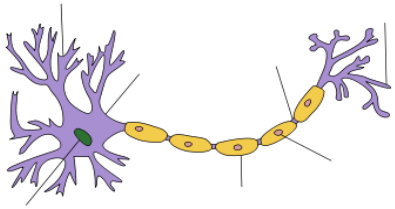
UMONS
Université de Mons

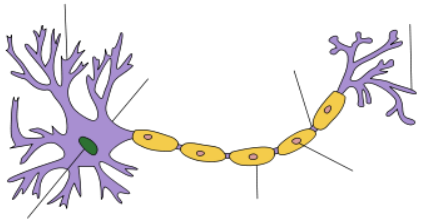
HANDS ON AI : DÉFI 1

QUELQUES PISTES POUR LE PROBLÈME DE CLASSIFICATION

& ARCHITECTURES DE LOCALISATION



Sidi Ahmed Mahmoudi



POLYTECH MONS

sidi.mahmoudi@umons.ac.be

17 Octobre 2019

Rappel : énoncé défi 1

Date de remise : 07/11/2019

Défi 1 : grille d'évaluation

Partie	Taux dévaluation
Partie 1 : classification	40 %
Partie 2 : localisation	20 %
Partie 3 : traitement temps réel (facultatif)	20 %
Qualité rapport	20 %
Qualité code + données	20 %
TOTAL	120 %

PLAN

I. Préparation et division des données

II. Paramètres d'entrainement

III. Architectures de localisation

IV. Exemple de localisation avec Yolo

V. Accès à la plateforme Floydhub

PLAN

I. **Préparation et division des données**

II. Paramètres d'entrainement

III. Architectures de localisation

IV. Exemple de localisation avec Yolo

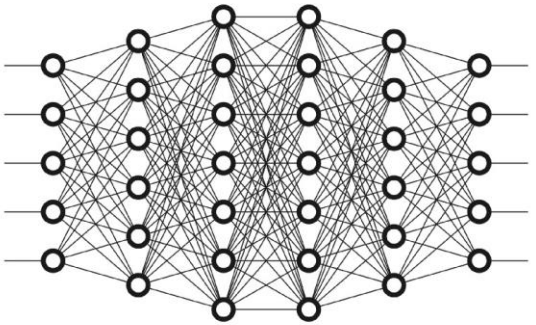
V. Accès à la plateforme Floydhub

Préparation et division des données

Préparation, division des données et entrainement

Training Set

```
# Model 2
model=Sequential()
model.add (Dense(200,input_dim=trainX.shape[1], activation='sigmo
model.add (Dense(100, input_dim=200, activation='sigmoid'))
model.add (Dense(60, input_dim=100, activation='sigmoid'))
model.add (Dense(30, input_dim=60, activation='sigmoid'))
model.add(Dense(nb_classes, activation='softmax'))
```

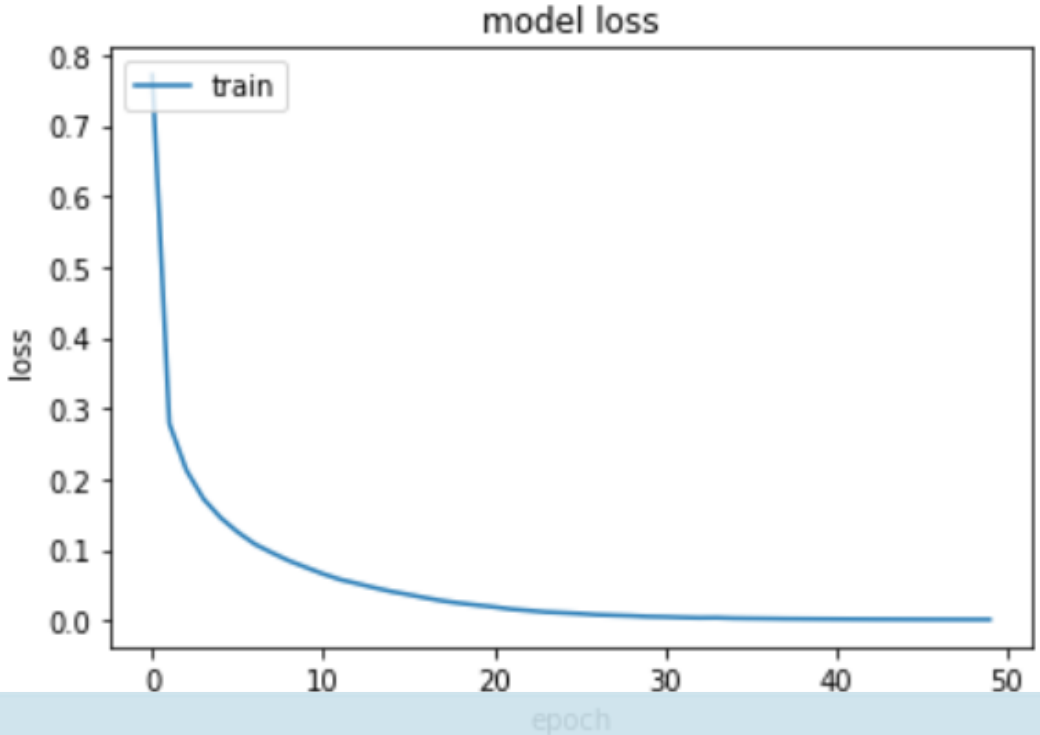


```
history=model.fit(x_train, y_train, epochs=2)
```

```
Epoch 1/2
60000/60000 [==============================] - 7s 111us/step - loss: 0.3656 - acc: 0.8994
Epoch 2/2
60000/60000 [==============================] - 7s 111us/step - loss: 0.3524 - acc: 0.9023
```

Préparation, division des données et entrainement

Training Set



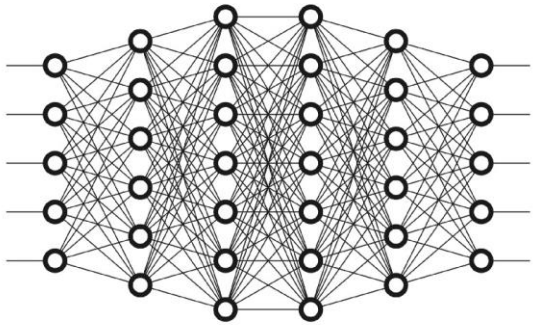
- Peut-on évaluer ce modèle ?

Préparation, division des données et entrainement

Training Set

Validation Set

```
# Model 2
model=Sequential()
model.add (Dense(200,input_dim=trainX.shape[1], activation='sigmo
model.add (Dense(100, input_dim=200, activation='sigmoid'))
model.add (Dense(60, input_dim=100, activation='sigmoid'))
model.add (Dense(30, input_dim=60, activation='sigmoid'))
model.add(Dense(nb_classes, activation='softmax'))
```

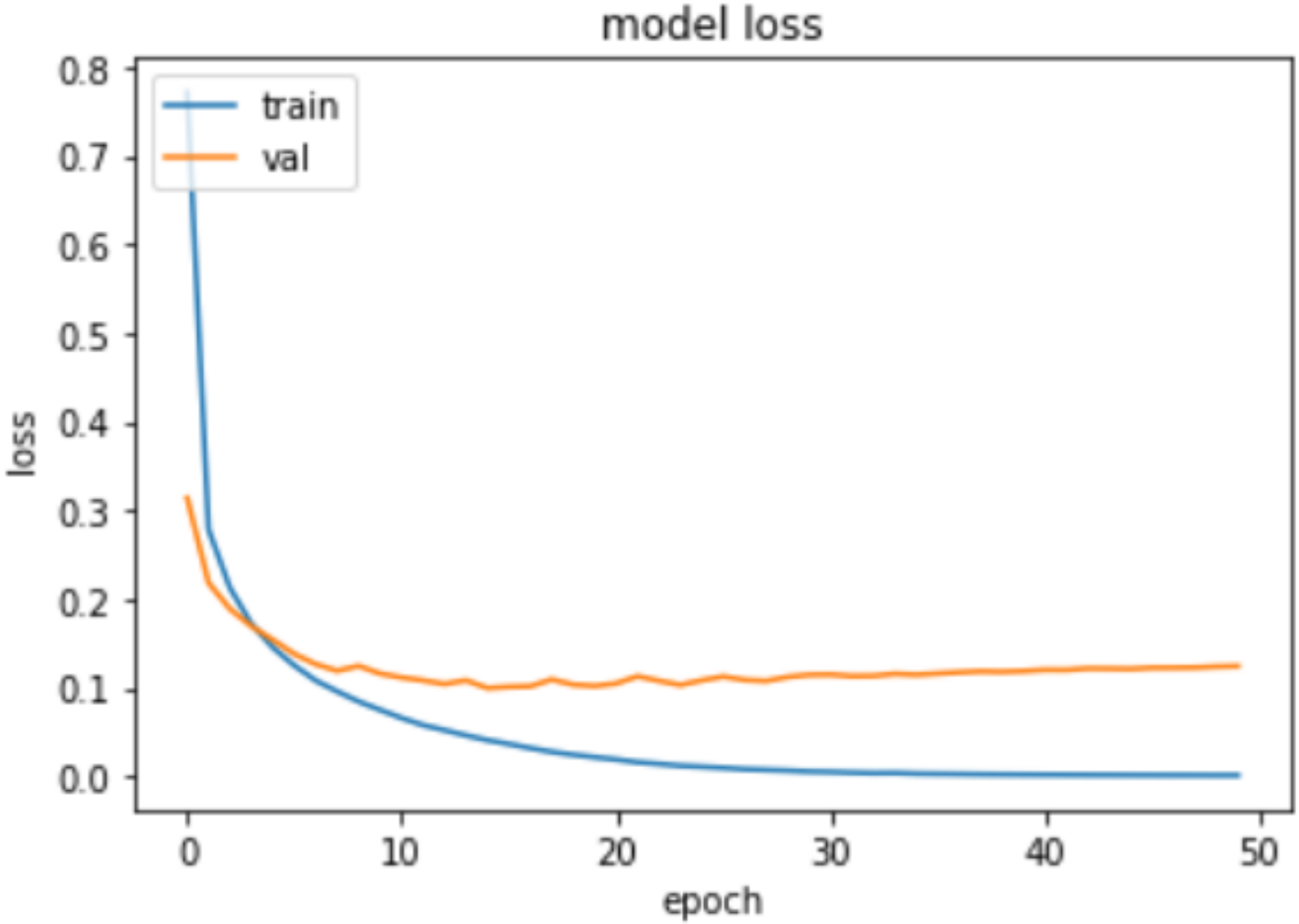


```
history=model.fit(x_train, y_train, validation_split = 0.2, epochs=2)
```

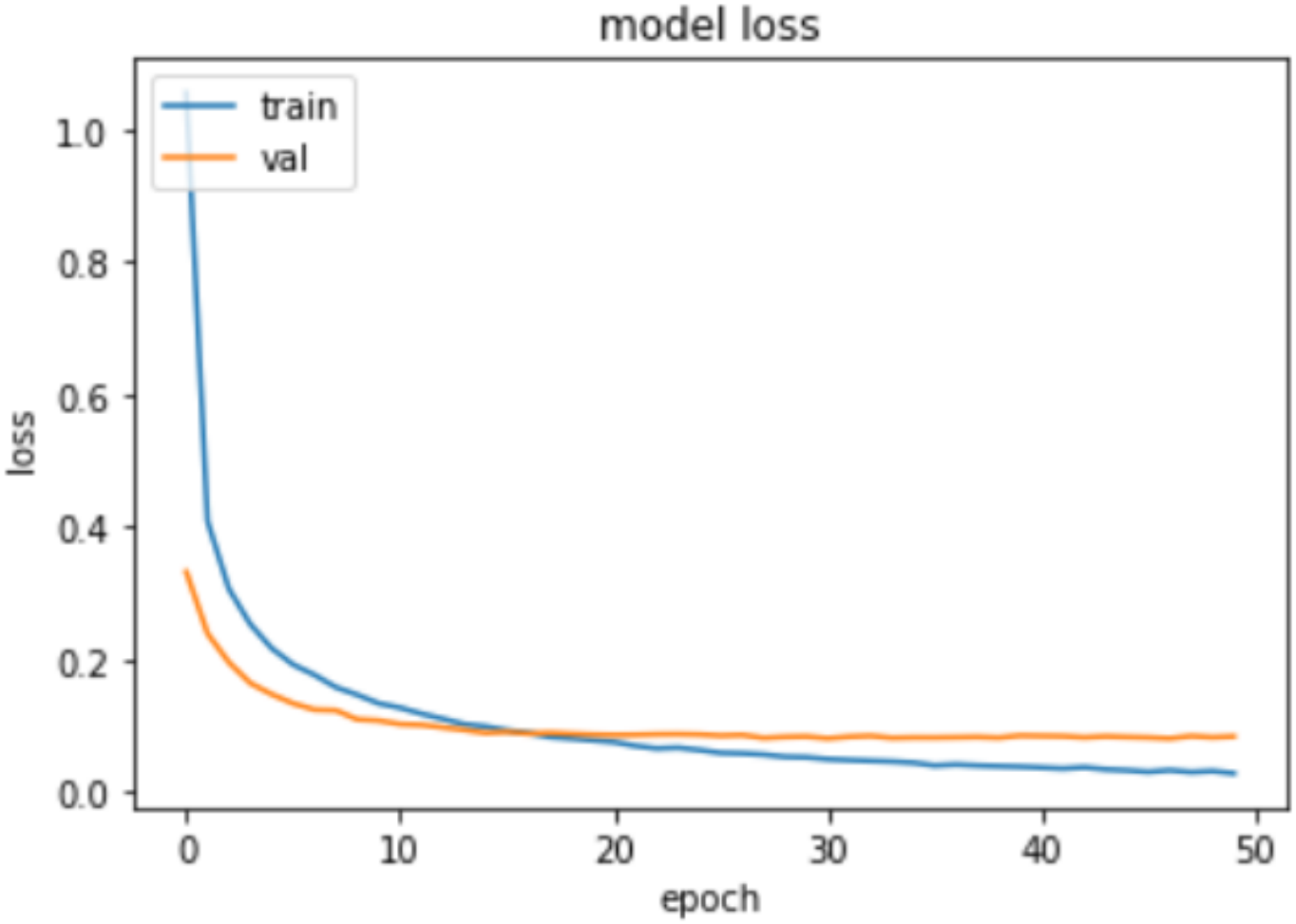
```
Train on 48000 samples, validate on 12000 samples
Epoch 1/2
48000/48000 [==============================] - 6s 123us/step - loss: 0.3483 - acc: 0.9028 - val_loss: 0.3206 - val_acc: 0.9113
Epoch 2/2
48000/48000 [==============================] - 6s 119us/step - loss: 0.3415 - acc: 0.9047 - val_loss: 0.3160 - val_acc: 0.9123
```

- Ou se trouve la différence ? Est-ce suffisant ?

