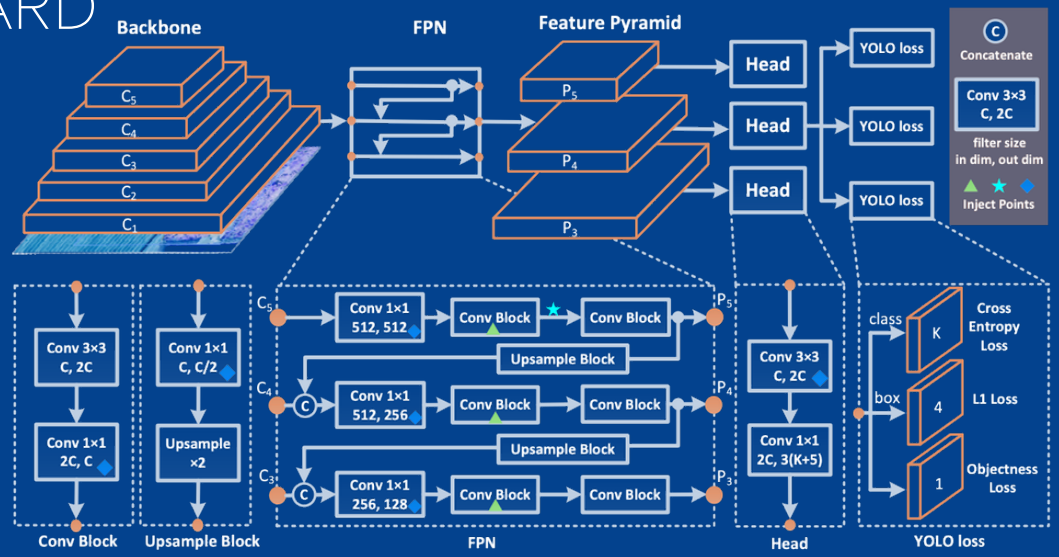
AARHUS UNIVERSITY

Lesson 11: Introduction to Advanced CNNs

CARSTEN EIE FRIGAARD

SPRING 2025



Agenda

- LLM Transformer recap,
- MNIST Search Quest Hi-Score Cake,
- Introduction to Advanced CNNs.

LLM: patterns/redundancy in text

Infinite monkey theorem

45 languages

Article Talk Read View source View history Tools

From Wikipedia, the free encyclopedia

The **infinite monkey theorem** states that a monkey hitting keys independently and at random on a typewriter keyboard for an infinite amount of time will almost surely type any given text, including the complete works of William Shakespeare.[a] More precisely, under the assumption of independence and randomness of each keystroke, the monkey would almost surely type every possible finite text an infinite number of times. The theorem can be generalized to state that any infinite sequence of

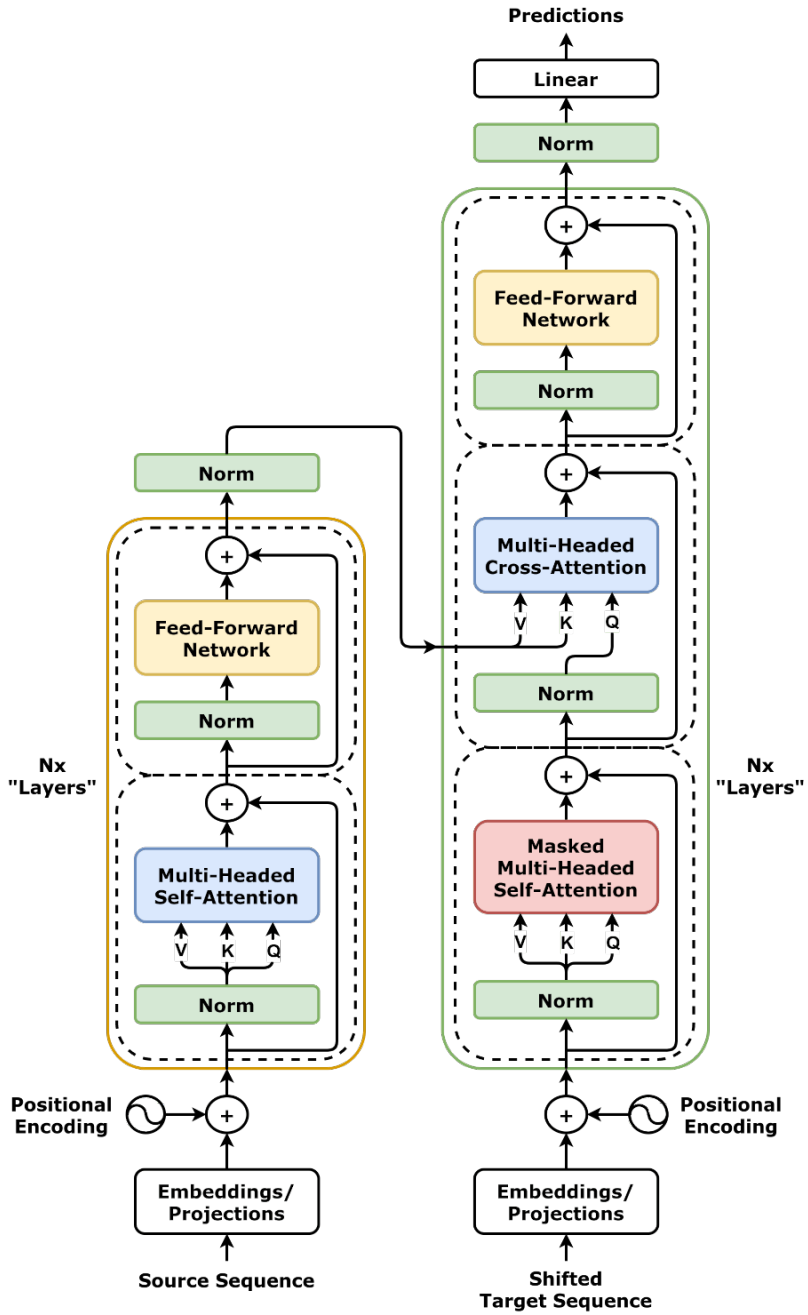
While a monkey is used as a mechanism for the thought experiment, it would be unlikely to ever write Hamlet.

[https://en.wikipedia.org/wiki/Infinite_monkey_theorem]

LLM: Transformers diagram

nanoGPT
configs/output/params:

X, $\mathbf{y}_{true}$,
max_iters,
AdamW,
train loss,
val loss,
n_layer,
n_head,
-temperature=0.8,
-start="Der kom
en solda"



nanoGPT setup: config/train_hca.py

```
dataset = 'hca'

[..removed-some-stuff..]

gradient_accumulation_steps = 1
batch_size = 64
block_size = 256     # context of up to 256 previous characters

# tiny-winy baby GPT model :)
n_layer = 3
n_head  = 3
n_embd  = 192
dropout = 0.2

learning_rate = 1e-3 # with baby net: afford to go a bit higher
max_iters     = 5000
lr_decay_iters= 5000 # make equal to max_iters usually
min_lr        = 1e-4 # learning_rate / 10 usually
beta2         = 0.99 # make a bit bigger because number of
                     #   tokens per iter is small
warmup_iters  = 100  # not super necessary potentially
```

Python package managers

..or setup/run nanoGPT on Your own hardware..

1. `pip`:
   fast, no version chekcing, `pip list -user`,
2. `conda`:
   low, version checking, `solving dependencies..`,
3. `mamba`:
   faster conda partial in `C++`,
4. `micromamba`:
   even faster conda, full `C++` implementation.

