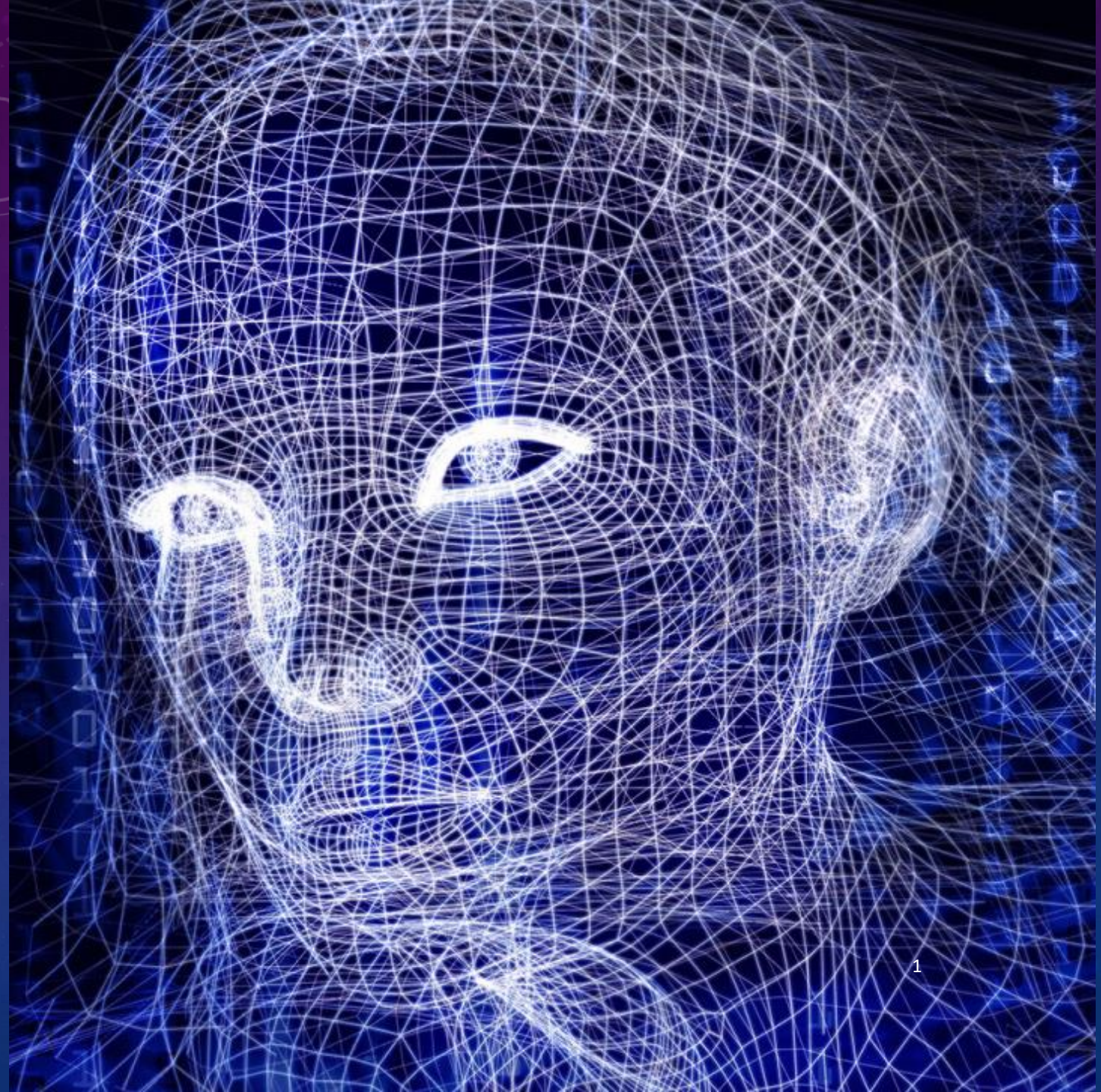
LECTURE SERIES FOR DIGITAL SURVEILLANCE SYSTEMS AND APPLICATION

INTRODUCTION TO DEEP LEARNING

徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of Science and Technology



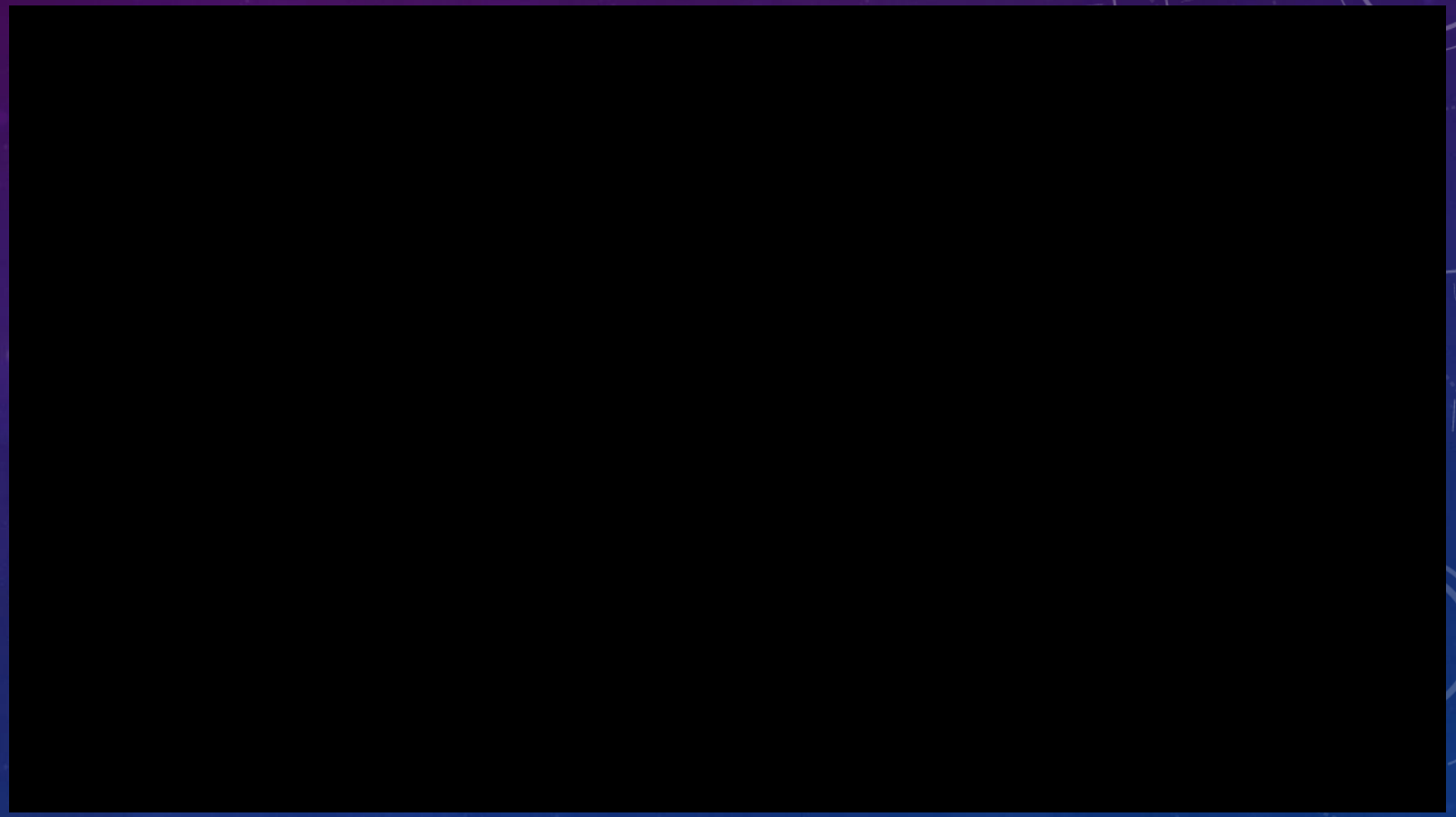
Convolutional Neural Networks

1) Activation Functions
2) Backpropagation
3) Dropout
4) Preprocessing Data
5) Weight Initialization
6) Batch Normalization
7) Babysitting the learning Process
8) Hyperparameter Optimization

Stanford lecture

Content:

2:00 CNN 4:50Activation Functions
25:19 Maxout
27:22 Preprocessing data
33:35 Weight Initialization
48:56 Batch Normalization
1:04:42 Babysitting the learning Process
1:10:22 Hyperparameter Optimization



https://www.youtube.com/watch?v=wEoyxE0GP2M&t=4s

Backpropagation – What is it really doing? – by BlueBrown

https://www.youtube.com/watch?v=Ilg3gGewQ5U&t=1s [13:53]

Backpropagation – Chain Rule – by BlueBrown

https://www.youtube.com/watch?v=tIeHLnjs5U8 [10:00]

Backpropagation

DEFINITION

1) **Calculate the forward phase** for each input-output pair $(\vec{x_d}, y_d)$ and store the results $\hat{y_d}$, a_j^k, and o_j^k for each node j in layer k by proceeding from layer 0, the input layer, to layer m, the output layer.

2) **Calculate the backward phase** for each input-output pair $(\vec{x_d}, y_d)$ and store the results $\frac{\partial E_d}{\partial w_{ij}^k}$ for each weight w_{ij}^k connecting node i in layer $k-1$ to node j in layer k by proceeding from layer m, the output layer, to layer 1, the input layer.

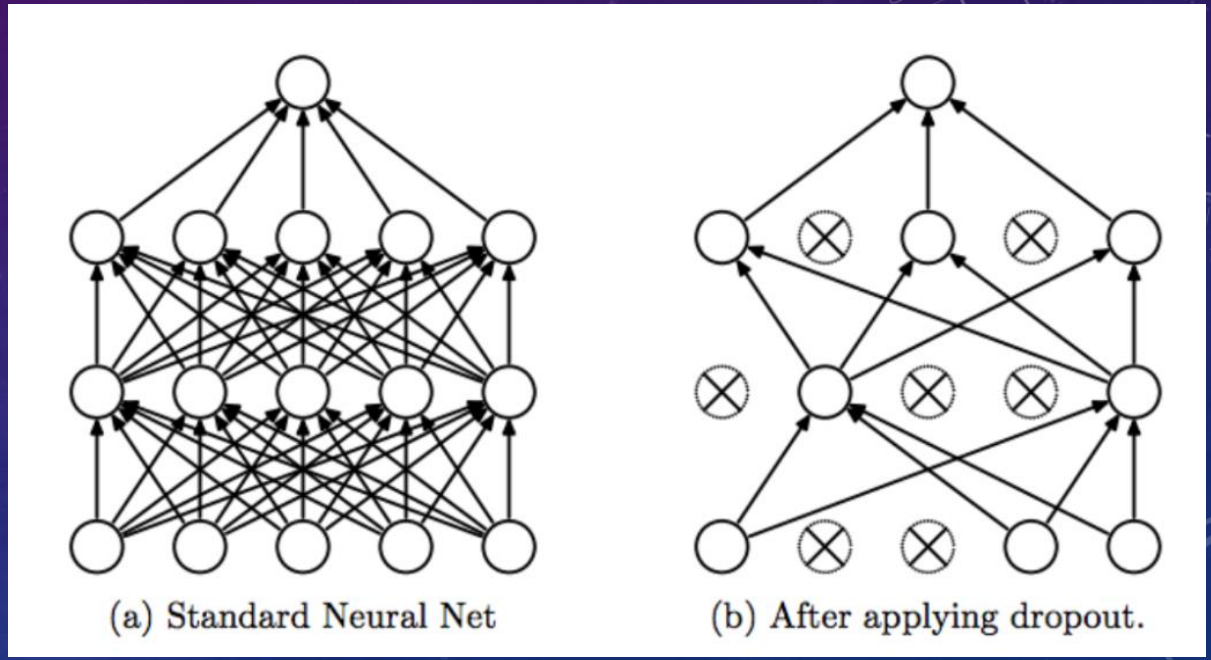
a) Evaluate the error term for the final layer δ_1^m by using the second equation.
b) Backpropagate the error terms for the hidden layers δ_j^k, working backwards from the final hidden layer $k = m - 1$, by repeatedly using the third equation.
c) Evaluate the partial derivatives of the individual error E_d with respect to w_{ij}^k by using the first equation.

3) **Combine the individual gradients** for each input-output pair $\frac{\partial E_d}{\partial w_{ij}^k}$ to get the total gradient $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ for the entire set of input-output pairs $X = \{(\vec{x_1}, y_1), \ldots, (\vec{x_N}, y_N)\}$ by using the fourth equation (a simple average of the individual gradients).

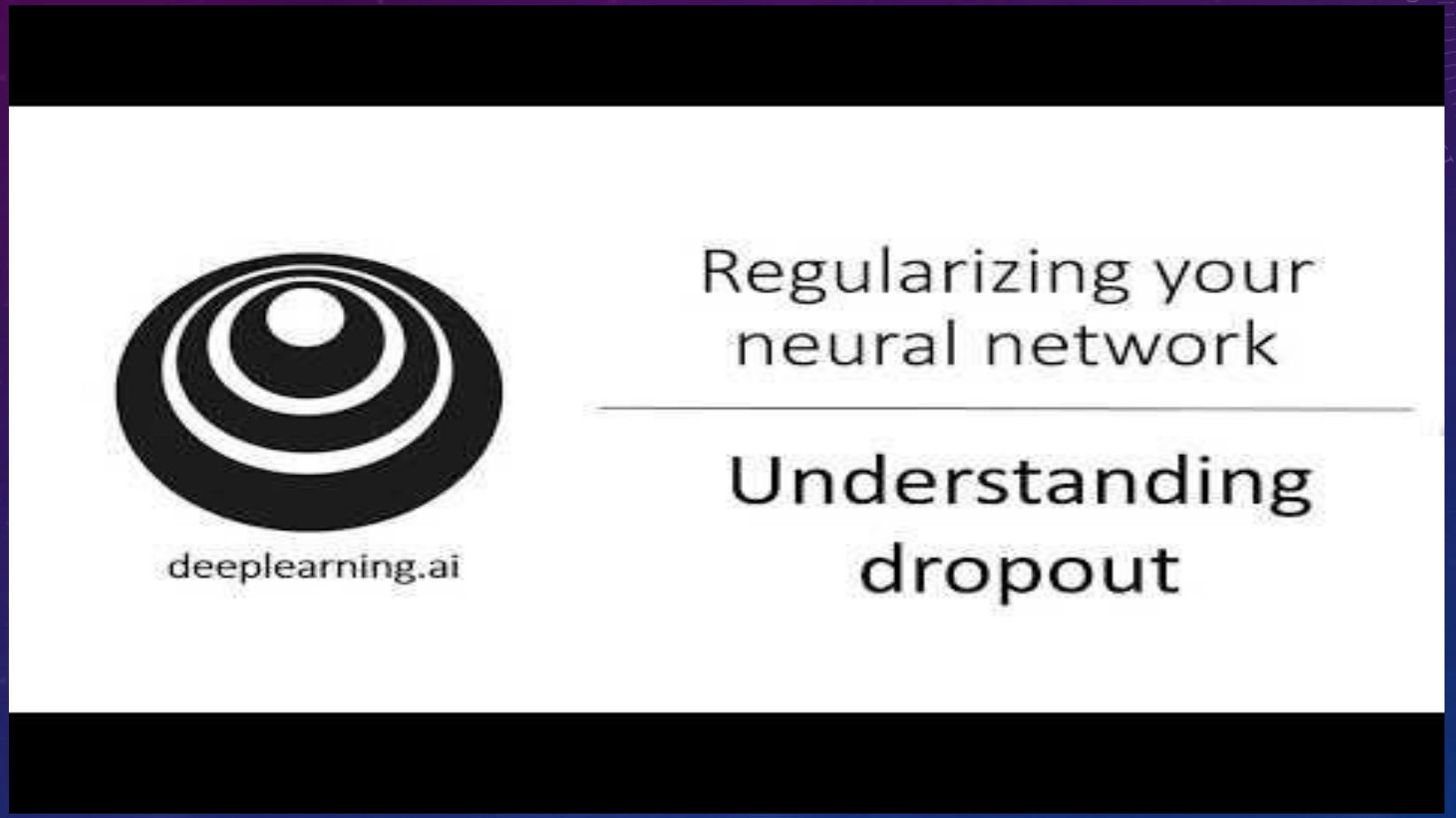
4) **Update the weights** according to the learning rate α and total gradient $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ by using the fifth equation (moving in the direction of the negative gradient).

Dropout

- Dropout is a term which is used deep learning and is a common technique used for training and enhancing the performance of Networks. We simply drop out units during training and avoid becoming too dependent on the feedback of single neurons.
- Regularization method



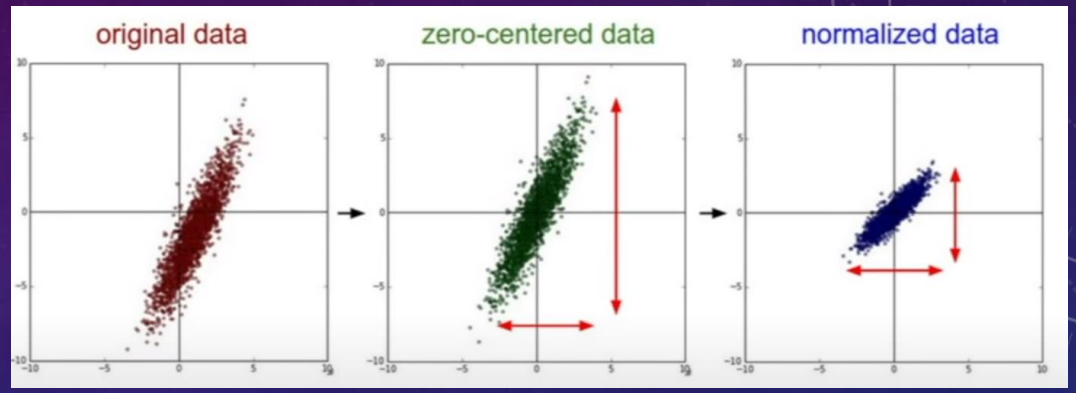
Dropout explained by Ng



https://www.youtube.com/watch?v=ARq74QuavAo&t=142s

Other Regularization methods are Preprocessing

- Consider the original data in red on the right as our input to any layer of our neural network. To make our network more robust to any kind of input data, we want to generalize this input data by first zero-centering and then normalizing the data.
- **Zero-centering:** subtract the mean
- **Normalizing:** divide by standard deviation



