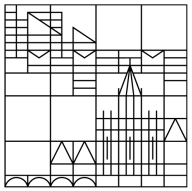
Universität Konstanz



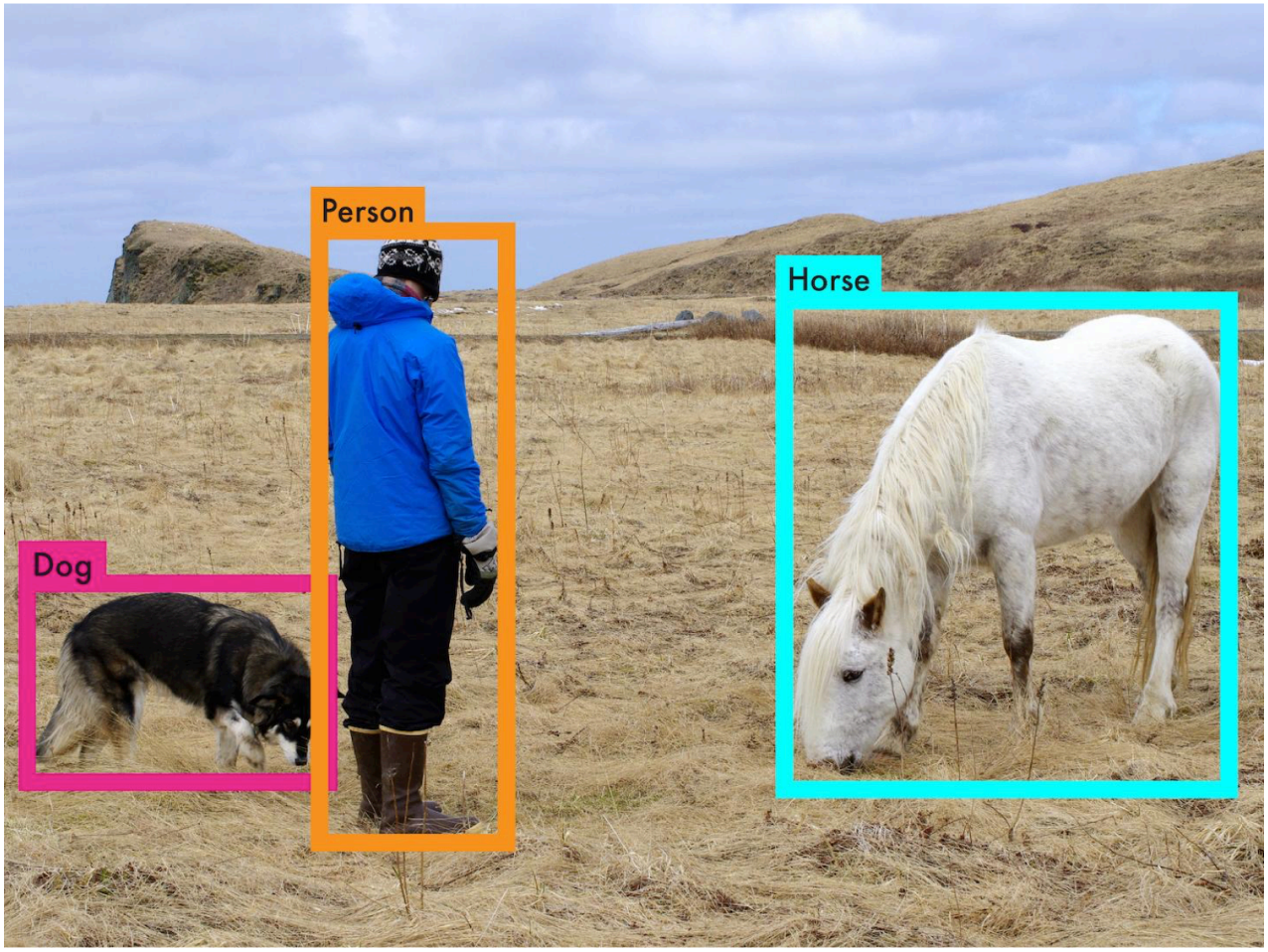
# Lecture 9 Deep Learning – Part 4 Object Detection and RNN

Dr. Hanhe Lin

Dept. of Computer and Information Science

University of Konstanz

# Object Detection




Bounding box and label

# CNN based algorithms

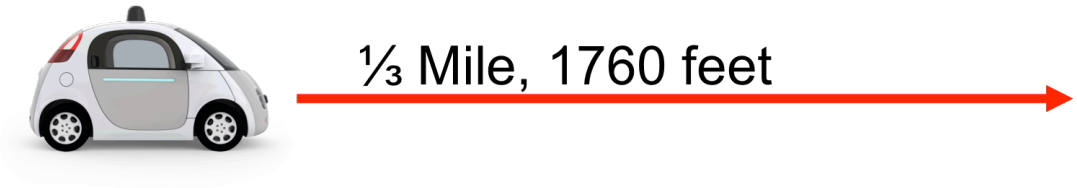
- Regional CNN (R-CNN)
  - Fast R-CNN
  - Faster R-CNN
- You only look once (YOLO)
- Hybrid: RetinaNet

# Accurate object detection is slow!

| | **Pascal 2007 mAP** | **Speed** |
|---|---|---|
| DPM v5 | 33.7 | 0.07 fps |
| R-CNN | 66.0 | 0.05 fps |

# Accurate object detection is slow!

| | **Pascal 2007 mAP** | **Speed** |
|---|---|---|
| DPM v5 | 33.7 | 0.07 fps |
| R-CNN | 66.0 | 0.05 fps |



# Accurate object detection is slow!

| | **Pascal 2007 mAP** | **Speed** |
|---|---|---|
| DPM v5 | 33.7 | 0.07 fps |
| R-CNN | 66.0 | 0.05 fps |
| Fast R-CNN | 70.0 | 0.5 fps |

# Accurate object detection is slow!

| | **Pascal 2007 mAP** | **Speed** |
|---|---|---|
| DPM v5 | 33.7 | 0.07 fps |
| R-CNN | 66.0 | 0.05 fps |
| Fast R-CNN | 70.0 | 0.5 fps |

# Accurate object detection is slow!

| | **Pascal 2007 mAP** | **Speed** |
|---|---|---|
| DPM v5 | 33.7 | 0.07 fps |
| R-CNN | 66.0 | 0.05 fps |
| Fast R-CNN | 70.0 | 0.5 fps |
| Faster R-CNN | 73.2 | 7 fps |

# Accurate object detection is slow!

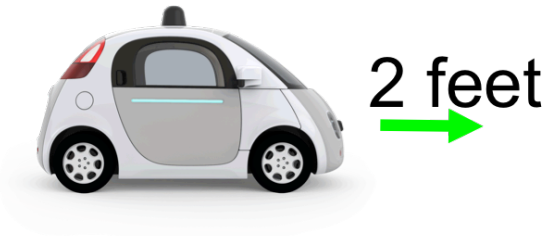
| | **Pascal 2007 mAP** | **Speed** |
|---|---|---|
| DPM v5 | 33.7 | 0.07 fps |
| R-CNN | 66.0 | 0.05 fps |
| Fast R-CNN | 70.0 | 0.5 fps |
| Faster R-CNN | 73.2 | 7 fps |

# Accurate object detection is slow!

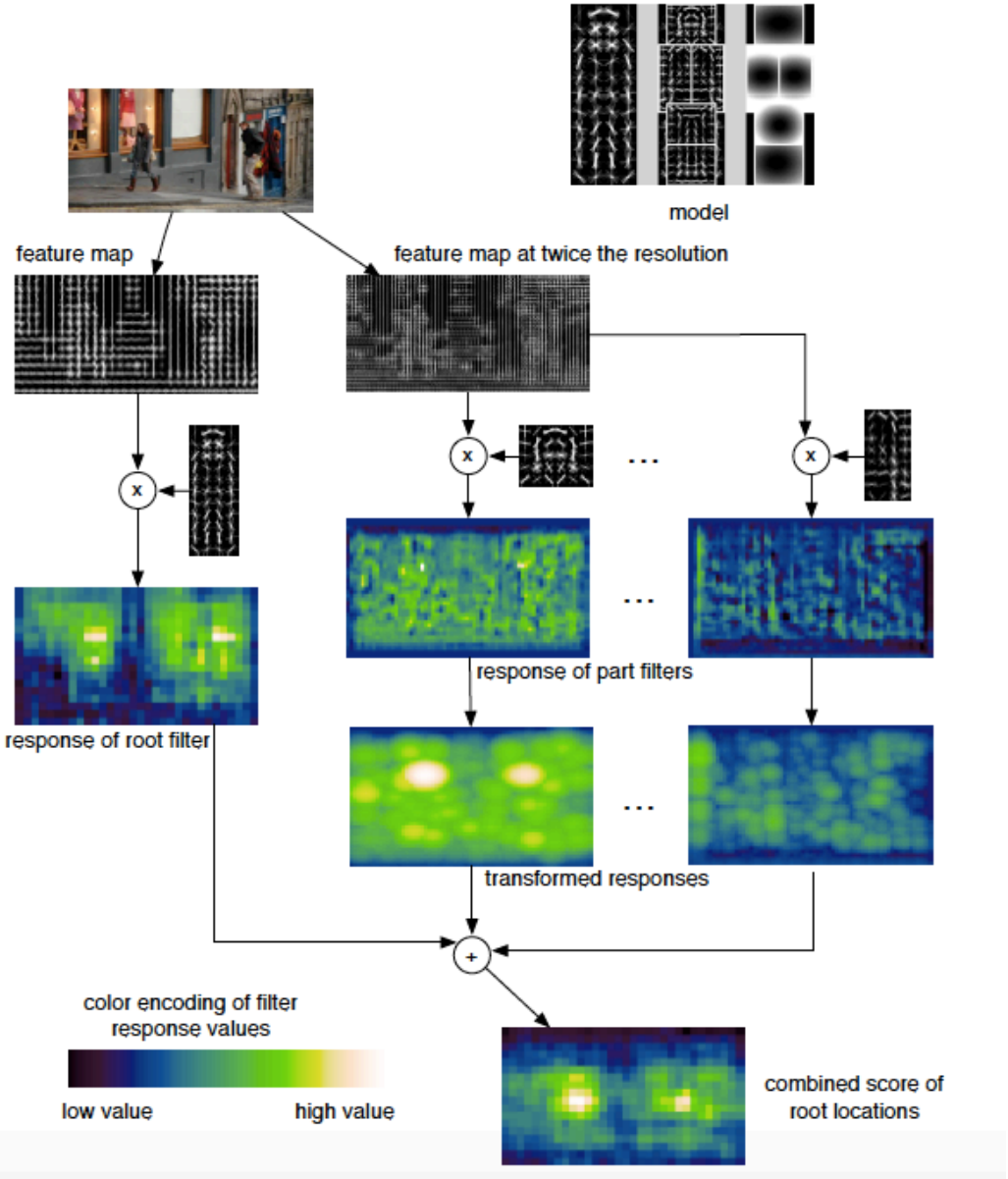
| | **Pascal 2007 mAP** | **Speed** |
|---|---|---|
| DPM v5 | 33.7 | 0.07 fps |
| R-CNN | 66.0 | 0.05 fps |
| Fast R-CNN | 70.0 | 0.5 fps |
| Faster R-CNN | 73.2 | 7 fps |
| YOLO | 69.0 | 45 fps |

# Accurate object detection is slow!

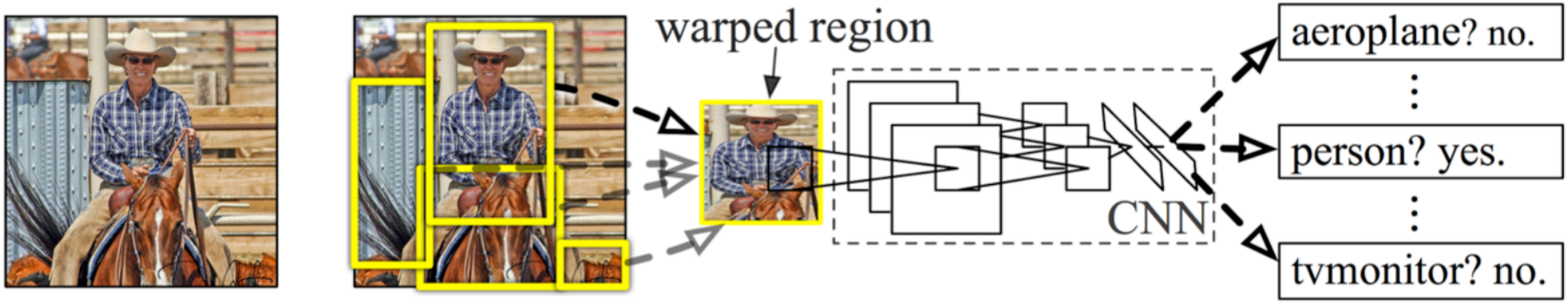
| | **Pascal 2007 mAP** | **Speed** |
|---|---|---|
| DPM v5 | 33.7 | 0.07 fps |
| R-CNN | 66.0 | 0.05 fps |
| Fast R-CNN | 70.0 | 0.5 fps |
| Faster R-CNN | 73.2 | 7 fps |
| YOLO | 69.0 | 45 fps |



