Reconocimiento de objetos

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf
https://machinelearningmastery.com/object-recognition-with-deep-learning/
https://www.youtube.com/watch?v=nDPWywWRIRo

Contenido

Teoría

- Introducción y definiciones
- Object Localization - Keypoints regression
- Object detection - Region Proposals
- Object detection - YOLO/SSD

Implementación

- Clasificación
- Localización + Clasificación (Keypoints)
- Localización + Clasificación + world
- YOLO - Detección de objetos

Algunas definiciones

Clasificación de imágenes: Predecir la clase de un objeto en una imagen.

- Entrada: Imagen
- Salida: Etiqueta. (Ej: Perro, Gato, Auto, Peaton)

Localización de objetos: Detectar la presencia de objetos en una imagen e indicar su ubicación con una ‘bounding box’

- Entrada: Imagen
- Salida: Una o más bounding boxes (X, Y, alto y ancho)

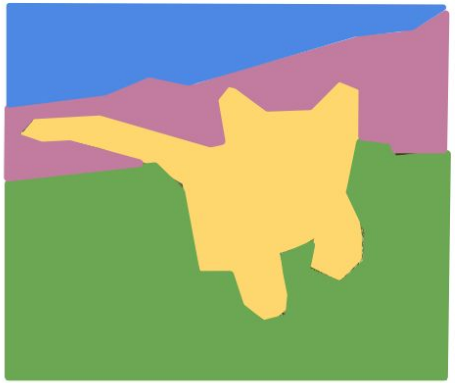
Detección de objetos: Detectar la clase de los objetos y su ubicación (Todos los autos en la foto, todos los peatones en la foto)

- Entrada: Imagen
- Salida: Todas las bounding boxes de la clase

Reconocimiento de objetos: Se lo suele usar como sinónimo de detección de objetos aunque a veces se considera que Detección son todos objetos de la misma clase. Dependiendo de la bibliografía esto se interpreta diferente. Para nosotros: Detección de objetos = Reconocimiento de objetos

Diferencias con segmentación

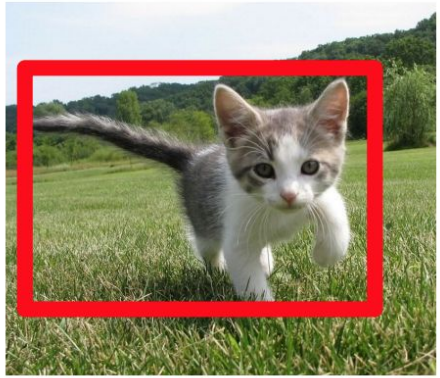
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

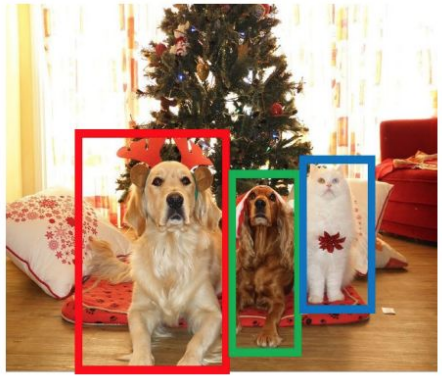
Classification + Localization



CAT

Single Object

Object Detection



DOG, DOG, CAT

Instance Segmentation

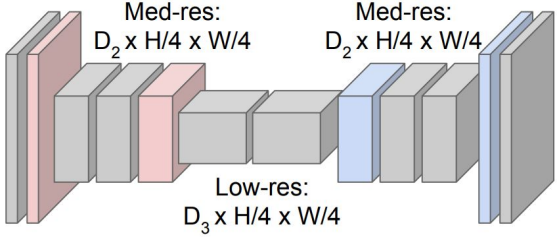
DOG, DOG, CAT

Multiple Object

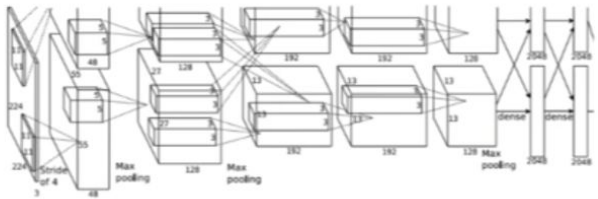
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Diferencias entre las técnicas

Segmentación: Modelo donde entrada y salida tienen la misma dimensión y se realiza clasificación a nivel de pixel



Detección: Modelo donde la entrada es la imagen y la salida son las clases encontradas + las bounding boxes



CAT: (x, y, w, h)

Sliding window

- Entreno una red que solo clasifica y la voy pasando por la imagen
- De qué tamaño elijo la ventana?
- Que aspect ratio?
- Con que stride avanzo?
- Cuanto procesamiento necesito?
- Simplifiquemos el problema -> Solo encontrar la ubicación de un objeto, o un número conocido

Object Localization

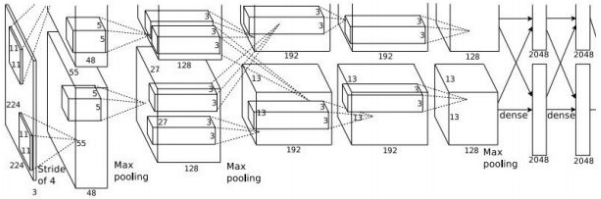
Keypoints regression

Localización de objetos

- Estimación de puntos clave por medio de una **regresión**
- Necesidad de tener una cantidad conocida a priori de puntos a determinar
- Multitask loss (L2 + Categorical Cross-Entropy)

Classification + Localization

This image is CC0 public domain



Correct label:
Cat

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Fully Connected:
4096 to 1000

Softmax Loss

Multitask Loss

+ → **Loss**

Vector:
4096

Fully Connected:
4096 to 4

Box Coordinates
(x, y, w, h)

L2 Loss

Correct box:
(x', y', w', h')

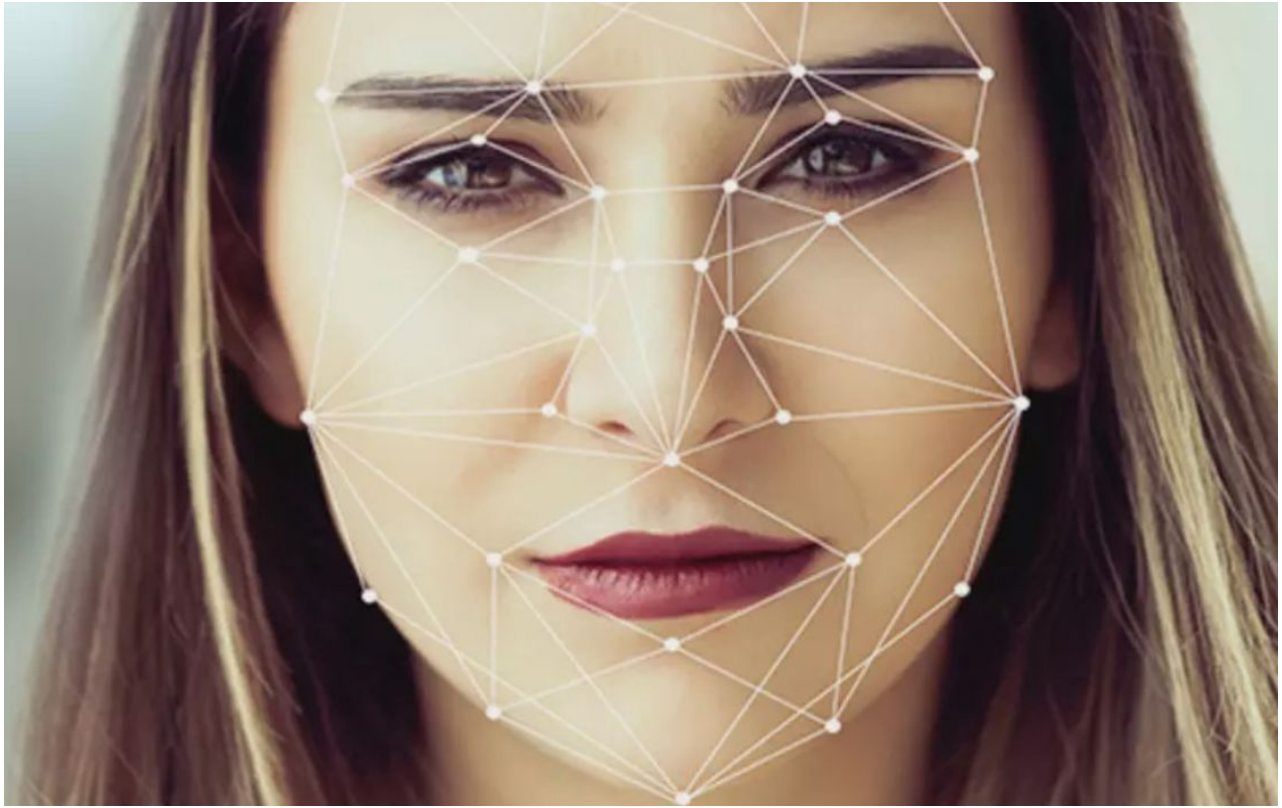
Treat localization as a regression problem!

Pose detection

Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

Facial Keypoints detection



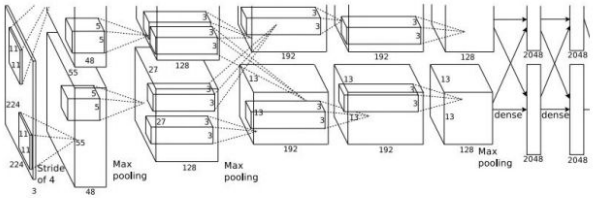
Instagram
Snapchat

Object Detection

Region Proposals

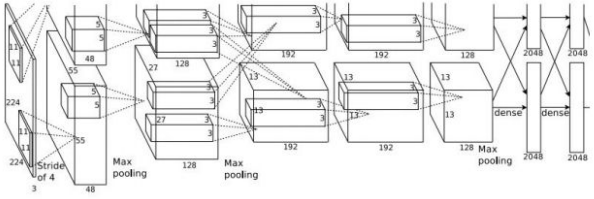
Object Detection as Regression?

Each image needs a different number of outputs!