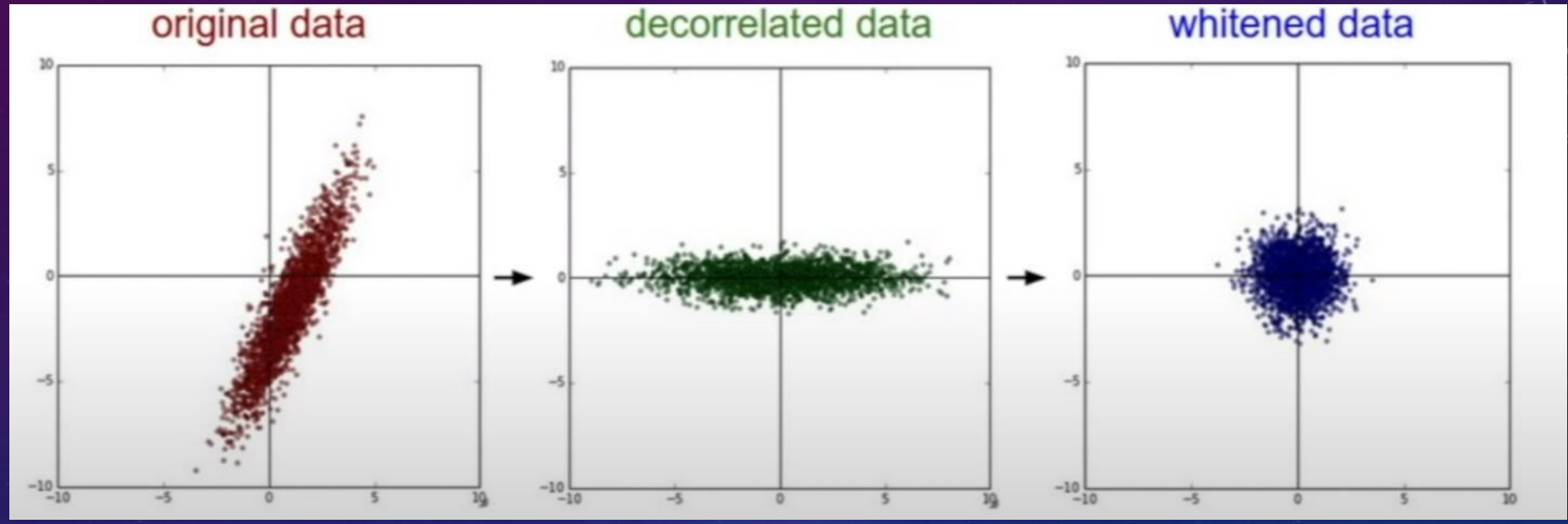

```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

Data Preprocessing for ML

https://www.youtube.com/watch?v=NBm4etNMT5k [3:30]

Other forms of regularization might be PCA and Whitening



Data has diagonal covariance matrix

Covariance matrix is identity matrix

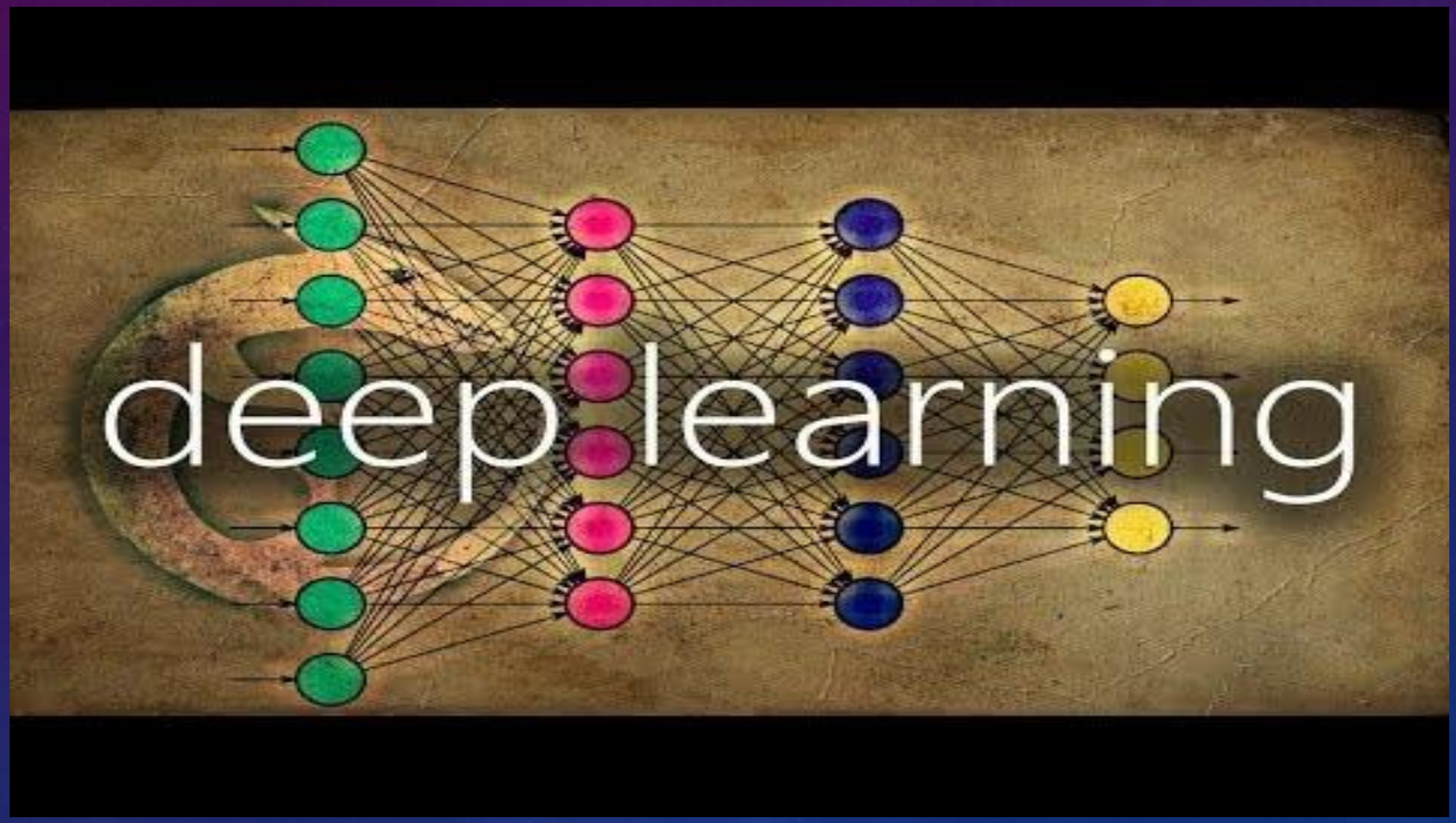
What is a covariance Matrix?

- *In probability theory and statistics, a **covariance matrix** (also known as **auto-covariance matrix**, **dispersion matrix**, **variance matrix**, or **variance–covariance matrix**) is a square matrix giving the covariance between each pair of elements of a given random vector. In the matrix diagonal there are variances, i.e., the covariance of each element with itself.*

$$\mathrm{K_{XX}} = \begin{bmatrix} \mathrm{E}[(X_1 - \mathrm{E}[X_1])(X_1 - \mathrm{E}[X_1])] & \mathrm{E}[(X_1 - \mathrm{E}[X_1])(X_2 - \mathrm{E}[X_2])] & \cdots & \mathrm{E}[(X_1 - \mathrm{E}[X_1])(X_n - \mathrm{E}[X_n])] \\ \mathrm{E}[(X_2 - \mathrm{E}[X_2])(X_1 - \mathrm{E}[X_1])] & \mathrm{E}[(X_2 - \mathrm{E}[X_2])(X_2 - \mathrm{E}[X_2])] & \cdots & \mathrm{E}[(X_2 - \mathrm{E}[X_2])(X_n - \mathrm{E}[X_n])] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[(X_n - \mathrm{E}[X_n])(X_1 - \mathrm{E}[X_1])] & \mathrm{E}[(X_n - \mathrm{E}[X_n])(X_2 - \mathrm{E}[X_2])] & \cdots & \mathrm{E}[(X_n - \mathrm{E}[X_n])(X_n - \mathrm{E}[X_n])] \end{bmatrix}$$

$$\mathrm{K_{XX}} = \mathrm{cov}[\mathbf{X}, \mathbf{X}] = \mathrm{E}[(\mathbf{X} - \mu_{\mathbf{X}})(\mathbf{X} - \mu_{\mathbf{X}})^{\mathrm{T}}] = \mathrm{E}[\mathbf{X}\mathbf{X}^T] - \mu_{\mathbf{X}}\mu_{\mathbf{X}}{}^T \quad \text{(Eq.1)}$$

Weight Initialization



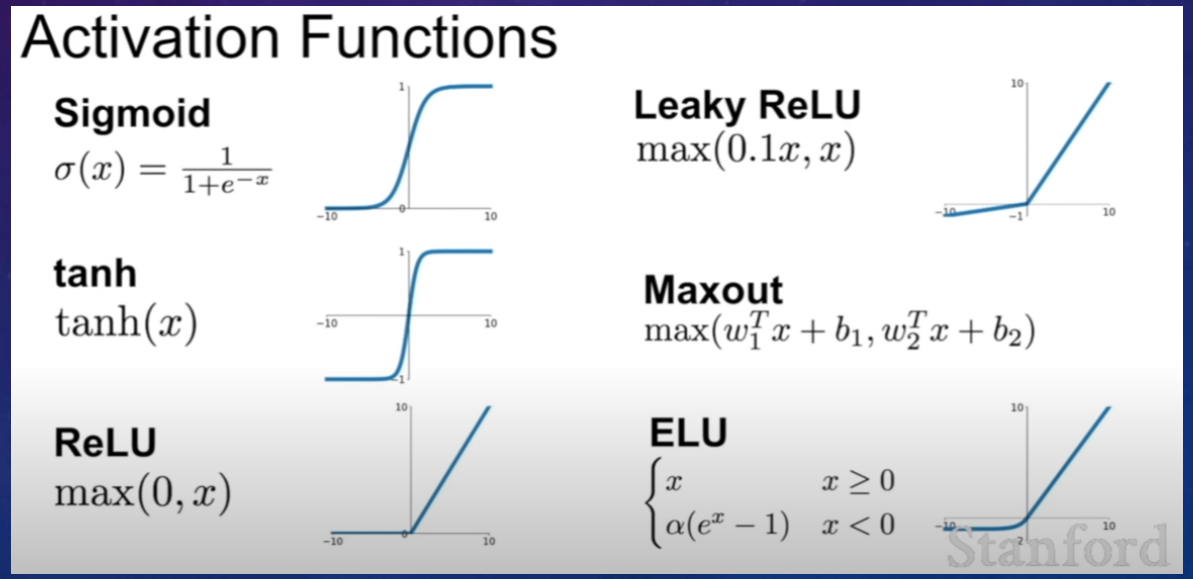
https://www.youtube.com/watch?v=8krd5qKVw-Q [10:00]

Weight Initialization – But Why?

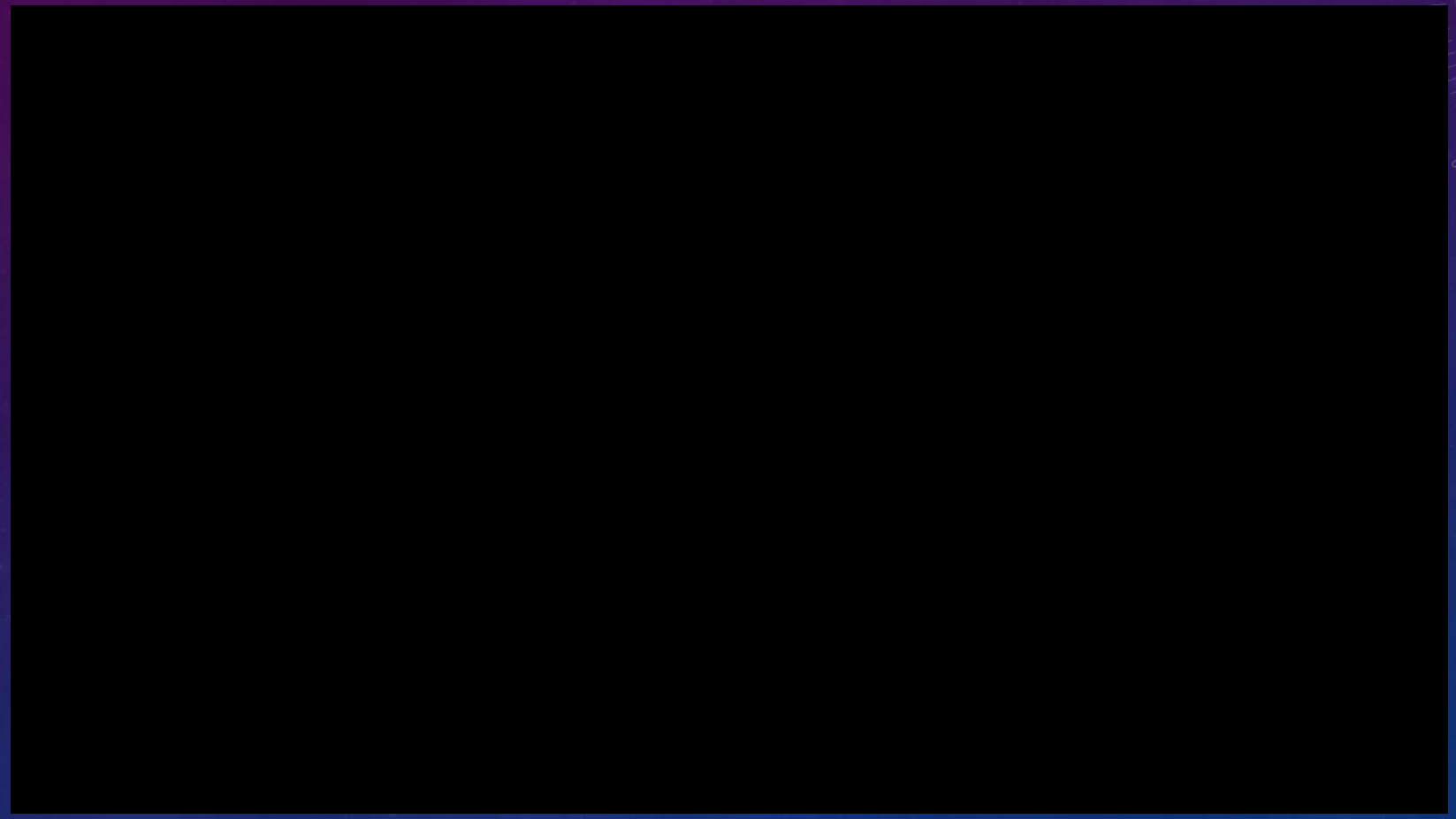
- Prevent layer activation outputs from vanishing (0) or exploding (∞) during forward pass
- This is to avoid the gradients become 0 or too large in order for the backpropagation algorithm to work

Exercise 1-1 – Weight Initialization and Forward Pass

- We have looked at two activation functions so far: tanh and ReLU. There are far more and it is an interesting area of research. Therefore, investigate the initialization behavior of two more activation functions of your choice. You can take some examples from the Stanford slides (below) or browse the internet for others. Use *Example1-1_Weight_Init_v2.ipynb* from moodle as a template.
- Upload your observations, comments and your code to Moodle.



Batch Normalization explained



https://www.youtube.com/watch?v=DtEq44FTPM4 [8:48]

Batch Normalization Mathematically

- Batch Normalization manipulates the layer inputs by calculating a batch's mean and variance. The data is then scaled and shifted
- Therefore, Batch Normalization is a special kind of preprocessing. The mathematical procedure can be seen on the right.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

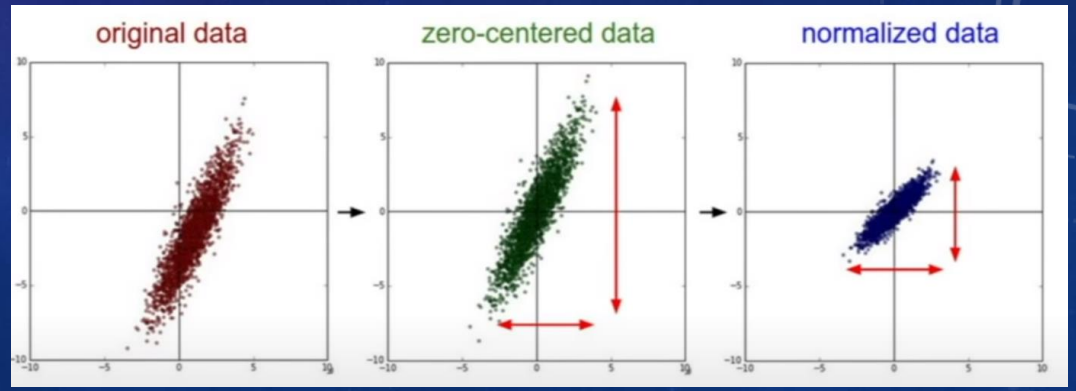
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

The simple equation behind batch normalization



Advantages of Batch Normalization

- Enables higher learning rates
- Accelerates the learning process
- Training Neural Nets with Sigmoid activations
- Bridged the other normalization methods such as Layer Normalization and weight normalization

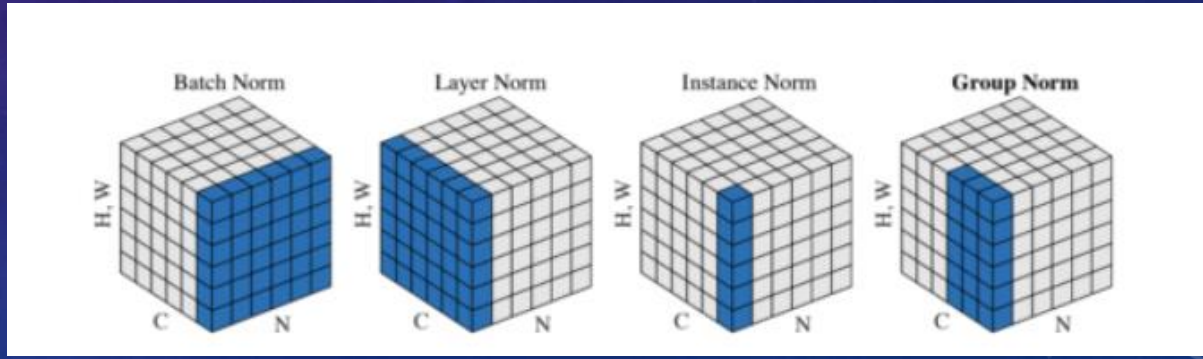
Layer Normalization

Mean of a layer

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

variance

H=# of hidden units in one layer. Details in the paper https://arxiv.org/pdf/1607.06450.pdf



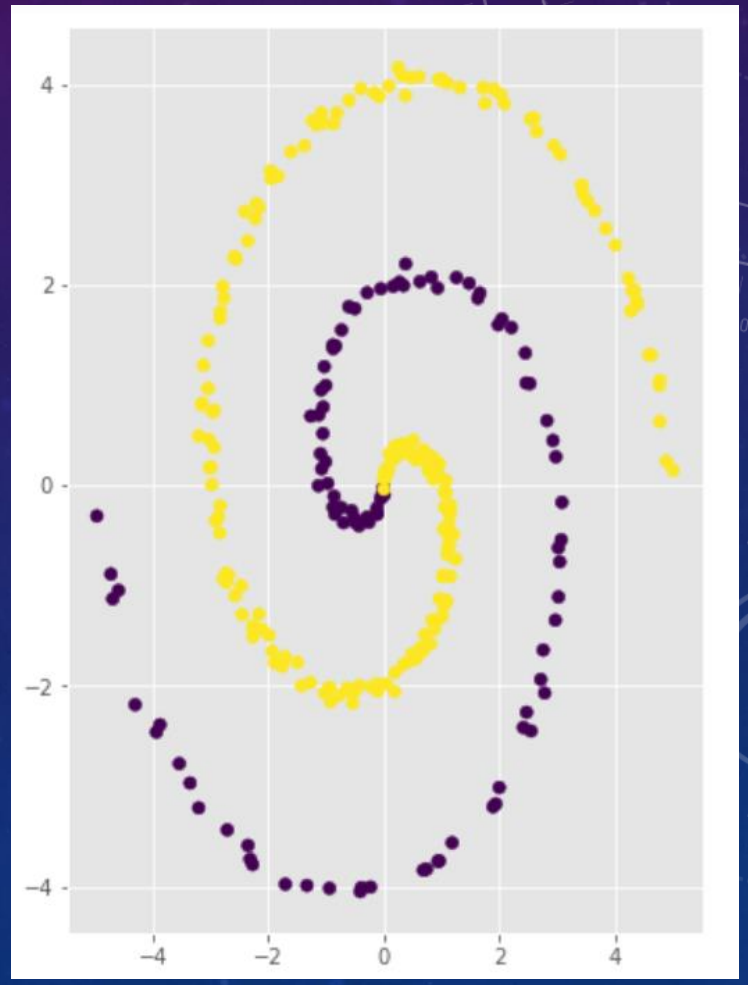
N is the batch axis, C is the Channel axis, (H,W) represent the spatial axis. The blue color indicates a normalization by same mean and variance, computed by the aggregation of the colored blocks

Babysitting the learning process

https://www.youtube.com/watch?v=qDbpYUbf3e0&feature=emb_logo [8:26]

Exercise 2-1 FCN Observe Training

- Now, we will use the spiral data. Please use the notebook, *Example2-1-FCN_Observe_Training.ipynb*, and run the same experiments for the spiral data. Make observations about the training and note them down
- You must modify the code in the following categories
  - Hidden layer size
  - Input data size
  - Number of layers
- Upload your observations, comments and your code to Moodle.



Hyperparameter optimization

https://www.youtube.com/watch?v=ttE0F7fghfk [9:50]

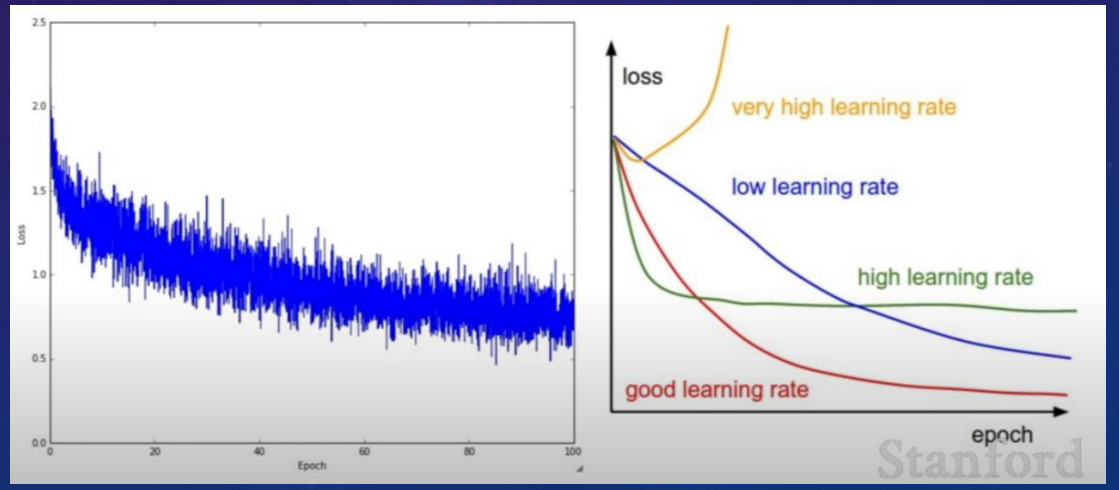
Exercise 3-1: Comparison of Different Channel on Feature Map

- Please download 3-1 Comparison of Different Channel on Feature Map.ipynb from moodle
- Change the output channel of the same layer, observe the feature map is different, and increase the training epoch

Upload your observations, comments and your code to Moodle in a docx.

Hyperparameter optimization

- Hyperparameters we can adjust
  - Network architecture
  - Learning rate, its decay schedule, update type
  - Regularization methods
- Observe the loss curve and your image outputs to make improvements



Exercise 4-1: Comparison of Deep Networks

- Please download 4-1 Deep network comparsion.ipynb from moodle
- Modify the **Net's** architecture and make its gradient not disappear (can be deleted layer, or change the components)

Upload your observations, comments and your code to Moodle in a docx.

DSS

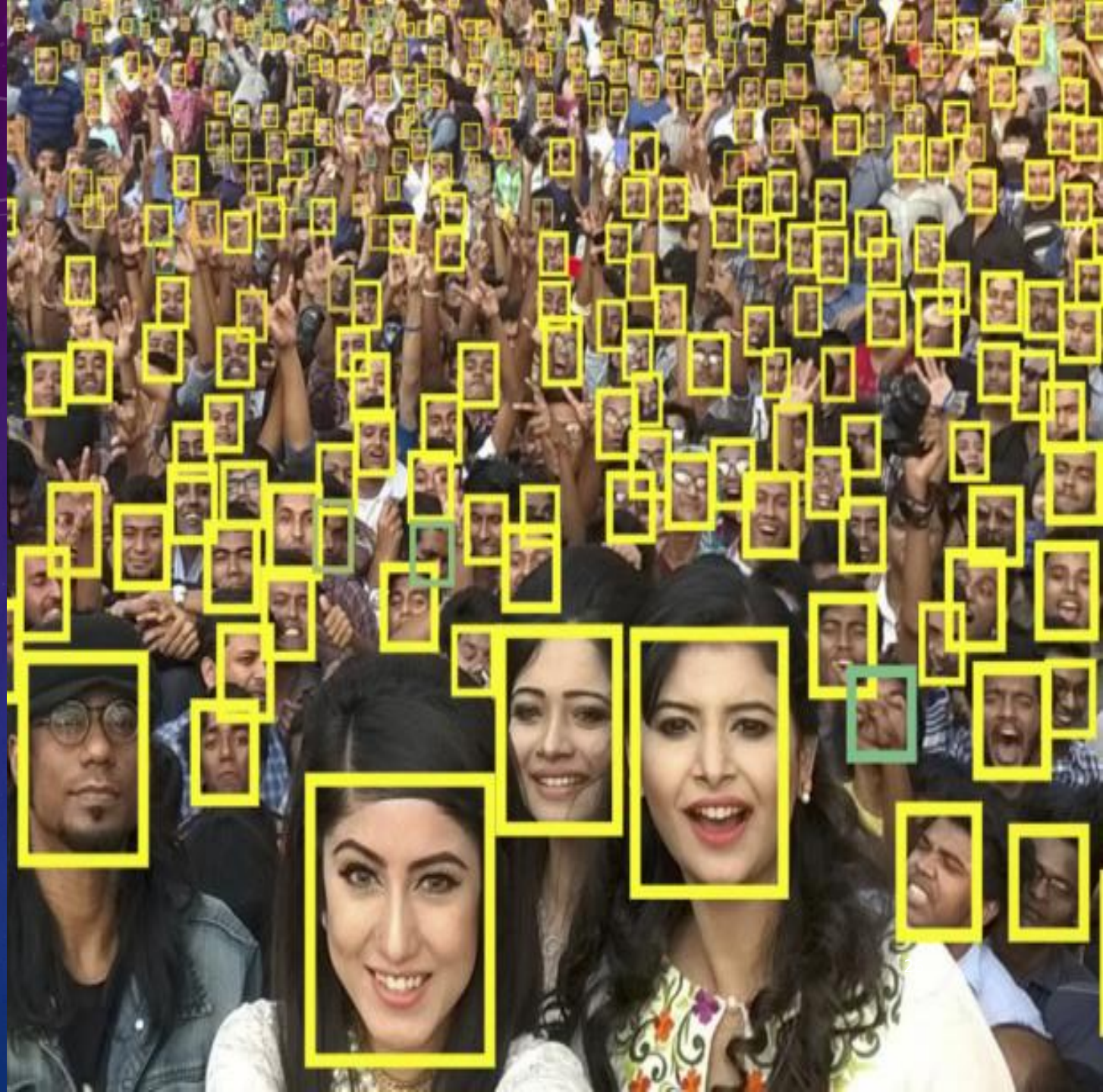
CH 2

OBJECT DETECTION

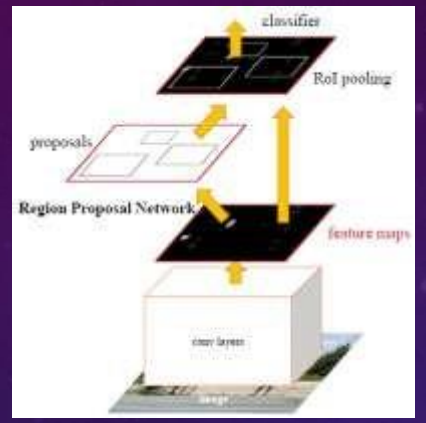
徐繼聖

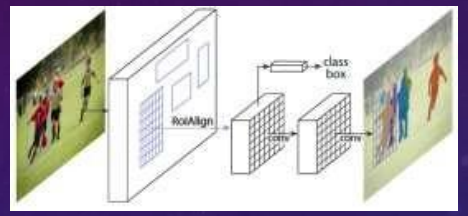
Gee-Sern Jison Hsu

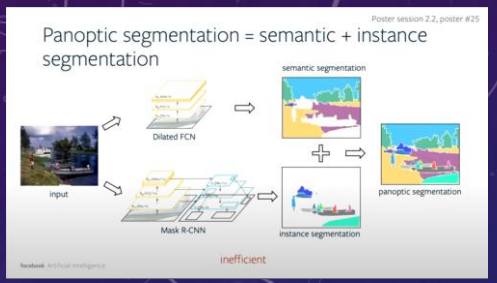
National Taiwan University of Science and Technology



The History of Object Detection in Deep Learning







RCNN

Faster RCNN

Mask RCNN

Panoptic Feature Pyramid Network

2013.11.11

2014.9 VggNet & InceptionNet

2015.4.30

2015.5.14

2015.6.8

2015.12.08

2015.12.10 ResNet

2015.12.25

2017.1.23

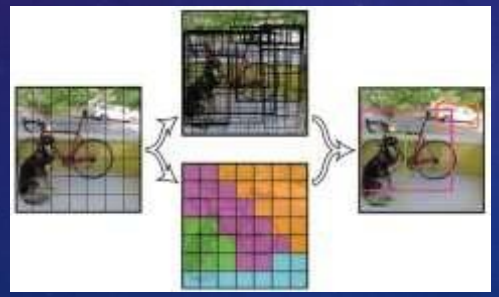
2017.3.20

2019.1.8

Fast RCNN

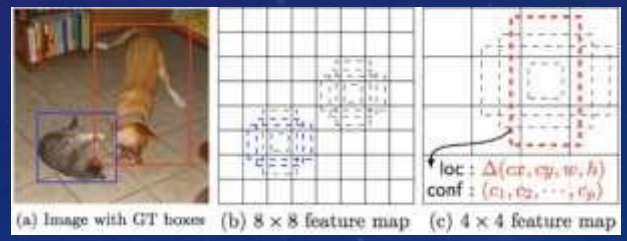
Yolo

Yolo v2



SSD

DSSD

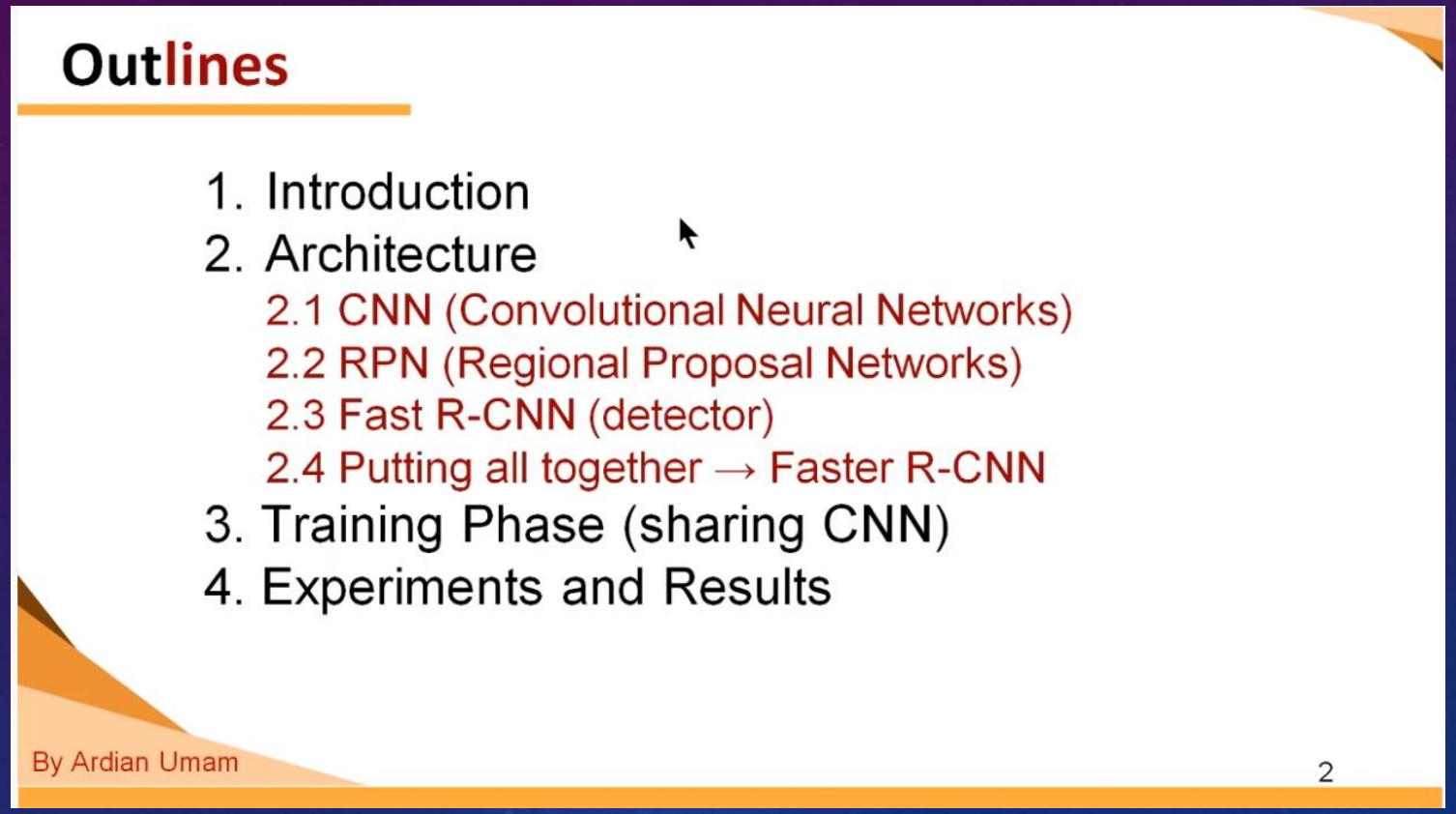


2012.12 AlexNet

Detection and Segmentation – Stanford University School of Engineering

https://www.youtube.com/watch?v=nDPWywWRIRo&list=PLC1qU-LWwrF64f4QKQT-Vg5Wr4qEE1Zxk&index=11
[1:14:25]

Introduction to How Faster R-CNN, Fast R-CNN and R-CNN Works



Refer to Appendix, Page 59

Contents

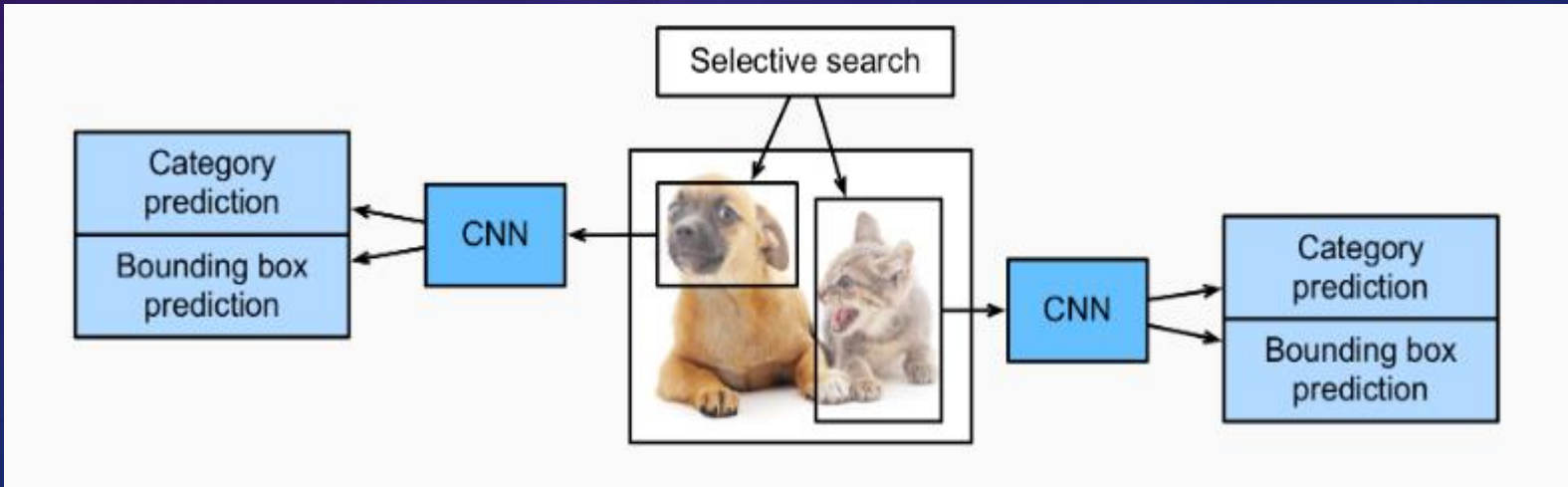
- Region-based CNN
- Fast R-CNN
- Faster R-CNN
- Mask R-CNN

Region-based CNN

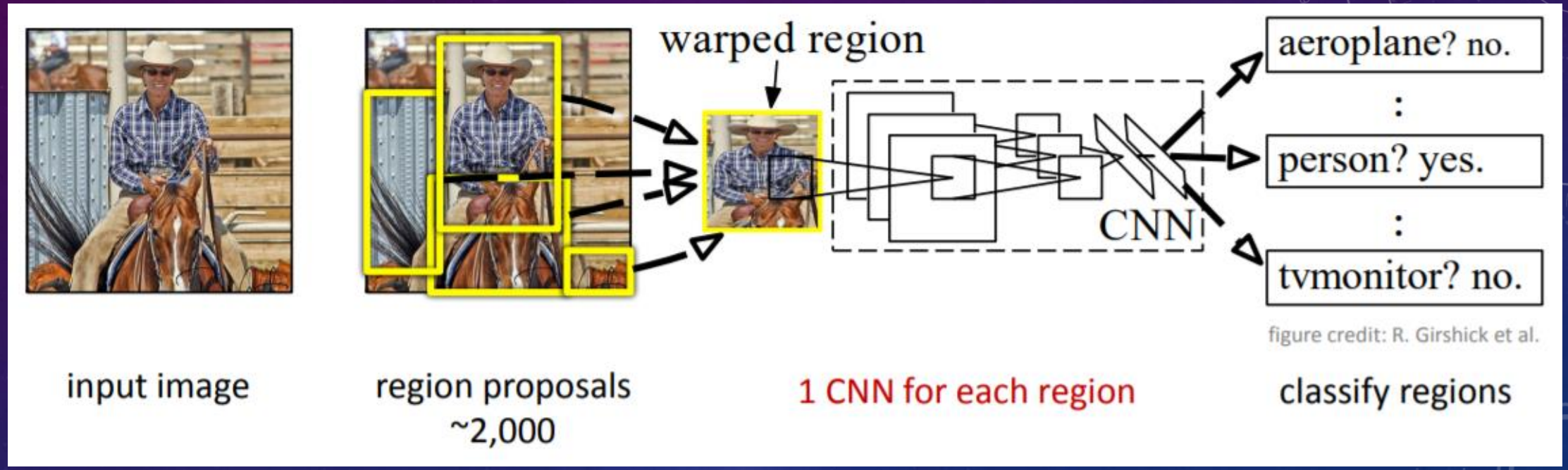
- Introduction
- Region-Based CNN
- R-CNN Algorithm
- Region Proposals – Selective Search

Region-based CNN

- R-CNN models first select several proposed regions from an image (for example, anchor boxes are one type of selection method) and then label their categories and bounding boxes (e.g., offsets).
