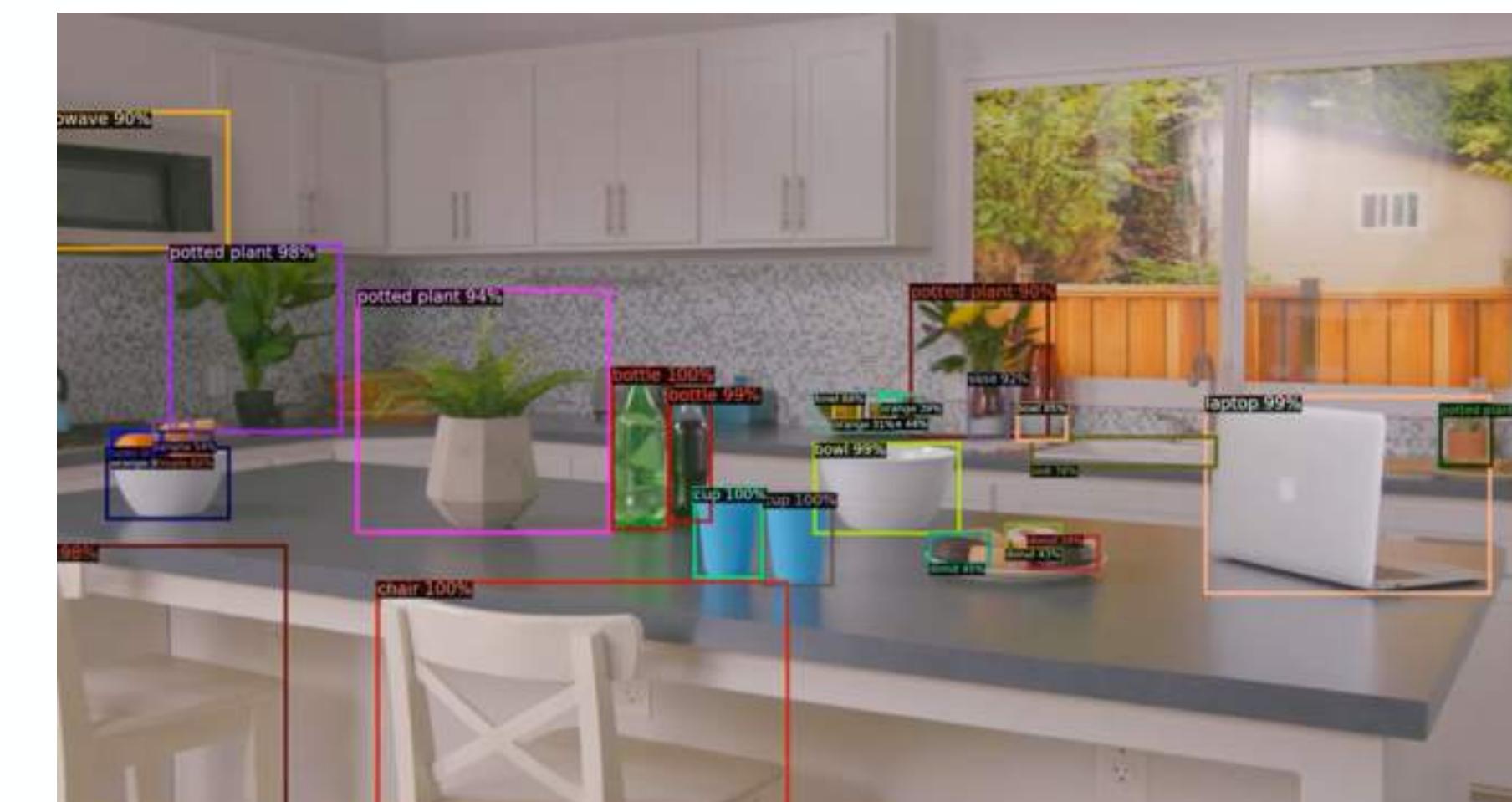
## Detectron2 - Object Detection

We take a look at Detectron2 for Object Detection

# Detectron2



- A **research platform** and **production library** for **object detection**.
- Developed and open sourced by **Facebook AI Research (FAIR)**
- Detectron2 takes over from the original Detectron model.
- It is faster, developed on PyTorch (original was made on Caffe2), more accurate and more modular
- Detectron2 features capabilities such as:
  - Cascade R-CNN
  - Mask R-CNN
  - PointRend
  - DensePose
  - DeepLab



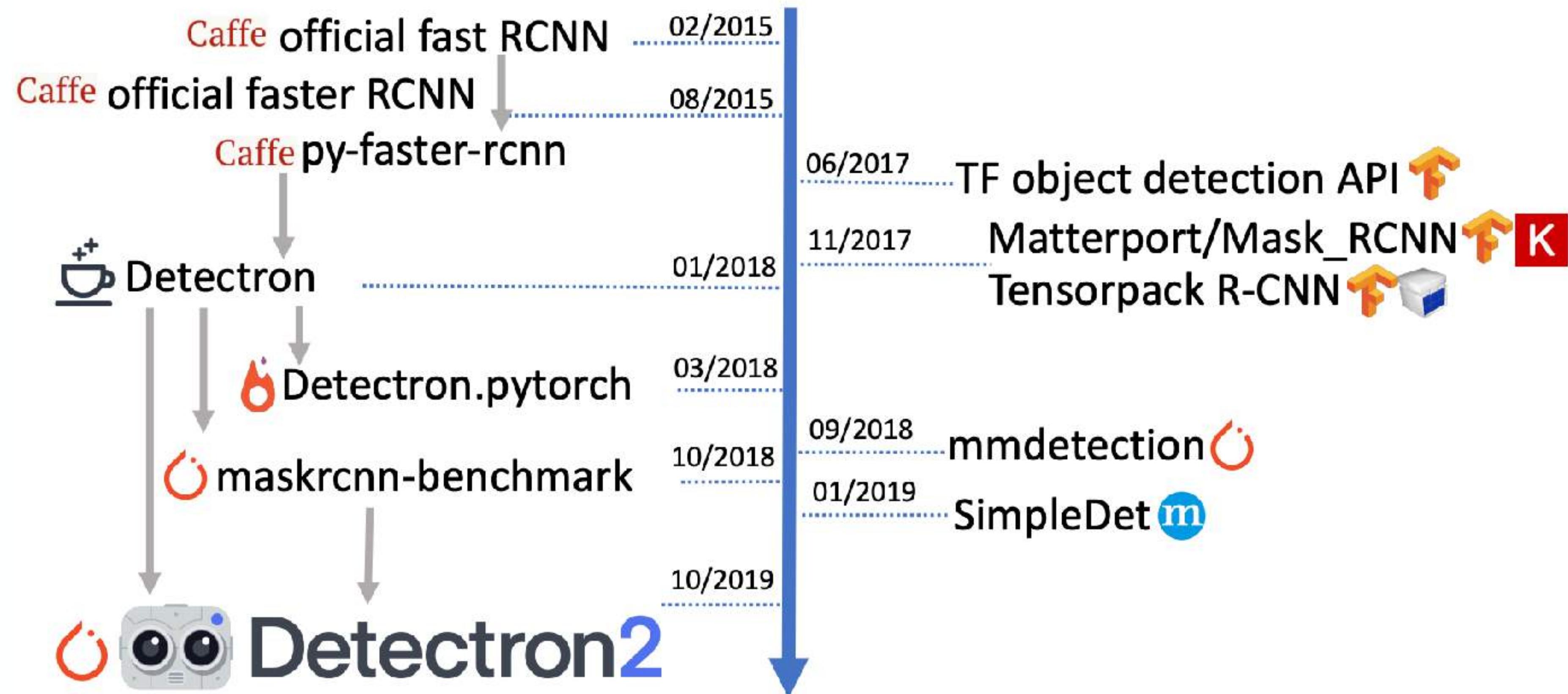
# What can it do?



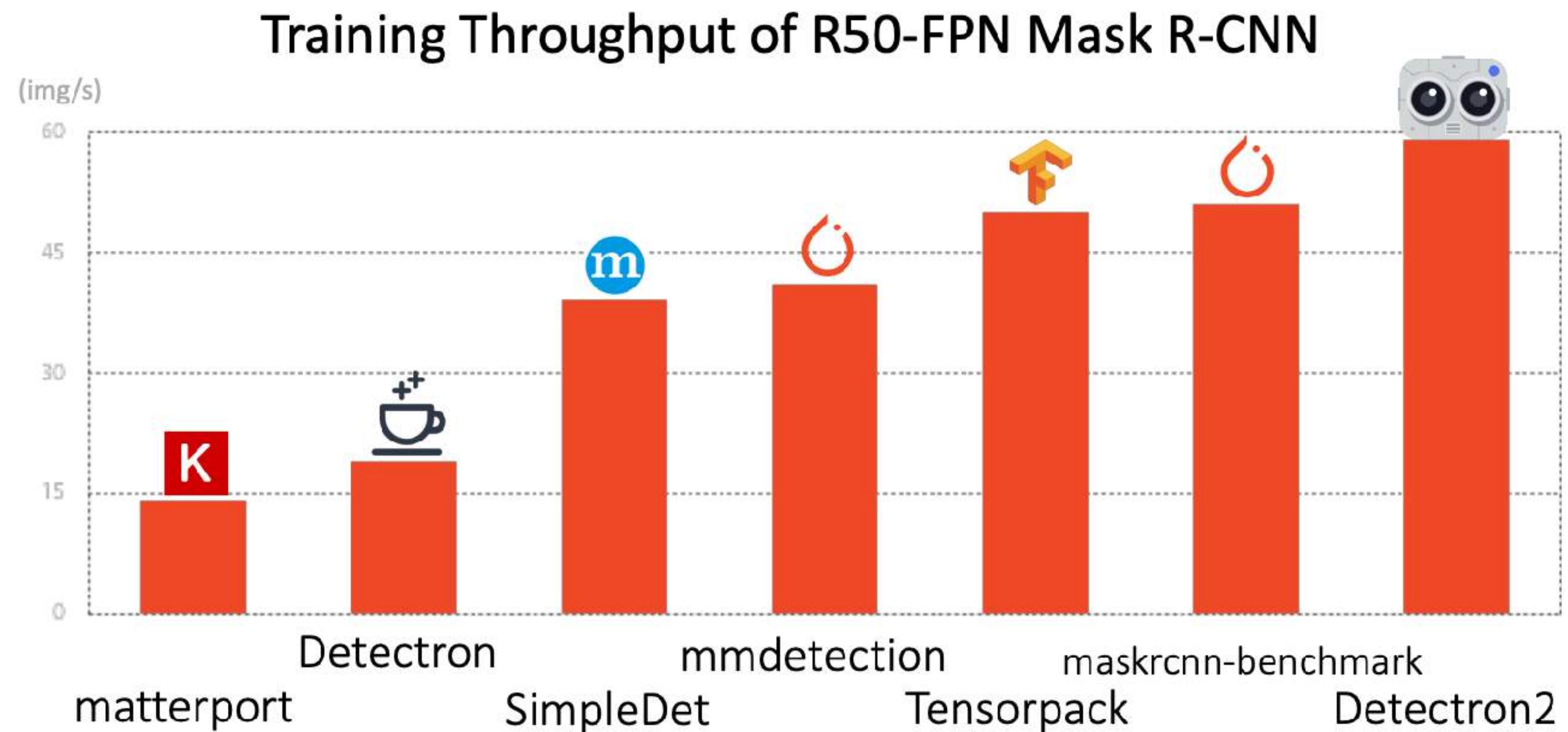
# Detectron2



# A timeline of Object Detectors



# Detectron2 is Faster to Train



Details at: [detectron2.readthedocs.io/en/latest/notes/benchmarks.html](https://detectron2.readthedocs.io/en/latest/notes/benchmarks.html)

# Detectron2 Segmentation

- As we've seen Detectron2 has Object Detection, Instance Segmentation masks, human pose prediction and semantic segmentation and panoptic segmentation (combines both Semantic and Instance Segmentation)