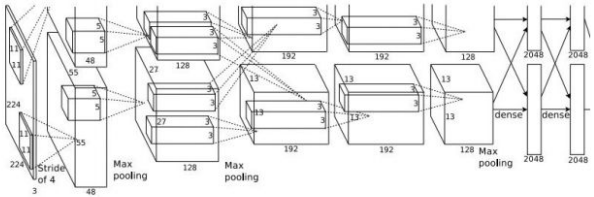
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

16 numbers



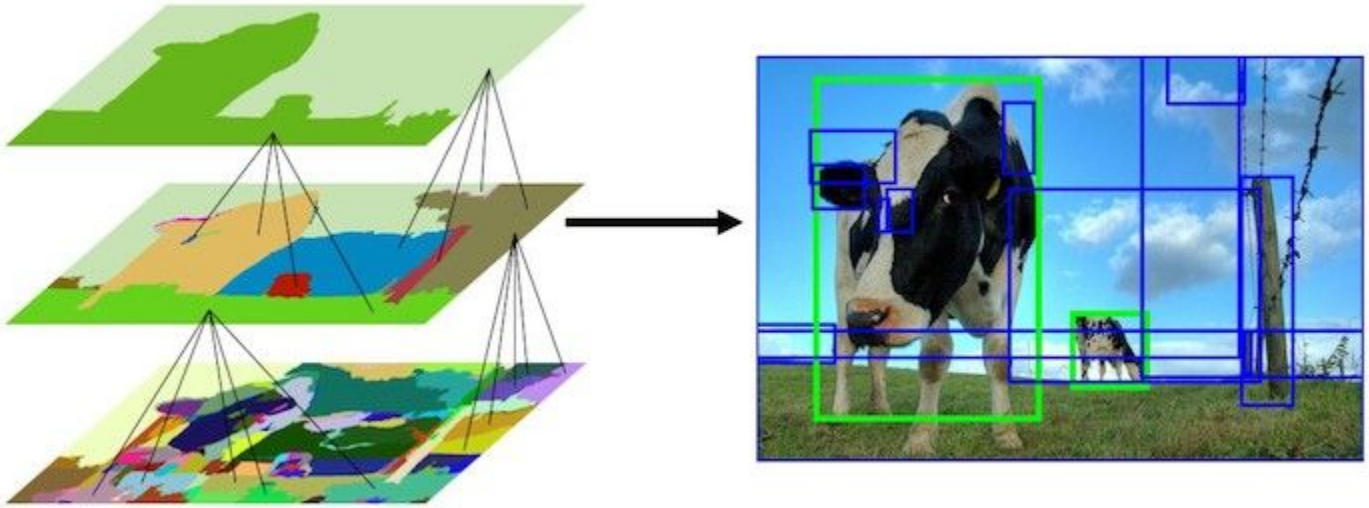
DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

Many numbers!

Region Proposals

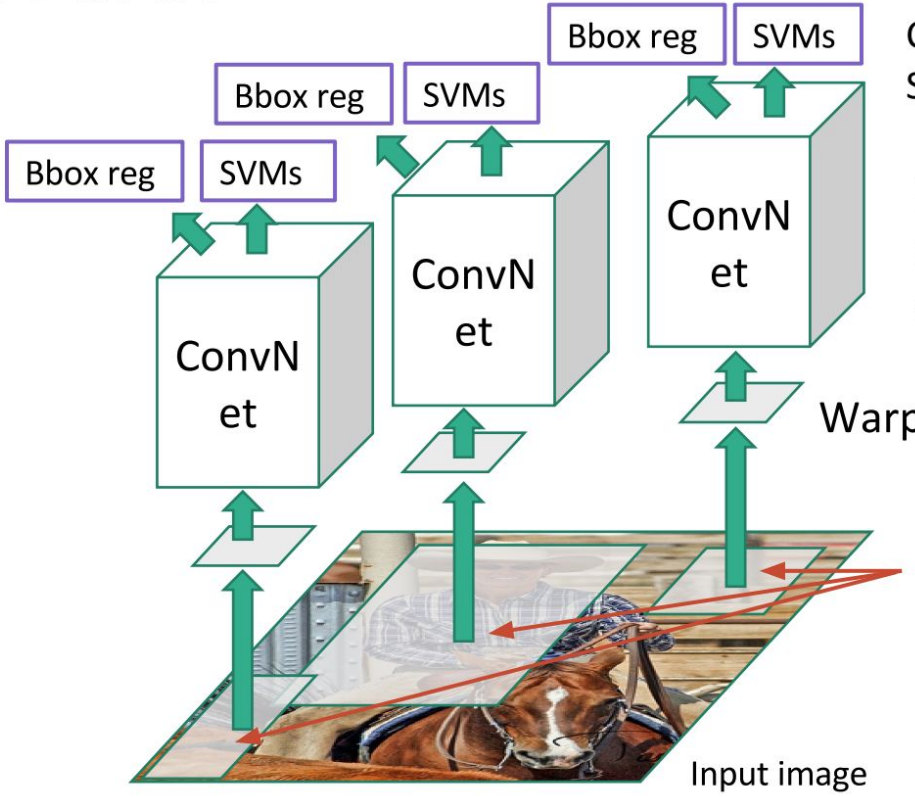
- Sliding window no es viable
- Localization + classification requiere el número de keypoints fijo
- Técnicas de segmentación para proponer regiones
- ROI (Region of Interest)
- Encuentra ‘blobs’

Selective Search



- No es una técnica de ML sino tradicional basada en el histograma de color
- Técnica que arroja 2000 regiones aprox en el orden del par de segundos con CPU
- Esta técnica tiene un alto recall: La mayoría son falsos positivos pero si hay un objeto en la imagen, va a haber una región que lo contenga.
- Una vez obtenidas las regiones aplico la CNN para clasificación

R-CNN



Linear Regression for bounding box offsets

Classify regions with SVMs

Forward each region through ConvNet

Warped image regions

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

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

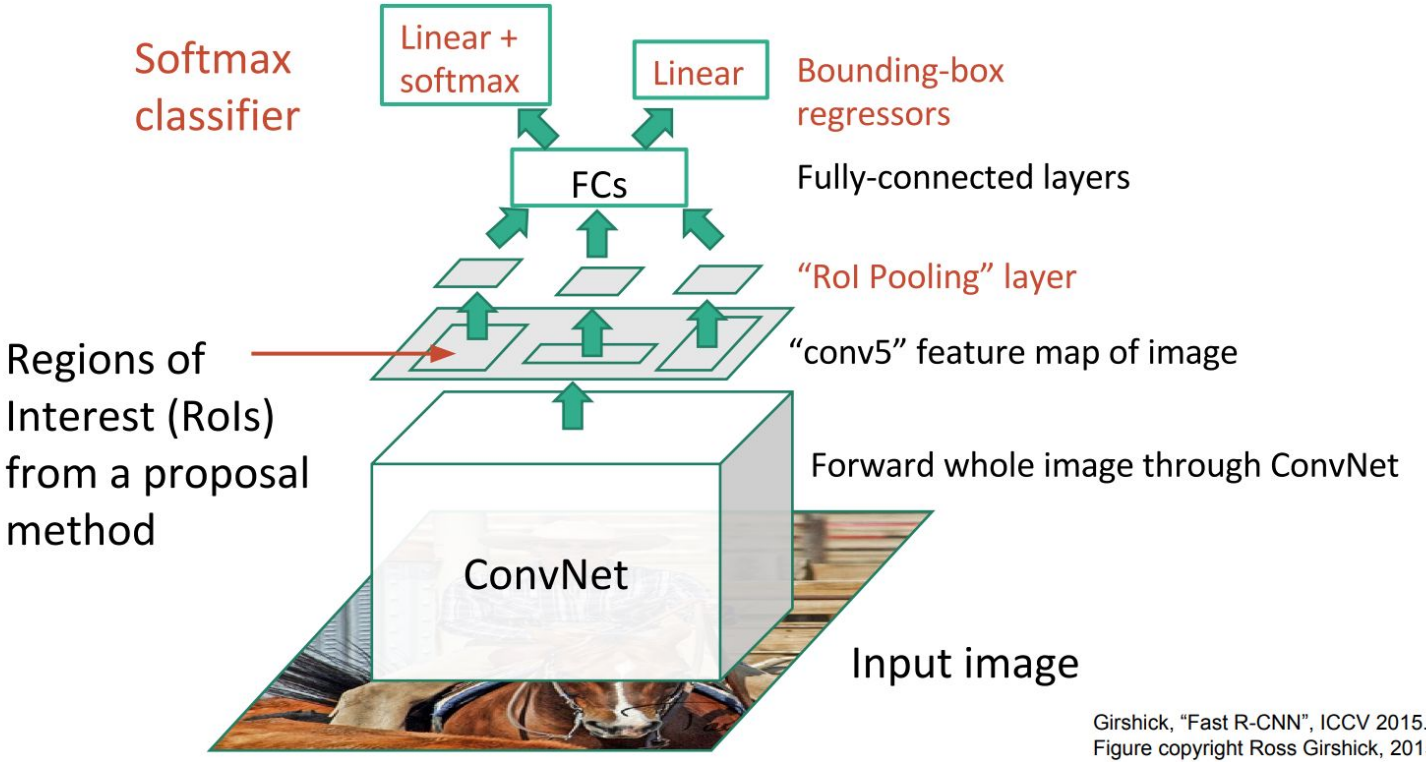
R-CNN

- Encuentra regiones con técnicas tradicionales (No deep learning)
- Las pasa por una CNN
- Obtiene la clasificación y la bounding box para corregir la región
- Es más eficiente que hacer sliding window
- Warping de las imágenes debido a que entran a una CNN con una capa densa al final
- Es muy lenta, más de 2 segundos debido al selective search principalmente
- Hay que agregar la clase NO_OBJECT

Salida de la CNN:

- Cantidad de clases (One hot enconding) + X + Y + W + H

Fast R-CNN



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

https://arxiv.org/pdf/1504.08083.pdf

Fast R-CNN

- Aplica la CNN una sola vez
- Realiza la búsqueda de los blobs en los features de salida de la CNN
- Hay que hacer reshape (ROI pooling) ya que hay una FC a la salida
- Se aplica de manera independiente a cada canal
- Aplica la FC muchas veces en vez de la CNN

