Object Detectors

Andrej Lúčny

Converting a classifier to a detector. Building blocks for training deep neural networks: Dropout, Batch normalization, residual connections. YOLO detector (YOLO v3). Regional convolutional networks (Faster RCNN)

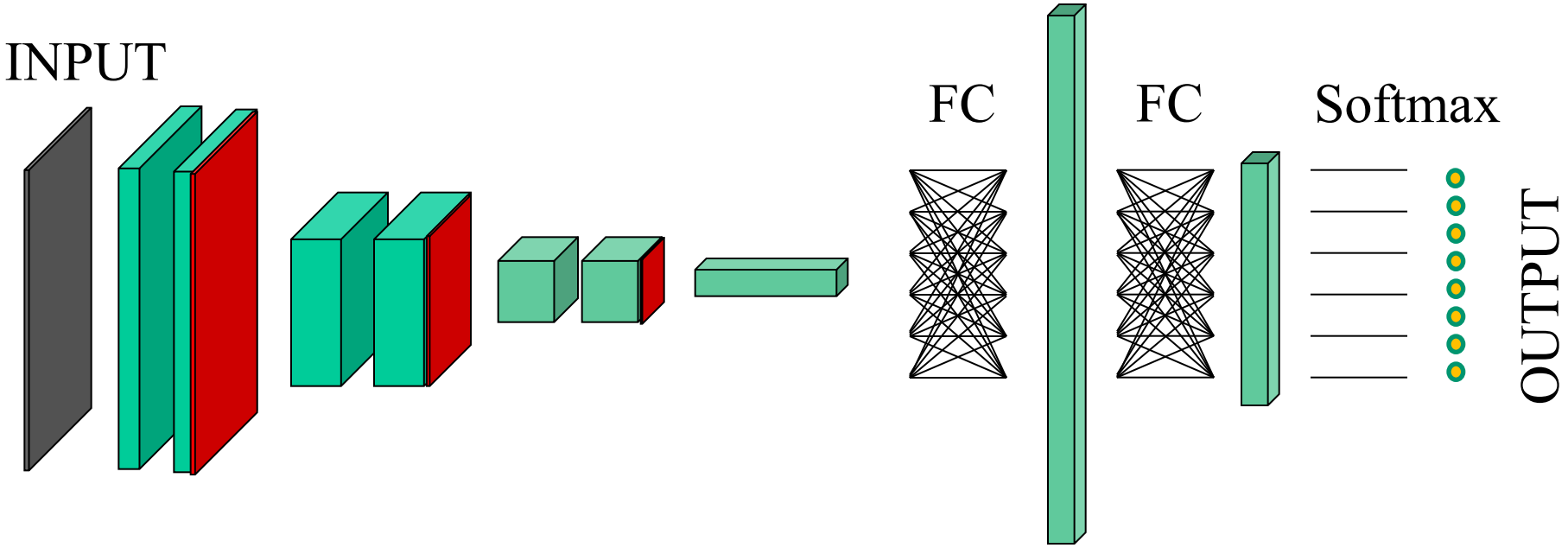
`https://github.com/andylucny/OAI`

Detection

Result of Detection

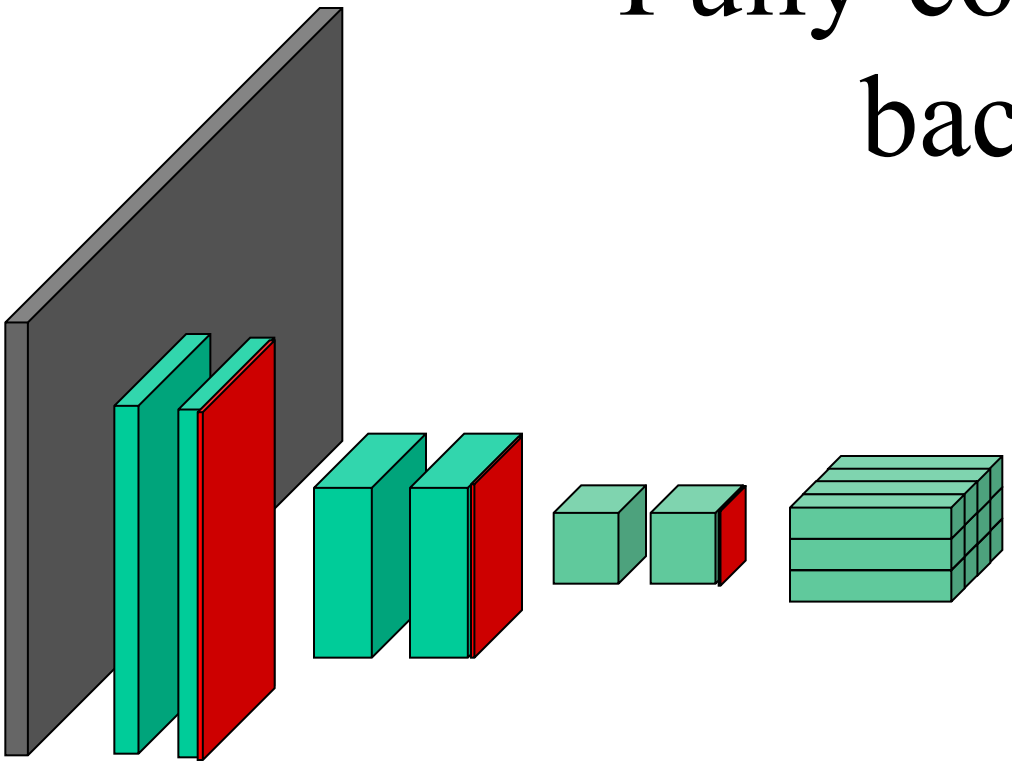
Dataset: train / test

Classifier



We already know to how make a classifier

Fully-convolutional backbone

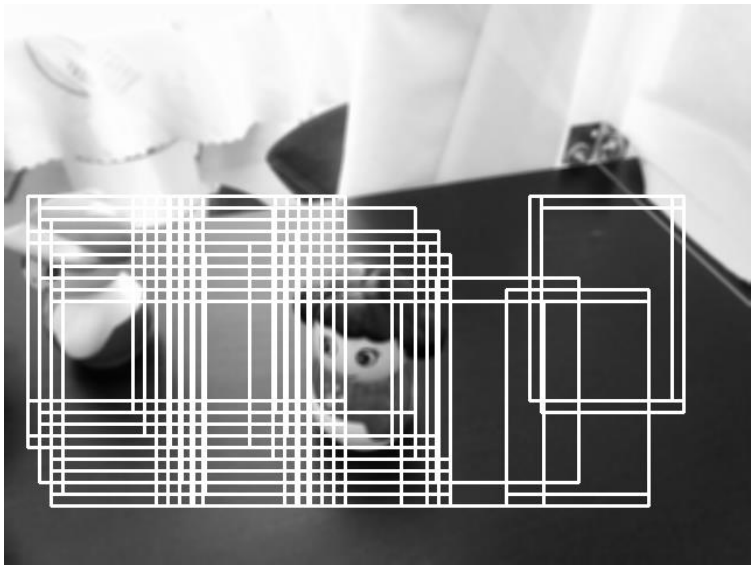


- We also know that the backbone provides us with the feature maps
- If we feed it a higher resolution image, the resolution of the feature map also increases.