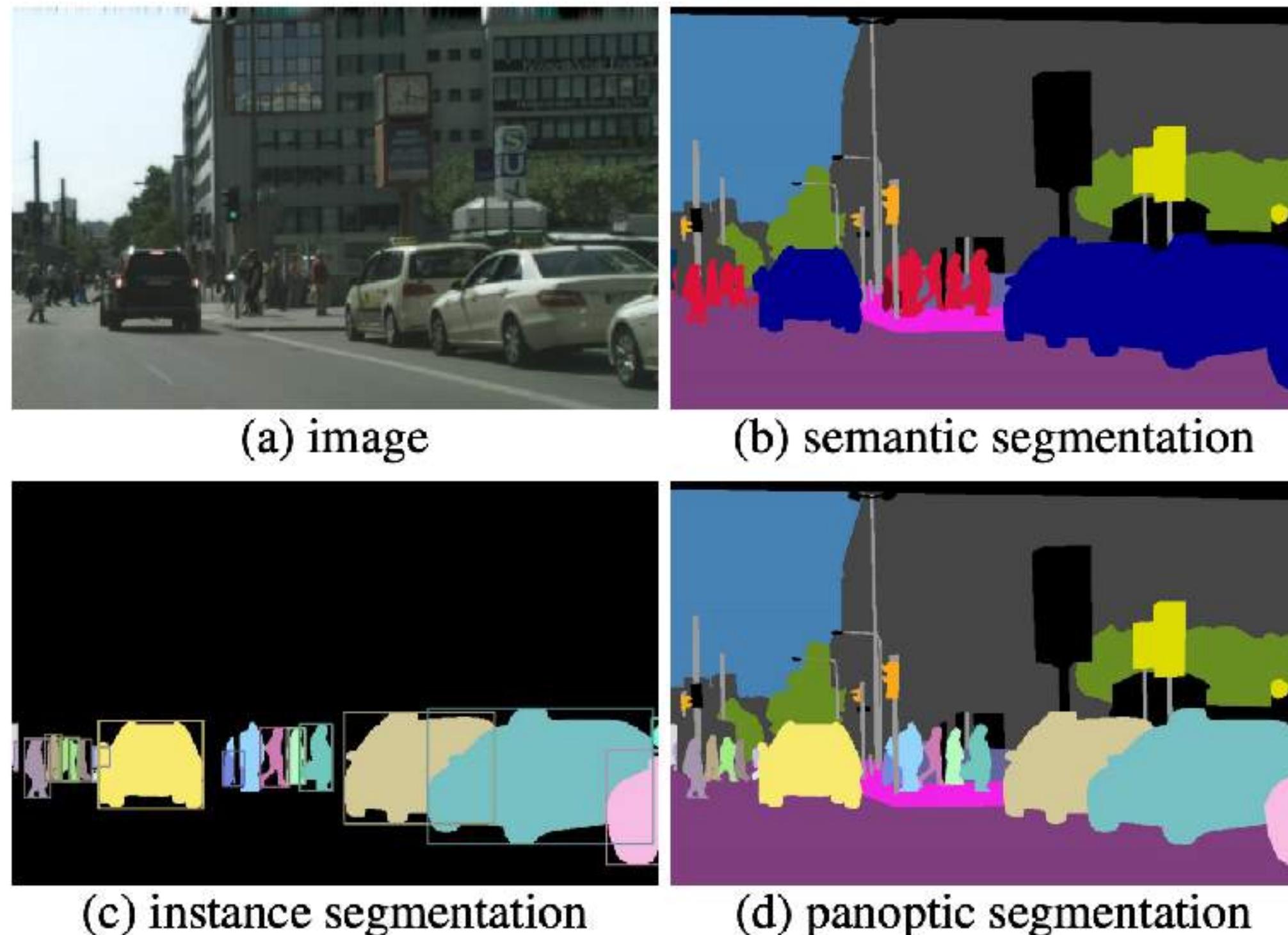


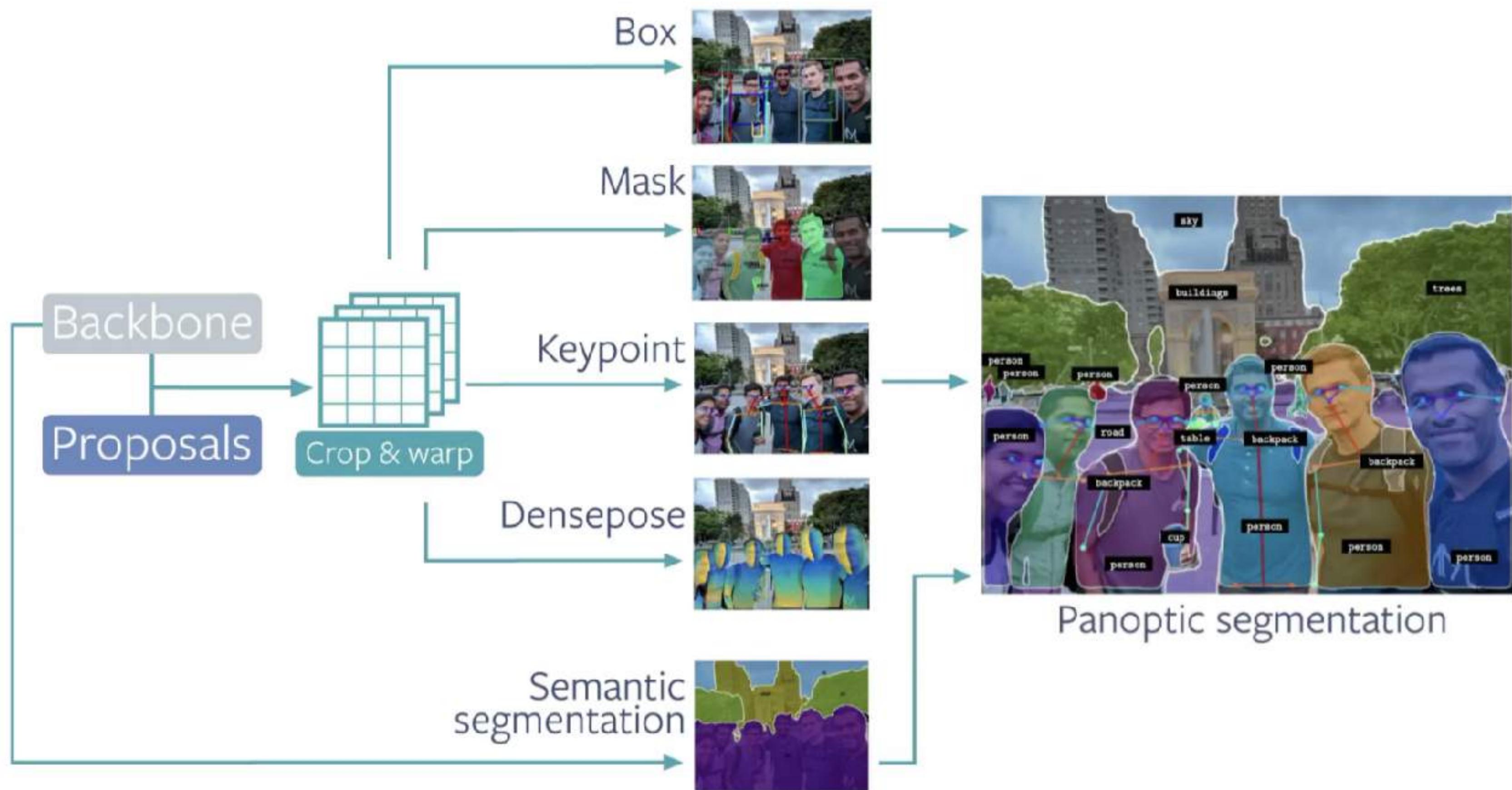
Figure 1: For a given (a) image, we show *ground truth* for: (b) semantic segmentation (per-pixel class labels), (c) instance segmentation (per-object mask and class label), and (d) the proposed *panoptic segmentation* task (per-pixel class+instance labels). The PS task: (1) encompasses both stuff and thing classes, (2) uses a simple but general format, and (3) introduces a uniform evaluation metric for all classes. Panoptic segmentation generalizes both semantic and instance segmentation and we expect the unified task will present novel challenges and enable innovative new methods.

<https://arxiv.org/pdf/1801.00868.pdf>

# Detectron2 Deployment

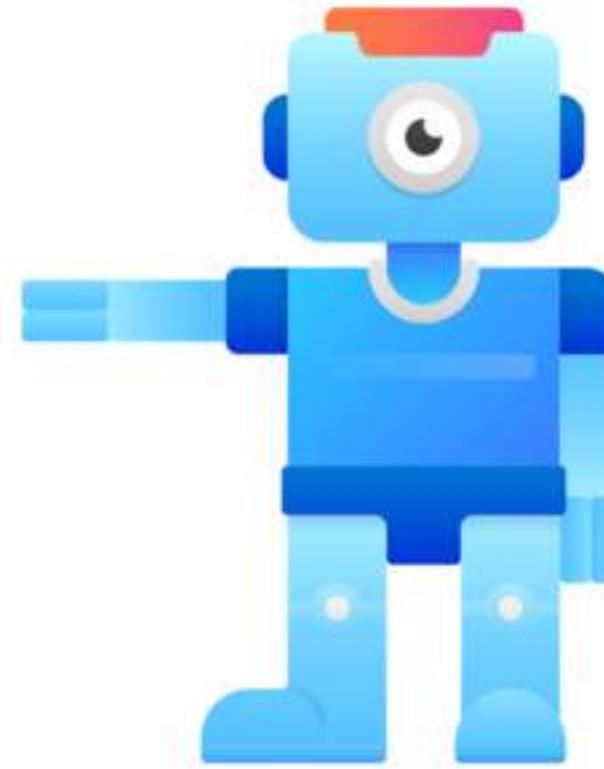
- **Detectron2go** - Facebook AI's computer vision engineers have implemented an additional software layer, **Detectron2go**.
- It makes it easier to deploy advanced new models to production.
- These features include standard training workflows with in-house datasets, network quantisation, and model conversion to optimised formats for cloud and mobile deployment.

# Detectron2 - Generalised R-CNN Models



# Detectron2 Customisation

- Detectron2's framework allows for each custom:
  - Models
  - Datasets
  - Data Loaders
  - Augmentation
  - Various Metrics/Tasks
  - Training logic

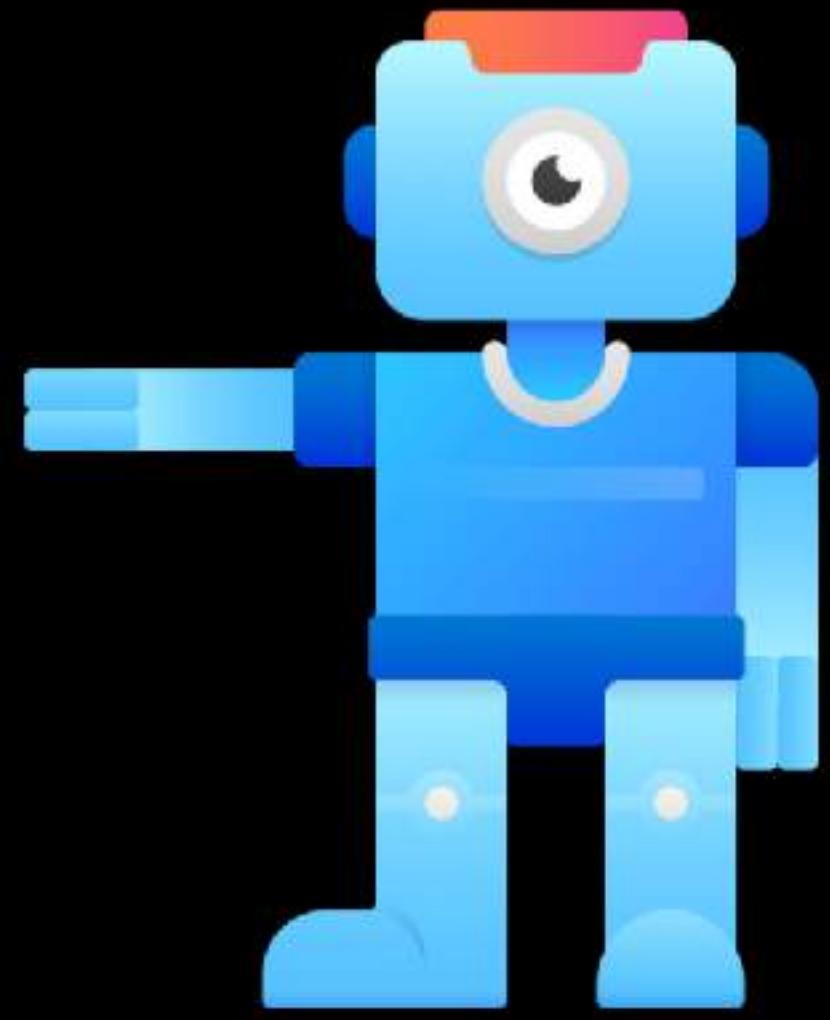


# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Next...

**Object Detection Projects! YOLO, Detectron2, EfficientDet, MobileNetSSD & Faster R-CNNs!**



# MODERN COMPUTER VISION

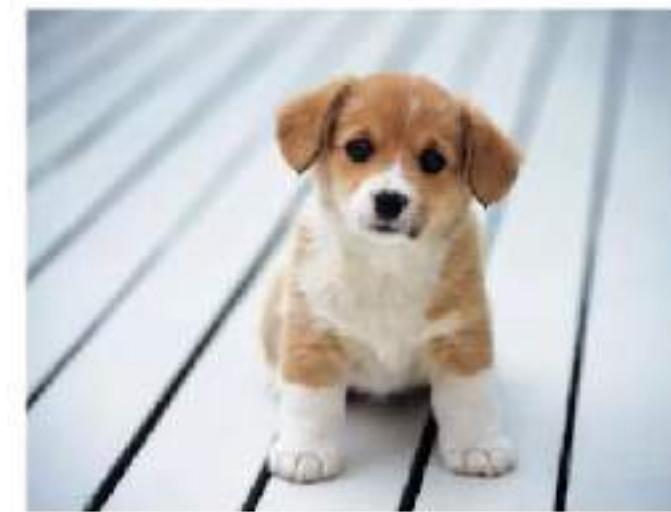
BY RAJEEV RATAN

## Segmentation with Deep Learning

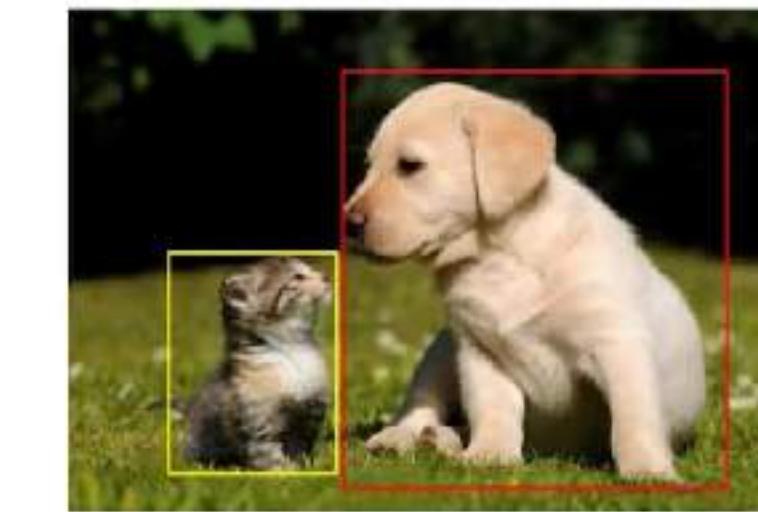
Methods of Image Segmentation Using Deep Learning

# What is Segmentation?

Is this a dog?