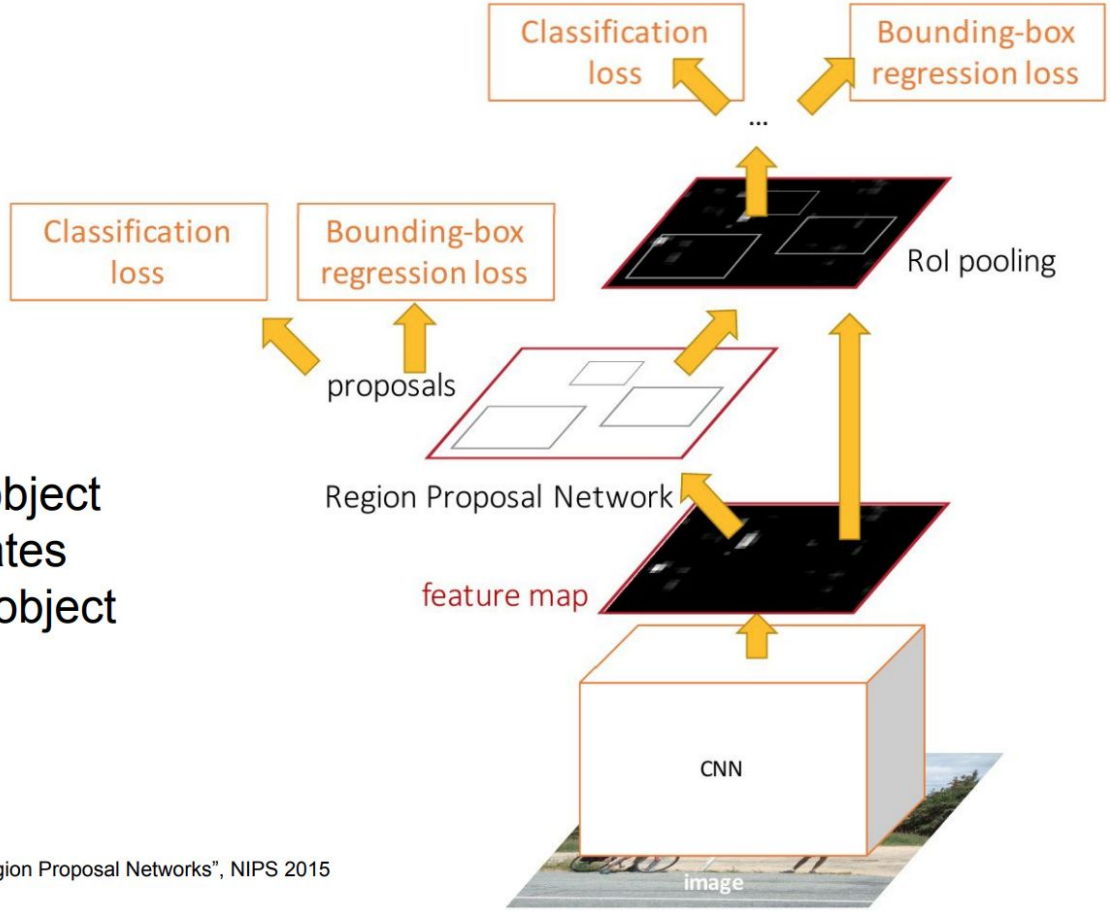
Faster R-CNN:

Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

Jointly train with 4 losses:

1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates

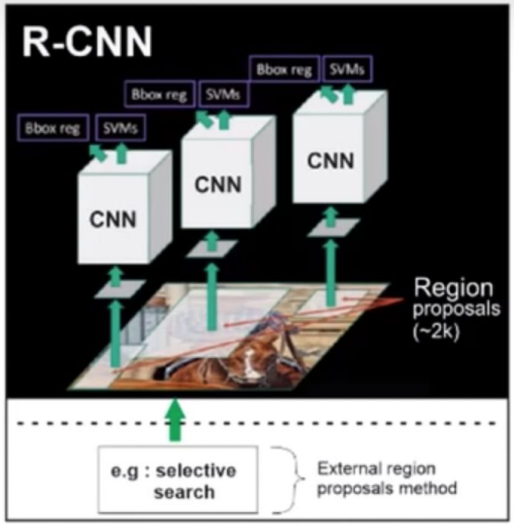


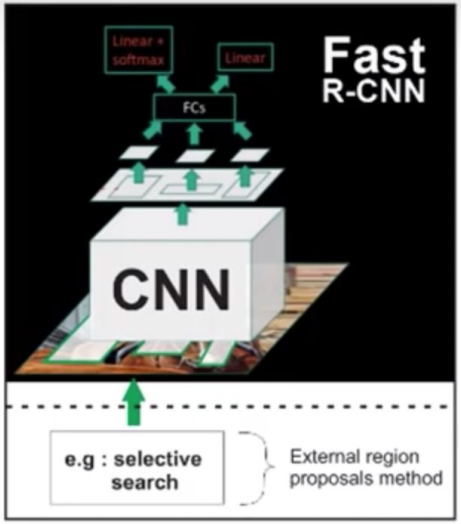
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

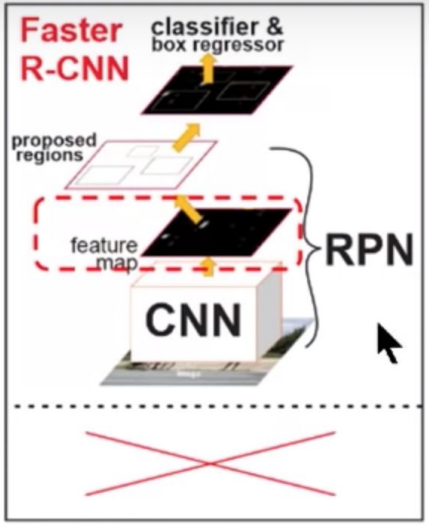
Faster R-CNN

La CNN da los regiones propuestas con una RPN (Region Proposal Network)

- RPN clasifica objeto/no objeto
- RPN propone bounding box de region
- Resto de la red propone la clase final y la bounding box
- Todo se suele entrenar en conjunto
- https://github.com/RockyXu66/Faster_RCNN_for_Open_Images_Dataset_Keras







	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x
mAP (VOC 2007)	66.0%	66.9%	66.9%

* Standford lecture notes on CNN by Fei Fei Li and Andrej Karpathy

Object Detectión
YOLO / SSD

YOLO: https://arxiv.org/pdf/1506.02640.pdf
YOLOv2 (YOLO9000): https://arxiv.org/abs/1612.08242
YOLOv3: https://arxiv.org/pdf/1804.02767.pdf

YOLO/SSD

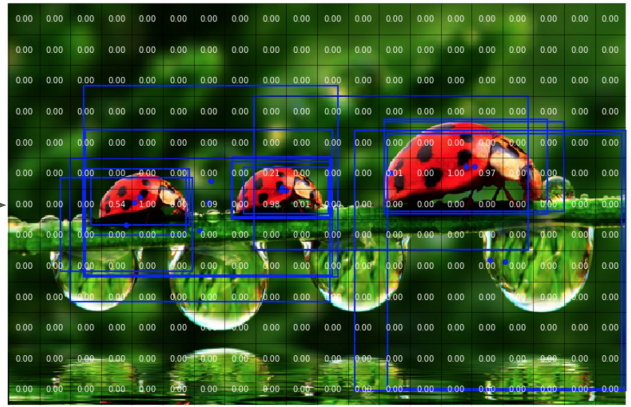
- YOLO (You Only Look Once)
- SSD (Single Shot Detection)

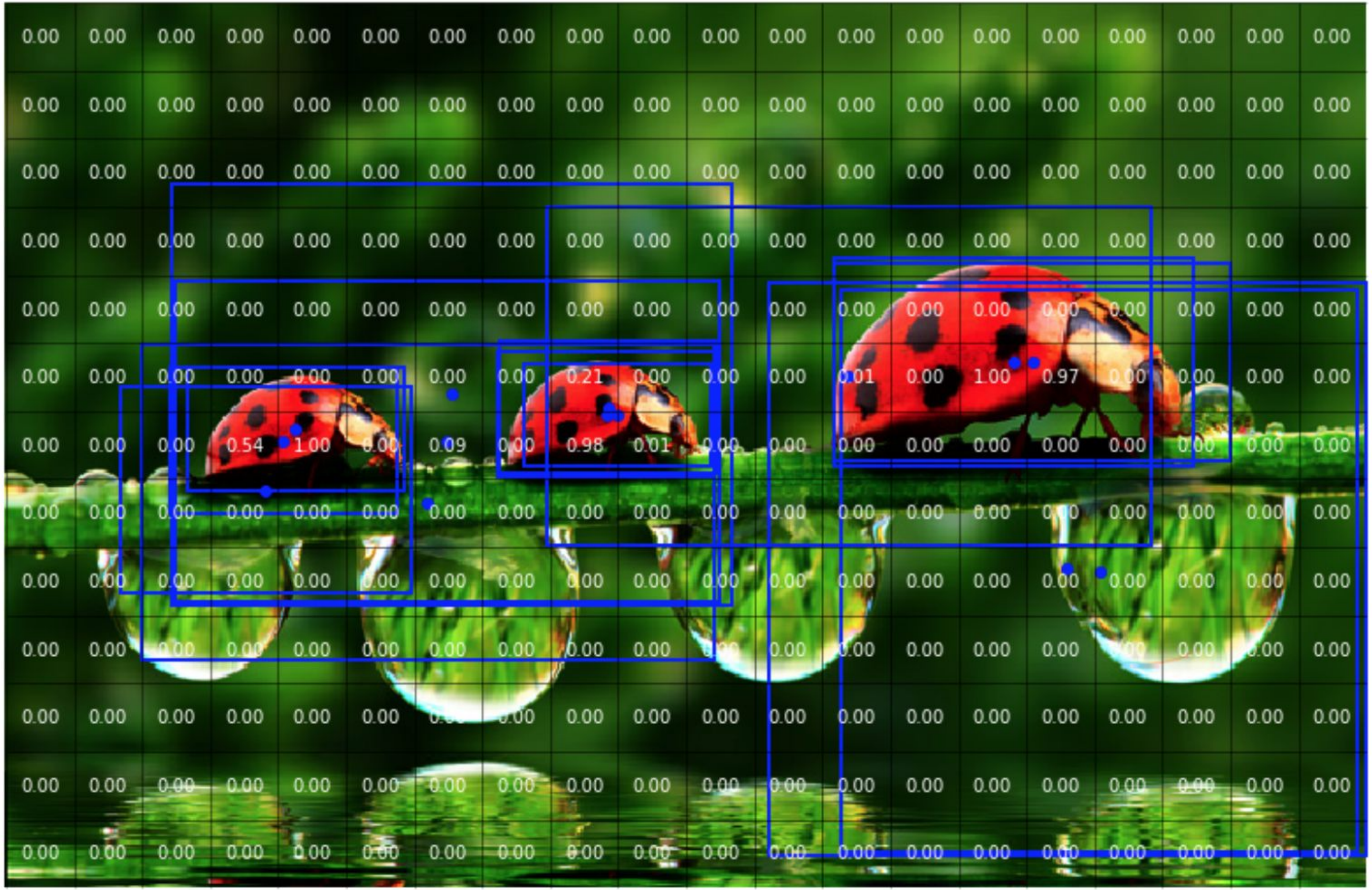
Características:

- Detectan múltiples objetos y su localización sin necesidad de saber a priori la cantidad.
- Hay un máximo pero suele ser muy superior a la cantidad de objetos que puede haber en una imágen
- All conv network (Se puede ingresar con distintos tamaños de imágenes
- Tiempo real (Video)
- Muy eficiente para problemas como Self-Driven Cars

Ejemplo sin anchor boxes (Link a Colab)

CNN (All convs)





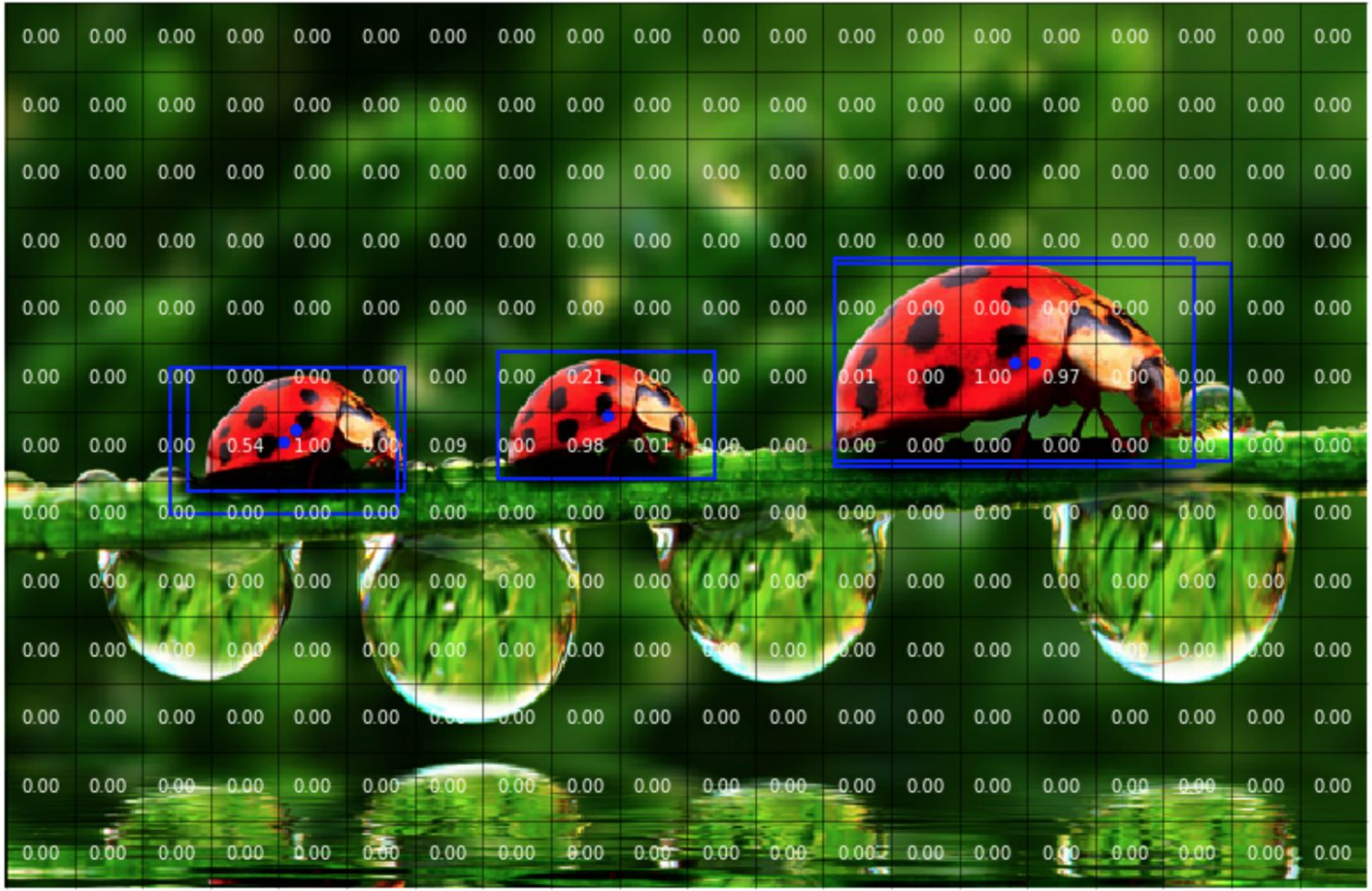
Po = Obj_conf > 0.001

Salida de la CNN

- Grid size: 32x32
- Image size: 640x416
- Cantidad de predicciones: 20x13
- Salida de la CNN: (Batch size, Alto en grids, Ancho en grids, Boxes, Predicciones)
- (1, 13, 20, 1, 13)

Predicciones:

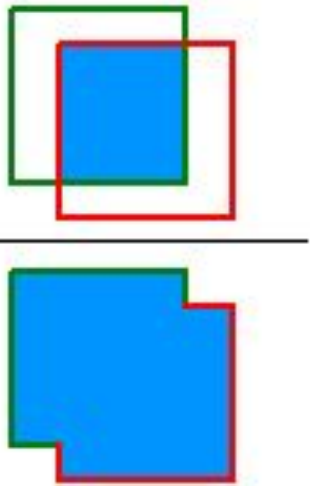
N_clases + Object_confidence + X + Y + W + H = 13 (para 8 clases)



Po = Obj_conf > 0.5

IoU (Intersection over Union) - Jaccard index

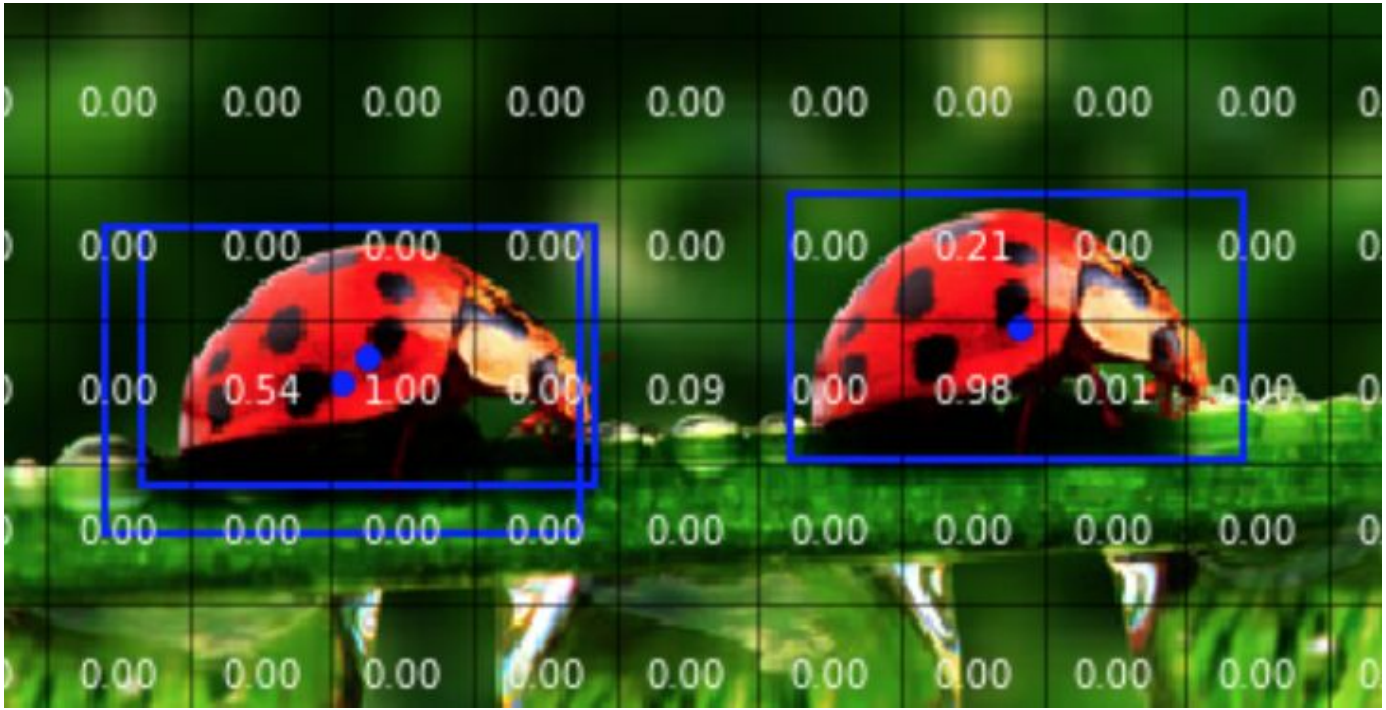
$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$



Puede variar entre 0 y 1

Non-max suppression

- X e Y pueden ser mayores que 1.