- Sliding window algorithm

Any classifier can be brute-forced into a detector using a floating-window algorithm:

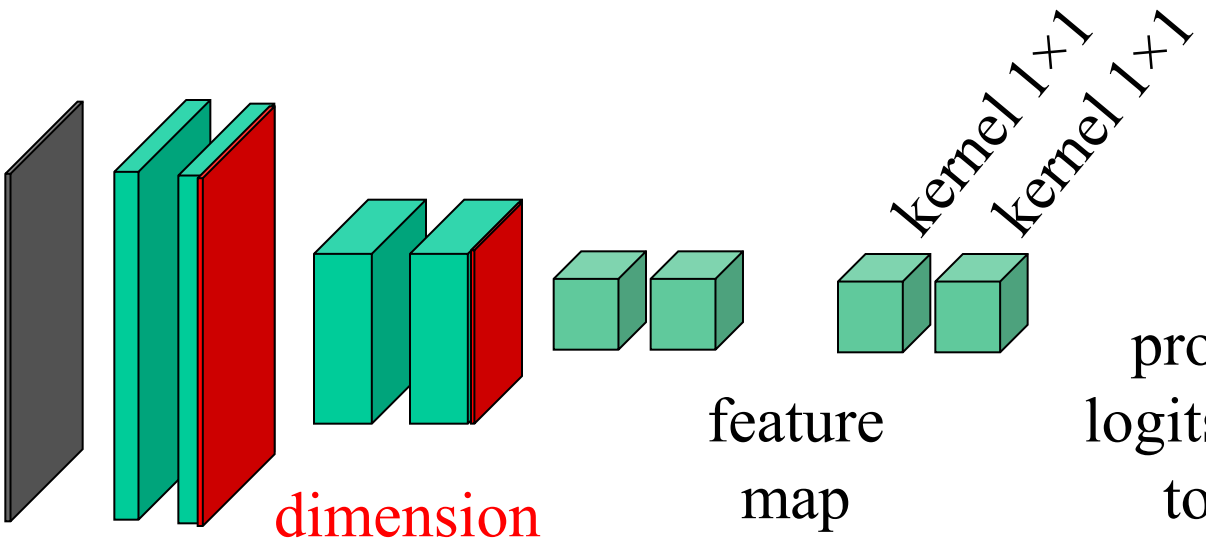
we test all possible occurrences of the object



Detector

- We will make it as a classifier that will look at different places in the image and classify whether they are the object we are looking for
- We will implement it as a perceptron that looks at all these places in parallel
- We can do this using two or more blocks of convolutional layers

(simple) Detector



dimension reduction

feature map

map of probabilities or logits of belonging to individual categories

blocks of convolutional layers

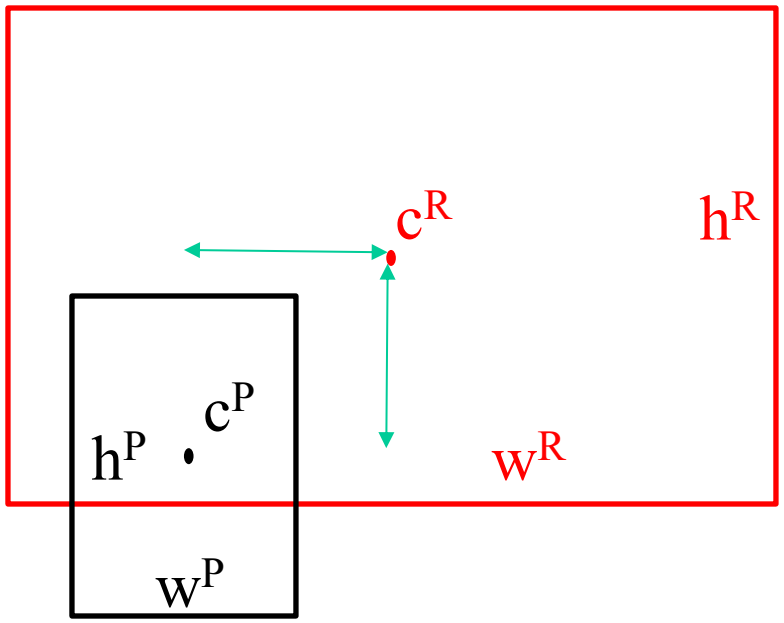
(e.g., object, non-object)

Disadvantages

- different sizes of objects
- lack of cooperation between viewpoints of nearby regions (if I see the number 5 here, I should also see 5 a little further away)
- we will see one object more than once, with its bounding rectangles overlapping
- phantom detections

Solution of different object size

- We will not only do classification but also regression of the object containing the rectangle



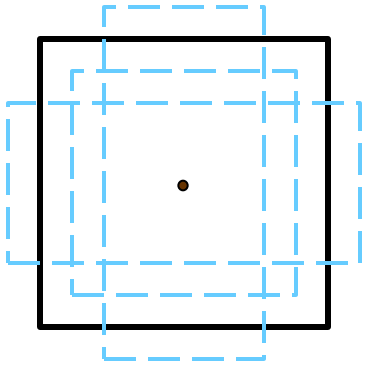
logits

$$\left(\underbrace{\frac{c_x^R - c_x^P}{w^P}}_{b_0}, \underbrace{\frac{c_y^R - c_y^P}{h^P}}_{b_1}, \ln \underbrace{\frac{w^P}{w^R}}_{b_2}, \ln \underbrace{\frac{h^P}{h^R}}_{b_3} \right)$$

$$(\underbrace{c_x^R - b_0\, w^P}_{c_x^P}, \underbrace{c_y^R - b_1\, h^P}_{c_y^P}, \underbrace{w^R\, e^{b_2}}_{w^P}, \underbrace{h^R\, e^{b_3}}_{h^P})$$

pixels

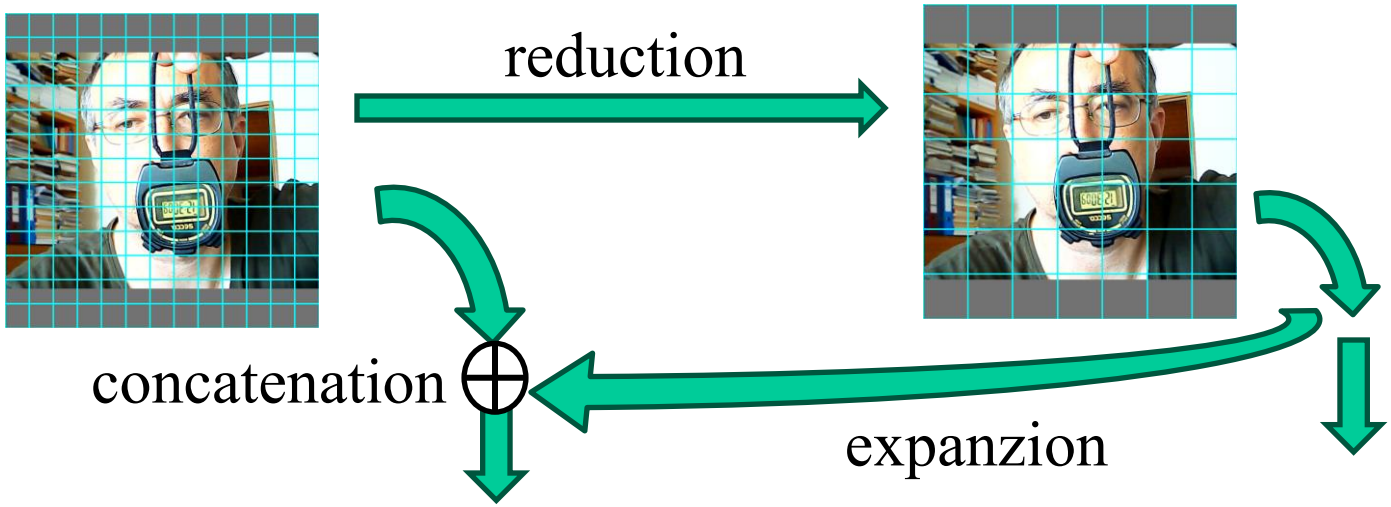
Anchors



- Each detection expresses x,y,w, and h relative to a particular anchor.
- YOLO v3 uses three anchors for each of the three object sizes (nine in total), each with a different aspect ratio
- Anchors in YOLO v3: [116x90, 156x198, 373x326] (output 52x52) [30x61, 62x45, 59x119] (output 26x26) [10x13, 16x30, 33x23] (output 13x13)