And virtual Python environments via: `venv` or `conda env`:

```
# conda environments:
#
base /opt/anaconda-2024.02
```

Qd MNIST Search Quest II

Fra tidligere semester

```
E2024:  Grp01:  score=0.9906, KERAS (DISQUALIFIED due t
E2024:  Grp30:  score=0.9725, gradientboostingclassifie
F2024:  Grp25:  score=0.972, RandomForestClassifier
E2023:  Grp13:  score=0.984, SVC
F2023:  Grp01:  score=0.981, SVC
E2022:  Grp05:  score=0.983, SVC
F2022:  Grp05:  score=0.990, SGDClassifier (iris?)
E2021:  Grp28:  score=0.973, KNN
F2021:  Grp20:  score=0.979, SVC
```

Qd MNIST Search Quest II

Grp24: Best estimator

CTOR: SVC(C=10, max_iter=10000, random_state=42, tol=1e-06)

```
01/04 16:53> Grp24:  score=0.98241, model=SVC
01/04 15:53> Grp24:  score=0.98237, model=SVC
01/04 14:46> Grp24:  score=0.98227, model=SVC
07/04 13:23> Grp04:  score=0.97620, model=HistGradientBoostingClf
07/04 18:47> Grp??:  score=0.97555, model=Pipeline, (30?)
07/04 23:23> Grp14:  score=0.97106, model=RandomForestClassifier
07/04 14:41> Grp21:  score=0.96924, model=RandomForestClassifier (27?)
07/04 11:27> Grp05:  score=0.96355, model=RandomForestClassifier
06/04 17:22> Grp20:  score=0.96190, model=RandomForestClf, dat=iris
07/04 23:05> Grp12:  score=0.91700, model=??
01/04 14:27> GRP17:  score=0.91314, model=SGDClassifier
05/04 22:30> Grp07:  score=0.91098, model=SGDClassifier
08/04 09:13> Grp23:  score=0.90575, model=SGDClassifier
01/04 13:52> Grp24:  score=0.90980, model=LinearSVC
01/04 13:41> Grp24:  score=0.89816, model=SGDClassifier
01/04 15:48> Grp28:  score=0.88098, model=SGDClassifier
07/04 20:56> Grp10:  score=0.85582, model=SGDClassifier
```

INTRO. TO ADVANCED CNNS

Computer Vision Tasks in ML

- classification,

- object detection,

- segmentation,
  - semantic segmentation
  - instance segmentation
  - panoptic segmentation

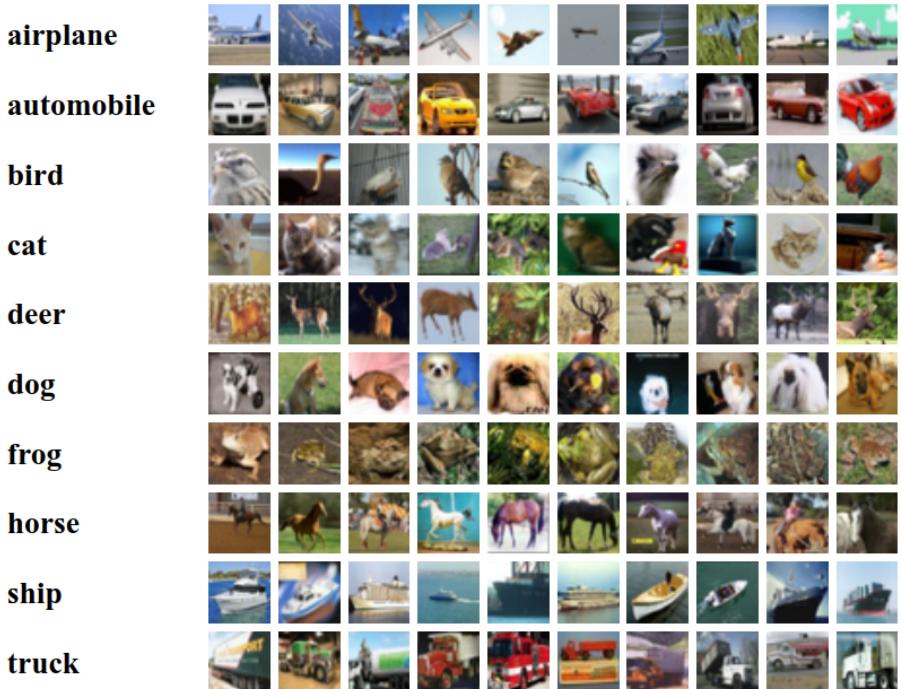
Figure 14-26. Semantic segmentation

Computer Vision Related Datasets

CICAR-10/100 (Canadian Institute For Advanced Research):

- *" The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset [..] The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. "*

[https://www.cs.toronto.edu/ kriz/cifar.html]



Computer Vision Related Datasets

COCO (Common Objects in Context):

- *" COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features: Object segmentation, Recognition in context, [..],* 330K images *(>200K labeled),* 1.5 million object instances, 80 object categories,.. *[..] "*

[https://cocodataset.org/#home]

Computer Vision Related Datasets

ImageNet: ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

- *" The ImageNet project is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also pro-vided ImageNet contains more than 20,000 categories [..] "*

[https://en.wikipedia.org/wiki/ImageNet]

Data Augmentation

Data Augmentation

Data augmentation artificially increases the size of the training set by generating many realistic variants of each training instance. This reduces overfitting, making this a regularization technique. The generated instances should be as realistic as possible: ideally, given an image from the augmented training set, a human should not be able to tell whether it was augmented or not. Moreover, simply adding white noise will not help; the modifications should be learnable (white noise is not).

For example, you can slightly shift, rotate, and resize every picture in the training set by various amounts and add the resulting pictures to the training set (see Figure 14-12). This forces the model to be more tolerant to variations in the position, orientation, and size of the objects in the pictures. If you want the model to be more tolerant to different lighting conditions, you can similarly generate many images with various contrasts. In general, you can also flip the pictures horizontally (except for text, and other non-symmetrical objects). By combining these transformations you can greatly increase the size of your training set.

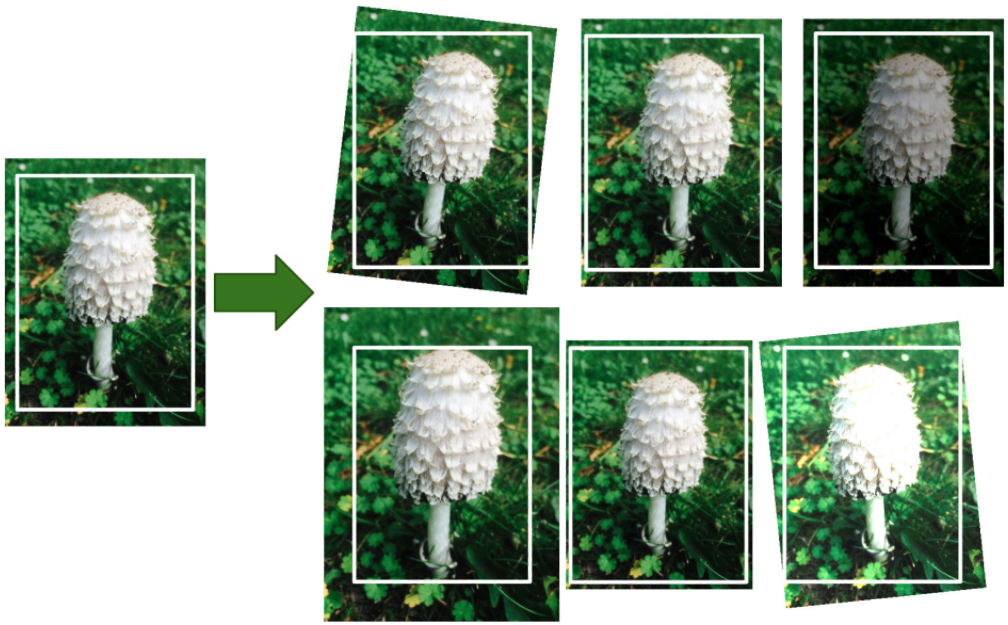
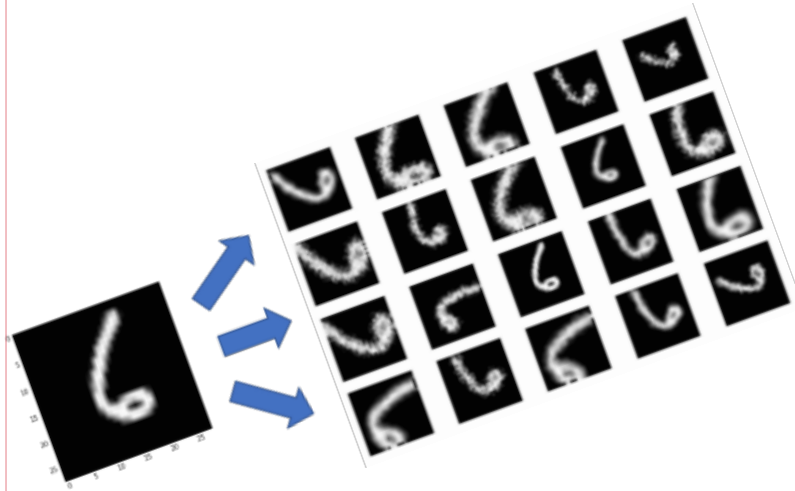
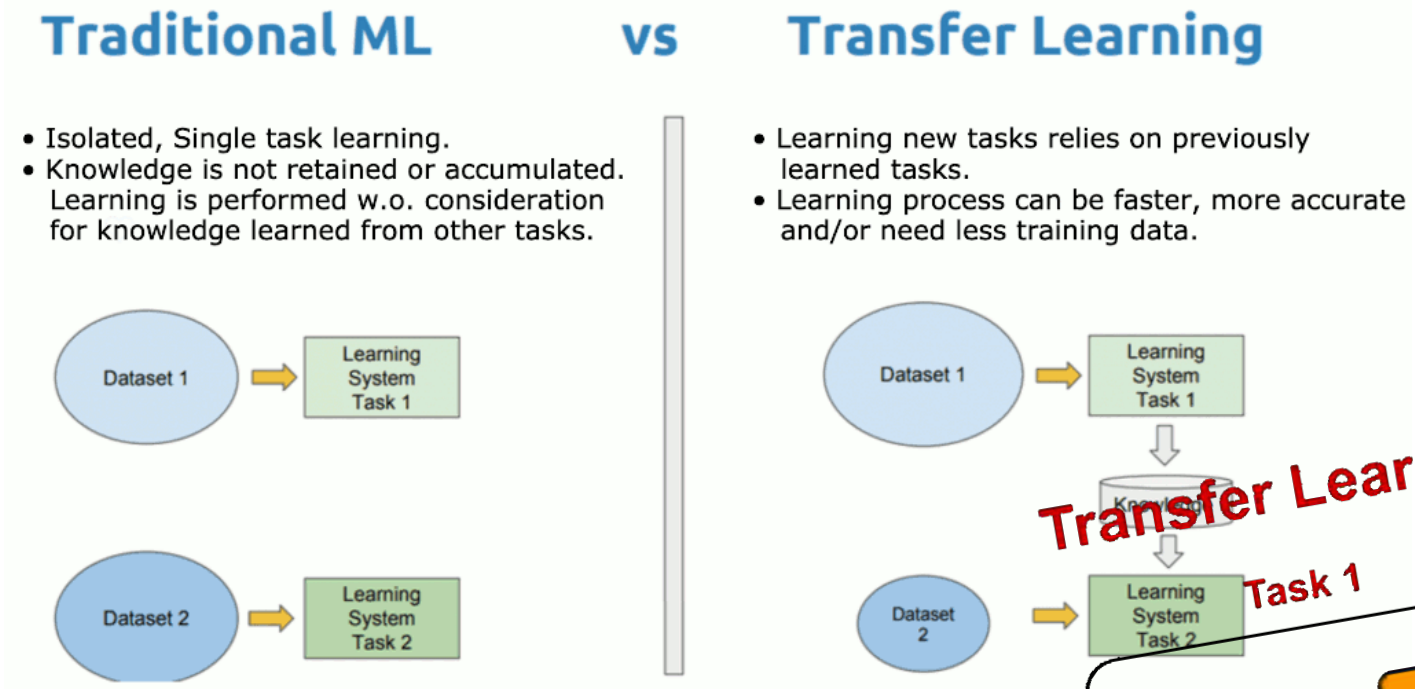