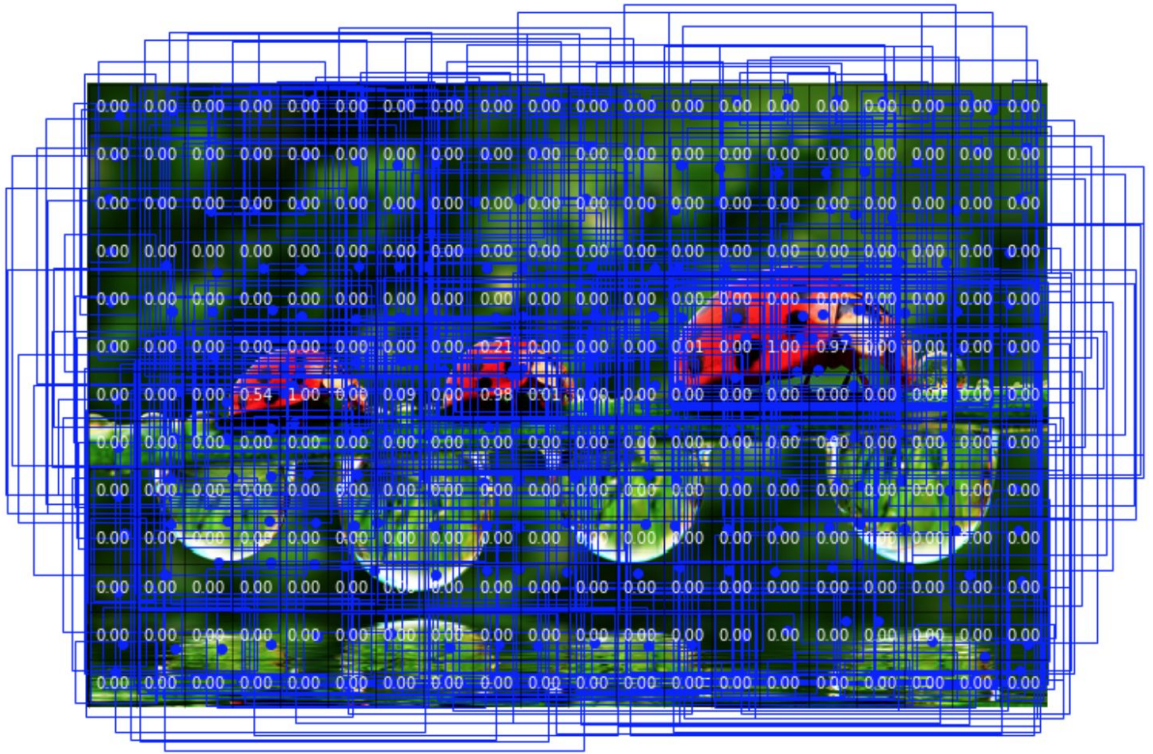
Ya descartamos todas las BB con Po<0.5

Tomamos la de mayor probabilidad (Po x P_class)

Buscamos todas las predicciones con IoU mayor a un umbral y las descartamos

Se realiza independientemente por cada clase

Salida de la red



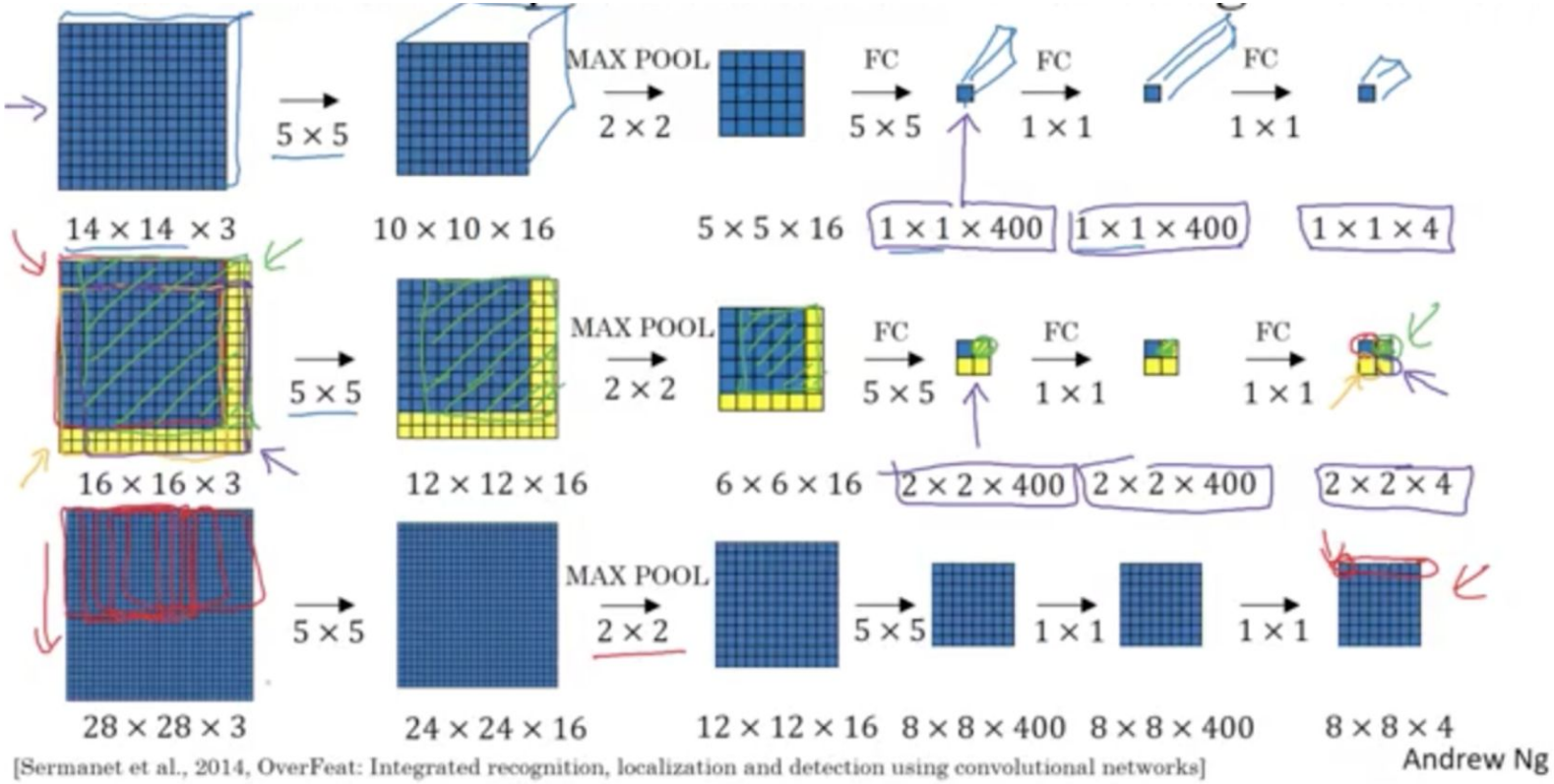
Po > 0.5

Luego de Non-Max suppression

Sliding window

- Volvemos a la idea de sliding window anterior
- Que hace una capa convolucional?
- Librerías muy eficientes que realizan la convolución

Detalles de la CNN (All conv)



[Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection using convolutional networks]

Andrew Ng

Anchor boxes

- En vez de predecir la BB desde cero se pueden definir un conjunto de posibles BB y luego estimar el error
- Al momento de entrenamiento hay que buscar cual es la que tiene mayor IoU. Estrategia: Si mayor IoU>0.5 usarla, si entre 0.4 y 0.5 ambiguo, menor a 0.4 NO_OBJECT
- Determinación de Anchor boxes:
  - Cuales son el mínimo y máximo tamaño que quiero detectar?
  - K-means
- Implementaciones:
  - YOLOv2: 5 Anchor Boxes
  - YOLOv3: 9 Anchor Boxes

Función de costo en YOLO

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

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

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

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

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

Detalles de la Implementación

https://github.com/deeplearning-itba/Object-Detection

Repositorio y dataset

- Repositorio: aca
- Bajar el dataset de aca: Dataset
- Descomprimirlo dataset
- Cambiar el path a la ubicación donde quedó el dataset

reduced_dataset_folder =
'/home/usuario/repos/Object_Localization/challenge_dataset_v3/'

Detalle de notebooks

- 00-Intro.ipynb: Links a blogs interesantes entre otras cosas
- **01-Classification-with-generator**
- **02-Localizacion**
- 03-Scale-problem
- **04-Localization_with_confidence**
- 05-Anchor-boxes
- 06-Sliding-window
- 07-YOLO-details
- **08-YOLO-Basic-Implement**
- 09-training_methods
- 10-yolo-helper-tests

Solo Clasificación

Classification-with-generator

DataGenerator de Keras y Data Augmentation

```
datagen_train = ImageDataGenerator(rescale=1./255, horizontal_flip=True, vertical_flip=True)

train_generator = datagen_train.flow_from_directory(
    classes = classes,
    directory=train_folder,
    target_size=target_size,
    color_mode="rgb",
    batch_size=batch_size,
    class_mode="categorical",
    shuffle=True,
    seed=42
)
```

```
Found 2401 images belonging to 8 classes.
```

```
datagen_val = ImageDataGenerator(rescale=1./255, horizontal_flip=True, vertical_flip=True)
val_generator = datagen_val.flow_from_directory(
    classes = classes,
    directory=val_folder,
    target_size=target_size,
    color_mode="rgb",
    batch_size=batch_size,
    class_mode="categorical",
    shuffle=True,
    seed=42
)
```

```
Found 480 images belonging to 8 classes.
```

```
next(val_generator)[0].shape
```

```
(64, 320, 320, 3)
```

```
next(val_generator)[1].shape
```

```
(64, 8)
```

```
next(val_generator)[1][:10]
```

```
array([[0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.]],
```

Transfer Learning (Ejemplo VGG-16)

```
N_trainable = 17
modelVGG16 = VGG16(include_top=False, weights='imagenet', input_shape=(*target_size, 3))
flatten_output = Flatten()(modelVGG16.output)
dense_1_output = Activation('relu')(Dense(128, name = 'Dense_1')(flatten_output))
dense_2_output = Activation('relu')(Dense(128, name = 'Dense_2')(dense_1_output))

classification = Dense(n_classes, activation='softmax', name='category_output')(dense_2_output)
```

```
model = Model(inputs=modelVGG16.input, outputs=classification)
```

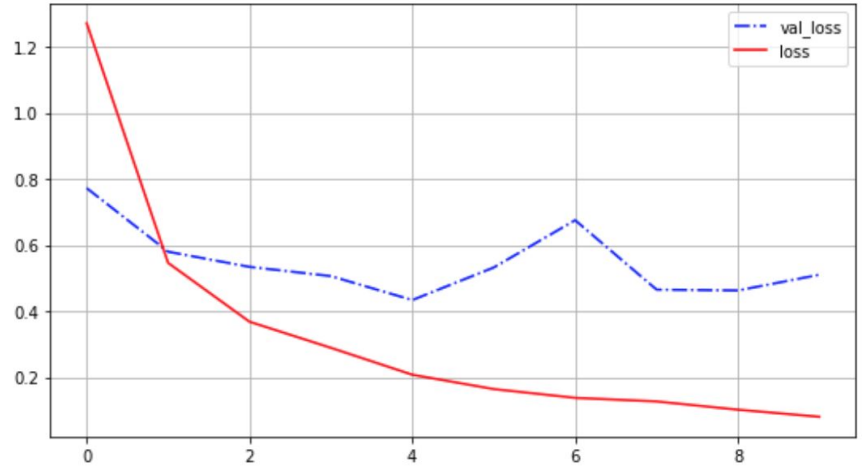
¿Cuantas capas son entrenables?