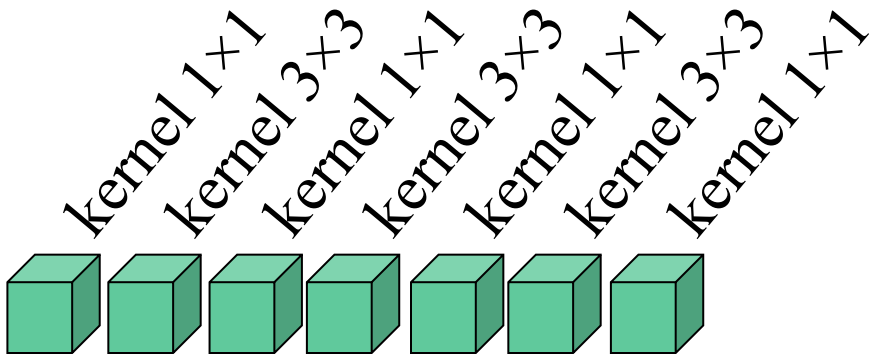
Solution of different object size

- We will use multiple feature maps with different resolutions We will enrich the preliminary detection results for more detailed feature maps



Solution of cooperation of neighbors

- We will alternate convolutional layers with 1×1 kernels with layers with 3×3 kernels



Solution of seeing one object more times

Non-Maximum Suppression

- In a loop, we take the detection with the highest confidence and discard all those that significantly overlap with it.

Solution of phantoms

- We regress objectness or confidence
- NMS only takes detections with confidence over a given threshold

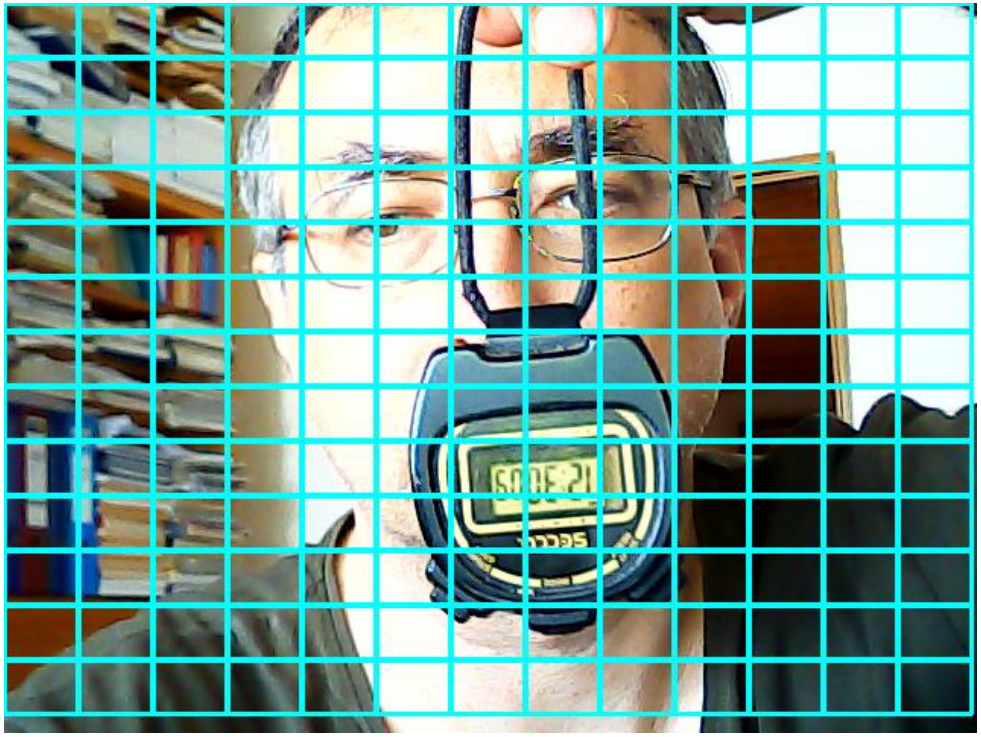
Single-class versus Multi-class

- If the object categories are mutually exclusive, we transform the logit values into probabilities using Softmax.

- However, if this is not the case (for example, we have both a wheel and a bicycle category), we use a sigmoid on the output.

YOLO Detector (You Look Only Once)

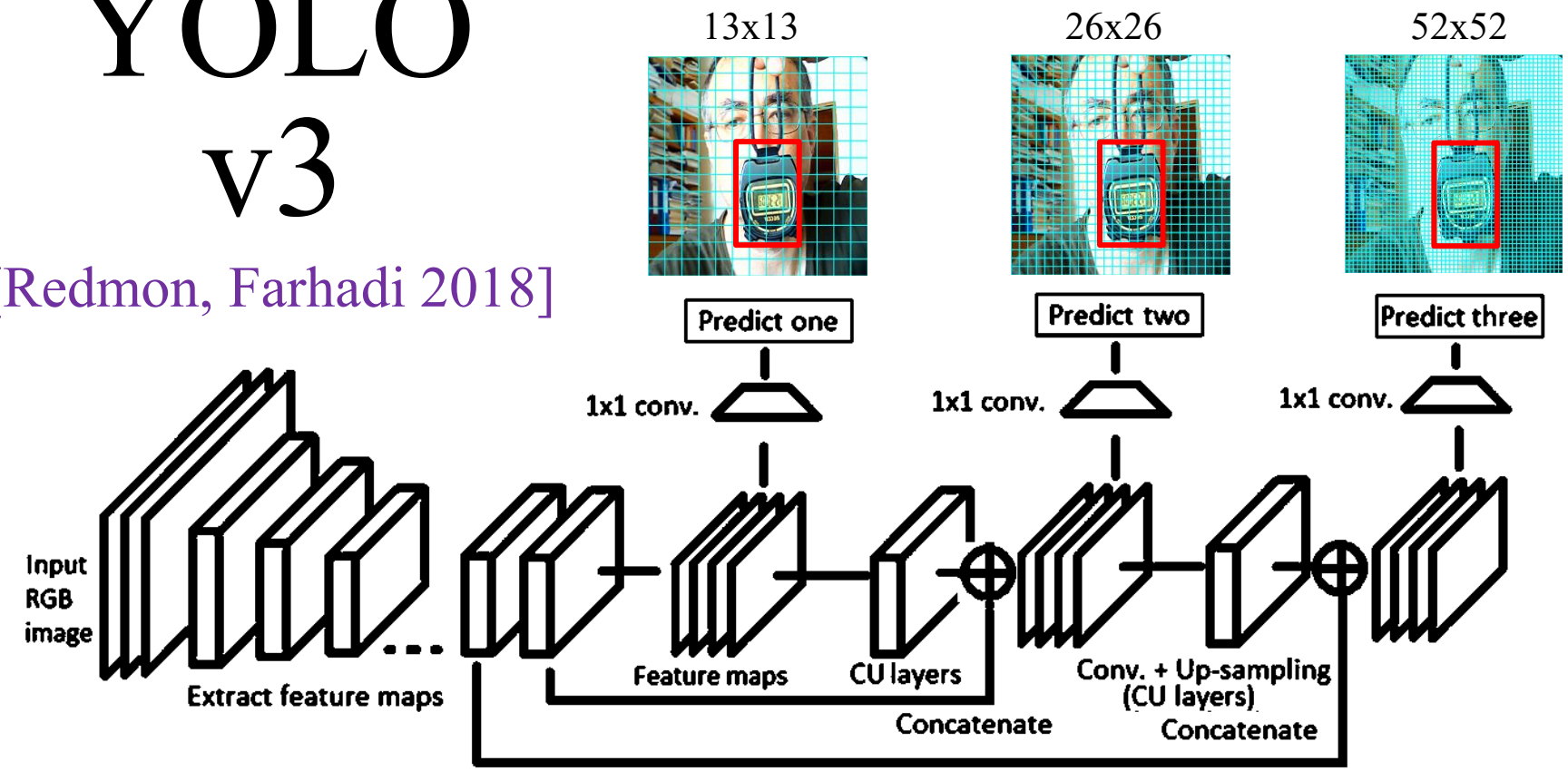
- We cover the image with non-overlapping regions.
- We perform classification and regression over each region and summarize the results.