- They use a CNN to perform forward computation to extract features from each proposed area. Afterwards, we use the features of each proposed region to predict their categories and bounding boxes.



Region-Based CNN Architecture



R-CNN pipeline

R-CNN Algorithm

Per-Image computation

Extract features by f_I

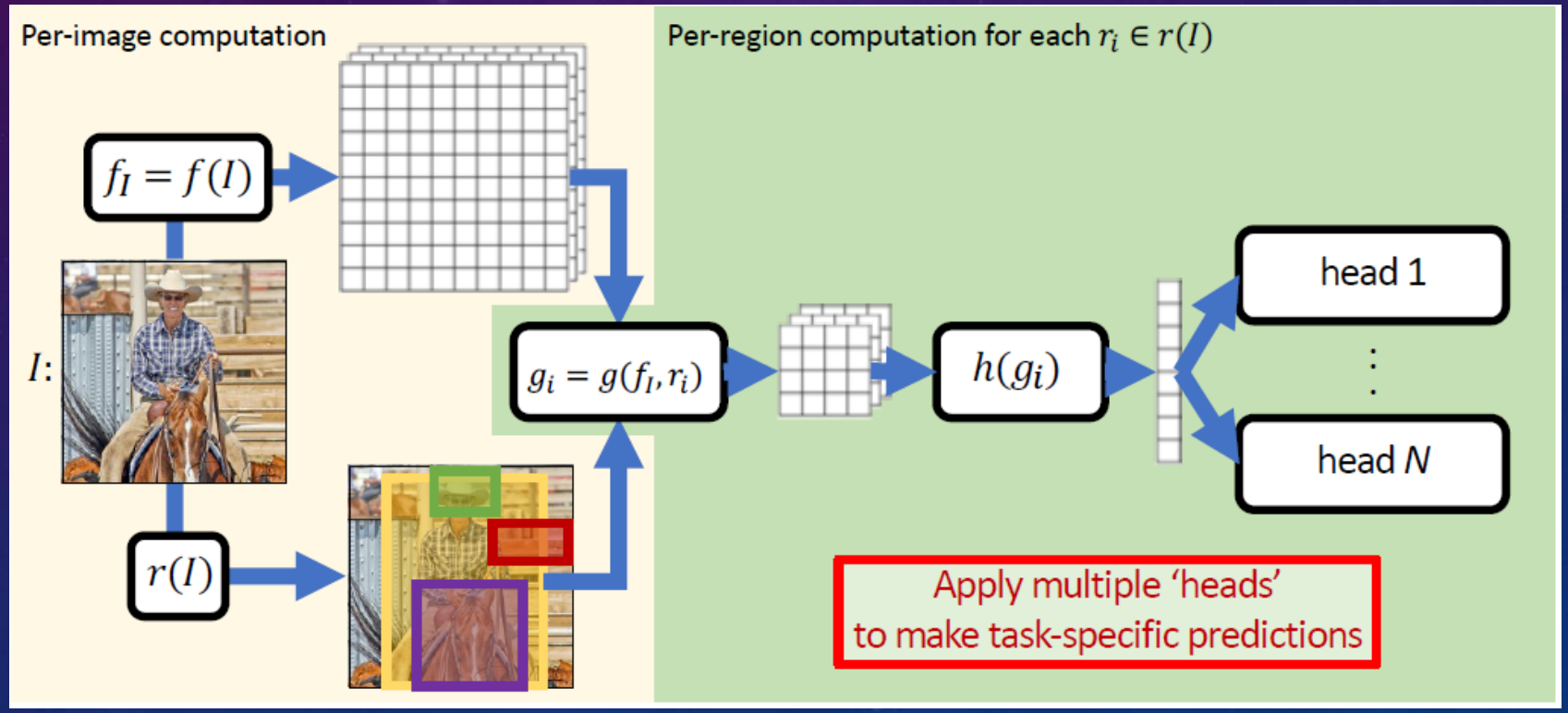
Region detection proposal $r(I)$

Per-Region computation

Extract region of Interest g_i

Pass each region to CovNet $h(g_i)$

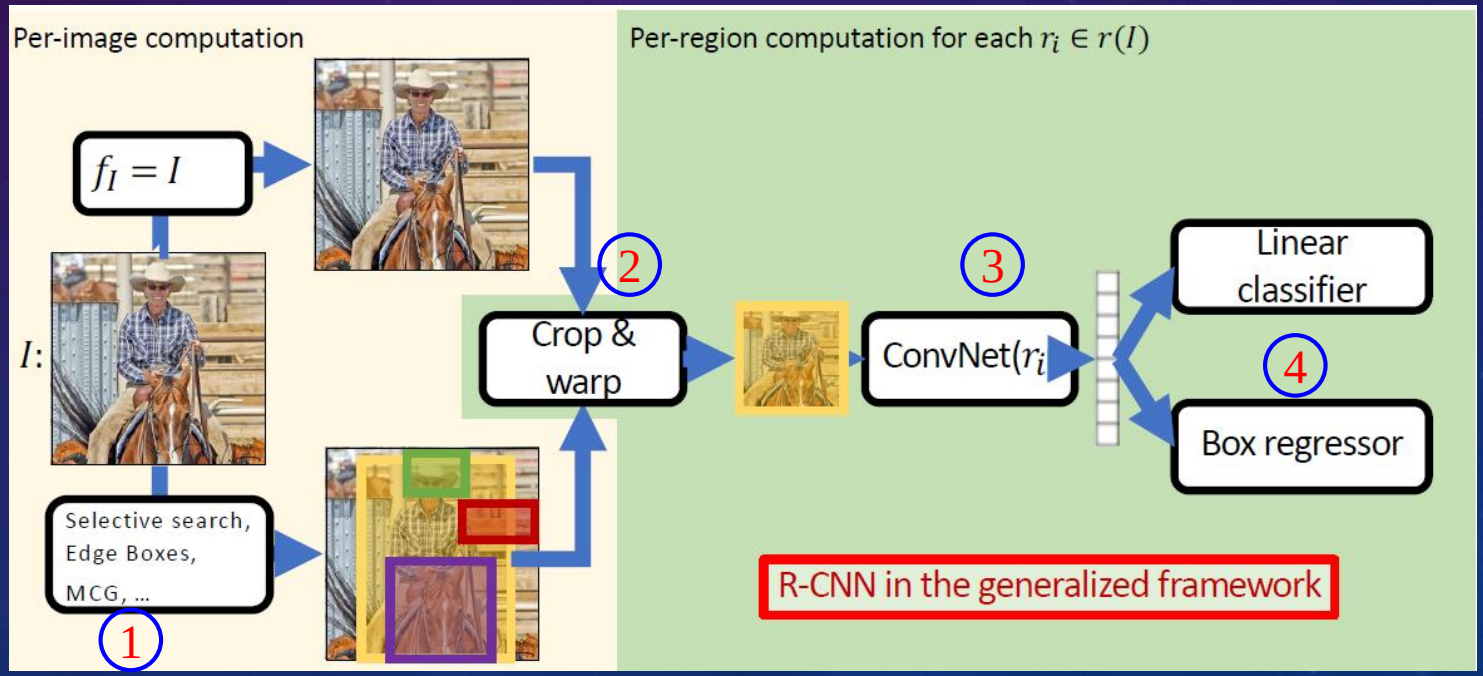
multiple heads to make prediction



R-CNN Algorithm

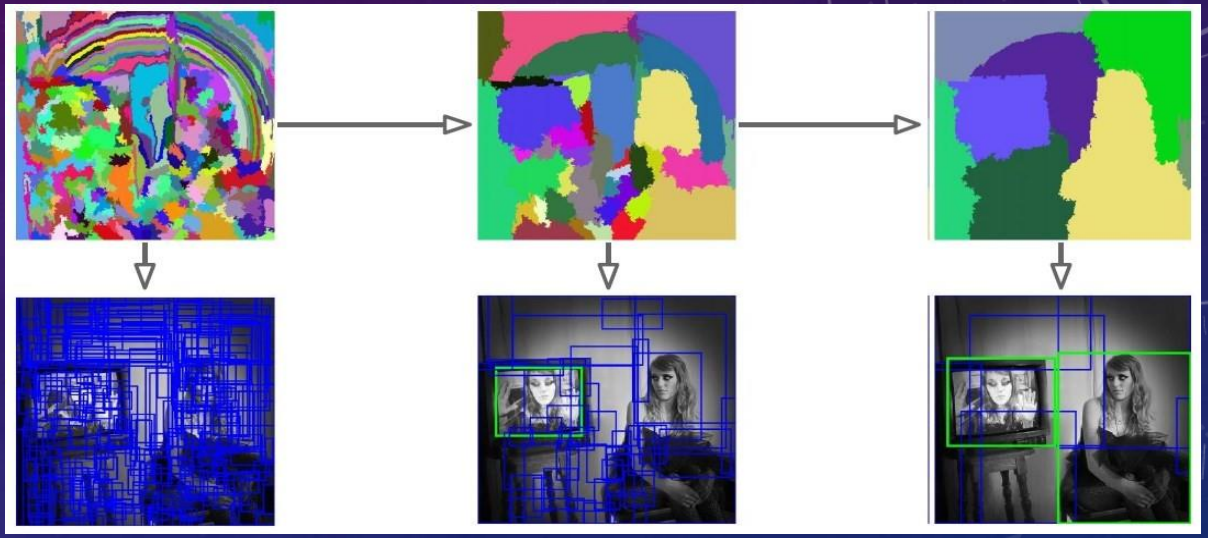
- Use selective search to find region proposal (~2k)
- Extract region of interest from original image
- Apply ConvNet to each region of interest
- Use support vector machine to find class label and linear regression to find bounding box offsets.

Problem:
- ✓ Very heavy per-region computation
- ✓ Ad hoc training objective
- ✓ Inference is very slow



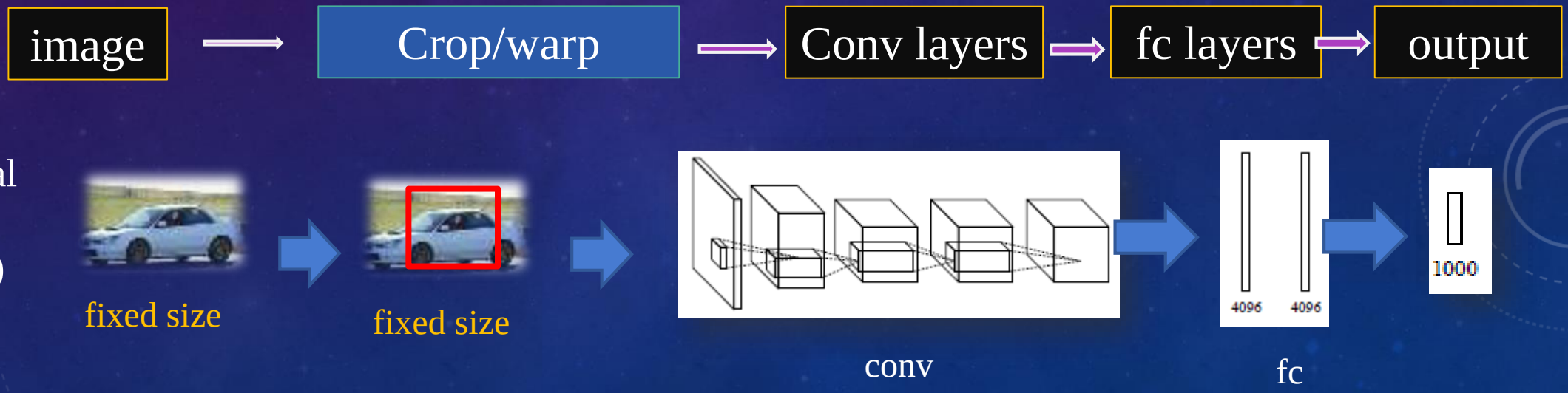
Region Proposals – Selective Search

- It is based on computing hierarchical grouping of similar regions based on color, texture, size and shape compatibility.
- Selective Search starts by over-segmenting the image based on intensity of the pixels using a graph-based segmentation method by Felzenszwalb and Huttenlocher.



R-CNN

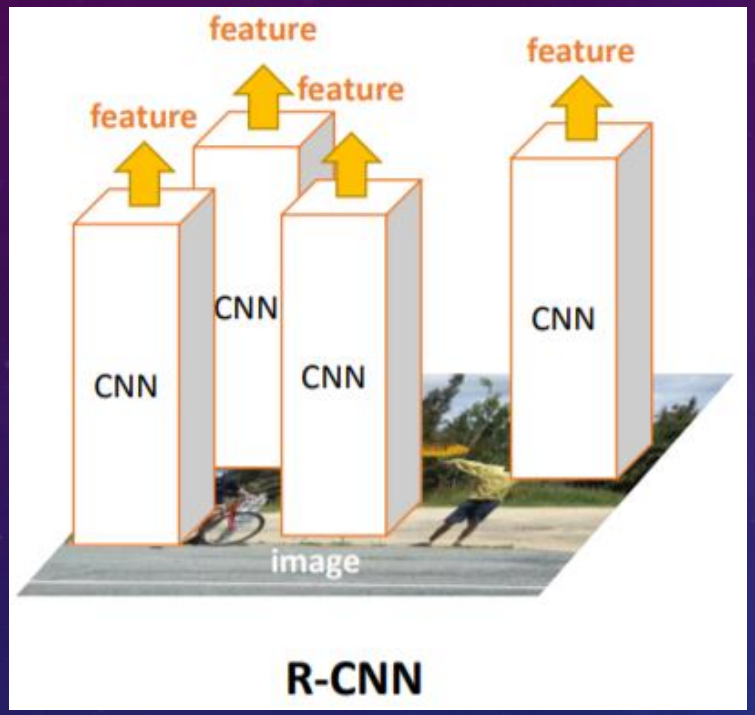
- In a typical CNN structure, a full connection is usually connected after the convolutional layer. The number of features of the fully connected layer is fixed, so the input size (fixed-size) is fixed when the network is input. But in reality, the image size of our input is always unable to meet the size required for input. However, the usual methods are cropping and warping.



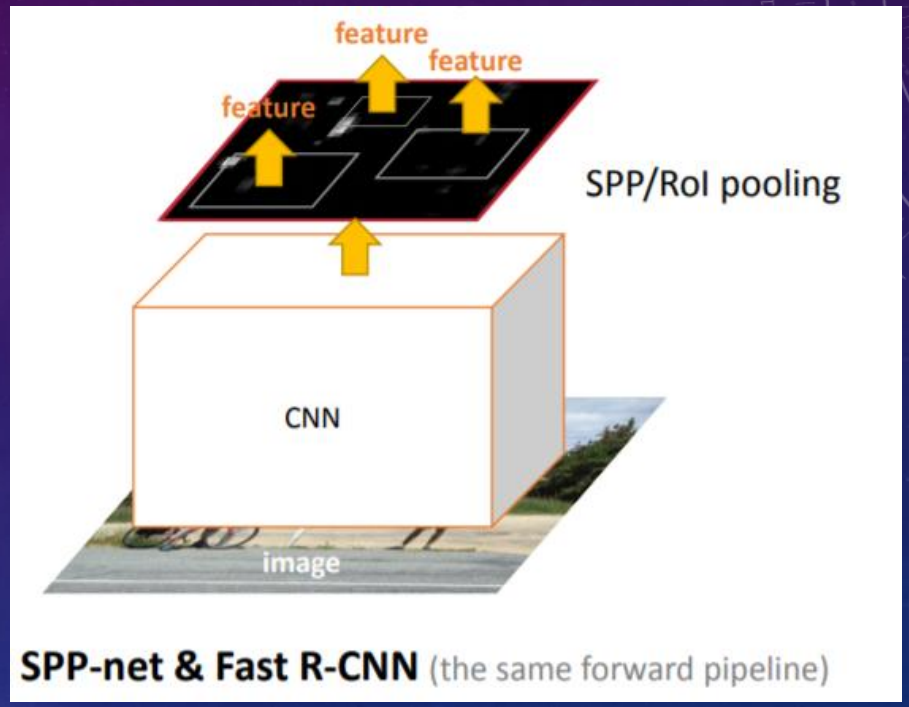
Fast R-CNN

- Spatial Pyramid Pooling net
- Fast R-CNN architecture
- What is the ROI pooling layer?
- ROI pooling Summary
- ROI pooling Summary
- Problems of Fast R-CNN

R-CNN vs. Fast R-CNN (Forward Pipeline)

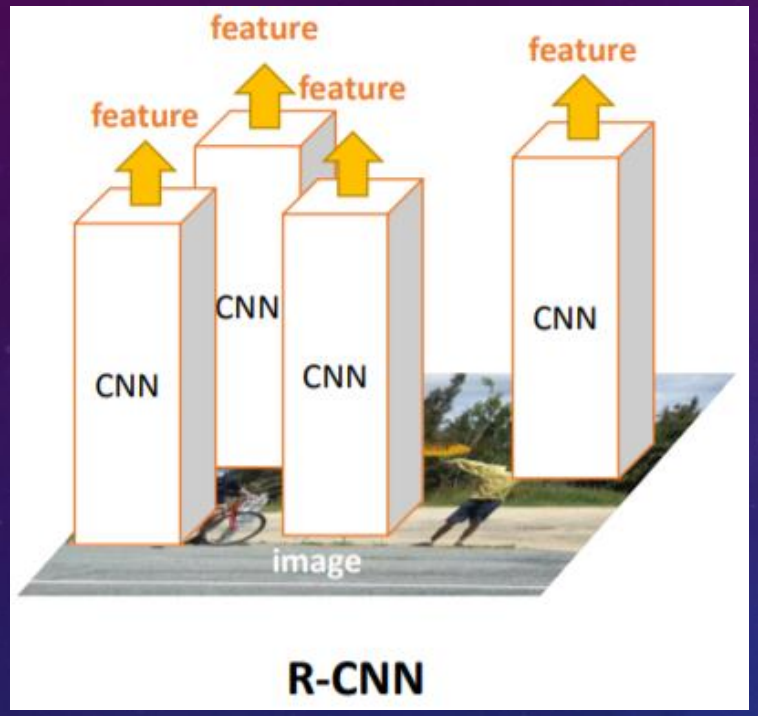


- Extract image regions
- 1 CNN per region (2000 CNNs)
- Classify region-based features

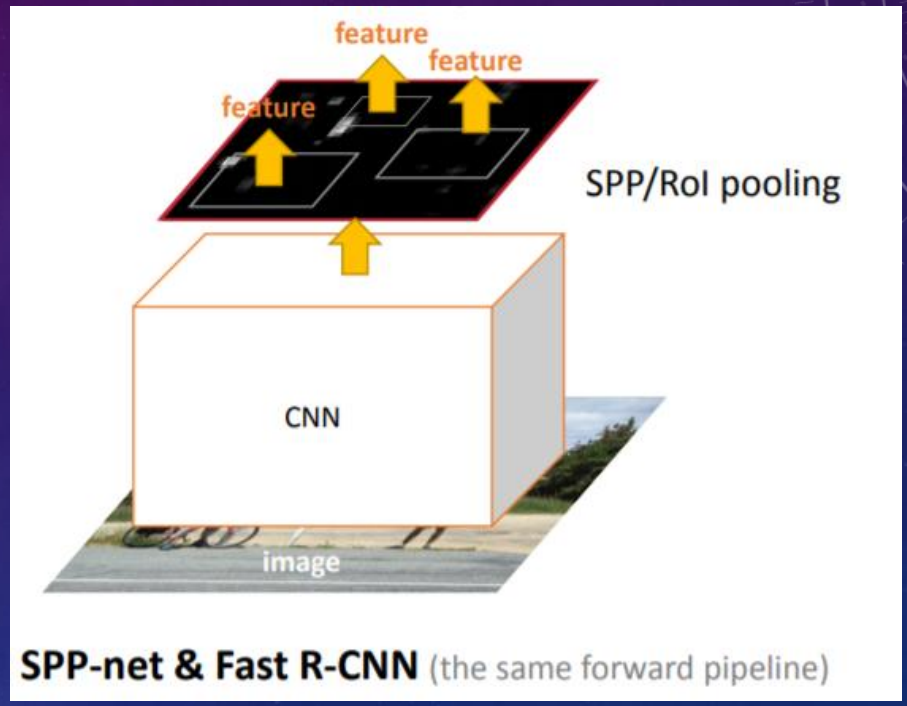


- 1 CNN on the entire image
- Extract features from feature map regions
- Classify region-based features

R-CNN vs. Fast R-CNN (Forward Pipeline)



- Complexity: ~224 × 224 × 2000



- Complexity: ~600 × 1000 × **1**
- ~160x faster than R-CNN

R-CNN vs. Fast R-CNN

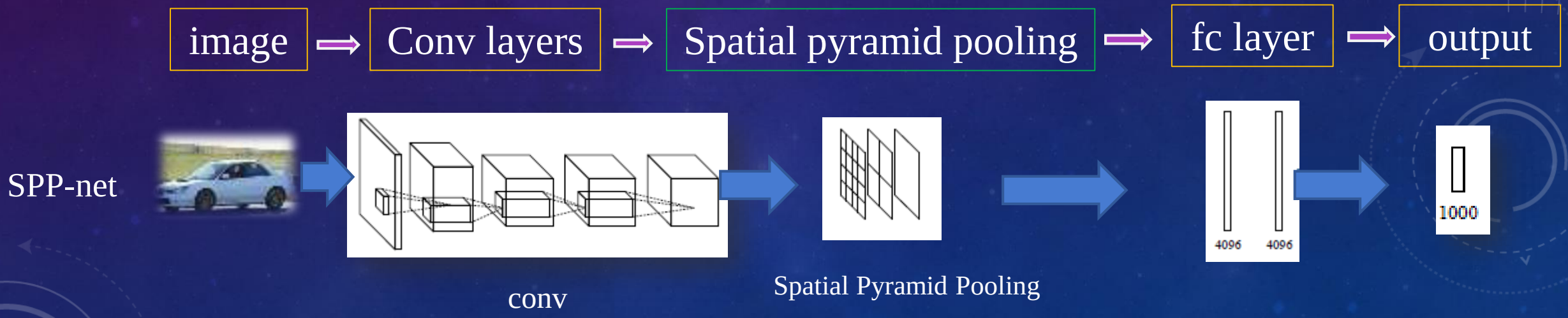
- Fix most of what's wrong with R-CNN and SPP-net

- Train the detector in a single stage, end-to-end
  - No caching features to disk
  - No post hoc training steps

- Train all layers of the network

Spatial Pyramid Pooling net

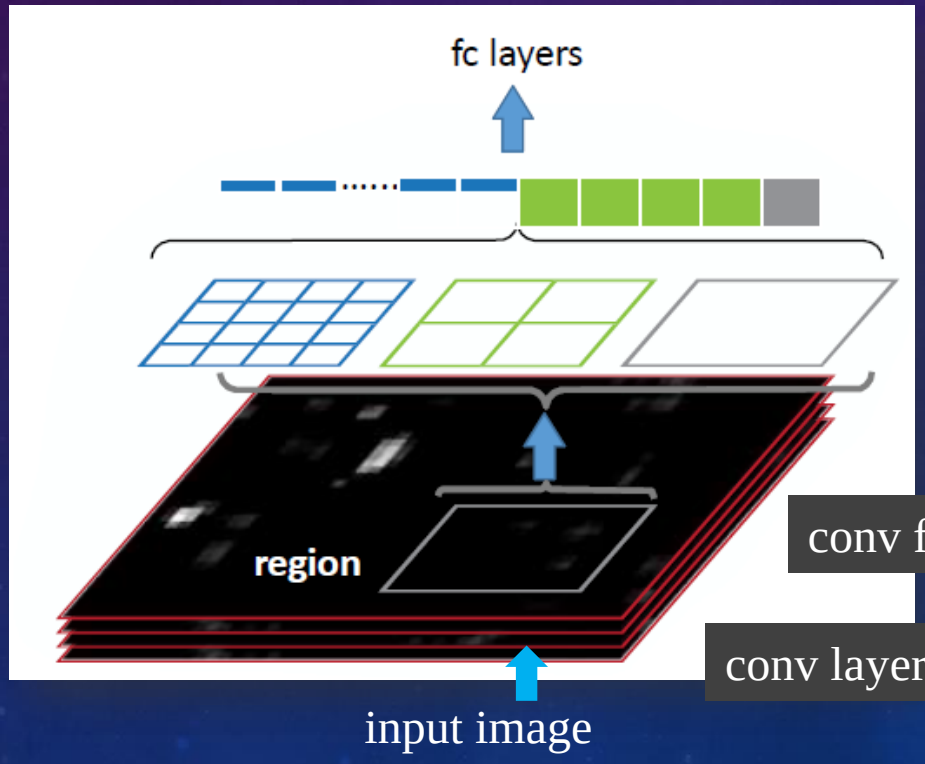
- SPP can produce a fixed size output regardless of the input size
- Use multiple windows (pooling window)
- SPP can use the same image with different scales as input to get the same length of pooling features.



Spatial Pyramid Pooling net

How do we get the fixed output?

SPP NET use multiple windows (pooling window, the blue, green, silver gray window in the input picture, then pooling the feature maps, the combined results will get a fixed length output).

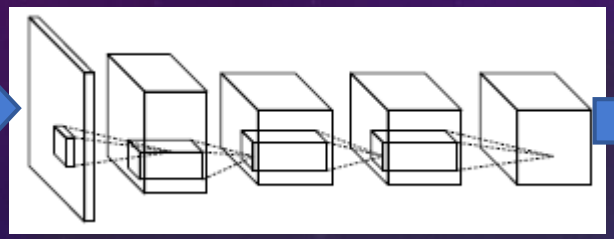


Spatial Pyramid Pooling

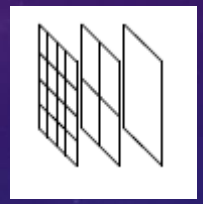
Spatial Pyramid Pooling (SPP)

image ⇒ Conv layers ⇒ Spatial pyramid pooling ⇒ fc layer ⇒ output

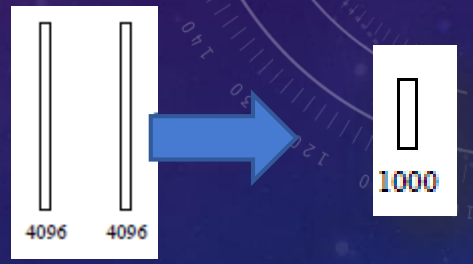
SPP-net



conv



Spatial Pyramid Pooling



Backward

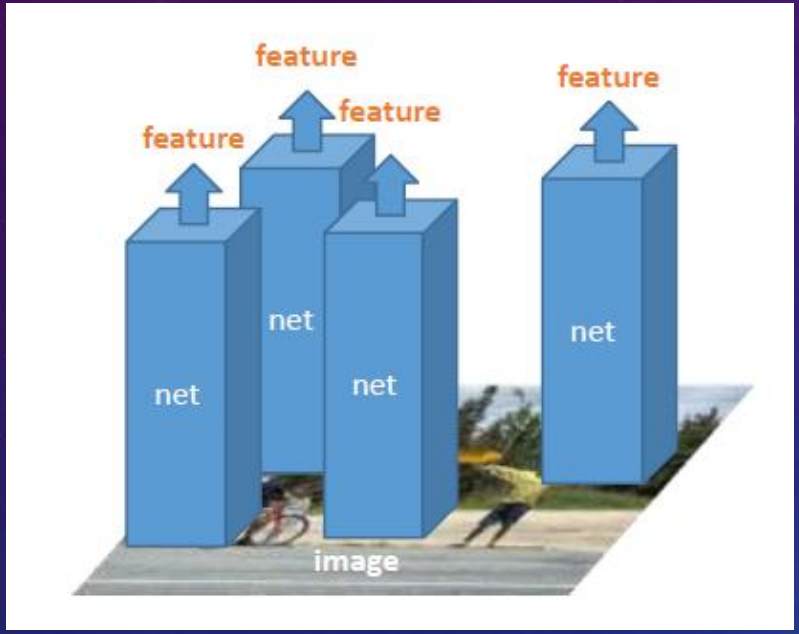
R times
Slow and Heavy
computation

R: # of RoIs

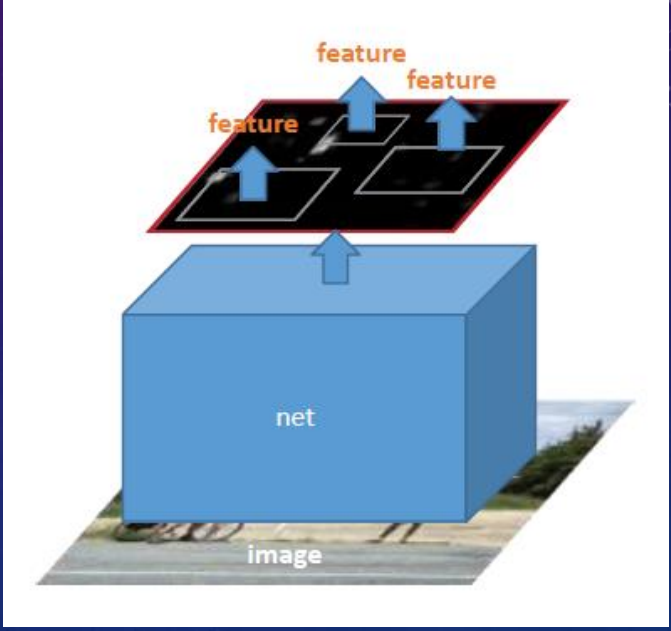
R times
back-propagation

RCNN vs. SPP

- image regions *vs*. feature map regions



R-CNN
2000 nets on image regions



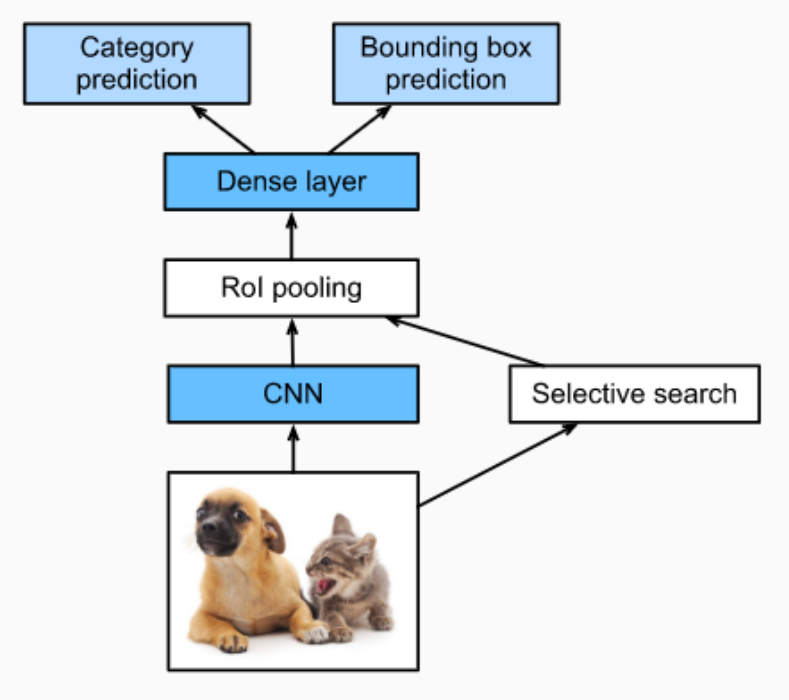
SPP-net
1 net on full image

Problems of R-CNN

- **Slow at test-time**: need to run full forward path of CNN for each region proposal
  - 13s/image on a GPU(K40)
  - 53s/image on a CPU
- **SVM and regressors are post-hoc**: CNN features not updated in response to SVMs and regressors
- **Complex multistage training pipeline** (84 hours using K40 GPU)
  - Fine-tune network with softmax classifier(log loss)
  - Train post-hoc linear SVMs(hinge loss)
  - Train post-hoc bounding-box regressions(squared loss)

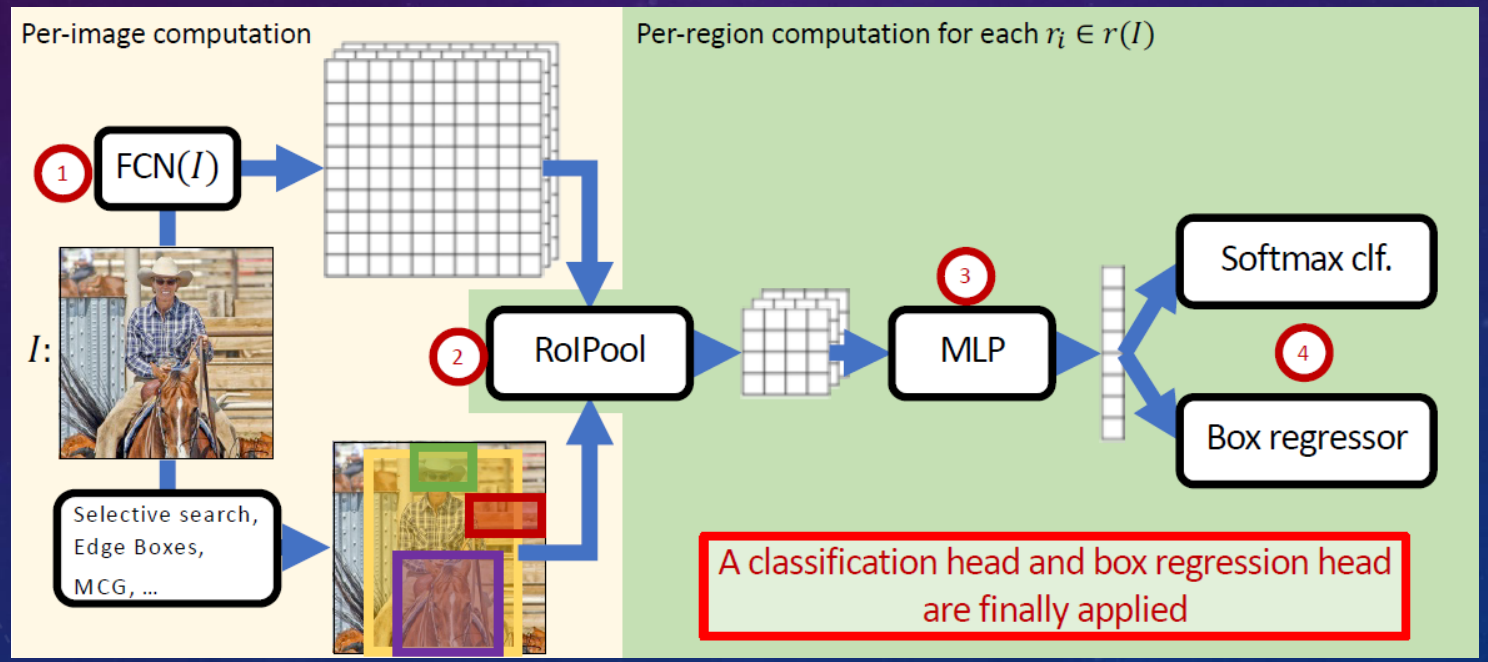
FAST R-CNN

- The main performance bottleneck of an R-CNN model is the need to independently extract features for each proposed region.
- As these regions have a high degree of overlap, independent feature extraction results in a high volume of repetitive computations. Fast R-CNN improves on the R-CNN by only performing CNN forward computation on the image as a whole.



FAST R-CNN

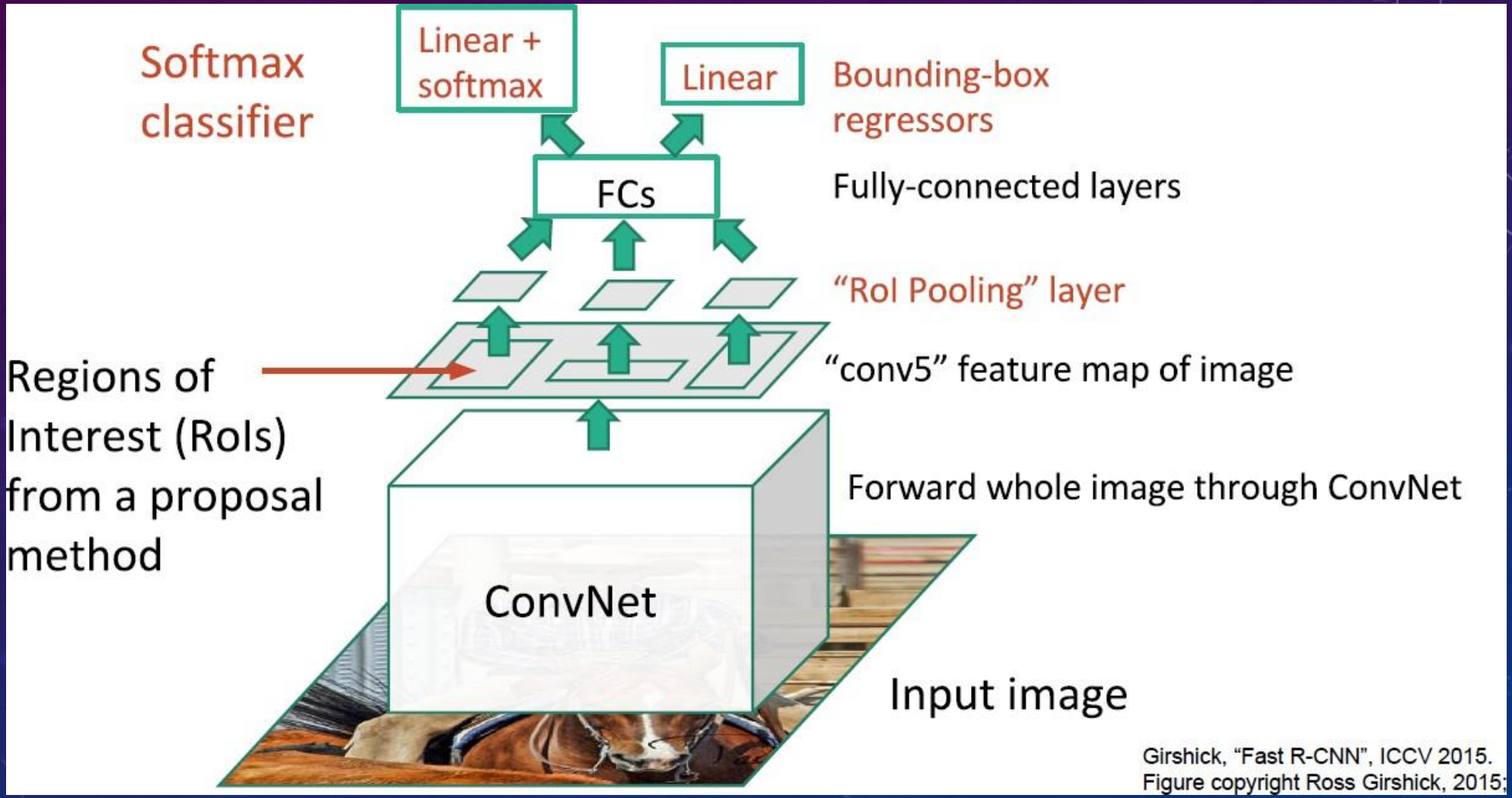
- Use fully convolutional network to extract features
- Extract region of interest from feature map and use region of interest (*RoI*) pooling layer extracts a fixed-length feature vector from the feature map.
- A sequence of fully connected (*fc*) layers that finally branch into two sibling output layers: one that produces softmax probability estimates over K object classes plus a "background" class and another layer that outputs four real-valued numbers for each of the K object classes.



Problem:

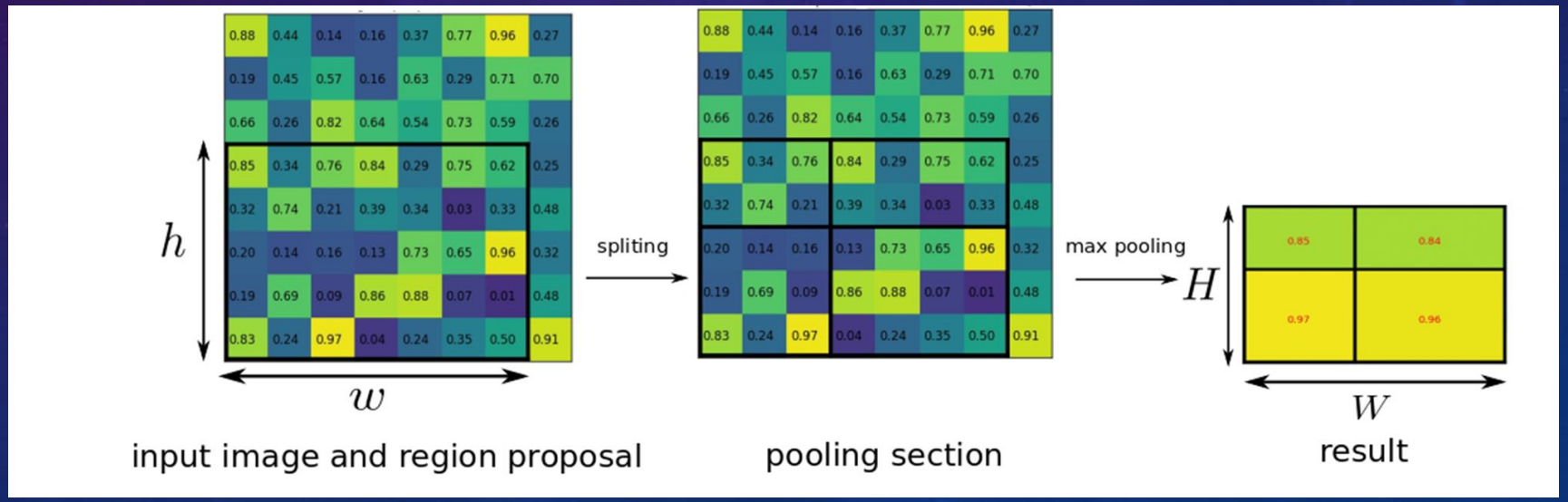
- ✓ Use selective search still slow
- ✓ Not end-to-end training

FAST R-CNN


Softmax classifier
Linear + softmax
Linear
Bounding-box regressors
FCs
Fully-connected layers
"RoI Pooling" layer
Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015;

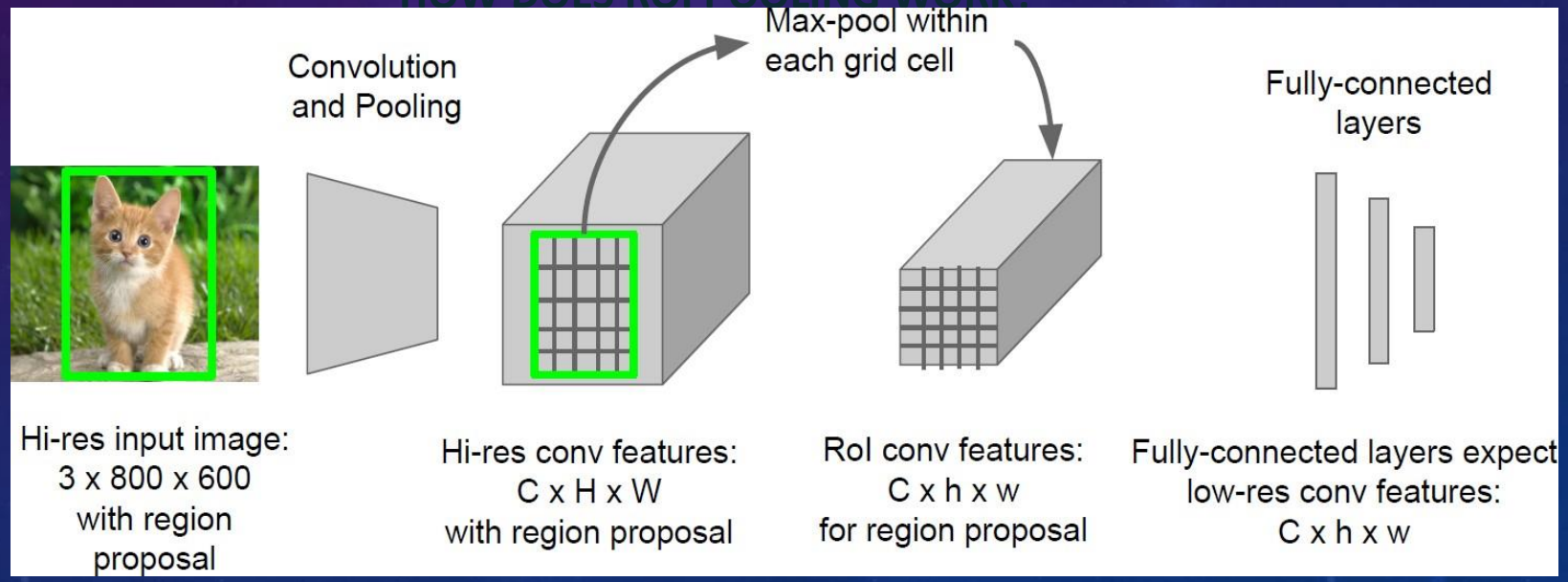
What is the ROI pooling layer?

- Suppose we got the region proposal (left) with $h \times w$, and we would like to have an output (right) of H×W sizes of output layer after pooling. Then, the area for each pooling area (middle) = $h/H \times w/W$.
- And in the example above, with input ROI of 5×7, and output of 2×2, the area for each pooling area is 2×3 or 3×3 after rounding.
- And the maximum value within the pooling window is taken as output value for each grid which is the same idea of conventional max pooling layer.



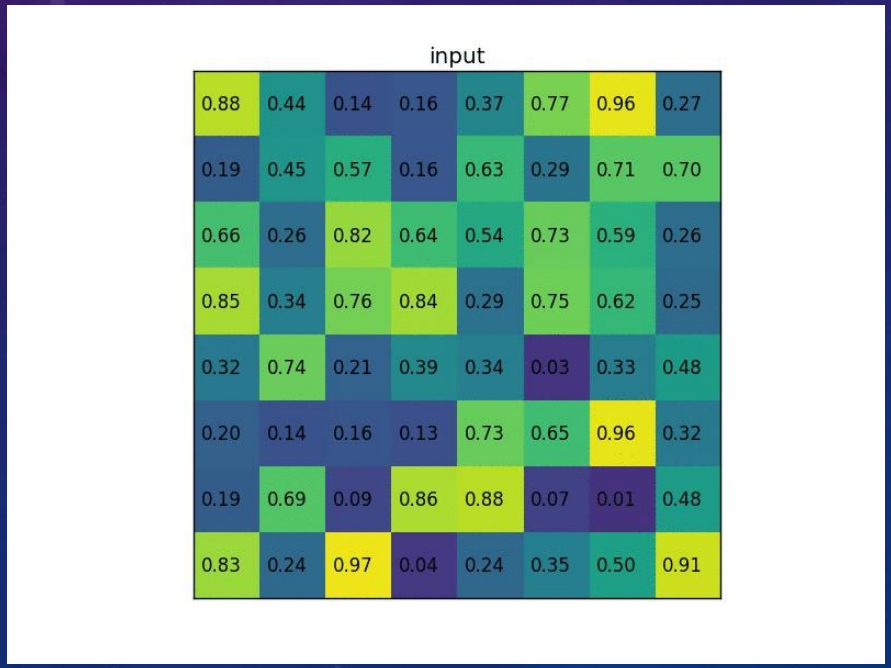
How does ROI pooling work?

- RoI max pooling works by dividing the **h × w** RoI window into an **H × W** grid of sub-windows of approximate size **h/H × w/W** and then max-pooling the values in each sub-window into the corresponding output grid cell.
- i.e. the "max-pooling" filter size becomes **h/H × w/W**. Pooling is applied how it is normally applied, independently to each feature map channel.



ROI pooling Summary

- ROI pooling summary:
- It’s used for object detection tasks
- It allows us to reuse the feature map from the convolutional network
- It can significantly speed up both train and test time
- It allows to train object detection systems in an end-to-end manner



Problems of Fast R-CNN