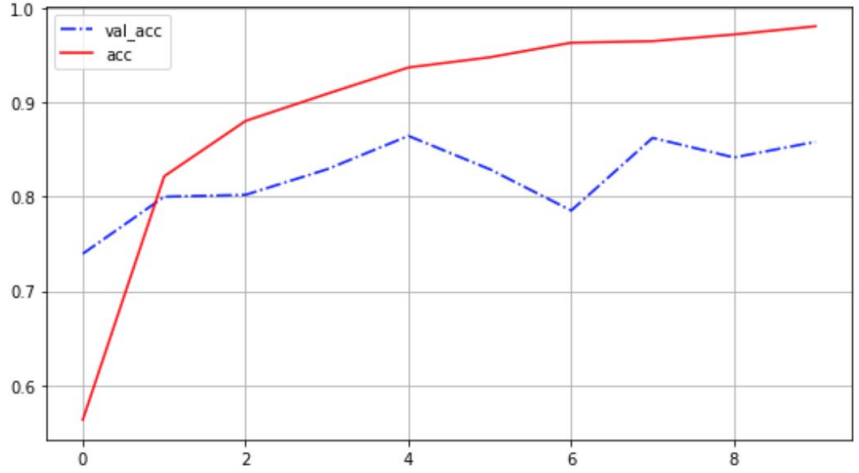
```
for layer in model.layers[N_trainable:]:
    layer.trainable = True
for layer in model.layers[:N_trainable]:
    layer.trainable = False
```

```
model.compile(loss=["categorical_crossentropy"], optimizer=Adam(lr=0.0001), metrics=["accuracy"])
```

Entrenamiento del modelo (fit_generator)

```
plot_losses = PlotLosses(1)
model.fit_generator(
        train_generator,
        steps_per_epoch=steps_per_epoch,
        epochs=10,
        validation_steps=len(val_generator),
        validation_data=val_generator,
        callbacks = [plot_losses, checkpoint]
        )
```





Localization + Classification

Keypoints regression

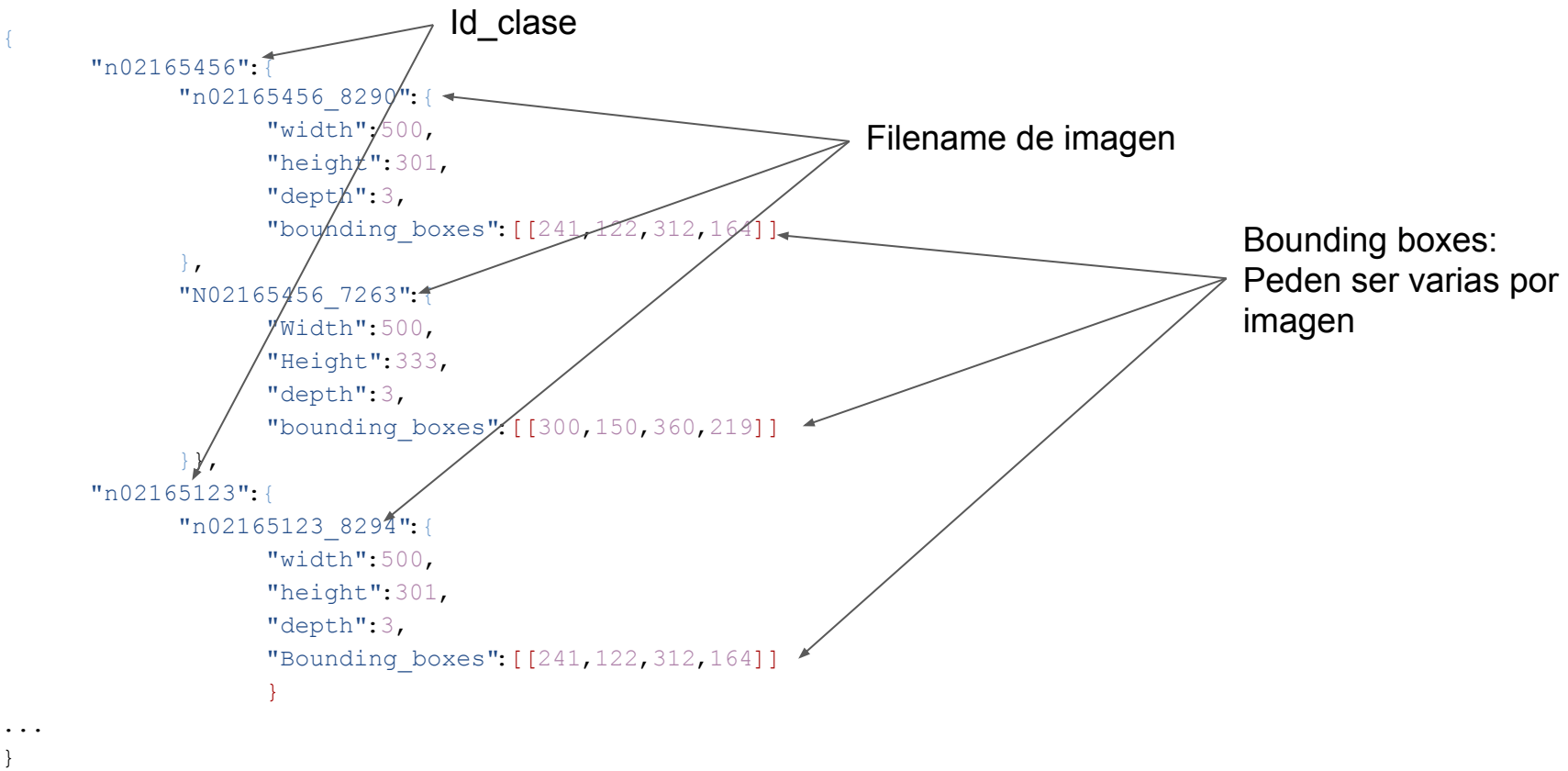
Datasets para Object detection

- COCO
- PASCAL VOC
- Open Image

Poseen distintos formatos en las anotaciones de las bounding boxes y hay que adaptarlas.

Algunos poseen una API para filtrar

Ejemplos


Id_clase
Filename de imagen
Bounding boxes:
Peden ser varias por
imagen
{
"n02165456":{
"n02165456_8290":{
"width":500,
"height":301,
"depth":3,
"bounding_boxes":[[241,122,312,164]]
},
"N02165456_7263":{
"Width":500,
"Height":333,
"depth":3,
"bounding_boxes":[[300,150,360,219]]
}},
"n02165123":{
"n02165123_8294":{
"width":500,
"height":301,
"depth":3,
"Bounding_boxes":[[241,122,312,164]]
}
...
}

Custom Data Generator

Keras ImageDataGenerator

```
next(val_generator)[0].shape
```

```
(64, 320, 320, 3)
```

```
next(val_generator)[1].shape
```

```
(64, 8)
```

```
next(val_generator)[1][:10]
```

```
array([[0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.]],
```

Custom DataGenerator

```
# [[images], [one_hot], [bboxes]]]
next(train_generator_multiple_outputs)[0].shape
```

```
(10, 320, 320, 3)
```

```
next(train_generator_multiple_outputs)[1][0]
```

```
array([[0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

```
next(train_generator_multiple_outputs)[1][1]
```

```
array([[0.286     , 0.64307229, 0.572     , 0.27409639],
       [0.533     , 0.54266667, 0.898     , 0.744     ],
       [0.464     , 0.58533333, 0.6       , 0.63733333],
       [0.527     , 0.43993994, 0.514     , 0.5975976 ],
       [0.477     , 0.51951952, 0.466     , 0.48048048],
       [0.493     , 0.48514851, 0.142     , 0.14851485],
       [0.511     , 0.50983146, 0.442     , 0.63202247],
       [0.364     , 0.48913043, 0.724     , 0.68478261],
       [0.502     , 0.42723005, 0.388     , 0.57276995],
       [0.572     , 0.43218085, 0.852     , 0.85904255]])
```

```python
class GeneratorMultipleOutputs(Sequence):
    def __init__(self, annotations_dict, folder, batch_size, flip = 'no_flip', concat_output=True, target_size=(375, 500), classes=None):
        self.concat_output = concat_output, self.flip = flip, self.annotations_dict = annotations_dict
        np.random.seed(seed=40)
        datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=False)
        self.generator = datagen.flow_from_directory(
            classes = classes, directory=folder, target_size=target_size, color_mode="rgb", batch_size=batch_size,
            class_mode="categorical", shuffle=True, seed=42
        )

    def get_image_object_center(self):
        # Obtengo bounding boxes de un batch,  Calculo centros y anchos luego de augmentation,  Y las devuelvo
        return centerXs, centerYs, box_widths, box_heights

    def __len__(self):
        return int(np.ceil(self.generator.samples / float(self.generator.batch_size)))

    def __getitem__(self, idx):
        data = next(self.generator)
        centerX, centerY, width, height = self.get_image_object_center()
        # object_detected_arr es un array que tiene un 0 si la imagen pertenece a la clase word
        # (El calculo no esta aca para hacer el código más sencillo)

        if self.concat_output:
            if self.has_world:
                output = np.hstack([classes_array, np.array([centerX, centerY, width, height]).T, object_detected_arr.T])
            else:
                output = np.hstack([classes_array, np.array([centerX, centerY, width, height]).T])
        else:
            if self.has_world:
                output = [classes_array, np.array([centerX, centerY, width, height]).T, object_detected_arr.T]
            else:
                output = [classes_array, np.array([centerX, centerY, width, height]).T]
        return (data[0], output)

    def __next__(self):
        return self.__getitem__(0)

    def __iter__(self):
        return self
```

Modelo

```
n_classes = len(classes)
flatten_output = Flatten()(modelVGG16.output)
dense_1_output = Activation('relu')(Dense(128, name = 'Dense_1')(flatten_output))
dense_2_output = Activation('relu')(Dense(128, name = 'Dense_2')(dense_1_output))


class_prediction = Dense(n_classes, activation='softmax', name='category_output')(dense_2_output)
bbox_prediction = Dense(4, name='bounding_box')(Dropout(0.5)(dense_2_output))

model = Model(inputs=modelVGG16.input, outputs=[class_prediction, bbox_prediction])
```

Multitask Loss y métricas

$$L = \beta L_{categorical} + \gamma L_{boundingbox}$$