What is there in image  
and where?



Which pixels belong to  
which object?

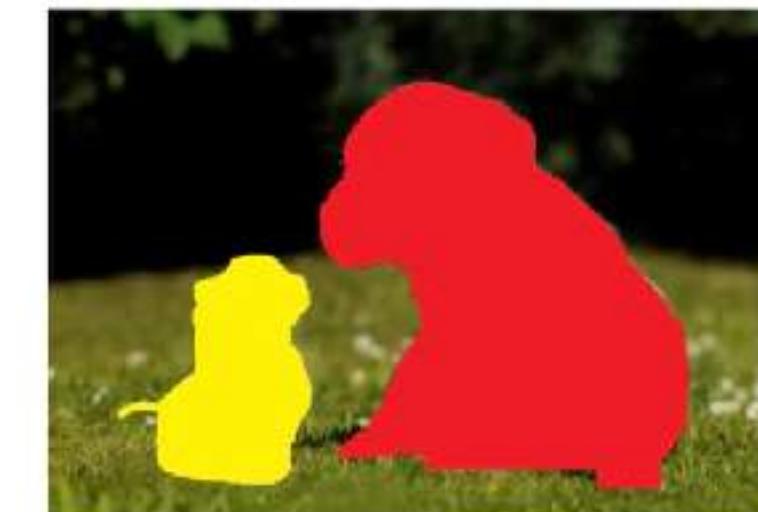


Image Classification

Object Detection

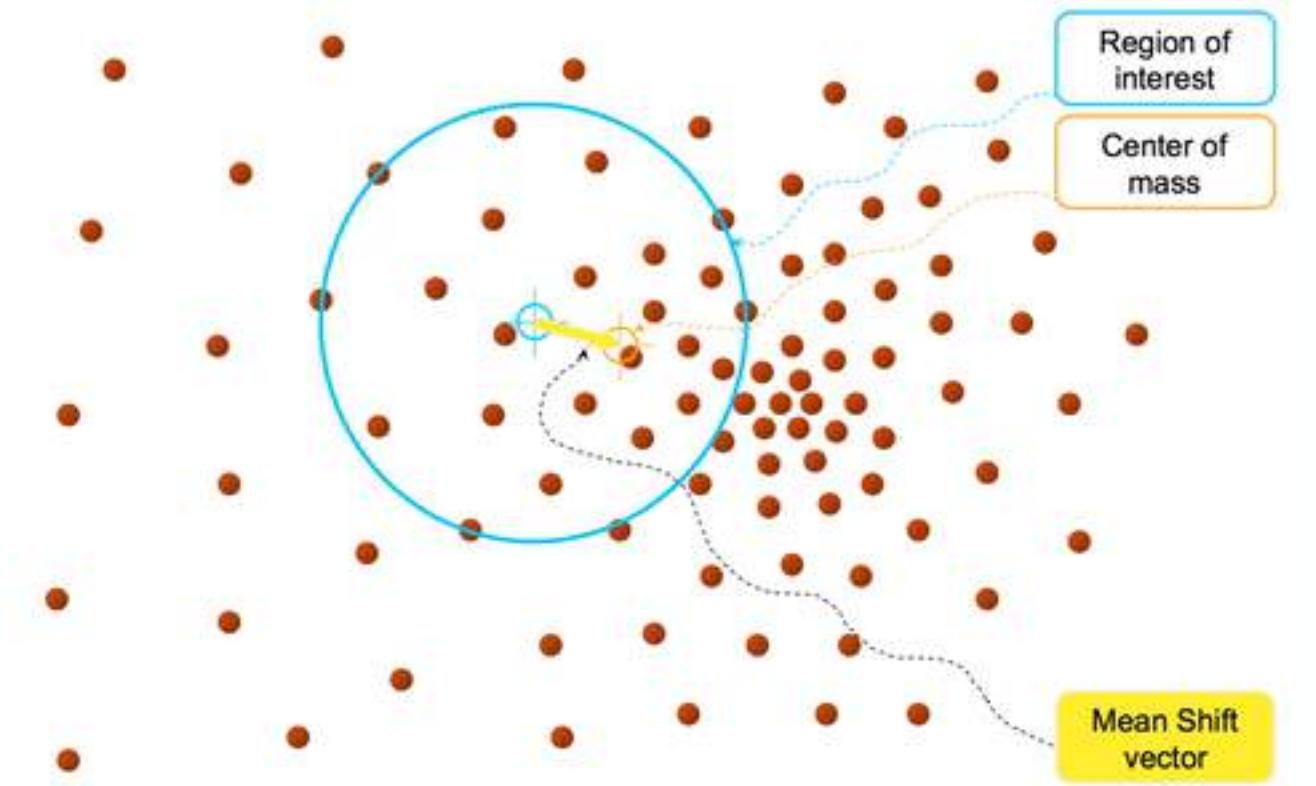
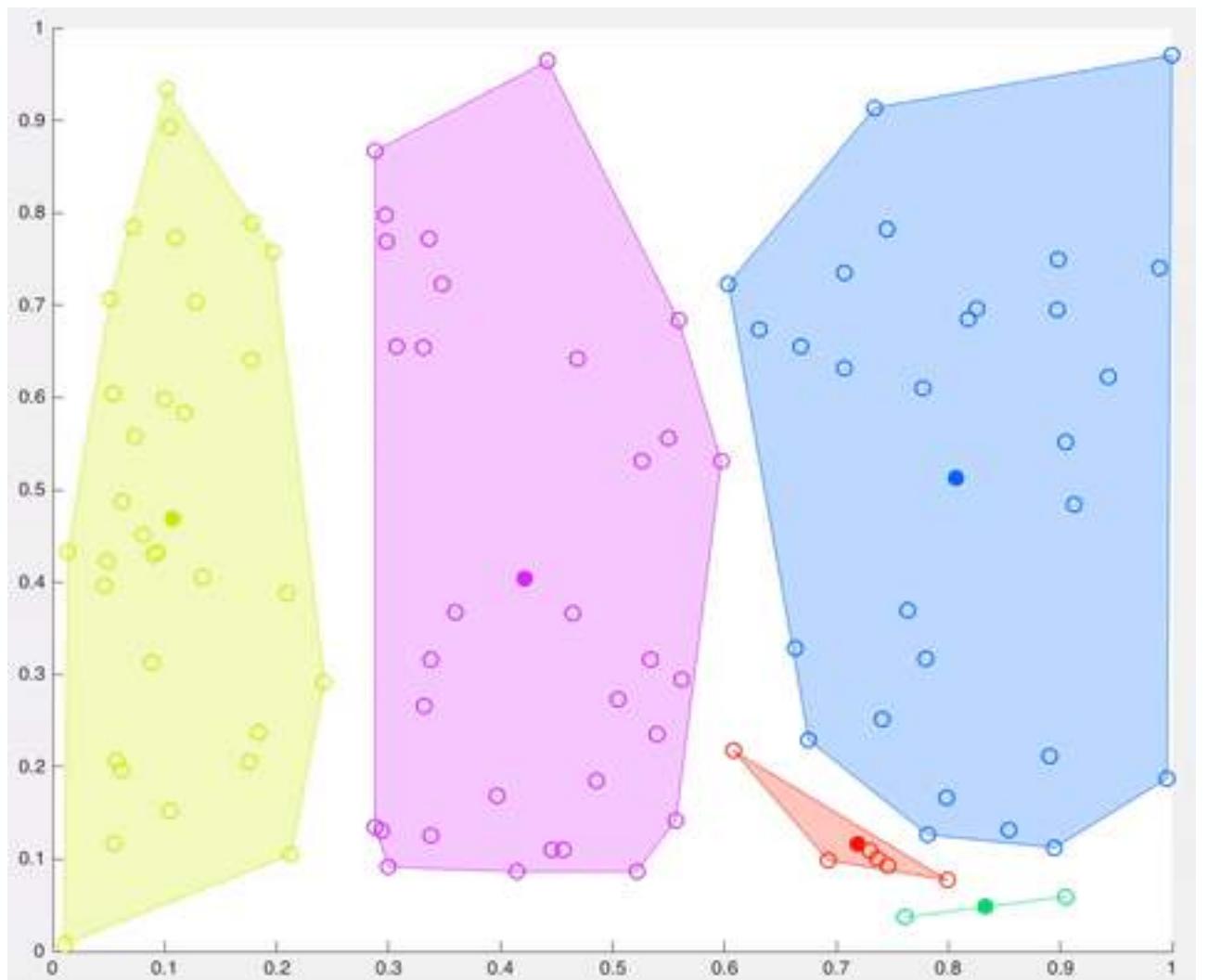
Image Segmentation

<https://www.thespuzz.com/new-deep-finding-out-model-brings-image-segmentation-to-edge-devices/>

- It is the process of **dividing an image into different regions** based on the **characteristics of pixels** to identify objects or boundaries to simplify an image and more efficiently analyse it.
- Think of it as predicting the class for each pixel in an image.

# Segmentation Techniques

- K-means is a widely used technique.
- Mean Shift is also a popular method of segmentation.



# K-Means Example

- Using K-means to extract/segment a cat

Picture and background:



Segmented: The pixels are partitioned into five groups, shown below. You can select the groups that form the kitten portion, discard the background (image 5, in this case), and move the segmented parts onto a separate background.



Final result: A kitty sleeping on a leaf!



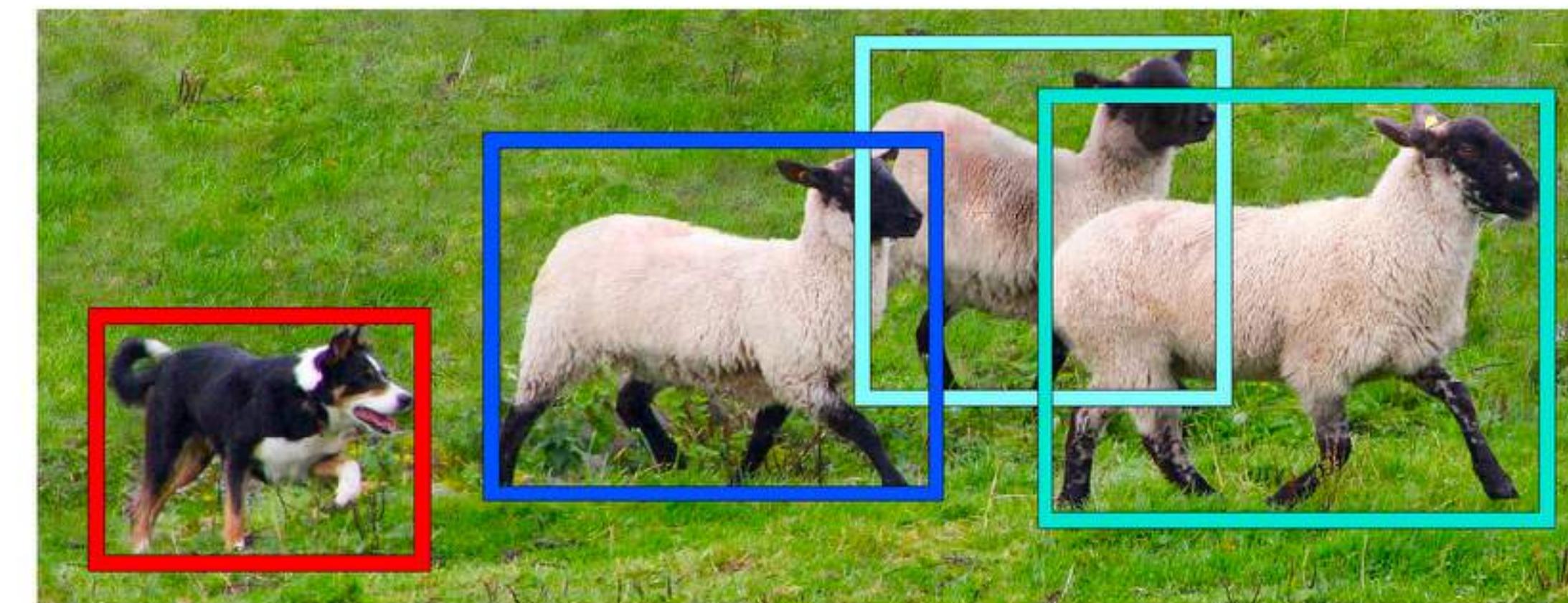
# These methods don't work well for Multiple Classes and cluttered Scenes