# Deformable Part Model (DPM)



# R-CNN: *Regions with CNN features*



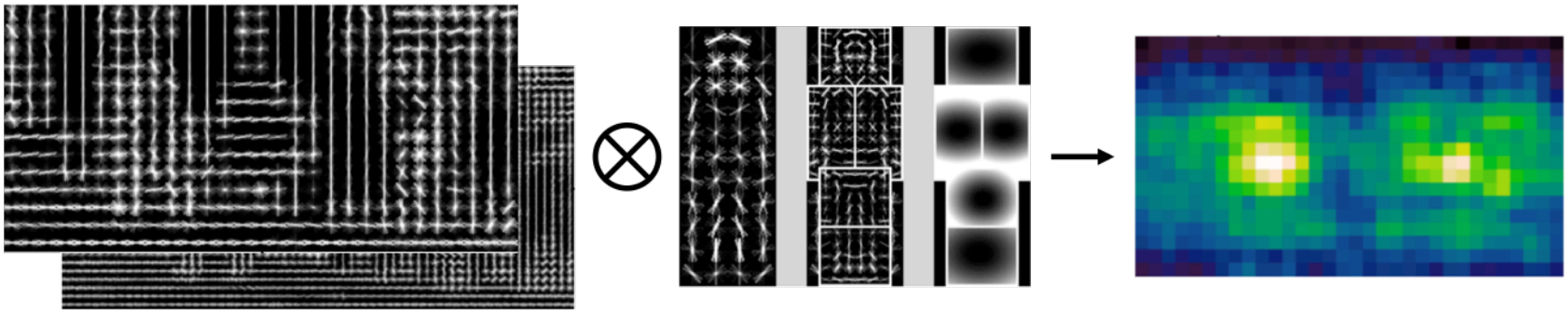

**1**. Input image

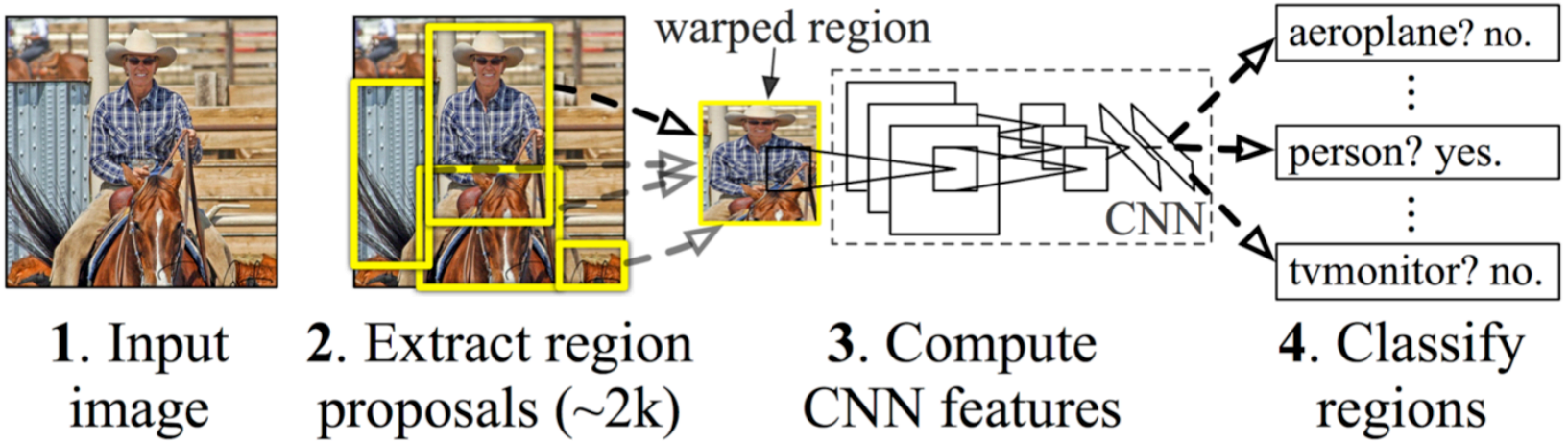
**2**. Extract region proposals (~2k)

**3**. Compute CNN features

**4**. Classify regions

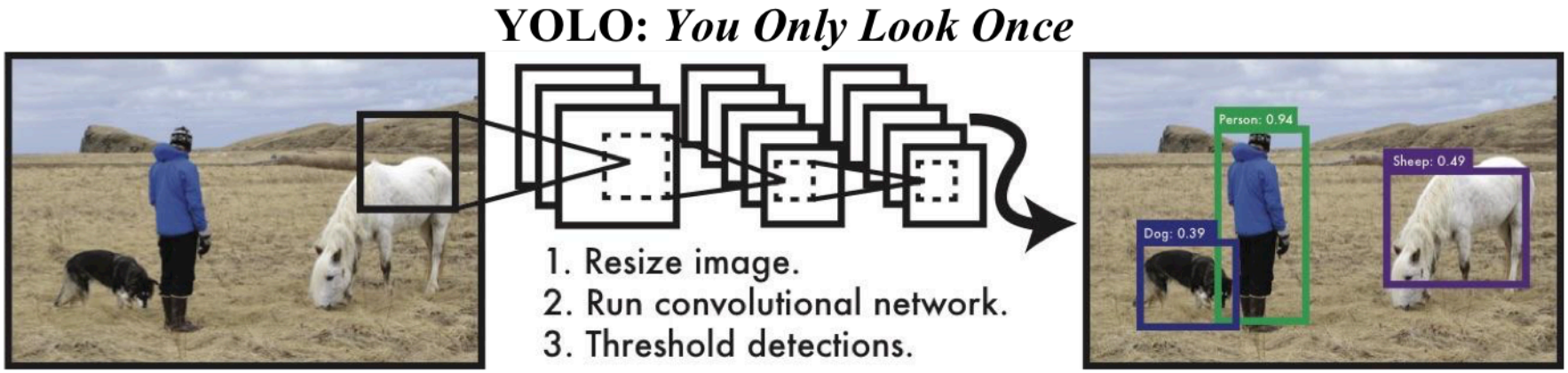
# Sliding window, DPM and R-CNN train region-based classifiers to perform detection



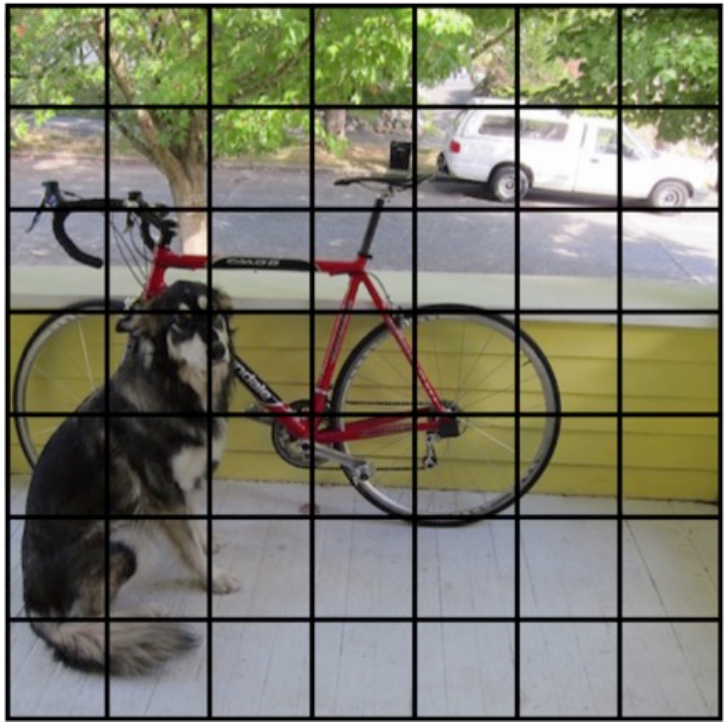


# How YOLO works

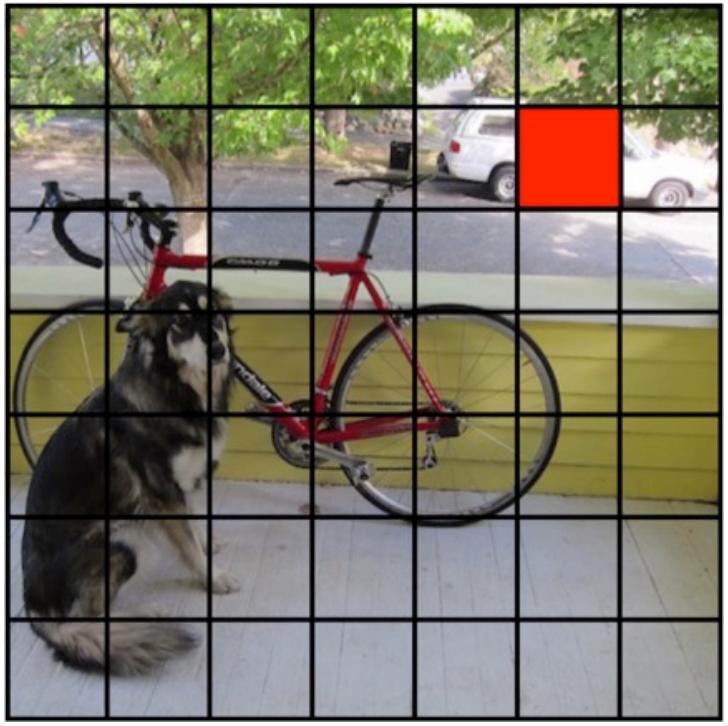
With YOLO, you only look once at an image to perform detection



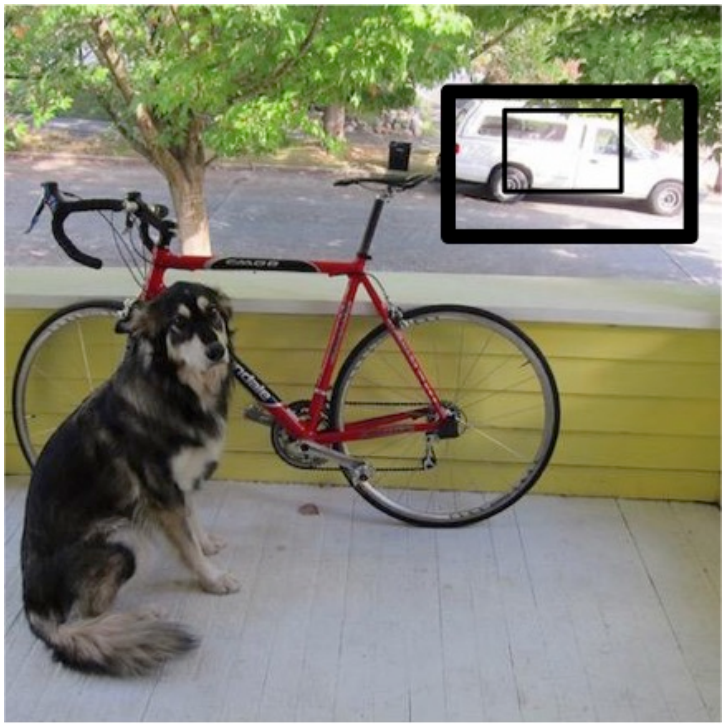
# Split the image into a grid



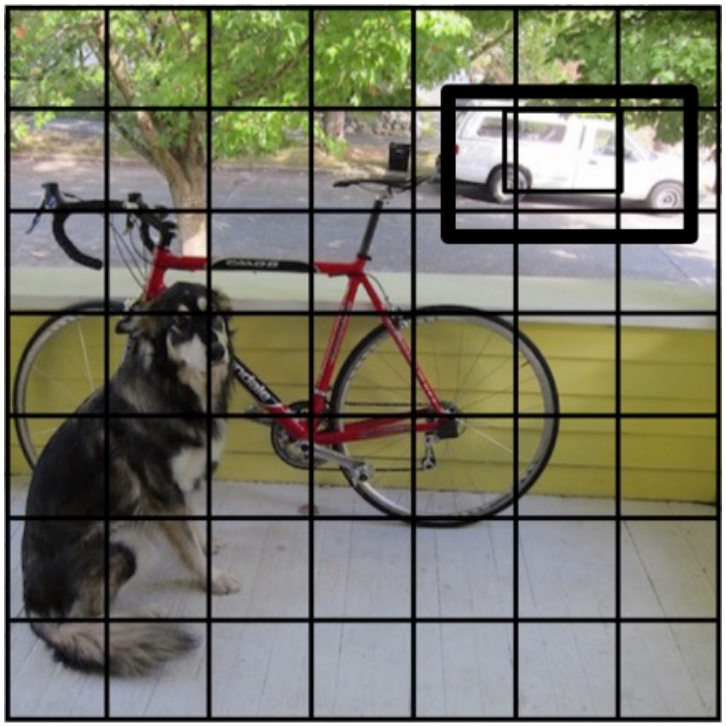
# Each cell predicts boxes and confidences: P(Object)



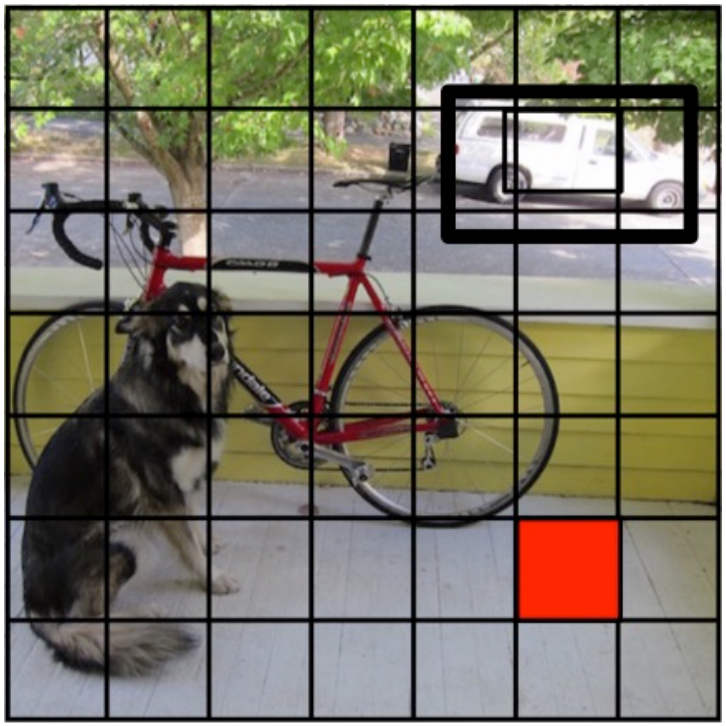
# Each cell predicts boxes and confidences: P(Object)