- The classifier gives the probability that the region belongs to the detected object
- The Regressor gives the bounding box containing the object to which the region belongs

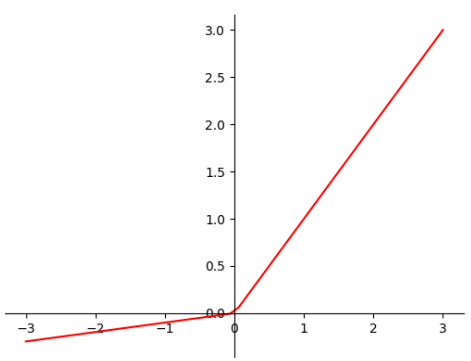
YOLO v3

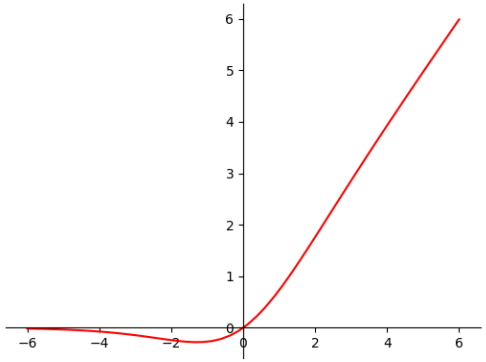
[Redmon, Farhadi 2018]



252 blocks, 5219 layers, max. depth around 200

Activation:
LeakyRelu or SiLU





YOLO v3 - output

- 13x13x255, 26x26x255, 52x52x255
- Each perceptron gives three detections (for three anchors), 3 × 85 = 255 values
- Every detection contains 85 values = 4+1+80 representing:
  4 … bounding box
  1 … confidence
  80 … probabilities for 80 categories

Transfer learning

YOLO v3 provides pretrained model for 80 categories of the COCO dataset:

person, bicycle, car, motorbike, aeroplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports, ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, sofa, pottedplant, bed, diningtable, toilet, tvmonitor, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush

- When training on our own data samples, we do not start with random parameters but with those of a pre-trained model.

Training deep networks

- Deep networks are difficult to train (the problem of vanishing gradient in backpropagation)
- Therefore, we insert building blocks into such networks that support training.

Dropout

$$drop(x) = \begin{cases} 0 & \text{if } random < p \\ x\frac{1}{1-p} & \text{otherwise} \end{cases}$$

if p = 0.2 20% set to zero

80% amplify

We insert Dropouts into network

Batch normalization

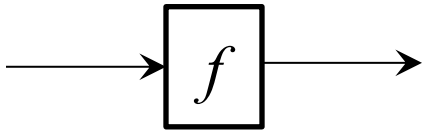
training $$bn(x) = \gamma \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

inference $$bn(x) = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

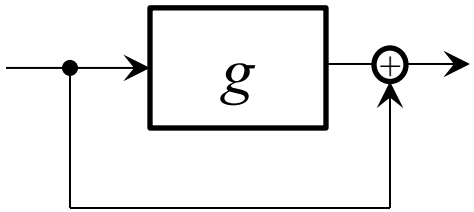
$$\mu = \sum_{i=1}^{t} (1-\alpha)^{t-i} \mu_{B_i} \qquad \sigma^2 = \sum_{i=1}^{t} (1-\alpha)^{t-i} \sigma_{B_i}^2$$