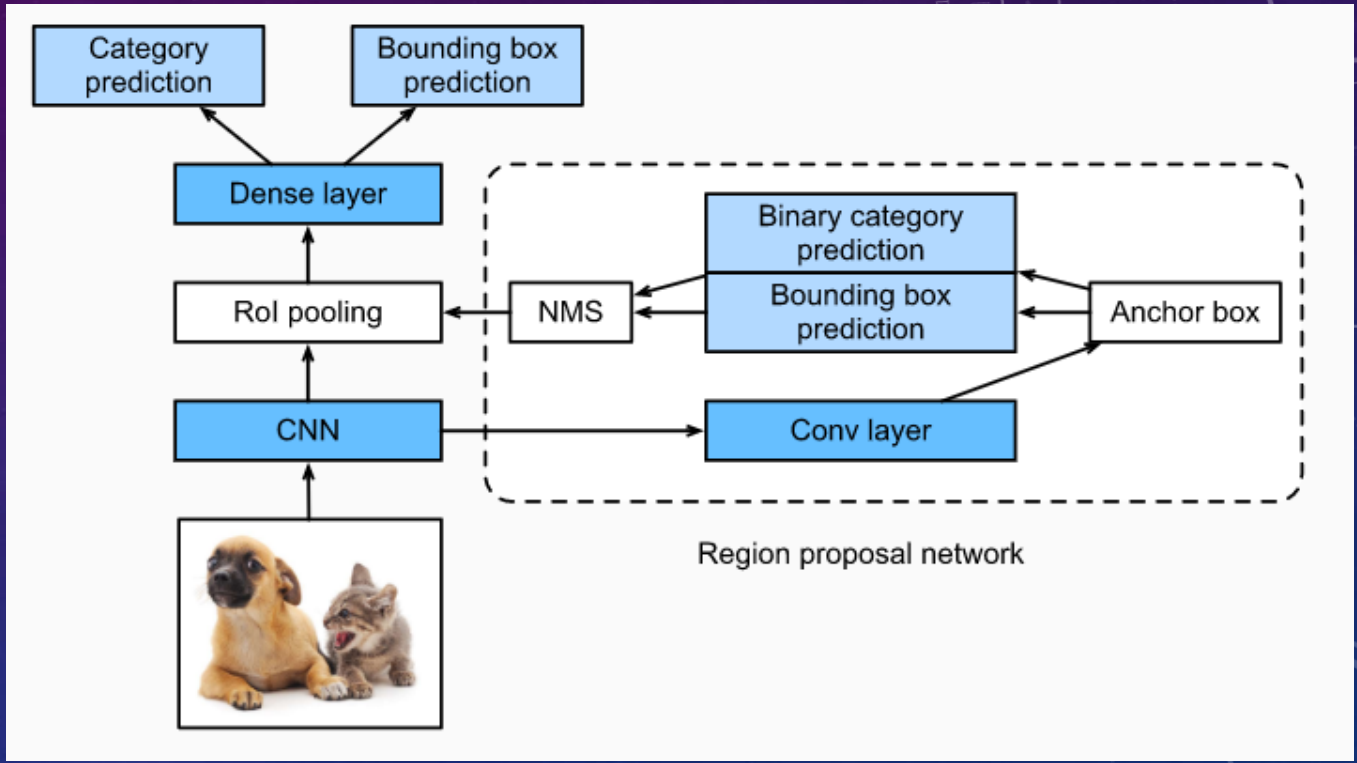
- Out-of-network region proposals are the test-time computational bottleneck
- Is it fast enough??

Faster R-CNN

- Faster R-CNN architecture
- Faster R-CNN(RPN + fast R-CNN)
- How to train faster R-CNN
- Training goal : share features
- How RPN (region proposal networks) works
- Region Proposal Network
- RPN(fully convolutional network)

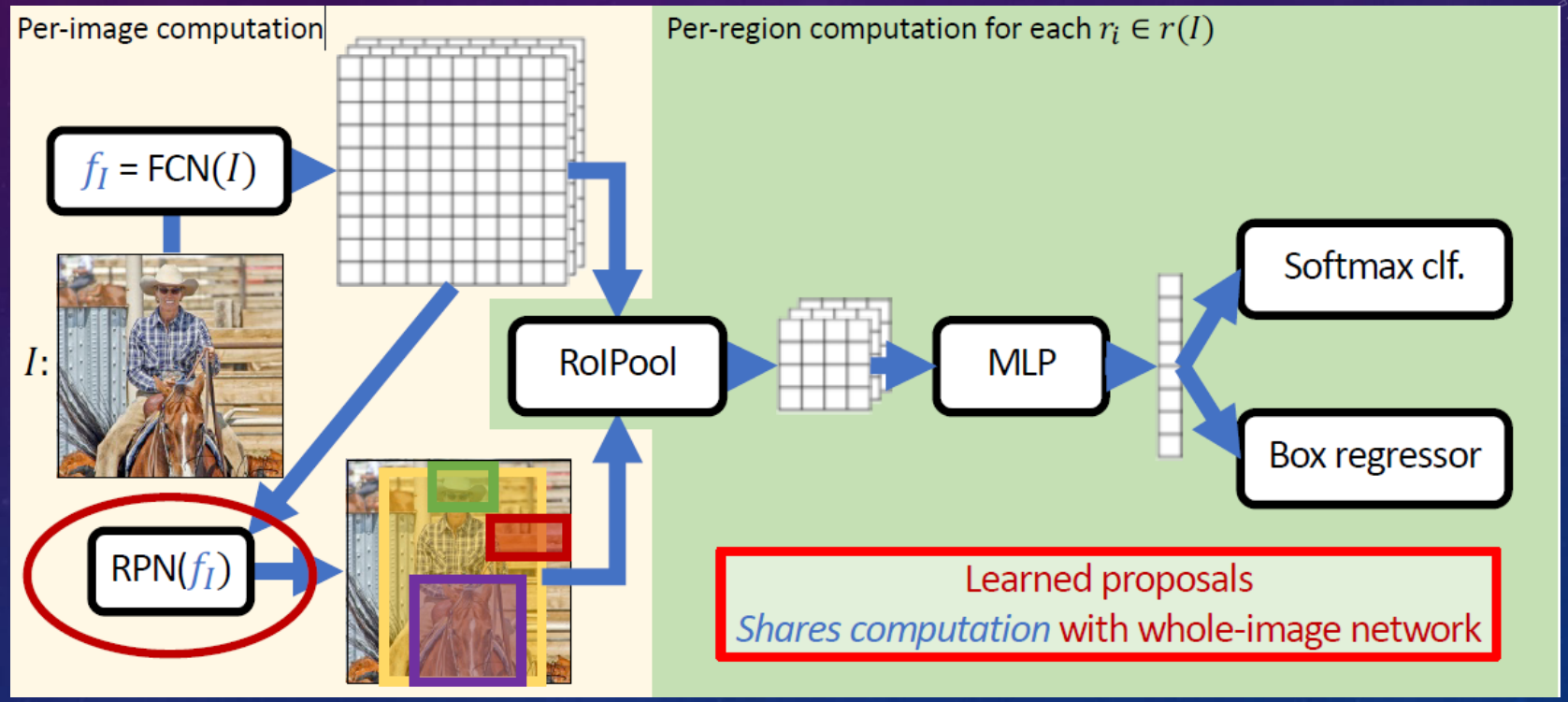
Faster R-CNN

- In order to obtain precise object detection results, Fast R-CNN generally requires that many proposed regions be generated in selective search. Faster R-CNN replaces selective search with a region proposal network. This reduces the number of proposed regions generated, while ensuring precise object detection.



Faster R-CNN

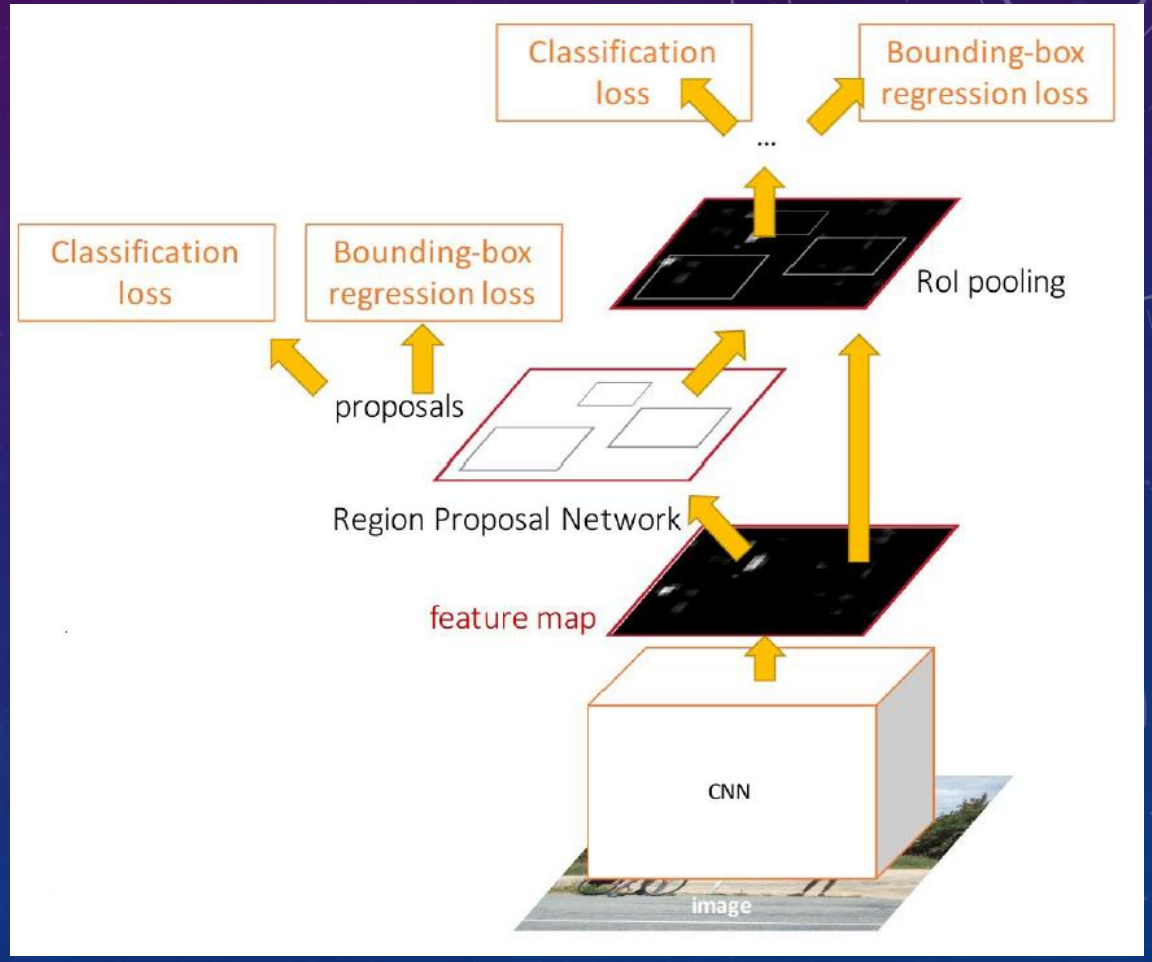
- Replace selective search with Region Proposal Network
- End-to-end training



Faster R-CNN

- Jointly train the whole network with 2 losses
  - RPN classifies object/non-object
  - RPN regresses bbox coordinates
- Results show the final classification score and bbox coordinates.

$$L = L_{cls} + L_{reg}$$



Loss Function

$$L_{cls} = -\frac{1}{N_{cls}} \sum_{\mathrm{i}} L_{cls}(p_i, p_i^*)$$

- Here i is the index of the anchor in the mini-batch.
- $L_{cls}(p_i, p_i^*)$: log loss over two classes (object vs non-object)
- p_i: output score from the classification branch for anchor i
- p_i^*: groundtruth label (1 or 0).

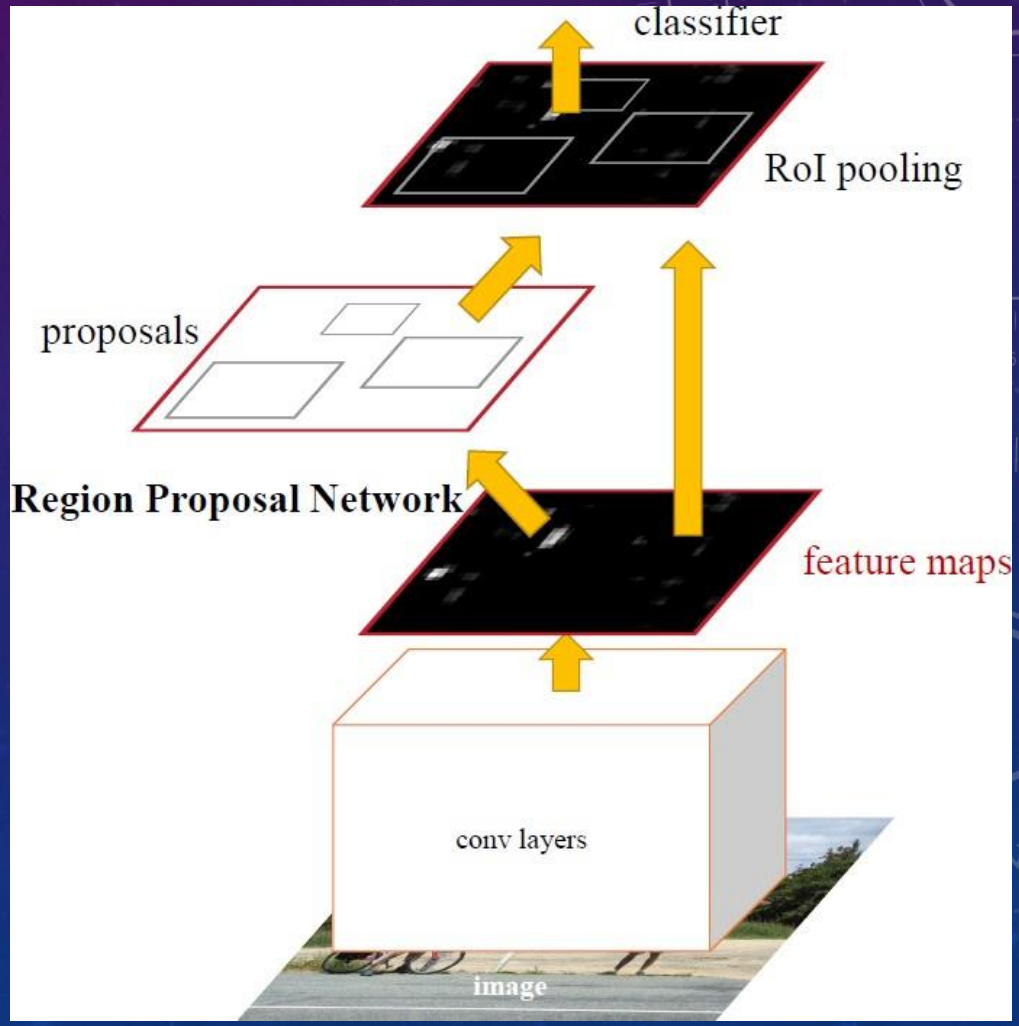
Loss Function

$$L_{reg} = \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

- The regression loss $L_{reg}(t_i, t_i^*)$ is activated only if the anchor actually contains an object.
- t_i: The output prediction of the regression layer and consists of 4 variables $[t_x, t_y, t_w, t_h]$.
- t_i^*: $t_x^* = (x^* - x_a)/w_a$, $t_y^* = (y^* - y_a)/h_a$, $t_w^* = \log(w^*/w_a)$, $t_h^* = \log(h^*/h_a)$

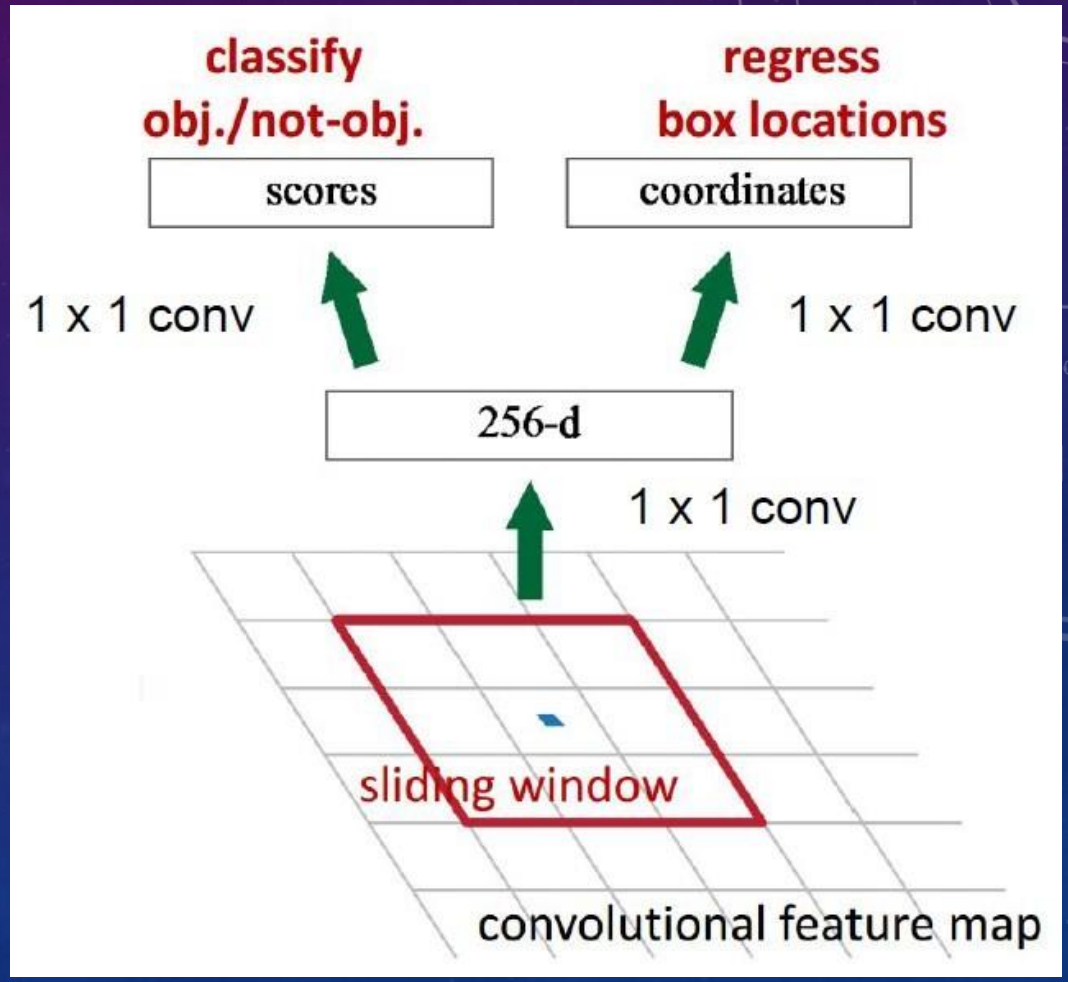
Faster R-CNN (RPN + Fast R-CNN)

- Insert a Region Proposal Network (RPN) after the last convolutional layer
- RPN trained to produce region proposals directly; no need for external region proposals
- After RPN, use RoI Pooling and an upstream classifier and bbox regressor just like Fast R-CNN



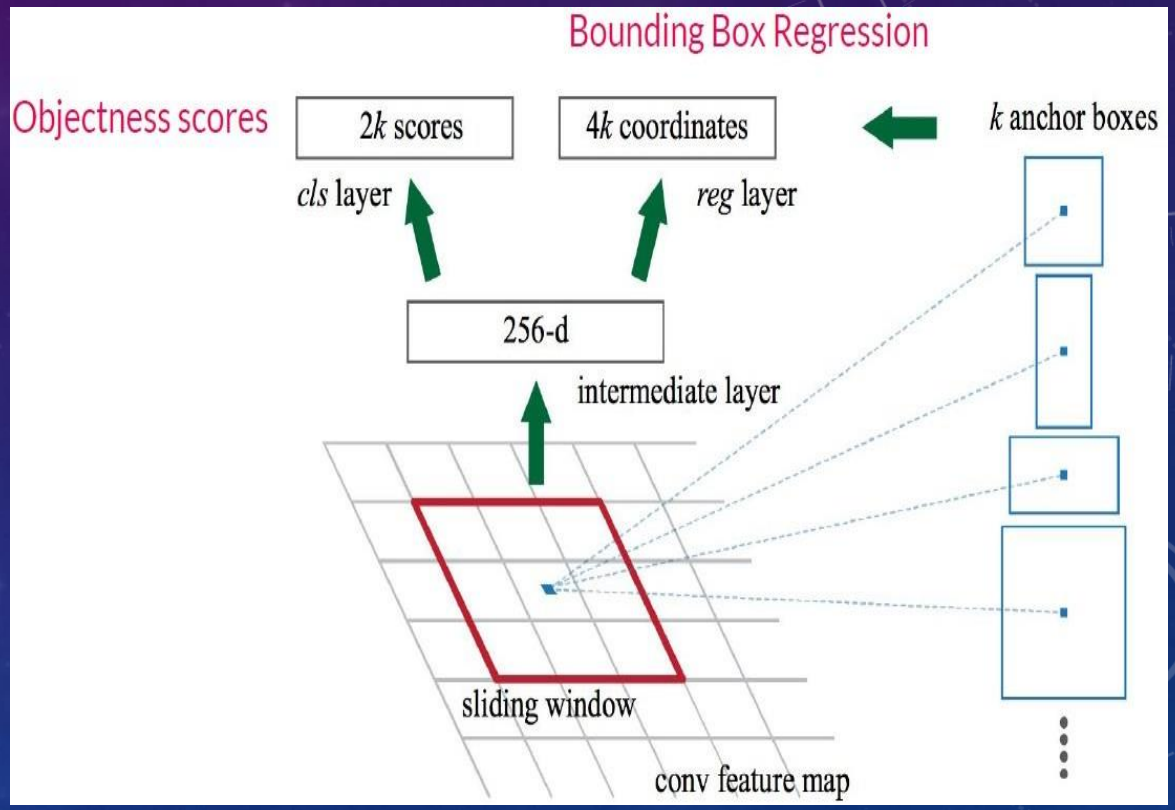
Region Proposal Network

- Slide a small window on the feature map
- Build a small network for
  - Classifying object or not-object
  - Regressing bbox locations
- Position of the sliding window provides localization information with reference to the image
- Box regression provides finer localization information with reference to this sliding window



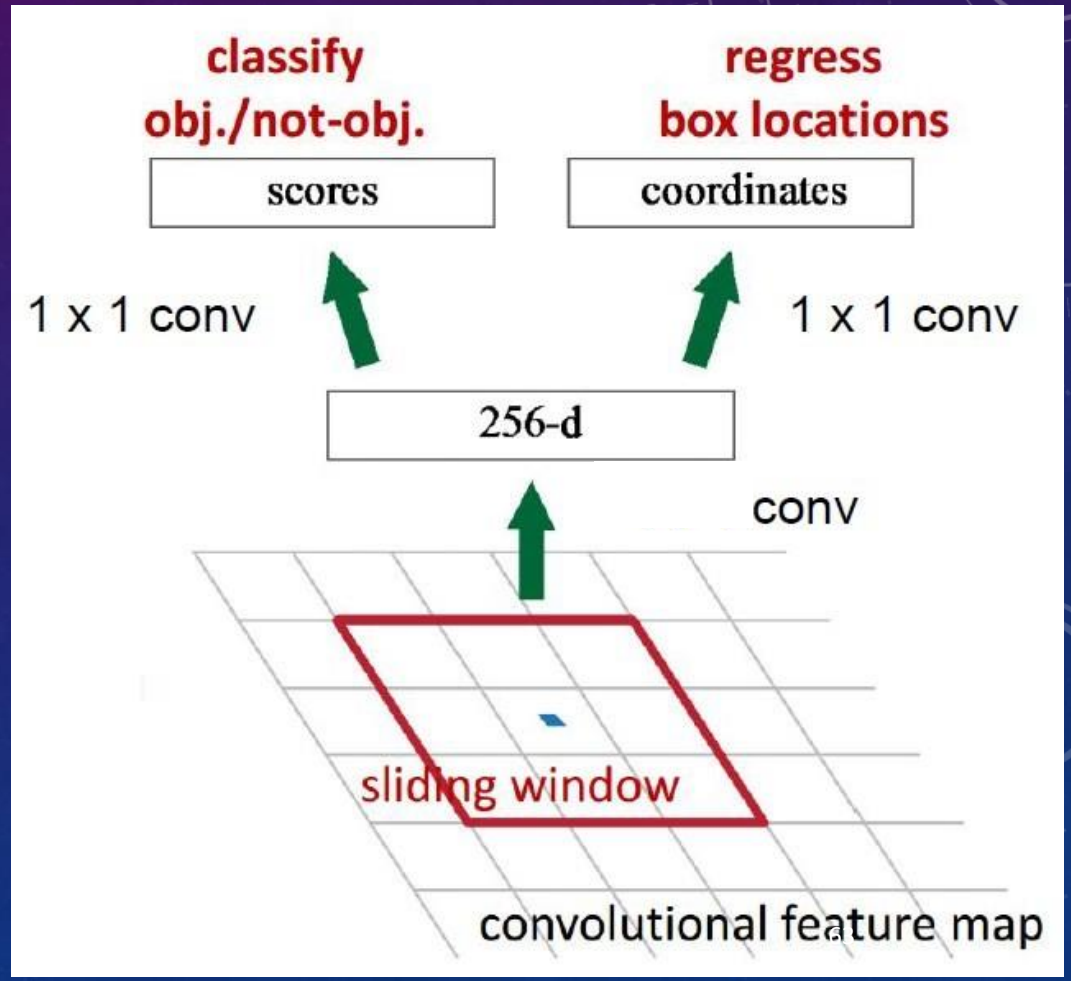
Region Proposal Network

- Use K anchor boxes at each location
- Anchors are translation invariant: use the same ones at every location
- Regression gives offsets from anchor boxes
- Classification gives the probability that each (regressed) anchor shows an object



RPN (Fully Convolutional Network)

- Intermediate Layer – 256(or 512) 3x3 filter, stride 1, padding 1
- Cls layer – 18(9x2) 1x1 filter, stride 1, padding 0
- Reg layer – 36(9x4) 1x1 filter, stride 1, padding 0



Anchors As References

- Anchors: pre-defined reference boxes
- Multi-scale/size anchors:
  - Multiple anchors are used at each position:
    3 scale(128x128, 256x256, 512x512) and 3 aspect rations(2:1, 1:1, 1:2) yield 9 anchors
  - Each anchor has its own prediction function
- Single-scale features, multi-scale predictions

Is Faster RCNN enough?

- RoI Pooling has some quantization operations
- These quantizations introduce misalignments between the RoI and the extracted features
- While this may not impact classification, it can make a negative effect on predicting bbox

Mask R-CNN

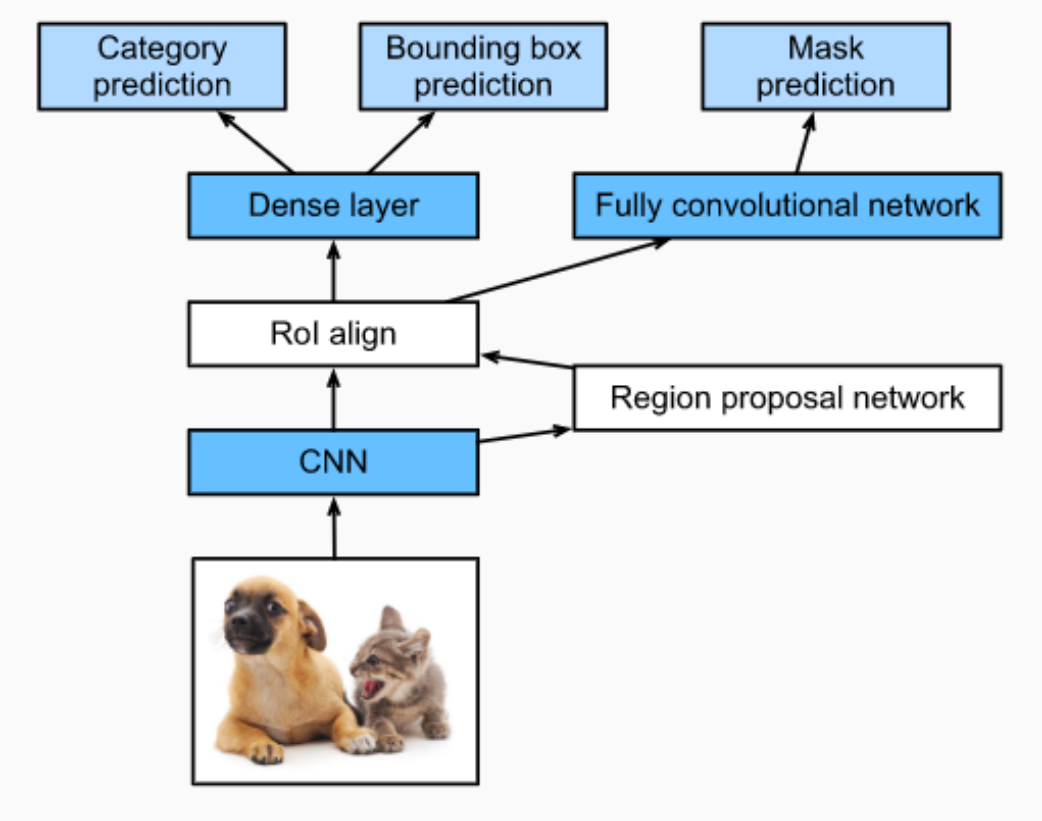
- Abstract
- Semantic Segmentation
- Instance Segmentation
- General Architecture
- Mask R-CNN RoI Alignment
- Mask R-CNN Network architecture

Mask R-CNN

https://www.youtube.com/watch?v=4tkgOzQ9yyo [9:34]

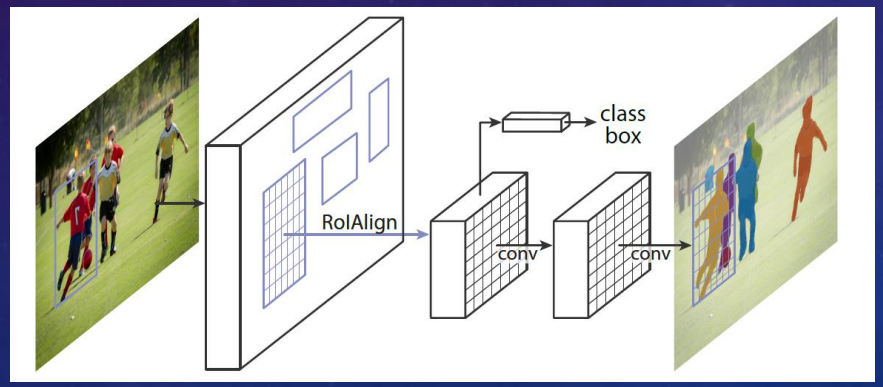
Mask R-CNN

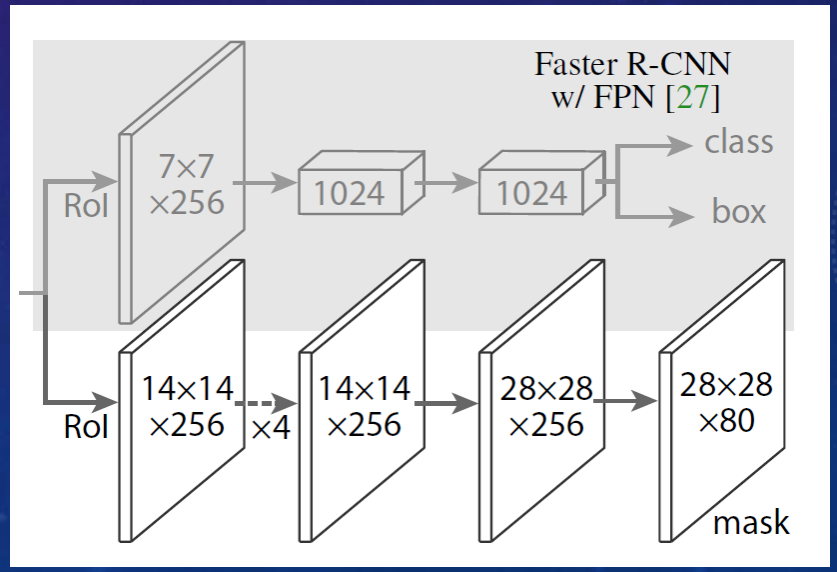
If training data is labeled with the pixel-level positions of each object in an image, a Mask R-CNN model can effectively use these detailed labels to further improve the precision of object detection.



Mask R-CNN

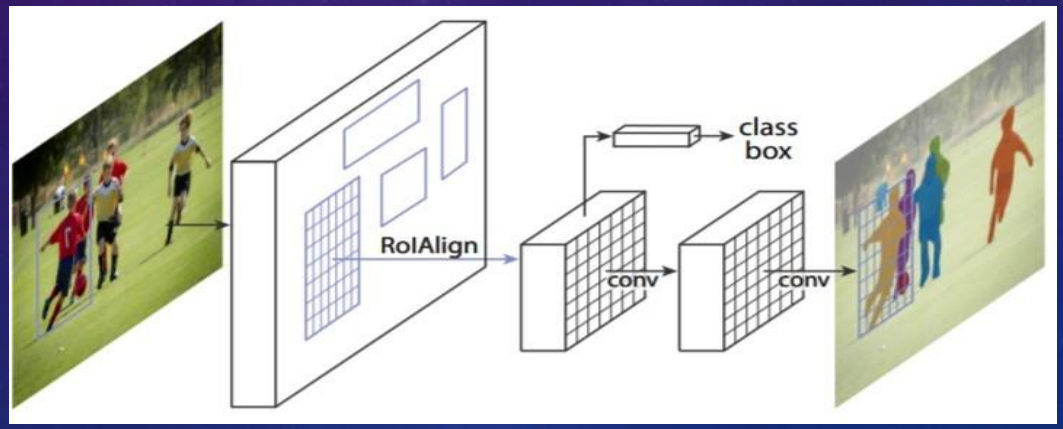
- Mask R-CNN, extends Faster R-CNN by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression
- The mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner.
- RoI Alignment: use bilinear interpolation to compute the exact values of the input features at four regularly sampled locations in each RoI bin.





Abstract

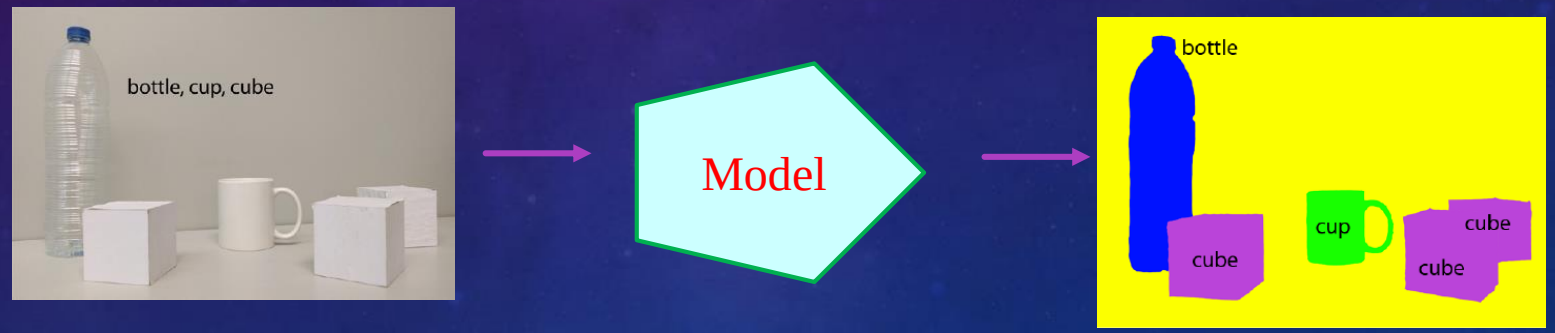
- Our approach efficiently detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance.
- Mask R-CNN extends Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition.



Mask R-CNN is simple to train and adds only a small overhead to Faster R-CNN, running at 5fps.

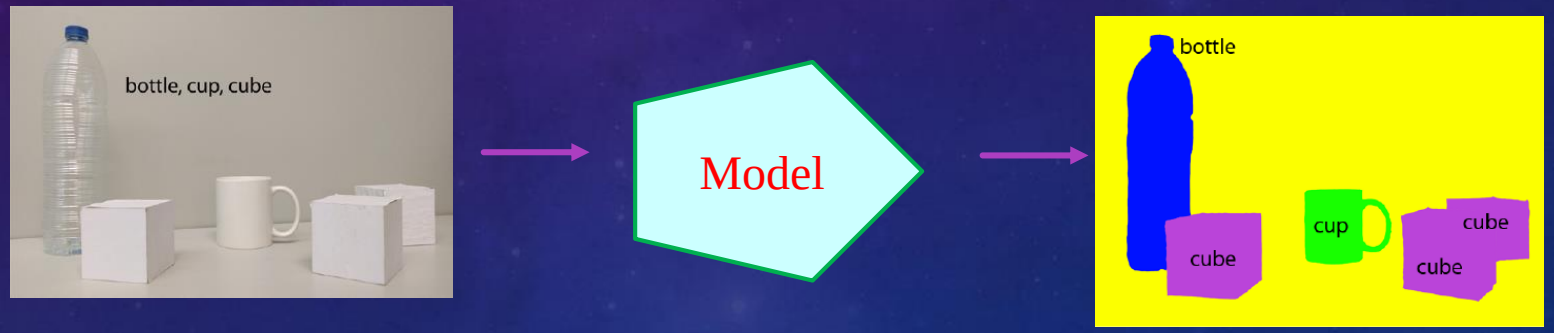
Semantic Segmentation

- Semantic segmentation refers to the process of linking each pixel in an image to a class label.
- These labels could include a person, car, flower, piece of furniture, etc., just to mention a few. We can think of semantic segmentation as image classification at a pixel level.



Instance Segmentation

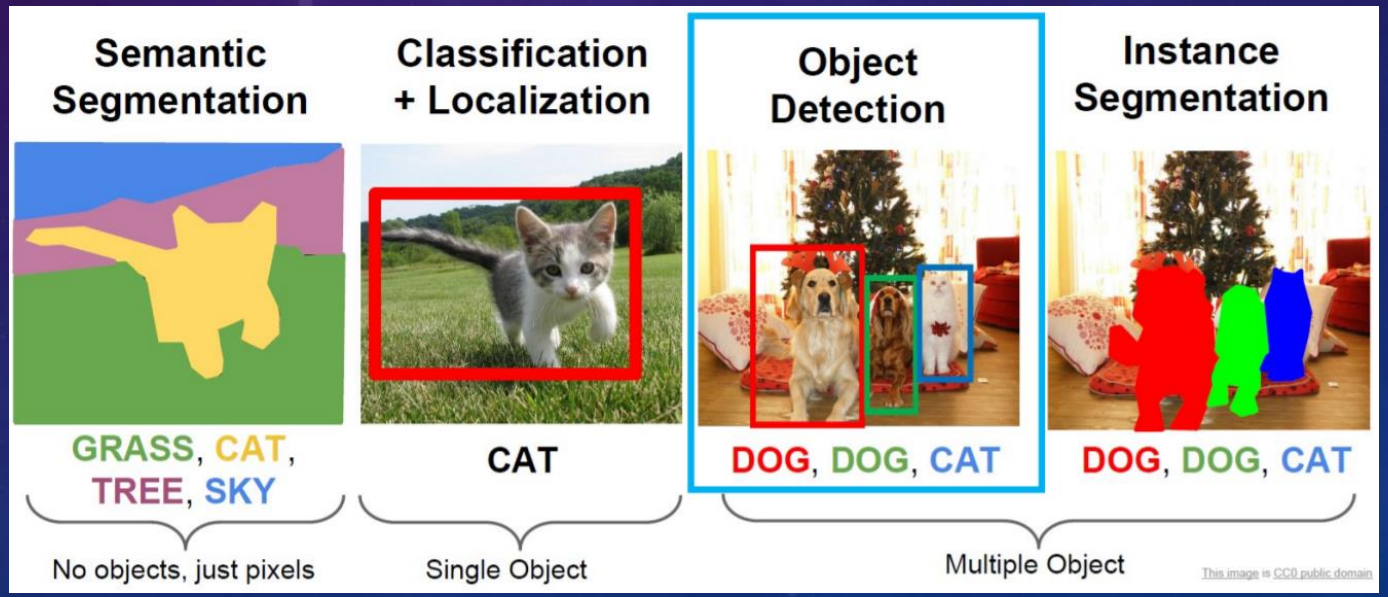
- For each pixel *I*, predict semantic label *l* and instance id *z*
- Instance segmentation = semantic segmentation + distinguished instances per class
- Closer to real scene understanding



Instance Segmentation

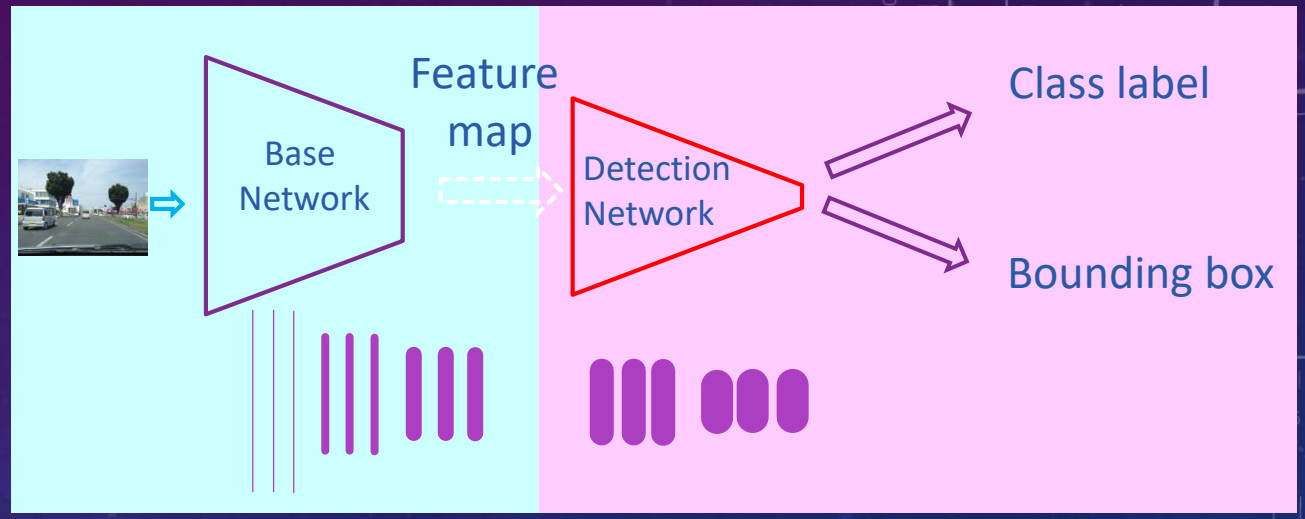
Instance segmentation combines elements from the classical computer vision tasks of object detection and segmentation.

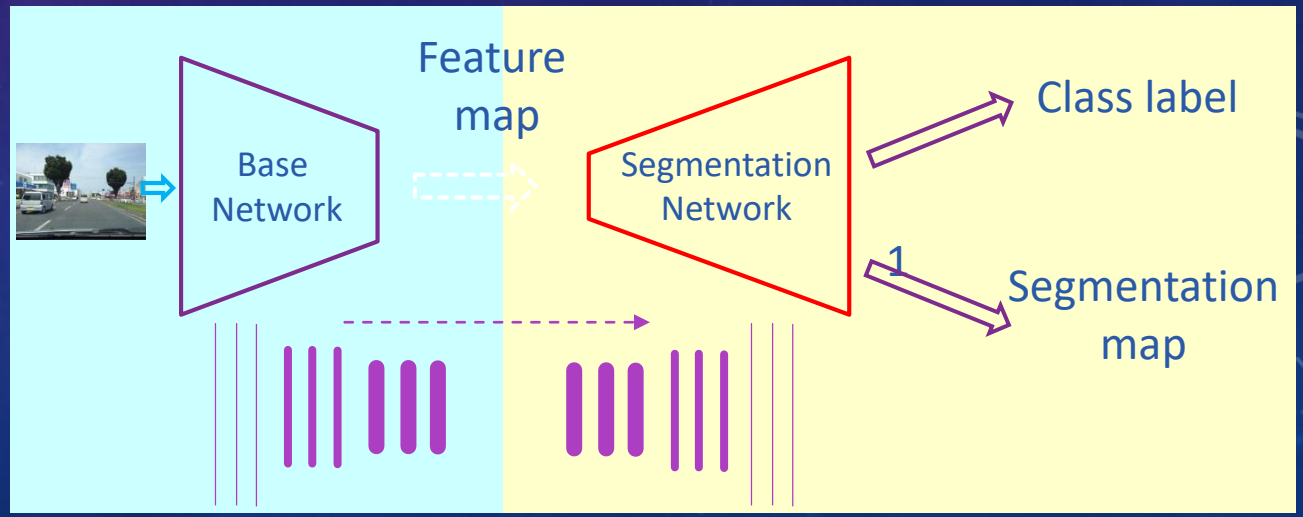
- Object detection is to classify individual objects and localize each with a bounding box
- Semantic segmentation, where the goal is to classify each pixel into a fixed set of categories without differentiating object instances.



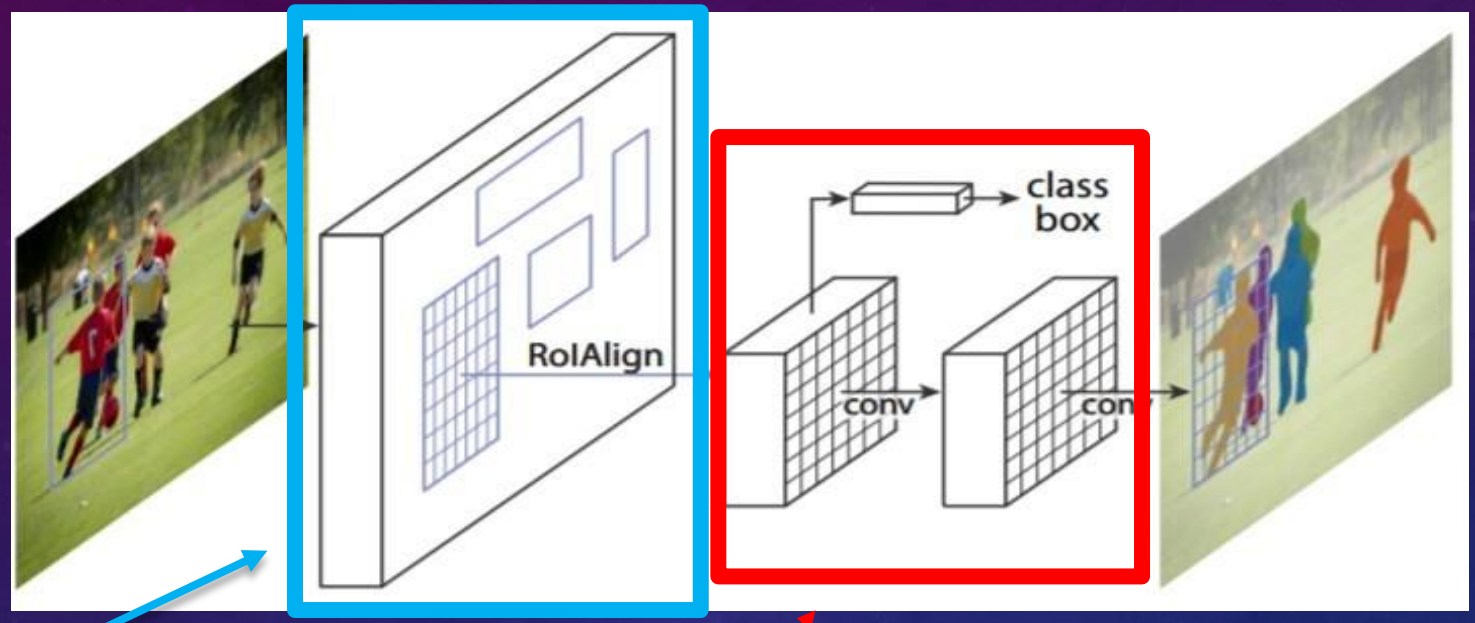
General Architecture

- Object Detection
  - Height and width decrease
  - Channel increase
  - Base network is pre-trained model
  - Output is a list of scalars or vectors
- Segmentation
  - Height and width decrease, then increase again.
  - Channel increases and then decreases
  - Base network is pre-trained model
  - Outputs are map and a lists of scalar.





Mask R-CNN Contribution



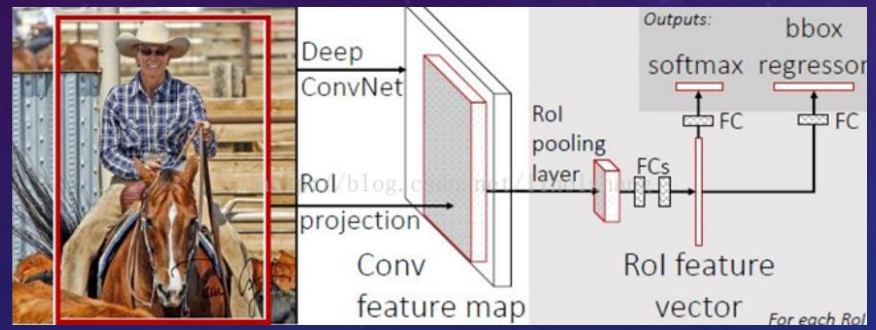
1. To fix the mis-alignment, we propose a simple, quantization-free layer, called RoI Alignment, that faithfully preserves exact spatial locations.

2. Adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression.

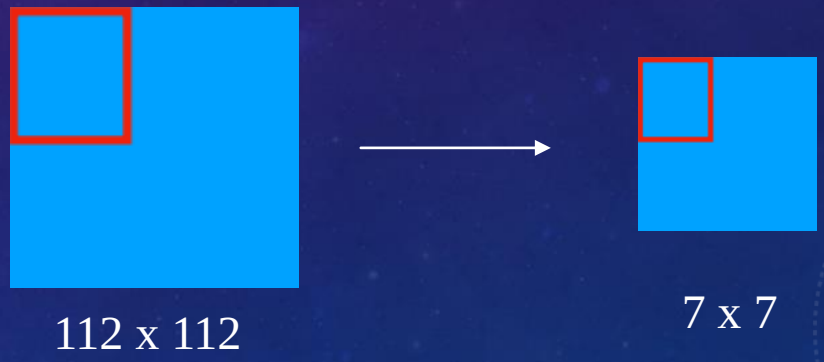
Mask R-CNN

We propose an RoI Alignment layer that removes the harsh quantization of RoI Pooling, properly aligning the extracted features with the input. RoI Alignment improves mask accuracy by relative 10% to 50%, showing bigger gains under stricter localization metrics