Figure 14-12. Generating new training instances from existing ones



Pretrained Models and Transfer Learning



[https://research.aimultiple.com/wp-content/uploads/2020/07/Transfer_Learning_Explanation-1920x943.png]
[https://www.topbots.com/wp-content/uploads/2019/12/cover_transfer_learning_1600px_web.jpg]

CNN Architectures

GoogLeNet, ResNet (Xception, SENet)

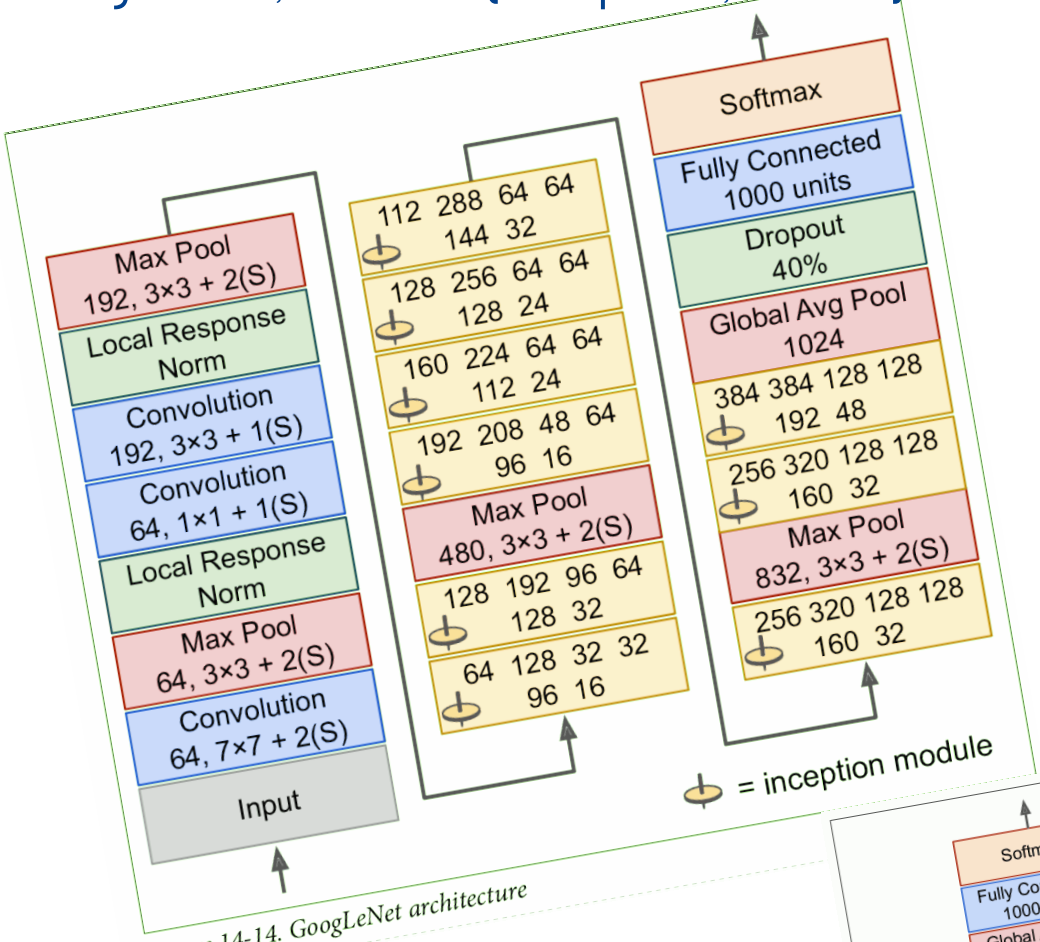


Figure 14-14. GoogLeNet architecture

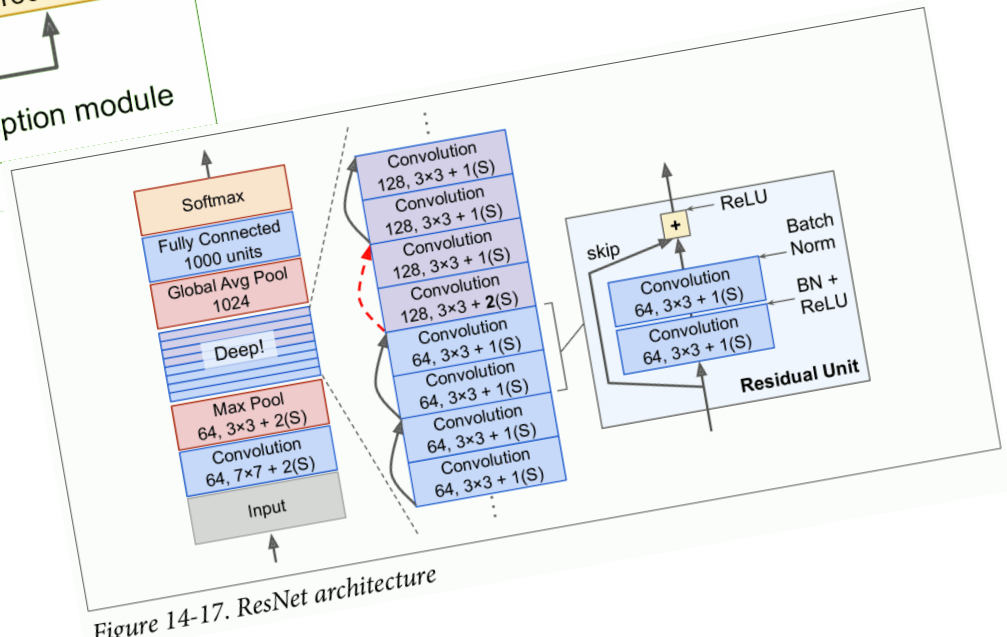


Figure 14-17. ResNet architecture

CNN Architectures

CNN Architectures:

- Fast R-CNN: .. (older)

  (Region-based Convolutional Neural Networks)
- Faster/Mask R-CNN:
  - ResNet-50 (R50), ResNet-101 (R101)
  - ResNeXt-152 (X101)?