# Image Classification/Recognition vs Object Detection

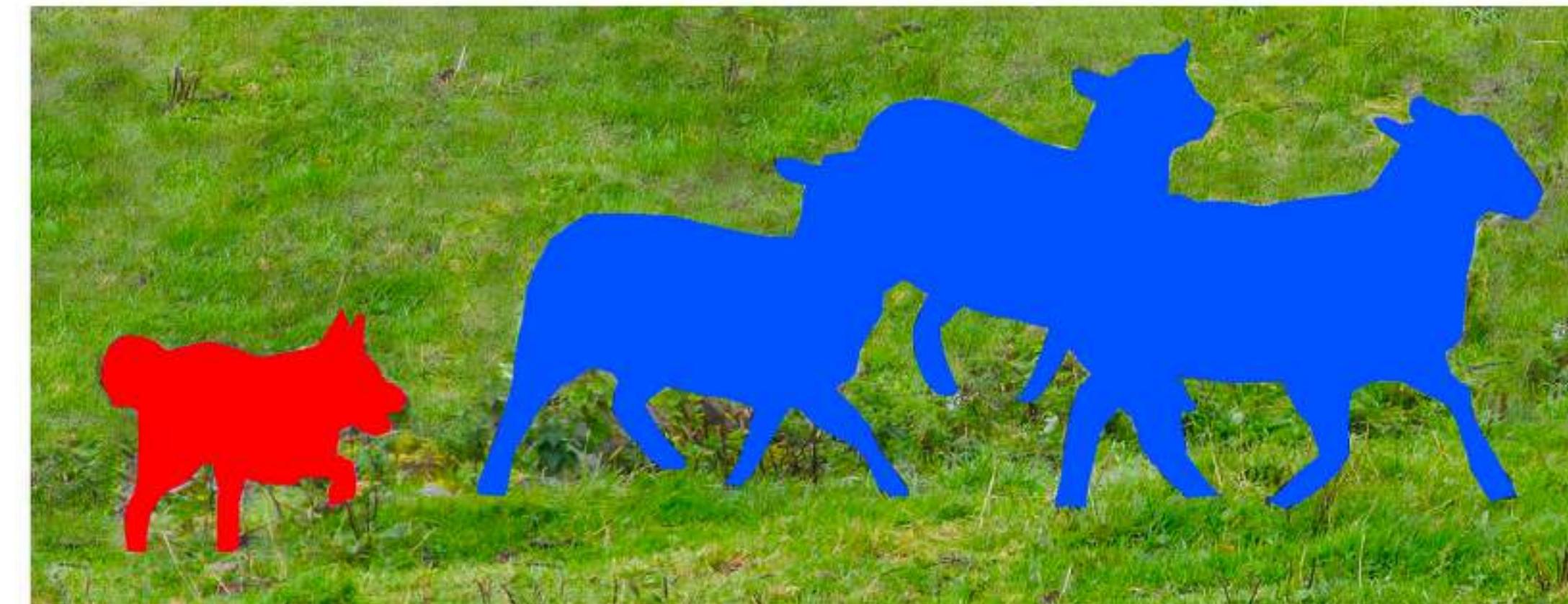


Image Recognition

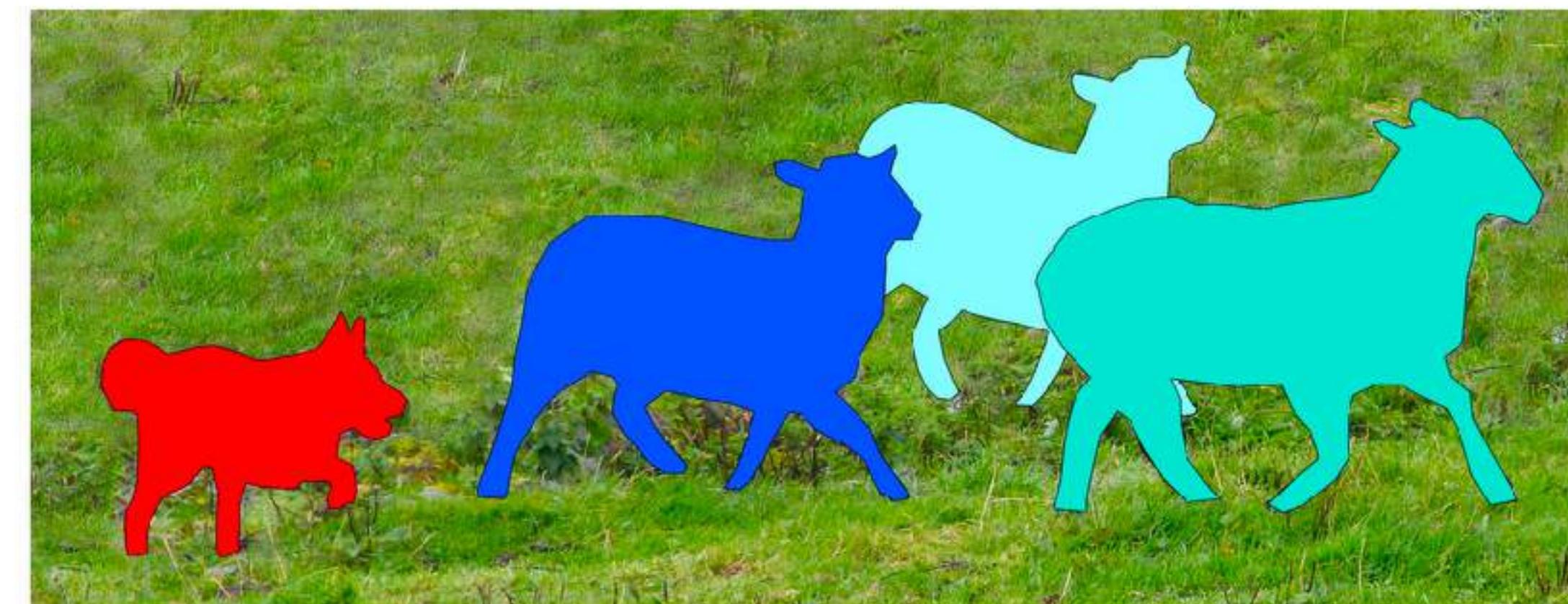


Object Detection

# Semantic vs Instance Segmentation

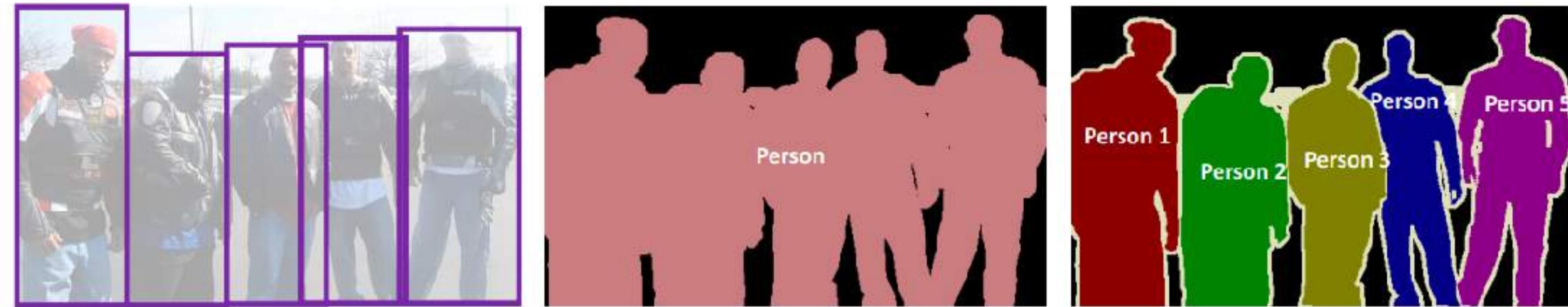


**Semantic Segmentation**



**Instance Segmentation**

# Semantic vs Instance Segmentation

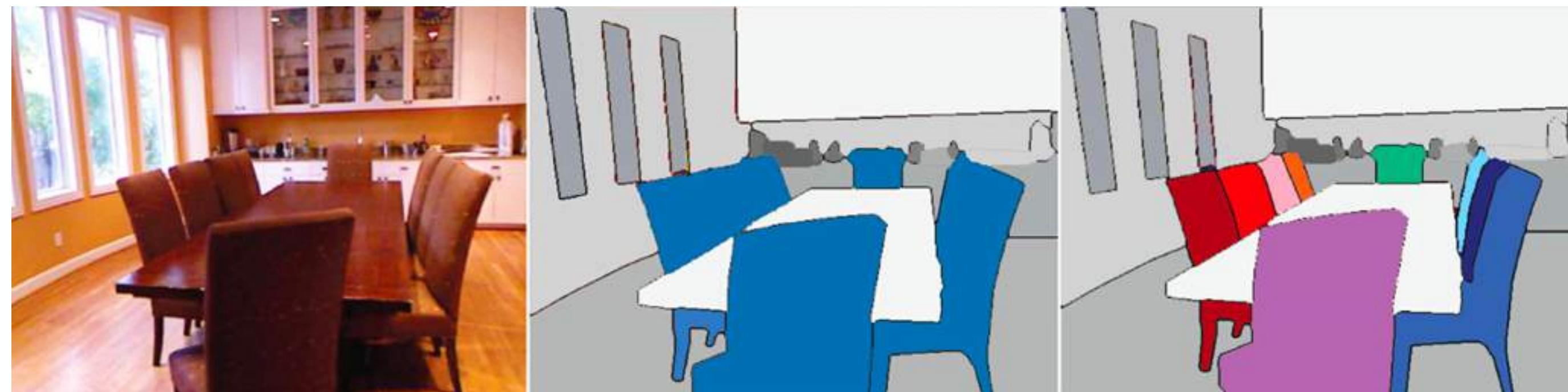


Object Detection

Semantic Segmentation

Instance Segmentation

[https://www.researchgate.net/figure/Semantic-segmentation-left-and-Instance-segmentation-right-8\\_fig1\\_339328277](https://www.researchgate.net/figure/Semantic-segmentation-left-and-Instance-segmentation-right-8_fig1_339328277)



Input Image

Semantic Segmentation

Instance Segmentation

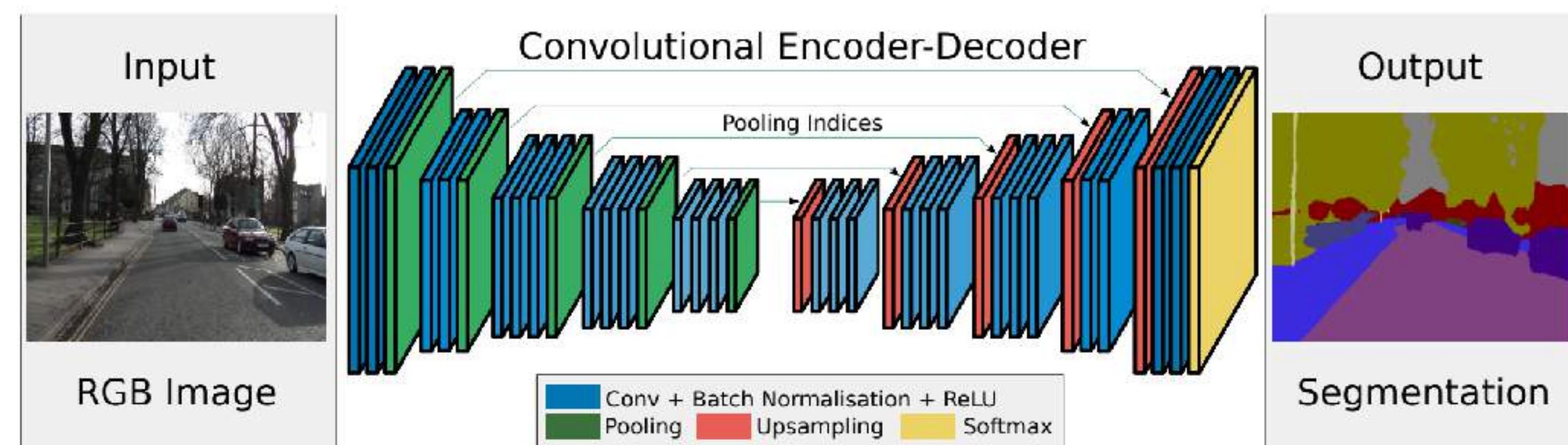
<https://datascience.stackexchange.com/questions/52015/what-is-the-difference-between-semantic-segmentation-object-detection-and-insta>