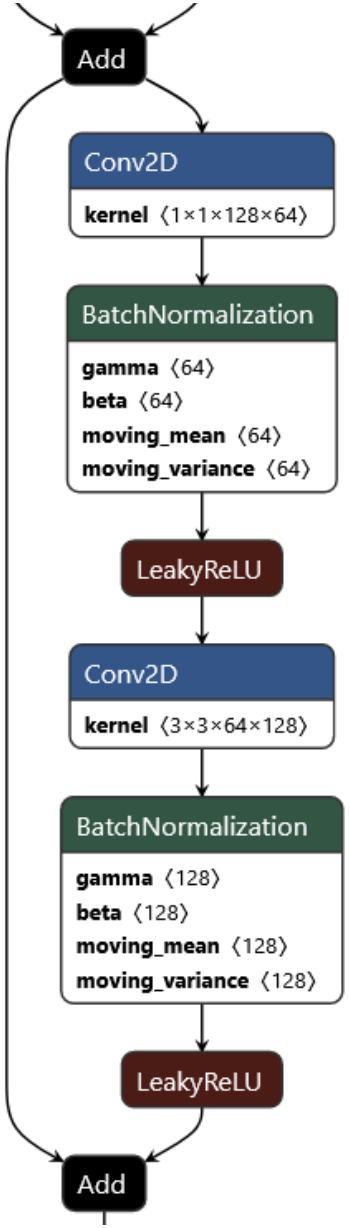
Residual connections

Instead of $f(x)$

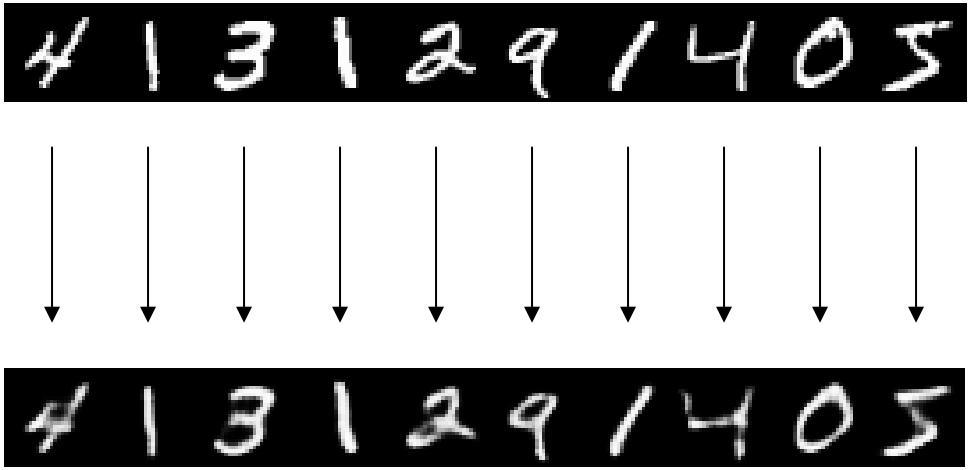


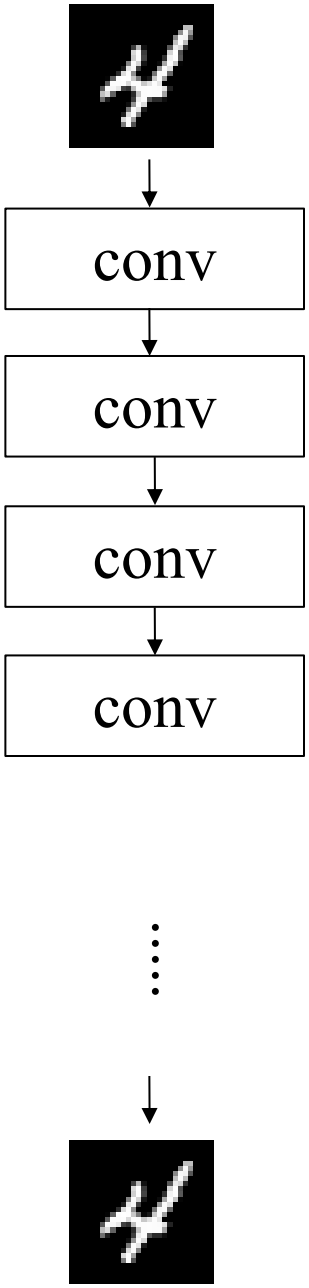
We seek residum $g := f(x) - x$

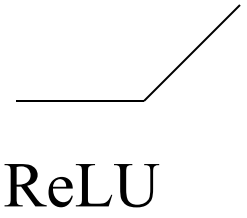


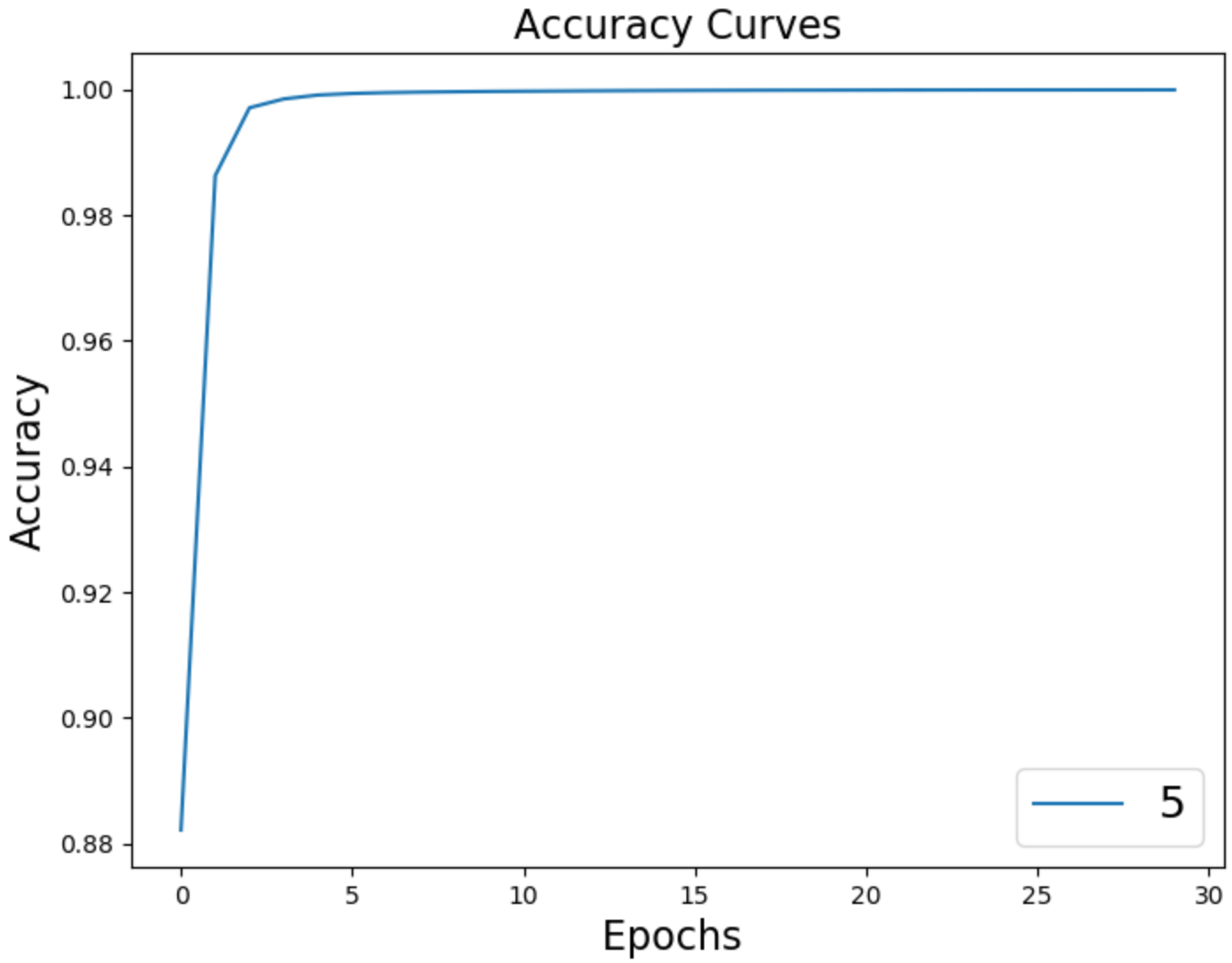


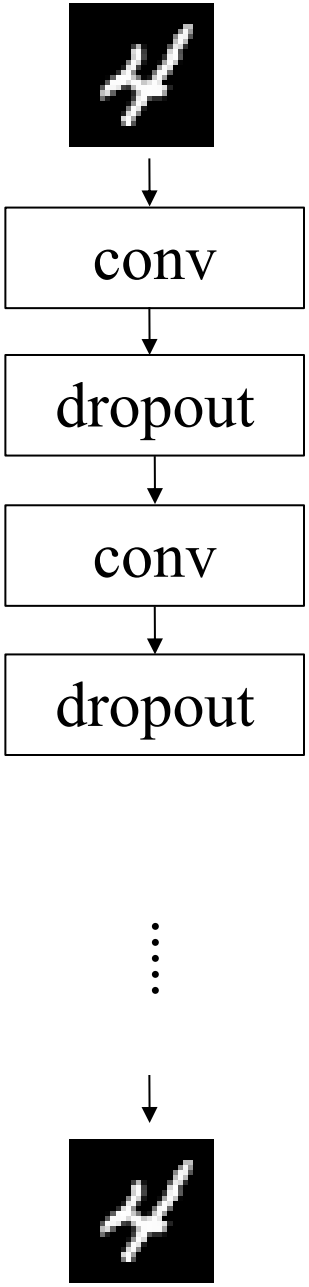
- The influence of these building blocks is clearly visible in identity training
- However, unlike an autoencoder, we do not reduce the dimension