CNN 'Frameworks':

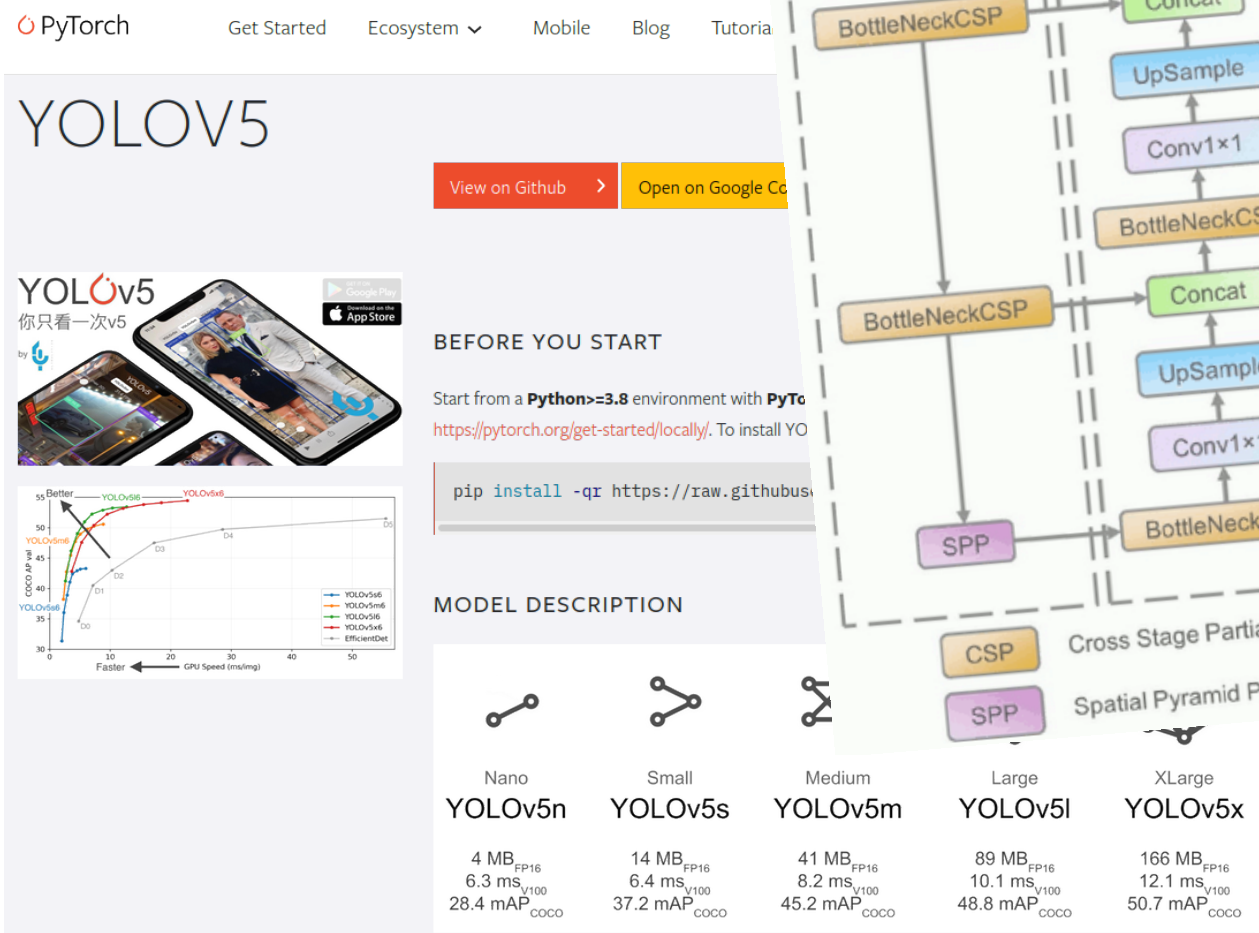
- Framework: Yolo (C, PyTorch)

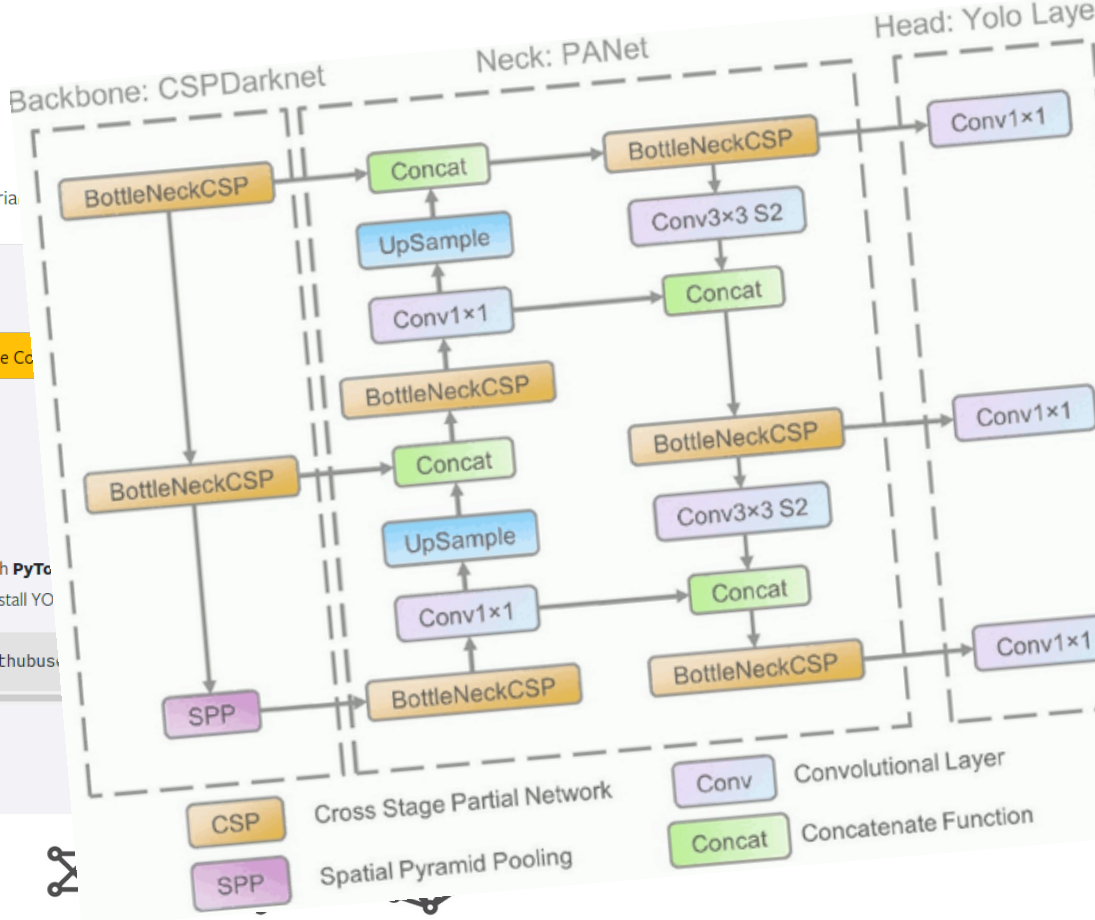
  Model Zoo:
  - Darknet53 nano, small, medium, large, xlarge
- Framwork: Detectron2 (PyTorch)

  Model Zoo:
  - Faster R-CNN: R50, R101, X101
  - RetinaNet: R50, R101 bases
  - RPN and Fast R-CNN: RPN R50-XX, Fast R-CNN R50-FPN..