# Popular Deep Learning Segmentation Models

- SegNet
- U-Net
- DeepLabV3
- Mask R-CNN

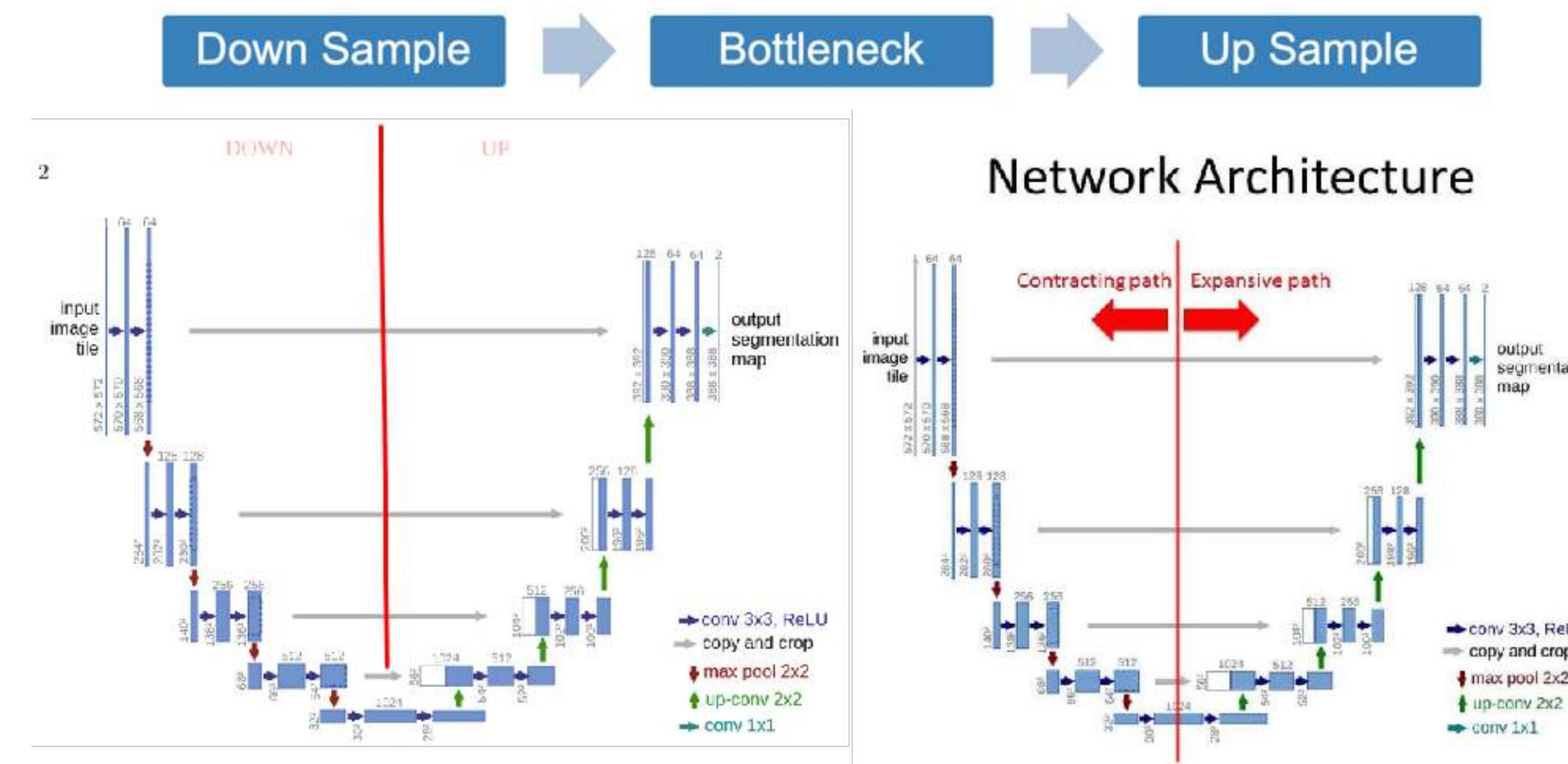
# SegNet

- Developed in 2015, SegNet is a semantic segmentation model.
- It consists of an encoder and decoder network followed by a pixel-wise classification layer.
- The architecture of the encoder network is topologically identical to the 13 convolutional layers in the VGG16 network.
- The role of the decoder network is to map the low resolution encoder feature maps to full input resolution feature maps for pixel-wise classification.



# U-Net

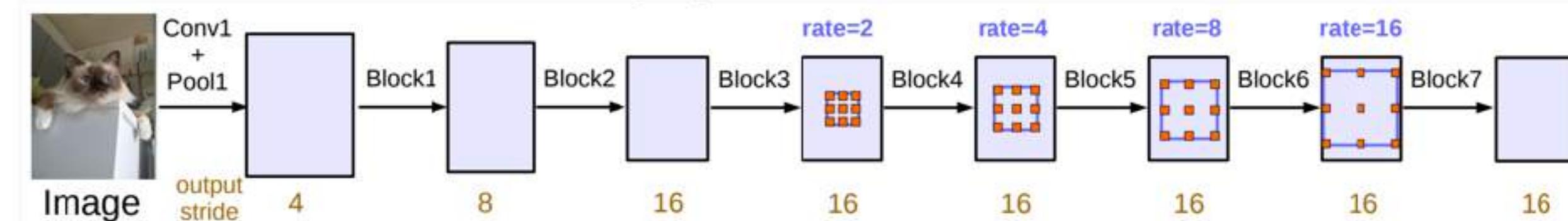
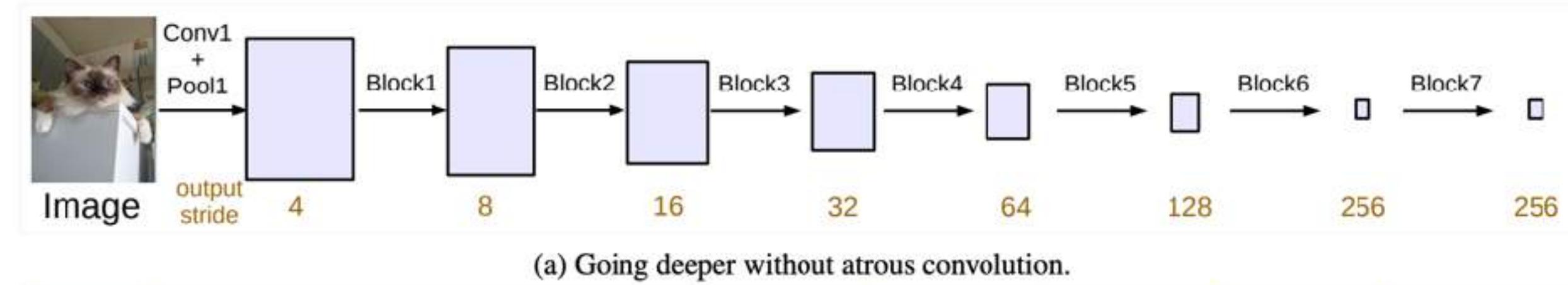
- Created in 2015, U-Net was a unique CNN developed for Biomedical Image Segmentation.
- U-Net has now become a very popular end-to-end encoder-decoder network for semantic segmentation.
- It has a unique Up-Down architecture which has a Contracting path and an Expansive path.



# DeepLabV3

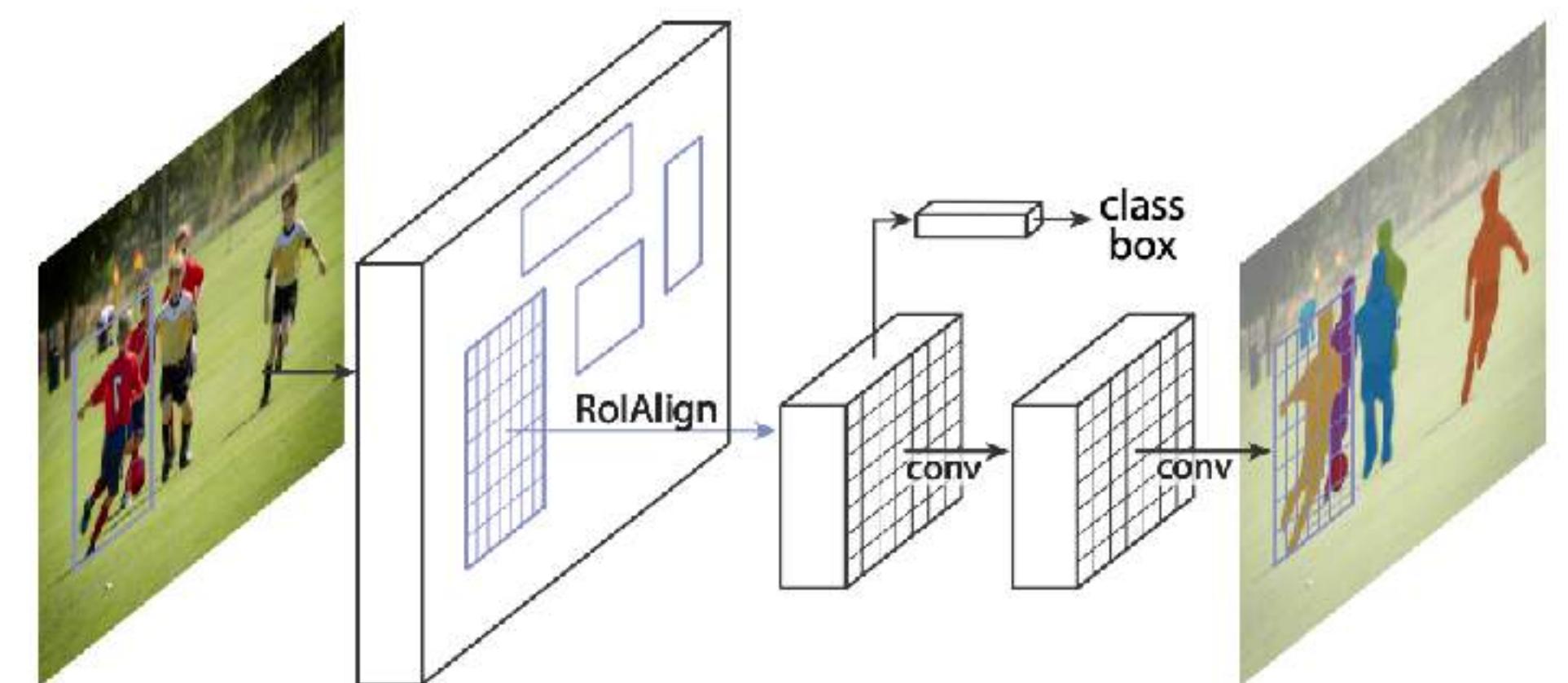


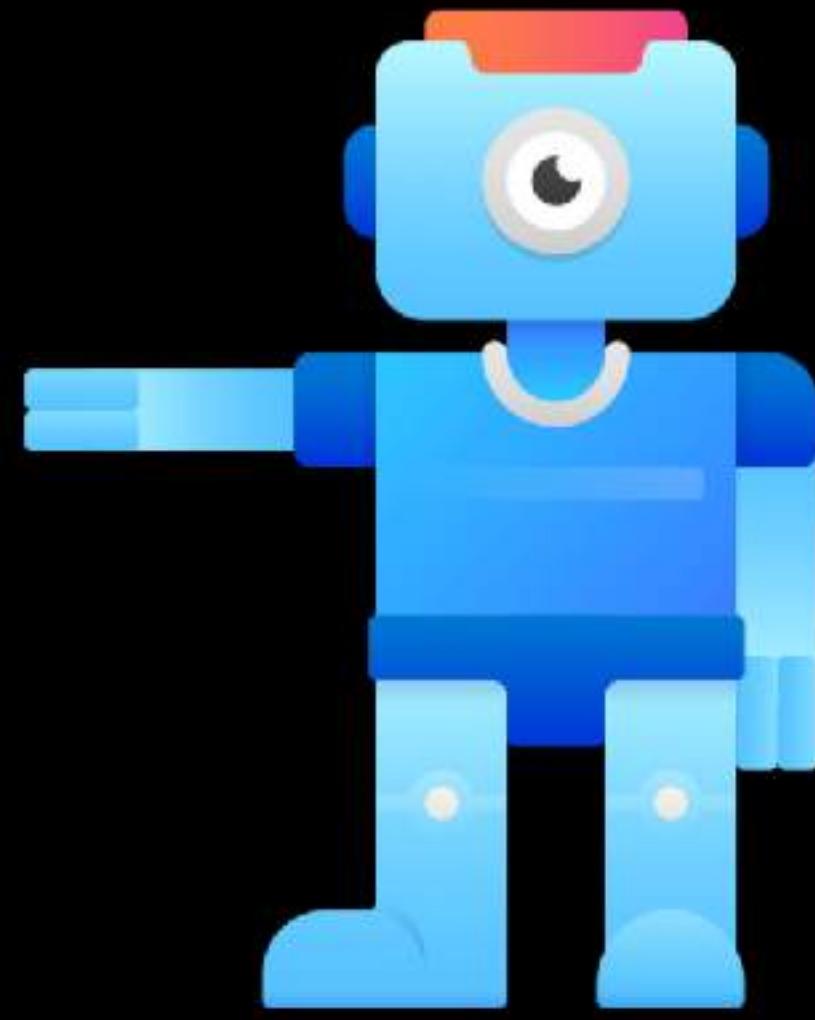
- Developed by Google in 2017 (original DeepLabV1 was released in 2014)
- DeepLabv3 is a semantic segmentation architecture that improves upon DeepLabv2 with several modifications.
- To handle the problem of segmenting objects at multiple scales, modules are designed which employ atrous (dilated) convolution in cascade or in parallel to capture multi-scale context by adopting multiple atrous/dilation rates.



# Mask R-CNNs

- Published in 2017, it is a general framework for object instance segmentation.
- Mask R-CNNs efficiently detect objects in an image while simultaneously generating a high-quality segmentation mask for each instance.
- Mask R-CNNs, extends Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition.
- Mask R-CNN is simple to train and adds only a small overhead to Faster R-CNN.