Préparation, division des données et entrainement



Avec le Dropout



Préparation, division des données et entrainement

- Données de test: soit téléchargés au début
- Données de test : split des données d'entrainement

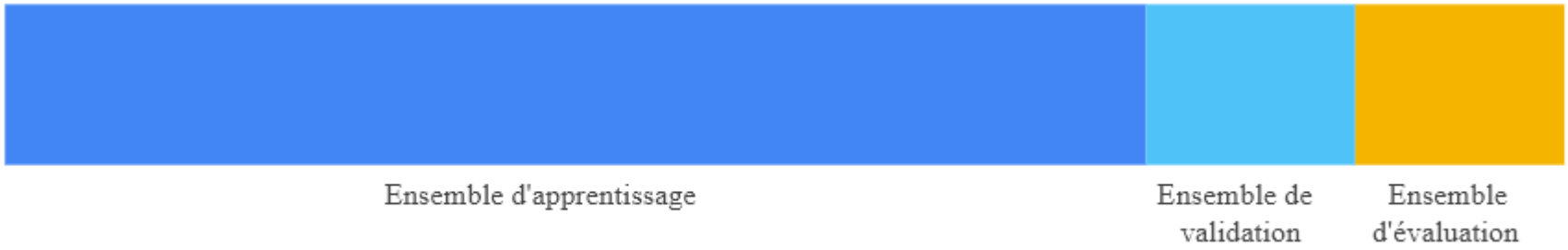
```
trainX, testX, trainY, testY=train_test_split(x_train, y_train, test_size=0.25, random_state=42)
```

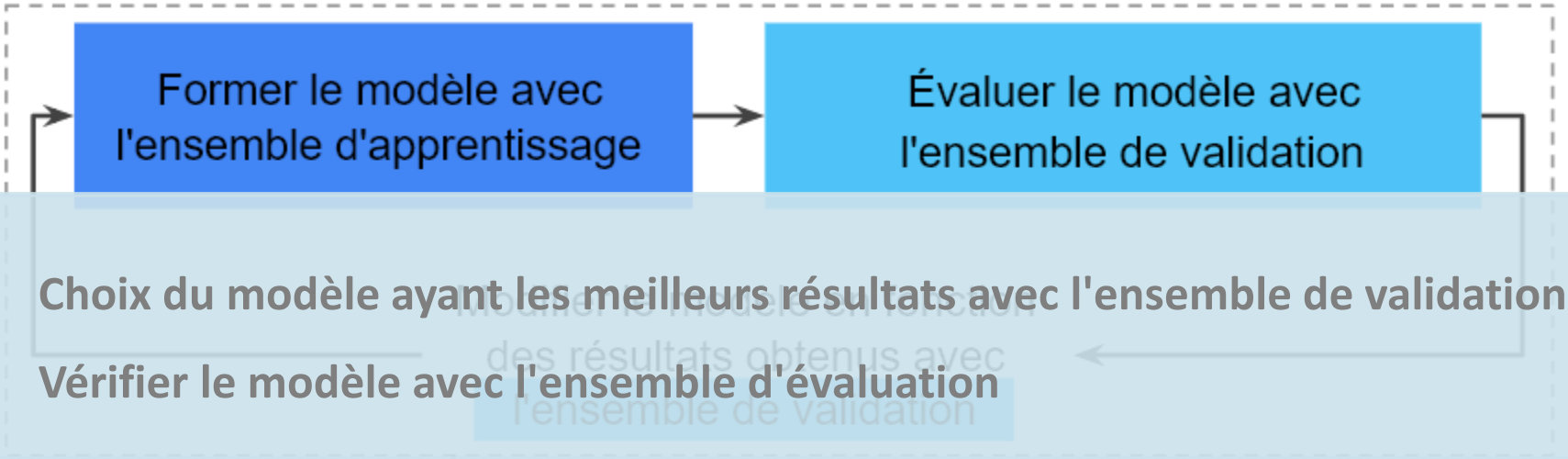
Préparation, division des données et entrainement


```
train_accuracy=model.evaluate(trainX, trainY)[1]
print ("Train accuracy = ", train_accuracy)
valid_accuracy=model.evaluate(validX, validY)[1]
print ("Valid accuracy = ", valid_accuracy)
test_accuracy=model.evaluate(x_test, y_test)[1]
print ("Test accuracy = ", test_accuracy)
```

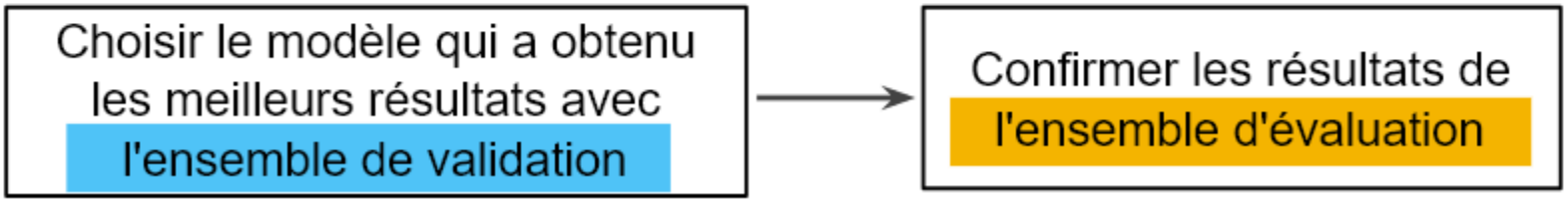
```
45000/45000 [==============================] - 3s 65us/step
Train accuracy =  0.9998666666666667
15000/15000 [==============================] - 1s 69us/step
Valid accuracy =  0.9796666666666667
10000/10000 [==============================] - 1s 68us/step
Test accuracy =  0.9821
```

La validation





- Choix du modèle ayant les meilleurs résultats avec l'ensemble de validation
- Vérifier le modèle avec l'ensemble d'évaluation



PLAN

I. Préparation et division des données

II. **Paramètres d'entrainement**

III. Architectures de localisation

IV. Exemple de localisation avec Yolo

V. Accès à la plateforme Floydhub

Paramètres d'entrainement

Epocs

- 1 époque : l'ensemble entier des données est passé dans le réseau 1 fois

Batch_size

- Nombre de données d'entrainement présents dans un batch
- Données divisés en plusieurs batchs

Itérations

- Nombre de batchs par époques : tailles des données/batch_size

Paramètres d'entrainement

Model.fit

- Tableau d'entités sous forme de valeurs x et de cible sous forme de valeurs y.
- Passe tous ensemble de données à la fois dans le réseau (données en RAM)
- Si toutes les données peuvent être chargées en mémoire (**petit datasets**).

Fit_generator

- Pas de transmission directe des données, passage par un générateur
- Génération de données : data augmentation
- Lors de l'utilisation de **grand jeu de données**.

Paramètres d'entrainement

Model.fit

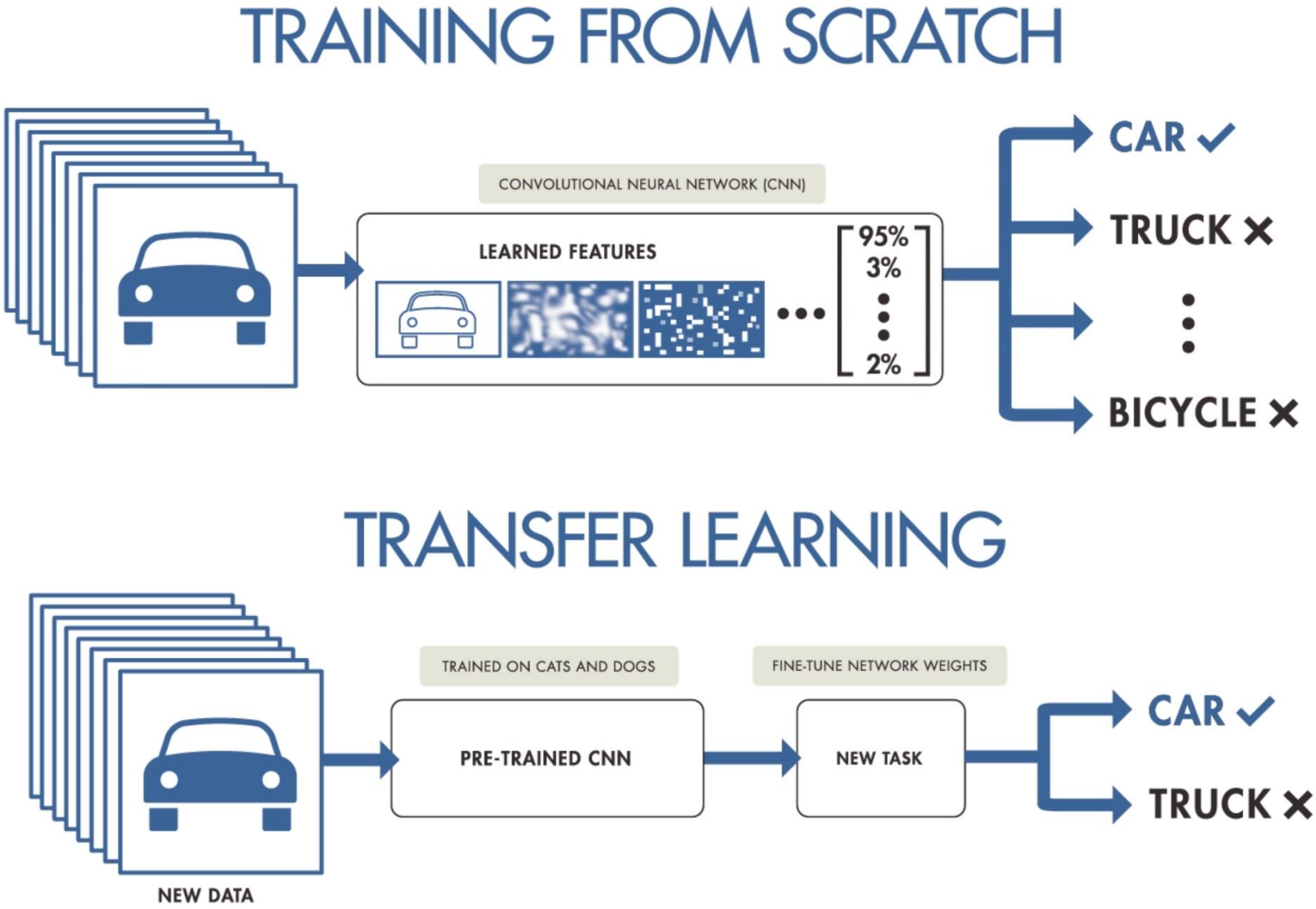
```
history=model.fit(x_train, y_train, validation_split=0.2, epochs=5)
```

```
history=model.fit_generator(
    generator=generate_from_paths_and_labels(
        input_paths=train_input_paths,
        labels=train_labels,
        batch_size=nbr_batch_size,
        input_size=(224,224,3)

    ),
    steps_per_epoch=math.ceil(len(train_input_paths) / nbr_batch_size),
    epochs=epochs,
    validation_data=generate_from_paths_and_labels(
        input_paths=val_input_paths,
        labels=val_labels,
        batch_size=nbr_batch_size,
        input_size=(224,224,3)
    ),
    validation_steps=math.ceil(len(val_input_paths) / nbr_batch_size),
    verbose=1
)
```

Paramètres d'entrainement

```
def generate_from_paths_and_labels(input_paths, labels, batch_size, input_size=(299,299)):

    num_samples = len(input_paths)
    while 1:
        perm = np.random.permutation(num_samples)
        input_paths = input_paths[perm]
        labels = labels[perm]
        for i in range(0, num_samples, batch_size):
            inputs = list(map(
                lambda x: image.load_img(x, target_size=input_size),
                input_paths[i:i+batch_size]
            ))
            inputs = np.array(list(map(
                lambda x: image.img_to_array(x),
                inputs
            )))
            inputs = preprocess_input(inputs)
            yield (inputs, labels[i:i+batch_size])
```

Transfer Learning



Pour ceux qui souhaitent démarrer la partie localisation

- Commencer le travail sur Google Colab
- Passer à vos comptes Floydhub lors du lancement des entraînements
- **Attention :** arrêter après la fin des entrainements (utiliser des jobs)
- Architectures de localisation : décrites ci-après (présentées le 24/10)
- Dossier de démarrage: base de données, code Yolo, paramètres,
- Dossier de démarrage : https://github.com/belarbi2733/keras_yolov3
- **But :** développer un modèle maximisant la précision des résultats
- **Test:** préparer une base de test assez diversifiée

Travailler avec Floydhub

- Instruction d'utilisation de Floydhub: (voir Moodle (Partie 2)
  1. Connexion/création de projet
  2. Chargement de base de données sur Floydhub
  3. Attacher le projet à une base de données existante
  4. Création de jobs