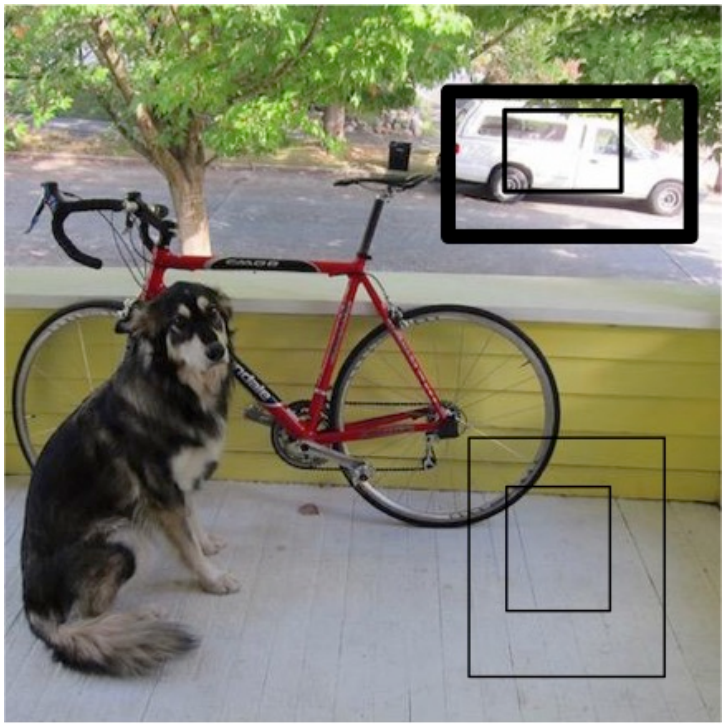
# Each cell predicts boxes and confidences: P(Object)



# Each cell predicts boxes and confidences: P(Object)

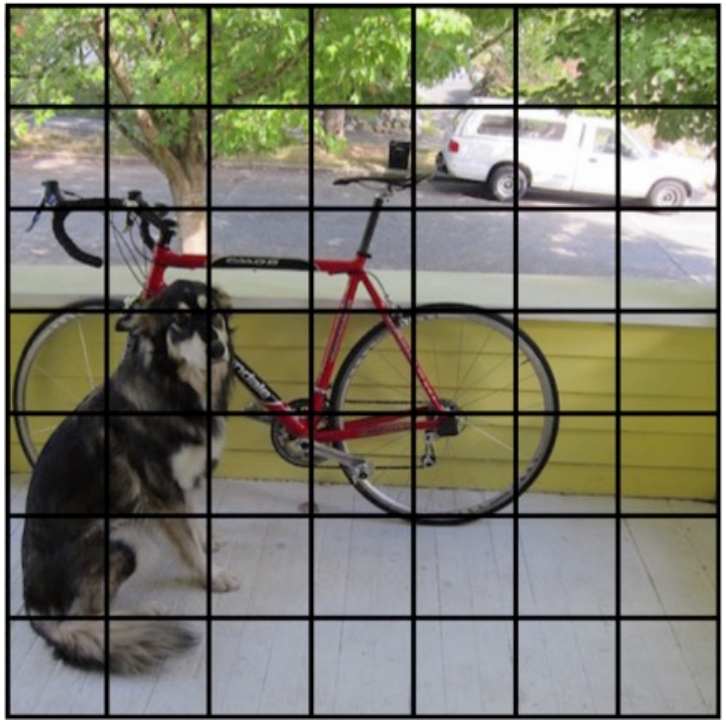


# Each cell predicts boxes and confidences: P(Object)

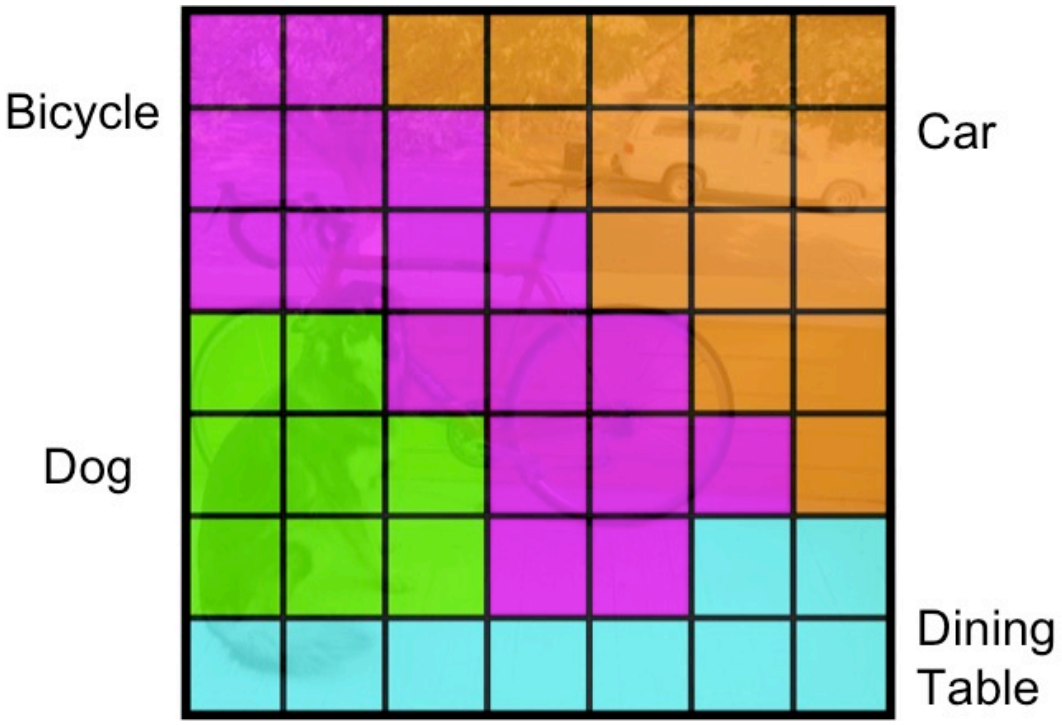


# Each cell predicts boxes and confidences: P(Object)

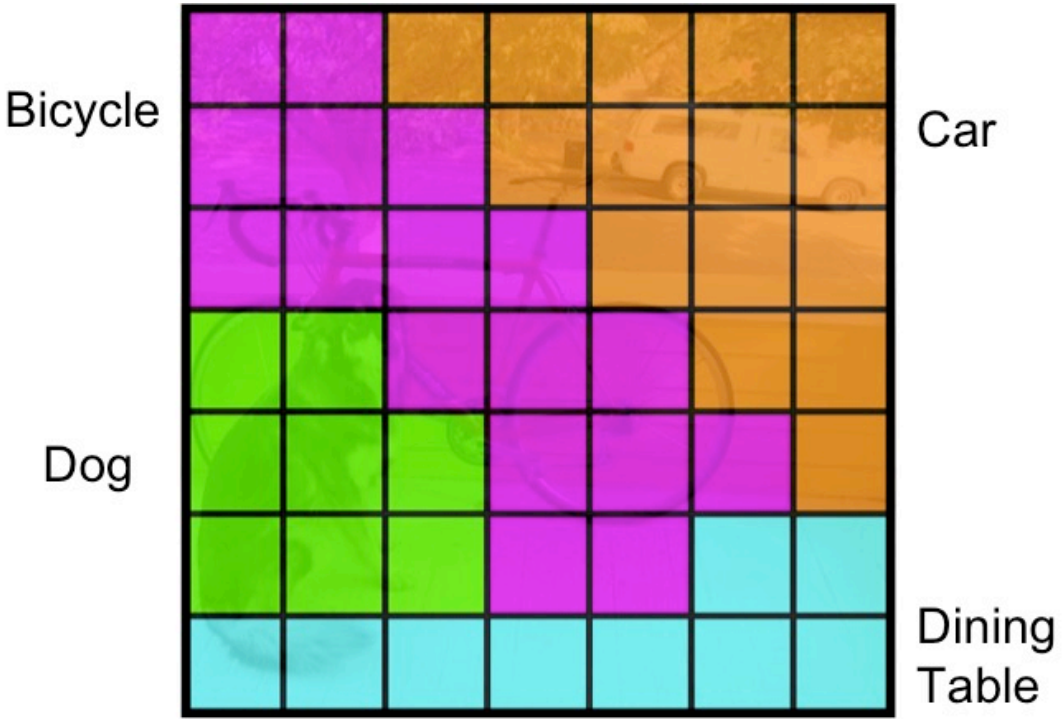
# Each cell also predicts a class probability



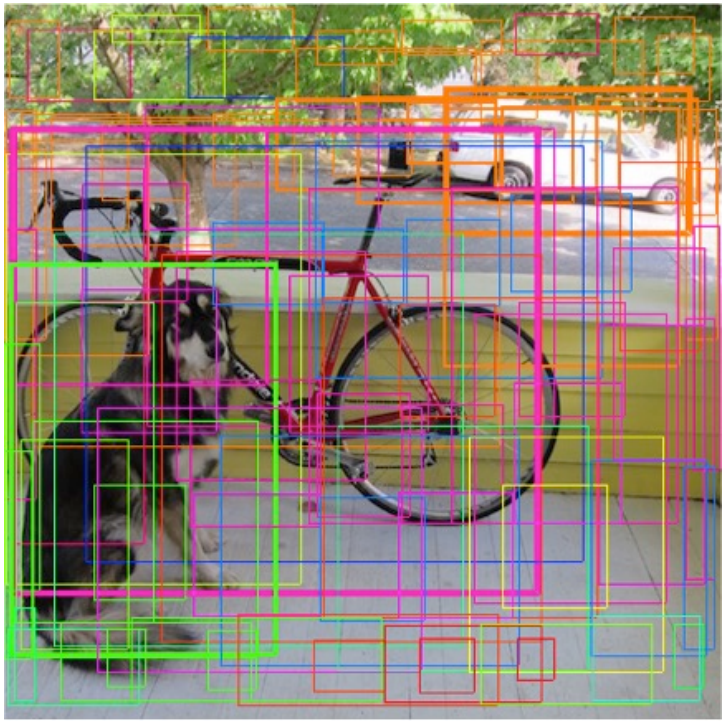
# Each cell also predicts a class probability



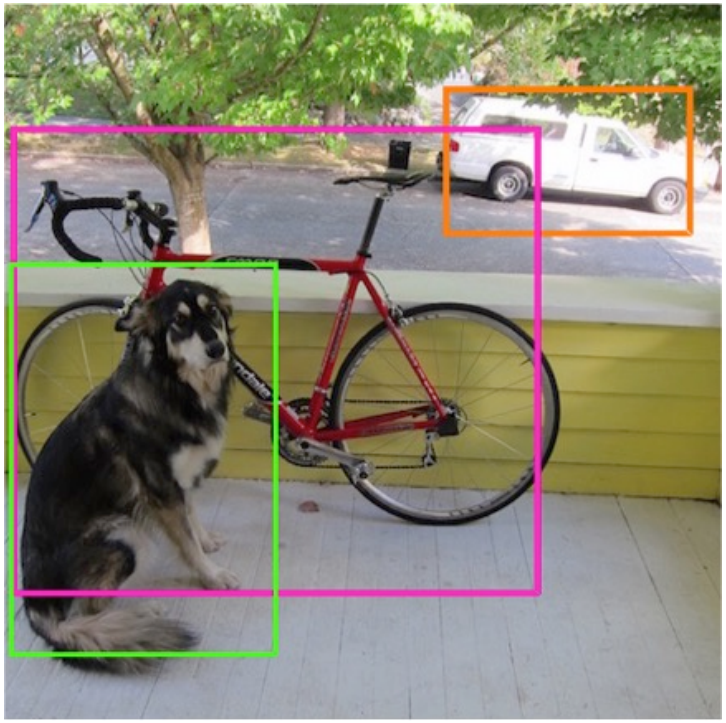
# Conditioned on object: P(Car | Object)