# MODERN COMPUTER VISION

BY RAJEEV RATAN

## DeepSORT

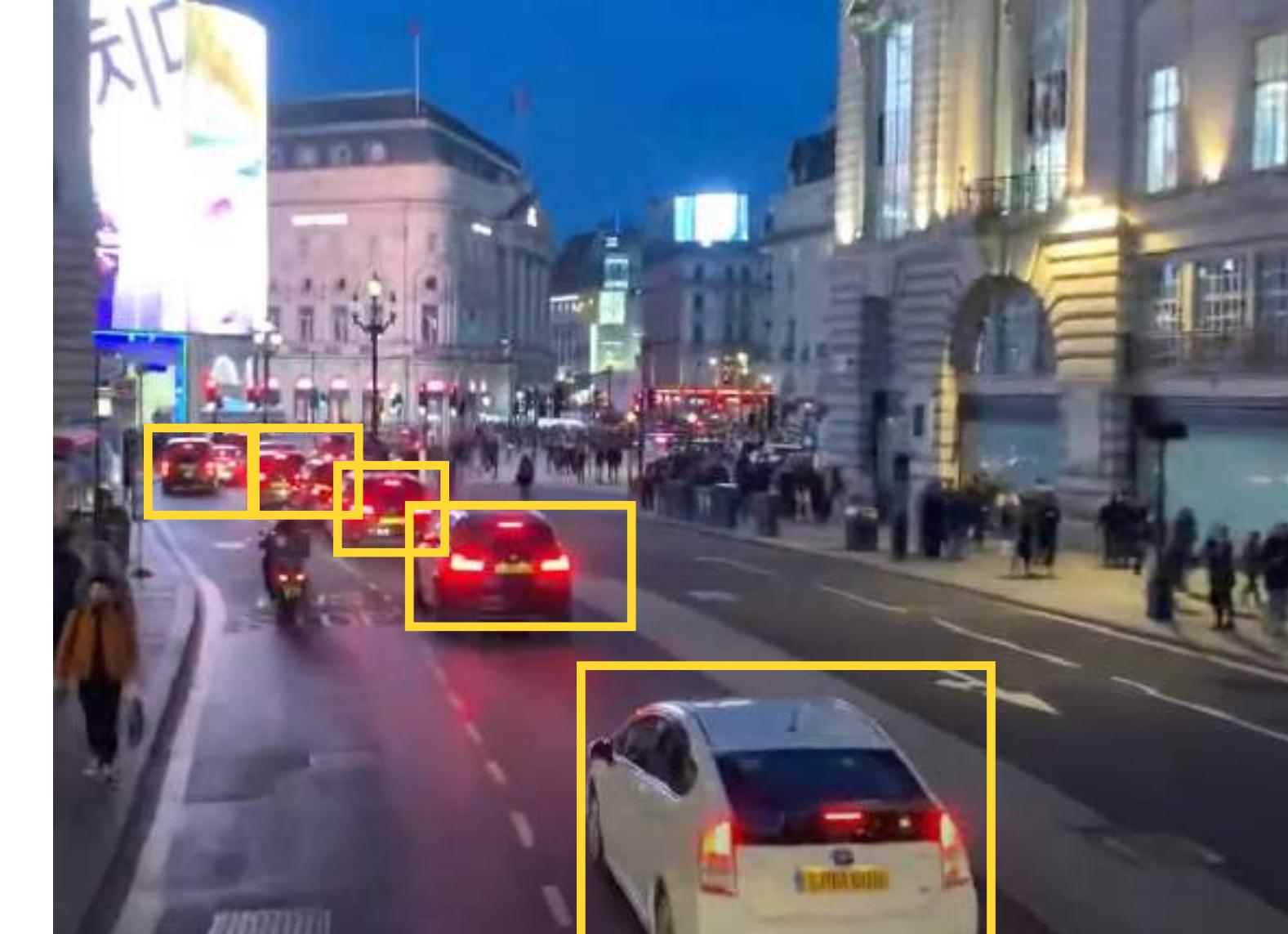
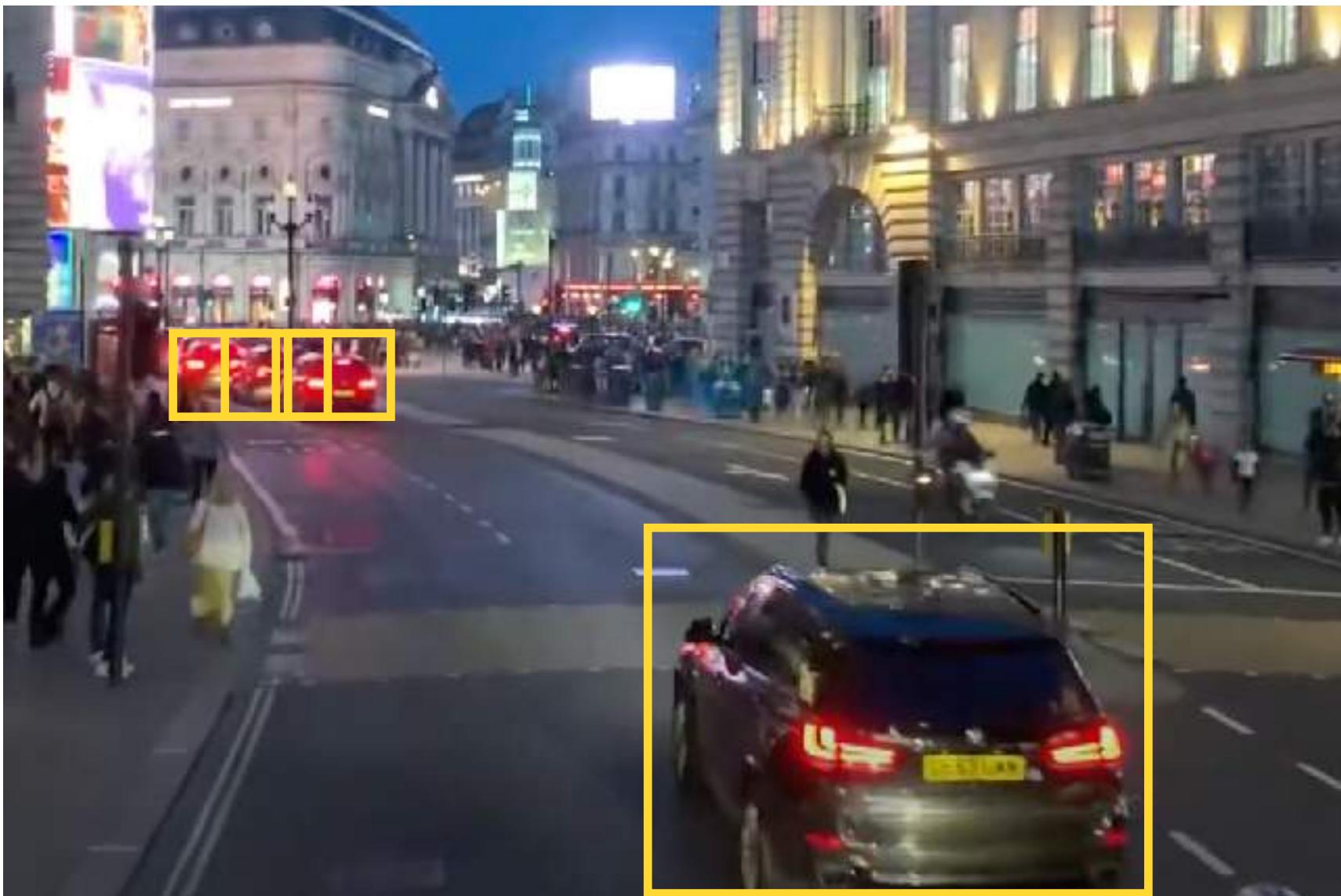
We take a look at DeepSORT which tracks detected objects using Deep Learning

# DeepSORT is a Deep Learning Based Tracker

- What's a Tracker?
- We just need to track objects right?
- Is that what Mean-shift and Optical Flow did? (refer back to earlier OpenCV sections)
- Tracking can be critical for certain applications.

# Why is Tracking Important?

- Look at two scenes, they're taken roughly 4 seconds apart.
- How do we keep track of individual cars?
- Detections change rapidly in fast moving scenes causing detection box jitter and instability.

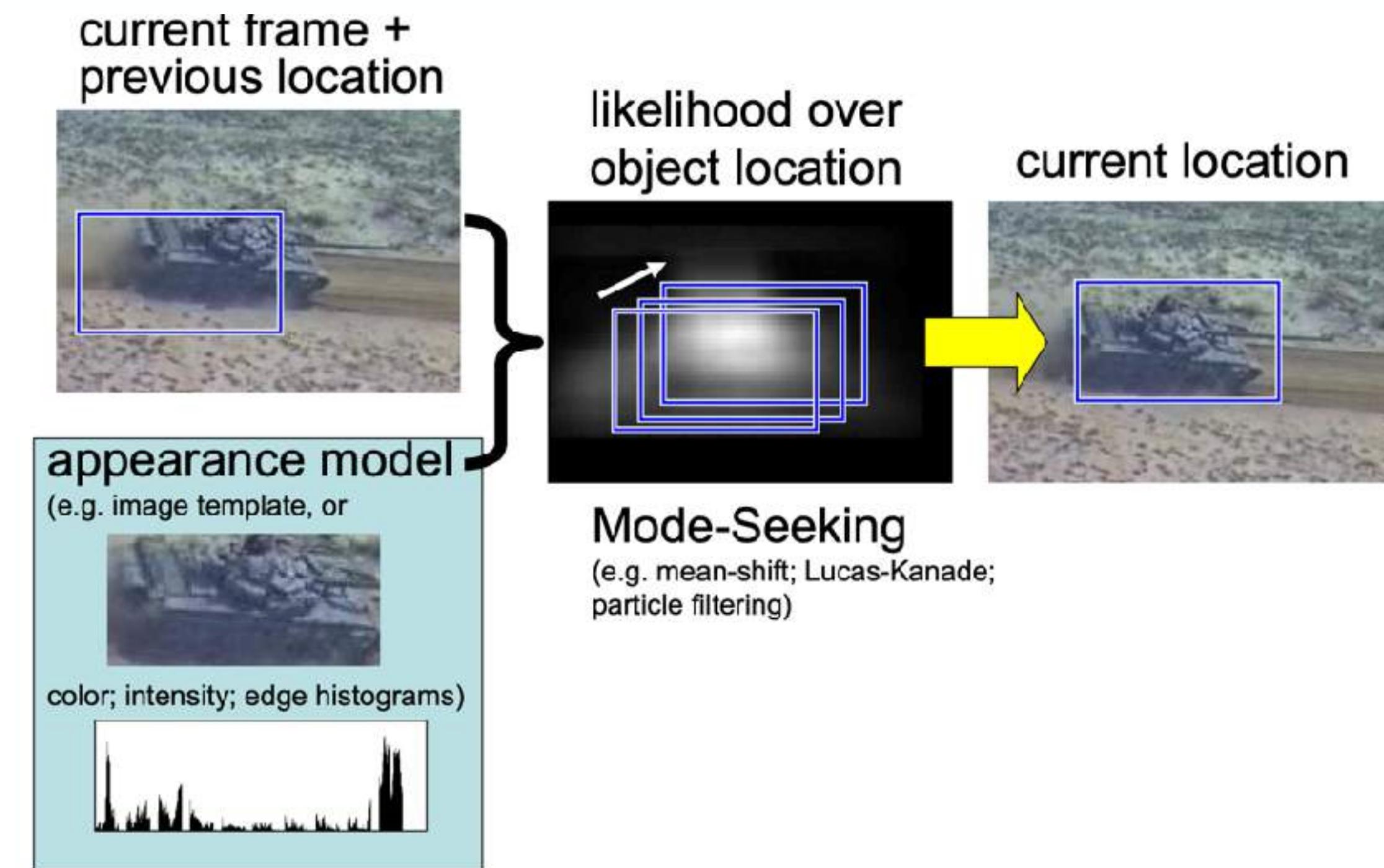


# Trackers

- You may recall in earlier OpenCV lessons we used Optical Flow and Mean shift trackers.
- Those are good, however, they fail in a few scenarios.

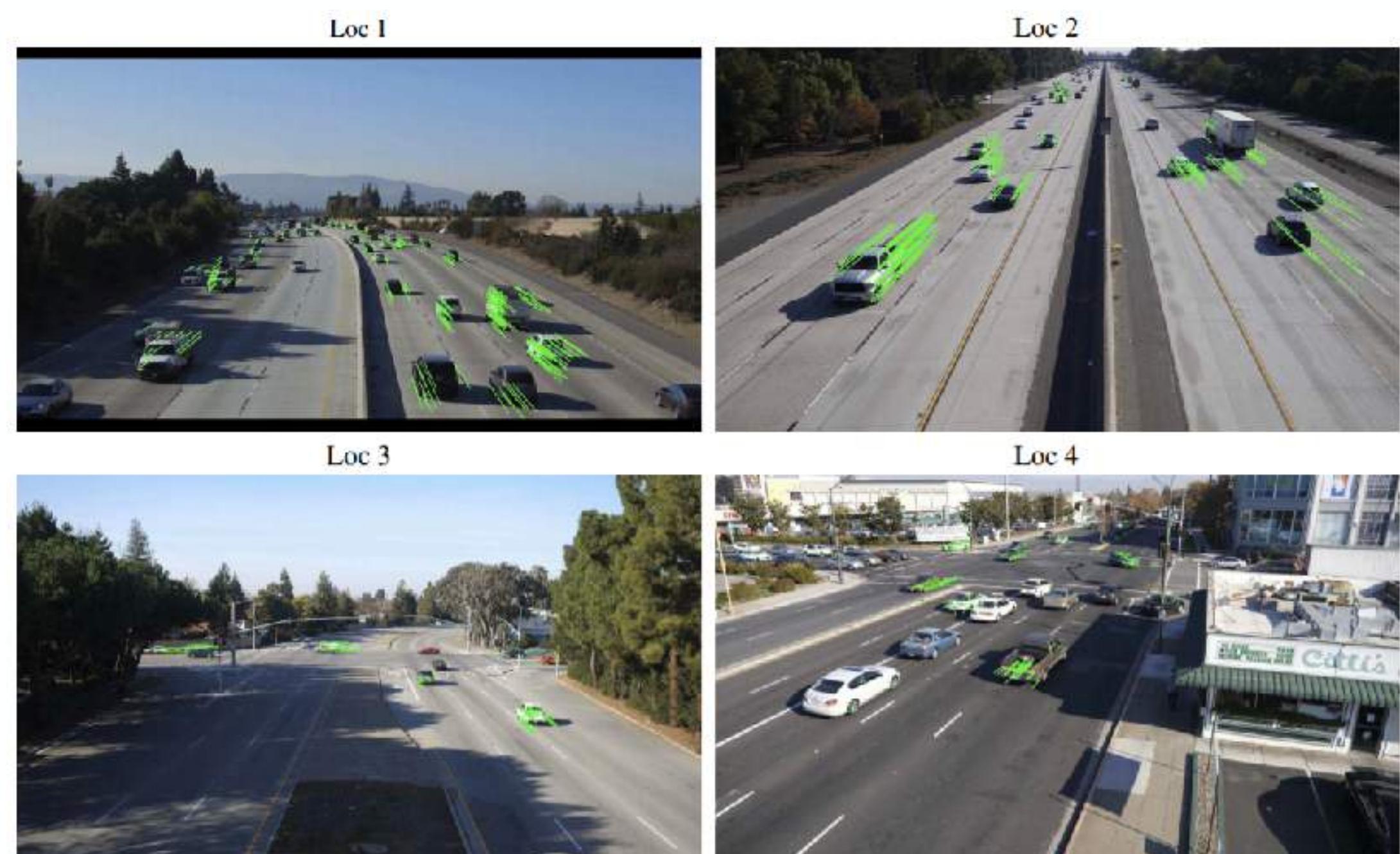
# Mean Shift Recap

- Mean shift an approach that tracks objects whose appearance is defined by histograms. Search within ‘near’ regions for object.