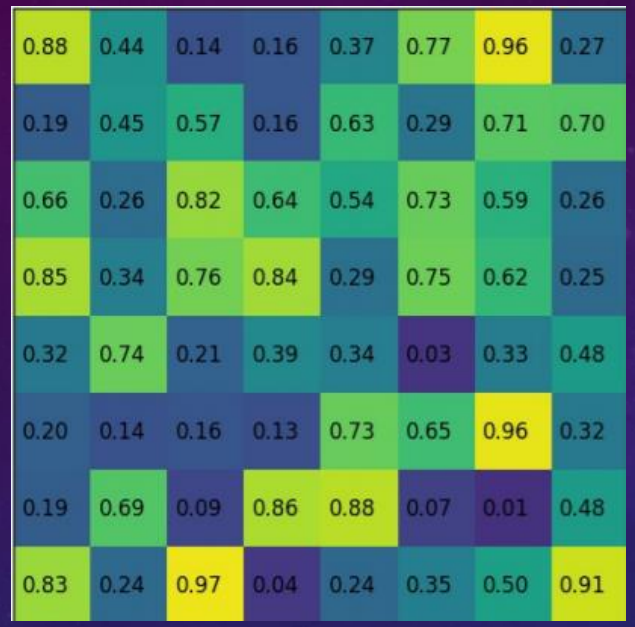
- Previous works - RoIPool



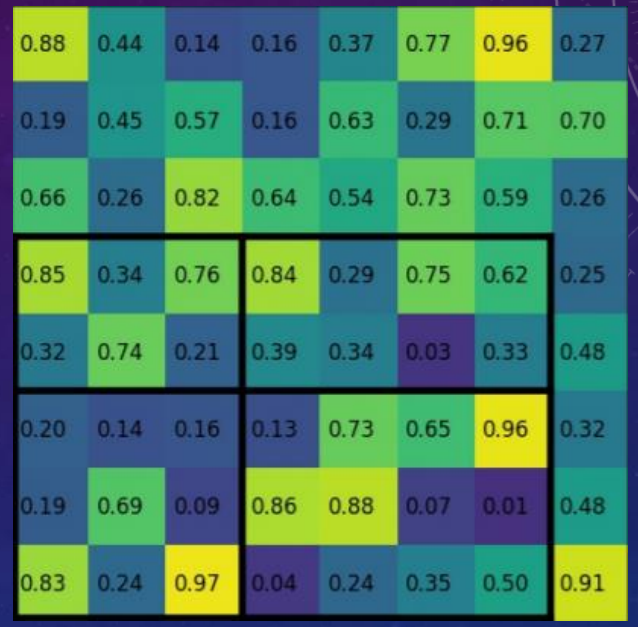
Max-pooling (Input is rounded of)



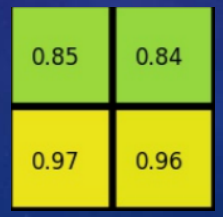
Faster R-CNN ROI Pooling



Input activation

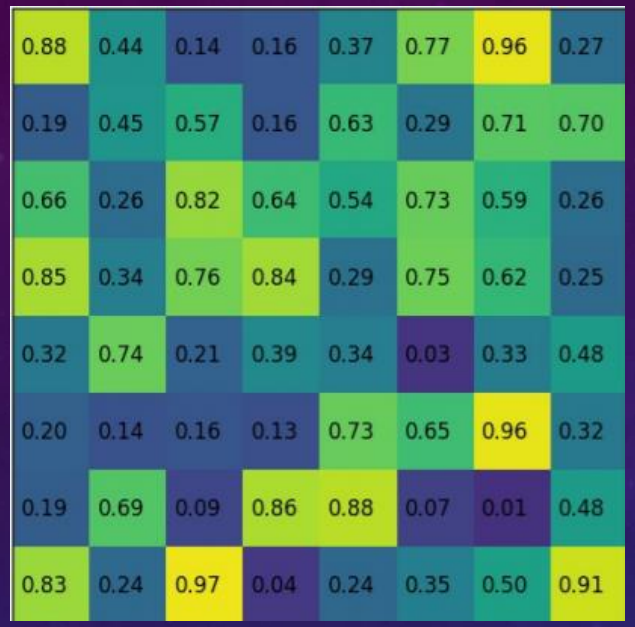


Region projection and pooling sections



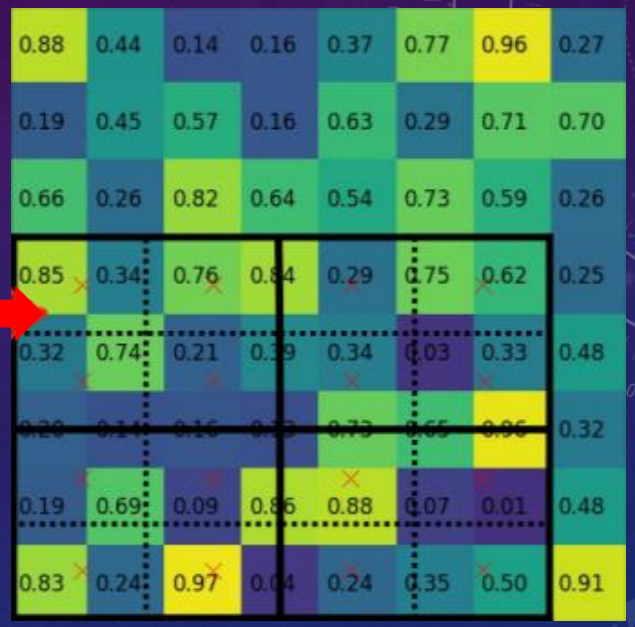
Max pooling output

Mask R-CNN RoI Alignment



Input activation

Region projection and pooling sections

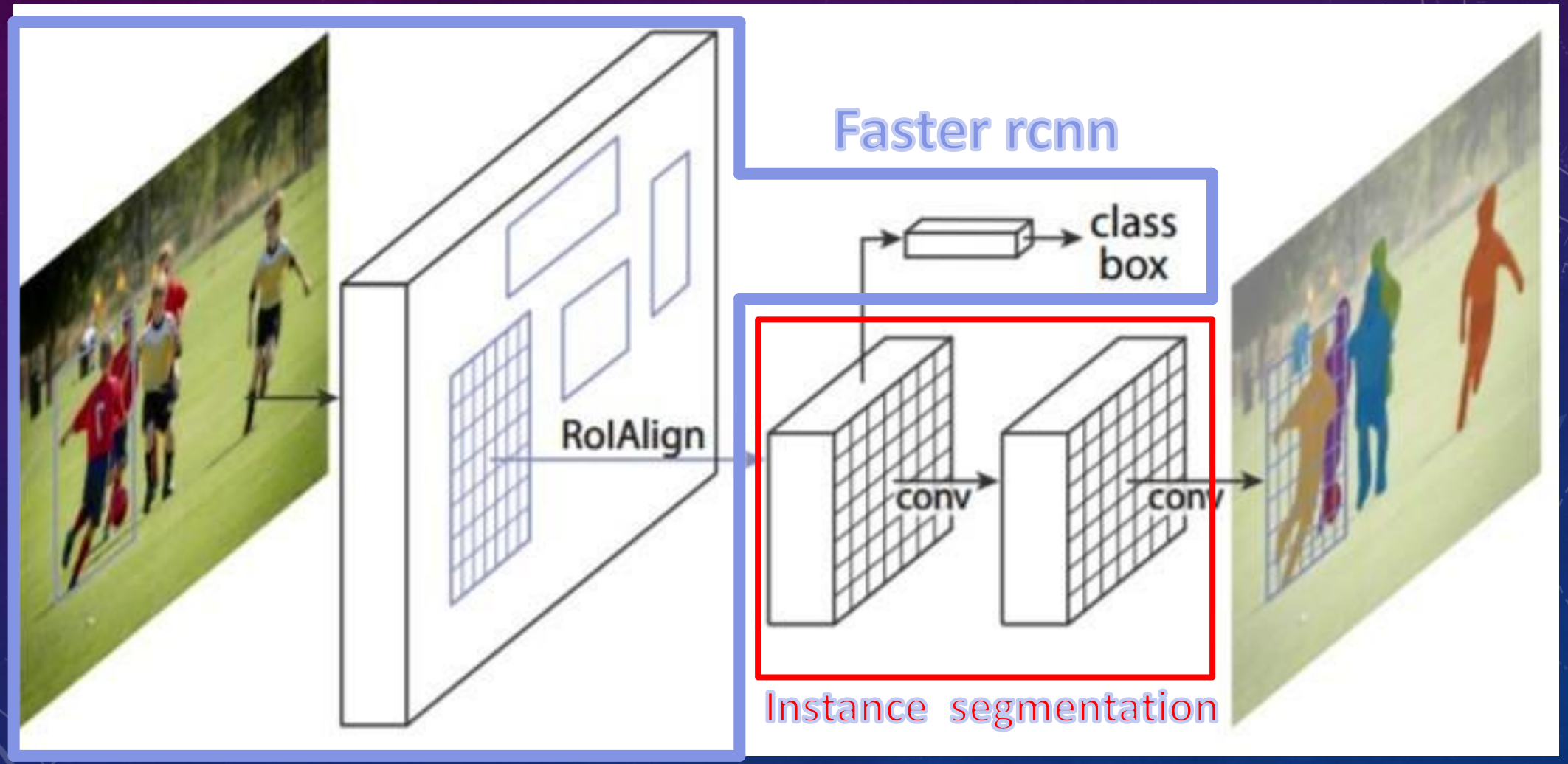


Sampling locations

Max pooling output

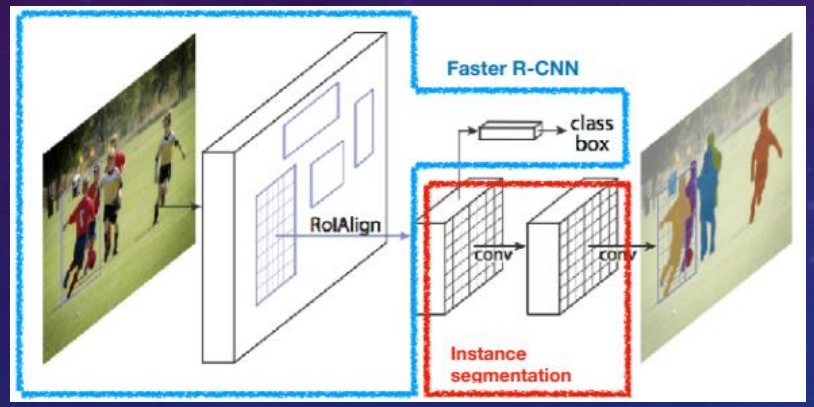
Bilinear interpolated values

Mask R-CNN Architecture



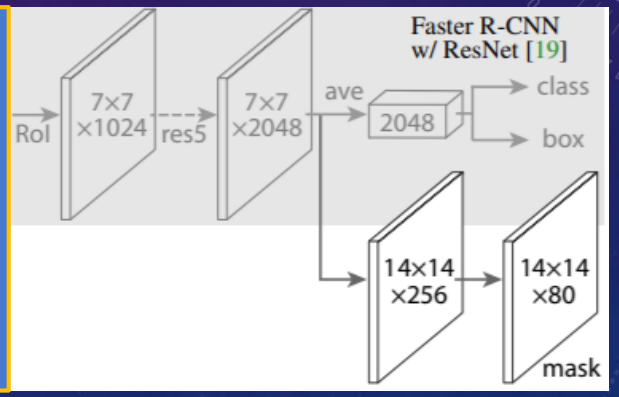
Mask R-CNN Network Architecture

- Mask R-CNN extends Faster R-CNN by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression

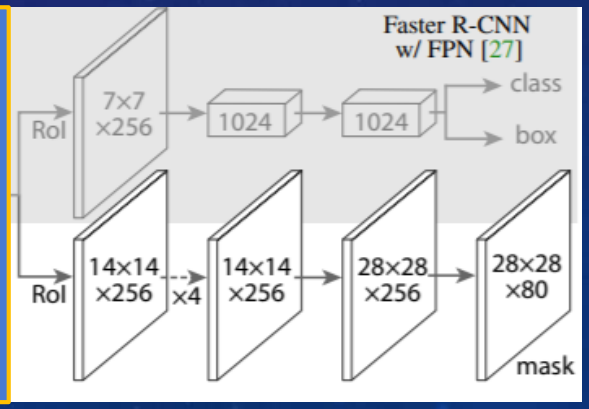


Faster R-CNN + Instance segmentation

ResNet or ResNeXt



FPN (Feature Pyramid Network)



Loss Function

$$L = L_{cls} + L_{reg} + L_{mask}$$

- L_{cls}, L_{reg} is the same in the faster R-CNN

Loss Function

$$L_{mask} = -\frac{1}{m^2}\sum_{1\le i,j\le m}\left[y_{i,j}\log \hat{y}_{i,j}^{k} + \left(1 - y_{i,j}\right)\log\left(1 - \hat{y}_{i,j}^{k}\right)\right]$$

- The mask branch has a K x m x m - dimensional output for each RoI and each class; K classes in total.
- L_{mask} is defined as the average binary cross-entropy loss, only including k-th mask if the region is associated with the ground truth class k

Summary

•R-CNN model selects several proposed regions and uses a CNN to perform forward computation and extract the features from each proposed region. It then uses these features to predict the categories and bounding boxes of proposed regions.

•Fast R-CNN improves on the R-CNN by only performing CNN forward computation on the image as a whole. It introduces an RoI pooling layer to extract features of the same shape from RoIs of different shapes.

•Faster R-CNN replaces the selective search used in Fast R-CNN with a region proposal network. This reduces the number of proposed regions generated, while ensuring precise object detection.

•Mask R-CNN uses the same basic structure as Faster R-CNN, but adds a fully convolution layer to help locate objects at the pixel level and further improve the precision of object detection.

Contents

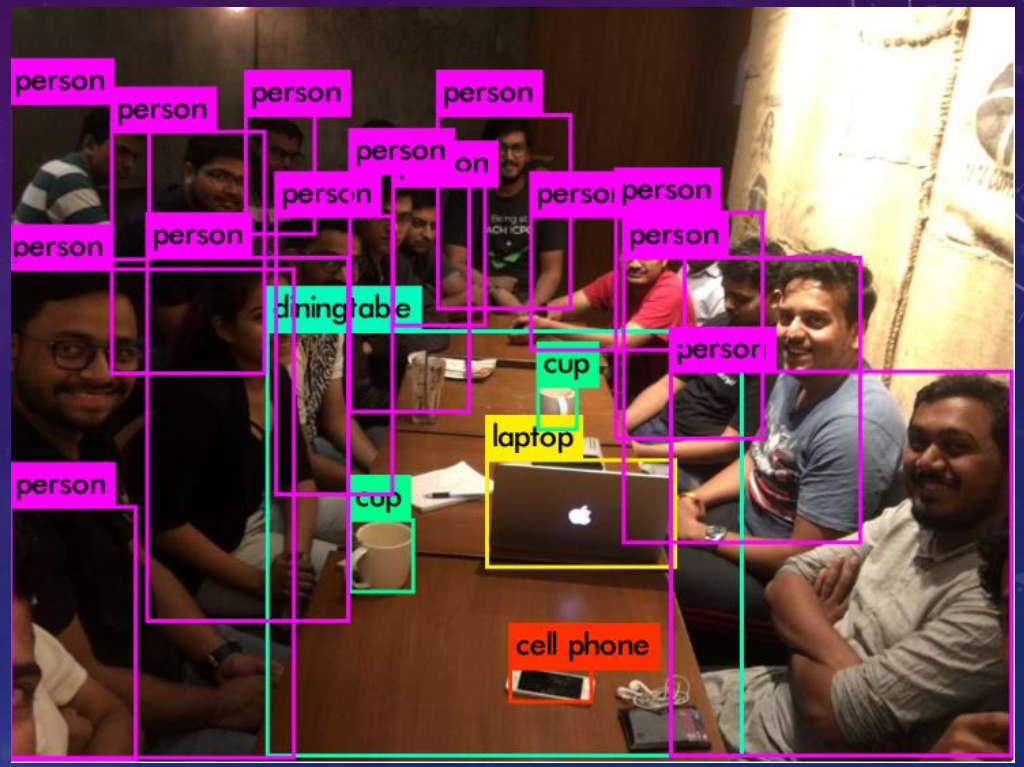
- What is Yolo?
- What is Darknet and how to use it?
- K-means Clustering
- Label-Img tool
- Data Preprocessing

YOLO Object Detection (TensorFlow tutorial)

https://www.youtube.com/watch?v=4eIBisqx9_g[21:50]

What is YOLO ?

- YOLO is an object detection algorithm. It detects multiple objects and creates a bounding box around them.
- YOLO frames object detection as a regression problem instead of a classification problem.
- YOLO brings a unified neural network architecture to the table, single architecture which does bounding box prediction and also gives out class probabilities.



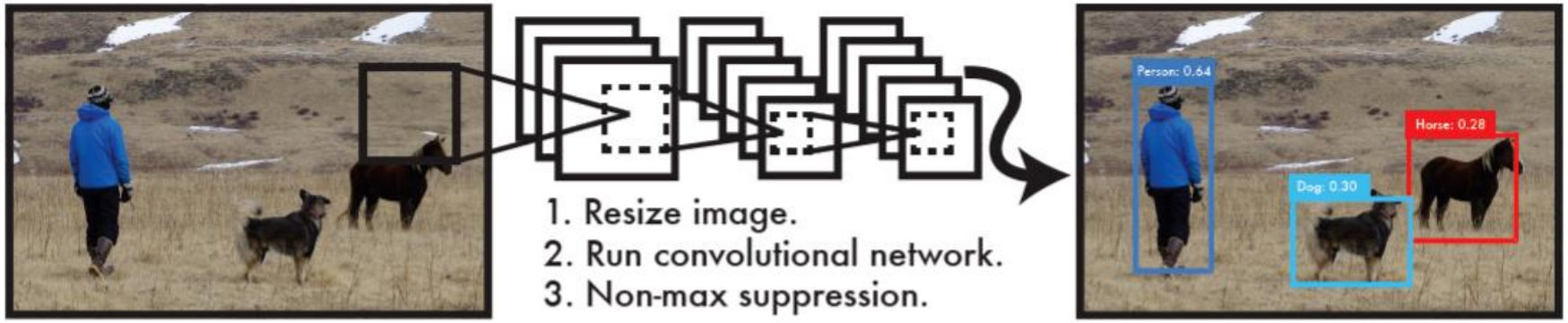
Detection System

- Processing images with YOLO is simple and straightforward

1.Resizes the input image to 488 x 488

2.Run a single convolutional network on the image

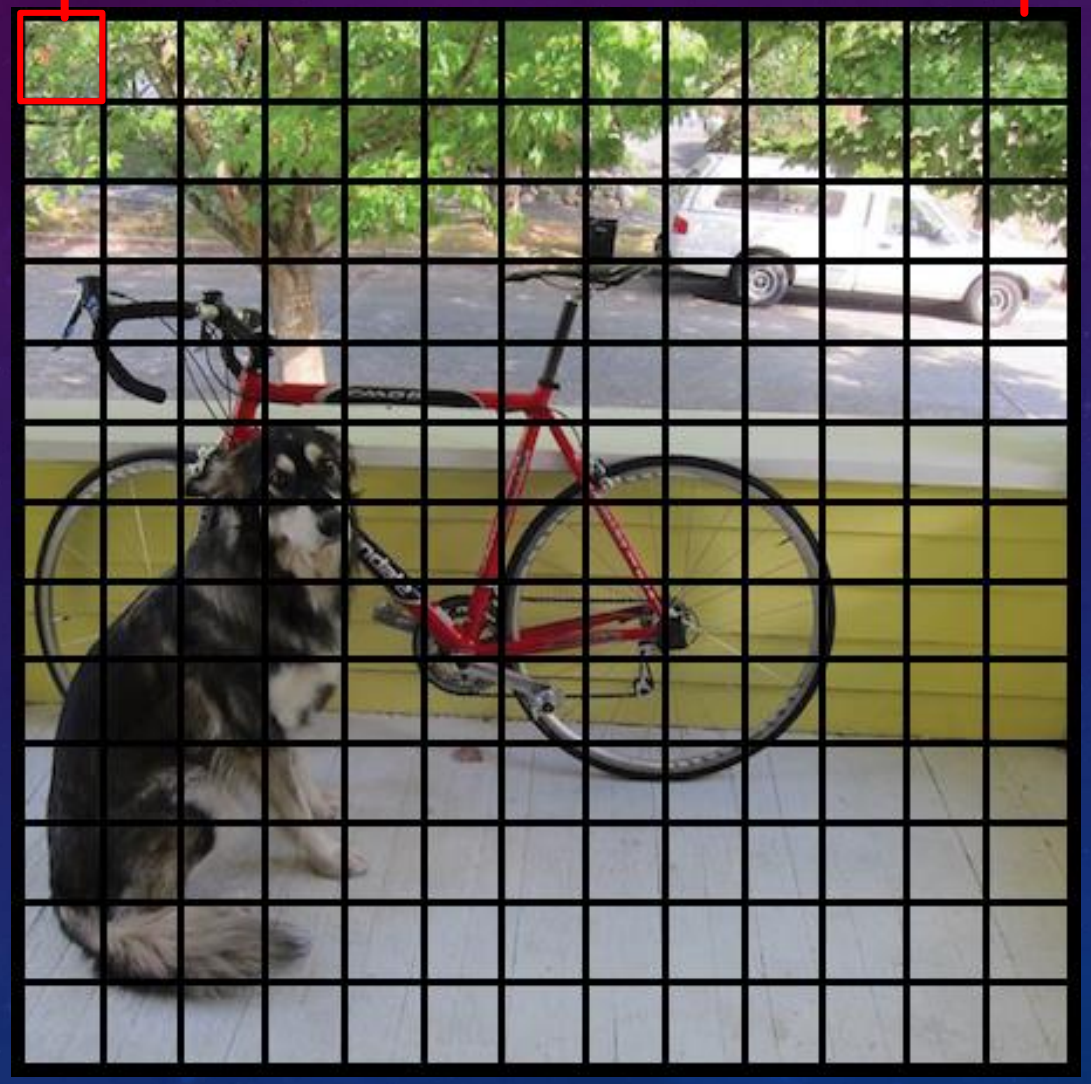
3.Thresholds the resulting detections by the model's confidence



Grid Cell

13

One grid cell



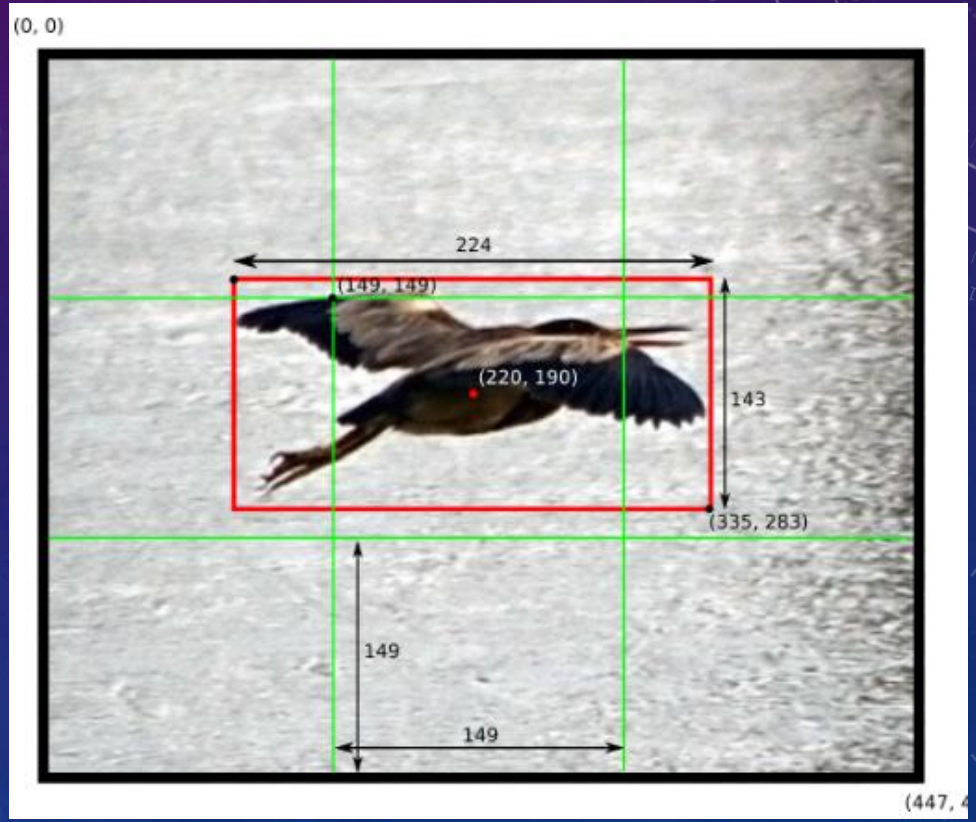
Resize picture to 448 x 448 and a grid cell size is 448/13

YOLO Bounding Box

Each grid cell predicts bounding. The bounding box prediction has 5 components: (*x*, *y*, *w*, *h*, *confidence*).

- (*x*, *y*) coordinates represent the center of the box, relative to the grid cell location. These coordinates are normalized to fall between 0 and 1.

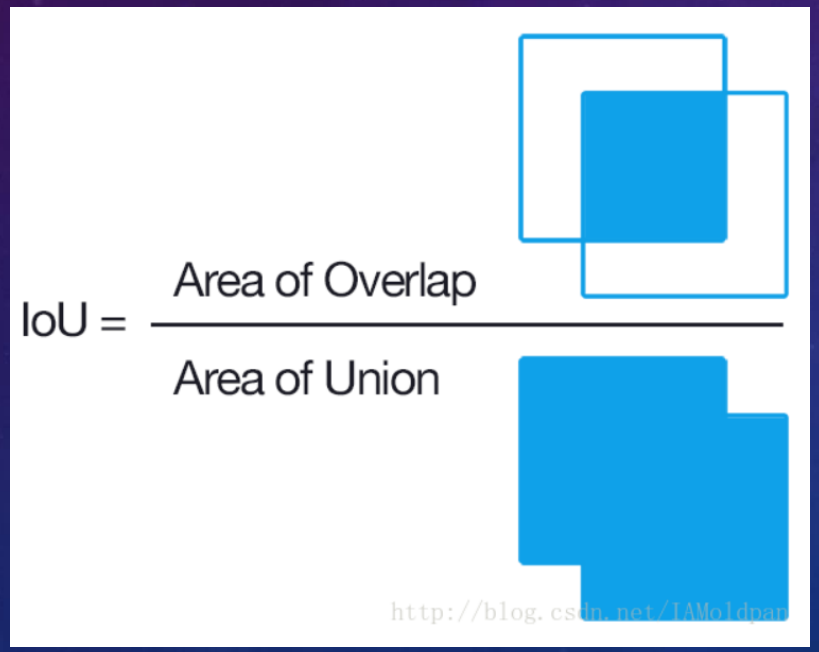
- (*w*, *h*) coordinates represent the bounding box's weight and height and box dimensions are also normalized to [0, 1], relative to the image size.

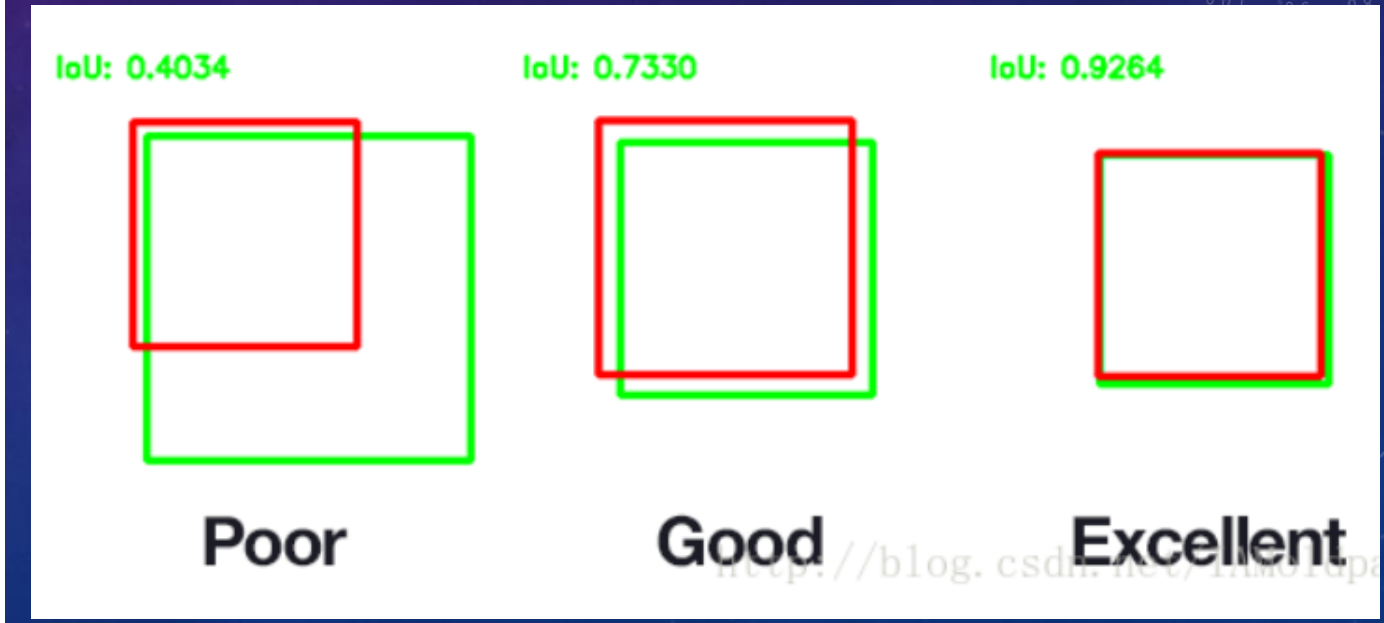


YOLO Bounding Box

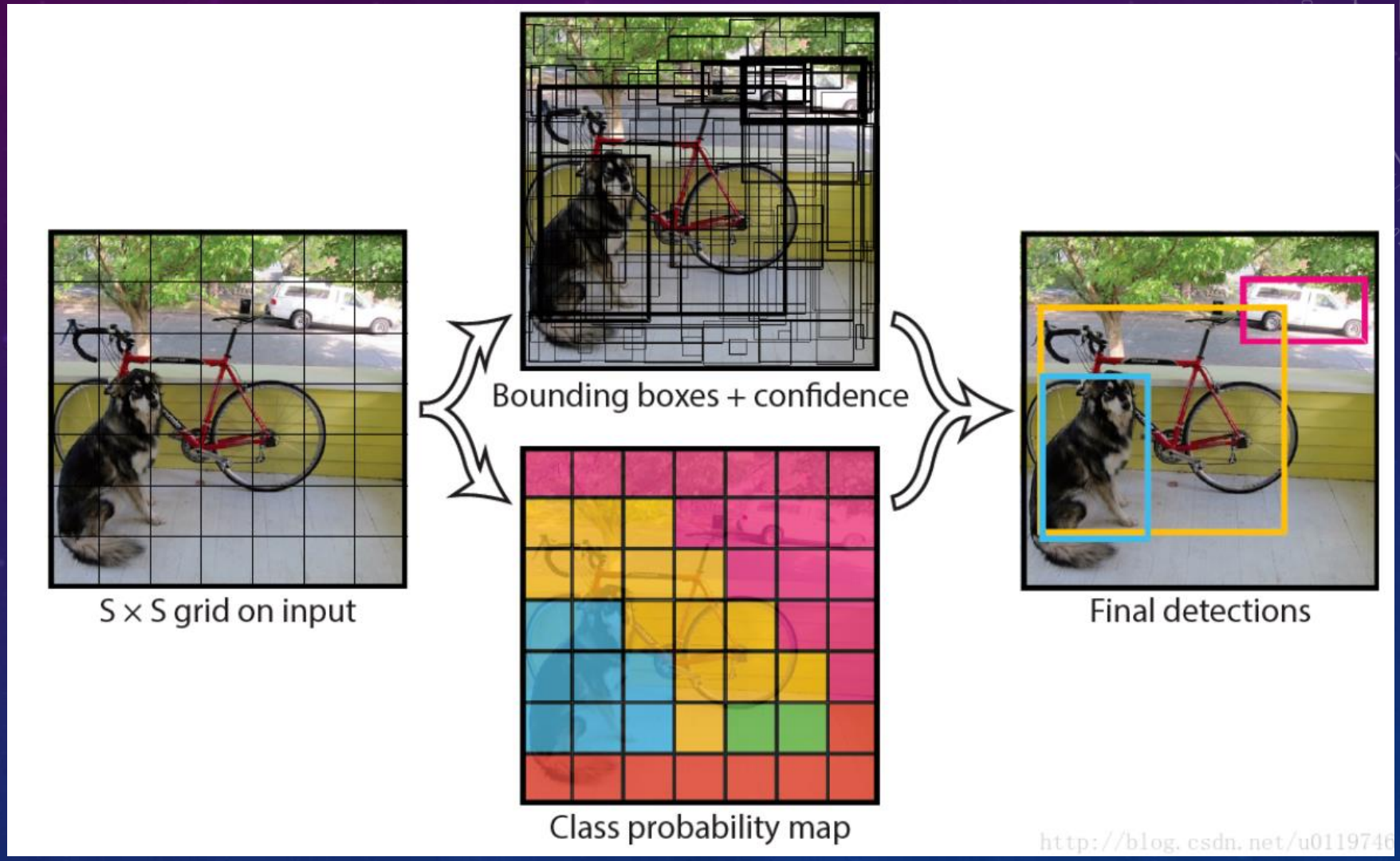
- Formally we define confidence as *Pr*(Object) * *IOU*(pred, truth) . If no object exists in that cell, the confidence score should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.

- IOU





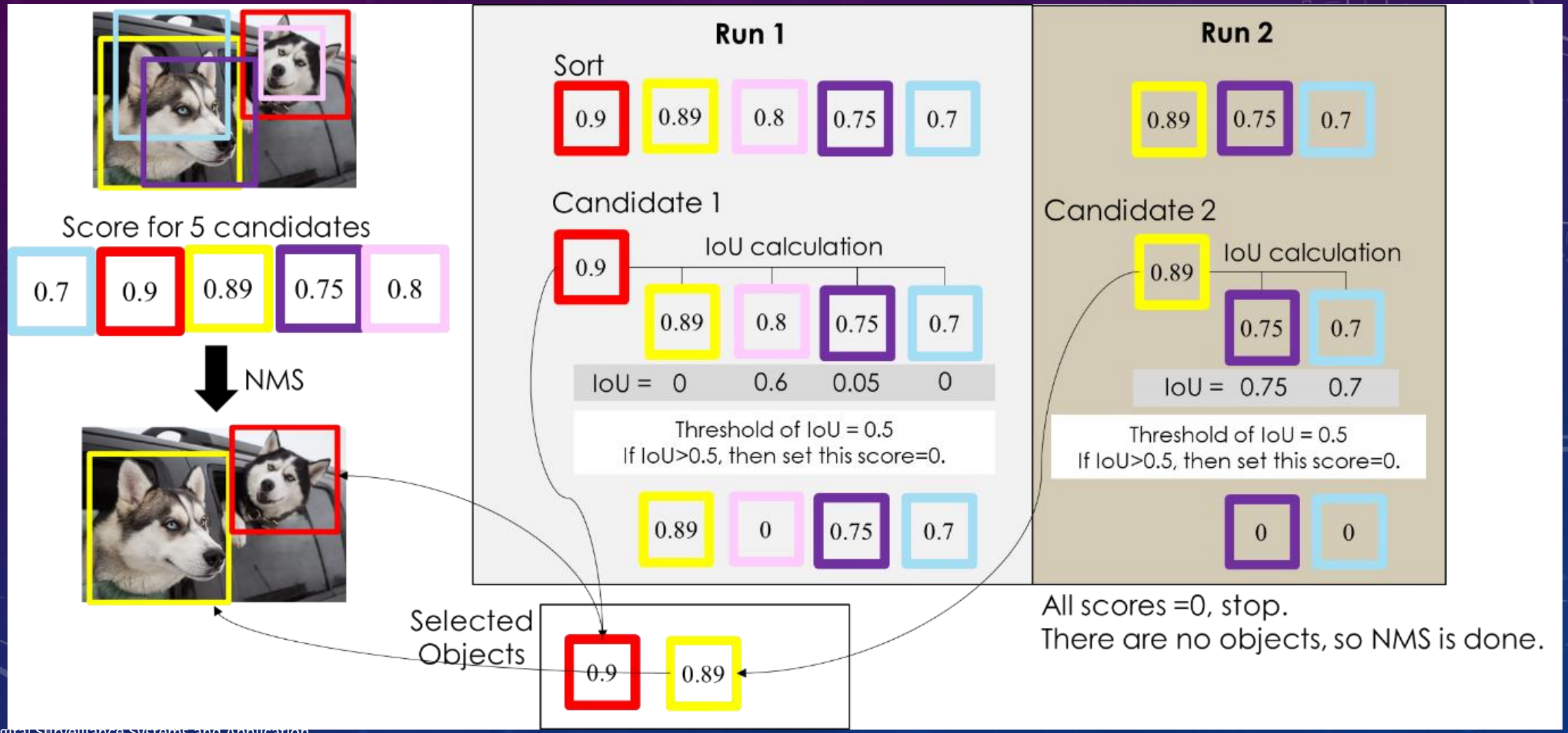
YOLO Object Detection Process



Non-Maximum Suppression (NMS)

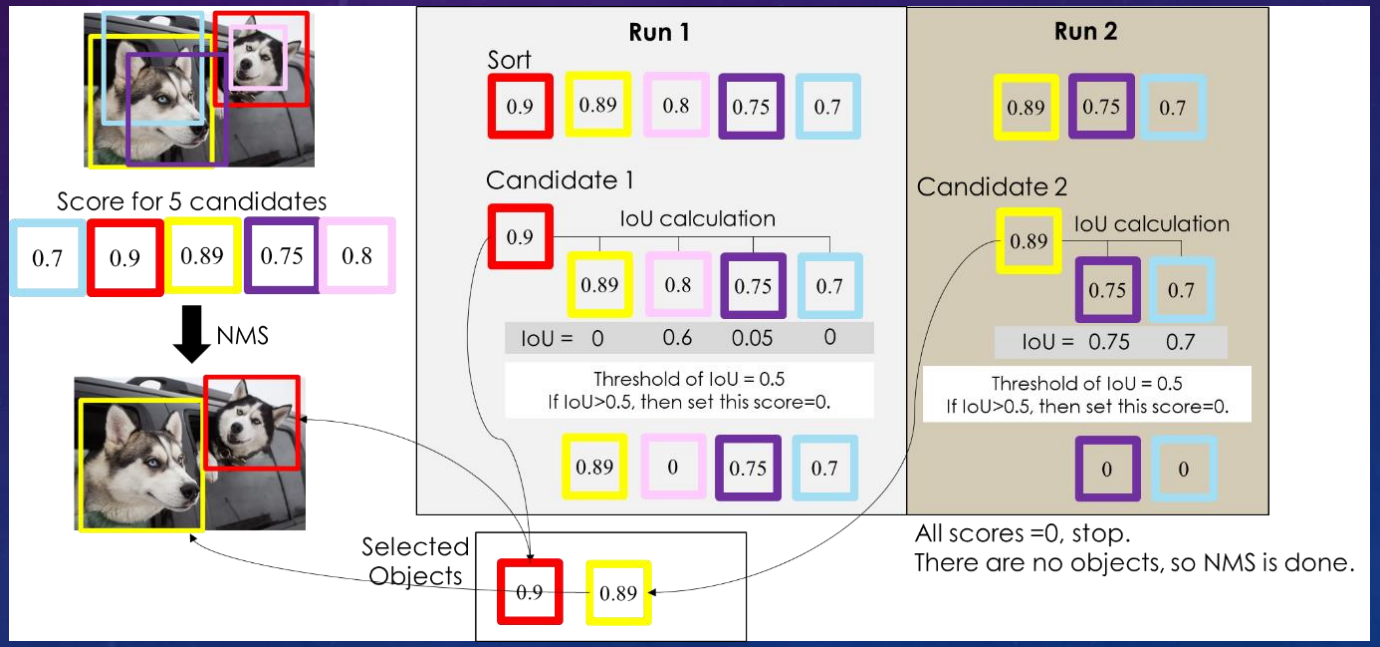
- Extracting a target detecting the highest score window
- For example, in the detection of pedestrians, the extracted features sliding window, after classification by the classifier, each window will get a score. But the sliding window can cause a lot of windows and other windows or circumstances exist that contains most of the cross. Then you need to use NMS to select those neighborhoods where the highest score (the maximum probability of a pedestrian), and inhibits those with low scores window.

Example: Non-Maximum Suppression (NMS)



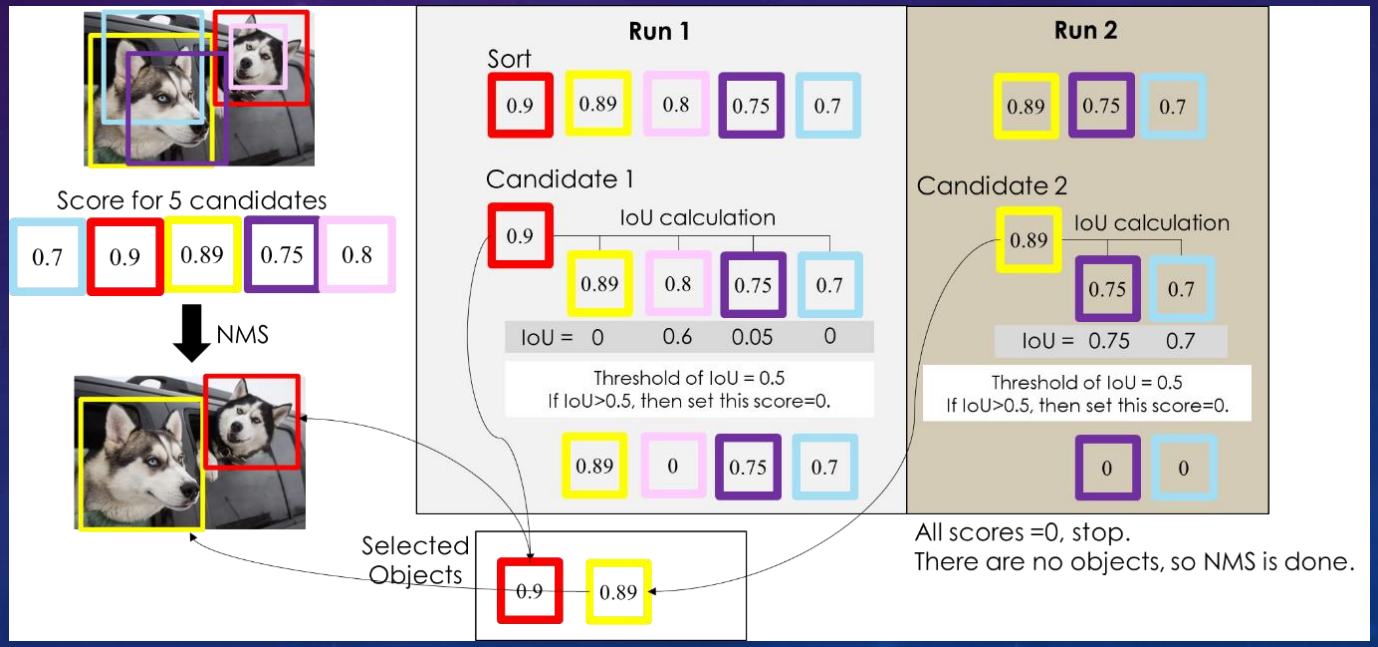
Example: Non-Maximum Suppression (NMS)

- Run 1: First BBox confidence in accordance with the degree of ordering, the highest degree of confidence BBox (red) will be elected to "determine the set of objects" within other BBox will see this step to select the best BBox be IoU calculation, if the pink IoU we set greater than 0.5 to 0.6, so the pink BBox confidence is set to 0.



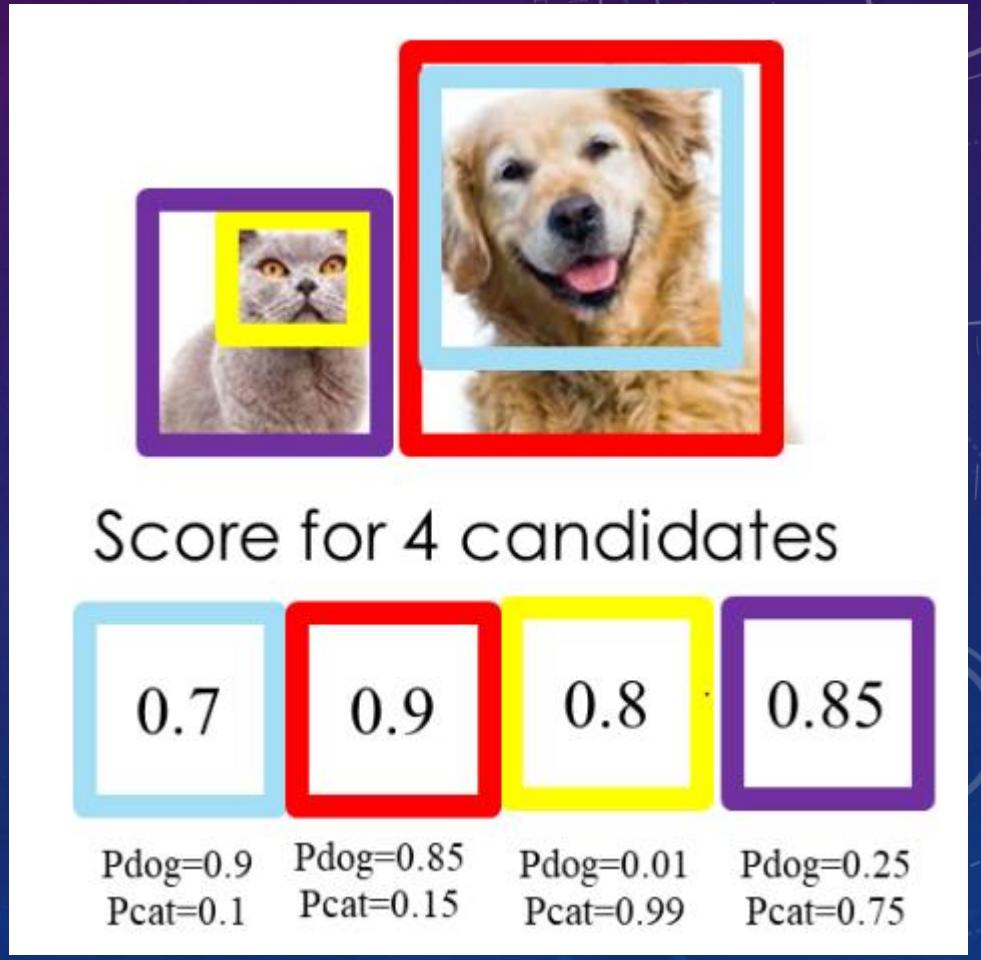
Example: Non-Maximum Suppression (NMS)

- Run 2: 0 irrespective of the degree of confidence and have "a set of objects is determined" in BBox, and the remaining items continue to be elected to the greatest degree of confidence BBox, this BBox (yellow) thrown into "the object is to determine the set of" rest BBox and the maximum degree of confidence selected Run2 BBox calculation IOU, other BBox are larger than 0.5, the other is set to 0 BBox confidence. "OK is a collection of articles" = {red BBox; yellow BBox}



Exercise 1 – Non-Maximum Suppression (NMS)

- Please do the same steps from the example and decide the bounding box of each objects in the figure.
- Please write down result in MS Word to Moodle



Loss Function

- Classification loss

  If an object is detected, the classification loss at each cell is the squared error of the class conditional probabilities for each class:

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

Where

1_i^{obj}=1 if an object appears in cell i, otherwise 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i.

Loss Function

- Localization loss

  The localization loss measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object.

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2]$$

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

Where

1_i^{obj}=1 if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

λ_{coord} increase the weight for the loss in the boundary box coordinates.