PLAN

I. Préparation et division des données

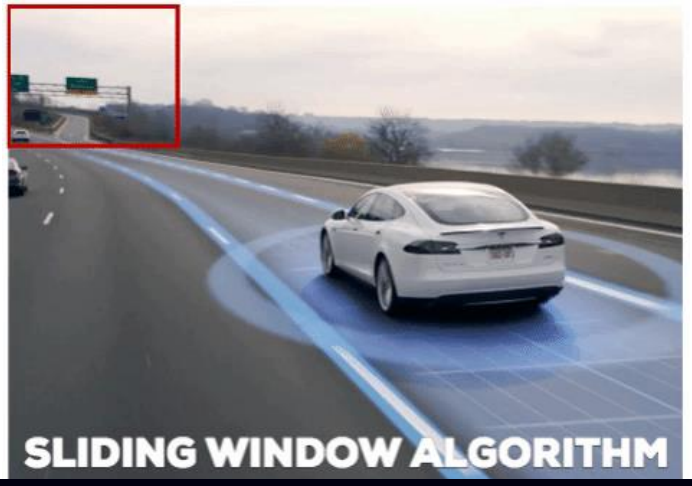
II. Paramètres d'entrainement

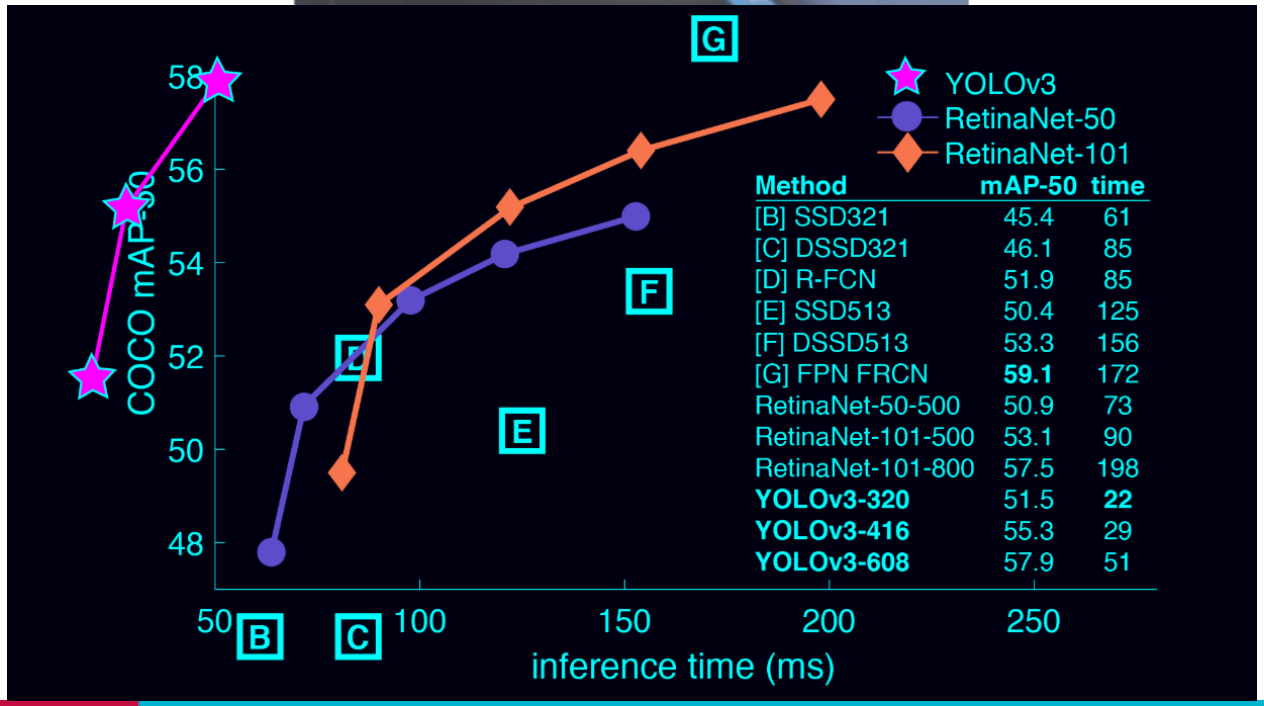
III. **Architectures de localisation**

IV. Exemple de localisation avec Yolo

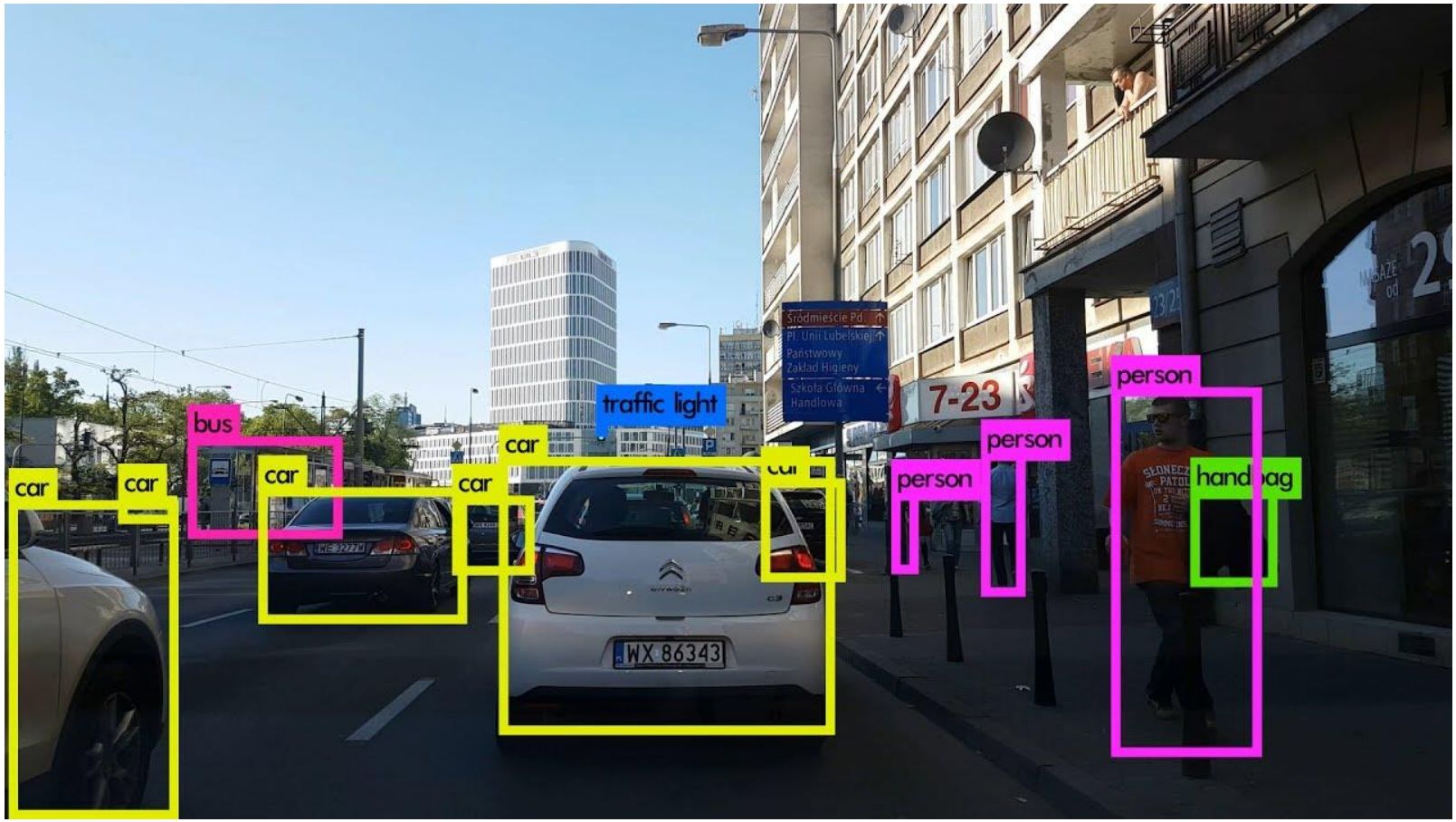
V. Accès à la plateforme Floydhub

Architectures de localisation d'objets

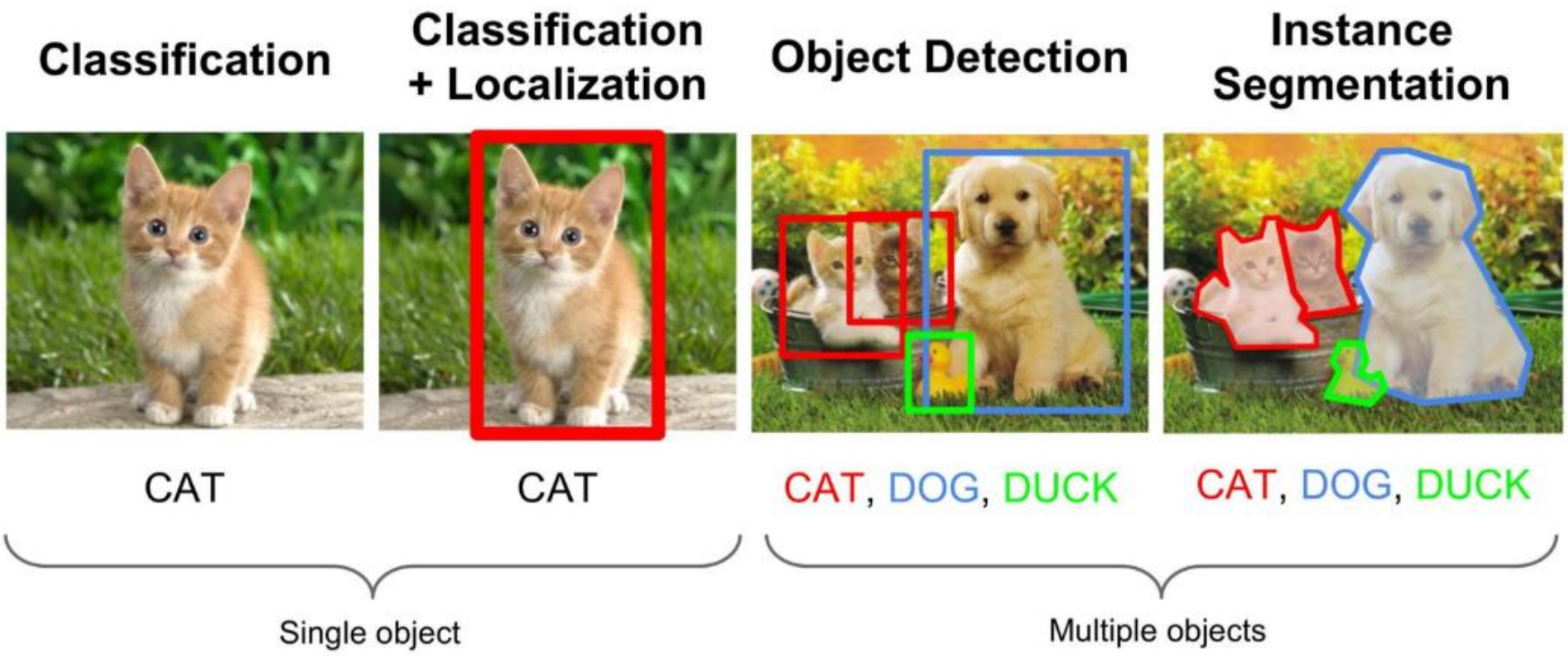




Architecture (Deep) de détection d'objets



Architecture (Deep) de détection d'objets



Comparison between image classification, object detection and instance segmentation.

https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

Architecture (Deep) de détection d'objets

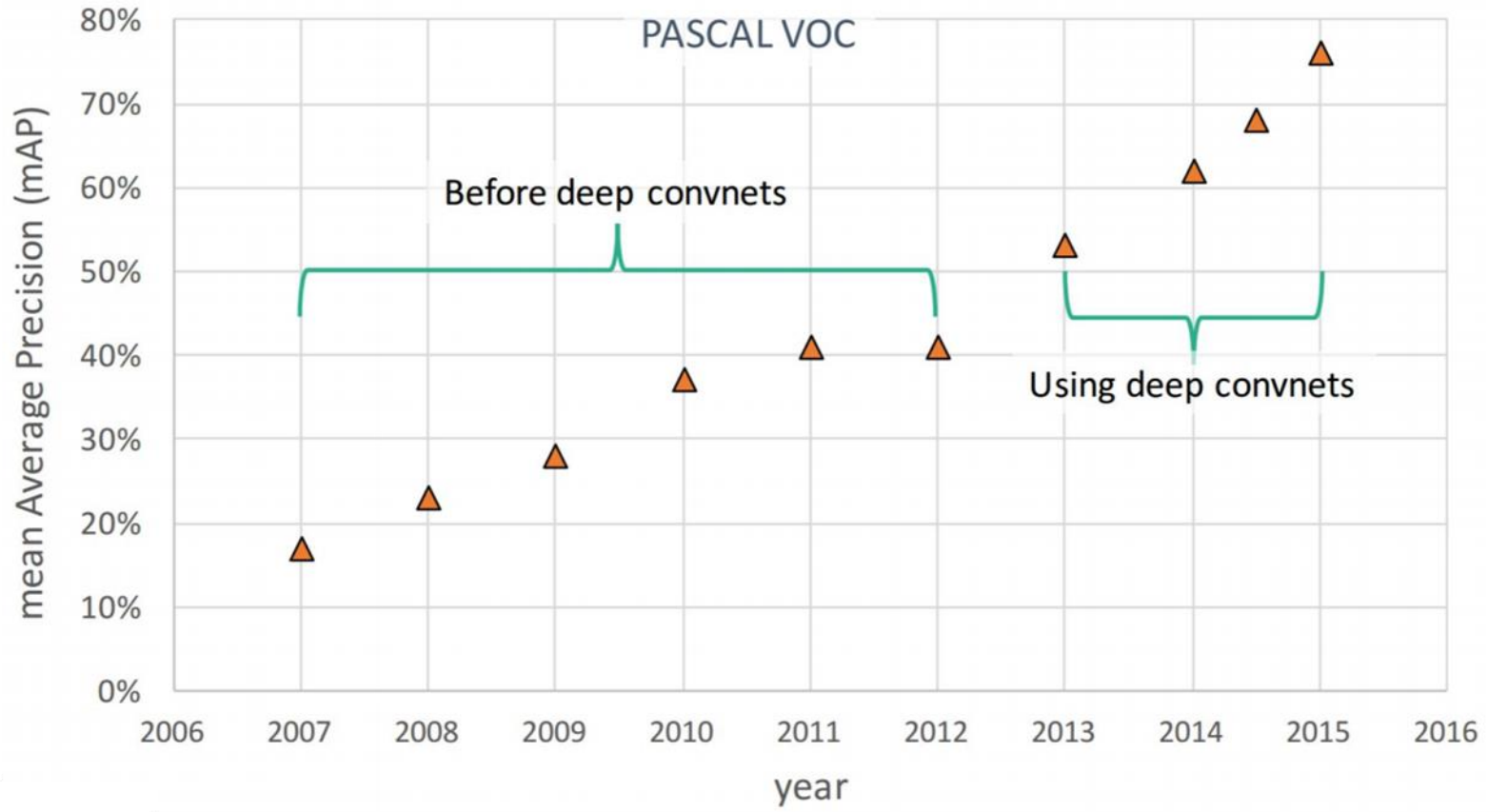
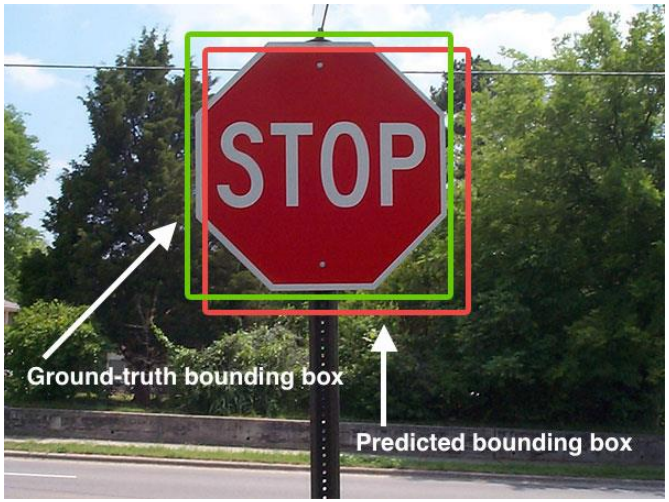


Figure copyright Ross Girshick, 2015.