```
losses = {"category_output": "categorical_crossentropy",
          "bounding_box": "mean_squared_error"
          #"bounding_box": IOU_loss
         }
metrics = {"category_output": "accuracy",
           "bounding_box": [iou, IOU_loss]
          }
lossWeights = {"category_output": 1.0,
               "bounding_box":  20.0,
              }
```

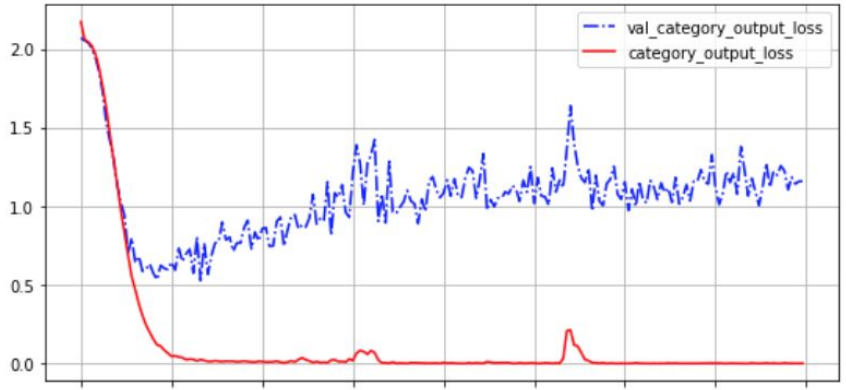
```
model.compile(loss=losses,
              optimizer=Adam(lr=0.001),
              metrics=metrics,
              loss_weights = lossWeights)
```

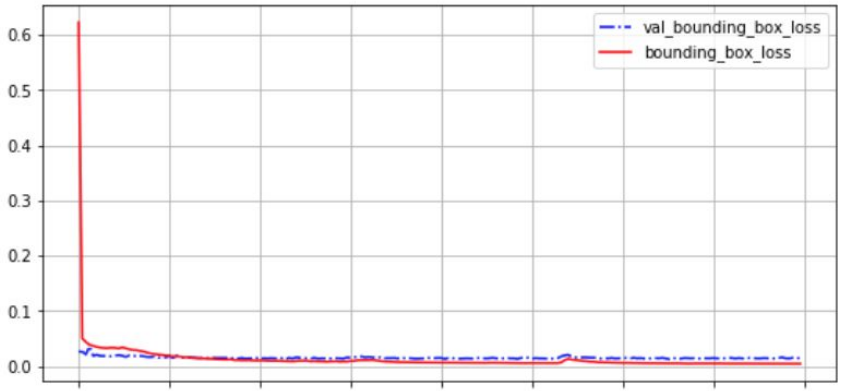
IoU Loss

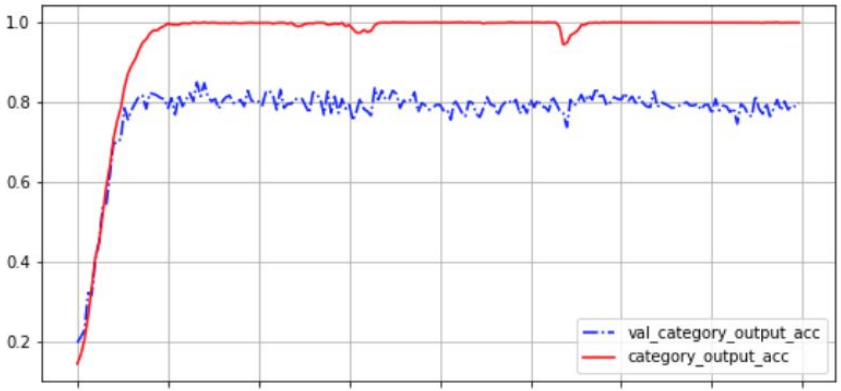
```
def iou(boxA, boxB):
    xA = K.stack([boxA[:,0]-boxA[:,2]/2, boxB[:,0]-boxB[:,2]/2], axis=-1)
    yA = K.stack([boxA[:,1]-boxA[:,3]/2, boxB[:,1]-boxB[:,3]/2], axis=-1)
    xB = K.stack([boxA[:,0]+boxA[:,2]/2, boxB[:,0]+boxB[:,2]/2], axis=-1)
    yB = K.stack([boxA[:,1]+boxA[:,3]/2, boxB[:,1]+boxB[:,3]/2], axis=-1)

    xA = K.max(xA, axis=-1)
    yA = K.max(yA, axis=-1)
    xB = K.min(xB, axis=-1)
    yB = K.min(yB, axis=-1)

    interX = K.zeros_like(xB)
    interY = K.zeros_like(yB)

    interX = K.stack([interX, xB-xA], axis=-1)
    interY = K.stack([interY, yB-yA], axis=-1)

    #because of these "max", interArea may be constant 0, without gradients, and you may have problems with no gradients.
    interX = K.max(interX, axis=-1)
    interY = K.max(interY, axis=-1)
    interArea = interX * interY

    boxAArea = (boxA[:,2]) * (boxA[:,3])
    boxBArea = (boxB[:,2]) * (boxB[:,3])
    iou = interArea / (boxAArea + boxBArea - interArea)
    return iou

def IOU_loss(boxA,boxB):
    iou_ = iou(boxA,boxB)
    return K.ones_like(iou_)-iou_
```

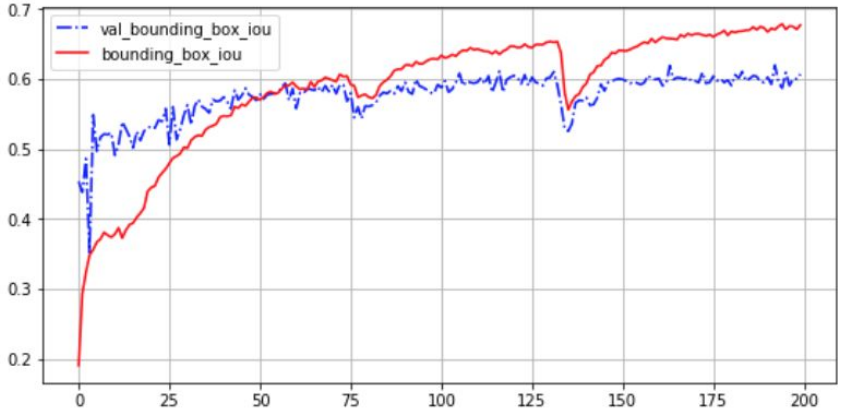
Entrenamiento

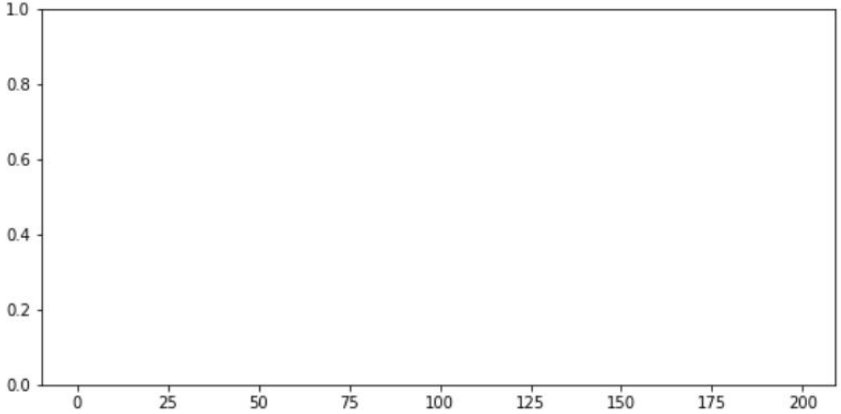
```
plot_losses = PlotLosses(1)
model.fit_generator(
        train_generator_multiple_outputs,
        #steps_per_epoch=steps_per_epoch,
        epochs=200,
        validation_data=val_generator_multiple_outputs, callbacks = [plot_losses, checkpoint]
        )
```




val_bounding_box_loss
bounding_box_loss
0.6
0.5
0.4
0.3
0.2
0.1
0.0


val_category_output_acc
category_output_acc
1.0
0.8
0.6
0.4
0.2


val_bounding_box_iou
bounding_box_iou
0.7
0.6
0.5
0.4
0.3
0.2
0
25
50
75
100
125
150
175
200


1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
200

Localization + Classification + world

Agregado de mundo

- Detección de objetos
- Cada grid detecta tantos objetos como anchor boxes tenga asociado
- La mayoría de las veces la detección es negativa
- Se agrega entonces la probabilidad de detección de un objeto (Confidence)

Custom Data Generator

Custom DataGenerator no confidence

```
# [[images], [[one_hot], [bboxes]]]
next(train_generator_multiple_outputs)[0].shape
```

```
(10, 320, 320, 3)
```

```
next(train_generator_multiple_outputs)[1][0]
```

```
array([[0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

```
next(train_generator_multiple_outputs)[1][1]
```

```
array([[0.286     , 0.64307229, 0.572     , 0.27409639],
       [0.533     , 0.54266667, 0.898     , 0.744     ],
       [0.464     , 0.58533333, 0.6       , 0.63733333],
       [0.527     , 0.43993994, 0.514     , 0.5975976 ],
       [0.477     , 0.51951952, 0.466     , 0.48048048],
       [0.493     , 0.48514851, 0.142     , 0.14851485],
       [0.511     , 0.50983146, 0.442     , 0.63202247],
       [0.364     , 0.48913043, 0.724     , 0.68478261],
       [0.502     , 0.42723005, 0.388     , 0.57276995],
       [0.572     , 0.43218085, 0.852     , 0.85904255]])
```

Custom DataGenerator with confidence

```
next(train_generator_multiple_outputs)[0].shape
```

```
(4, 320, 320, 3)
```

```
print(next(train_generator_multiple_outputs)[1])
```