Loss Function

- Confidence loss

  If an object is detected in the box, the confidence loss (measuring the objectness of the box) is:

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B} 1_{ij}^{obj}(C_i - \widehat{C}_i)^2$$

  $\widehat{C}_i$ is the box confidence score of the box j in cell i.

  1_i^{obj}=1 if the j th boundary in cell i is responsible for detection the object, otherwise 0.

  If an object is not detected in the box, the confidence loss is:

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

  1_{ij}^{noobj} is the complement of 1_{ij}^{obj}.

  $\widehat{C}_i$ is the box confidence score of the box j in cell i.

  λ_{noobj} weights down the loss when detecting background.

Loss Function

- Final loss

The final loss adds localization, confidence and classification losses together.:

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

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

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C_i})^2$$

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

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

YOLOv2 Improvement

- Batch Normalization (BN)
- High Resolution Classifier
- Convolutions with Anchor Boxes
- Dimension Clusters
- Direct Location Prediction
- Fine-Grained Features

YOLOv2 Improvement

- Batch Normalization (BN)
  - BN is used on all convolutional layers in YOLOv2.
  - 2% improvement in mAP.
- High Resolution Classifier
  - After trained by 224×224 images, YOLOv2 also uses 448×448 images for fine-tuning the classification network for 10 epochs on ImageNet.
  - 4% increase in mAP.

YOLOv2 Improvement

- Convolutions with Anchor Boxes
  - YOLOv2 removes all fully connected layers and uses anchor boxes to predict bounding boxes.
  - One pooling layer is removed to increase the resolution of output.
  - And 416×416 images are used for training the detection network now.
  - And 13×13 feature map output is obtained, i.e. 32× downsampled.
  - Without anchor boxes, the intermediate model got 69.5% mAP and recall of 81%.
  - With anchor boxes, 69.2% mAP and recall of 88% are obtained. Though mAP is dropped a little, recall is increased by large margin.

YOLOv2 Improvement

- Dimension Clusters
  - The sizes and scales of Anchor boxes were pre-defined without getting any prior information, just like the one in Faster R-CNN.
  - Using standard Euclidean distance based k-means clustering is not good enough because larger boxes generate more error than smaller boxes
  - YOLOv2 uses k-means clustering which leads to good IOU scores:

$$d(\text{box}, \text{centroid}) = 1 - \text{IOU}(\text{box}, \text{centroid})$$

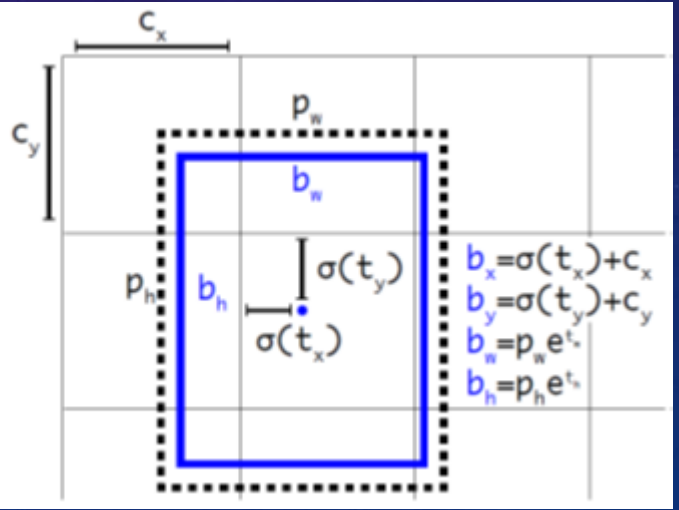
Cluster IOU

  - k = 5 is the best value with good tradeoff between model complexity and high recall.
  - IOU based clustering with 5 anchor boxes (61.0%) has similar results with the one in Faster R-CNN with 9 anchor boxes (60.9%).
  - IOU based clustering with 9 anchor boxes got 67.2%.

YOLOv2 Improvement

- Direct Location Prediction
  - YOLOv1 does not have constraints on location prediction which makes the model unstable at early iterations. The predicted bounding box can be far from the original grid location.
  - YOLOv2 bounds the location using logistic activation σ, which makes the value fall between 0 to 1:

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$



YOLOv2 Improvement

- Fine-Grained Features
  - The 13×13 feature map output is sufficient for detecting large object.
  - To detect small objects well, the 26×26×512 feature maps from earlier layer is mapped into 13×13×2048 feature map, then concatenated with the original 13×13 feature maps for detection.
- Multi-Scale Training
  - For every 10 batches, new image dimensions are randomly chosen.
  - The image dimensions are {320, 352, …, 608}.
  - The network is resized and continue training.

YOLOv3 Improvement

- Class Prediction
  - YOLO applies a softmax function to convert scores into probabilities that sum up to one.
  - YOLOv3 uses multi-label classification. For example, the output labels may be “pedestrian” and “child” which are not non-exclusive. (the sum of output can be greater than 1 now.)
  - YOLOv3 replaces the softmax function with independent logistic classifiers to calculate the likeliness of the input belongs to a specific label. Instead of using mean square error in calculating the classification loss, YOLOv3 uses binary cross-entropy loss for each label.
- Bounding box prediction & Cost function calculation
- Feature Pyramid Networks (FPN) like Feature Pyramid

YOLOv3 Improvement

- Bounding box prediction & cost function calculation
  - YOLOv3 predicts an objectness score for each bounding box using logistic regression.
  - YOLOv3 changes the way in calculating the cost function. If the bounding box prior (anchor) overlaps a ground truth object more than others, the corresponding objectness score should be 1.
  - For other priors with overlap greater than a predefined threshold (default 0.5), they incur no cost.
  - Each ground truth object is associated with one boundary box prior only. If a bounding box prior is not assigned, it incurs no classification and localization lost, just confidence loss on objectness.
  - We use tx and ty (instead of bx and by) to compute the loss.

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
$$

YOLOv3 Improvement

- Feature Pyramid Networks (FPN) like Feature Pyramid
  - YOLOv3 makes 3 predictions per location. Each prediction composes of a boundary box, a objectness and 80 class scores, i.e. $N \times N \times [3 \times (4 + 1 + 80)]$ predictions.
  - YOLOv3 makes predictions at 3 different scales (similar to the FPN):

1. In the last feature map layer.
2. Then it goes back 2 layers back and upsamples it by 2. YOLOv3 then takes a feature map with higher resolution and merge it with the upsampled feature map using element-wise addition. YOLOv3 apply convolutional filters on the merged map to make the second set of predictions.
3. Repeat 2 again so the resulted feature map layer has good high-level structure (semantic) information and good resolution spatial information on object locations.

K-Means

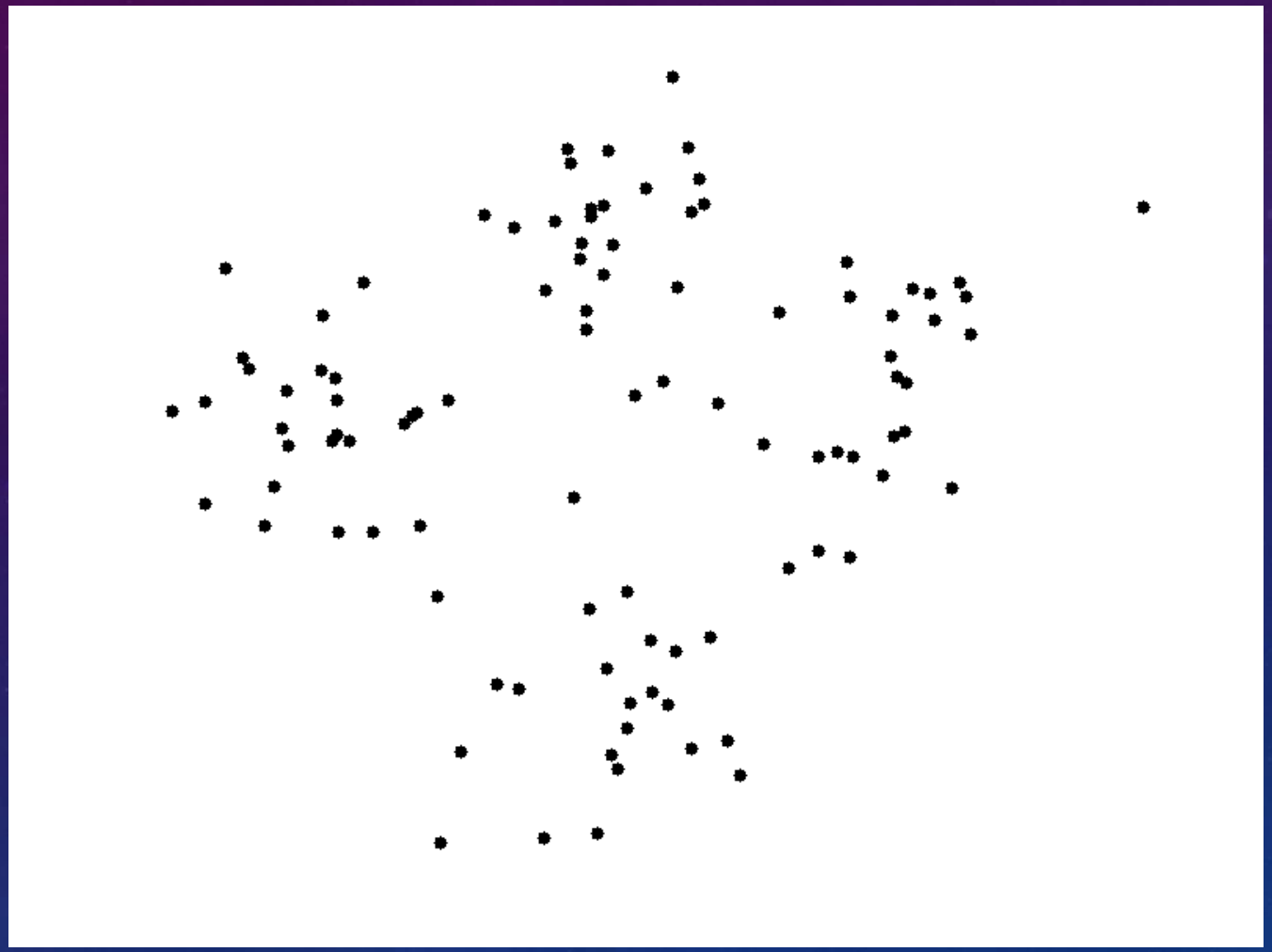
- K-Means is one of the simplest unsupervised learning algorithms that solve the clustering problems.
- The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters).
- The main idea is to define k centers, one for each cluster.

K-Means

The algorithm works as follows:

1. First we initialize k points, called means.
2. We categorize each items to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

K-Means



Exercise 2 – K-Means Practice

1. Please adjust the number of clusters and data by yourself and observe the changes.

- The needed data “k-means.py” is given on Moodle.

Please write down result and your code in MS Word to Moodle

K-Means