CNN Architectures

YOLO version 3/4/5[..]/8

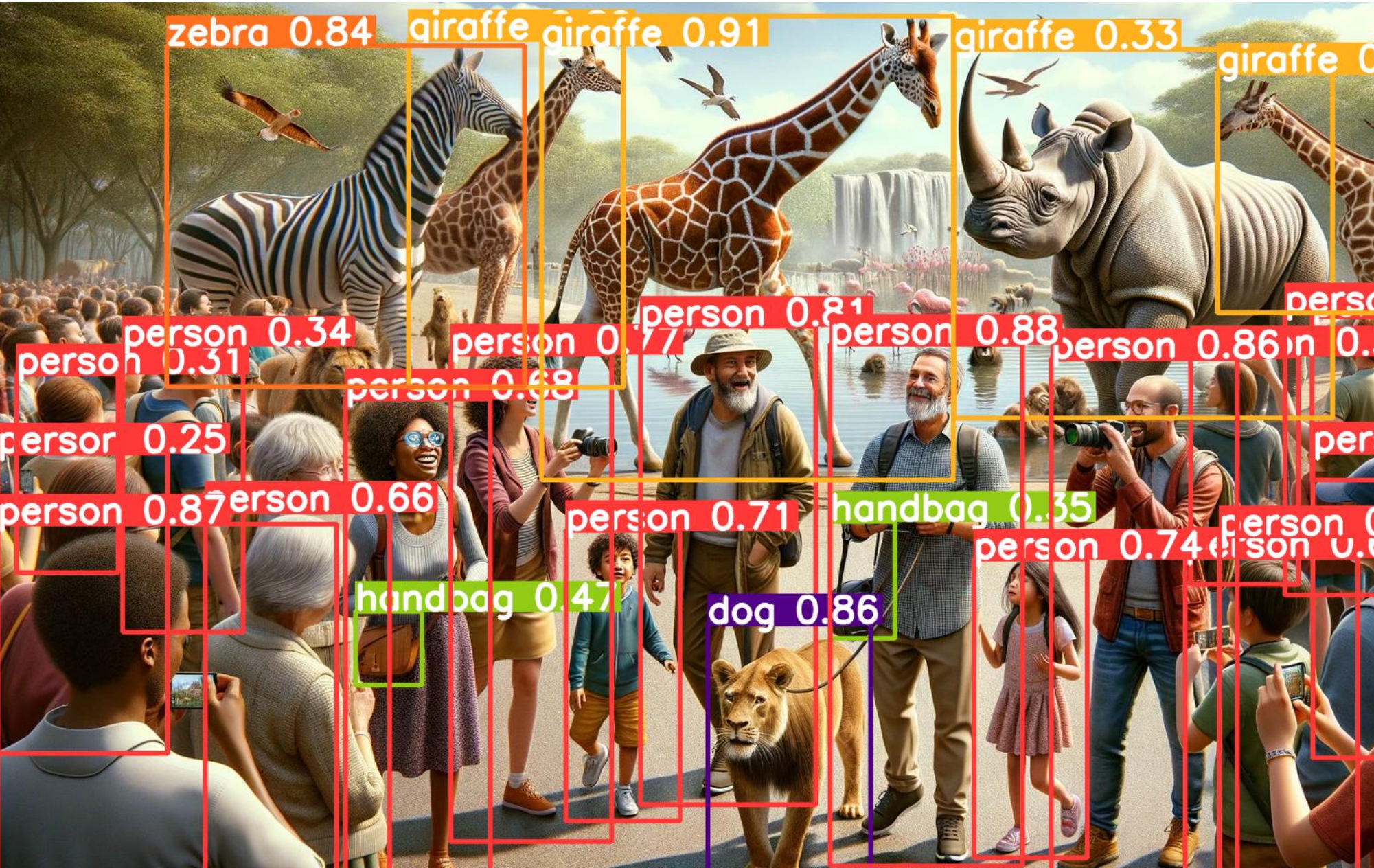




- *" All the YOLOv5 models are composed of the same 3 components: CSP-Darknet53 as a backbone, SPP and PANet in the model neck and the head used in YOLOv4 "*

[https://iq.opengenus.org/yolov5/]

YOLO Examples



Pretrained Models for Transfer Learning

Detectron2 ImageNet Pretrained Zoo

ImageNet Pretrained Models

It's common to initialize from backbone models pre-trained on ImageNet classification tasks. The following backbone models are available:

- R-50.pkl: converted copy of MSRA's original ResNet-50 model.
- R-101.pkl: converted copy of MSRA's original ResNet-101 model.
- X-101-32x8d.pkl: ResNeXt-101-32x8d model trained with Caffe2 at FB.
- R-50.pkl (torchvision): converted copy of torchvision's ResNet-50 model. More details can be found in the conversion script.

Note that the above models have **different** format from those provided in Detectron: we do not fuse BatchNorm into an affine layer. Pretrained models in Detectron's format can still be used. For example:

- X-152-32x8d-IN5k.pkl: ResNeXt-152-32x8d model trained on ImageNet-5k with Caffe2 at FB (see ResNeXt paper for details on ImageNet-5k).
- R-50-GN.pkl: ResNet-50 with Group Normalization.
- R-101-GN.pkl: ResNet-101 with Group Normalization.

[https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md]

Performance Metric: IoU

Intersection over Union (IoU)

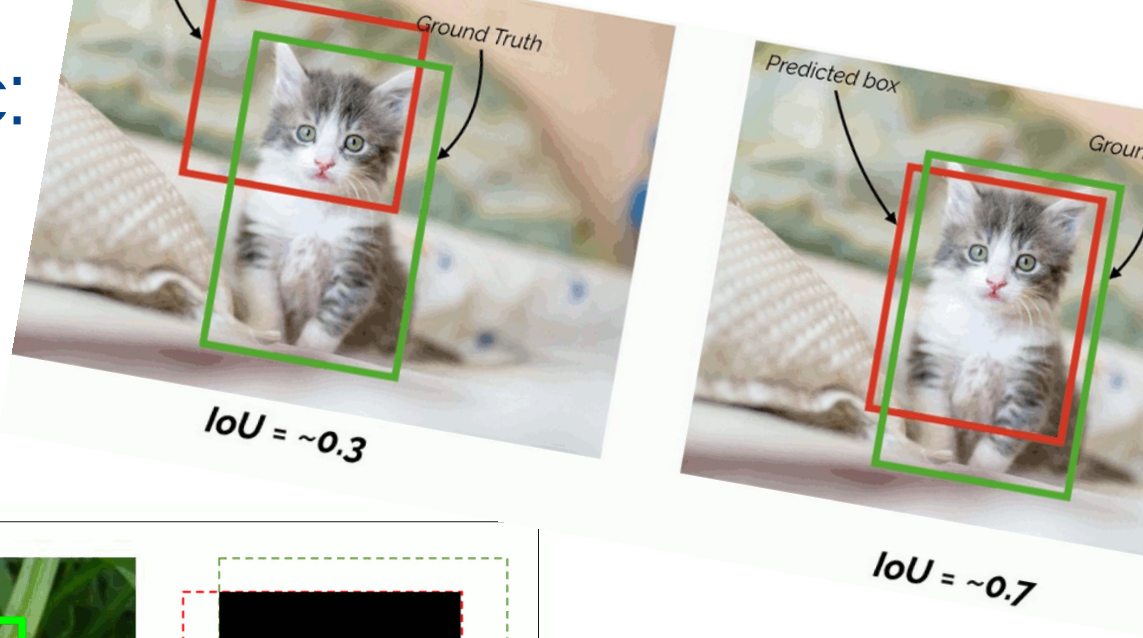


Figure 14-23. Intersection over Union (IoU) Metric for Bounding Boxes

Range: 0 <= IoU <= 1 (best)

New hyperparameter:

threshold for 'correct' detection, say IoU > 0.5.

Performance Metric: mAP

Mean Average Precision (mAP)

Mean Average Precision (mAP)

A very common metric used in object detection tasks is the *mean Average Precision* (mAP). "Mean Average" sounds a bit redundant, doesn't it? To understand this metric, let's go back to two classification metrics we discussed in Chapter 3: precision and recall. Remember the tradeoff: the higher the recall, the lower the precision. You can visualize this in a Precision/Recall curve (see Figure 3-5). To summarize this curve into a single number, we could compute its Area Under the Curve (AUC). But note that the Precision/Recall curve may contain a few sections where precision actually goes up when recall increases, especially at low recall values (you can see this at the top left of Figure 3-5). This is one of the motivations for the mAP metric.

Suppose the classifier has a 90% precision at 10% recall, but a 96% precision at 20% recall: there's really no tradeoff here: it simply makes more sense to use the classifier at 20% recall rather than at 10% recall, as you will get both higher recall and higher precision. So instead of looking at the precision *at* 10% recall, we should really be looking at the *maximum* precision that the classifier can offer with *at least* 10% recall. It would be 96%, not 90%. So one way to get a fair idea of the model's performance is to compute the maximum precision you can get with at least 0% recall, then 10% recall, 20%, and so on up to 100%, and then calculate the mean of these maximum precisions. This is called the *Average Precision* (AP) metric. Now when there are more than 2 classes, we can compute the AP for each class, and then compute the mean AP (mAP). That's it!