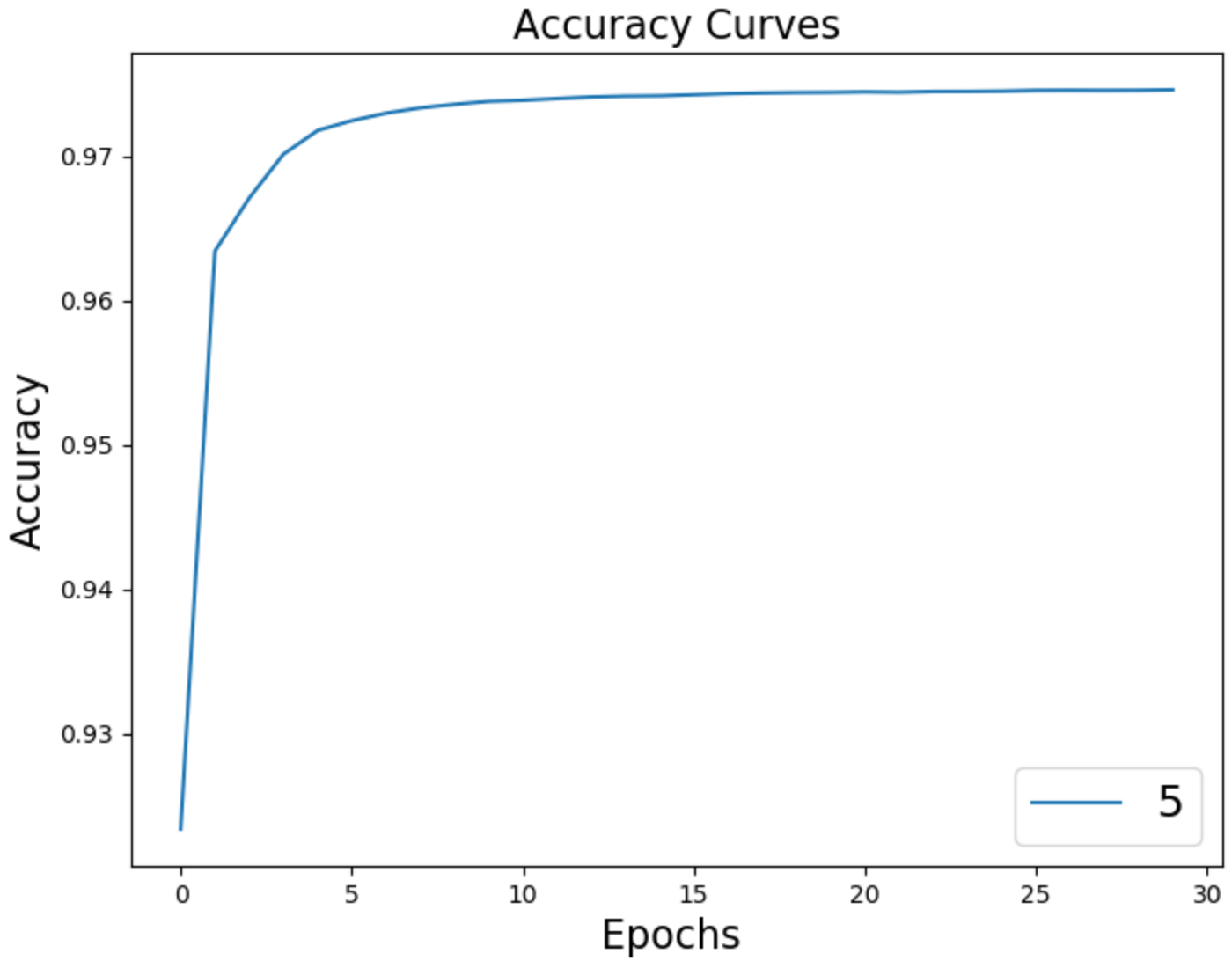

conv
conv
conv
conv

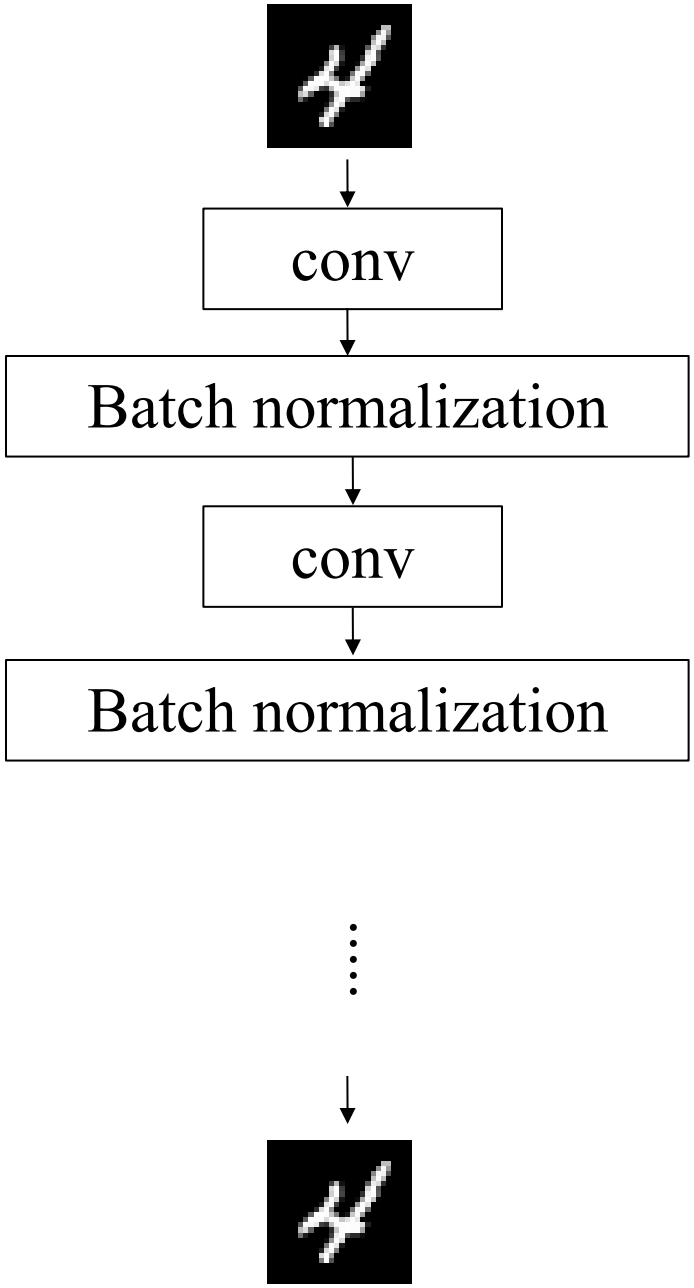

ReLU


Accuracy Curves
1.00
0.98
0.96
0.94
0.92
0.90
0.88
Accuracy
0
5
10
15
20
25
30
Epochs
5