```
seed_num = 3
dot_num = 50
```

3 clusters and 50data

```
#initial element
x = np.random.randint(0, 500, dot_num)
y = np.random.randint(0, 500, dot_num)

#initial cluster center
kx = np.random.randint(0, 500, seed_num)
ky = np.random.randint(0, 500, seed_num)



#2 point distance
def dis(x, y, kx, ky):
    return int(((kx-x)**2 + (ky-y)**2)**0.5)
```

```
#Group each element
def cluster(x, k, kx, ky):
    team = []
    for i in range(3):
        team.append([])
    mid_dis = 99999999
    for i in range(dot_num):
        for j in range(seed_num):
            distant = dis(x[i], y[i], kx[j], ky[j])
            if distant < mid_dis:
                mid_dis = distant
                flag = j
        team[flag].append([x[i], y[i]])
        mid_dis = 99999999
    return team
```

→ Number of cluster

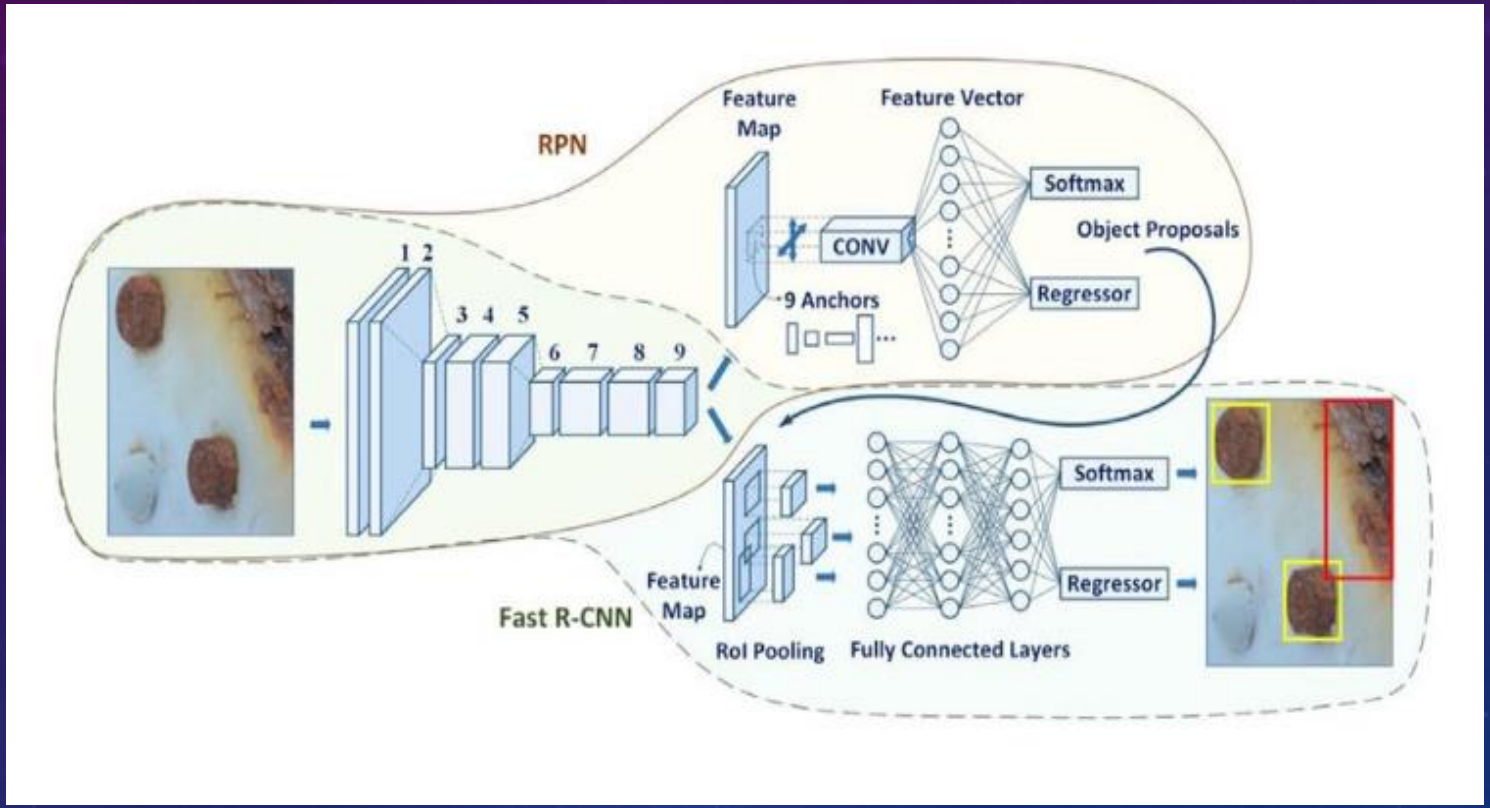
K-Means

```
 #k-means Grouping
def kmeans(x, y, kx, ky, fig):
    team = cluster(x, y, kx, ky)
    nkx, nky = re_seed(team, kx, ky)

    # plot: nodes connect to seeds
    cx = []
    cy = []
    line = plt.gca()
    for index, nodes in enumerate(team):
        for node in nodes:
            cx.append([node[0], nkx[index]])
            cy.append([node[1], nky[index]])
        for i in range(len(cx)):
            line.plot(cx[i], cy[i], color='r', alpha=0.3)
        cx = []
        cy = []
```

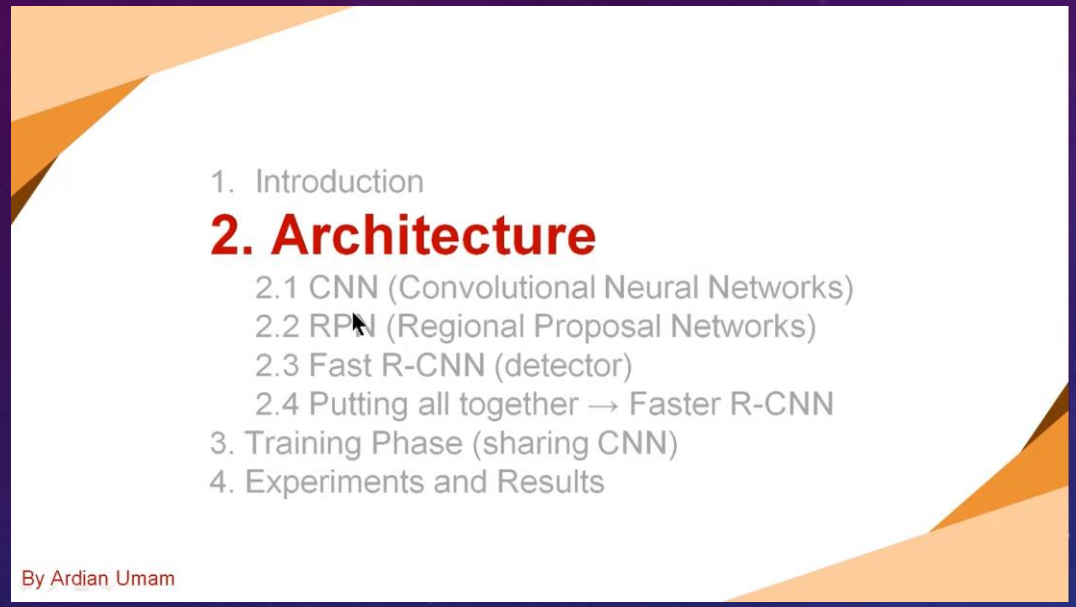
Appendix

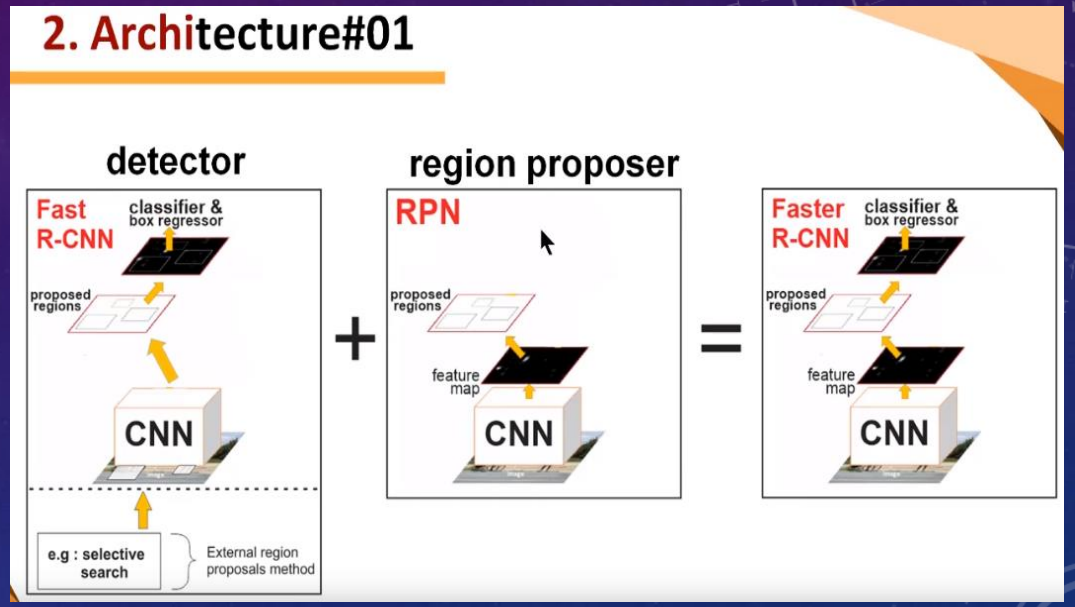
Introduction to How Faster R-CNN, Fast R-CNN and R-CNN Works



https://www.youtube.com/watch?v=v5bFVbQvFRk&list=PLkRkKTC6HZMzp28TxR_fJYZ-K8Yu3EQw0&index=1 [8:40]

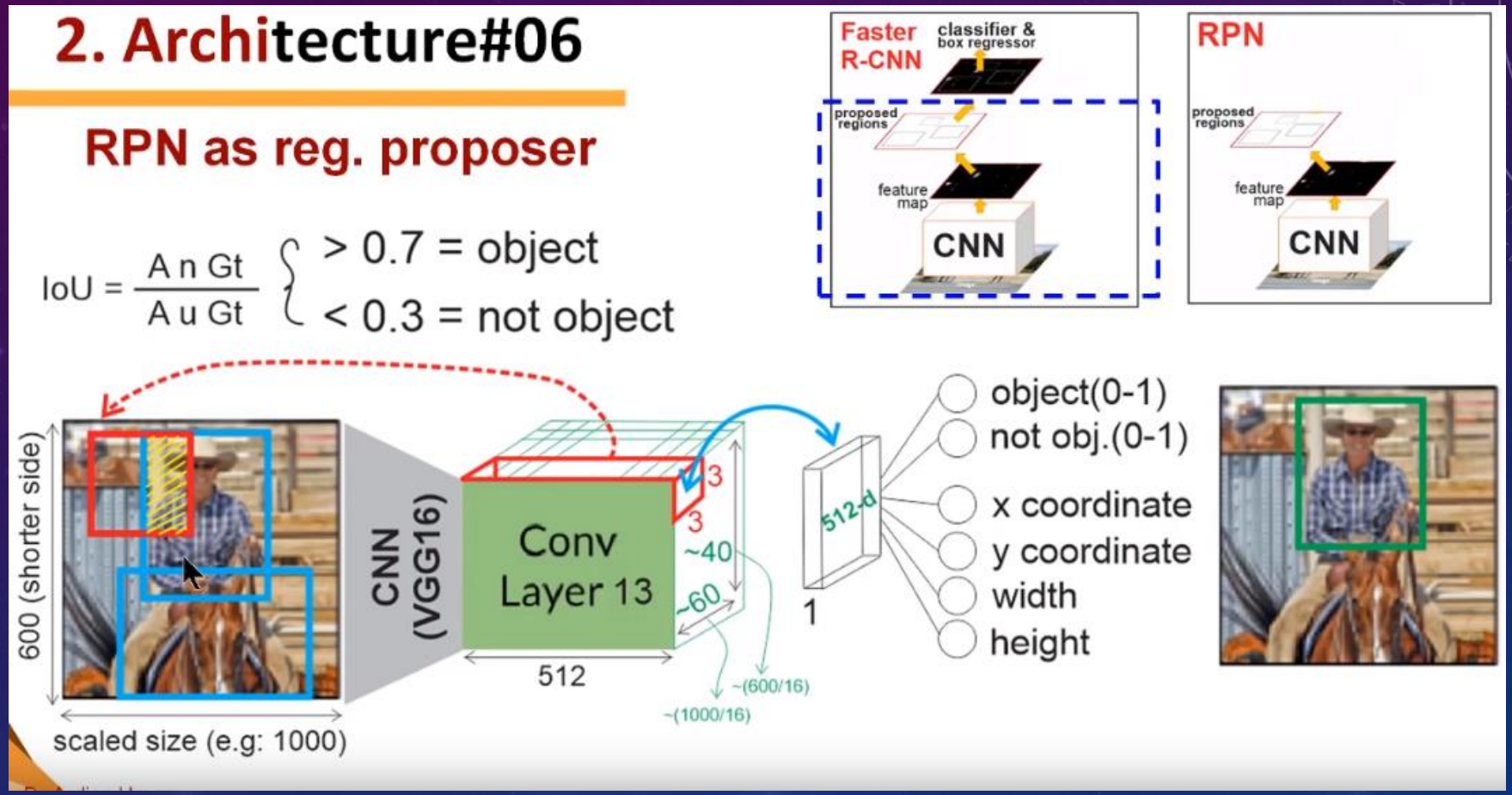
Faster R-CNN architecture





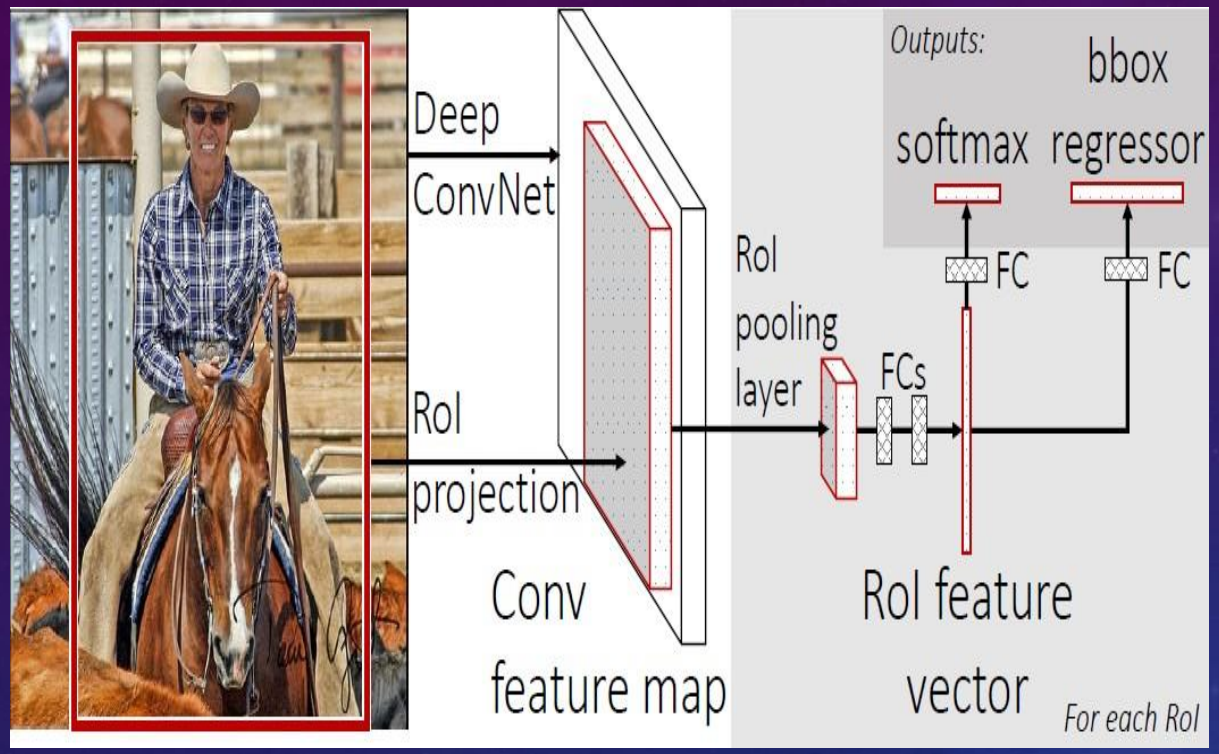
https://www.youtube.com/watch?v=c1_g6tw69bU&list=PLkRkKTC6HZMzp28TxR_fJYZ-K8Yu3EQw0&index=2 [5:07]

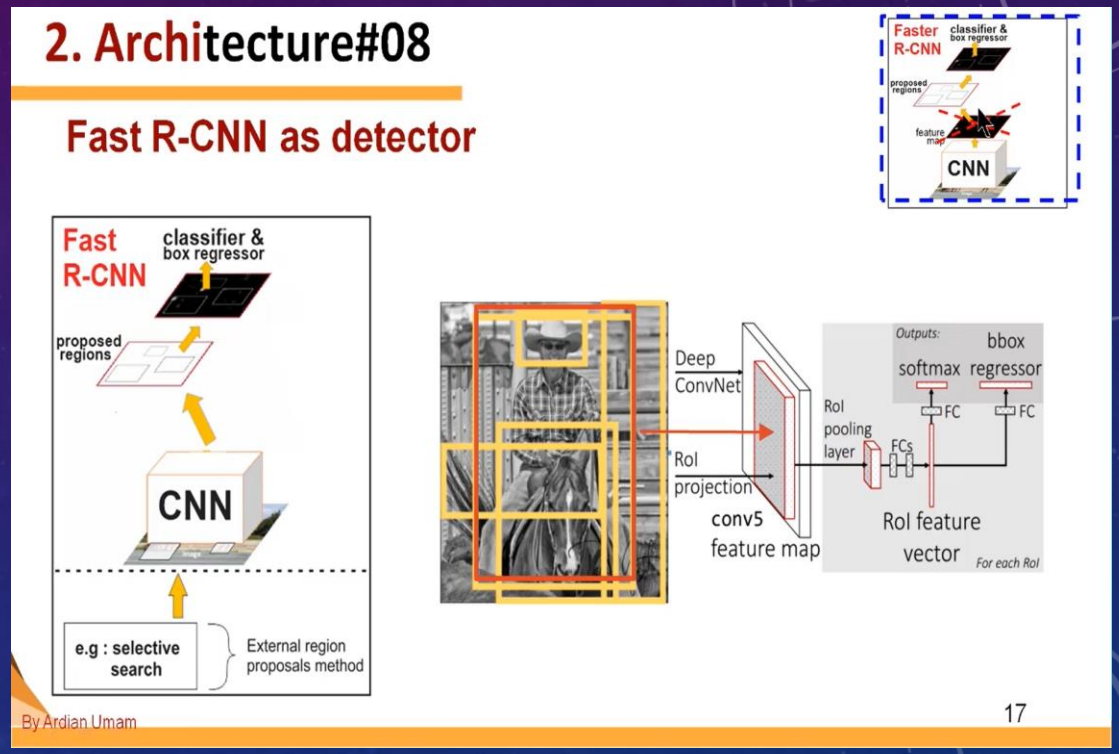
How RPN (Region Proposal Network) works



https://www.youtube.com/watch?v=X3IlbjQs190&list=PLkRkKTC6HZMzp28TxR_fJYZ-K8Yu3EQw0&index=3 [18:10]

How Fast R-CNN Works





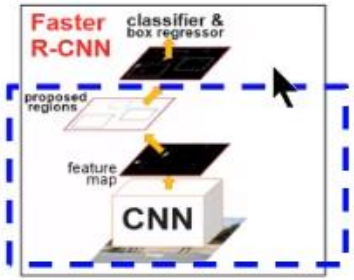
https://www.youtube.com/watch?v=xzw3lcdllOU&list=PLkRkKTC6HZMzp28TxR_fJYZ-K8Yu3EQw0&index=4 [2:51]

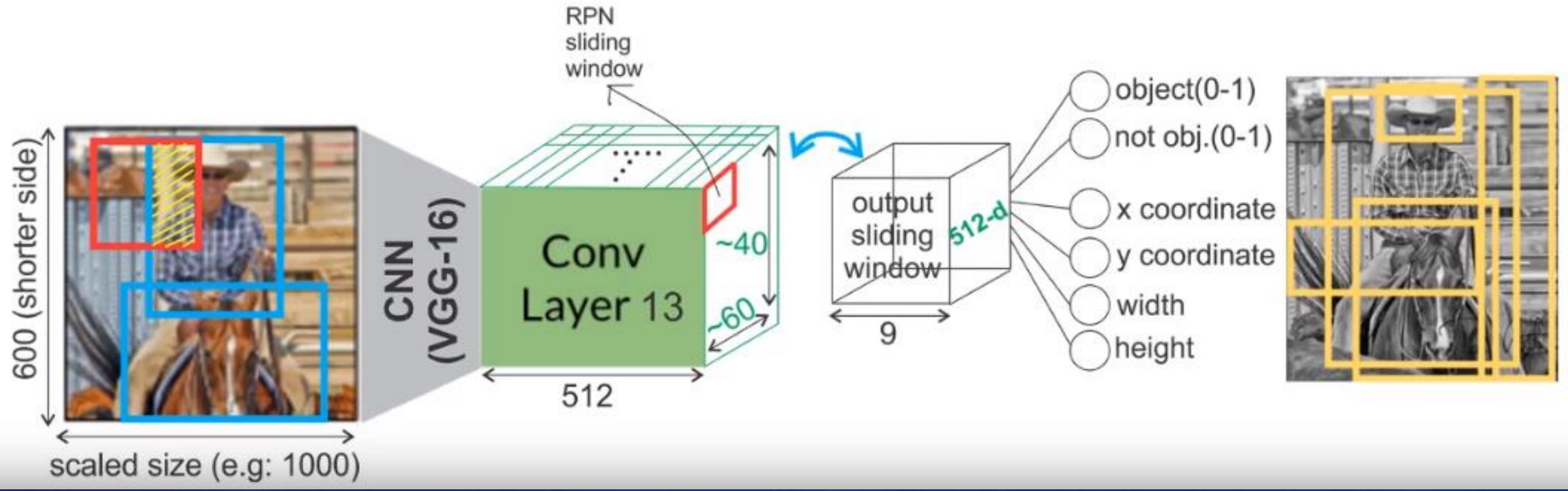
How To Train Faster R-CNN

3. Training phase#03

Step 1 (Proposer)

1. Train RPN, initialized with ImageNet pre-trained model





https://www.youtube.com/watch?v=cSO1nUj495Y&list=PLkRkKTC6HZMzp28TxR_fJYZ-K8Yu3EQw0&index=5 [13:55]

Experiments and Results of Faster R-CNN

Experiments & Results#1

Experiments: Datasets

Visual Object Classes Challenge 2012 (VOC2012)

PASCAL2
Pattern Analysis, Statistical Modelling and Computational Learning

VOC 2007

5k trainval images
5k test images
20 object categories

80k training images
40k validation set images
20k test images
80 object categories

Microsoft COCO
Common Objects in Context

By Ardian Umam

29

https://www.youtube.com/watch?v=306ieamtvGM&list=PLkRkKTC6HZMzp28TxR_fJYZ-K8Yu3EQw0&index=6 [20:51]

How to train your own model?

Step 1 : Label-img : get the annotation (xml) file

Step 2 : XML->TXT

Step 3 : Calculate Anchor Box By K-Means

Step 4 : Start training

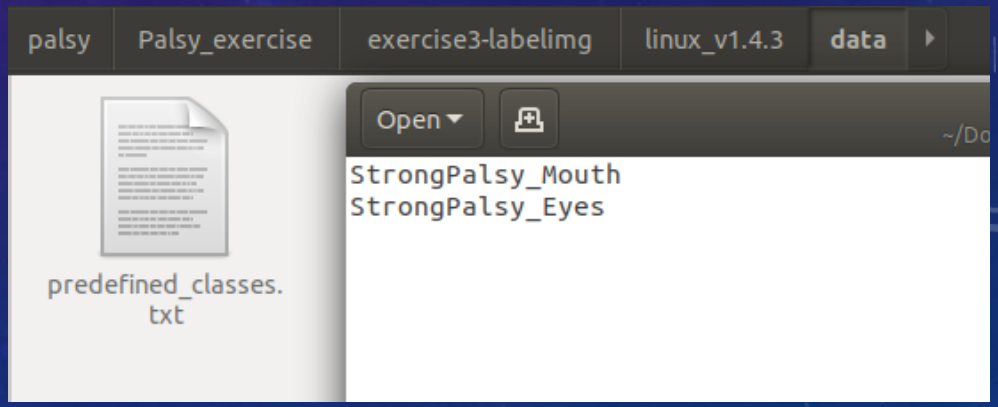
Step 5 : Test Model

Step 1-1 Label-img

- Unzip the Palsy_exercise.zip
- Open “exercise3-labelimg/linux_v1.4.3/labelimg”
- Open “data” folder and open “pretrained_classes.txt”
- Type the name of class:

  1.StrongPalsy_Mouth

  2.StrongPalsy_Eyes



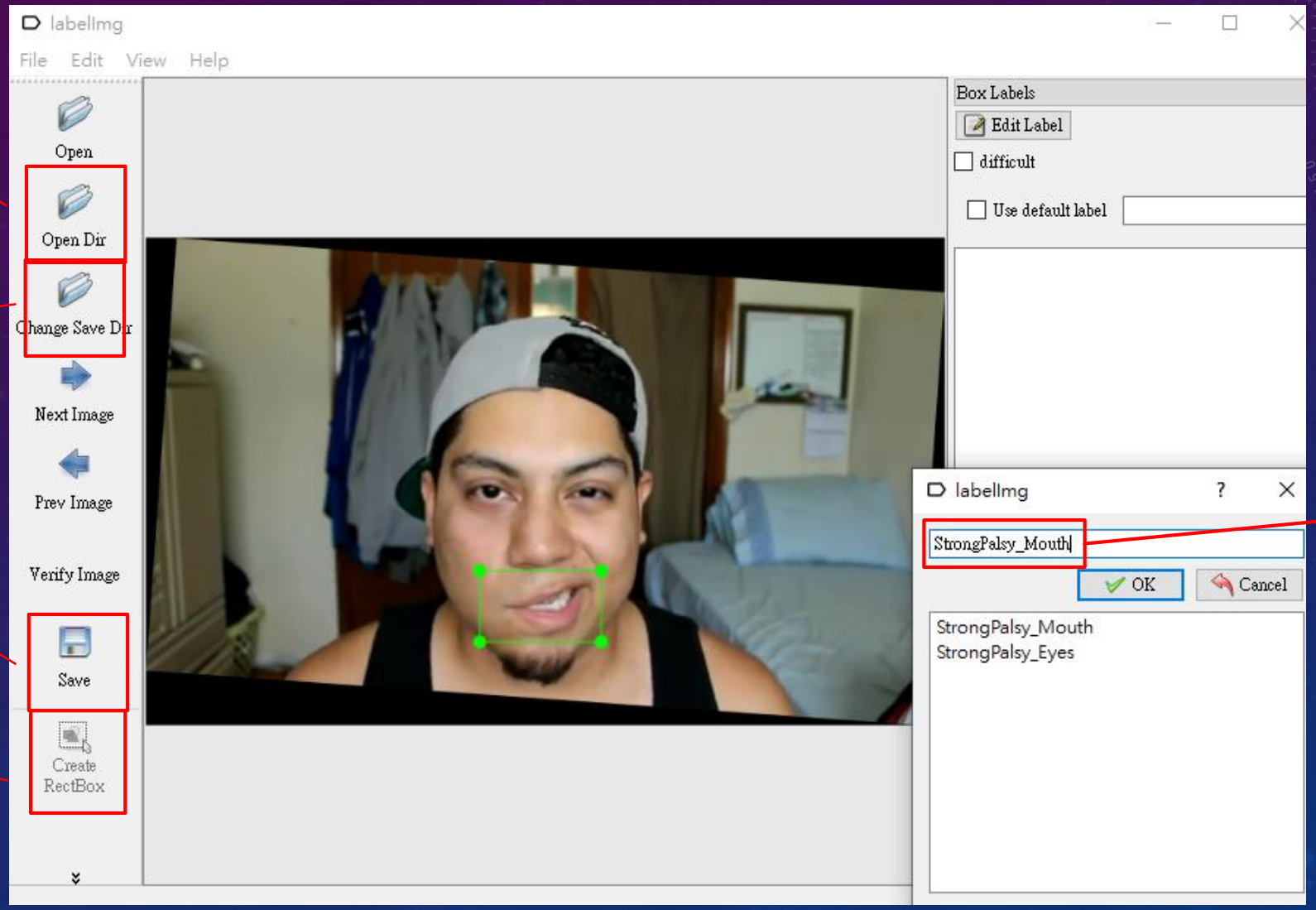
Step 1-2 Label-img

Choose the image dir

Choose the save dir

Save as xml file

Use Create RectBox to crop palsy part



Choose class
1.StrongPalsy_Mouth
2.StrongPalsy_Eyes

Step 1-3 Label-img