# Combine the box and class prediction

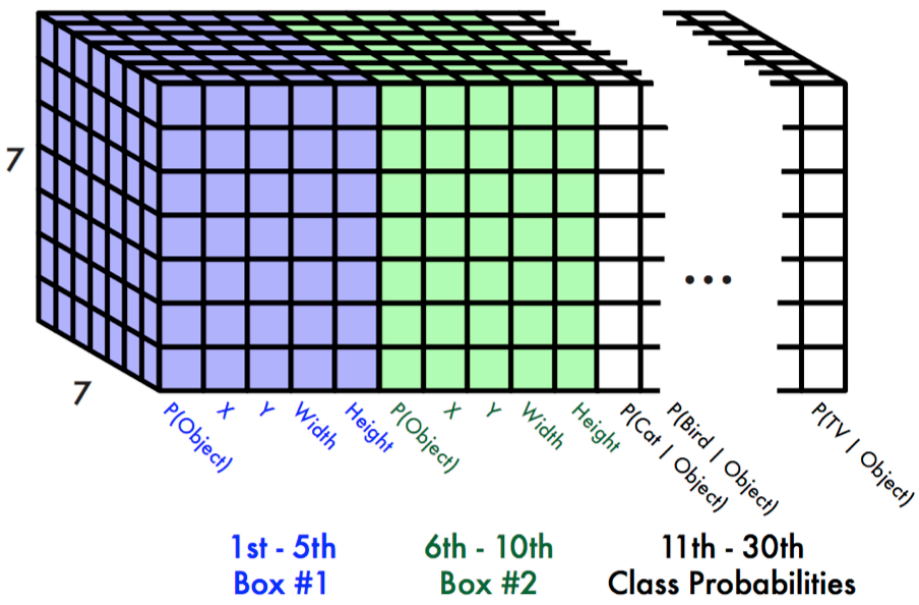


# Non-Maximum Suppression (NMS) and threshold detections



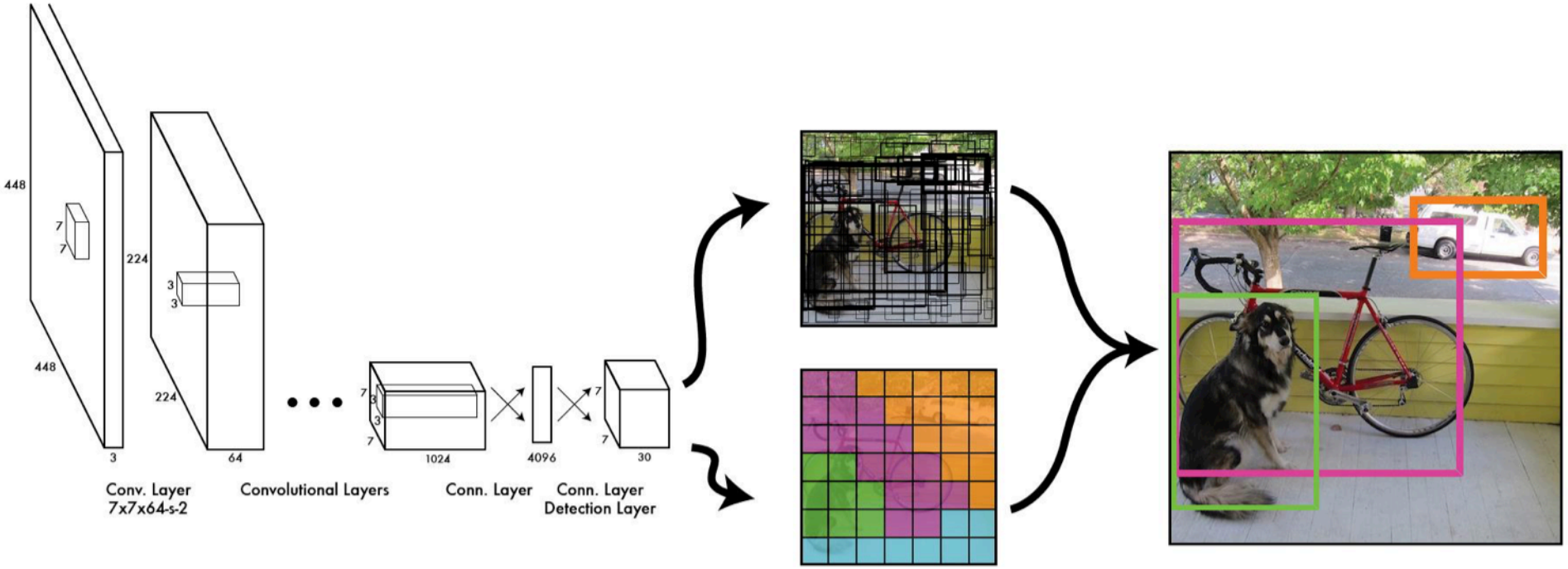
# This parameterization fixes the output size

- Each cell predicts:
  - For each bounding box:
    - 1 confidence value
    - 4 coordinates (x, y, w, h)
  - Some number of class probabilities

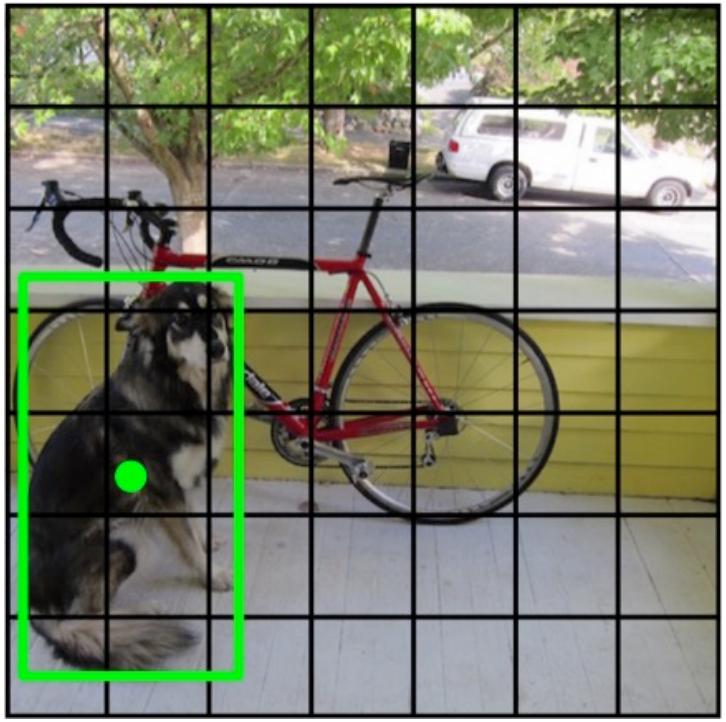



- For Pascal VOC:
  - 7×7 grid
  - 2 bounding boxes /cell
  - 20 classes
  - $7 \times 7 \times (2 \times 5 + 20) = \mathbf{1470\ outputs}$

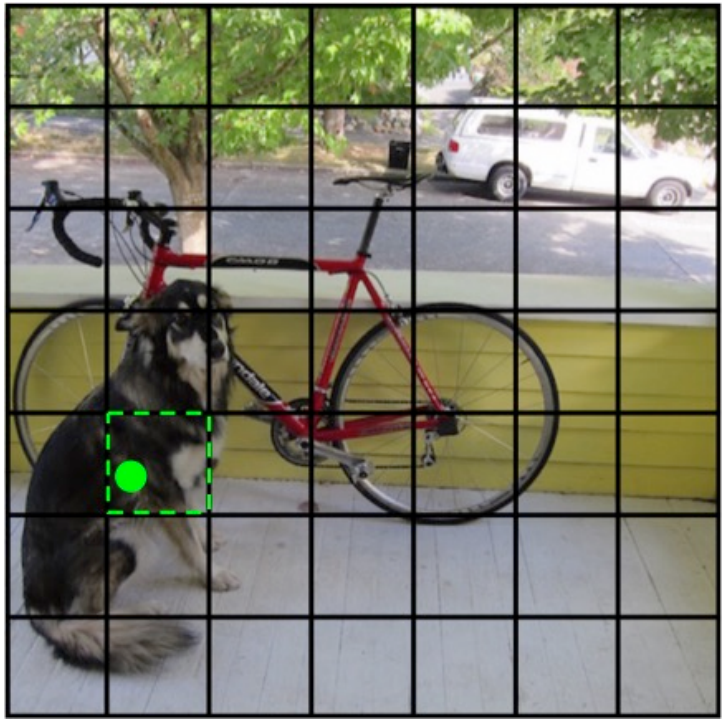
# Train one neural network to be a whole detection pipeline



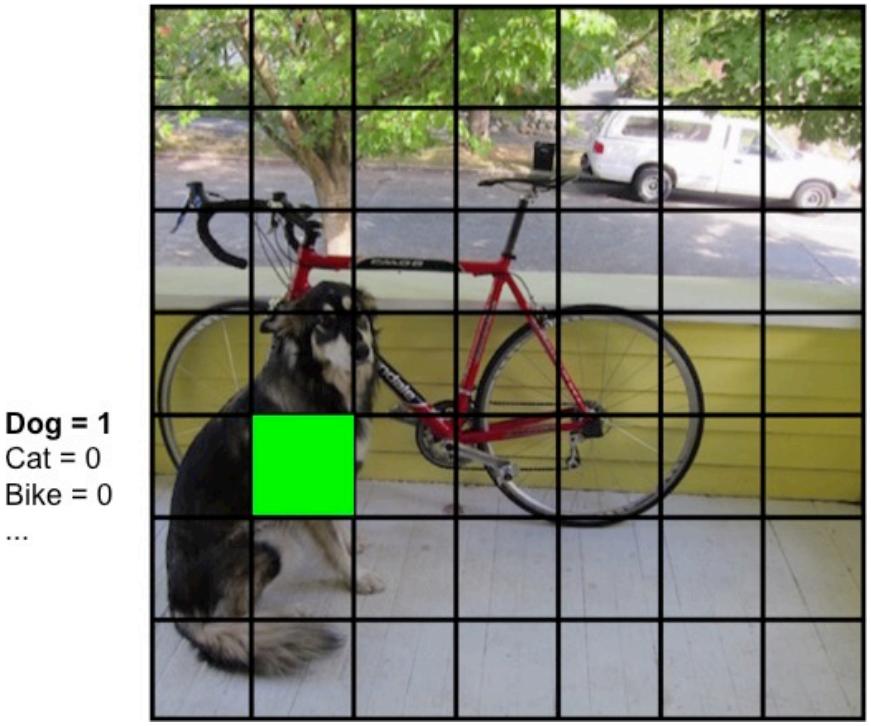
# During training, match example to the right cell



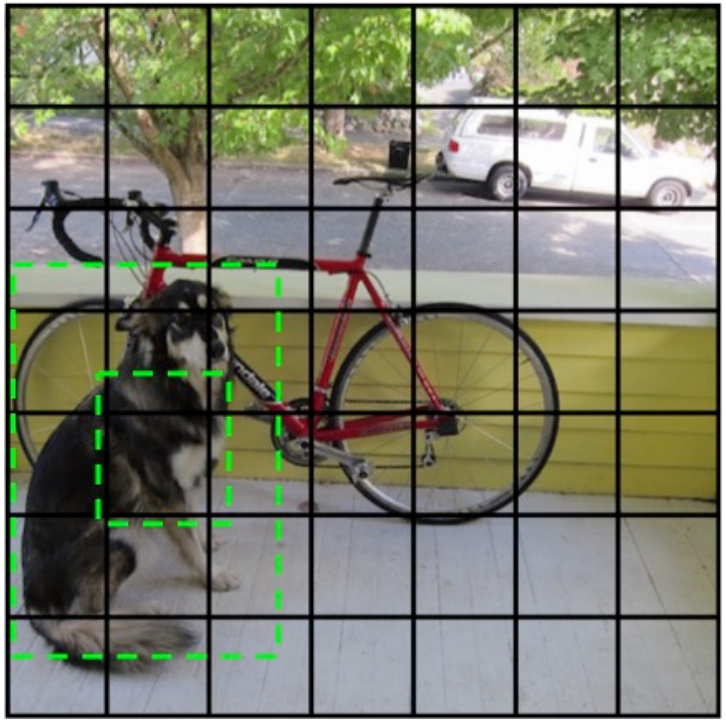
# During training, match example to the right cell