However, in an object detection systems, there is an additional level of complexity: what if the system detected the correct class, but at the wrong location (i.e., the bounding box is completely off)? Surely we should not count this as a positive prediction. So one approach is to define an IOU threshold: for example, we may consider that a prediction is correct only if the IOU is greater than, say, 0.5, and the predicted class is correct. The corresponding mAP is generally noted mAP@0.5 (or mAP@50%, or sometimes just AP_{50}). In some competitions (such as the Pascal VOC challenge), this is what is done. In others (such as the COCO competition), the mAP is computed for different IOU thresholds (0.50, 0.55, 0.60, …, 0.95), and the final metric is the mean of all these mAPs (noted AP@[.50:.95] or AP@[.50:0.05:.95]). Yes, that's a mean mean average.

PR area under the curve (PR AUC)

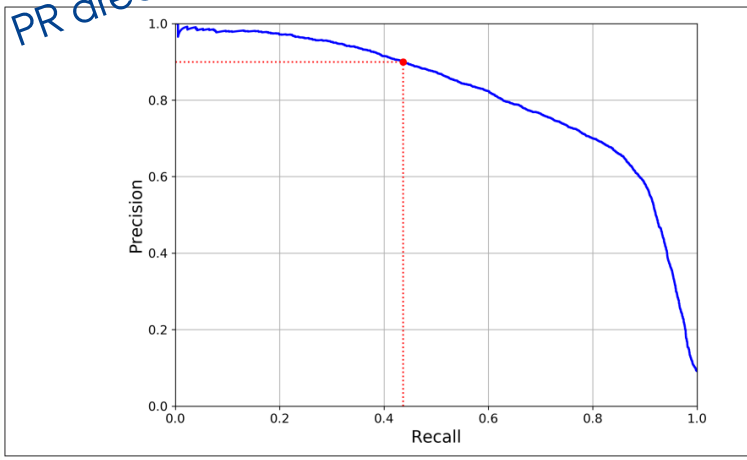


Figure 3-5. Precision versus recall

Receiver operating characteristic, ROC, area under the curve (ROC AUC)

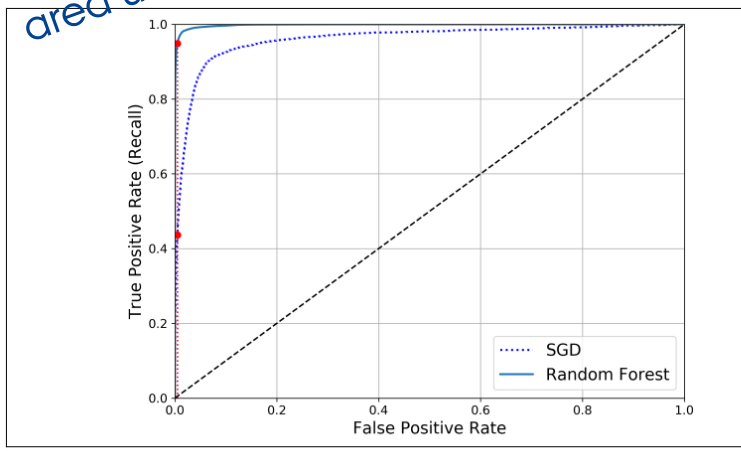
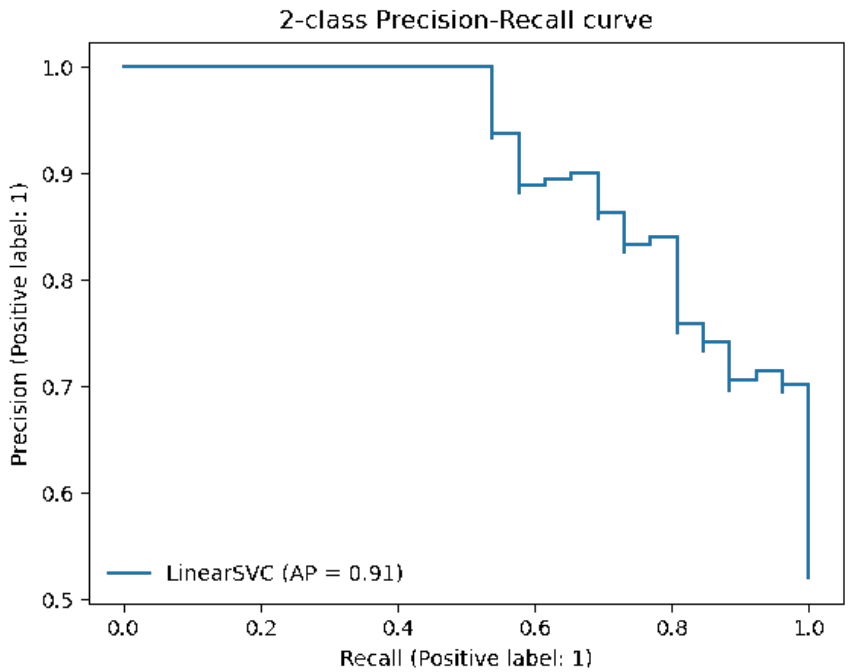


Figure 3-7. Comparing ROC curves

Performance Metric: mAP

Mean Average Precision (mAP)



Average Precision, AP summarizes a precision-recall curve

$$AP = \sum_n (R_n - R_{(n-1)}) P_n$$

and threhold $n = 0.1, 0.2, .., 0.9$ for predicted scores

```
pred_scores = [0.7, 0.3, 0.5, 0.9 [..], 0.3]
```

mAP is the mean over all classes for AP.

mAP@0.5 = mAP@50% = AP_{50} =>

mAP at IoU > 0.5

and

mAP@[.50:.95] or mAP@[.50:0.05:.95]