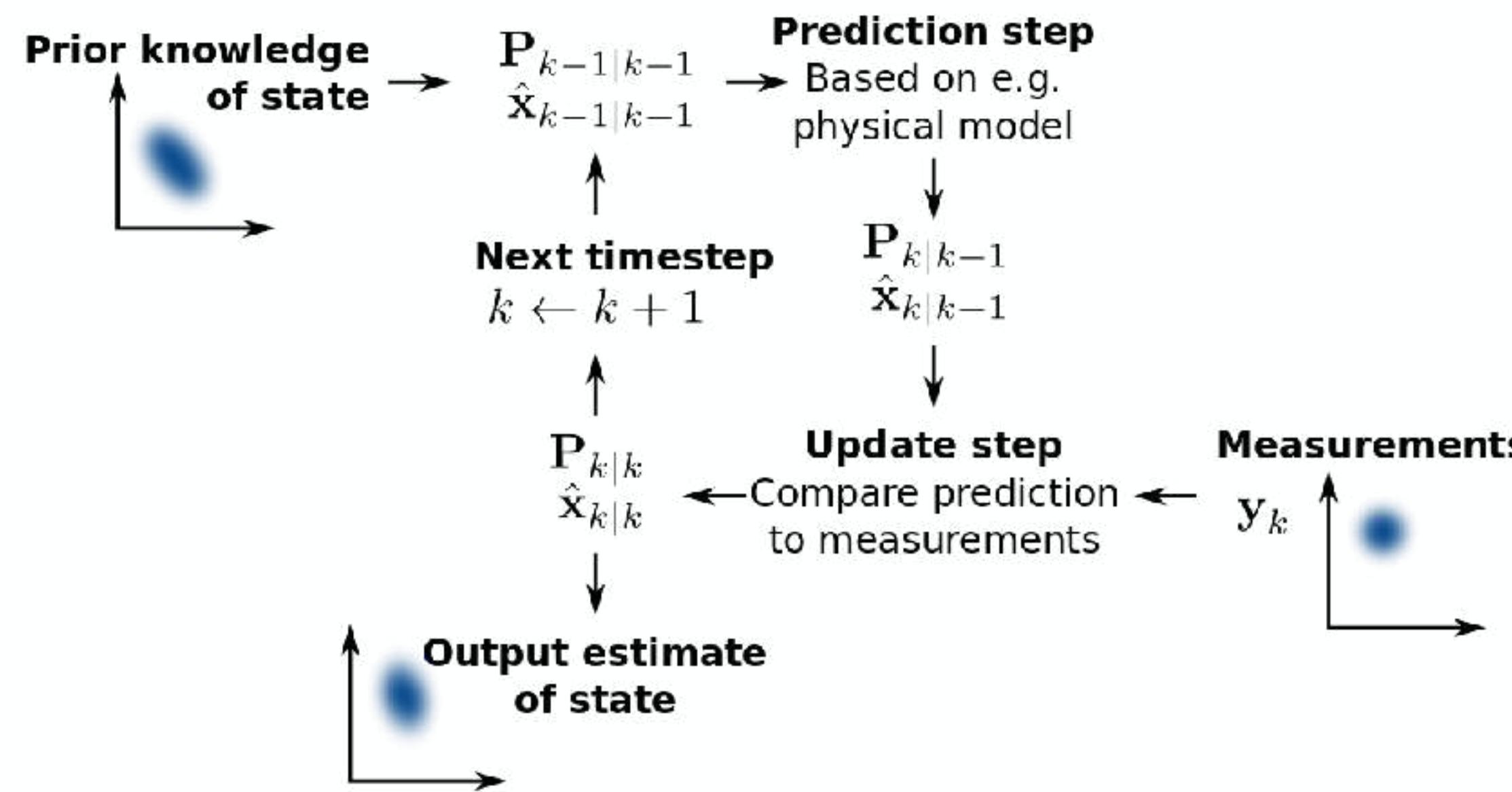
# Optical Flow

- Optical flow is the motion of objects between consecutive frames of sequence, caused by the relative movement between the object and camera.

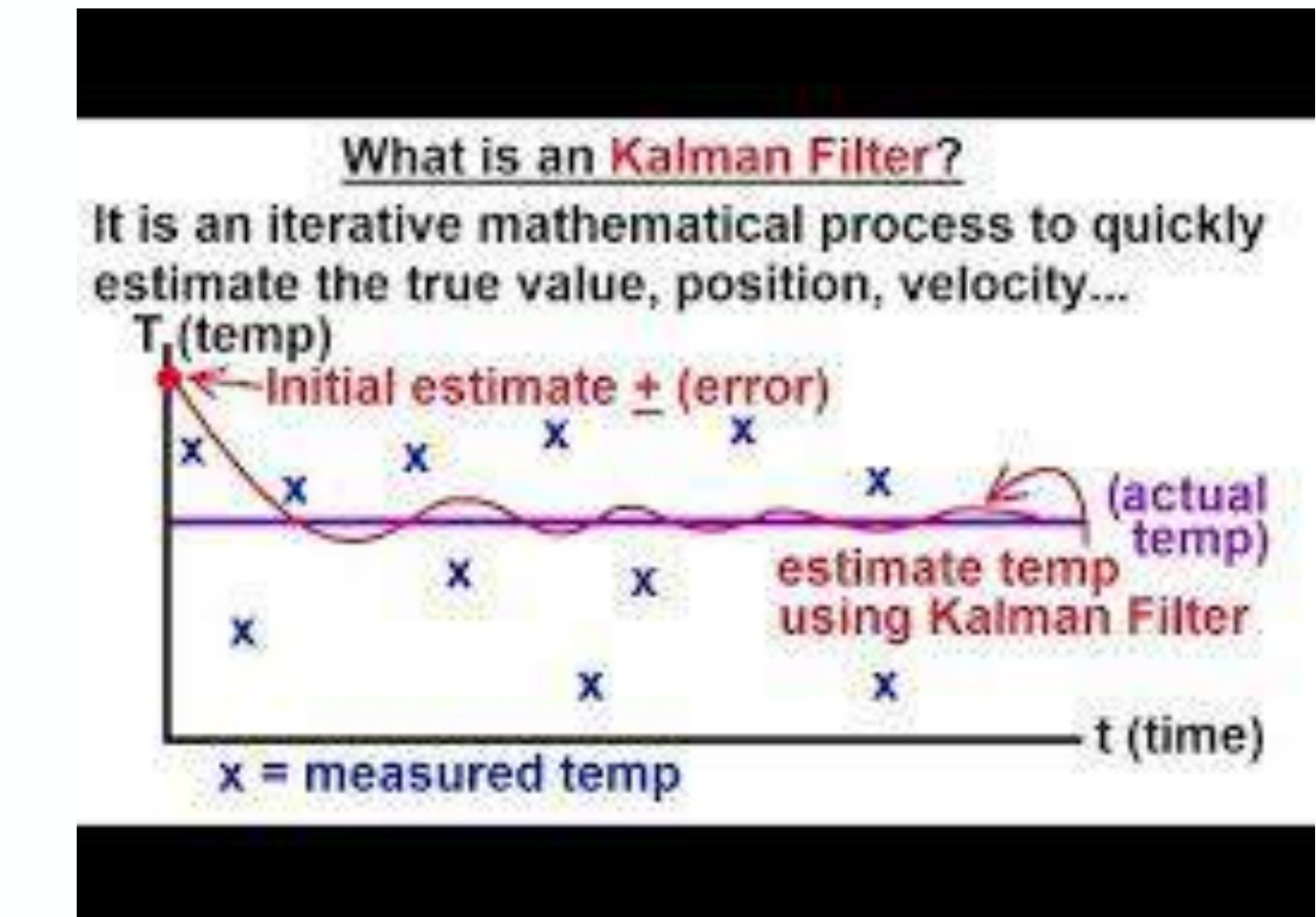


# Kalman Filtering

- Famous and ingenious, it is very commonly used in Engineering and Robotics problems where stable estimates are needed for temporal measurement data.
- We use the available detections and previous predictions to arrive at the best guess of the current state, along with an error probability.
- It's used in tracking by treating the noisy bounding box as the input.



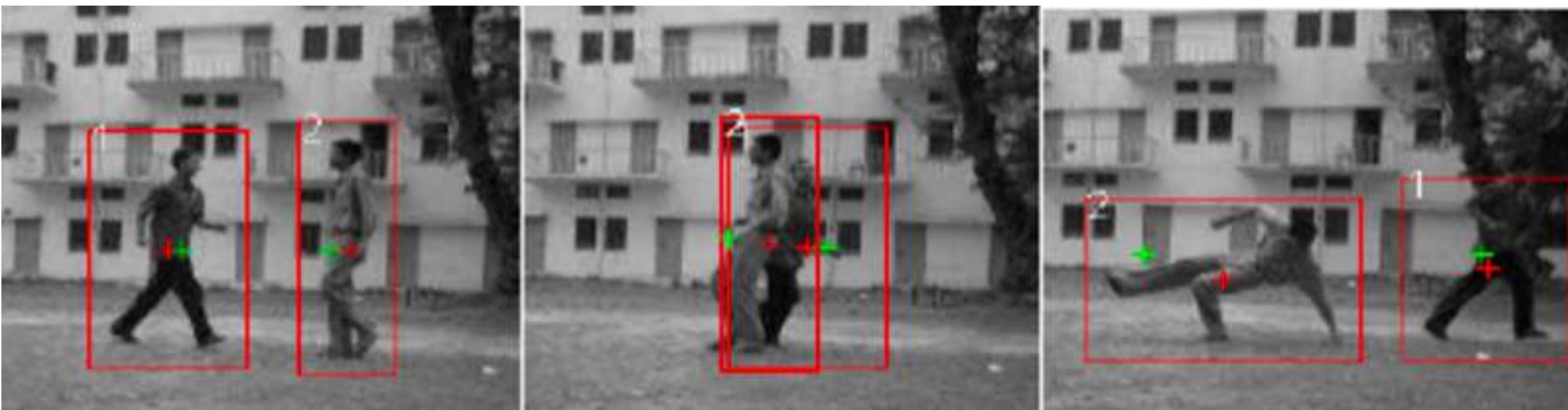
[https://en.wikipedia.org/wiki/Kalman\\_filter](https://en.wikipedia.org/wiki/Kalman_filter)



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

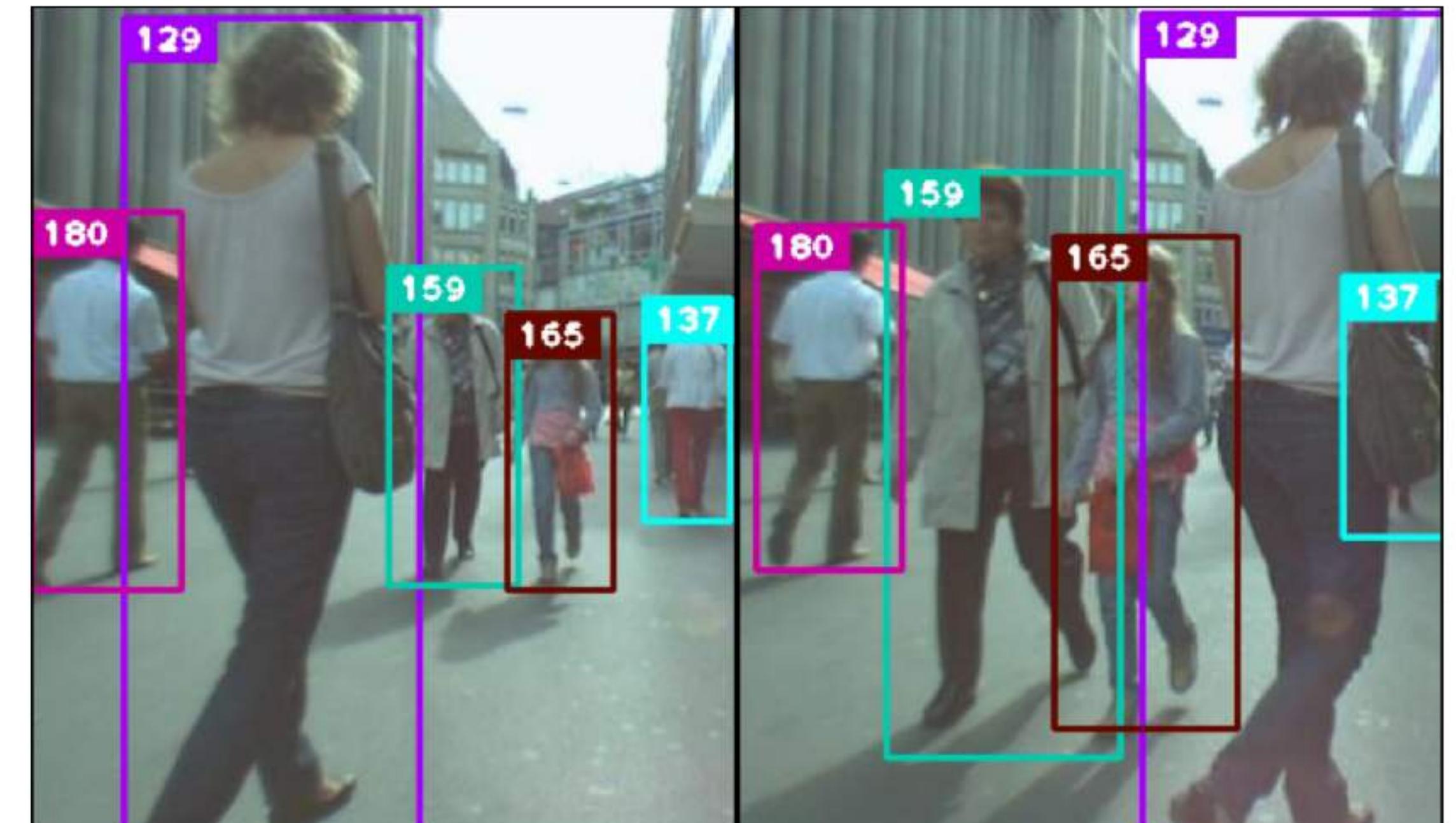
# Challenges

- Computationally heavy algorithms
- Susceptible to noise and fast camera movement/panning
- Occlusions can cause objects
- Do we have a better solution?