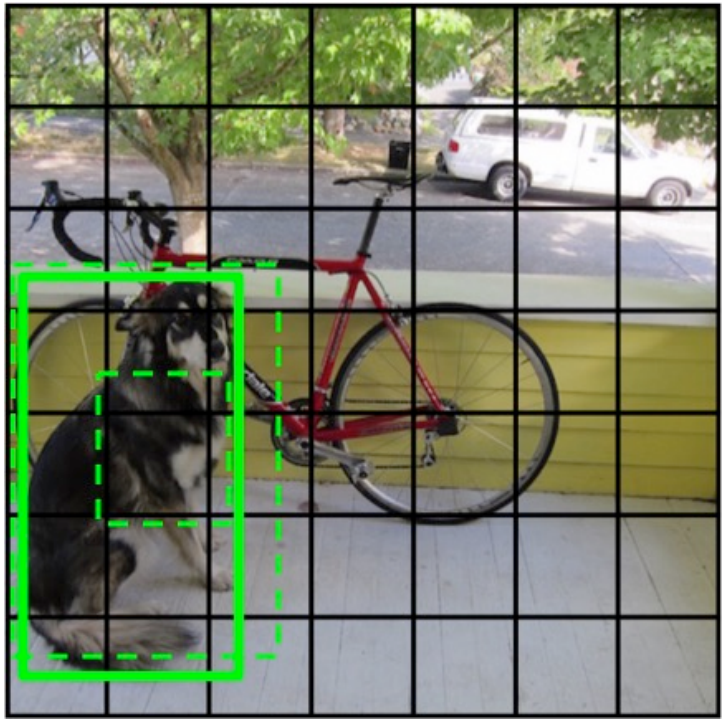
# Adjust that cell's class prediction



# Look at that cell's predicted boxes

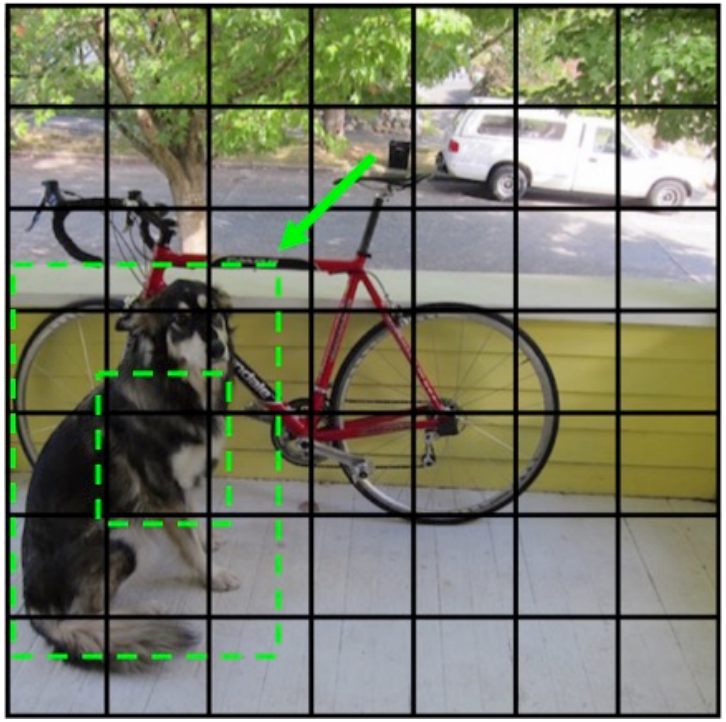


# Find the best one, adjust it, increase the confidence

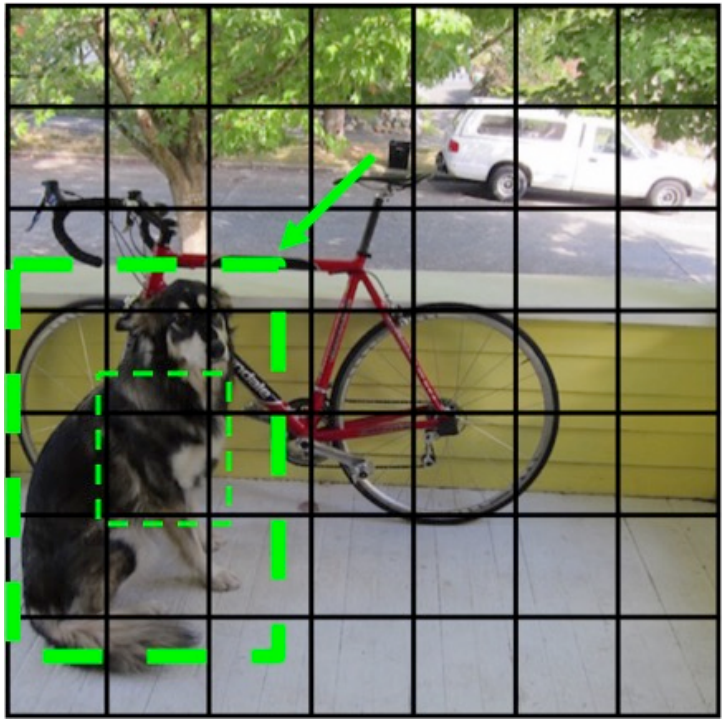


IoU

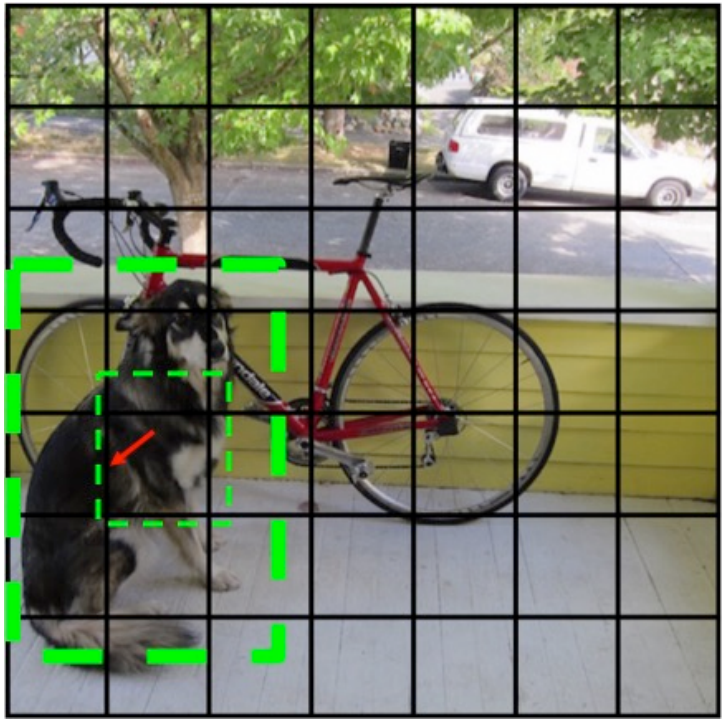
# Find the best one, adjust it, increase the confidence



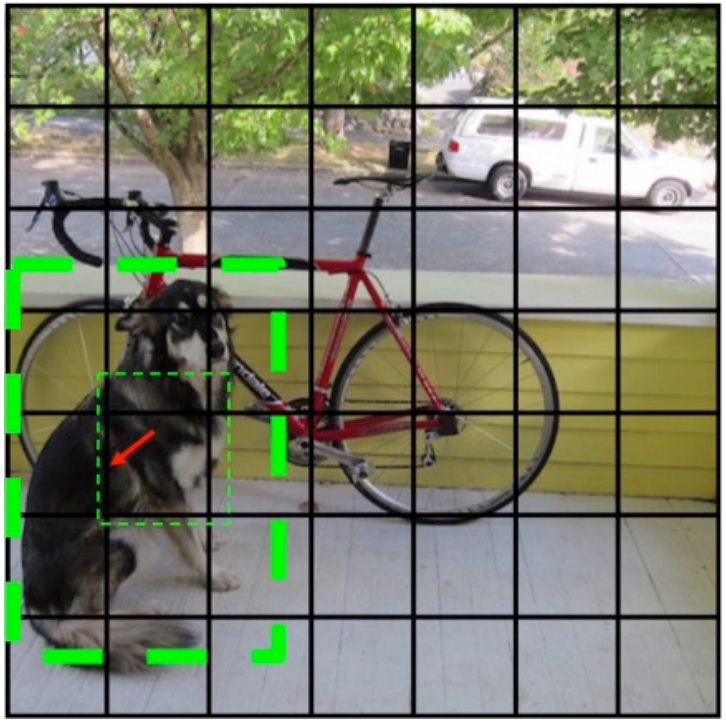
# Find the best one, adjust it, increase the confidence



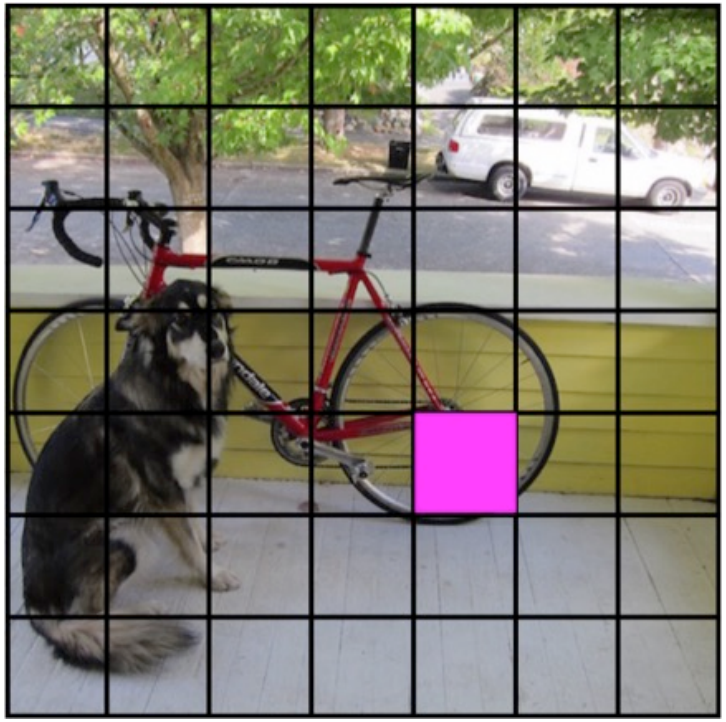
# Decrease the confidence of other boxes



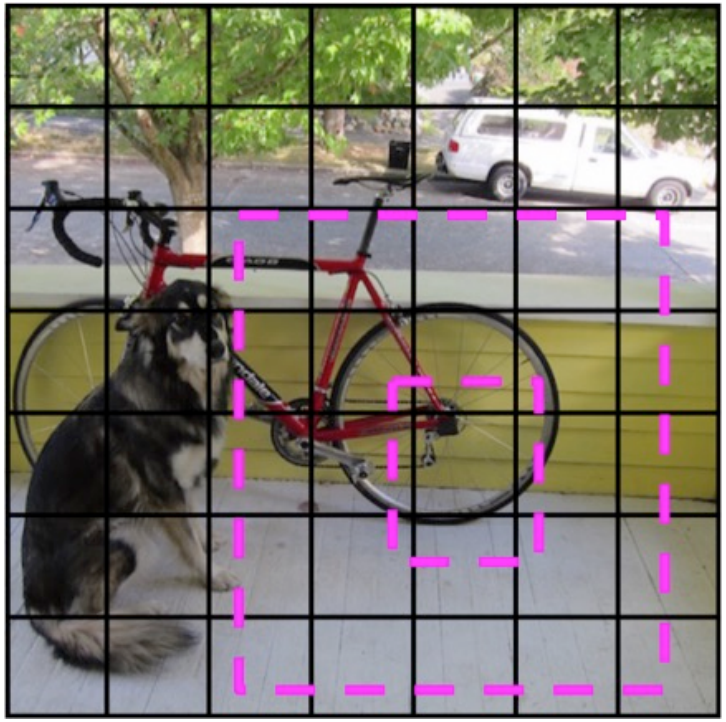
# Decrease the confidence of other boxes



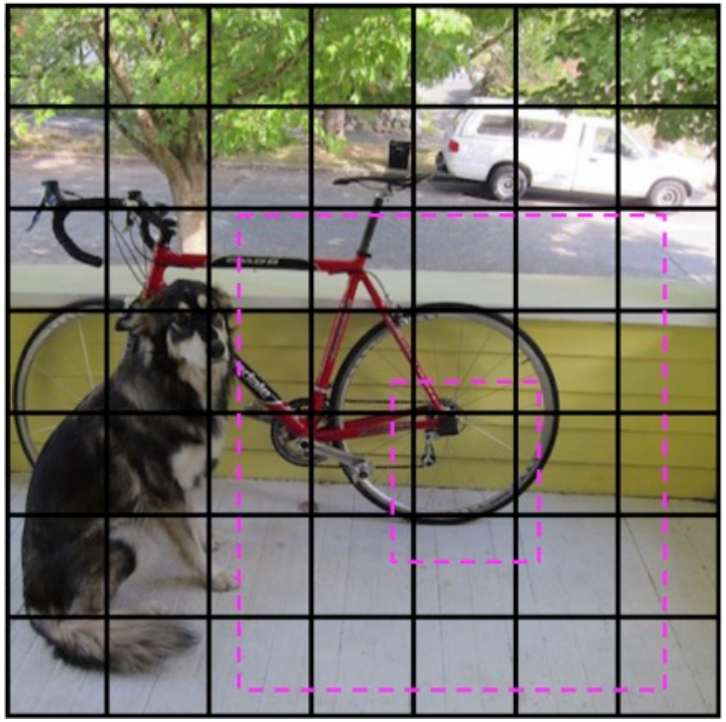
Some cells don’t have any ground truth detections!



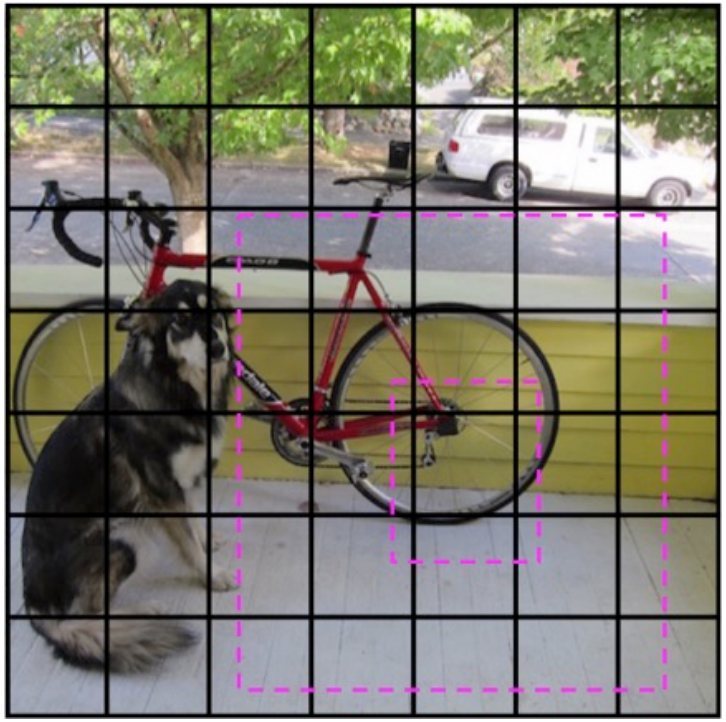
Some cells don’t have any ground truth detections!



# Decrease the confidence of these boxes



# Don’t adjust the class probabilities or coordinates



# Terms for loss function

- IOU: intersection over union between the predicted box and the ground truth box
- C: confidence
- P(Class|Object): if there is an object, what is its probability on that class
- S: number of grid
- K: number of classes
- Bounding box: (confidence, x, y, w, h)
- Output size = $S \times S \times (B \times 5 + K)$

- Training tricks:
  - Pretraining on ImageNet
  - SGD with decreasing learning rate
  - Extensive data augmentation

# Loss function

- 1st term: penalize location of center (x, y)
- 2nd term: penalize height and width predictions (w, h). Square root is used to penalize smaller bounding boxes more compared to larger bounding boxes
- 3rd term: if object is present, increase the confidence to IOU
- 4th term: if object is not present, decrease the confidence to 0
- 5th term: object detection loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

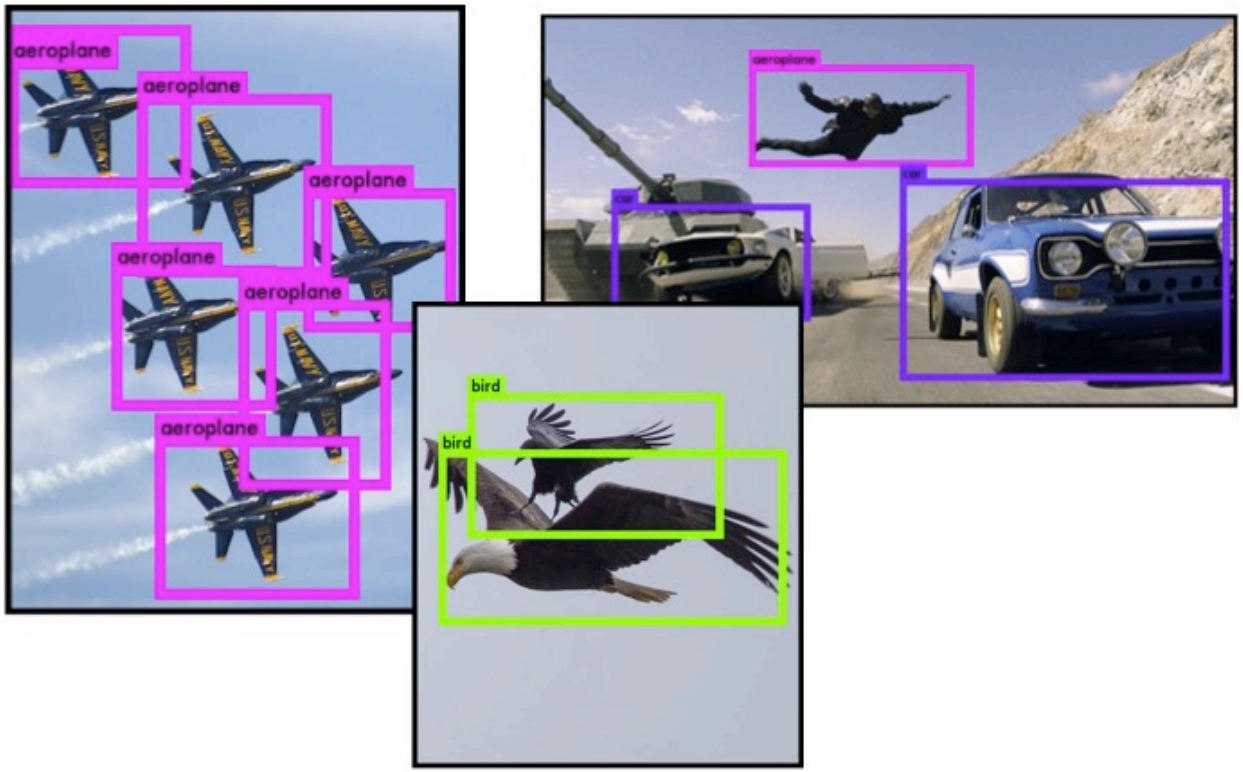
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_i \right)^2$$