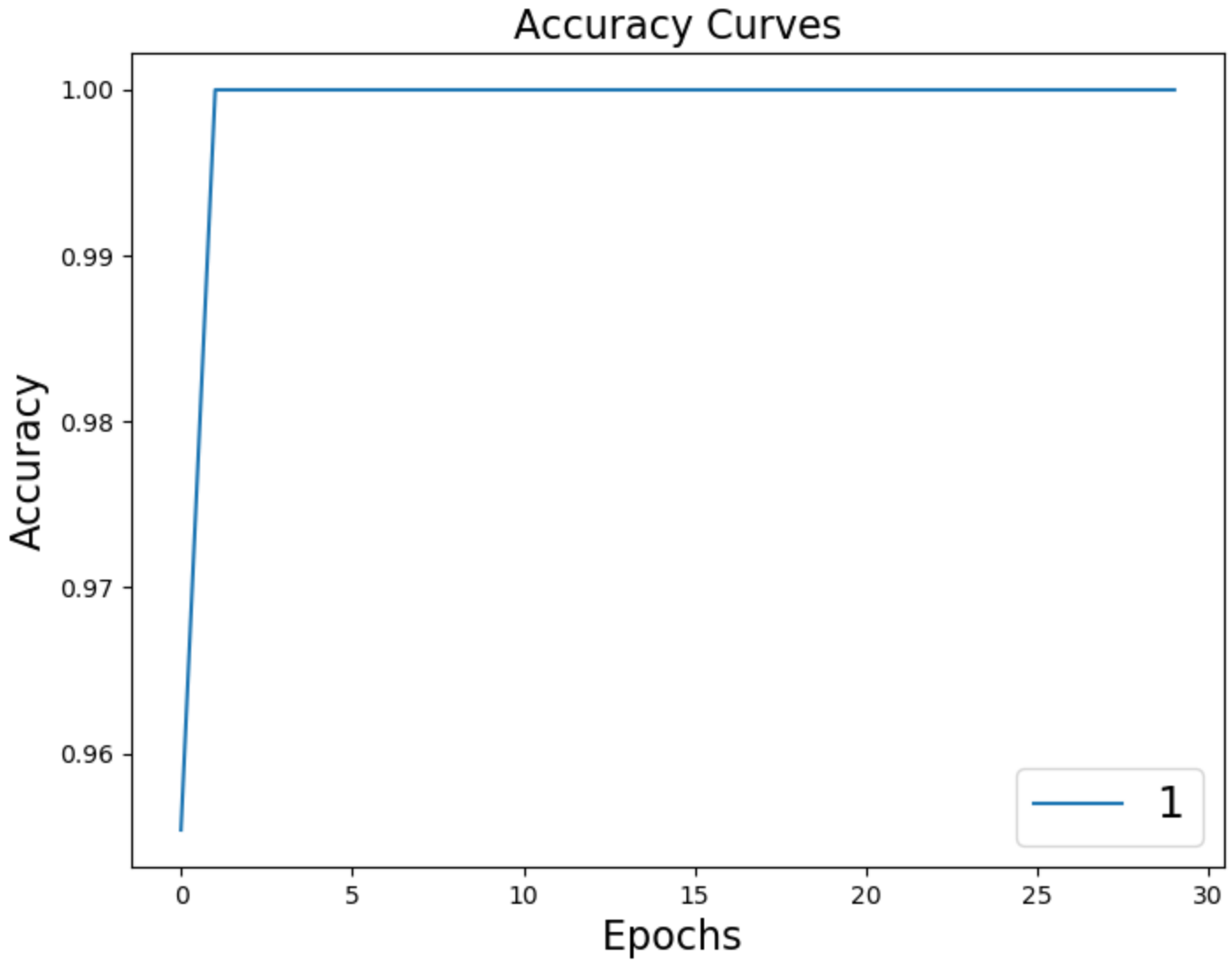

conv
dropout
conv
dropout

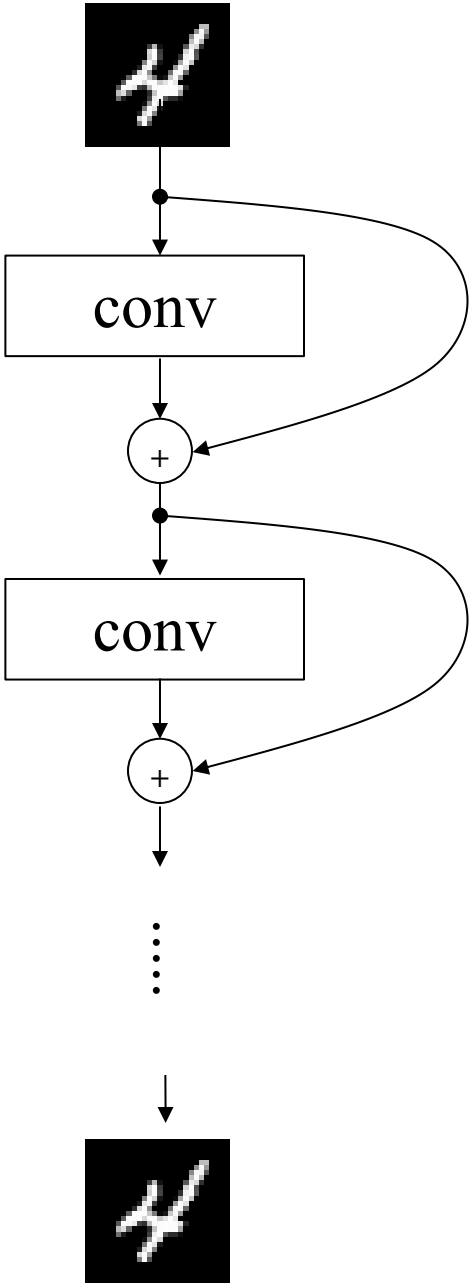
- Dropout


Accuracy Curves
Accuracy
0.97
0.96
0.95
0.94
0.93
0
5
10
15
20
25
30
Epochs
5

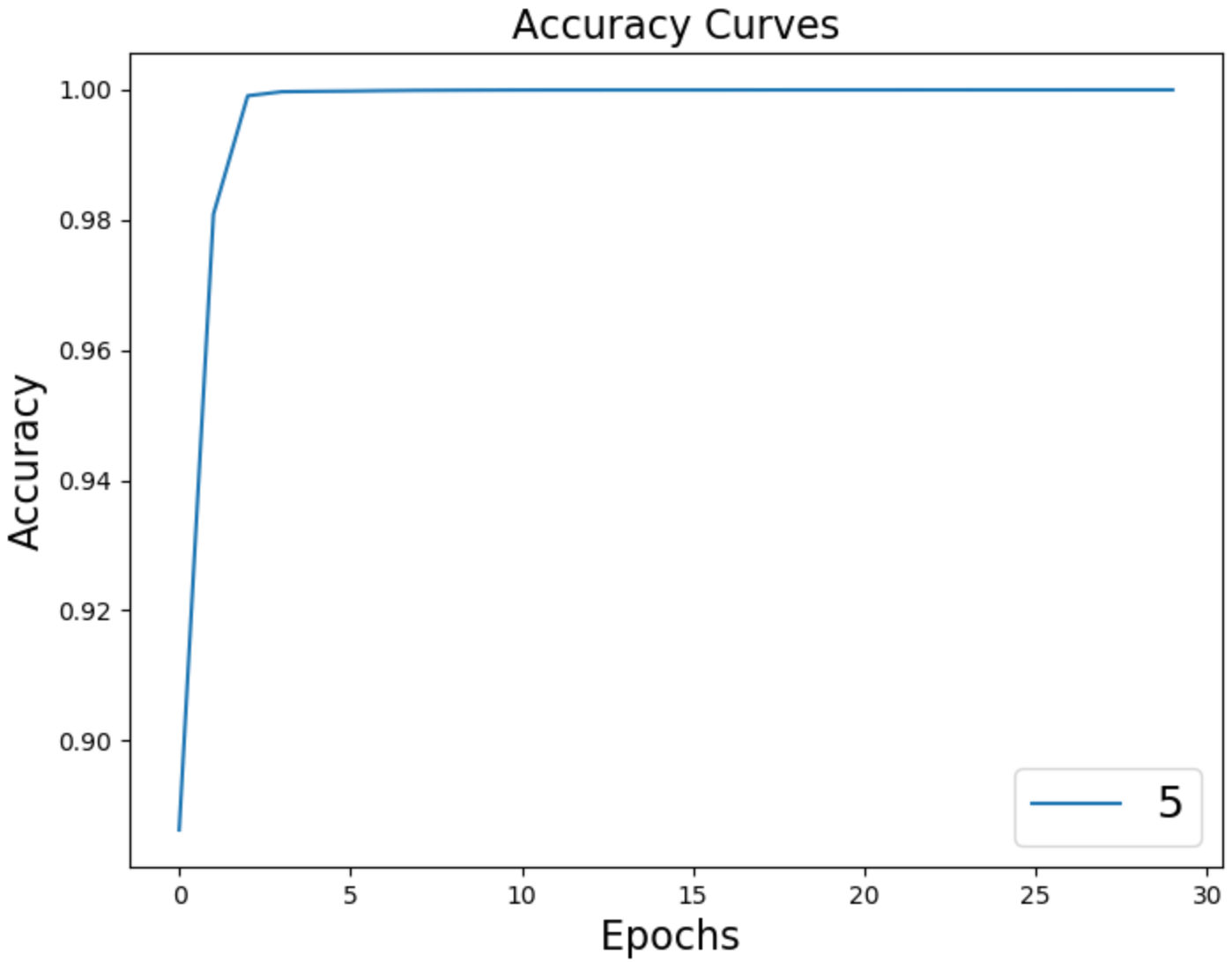

conv
Batch normalization
conv
Batch normalization

- Batch normalization




conv
+
conv
+

- Residuals


Accuracy Curves
1.00
0.98
0.96
0.94
0.92
0.90
Accuracy
0
5
10
15
20
25
30
Epochs
5

Augmentation

- Annotating a dataset is labor-intensive, so we look for ways to multiply the number of examples in the dataset.