Architecture (Deep) de détection d'objets

Métriques

IOU – Intersection Over Union



$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Architecture (Deep) de détection d'objets

Bases de données

Pascal VOC : 2005 / 2012

- http://host.robots.ox.ac.uk/pascal/VOC/
- M. Everingham et al. : The PASCAL Visual Object Classes Challenge: A Retrospective
- 2005 : 4 classes (1578 images - 2209 objets) – 2006 : 10 classes… 2012 : 20 classes (11,530 images - 27,450 objets)

COCO : 2015 - …

- http://cocodataset.org/#home
- Détection & segmentation
- 80 classes (200,000 images - 500,000 objets)

ImageNet – 2012

- http://www.image-net.org/
- 1000 classes (1,2 millions d'images)

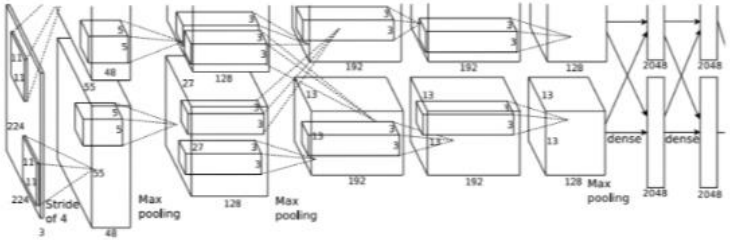
YOUTUBE (vidéos) – 2017

- https://research.google.com/youtube-bb/
- 23 classes, 5.6m bounding boxes

Architecture (Deep) de détection d'objets

Détection d'objets comme une classification: Fenêtre glissante

Appliquer un CNN à plusieurs parties de l'image, le CNN classifie chaque partie comme objet ou background.

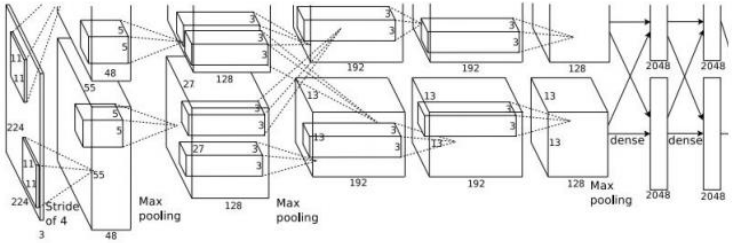


Dog? NO
Cat? NO
Background? YES

Architecture (Deep) de détection d'objets

Détection d'objets comme une classification: Fenêtre glissante

Appliquer un CNN à plusieurs parties de l'image, le CNN classifie chaque partie comme objet ou background.

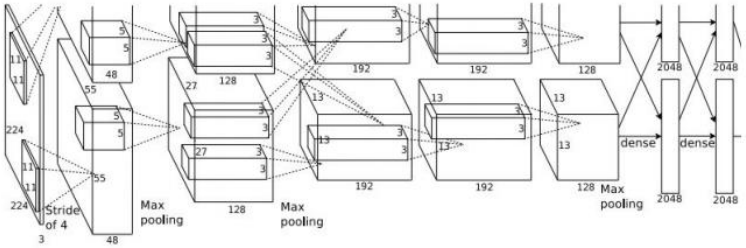


Dog? YES
Cat? NO
Background? NO

Architecture (Deep) de détection d'objets

Détection d'objets comme une classification: Fenêtre glissante

Appliquer un CNN à plusieurs parties de l'image, le CNN classifie chaque partie comme objet ou background.

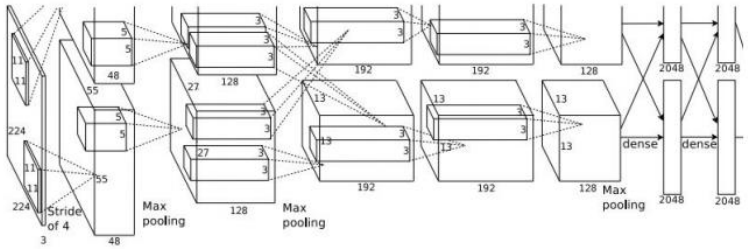


Dog? NO
Cat? YES
Background? NO

Architecture (Deep) de détection d'objets

Détection d'objets comme une classification: Fenêtre glissante

Appliquer un CNN à plusieurs parties de l'image, le CNN classifie chaque partie comme objet ou background.



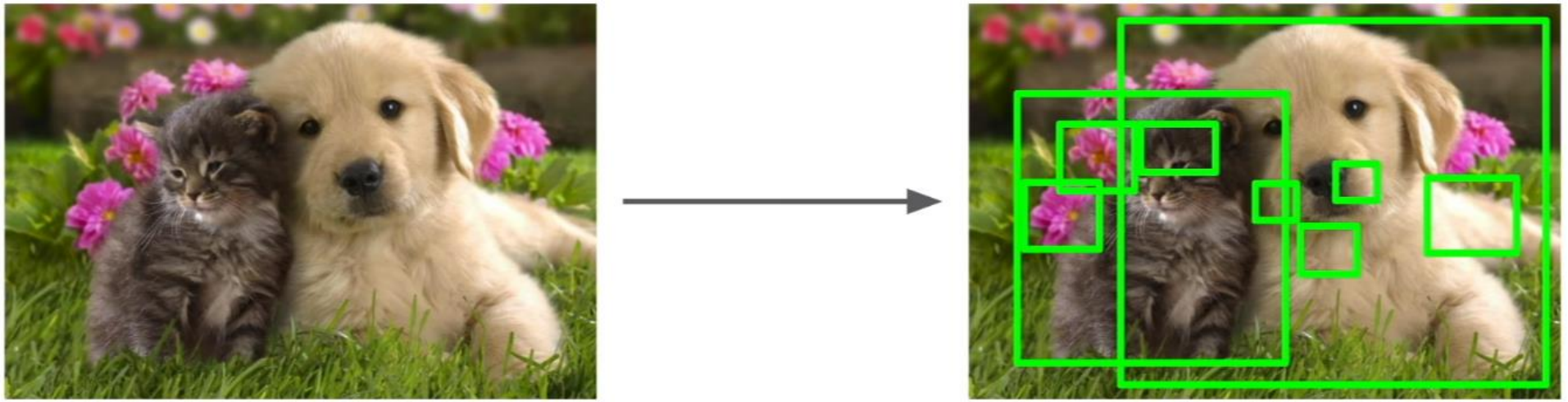
Dog? NO
Cat? YES
Background? NO

Problème: Besoin d'appliquer des CNN à un grand nombre de parties de l'image, grand nombre de tailles… Beaucoup de calcul!

Architecture (Deep) de détection d'objets

Region Proposals (Propositions de régions)

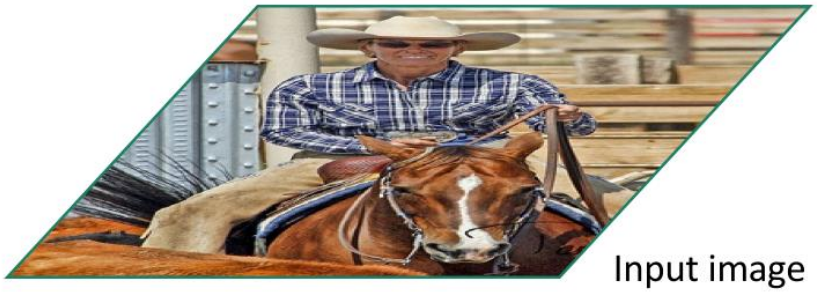
- Trouver des régions de l'image qui sont susceptibles de contenir un objet
- Relativement rapide. Selective Search donne 1000 régions en quelques secondes sur CPU



Alexe et al, "Measuring the objectness of image windows", TPAMI 2012
Uijlings et al, "Selective Search for Object Recognition", IJCV 2013
Cheng et al, "BING: Binarized normed gradients for objectness estimation at 300fps", CVPR 2014
Zitnick and Dollar, "Edge boxes: Locating object proposals from edges", ECCV 2014

Architecture (Deep) de détection d'objets

R-CNN



Input image

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

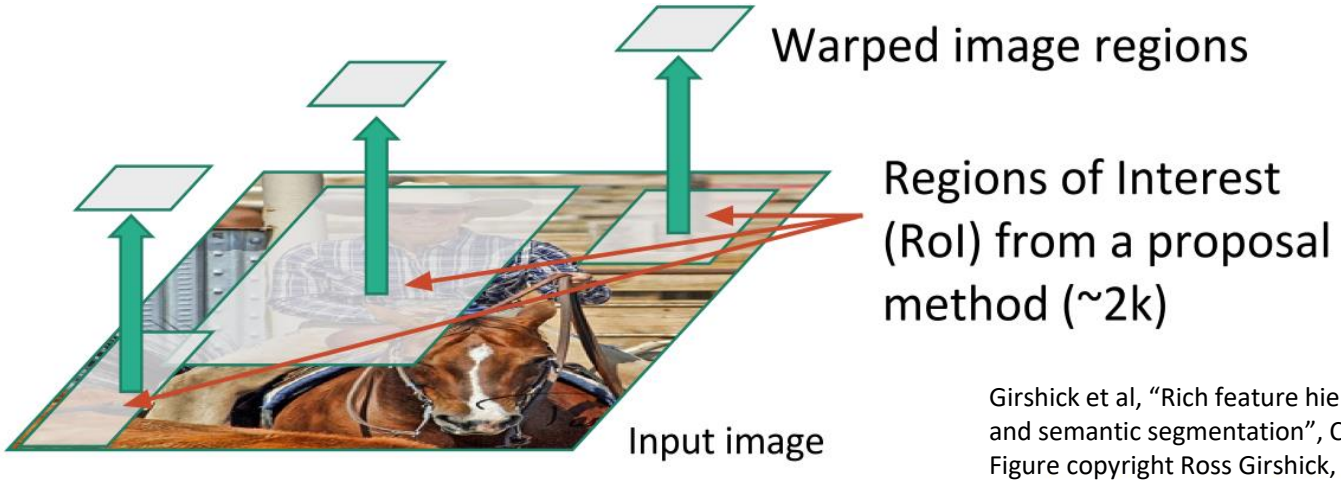
R-CNN

Regions of Interest (RoI) from a proposal method (~2k)

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

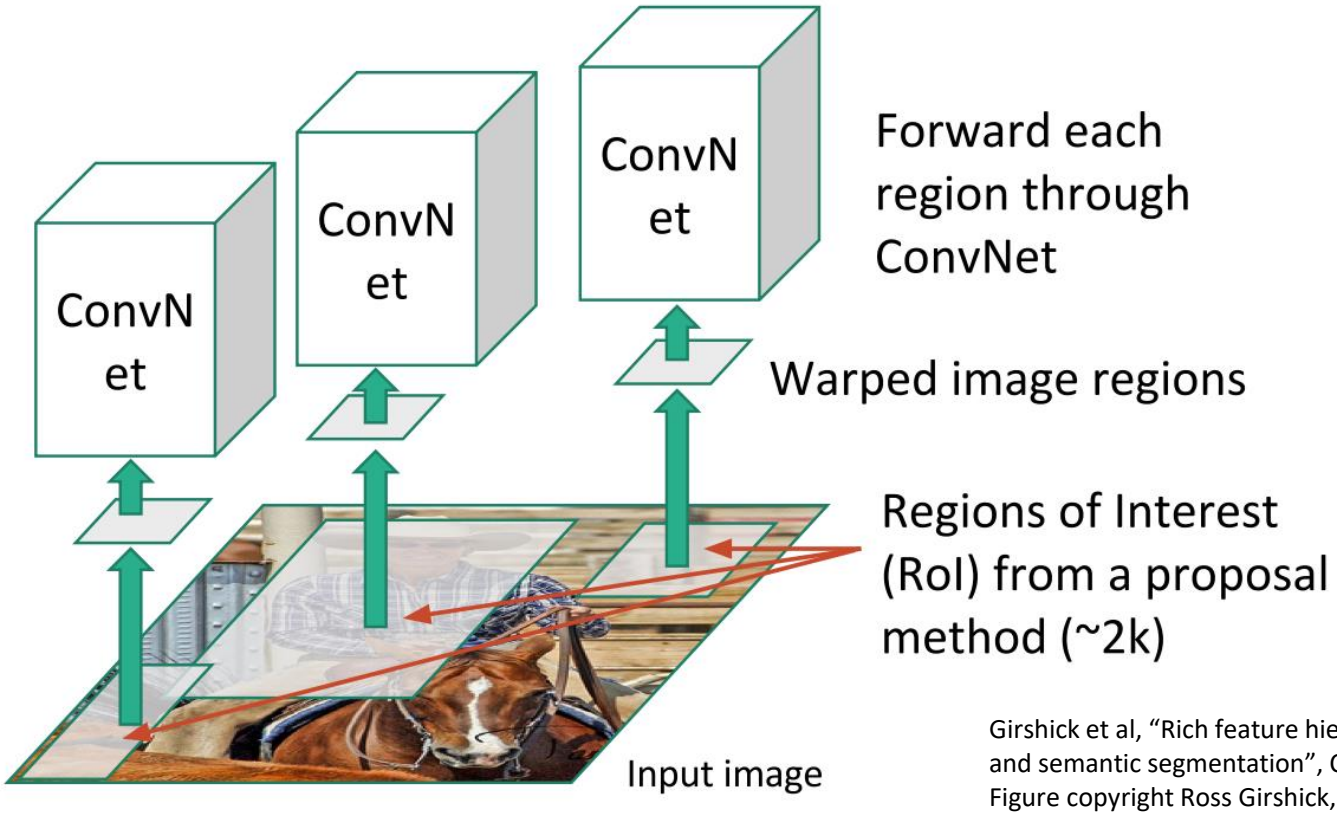
R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

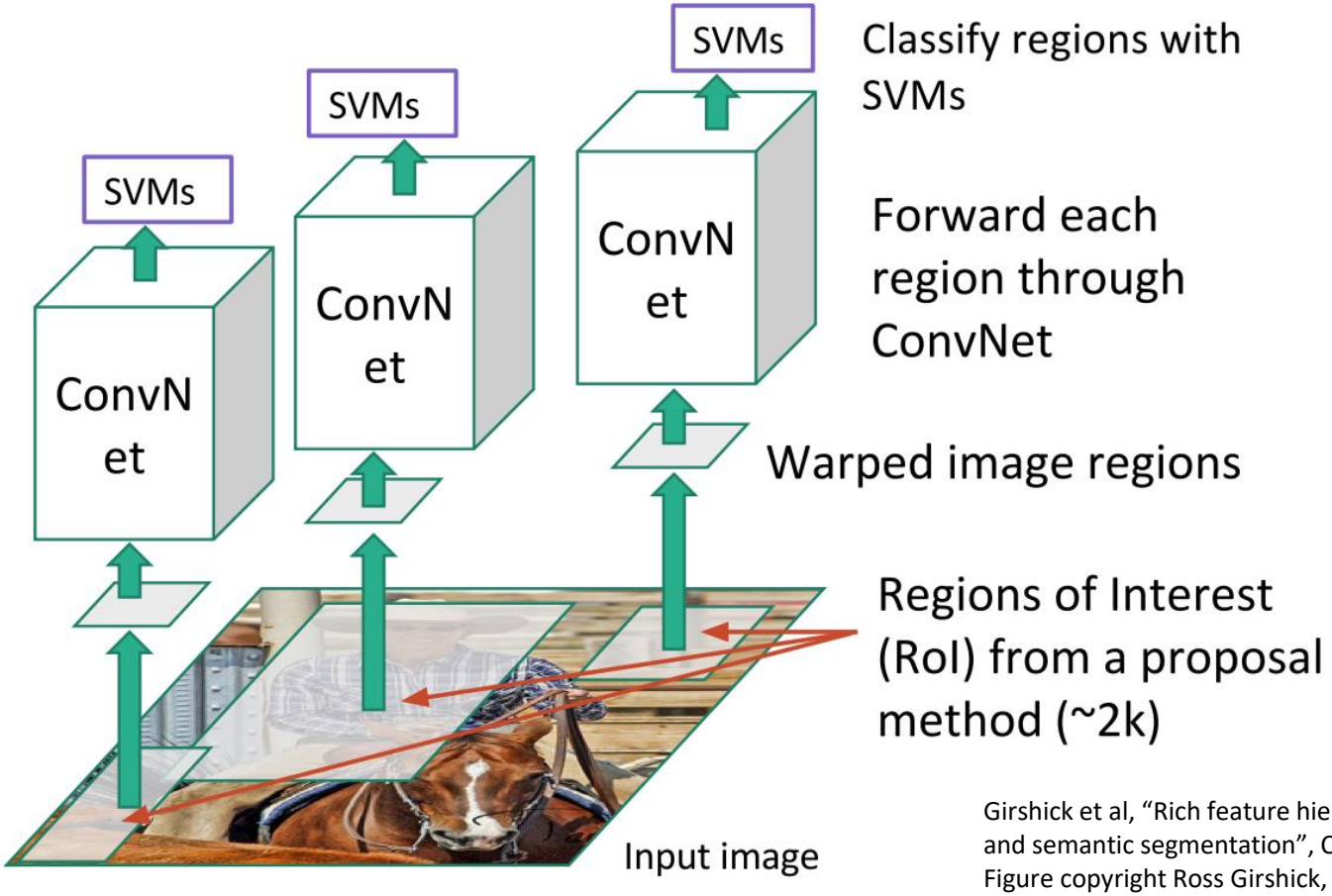
R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