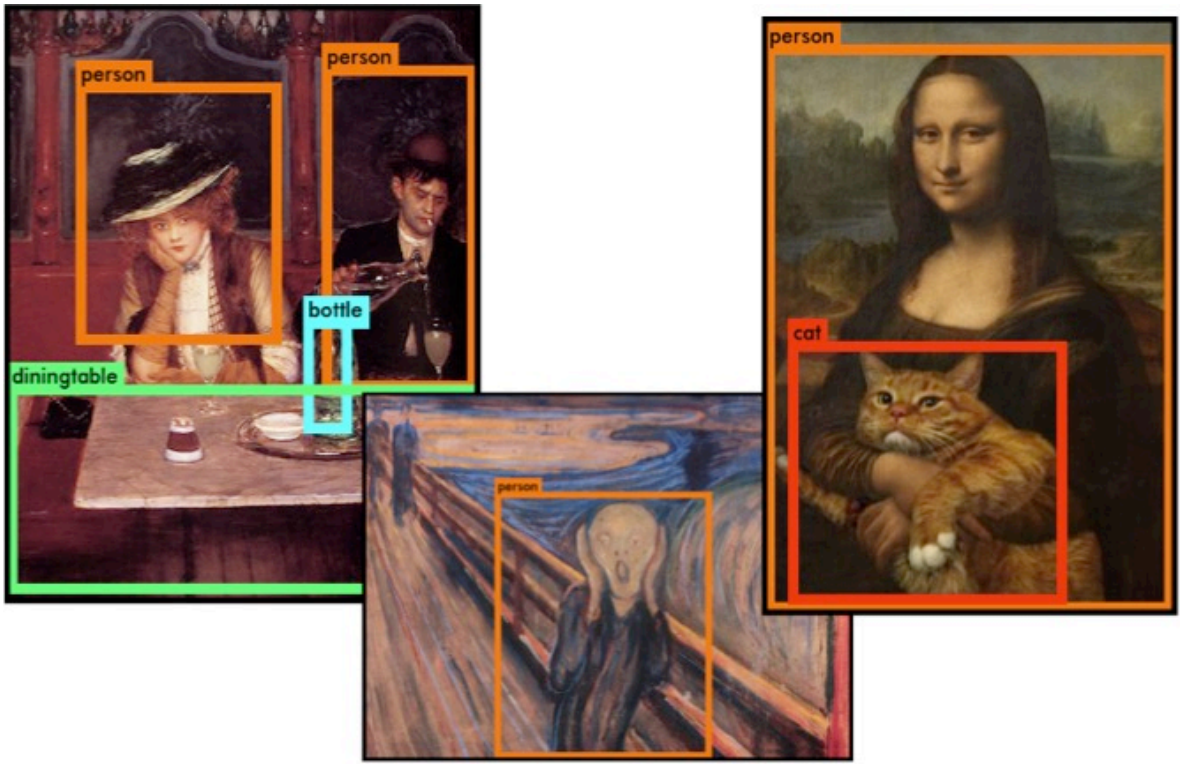
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

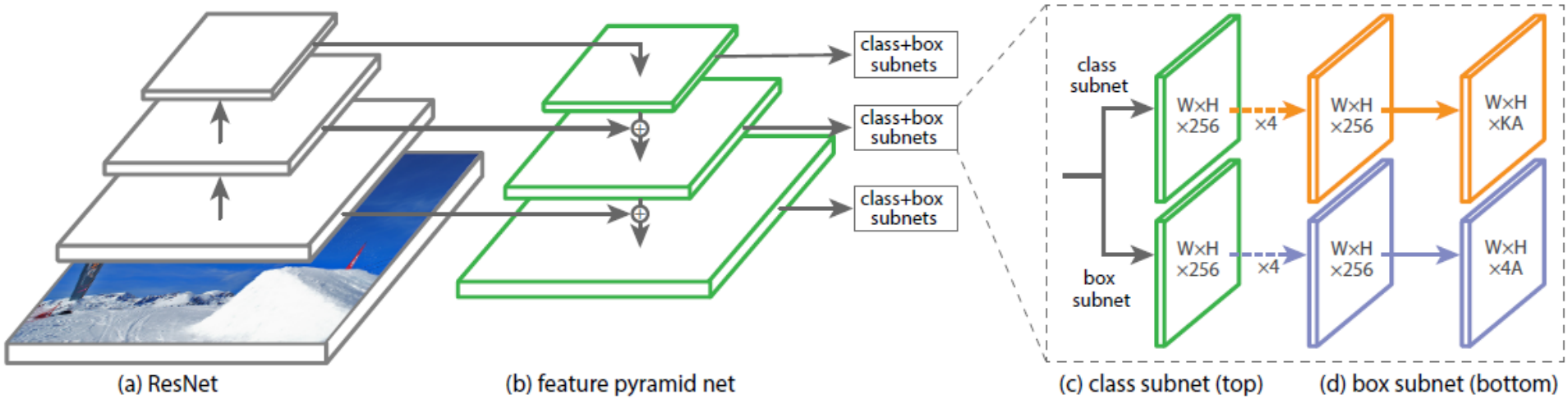
# YOLO works across a variety of natural images



# It also generalizes well to new domains (like art)



# A hybrid: RetinaNet




- Multi-scale feature pyramid instead of one-scale feature map
- Focal loss to address the limitation of extreme foreground-background class imbalance
- Outstanding performance and faster than regional based CNN

# Recurrent Neural Network (RNN)

# Introduction

- A family of neural networks for processing sequential data
- Can process sequences of variable length
- Share the same weights across several time steps
- Different from convolution across a 1-D temporal sequence, which is shallow
- Usually want to predict a vector at some time steps

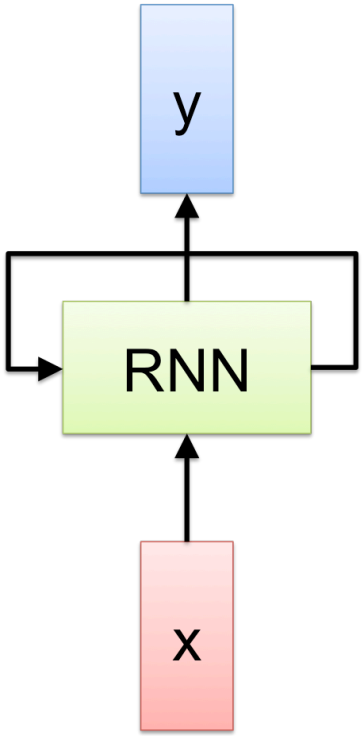
# Mathematical representation

- A recurrence formula at every time step:

$$h_t = f(h_{t-1}, x_t; W),$$

where $h$ is represented as the state, and $x_t$ is the input vector at some time step $t$ (Omit bias for simplicity)

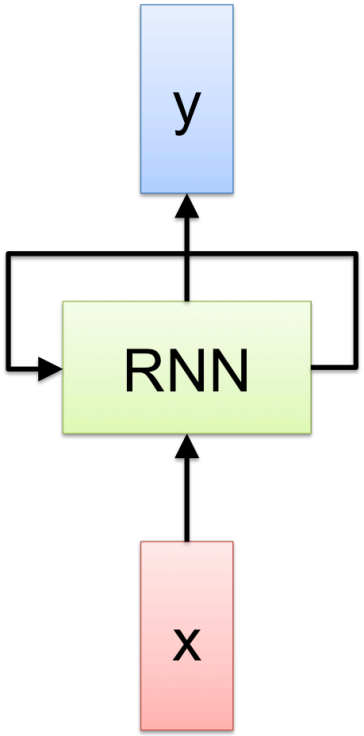
- Typical RNNs will add extra architectural features such as output layers that read information out of the state $h$ to make predictions



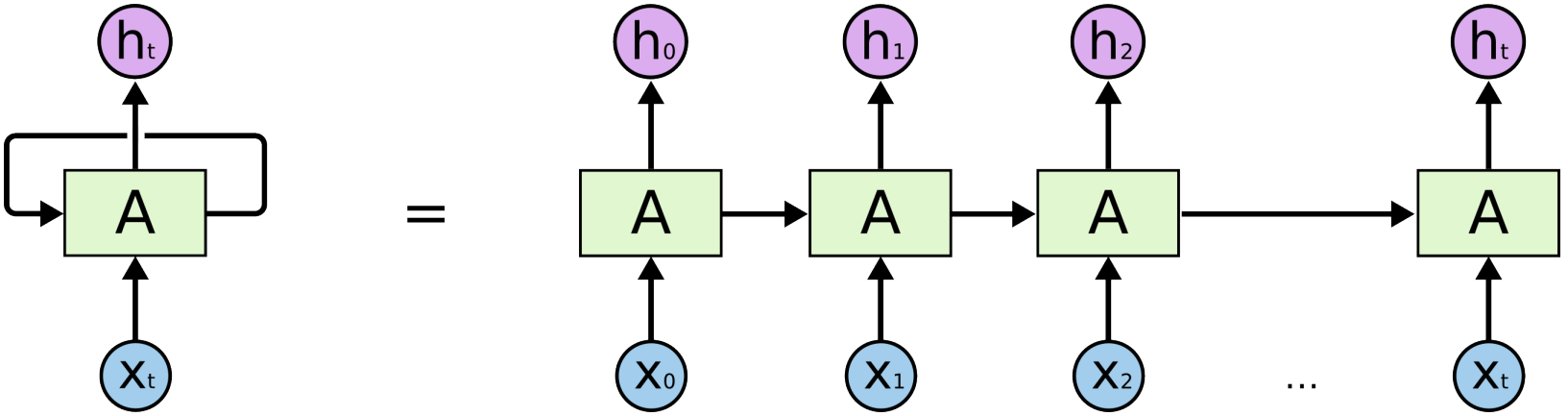
# (Vanilla) RNN

$$h_t = f(h_{t-1}, x_t; W)$$
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = \sigma(W_{hy} h_t)$$



# Unfolding Computational Graphs

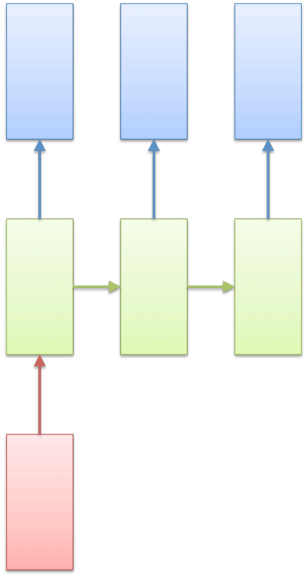


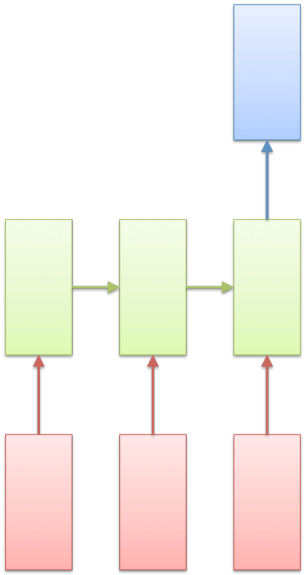

- Represent the unfolded recurrence after $t$ steps with a function $g_t$:

$$h_t = g_t(x_t, x_{t-1}, \cdots, x_1, x_0) = f(h_{t-1}, x_t; W)$$

- The unfolding process thus introduces two major advantages
  - Regardless of the sequence length, the learned model always has the same input size, because it is specified in terms of transition from one state to another state
  - Re-use the same weight matrix at every time-step

# RNN variants

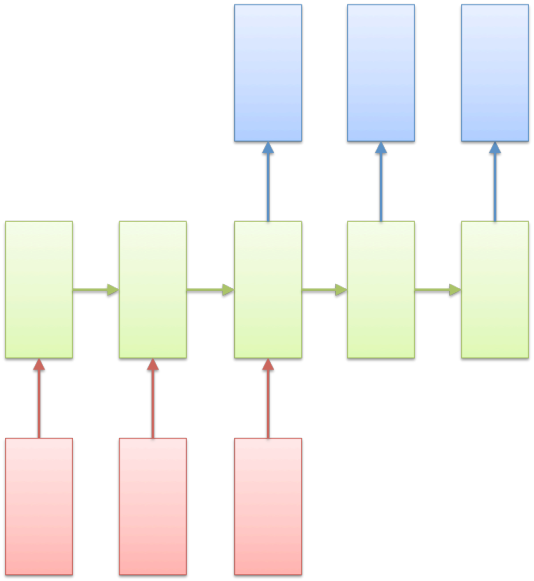




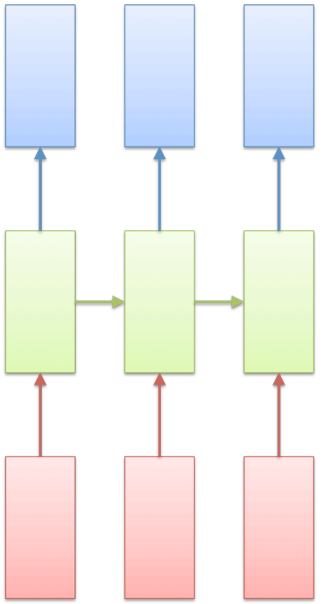
- One to many
- Image captioning: image → sequence of words