is **mean mean average!**

Naïve Object Detection: Sliding Window

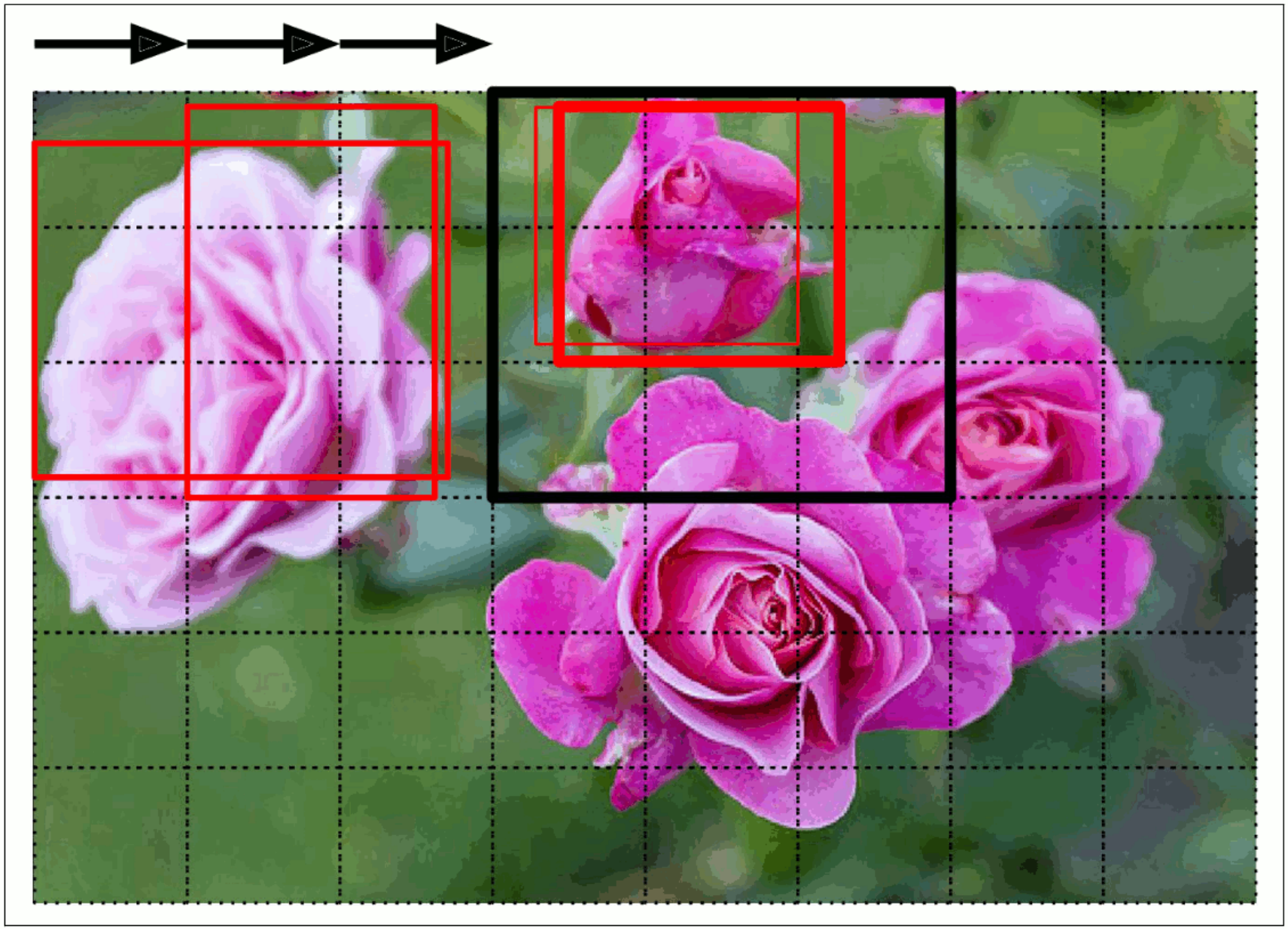


Figure 14-24. Detecting Multiple Objects by Sliding a CNN Across the Image

Fully Connected to Fully Convolutional Networks: YOLO

- It outputs two bounding boxes for each grid cell (instead of just one), which allows the model to handle cases where two objects are so close to each other that their bounding box centers lie within the same cell. Each bounding box also comes with its own objectness score.
- YOLO also outputs a class probability distribution for each grid cell, predicting 20 class probabilities per grid cell since YOLO was trained on the PASCAL VOC dataset, which contains 20 classes. This produces a coarse *class probability map*. Note that the model predicts one class probability distribution per grid cell, not per bounding box. However, it's possible to estimate class probabilities for each bounding box during postprocessing, by measuring how well each bounding box matches each class in the class probability map. For example, imagine a picture of a person standing in front of a car. There will be two bounding boxes: one large horizontal one for the car, and a smaller vertical one for the person. These bounding boxes may have their centers within the same grid cell. So how can we tell which class should be assigned to each bounding box? Well, the class probability map will contain a large region where the "car" class is dominant, and inside it there will be a smaller region where the "person" class is dominant. Hopefully, the car's bounding box will roughly match the "car" region, while the person's bounding box will roughly match the "person" region: this will allow the correct class to be assigned to each bounding box.

YOLO was originally developed using Darknet, an open source deep learning framework initially developed in C by Joseph Redmon, but it was soon ported to TensorFlow, Keras, PyTorch, and more. It was continuously improved over the years, with YOLOv2, YOLOv3, and YOLO9000 (again by Joseph Redmon et al.), YOLOv4 (by Alexey Bochkovskiy et al.), YOLOv5 (by Glenn Jocher), and PP-YOLO (by Xiang Long et al.).

Each version brought some impressive improvements in speed and accuracy, using a variety of techniques; for example, YOLOv3 boosted accuracy in part thanks to *anchor priors*, exploiting the fact that some bounding box shapes are more likely than others, depending on the class (e.g., people tend to have vertical bounding boxes, while cars usually don't). They also increased the number of bounding boxes per grid cell, they trained on different datasets with many more classes (up to 9,000 classes organized in a hierarchy in the case of YOLO9000), they added skip connections to recover some of the spatial resolution that is lost in the CNN (we will discuss this shortly, when we look at semantic segmentation), and much more. There are many variants of these models too, such as YOLOv4-tiny, which is optimized to be trained on less powerful machines and which can run extremely fast (at over 1,000 frames p

Mean Average Precision

A very common metric used in object detection tasks is the mean average precision. "Mean average" sounds a bit redundant, doesn't it? To understand this metric, let's go back to two classification metrics we discussed in Chapter 3: precision and recall. Remember the trade-off: the higher the recall, the lower the precision. You can visualize this in a precision/recall curve (see Figure 3-6). To summarize this curve into a single number, we could compute its area under the curve (AUC). But note that the precision/recall curve may contain a few sections where precision actually goes up when recall increases, especially at low recall values (you can see this at the top left of Figure 3-6). This is one of the motivations for the mAP metric.

Suppose the classifier has 90% precision at 10% recall, but 96% precision at 20% recall. There's really no trade-off here: it simply makes more sense to use the classifier at 20% recall rather than at 10% recall, as you will get both higher recall and higher precision. So instead of looking at the precision *at* 10% recall, we should really be looking at the *maximum* precision that the classifier can offer with *at least* 10% recall. It would be 96%, not 90%. Therefore, one way to get a fair idea of the model's performance is to compute the maximum precision you can get with at least 0% recall, then 10% recall, 20%, and so on up to 100%, and then calculate the mean of these maximum precisions. This is called the *average precision* (AP) metric. Now when there are more than two classes, we can compute the AP for each class, and then compute the mean AP (mAP). That's it!

In an object detection system, there is an additional level of complexity: what if the system detected the correct class, but at the wrong location (i.e., the bounding box is completely off)? Surely we should not count this as a positive prediction. One approach is to define an IoU threshold: for example, we may consider that a prediction is correct only if the IoU is greater than, say, 0.5, and the predicted class is correct. The corresponding mAP is generally noted mAP@0.5 (or mAP@50%, or sometimes just AP_{50}). In some competitions (such as the PASCAL VOC challenge), this is what is done. In others (such as the COCO competition), the mAP is computed for different IoU thresholds (0.50, 0.55, 0.60, ..., 0.95), and the final metric is the mean of all these mAPs (noted mAP@[.50:.95] or mAP@[.50:0.05:.95]). Yes, that's a mean mean average.

[HOML 3rd.ed., pp.528-529.]