```
<annotation>
    <folder>1</folder>
    <filename>146.bmp</filename>
    <path>Y:/108_Steven/1/1/146.bmp</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>664</width>
        <height>405</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>StrongPalsy_Mouth</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>289</xmin>
            <ymin>279</ymin>
            <xmax>378</xmax>
            <ymax>334</ymax>
        </bndbox>
    </object>
</annotation>
```

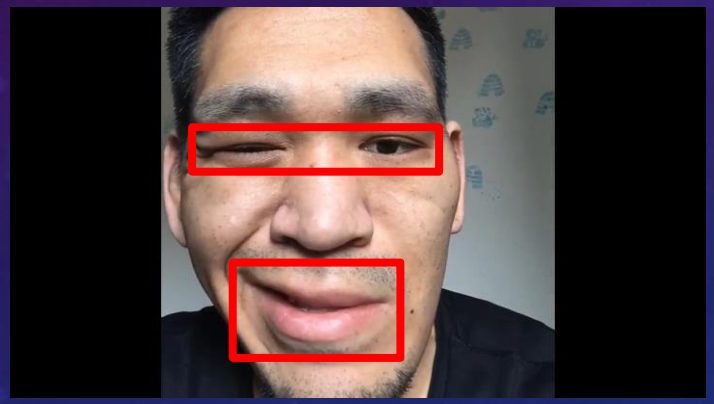
The size of image

The class

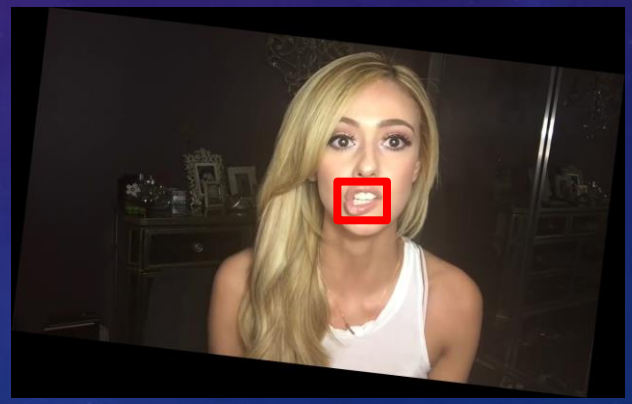
The position of box

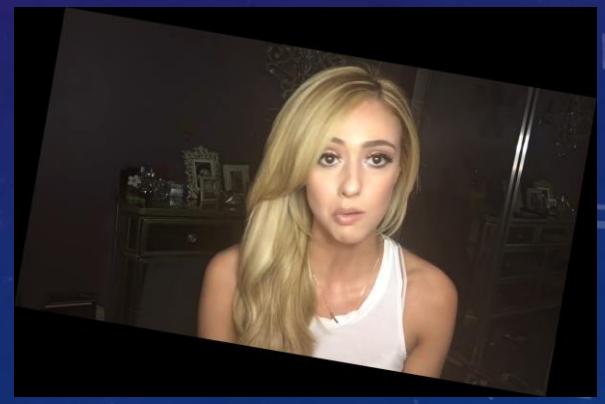
Step 1-4 Label Example

- Three special cases:
  - Case 1 : If only one eye has symptoms, please crop the whole eye region
  - Case 2 : Strength definition



Case 1





Case 2

Step 2-1 XML->TXT

Open “Palsy_exercise/exercise4-xmltotxt/xmltotxt.py”

```
import os
import sys
import xml.etree.ElementTree as ET
import glob

def xml_to_txt(indir,outdir):

    os.chdir(indir)
    annotations = os.listdir('.')
    annotations = glob.glob(str(annotations)+'*.xml')
    print(annotations)

    for i, file in enumerate(annotations):
```

Define function

Load xml file in folder

```
        in_file = open(file)
        tree=ET.parse(in_file)
        root = tree.getroot()
        filename = root.find('filename').text
        for obj in root.iter('object'):
            current = list()
            name = obj.find('name').text
            xmlbox = obj.find('bndbox')
            xn = xmlbox.find('xmin').text
            xx = xmlbox.find('xmax').text
            yn = xmlbox.find('ymin').text
            yx = xmlbox.find('ymax').text
            size = root.find('size')
            x = size.find('width').text
            y = size.find('height').text
            x_1 = ((float(xn)+float(xx))/2)/float(x)
            y_1 = ((float(yn)+float(yx))/2)/float(y)
            w = (float(xx)-float(xn))/float(x)
            h = (float(yx)-float(yn))/float(y)
```

Find bndbox tag

Read bounding box coordinate

Read image size

normalization

Step 2-2 XML->TXT

```
if name == 'StrongPalsy_Mouth‘
        file_save = file.split('.')[0]+'.txt'
        file_txt = outdir+"/"+file_save
        f_w = open(file_txt, 'a+')
        f_w.write('0 '+str(x_1)+' '+str(y_1)+' '+str(w)+' '+str(h)+'\n')

if name == 'StrongPalsy_Eyes’:
        file_save = file.split('.')[0]+'.txt'
        file_txt = outdir+"/"+file_save
        f_w = open(file_txt, 'a+')
        f_w.write('1 '+str(x_1)+' '+str(y_1)+' '+str(w)+' '+str(h)+'\n')

indir='/home/avlab/Documents/John/palsy/Palsy_total/Image_large_XML’

outdir='/home/avlab/Documents/John/palsy/Palsy_total/image'

for foldername in os.listdir(indir):

    folderdir = os.path.join(indir, foldername)
    outfolderdir = os.path.join(outdir, foldername)

    xml_to_txt(folderdir,outfolderdir)
```

Write in txt

Input xml path

Output txt path (to dataset)

Run the define function

```
<name>StrongPalsy_Mouth</name>
<pose>Left</pose>
<bndbox>
    <xmin>295</xmin>
    <ymin>228</ymin>
    <xmax>372</xmax>
    <ymax>357</ymax>
</bndbox>
```

Step 2-3 XML->TXT

xml file

```
<annotation>
Y:\108_Steven\comp4_det_test_StrongPalsy_Mouth15.txt
    <filename>146.bmp</filename>
    <source>
        <database>The PALSY Database</database>
        <annotation>PALSY </annotation>
        <image>flickr</image>
        <image>flickr</image>
        <flickrid>341012865</flickrid>
    </source>
    <owner>
        <flickrid>Fried Camels</flickrid>
        <name>Jinky the Fruit Bat</name>
    </owner>
    <size>
        <width>664</width>
        <height>405</height>
        <depth>3</depth>
    </size> <segmented>0</segmented>
    <object>
        <name>SlightPalsy_Eyes</name>
        <pose>Left</pose>
        <bndbox>
            <xmin>244</xmin>
            <ymin>180</ymin>
            <xmax>413</xmax>
            <ymax>275</ymax>
        </bndbox>
    </object>
    <object>
        <name>StrongPalsy_Mouth</name>
        <pose>Left</pose>
        <bndbox>
            <xmin>295</xmin>
            <ymin>228</ymin>
            <xmax>372</xmax>
            <ymax>357</ymax>
        </bndbox>
    </object>
</annotation>
```

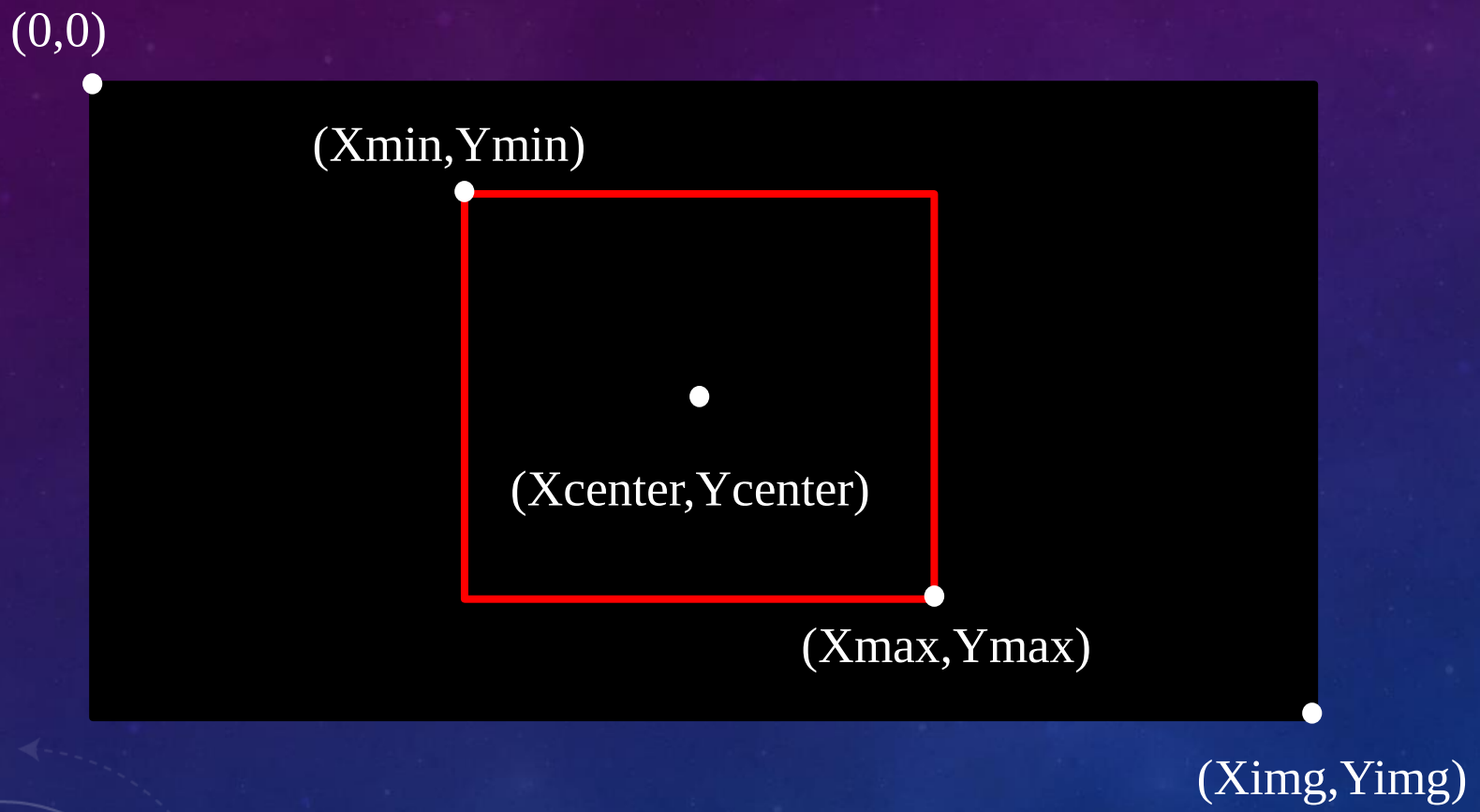
txt file

```
0 0.5022590361445783 0.7222222222222222 0.11596385542168675 0.31851851851851853
```

class

x y w h with normalization

Step 2-4 Scale Normalization



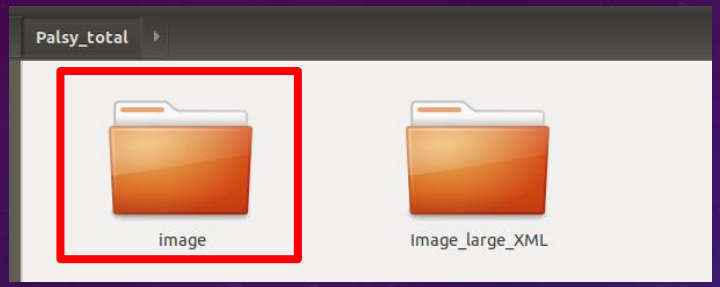
Xnor = Xcenter / Ximg

Ynor = Ycenter / Yimg

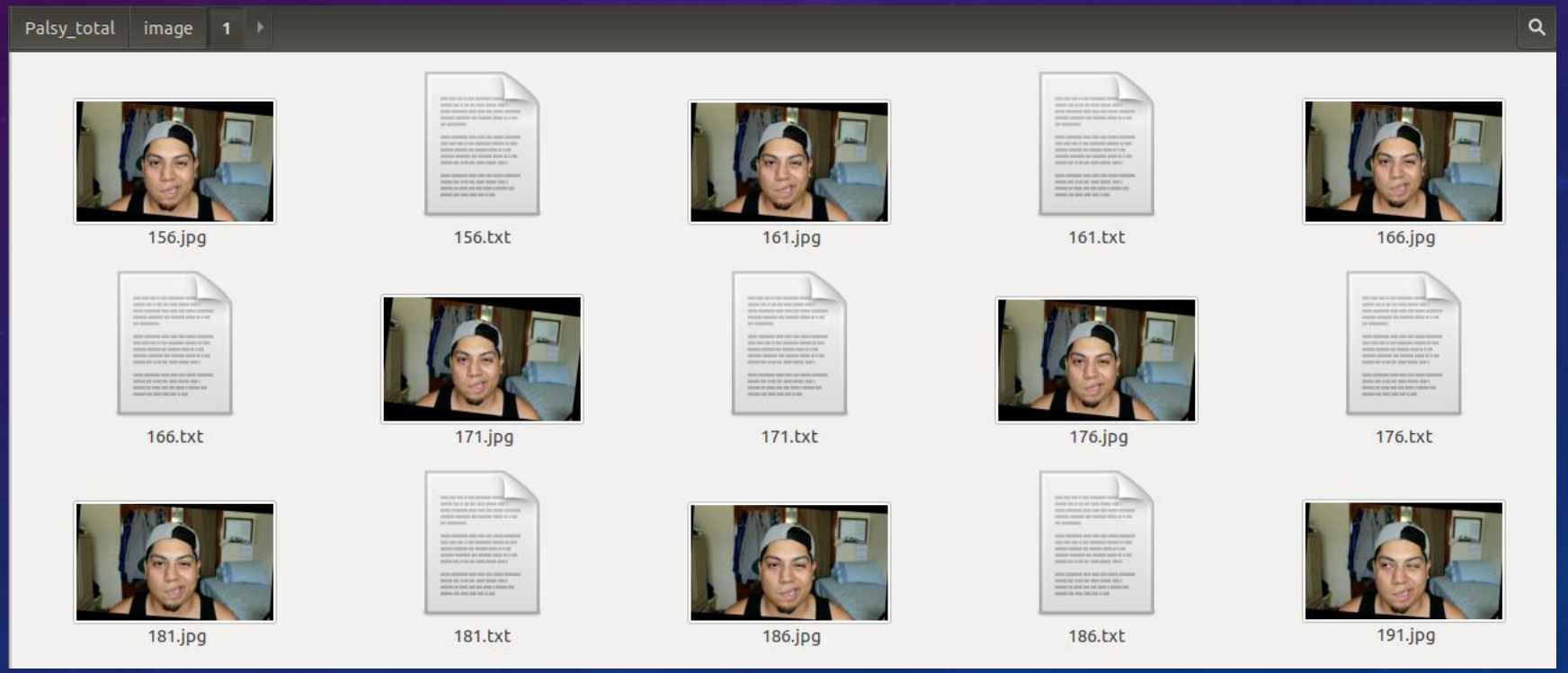
Wnor = (Xmax-Xmin) / Ximg

Hnor = (Ymax-Ymin) / Yimg

Step 2-5 Confirm the txt file



Confirm the txt file is already created in folder.



Step 2-6 Data Preprocessing (Train/Valid list)

Open “Palsy_exercise/create_list.py”

```
import os
xmldir='/home/avlab/Documents/John/palsy/Palsy_total/image '
imgdir='/home/avlab/Documents/John/palsy/Palsy_total/image'

num = len(os.listdir(xmldir))
print('The total number of subjects : {}'.format(num))
valid = os.listdir(xmldir)[5]
print('Valid Subject : {}'.format(valid))

for foldername in os.listdir(xmldir):
    if foldername == valid:
        for filename in os.listdir(xmldir+'/' + foldername):
            if filename.endswith(".txt"):
                with open('./valid.txt','a+') as file:
                    file.write(imgdir+'/'+foldername+'/'+filename[:-4]+'.jpg'+'\n')
    else:
        for filename in os.listdir(xmldir+'/'+foldername):
            if filename.endswith(".txt"):
                with open('./train.txt','a+') as file:
                    file.write(imgdir+'/'+foldername+'/'+filename[:-4]+'.jpg'+'\n')
```

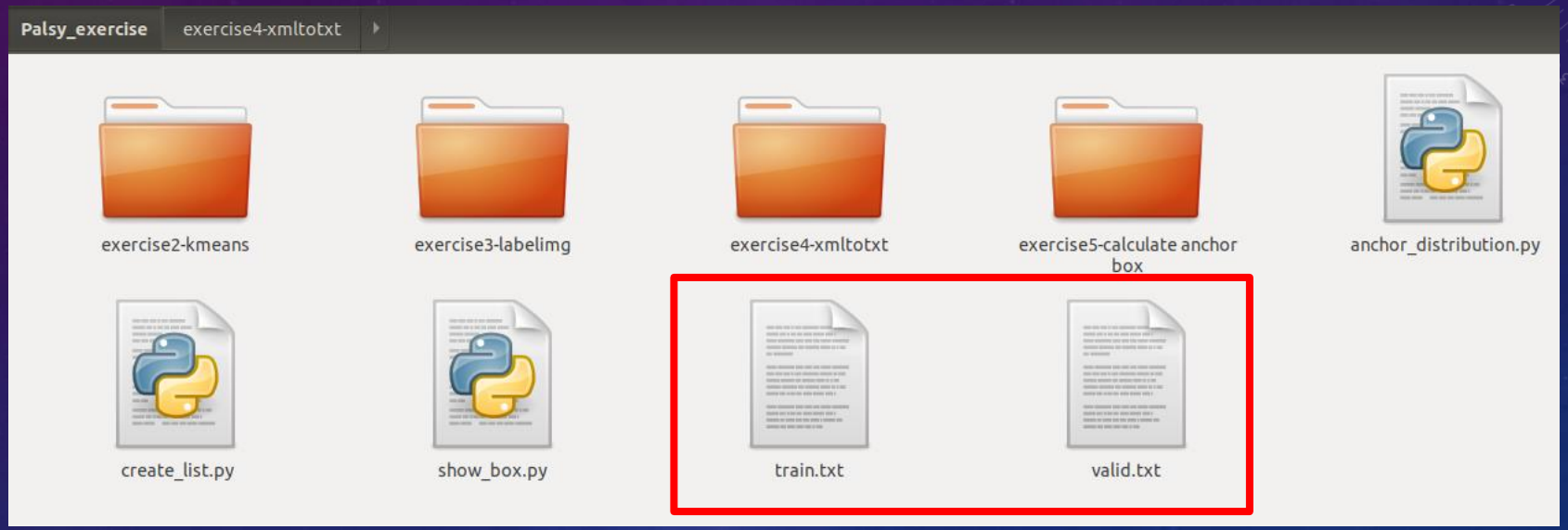
Choose the path of dataset.

Choose the subject we want to test.

Open txt file and write the train data’s path in it.

Step 2-7 Confirm the Train/Valid list

Confirm the Train/Valid list is already created in folder.



Step 3-1 Calculate Anchor Box By K-Means

Open “Palsy_exercise/exercise5- calculate anchor box /anchor.py”

```
def IOU(x,centroids):
    '''
    :param x: a ground truth of w, h
    :param centroids:  The set of w, h of anchor [[w,h),(),...]
    :return: A set of IoU values for a single ground truth box and all
k anchor boxes
    '''
    IoUs = []
    w, h = x  # w,h of ground truth
    for centroid in centroids:
        c_w,c_h = centroid   #w,h of anchor
        if c_w>=w and c_h>=h:
            iou = w*h/(c_w*c_h)
        elif c_w>=w and c_h<=h:
            iou = w*c_h/(w*h + (c_w-w)*c_h)
        elif c_w<=w and c_h>=h:
            iou = c_w*h/(w*h + c_w*(c_h-h))
        else:
            iou = (c_w*c_h)/(w*h)
        IoUs.append(iou)
    return np.array(IoUs)
```

```
def avg_IOU(X,centroids):
    '''
    :param X: The set of w, h of ground truth [(w,h),(),...]
    :param centroids: The set of w, h of anchor [[w,h),(),...]
    '''
    n,d = X.shape
    sum = 0.
    for i in range(X.shape[0]):
        sum+= max(IOU(X[i],centroids))
        return sum/n
```

Step 3-2 Calculate Anchor Box By K-Means

```
def write_anchors_to_file(centroids,X,anchor_file,input_shape,yolo_version):
    '''
    :param centroids: The set of w, h of anchor [[w,h),(),...]
    :param X: The set of w, h of ground truth [(w,h),(),...]
    :param anchor_file: output path for Anchor and average IoU
    '''
    f = open(anchor_file,'w')

    anchors = centroids.copy()
    print(anchors.shape)

    if yolo_version=='yolov2':
        for i in range(anchors.shape[0]):
            anchors[i][0]*=input_shape/32.
            anchors[i][1]*=input_shape/32.
    elif yolo_version=='yolov3':
        for i in range(anchors.shape[0]):
            anchors[i][0]*=input_shape
            anchors[i][1]*=input_shape
    else:
        print("the yolo version is not right!")
        exit(-1)
    widths = anchors[:,0]
```

```
sorted_indices = np.argsort(widths)

    print('Anchors = ', anchors[sorted_indices])

    for i in sorted_indices[:-1]:
        f.write('%0.2f,%0.2f, '%(anchors[i,0],anchors[i,1]))

    #there should not be comma after last anchor, that's why
    f.write('%0.2f,%0.2f\n'%(anchors[sorted_indices[-
          1:],0],anchors[sorted_indices[-1:],1]))

    f.write('%f\n'%(avg_IOU(X,centroids)))
    print()
```

Step 3-3 Calculate Anchor Box By K-Means

```
def kmeans(X,centroids,eps,anchor_file,input_shape,yolo_version):

    N = X.shape[0] #Number of ground truth
    iterations = 0
    print("centroids.shape",centroids)
    k,dim = centroids.shape
#The number of anchors k and w, h two dimensions, default dim are 2
    prev_assignments = np.ones(N)*(-1)
#Assign initial labels to each ground truth
    iter = 0
    old_D = np.zeros((N,k))
#Initialize the IOU of each ground truth for each anchor

    while True:
        D = []
        iter+=1
        for i in range(N):
            d = 1 - IOU(X[i],centroids)
            D.append(d)
        D = np.array(D)
# D.shape = (N,k)  Get each ground truth for each anchor IoU

        print("iter {}: dists = {}".format(iter,np.sum(np.abs(old_D-D))))
#Calculate the change value of each iteration and the previous IoU
```

```
        #assign samples to centroids
        assignments = np.argmin(D,axis=1)  #Assign each ground truth to the
anchor sequence with the smallest distance d

        if (assignments == prev_assignments).all() :
#If the result of the previous assignment is the same as the result of this time,
the anchor and the average IoU are output.
            print("Centroids = ",centroids)

            write_anchors_to_file(centroids,X,anchor_file,input_shape,yolo_versio
            n)
            return

        #calculate new centroids
        centroid_sums=np.zeros((k,dim),np.float)
#Initialize to sum w and h for each cluster
        for i in range(N):
            centroid_sums[assignments[i]]+=X[i]
#Accumulate w and h of ground truth in each cluster separately
        for j in range(k):          #Average the w and h in the cluster
            centroids[j] = centroid_sums[j]/(np.sum(assignments==j)+1)

        prev_assignments = assignments.copy()
        old_D = D.copy()
```

Step 3-4 Calculate Anchor Box By K-Means

```
def main(argv):
    parser = argparse.ArgumentParser()
    parser.add_argument('-filelist', default =
        /home/avlab/Documents/John/palsy/Palsy_exercise/train.txt',
                help='path to filelist\n' )
    parser.add_argument('-output_dir', default = r'.\scripts', type = str,
                help='Output anchor directory\n' )
    parser.add_argument('-num_clusters', default = 5, type = int,
                help='number of clusters\n' )
#choose YOLO version
    parser.add_argument('-yolo_version', default='yolov2', type=str,
                help='yolov2 or yolov3\n')
    parser.add_argument('-yolo_input_shape', default=416, type=int,
                help='input images shape，multiples of 32. etc. 416*416\n')
    args = parser.parse_args()

    if not os.path.exists(args.output_dir):
        os.mkdir(args.output_dir)
```

The path of train list

Define the number of clusters

Define the version of yolo

```
f = open(args.filelist)

    lines = [line.rstrip('\n') for line in f.readlines()]

    annotation_dims = []

    for line in lines:
        line = line.replace('.jpg','.txt')
        print(line)
        f2 = open(line)
        for line in f2.readlines():
            line = line.rstrip('\n')
            w,h = line.split(' ')[3:5]
            annotation_dims.append((float(w),float(h)))
    annotation_dims = np.array(annotation_dims)

    eps = 0.005

    if args.num_clusters == 0:
        for num_clusters in range(1,11): #we make 1 through 10 clusters
```

Step 3-5 Calculate Anchor Box By K-Means

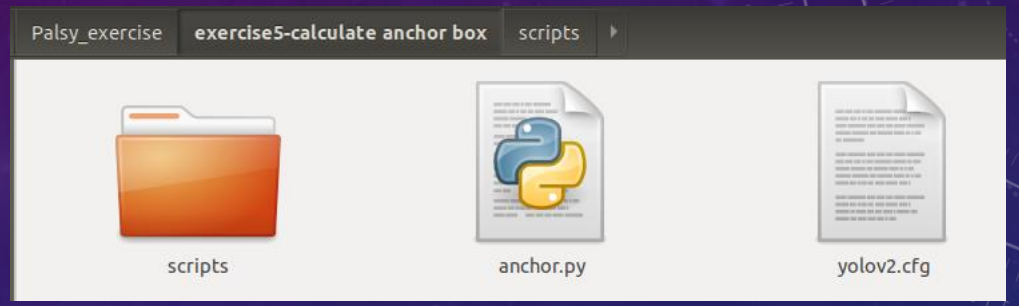
```
anchor_file = join( args.output_dir,'anchors%d.txt'%(num_clusters))

        indices = [ random.randrange(annotation_dims.shape[0]) for i in
                    range(num_clusters)]
        centroids = annotation_dims[indices]

kmeans(annotation_dims,centroids,eps,anchor_file,args.yolo_input_shape,ar
       gs.yolo_version)
        print('centroids.shape', centroids.shape)
   else:
      anchor_file = join( args.output_dir,'anchors%d.txt'%(args.num_clusters))
      indices = [ random.randrange(annotation_dims.shape[0]) for i in
                  range(args.num_clusters)]
      centroids = annotation_dims[indices]

kmeans(annotation_dims,centroids,eps,anchor_file,args.yolo_input_shape,ar
       gs.yolo_version)
      print('centroids.shape', centroids.shape)
```

Confirm the “anchor5.txt” is already created in folder.



Result :