- Many to one
- Sentiment classification: sequence of words → sentiment

# RNN variants

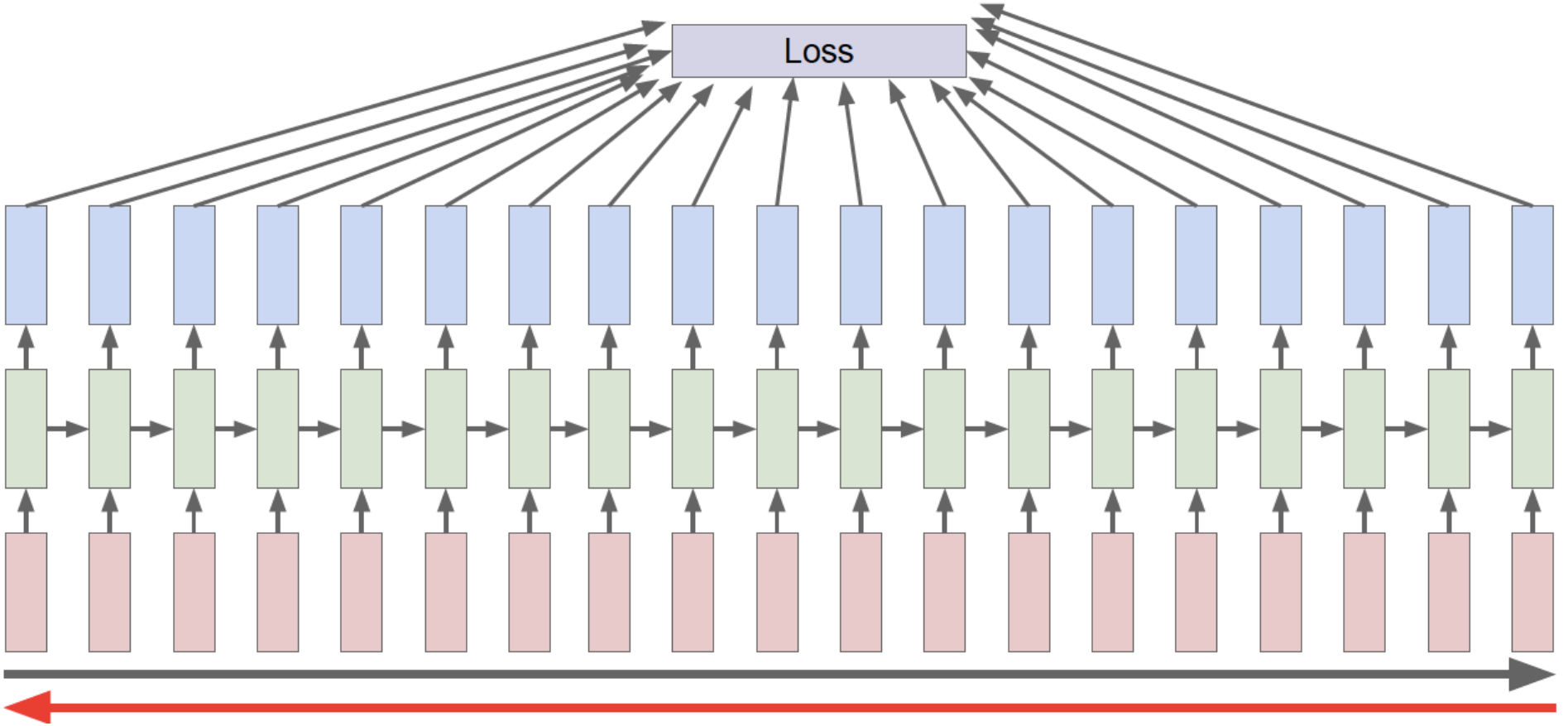


- Many to many
- Machine translation: sequence of words → sequence of words



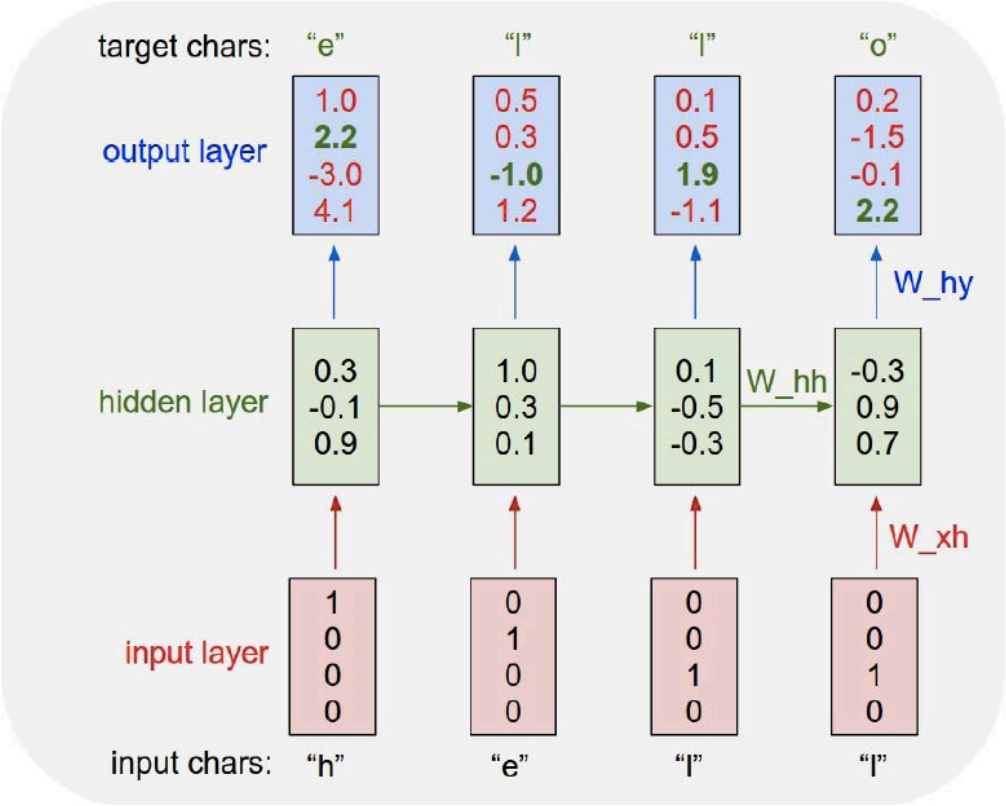
- Many to many
- Video classification on frame level

# Propagation of RNN




- Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

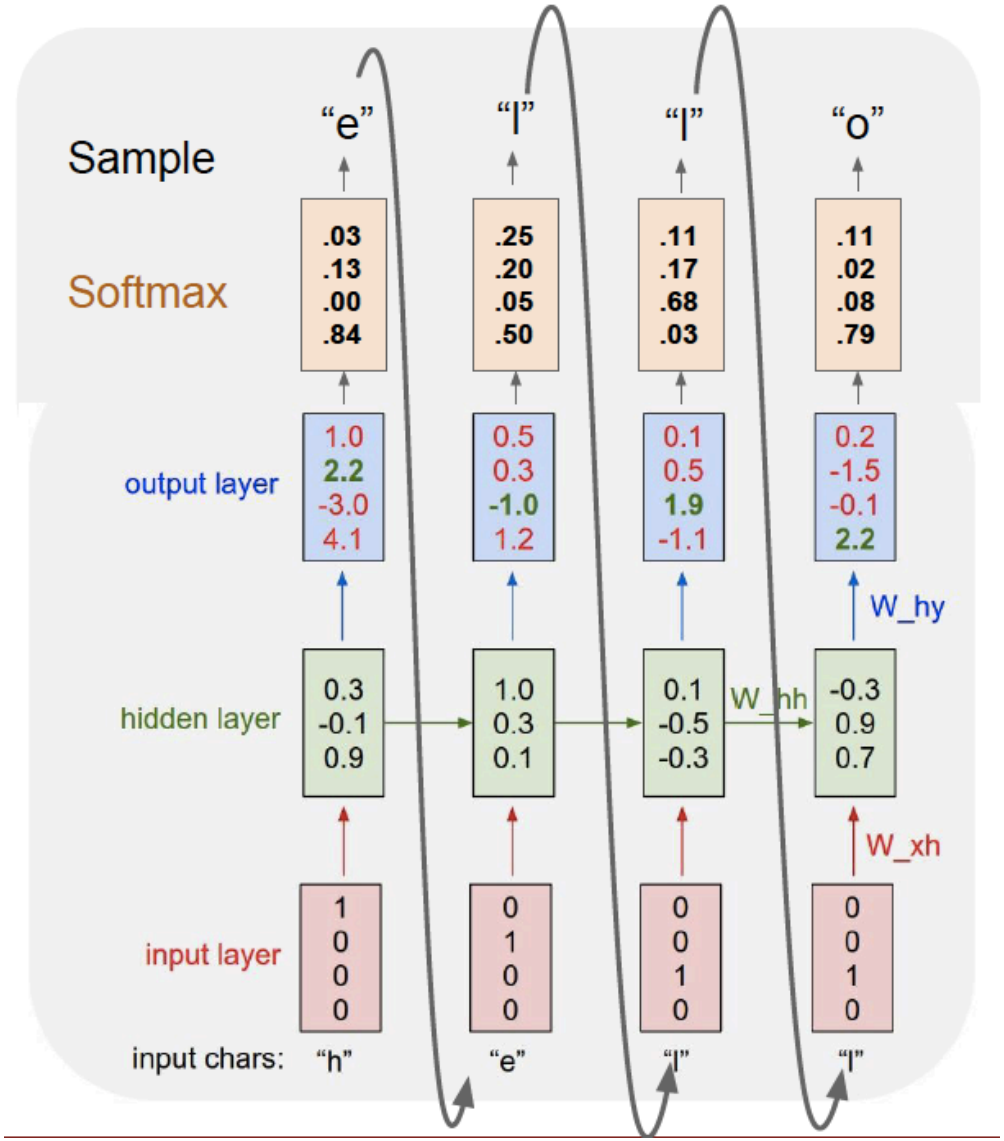
# Example: character-level language model




- Example training sequence: "hello"

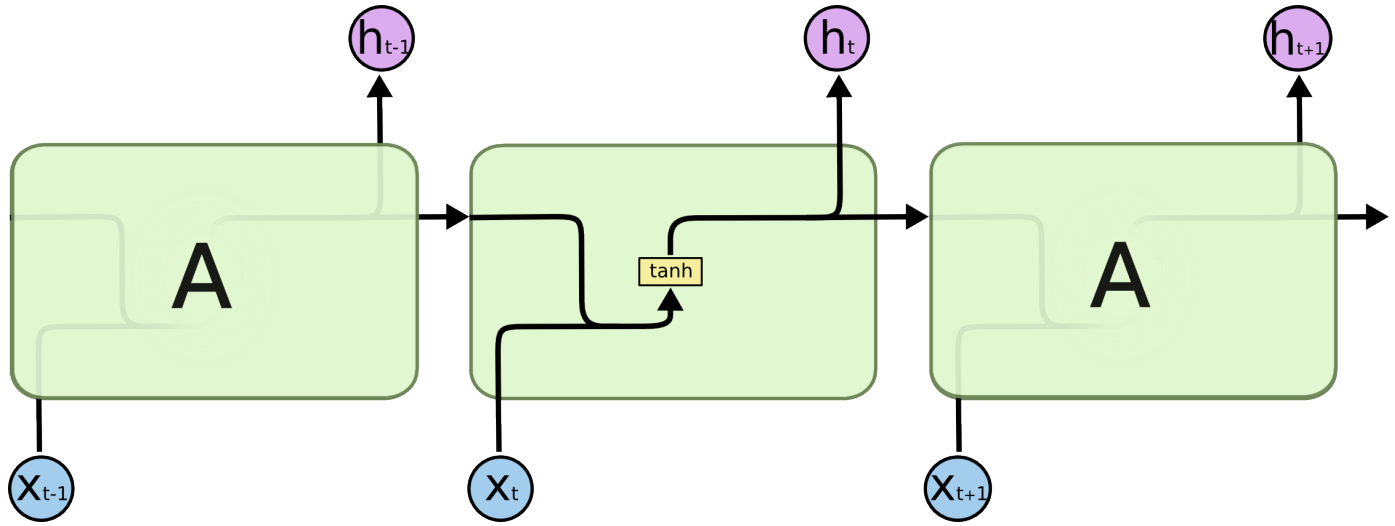
# Example: character-level language model

- At test time, sample characters one at a time, feed back to model
- What kind of RNN is it?