R-CNN

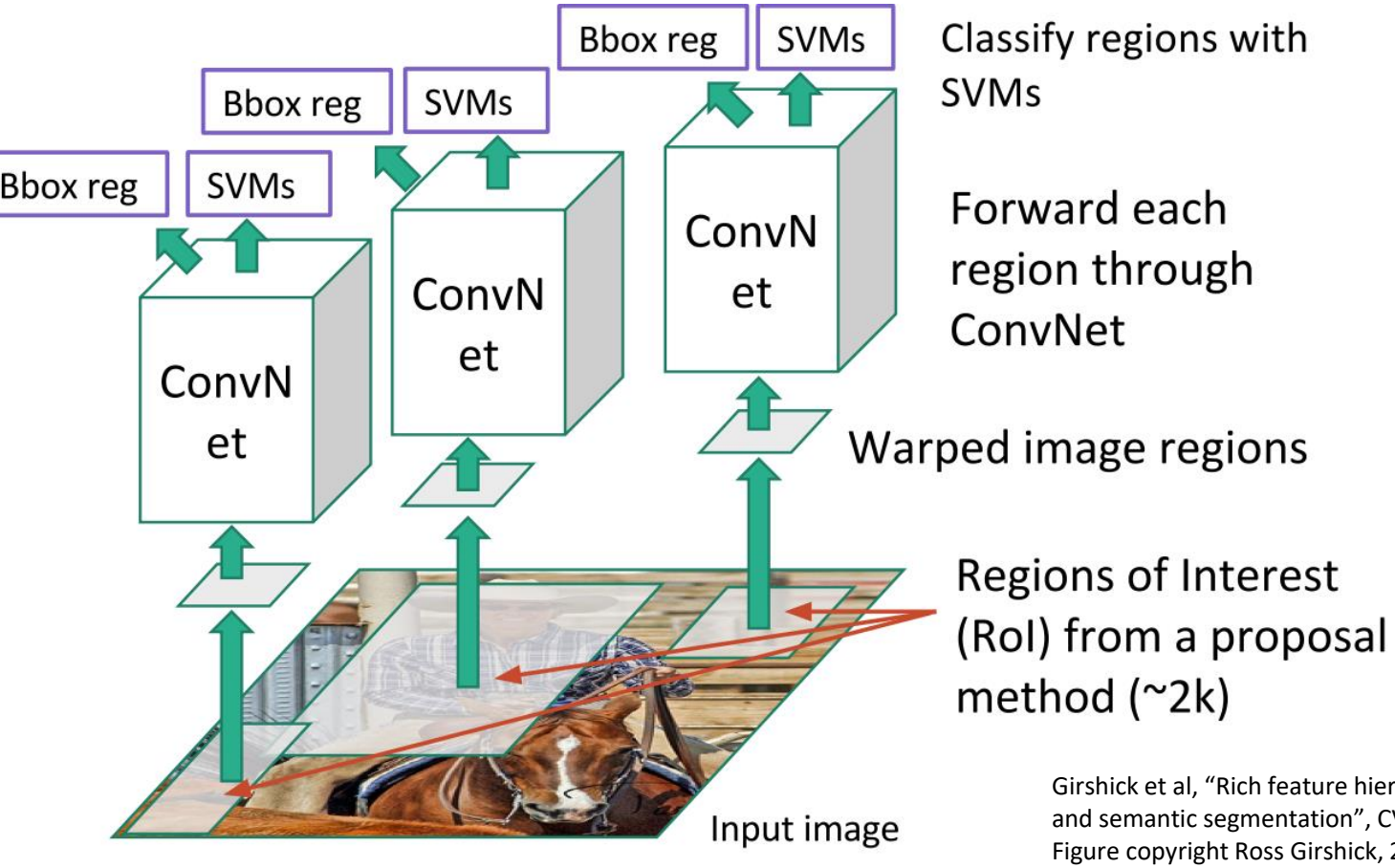


Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

R-CNN

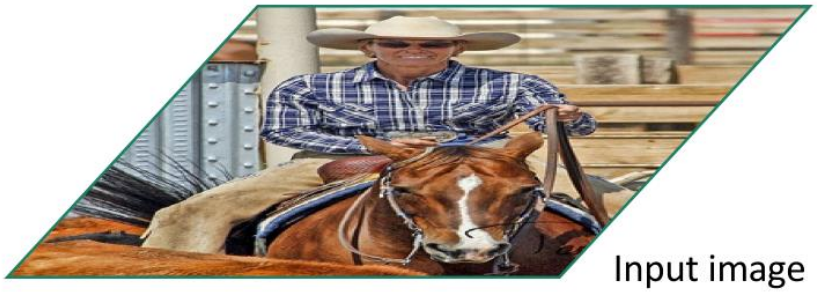
Régression Linéaire pour les Bounding Box



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

Fast R-CNN

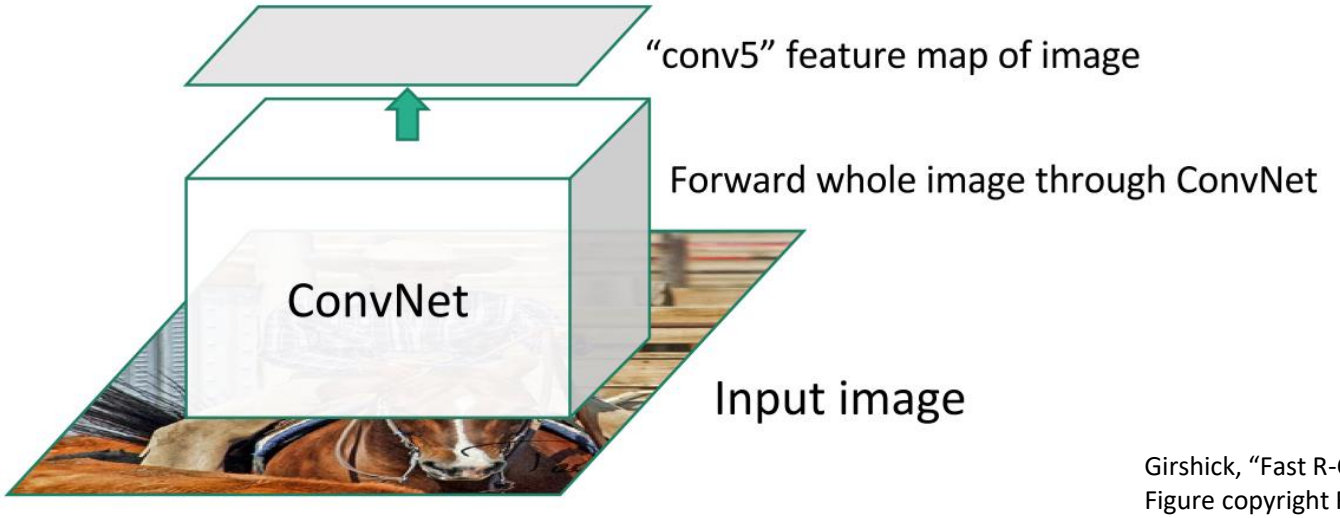


Input image

Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

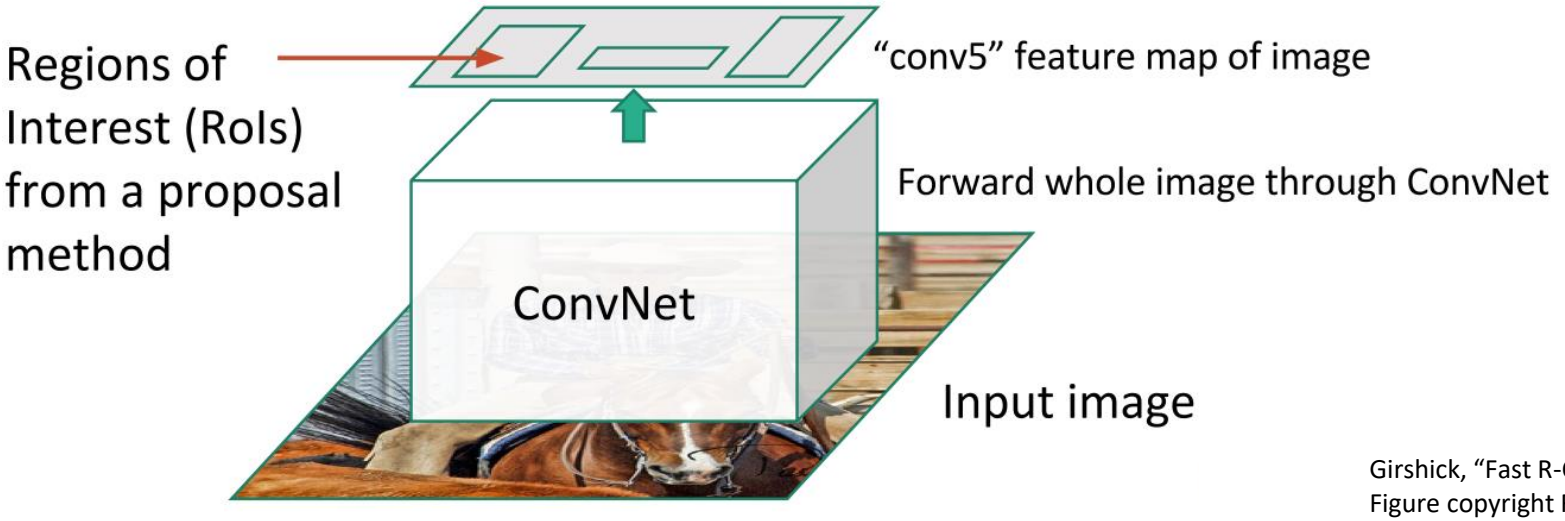
Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

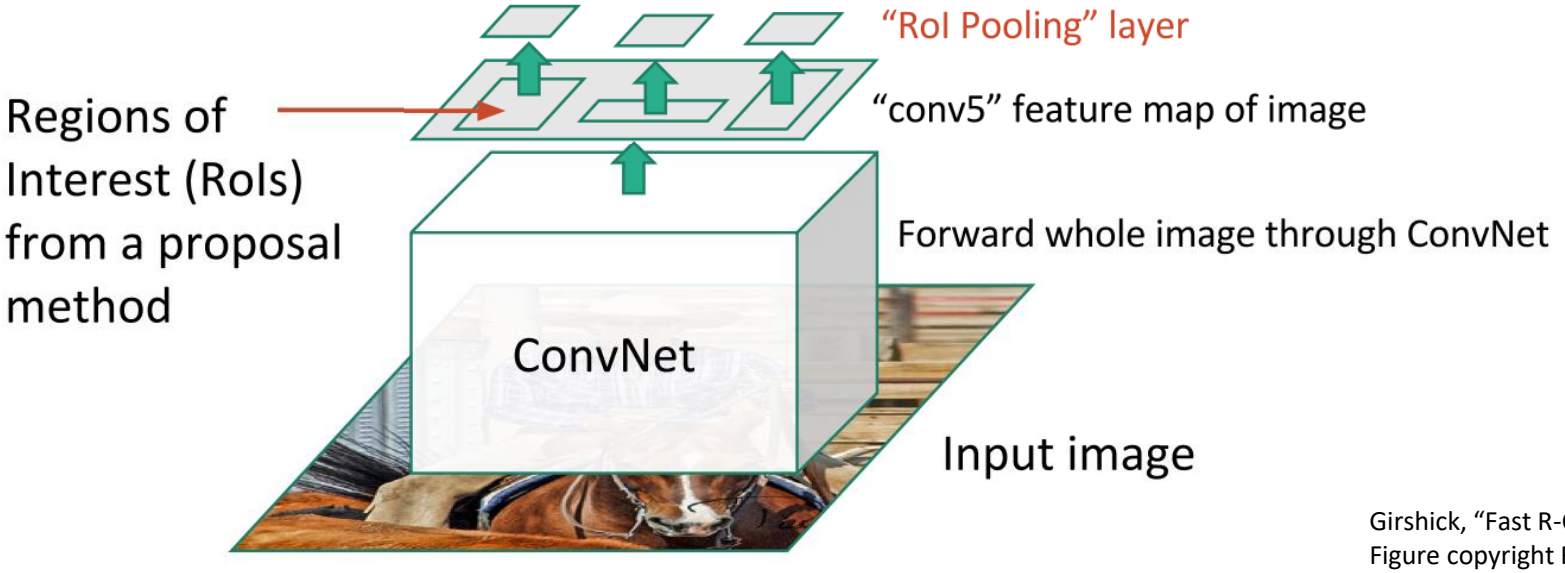
Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