- For example, in a dataset of faces, a vertical flip doubles the dataset, because the faces are not perfectly symmetrical.
- Operations that produce another sample from one sample (by enlarging, reducing, rotating, shifting, flipping, recoloring, ...) are called augmentations.

Training YOLO v3

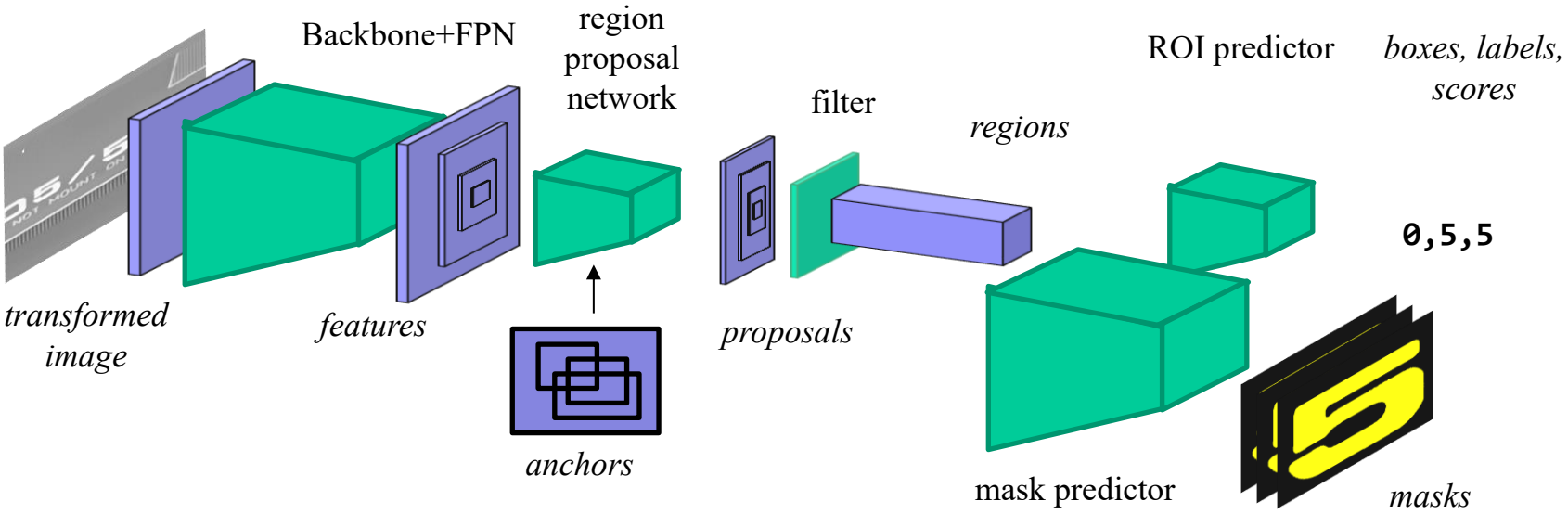
https://youtu.be/CPGR26Fhcro

https://youtu.be/ENYEUUbqjIE

Processing

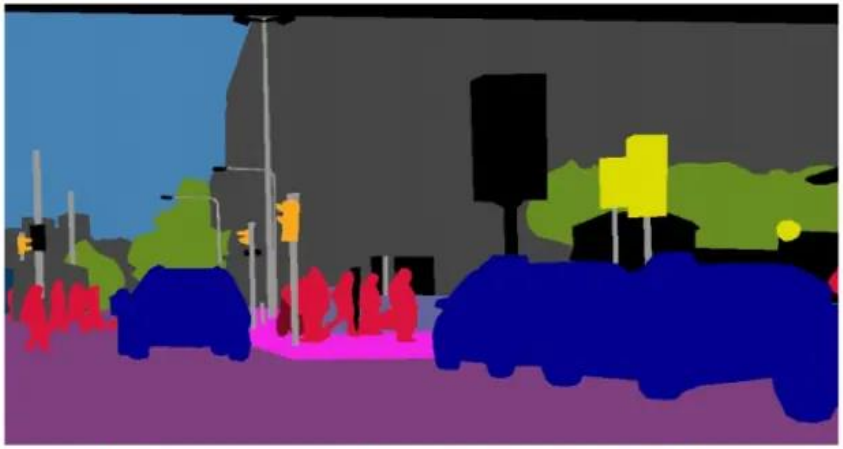
Region-based convolutional networks

- Faster RCNN, MaskRCNN

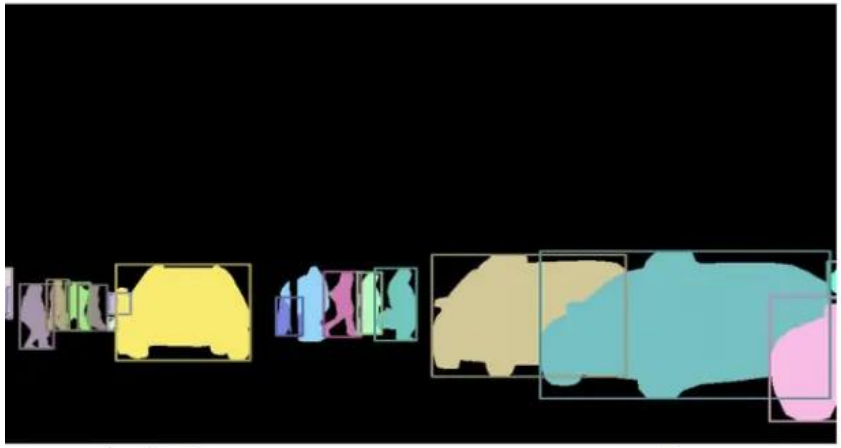


Segmentation

(a) image



(b) semantic segmentation



(c) instance segmentation

(d) panoptic segmentation