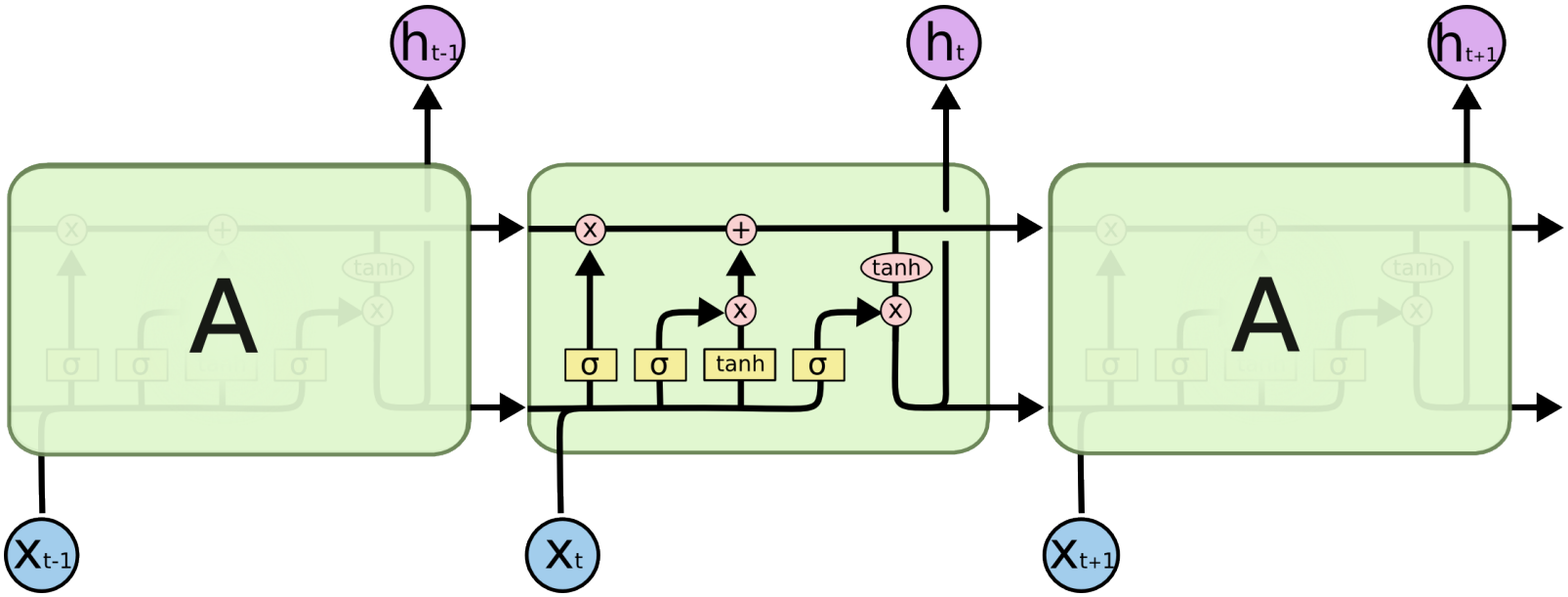


# LSTM – motivation

- Long short-term memory (LSTM)
- Problem of vanilla RNN: backward flow of gradients can explode or vanish

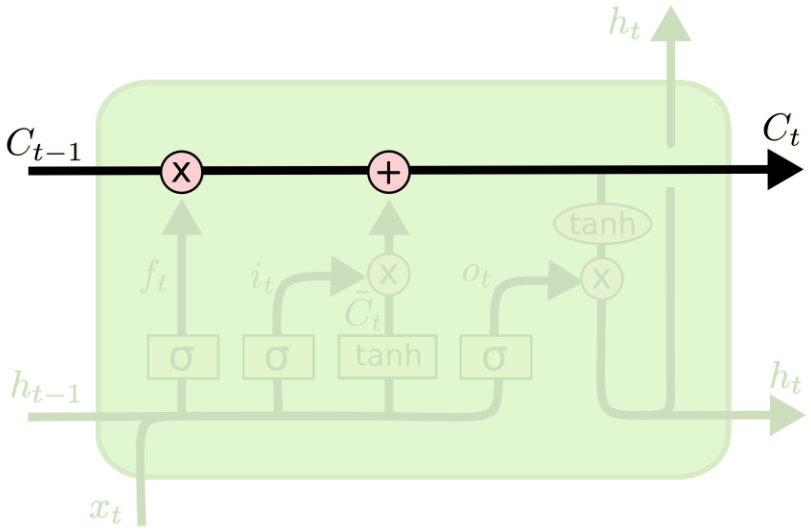


# LSTM overview



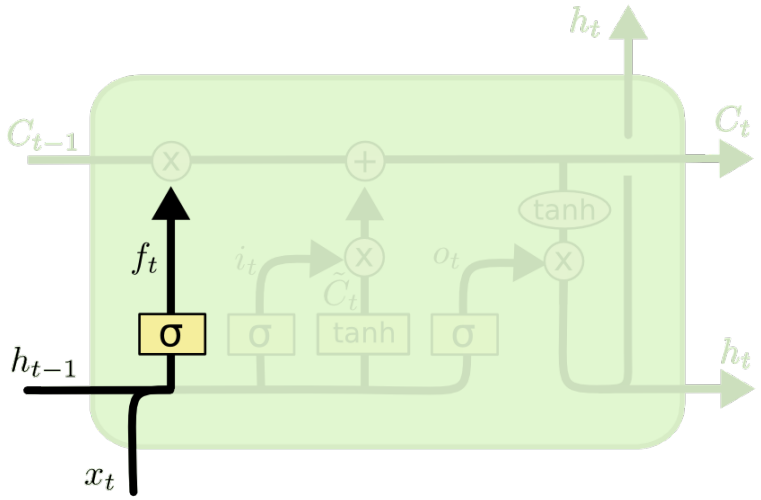
# LSTM: Memory cell

- "Memory" structure of the LSTM
- Controlled / modified by two gates
- Workflow: forget → update → predict



# LSTM: Forget gate

- $f_t = \sigma(W_f[h_{t-1}, x_t])$
- 0 = "forget all", 1 = "remember all"



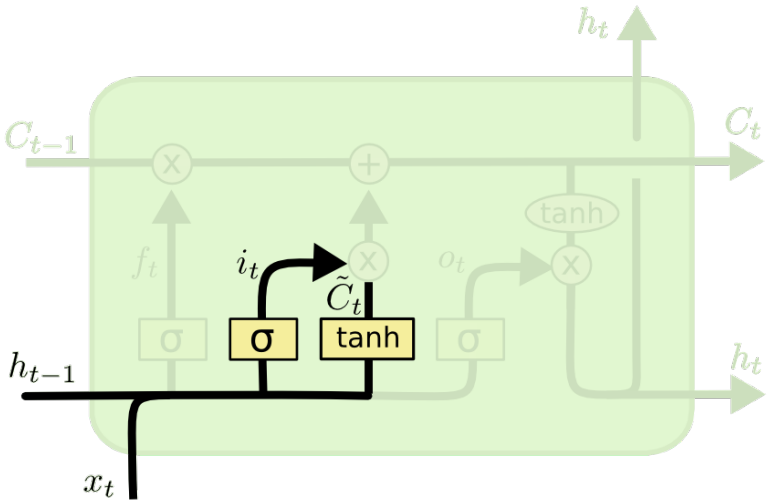
# LSTM: Decide what to learn

- Input gate: the degree to which the new memory content is added to the memory cell

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$

- The new memory content is:

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t])$$

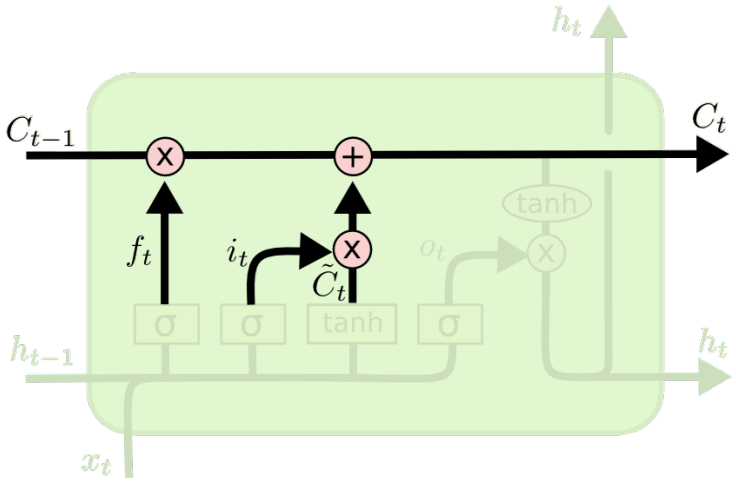


# LSTM: Update memory cell

- Partially forget the existing memory and add a new memory content

$$C_t = C_{t-1} * f_t + i_t * \tilde{C}_t$$

where $*$ represents the Hadamard product



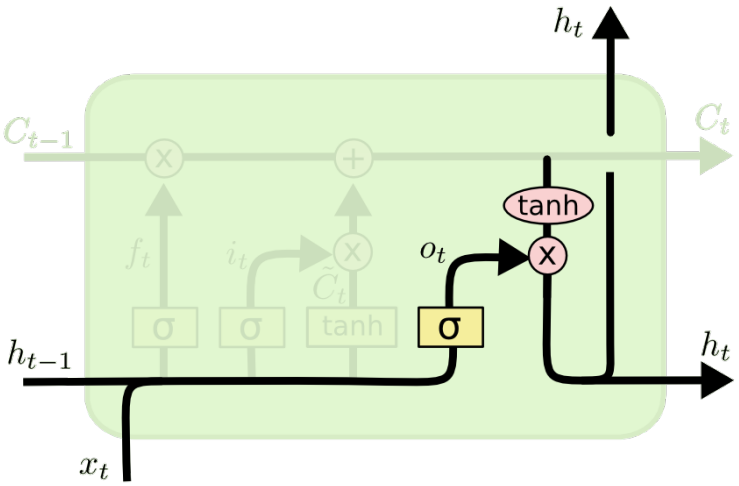
# LSTM: Predict new state

- Output gate:

$$o_t = \sigma(W_o[h_{t-1}, x_t])$$

- The state, or the activation of LSTM is:

$$h_t = o_t * \tanh(C_t)$$

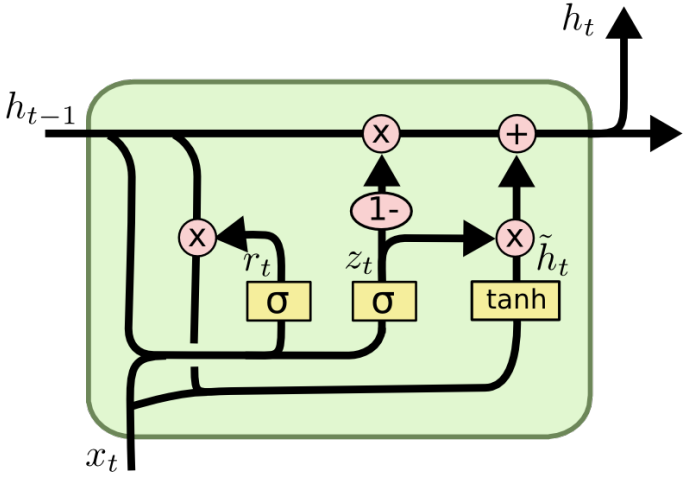


# Gated Recurrent Unit (GRU)

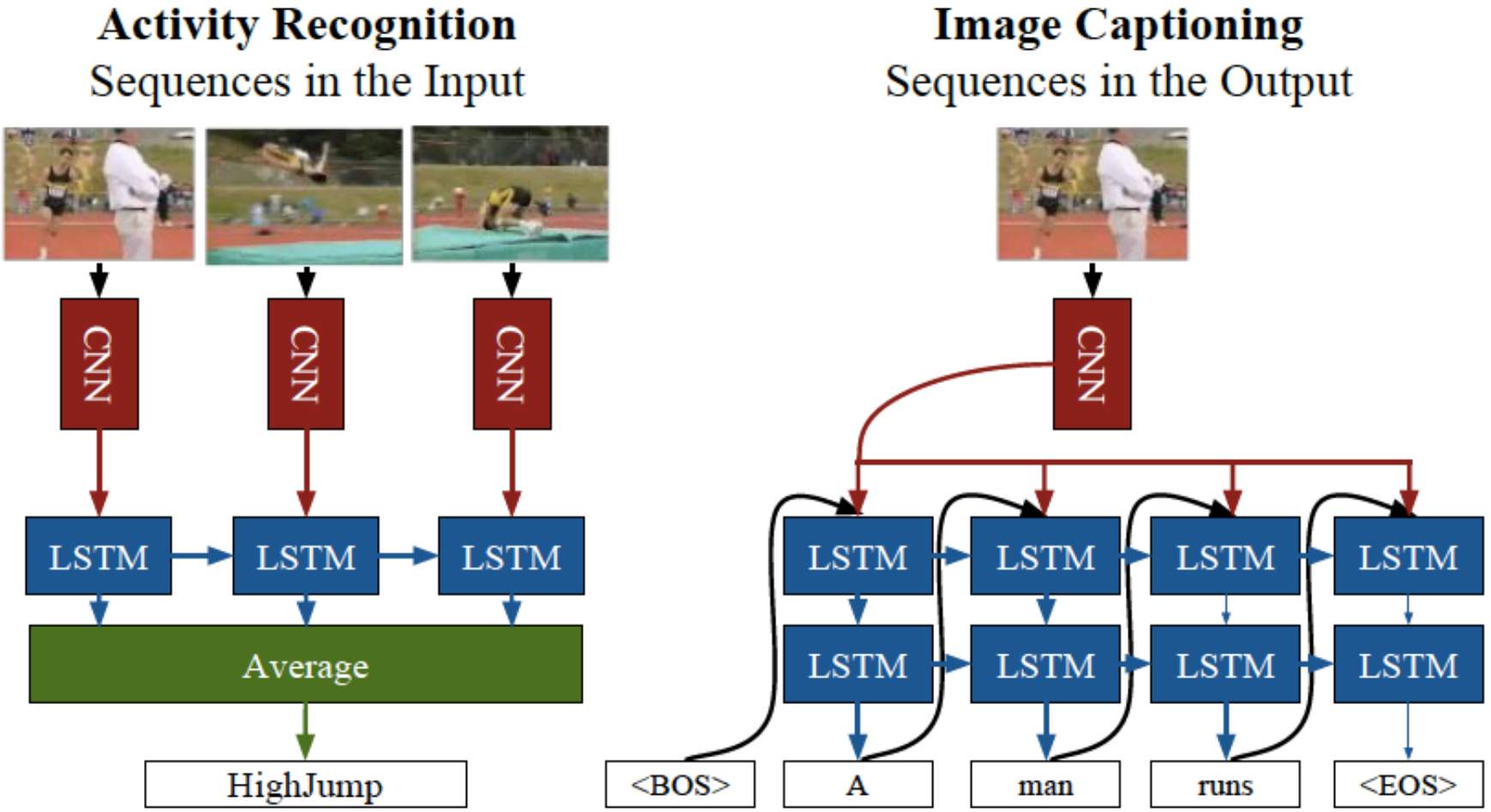
- Compute as:

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$
$$r_t = \sigma(W_r[h_{t-1}, x_t])$$
$$\tilde{h}_t = \tanh(W[r_t * h_{t-1}, x_t])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Combine the forget and input gates into a single "update gate"



# Combine CNNs with RNNs



# More advanced: Transformer

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In Advances in neural information processing systems 2017 (pp. 5998-6008).