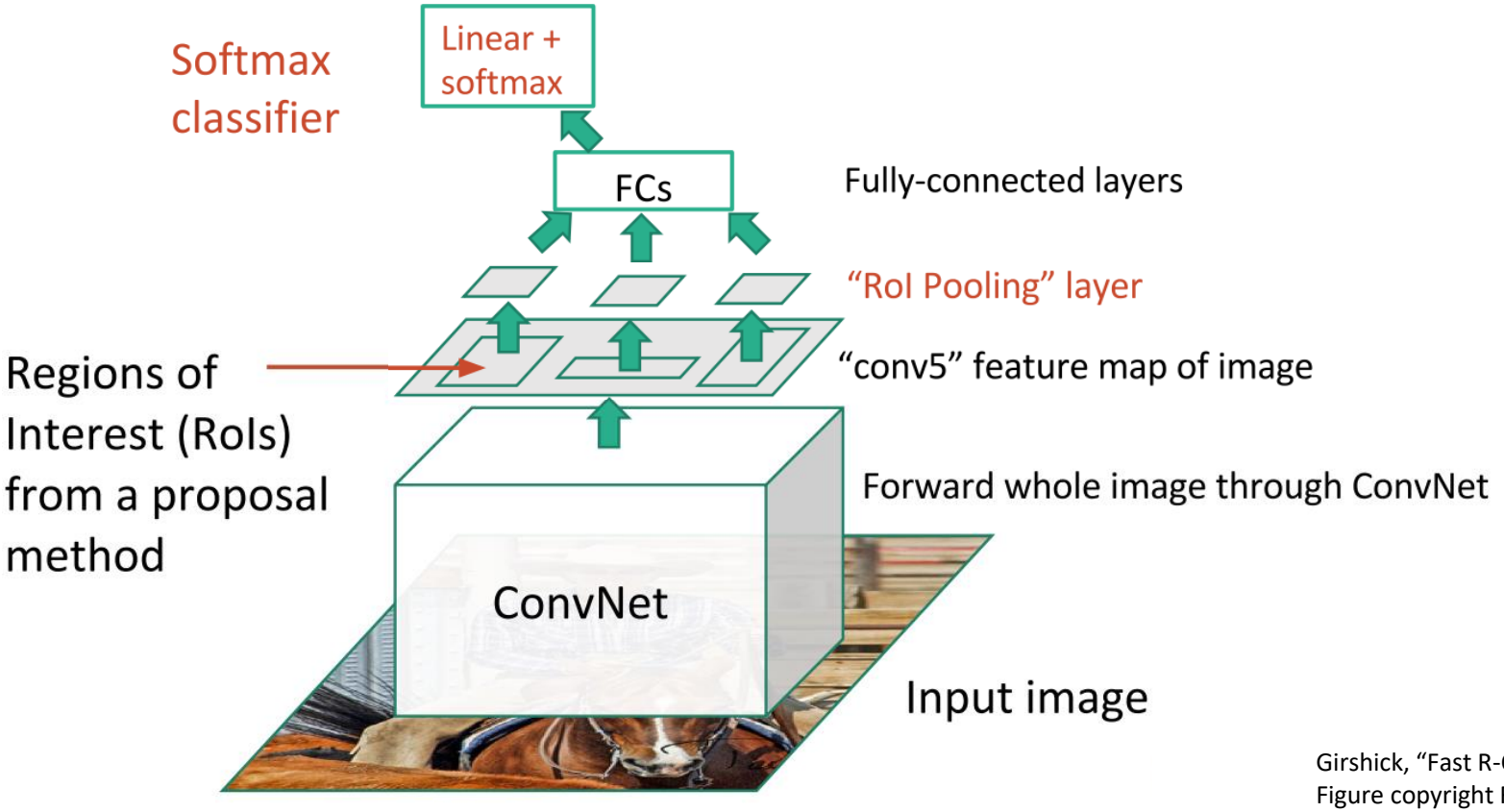
Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

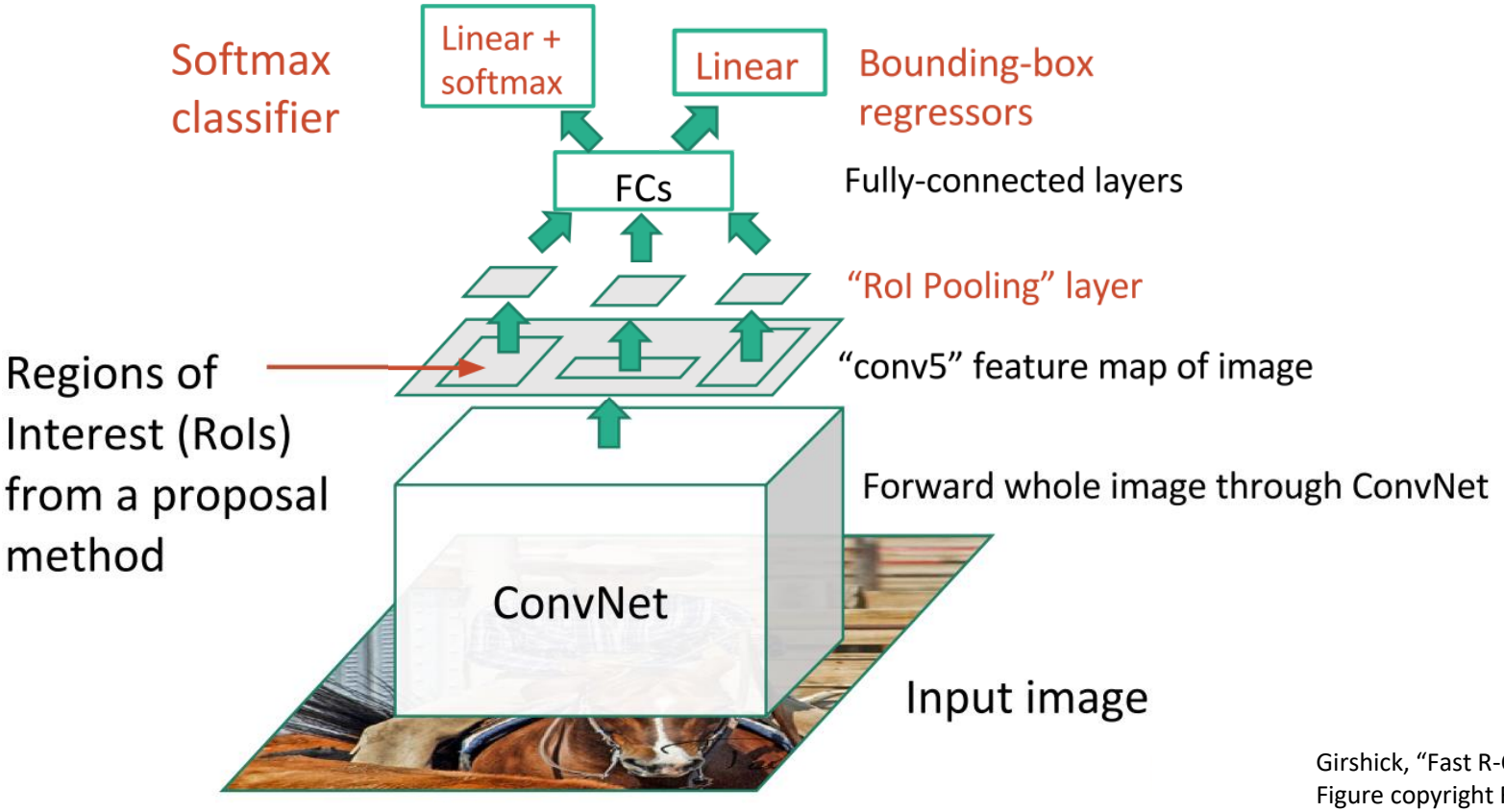
Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

Fast R-CNN

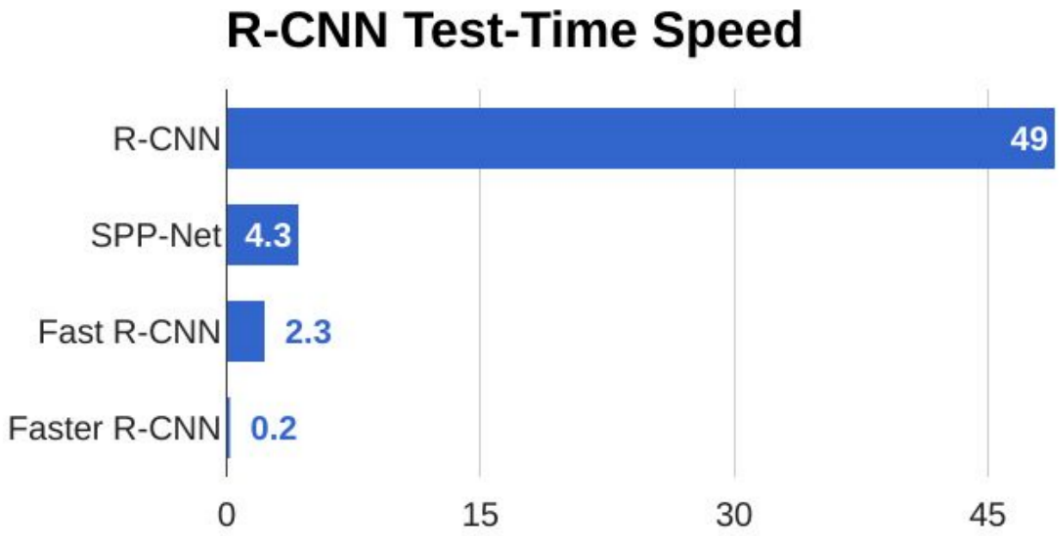


Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source.

Architecture (Deep) de détection d'objets

Faster R-CNN

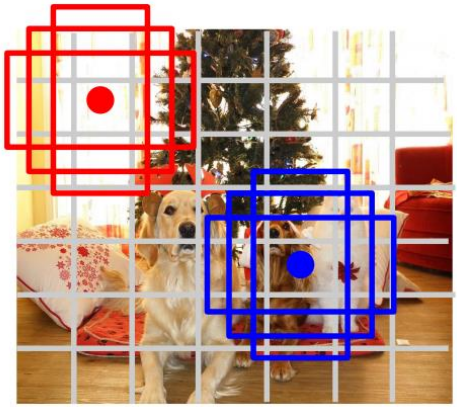
Laisser le CNN chercher lui-même les propositions



Architecture (Deep) de détection d'objets

Détection sans Propositions : YOLO / SSD

Input image
3 x H x W



Divide image into grid
7 x 7

Un ensemble de ***B* boxes de base** centrées à chaque cellule de la grille
Ici ***B* = 3**

Dans chaque grille:

- Régression à partir de chacune des *B* boxes de base à une box finale de 5 paramètres:
  (dx, dy, dh, dw, confidence)
- Prédire les scores pour chacune des C classes (Background inclue)

Output:
7 x 7 x (5 * B + C)

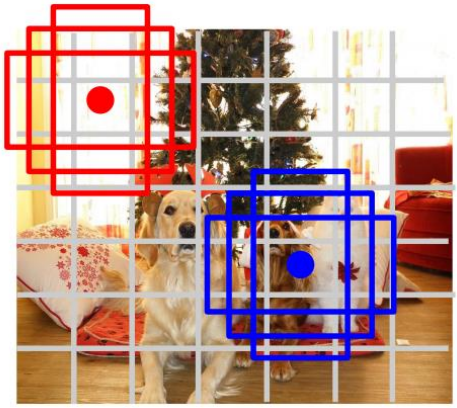
Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

Architecture (Deep) de détection d'objets

Détection sans Propositions : YOLO / SSD

Aller d'une image d'entrée à un vecteur de scores avec un seul réseau conv.

Input image
3 x H x W



Divide image into grid
7 x 7

Un ensemble de ***B* boxes de base** centrées à chaque cellule de la grille
Ici ***B* = 3**

Dans chaque grille:

- Régression à partir de chacune des *B* boxes de base à une box finale de 5 paramètres:
  (dx, dy, dh, dw, confidence)
- Prédire les scores pour chacune des C classes (Background inclue)

Output:
7 x 7 x (5 * B + C)

Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

Architecture (Deep) de détection d'objets

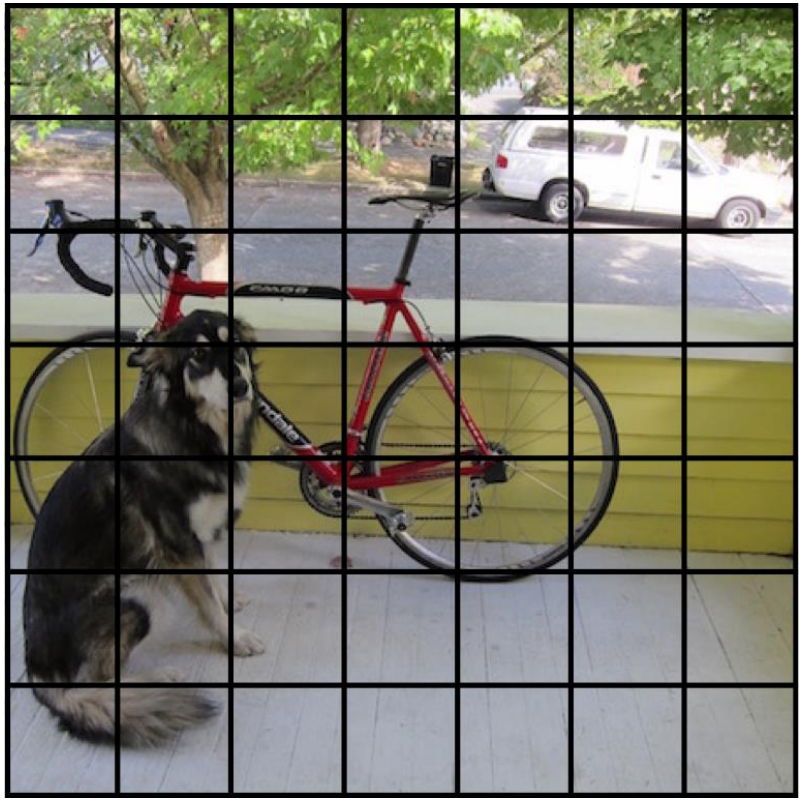
YOLO

Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

On divise l'image en grille (ex: 7*7)

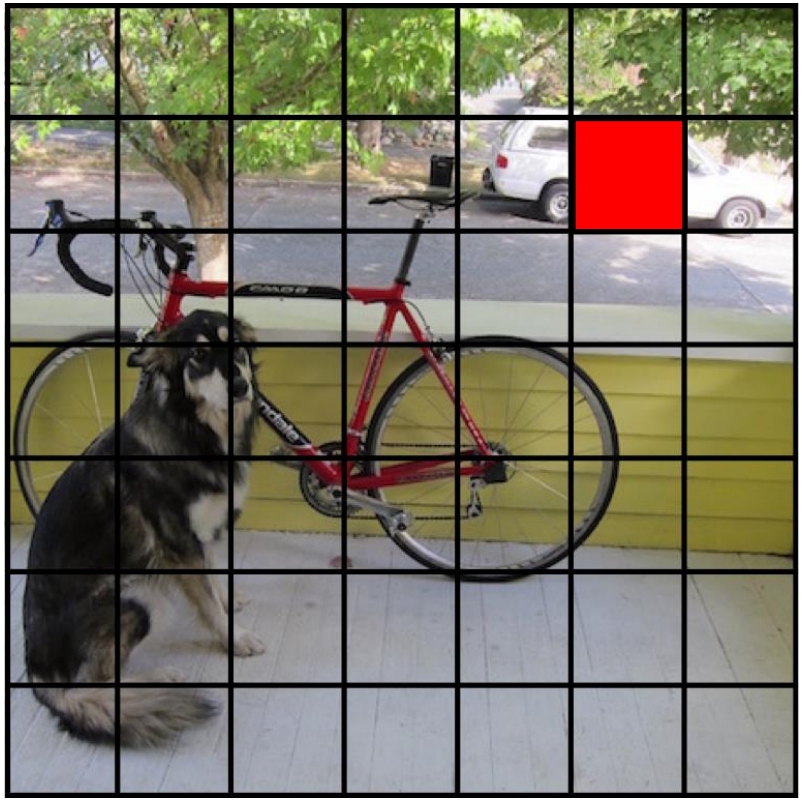


Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

Chaque cellule prédit des boxes (ex: 2) et des scores: P(Objet)



Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

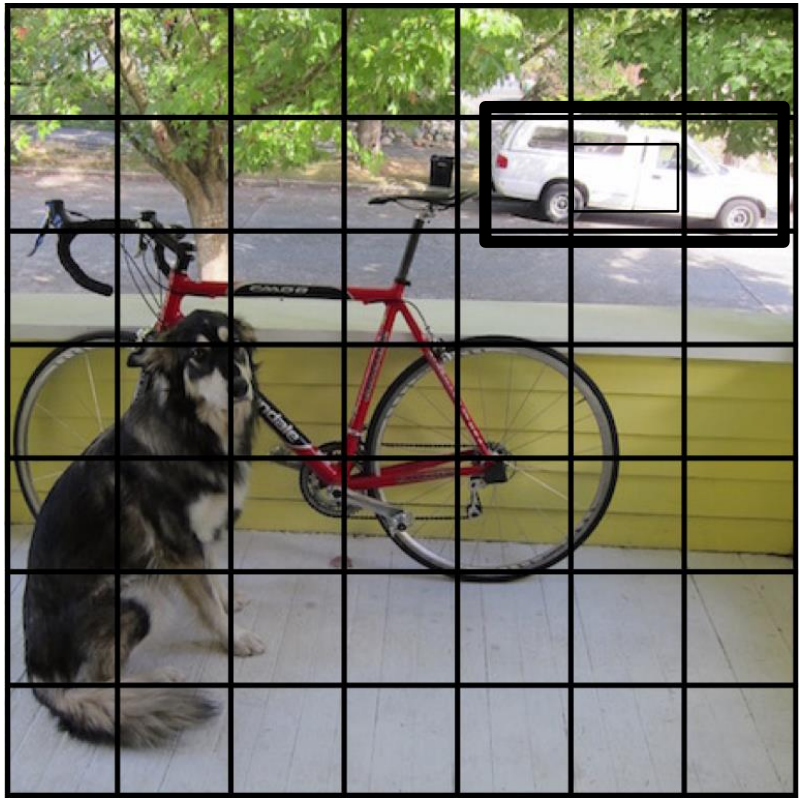
Chaque cellule prédit des boxes (ex: 2) et des scores: P(Objet)

Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

Chaque cellule prédit des boxes (ex: 2) et des scores: P(Objet)

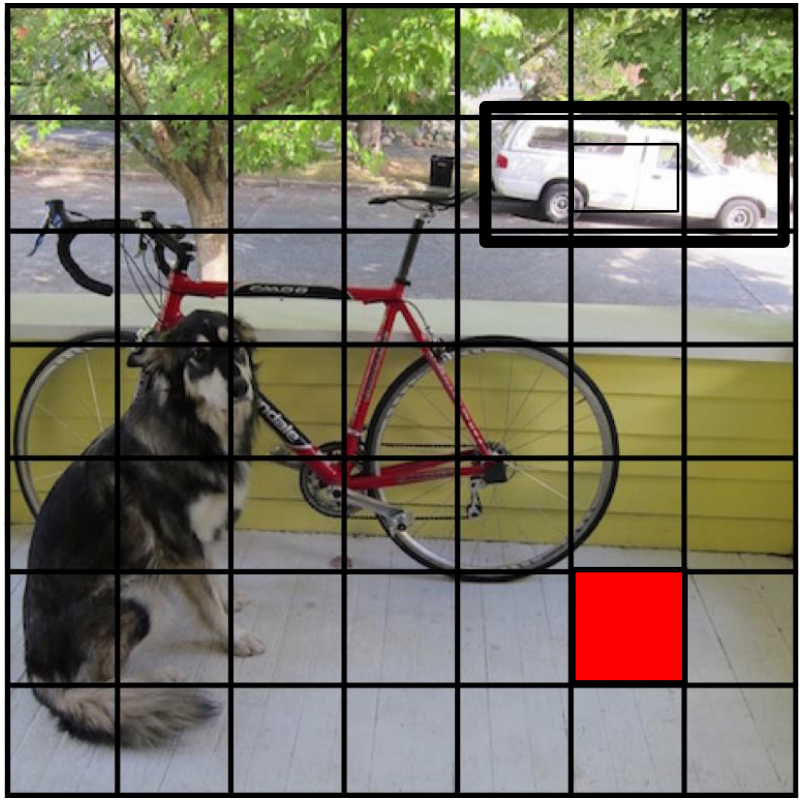


Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

Chaque cellule prédit des boxes (ex: 2) et des scores: P(Objet)



Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

Chaque cellule prédit des boxes (ex: 2) et des scores: P(Objet)

Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

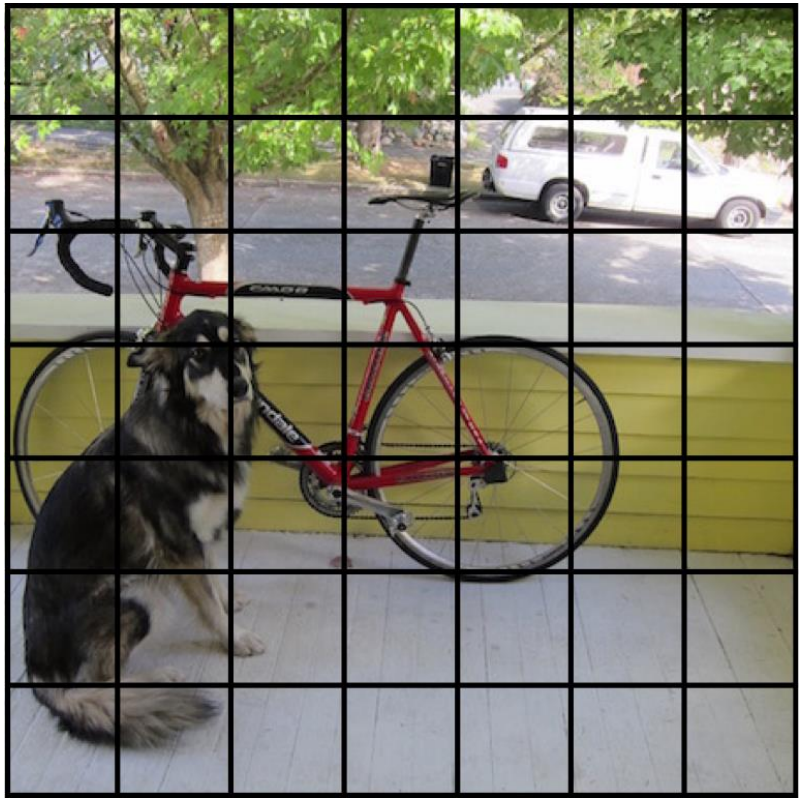
Chaque cellule prédit des boxes (ex: 2) et des scores: P(Objet)

Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

Architecture (Deep) de détection d'objets

YOLO

Chaque cellule prédit aussi des probabilités de classes

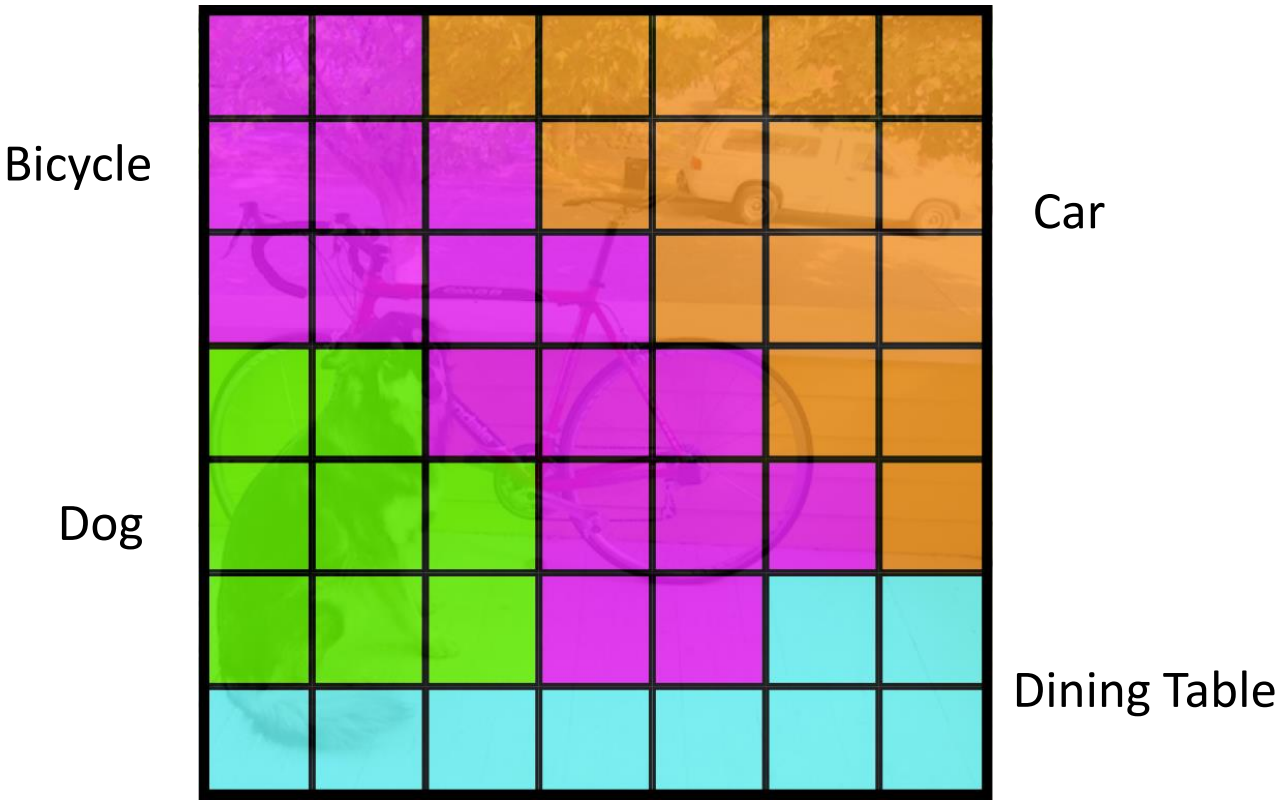


Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

Chaque cellule prédit aussi des probabilités de classes (P(Car |Objet)

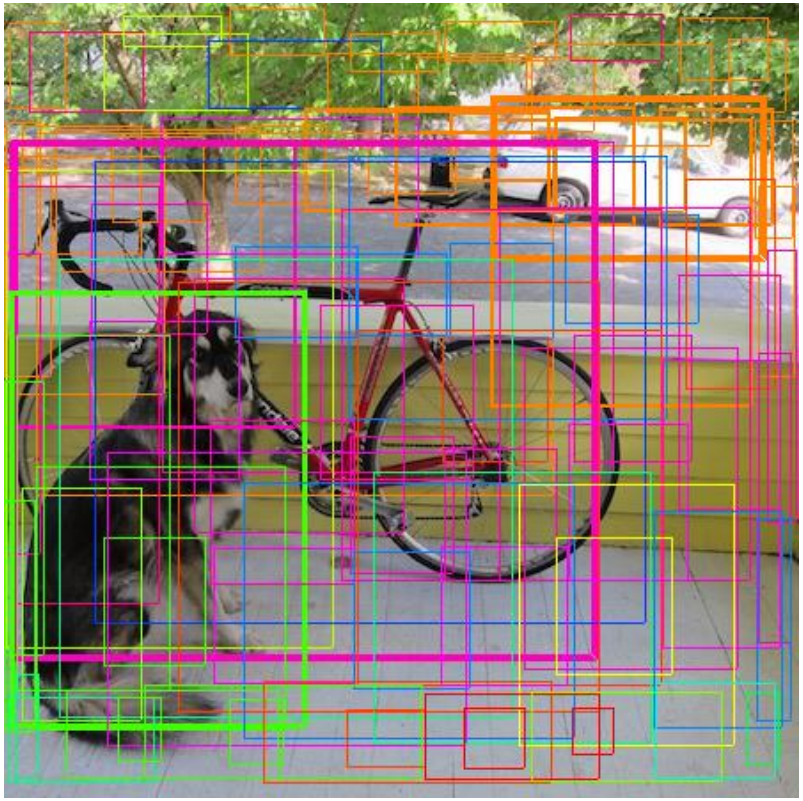


Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

Ensuite on combine les deux prédictions (boxes & classe)

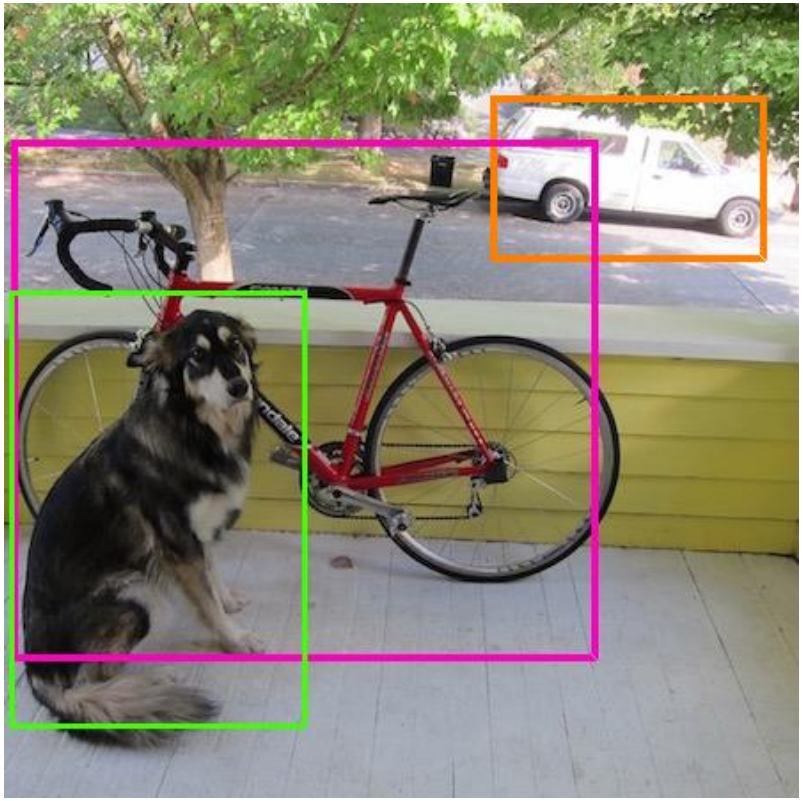


Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

Finalement on ajoute un seuillage de détection (NMS – Non-Maximum Suppression)