YOLO, You Only Look Once

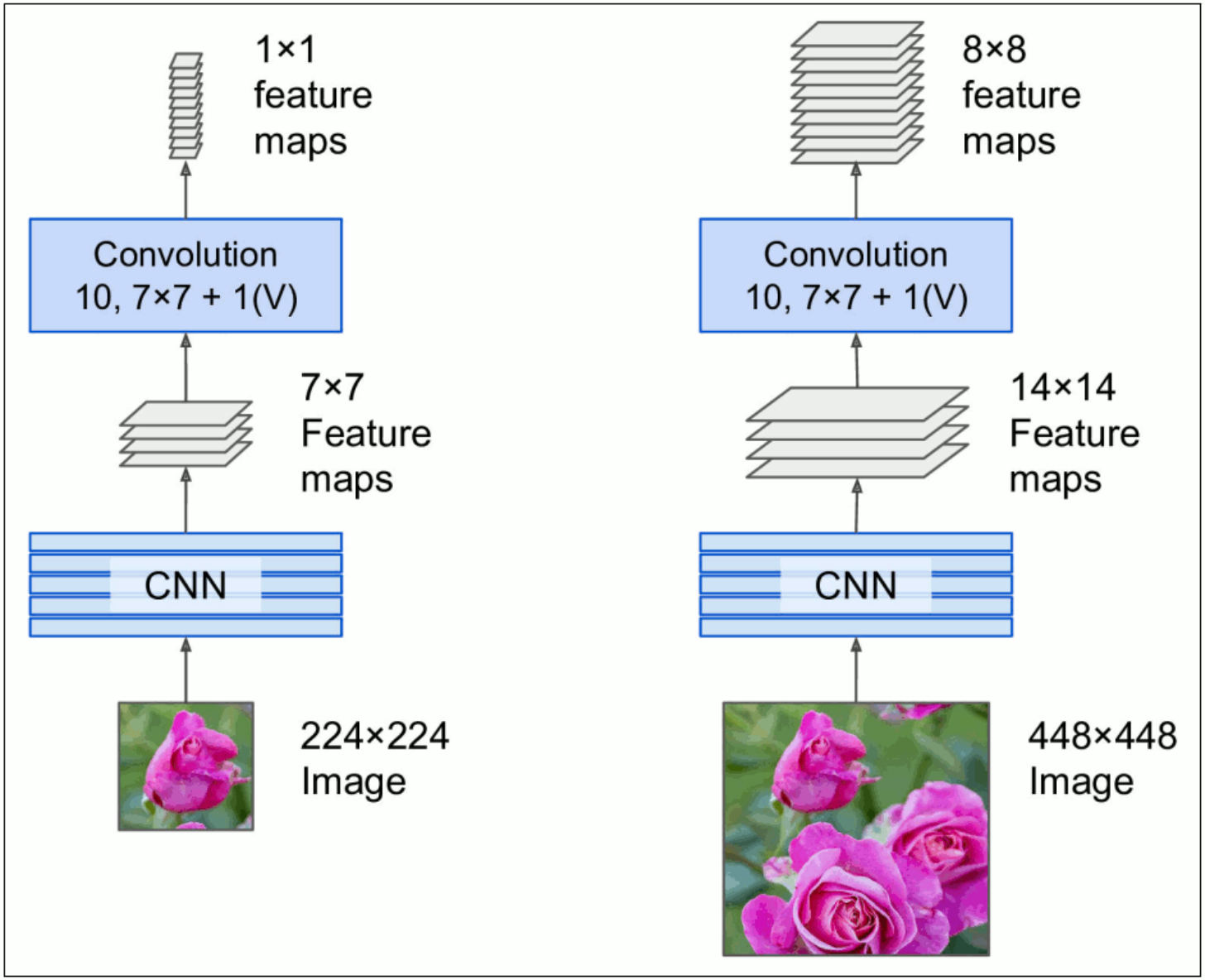
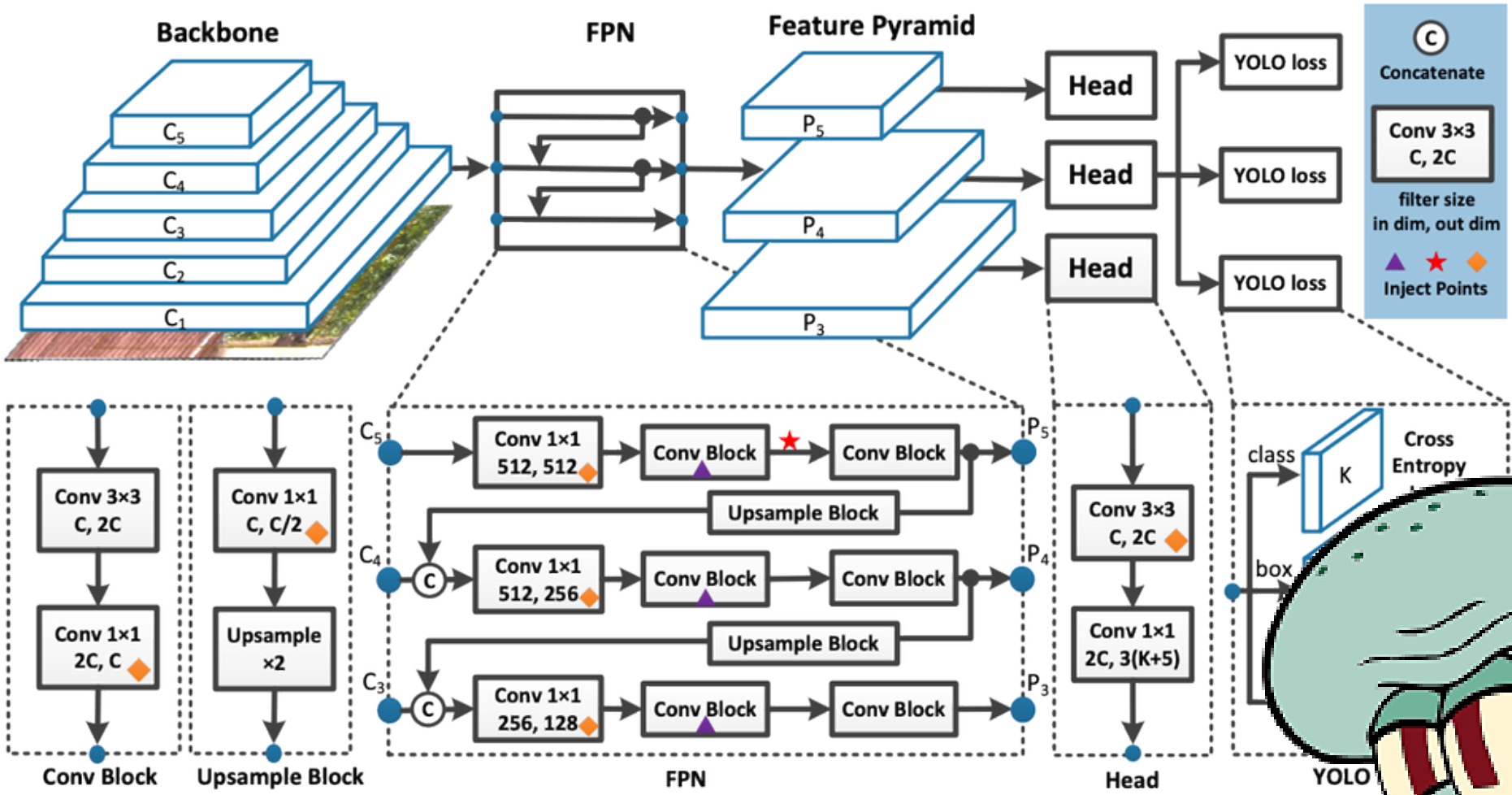


Figure 14-25. A Fully Convolutional Network Processing a Small Image (left) and a Large One (right)

YOLO, You Only Look Once



[https://blog.roboflow.com/yolov7-breakdown/]

YOLO, You Only Look Once

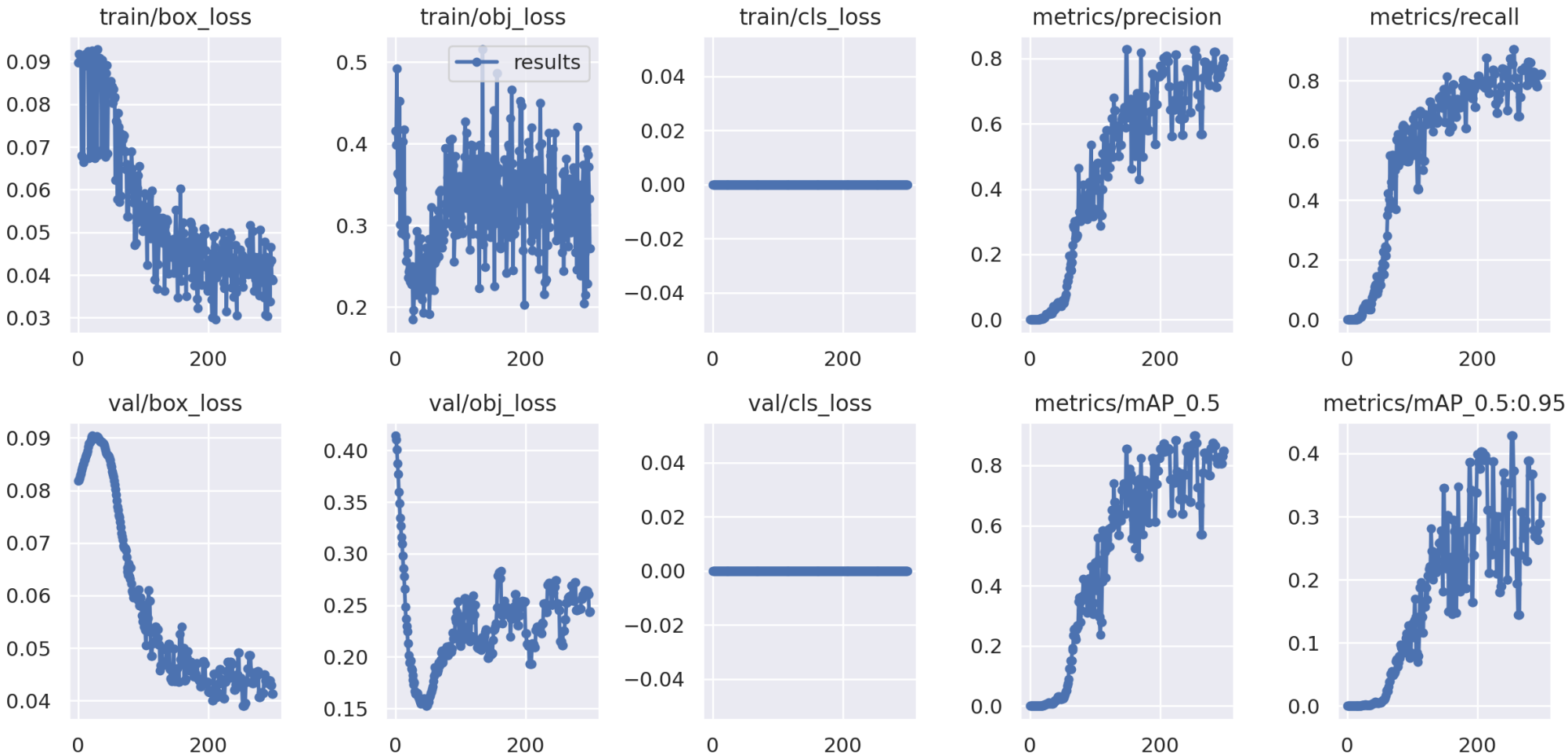
My Postdam car recognition on 26640x26640 sized images

YOLO, You Only Look Once

My Postdam car recognition on 26640x26640 sized images

YOLO, You Only Look Once

Generated learning curves



YOLO, You Only Look Once

Generated learning curves