# DeepSORT - Simple Online Real-time Tracking

- DeepSORT is the most widely used Tracker in modern scenarios.
- It is simple framework that performs **Multi-Object Tracking**.
- It performs Kalman filtering in image space and frame-by-frame data association using the Hungarian method with an association metric that measures bounding box overlap.
- Take a look at the paper here - <https://arxiv.org/abs/1703.07402>



# DeepSORT - Kalman Filtering

- Deep Sort uses a **Kalman** tracking scenario that is defined on the eight-dimensional state space ( $u, v, a, h, u', v', a', h'$ ) that contains the bounding box center position ( $u, v$ ),  $a$  is the aspect ratio and  $h$ , the height of the image.
- The other variables have only absolute position and velocity factors since we are assuming a simple linear velocity model.
- The Kalman filter is used to factor in the **noise** in the detection as well as the **prior** state.
- For each detection, we create a **Track**, with its **associated state information**.
- There is a parameter to track and delete tracks for objects no longer in the scene.
- Duplicate tracks are deleted as there is a minimum number of detections threshold for the first few frames.
- So now, when we have a new bounding boxes tracked from the Kalman filter, this leads to **the problem where we now have to associate new detections with new predictions**.

# DeepSORT - The Assignment Problem

- Given that they are processed independently it results in an **assignment problem** because we have no idea how to associate **track\_i** with incoming **detection\_k**.
- This was solved by:
  - A **distance metric** to quantify the association.
  - An **efficient algorithm** to associate the data.
- The DeepSORT authors decided to use the squared **Mahalanobis** distance (effective metric when dealing with distributions) to incorporate the uncertainties from the Kalman filter.
- Thresholding this distance can give us a very good idea of the actual associations (better than euclidian since we're measuring distance between two distributions).
- They used the standard **Hungarian algorithm**, which is a very effective and simple combinatorial optimisation algorithm that solves the **assignment problem**.

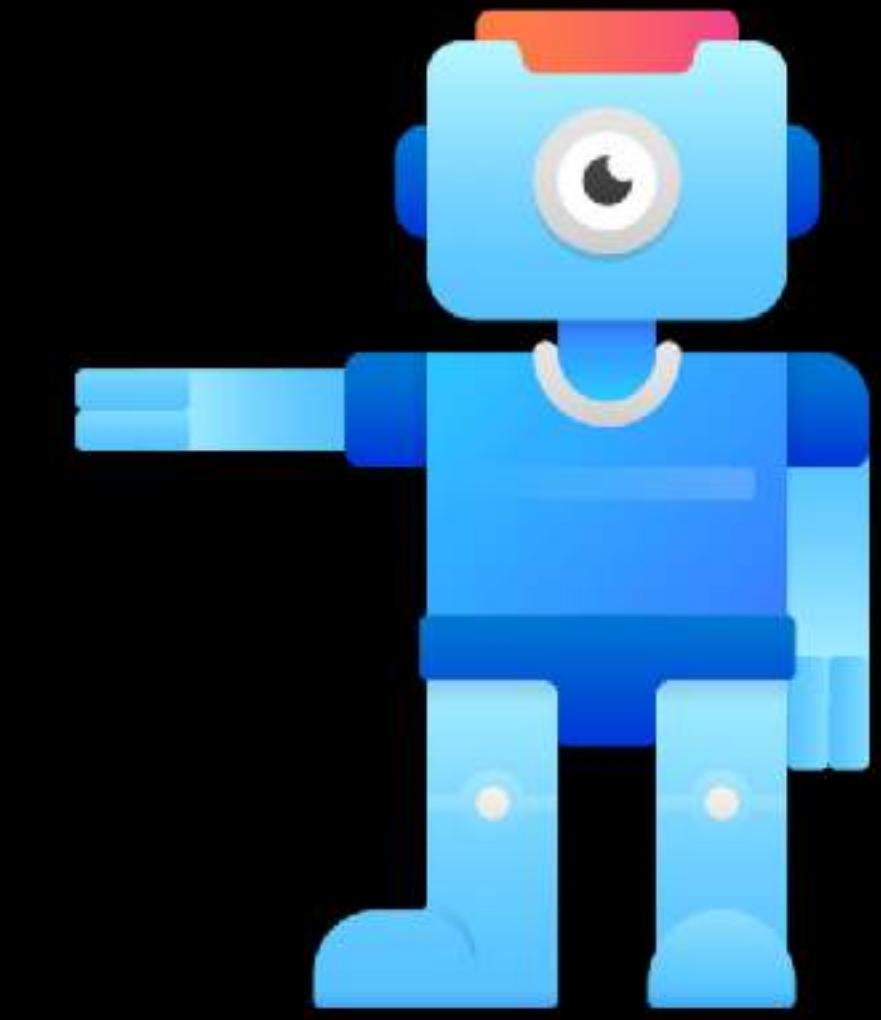
# DeepSORT - Handling Big Occlusions

- So far we have:
  - An object detector giving us the bounding boxes
  - A Kalman filter used to track detections so provide the missing tracks
  - Hungarian algorithm to solve the association problem
- The missing piece here is how do deal with big occlusions?
- DeepSORT introduced another distance metric that was based on the **visual appearance** of the object.

# DeepSORT - Feature Descriptor

- DeepSORT uses a pertained classifier (variation of a ResNET) as a feature extractor.
- It uses a 128 Dimension vector to describe the appearance of the object
- This allows us match objects that go missing from the scene back to their original track after re-appearing in the scene.

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and $\ell_2$ normalization		128



# MODERN COMPUTER VISION

BY RAJEEV RATAN

# Vision Transformers

Are ViTs replacing CNNs?

# What are Transformers?

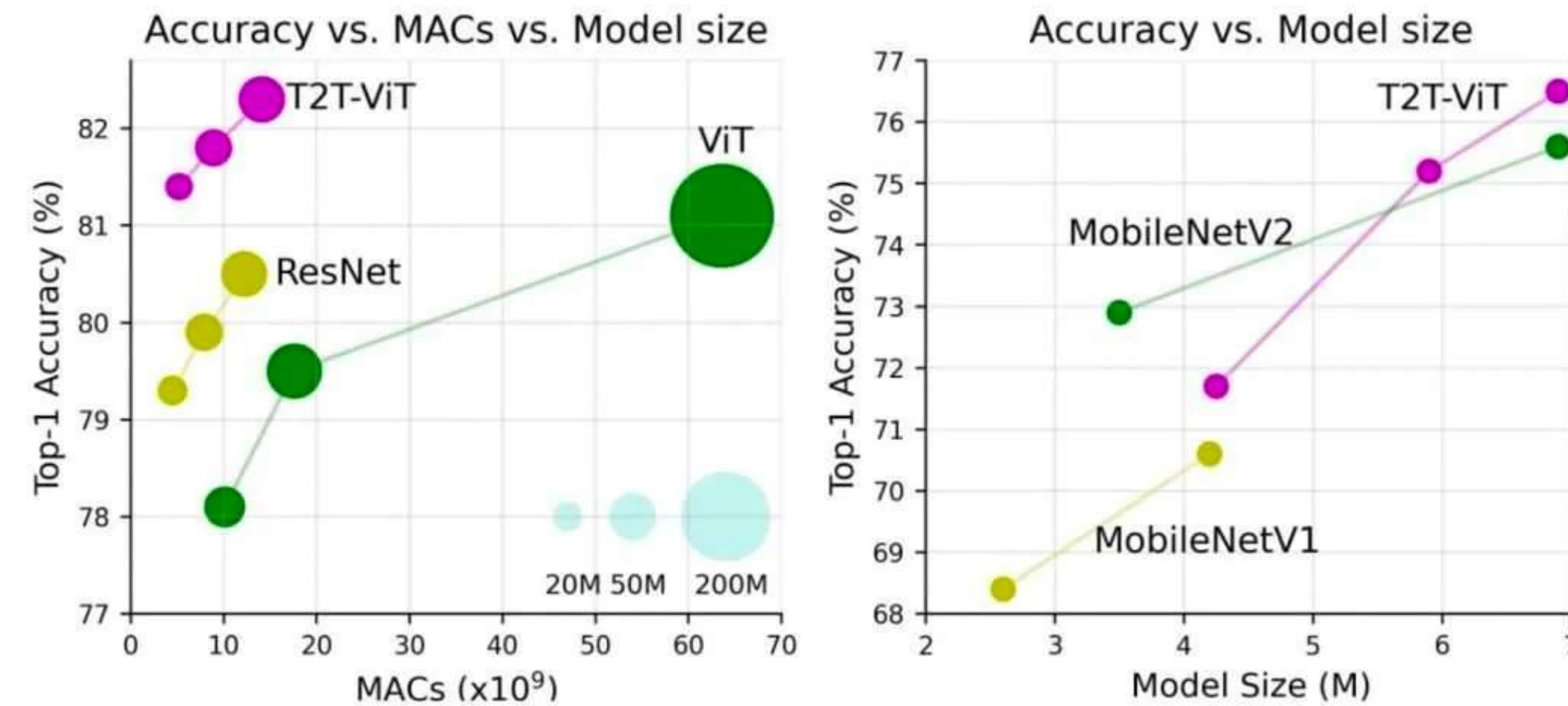


- Widely used in NLP after the famous paper ‘Attention is all you need’ was published in 2017.
- Transformers use something called **self-attention**
- It allows us to take a sequential input and finds the correlations between different features
- Gained a lot of traction in Computer Vision from 2020 onward



# ViTs are actually beating the best CNNs in many benchmarks

## However, CNN's still hold their own in many cases



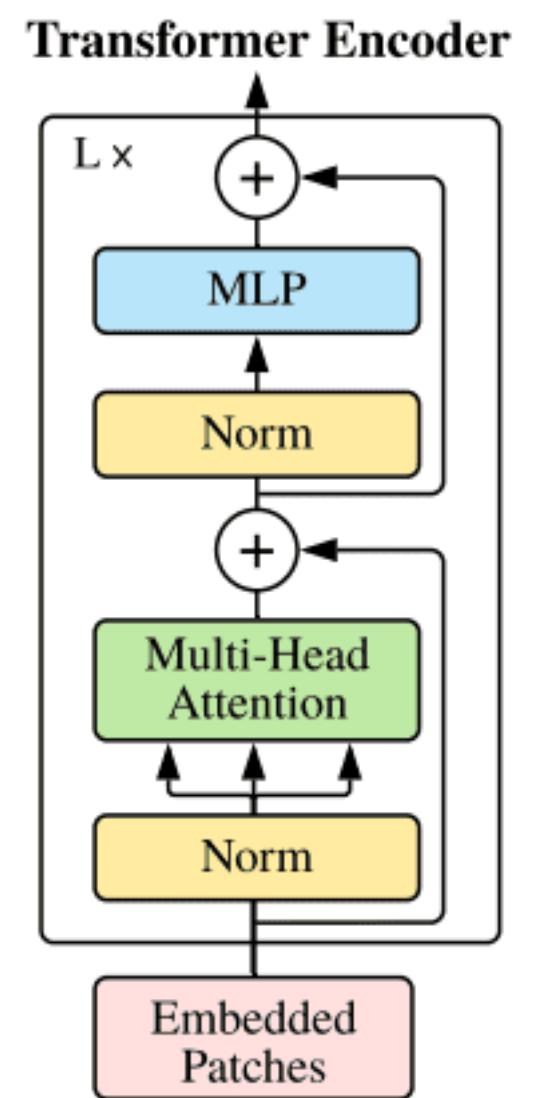
# How is this used for Images?

- Split an image into patches
- Flatten the patches
- Create a lower-dimensional linear embeddings from the flattened patches
- Add positional embeddings (trainable position embedding used typically)
- Feed the sequence as an input to a standard transformer encoder
- Pre-train the model with image labels (fully supervised on a huge dataset)
- Fine tune on the downstream dataset for image classification



# More about Vision Transformers

- Image patches are equivalent to sequence tokens (words in NLP)
- We can vary the number of blocks to get allowing for deeper networks
- ViTs require A LOT of data to be trained to beat SOTA CNNs
- However you can pre-train on a larger dataset and fine-tune on smaller ones (change the MLP head)



<https://arxiv.org/abs/2010.11929>