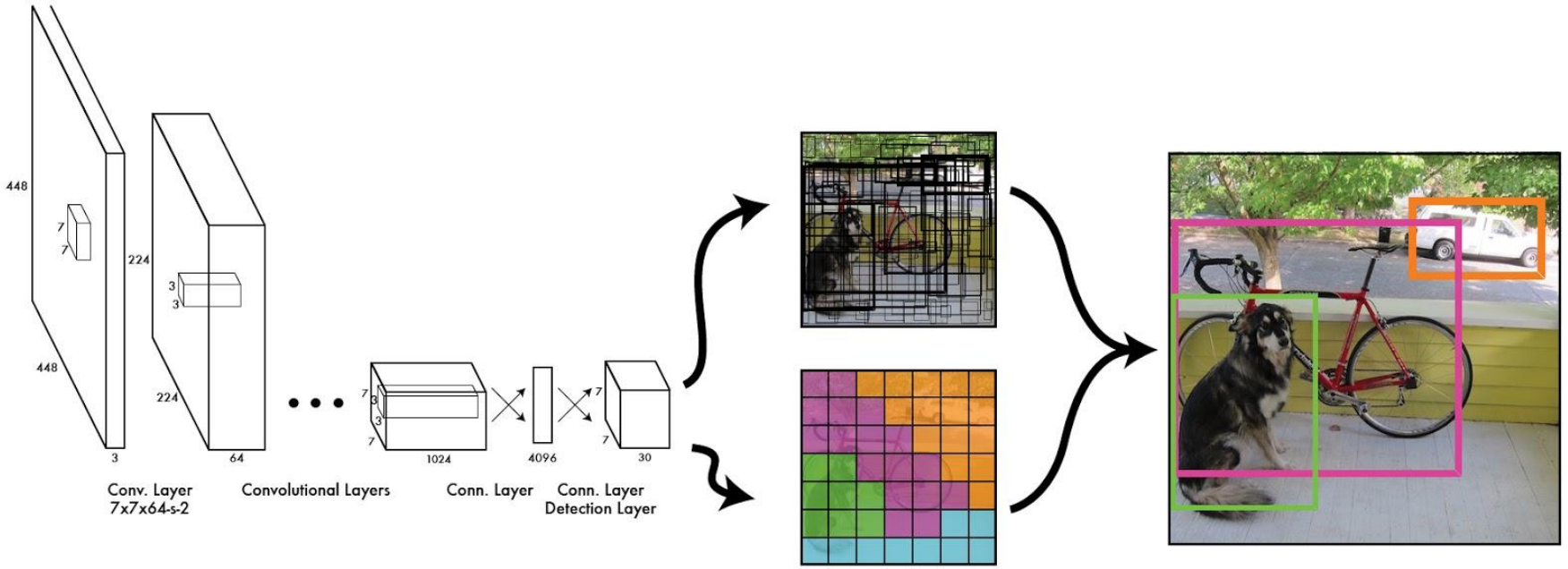


Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

YOLO

Un seul réseau est entrainé à faire tout le travail de détection



Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

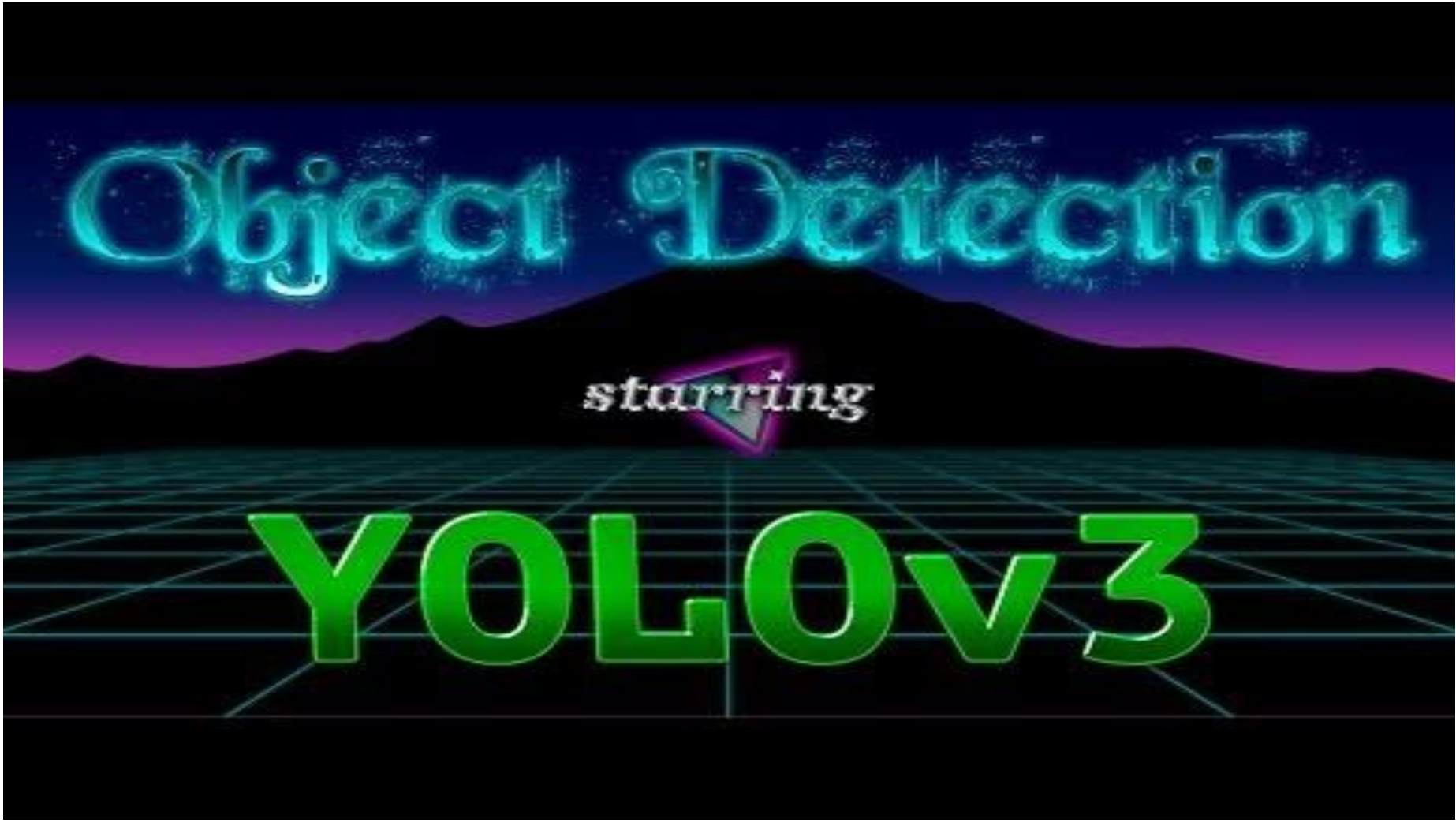
YOLO

Très rapide, mais pas assez précis

	Pascal 2007 mAP	**Speed**	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	63.4	45 FPS	22 ms/img

Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016

Architecture (Deep) de détection d'objets

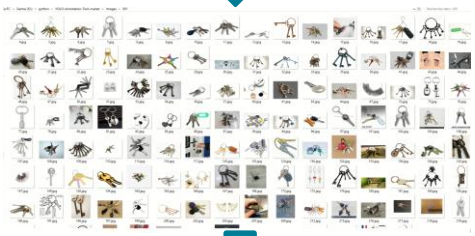


Comment annoter les données ?

Google Image

Rechercher

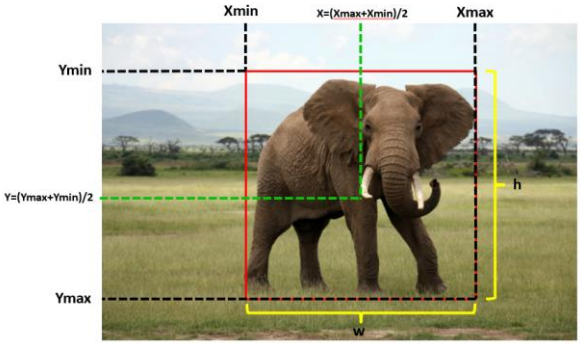
Génération des images



Annotation

class	xmin	ymin	xmax	ymax
0	6	21	293	202

Sortie : fichier txt

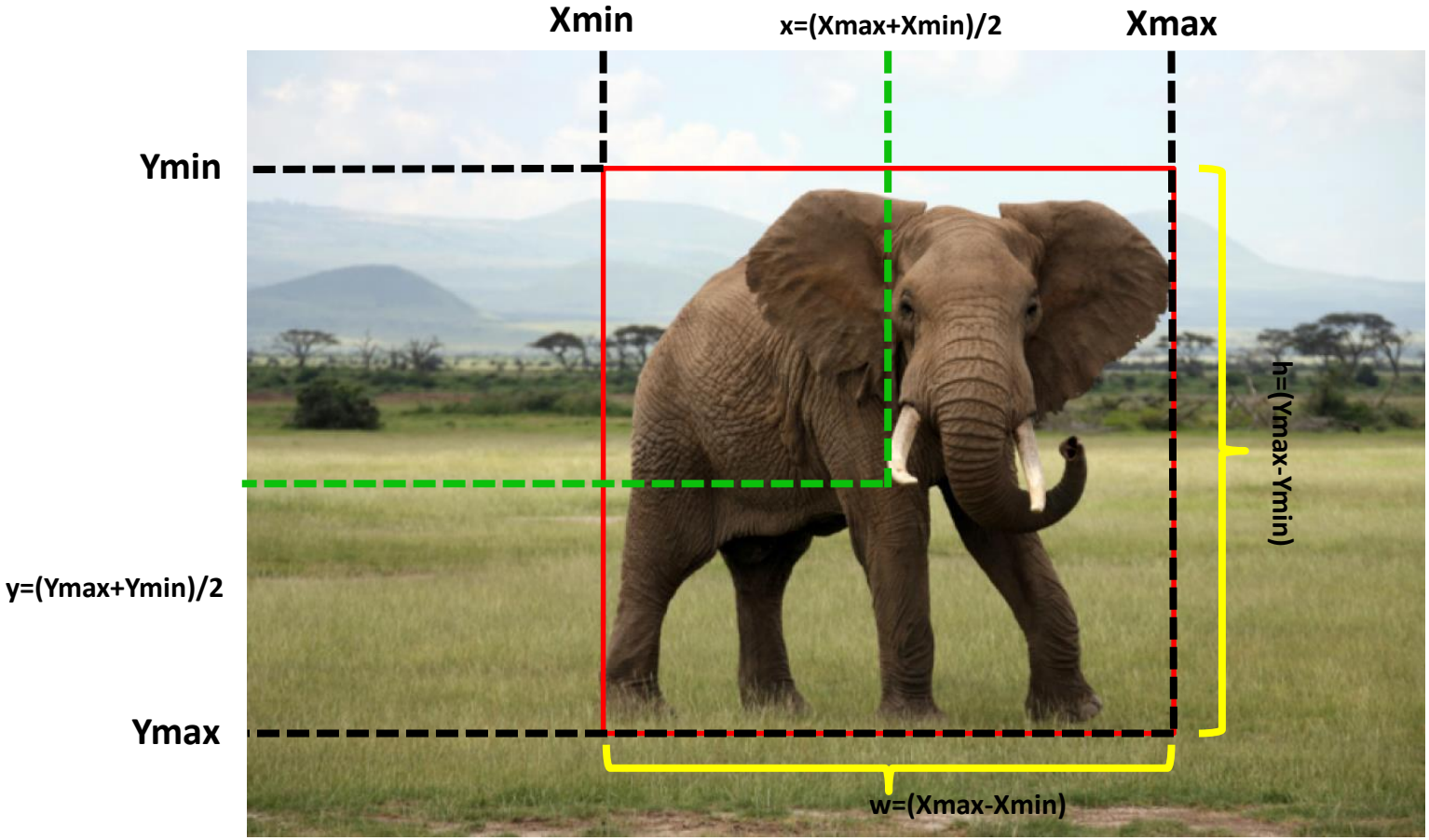


Annotation format YOLO

Résultat final
Fichier txt

class	x	w	y	h
0	0.498333333333	0.536057692308	0.956666666667	0.870192307692

Comment annoter les données ?



x=x*(1/image.width)
w=w*(1/image.width)
y=y*(1/image.height)
h=h*(1/image.height)

class	x	w	y	h
0	0.498333333333	0.536057692308	0.956666666667	0.870192307692

- **Outils d'annotation manuelle :**

  a. LabelMe. http://labelme.csail.mit.edu/Release3.0/
  b. labelImg : https://github.com/tzutalin/labelImg
  c. Supervisely : https://supervise.ly/
  d. https://www.pyimagesearch.com/2017/12/04/how-to-create-a-deep-learning-dataset-using-google-images/
  e. Power IA (Semi-Automatique) : https://github.com/IBM/powerai-vision-object-detection#2-login-to-powerai-vision

- **Détection d'objets :**
  a. https://arxiv.org/pdf/1611.10012.pdf
  b. https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852
  c. https://medium.com/comet-app/review-of-deep-learning-algorithms-for-image-classification-5fdbca4a05e2

- **SSD :** https://github.com/tensorflow/models/tree/master/research/object_detection

- **Yolo :** https://pjreddie.com/darknet/yolo/

Anaconda pour ceux qui souhaitent travailler sur leurs PCs

- Anaconda sous Windows : https://medium.com/@GalarnykMichael/install-python-on-windows-anaconda-c63c7c3d1444
- Anaconda sous Ubuntu 16.04 : https://www.digitalocean.com/community/tutorials/how-to-install-the-anaconda-python-distribution-on-ubuntu-16-04

Architecture (Deep) de détection d'objets

Encore plus…

- DCN (Deformable Conv. Networks) – 2017 : https://arxiv.org/abs/1703.06211
- DSSD (Deformable Conv. Networks) – 2017 : https://arxiv.org/abs/1701.06659
- Fast YOLO : https://arxiv.org/abs/1709.05943
- FSSD: https://arxiv.org/abs/1712.00960
- Feature-Fused SSD : https://arxiv.org/abs/1709.05054
- Learning Transferable Architectures for Scalable Image Recognition - 2017 : RNN learn CNN : https://arxiv.org/abs/1707.07012
- RetinaNet – 2018 : https://arxiv.org/abs/1708.02002
- YOLO V3 – 2018 : https://arxiv.org/abs/1804.02767

PLAN

I. Préparation et division des données

II. Paramètres d'entrainement

III. Architectures de localisation

IV. **Exemple de localisation avec Yolo**

V. Accès à la plateforme Floydhub

Exemple de localisation avec Yolo

PLAN

I. Préparation et division des données

II. Paramètres d'entrainement

III. Architectures de localisation

IV. Exemple de localisation avec Yolo

V. **Accès à la plateforme Floydhub**

Manuel partagé via Moodle (Partie 2)


Merci