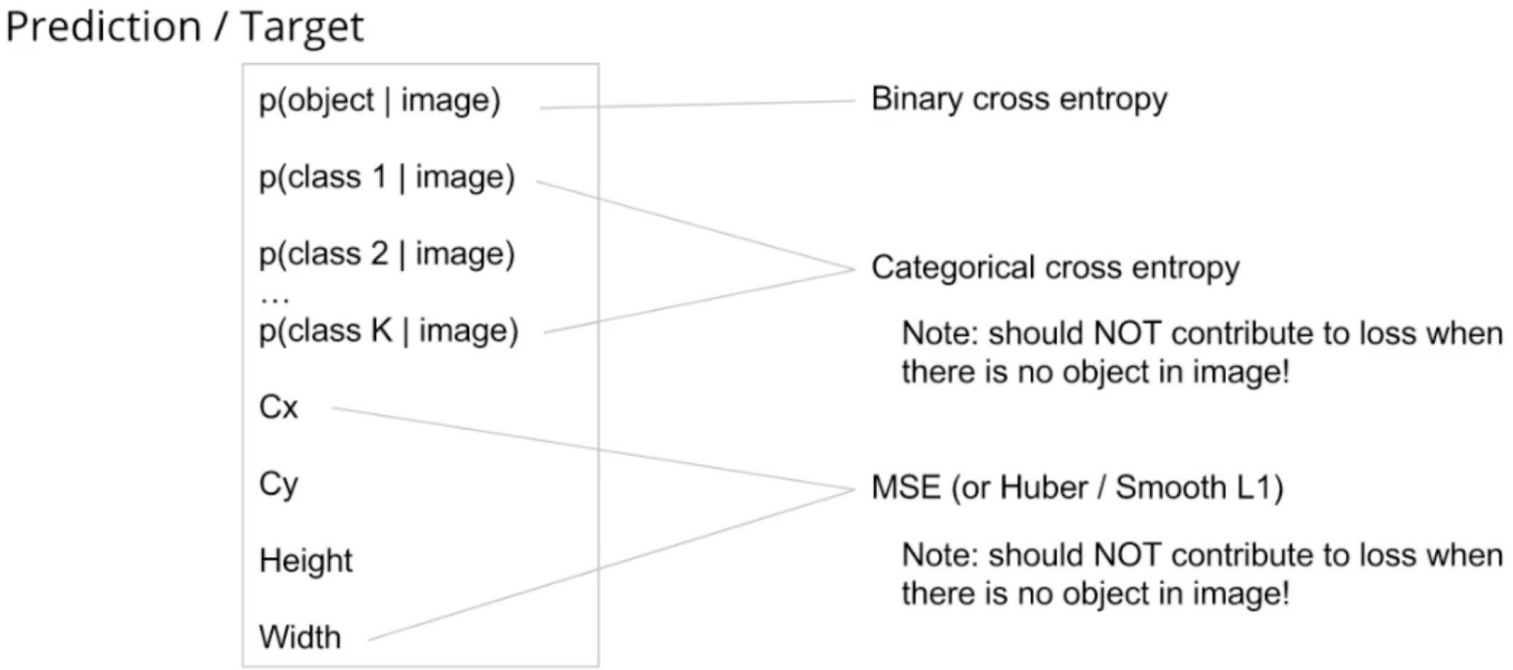
```
[[ 0.          0.          0.          0.          1.          0.
   0.          0.          0.65        0.47443182  0.564       0.60227273
   1.        ]
 [ 0.          1.          0.          0.          0.          0.
   0.          0.          0.355       0.51951952  0.706       0.95495495
   1.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.         -0.         -0.         -0.         -0.
   0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.         -0.         -0.         -0.         -0.
   0.        ]]
```

```python
class GeneratorMultipleOutputs(Sequence):
    def __init__(self, annotations_dict, folder, batch_size, flip = 'no_flip', concat_output=True, target_size=(375, 500), classes=None):
        self.concat_output = concat_output, self.flip = flip, self.annotations_dict = annotations_dict
        np.random.seed(seed=40)
        datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=False)
        self.generator = datagen.flow_from_directory(
            classes = classes, directory=folder, target_size=target_size, color_mode="rgb", batch_size=batch_size,
            class_mode="categorical", shuffle=True, seed=42
        )

    def get_image_object_center(self):
        # Obtengo bounding boxes de un batch,  Calculo centros y anchos luego de augmentation,  Y las devuelvo
        return centerXs, centerYs, box_widths, box_heights

    def __len__(self):
        return int(np.ceil(self.generator.samples / float(self.generator.batch_size)))

    def __getitem__(self, idx):
        data = next(self.generator)
        centerX, centerY, width, height = self.get_image_object_center()
        # object_detected_arr es un array que tiene un 0 si la imagen pertenece a la clase word
        # (El calculo no esta aca para hacer el código más sencillo)

        if self.concat_output:
            if self.has_world:
                output = np.hstack([classes_array, np.array([centerX, centerY, width, height]).T, object_detected_arr.T])
            else:
                output = np.hstack([classes_array, np.array([centerX, centerY, width, height]).T])
        else:
            if self.has_world:
                output = [classes_array, np.array([centerX, centerY, width, height]).T, object_detected_arr.T]
            else:
                output = [classes_array, np.array([centerX, centerY, width, height]).T]
        return (data[0], output)

    def __next__(self):
        return self.__getitem__(0)

    def __iter__(self):
        return self
```

Modelo

```
modelVGG16 = VGG16(include_top=False, weights='imagenet', input_shape=(*target_size, 3))

flatten_output = Flatten()(modelVGG16.output)
dense_1_output = Activation('relu')(Dense(128, name = 'Dense_1')(flatten_output))
dense_2_output = Activation('relu')(Dense(128, name = 'Dense_2')(dense_1_output))

classification = Dense(n_classes, activation='softmax', name='category_output')(dense_2_output)
bounding_box = Dense(4, activation=None, name='bounding_box')(Dropout(0.25)(dense_2_output))
confidence = Dense(1, activation='sigmoid', name='obj_confidence')(dense_2_output)

all_outs = Concatenate(name='concatenated_outputs')([classification, bounding_box, confidence])
model = Model(inputs=modelVGG16.input, outputs=[all_outs])
```

Multitask loss



$$L = \alpha L_{binary} + \beta L_{categorical} + \gamma L_{boundingbox}$$

Multitask loss en keras

```
#losses = {"concatenated_outputs": mse_custom_loss}
losses = {"concatenated_outputs": yolo_loss}
#losses = {"concatenated_outputs": custom_loss}
#losses = {"concatenated_outputs": custom_loss_IOU}

metrics = {"concatenated_outputs": [classes_acc,
                                    confidence_acc_with_sigmoid,
                                    #confidence_acc,
                                    bounding_box_mse,
                                    iou_v2,
                                    cat_cross_entropy_loss,
                                    bin_cross_entropy_loss
                                   ]}
```

```
model.compile(loss=losses,
              optimizer=SGD(lr=1e-8),
              metrics=metrics)
```

Yolo Loss

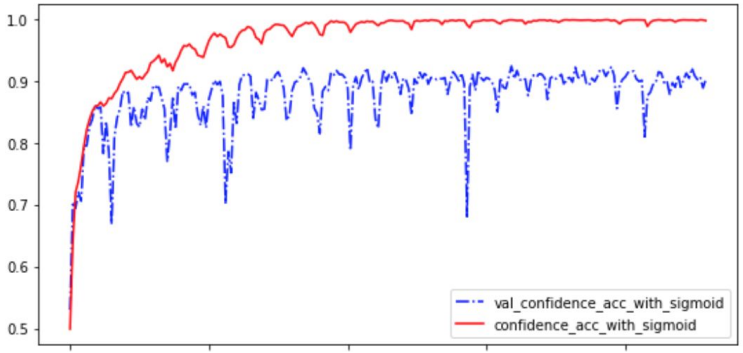
```
def yolo_loss(y_true, y_pred):
    # Indices donde hay objeto
    indexes = tf.where(K.equal(y_true[:,:,:,:,0], K.ones_like(y_true[:,:,:,:,0])))
    # Indices donde no hay objetos
    indexes_neg = tf.where(K.equal(y_true[:,:,:,:,0], K.zeros_like(y_true[:,:,:,:,0])))
    y_true_pos = tf.gather_nd(y_true, indexes) # Unos que corresponden a ground truth con objetos
    y_pred_pos = tf.gather_nd(y_pred, indexes) # Predicción para ground truth con objetos
    y_true_neg = tf.gather_nd(y_true, indexes_neg) # Ceros que corresponden a ground truth sin objetos
    y_pred_neg = tf.gather_nd(y_pred, indexes_neg) # Predicción para ground truth sin objetos
    # Cálculo de entropía categórica - solo suma cuando hay objeto
    classes_cross_entropy = K.categorical_crossentropy(y_true_pos[:,1:1+n_classes], K.softmax(y_pred_pos[:,1:1+n_classes]))
    # Cálculo de bounding box - solo suma cuando hay objeto
    bounding_box_mse = K.mean(K.square(y_pred_pos[:,1+n_classes:1+n_classes+4] - y_true_pos[:,1+n_classes:1+n_classes+4]), axis=-1)
    # Cálculo de entropía binaria para cuando hay objeto
    confidence_cross_entropy_pos = K.mean(K.binary_crossentropy(y_true_pos[:,:1], K.sigmoid(y_pred_pos[:,:1])), axis=-1)
    # Cálculo de entropía binaria para cuando NO hay objeto (Estan separadas para poder pesarlas distinto)
    confidence_cross_entropy_neg = 0.01*K.mean(K.binary_crossentropy(y_true_neg[:,:1], K.sigmoid(y_pred_neg[:,:1])), axis=-1)
    # Devuelvo la suma de todas las losses
    return k_classification*K.mean(classes_cross_entropy) + k_bounding_boxes*K.mean(bounding_box_mse) +
k_confidence*K.mean(confidence_cross_entropy_pos) + K.mean(confidence_cross_entropy_neg)
```

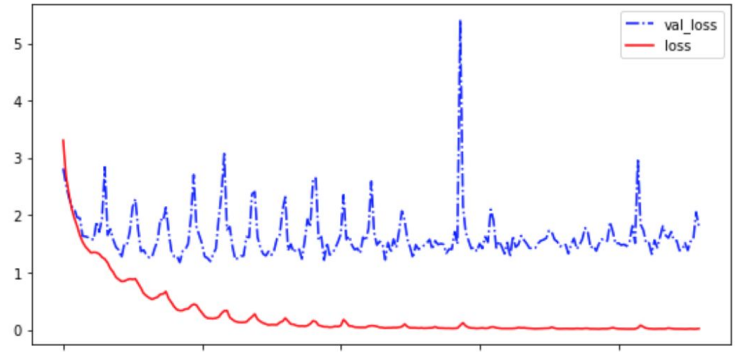
MSE Custom loss

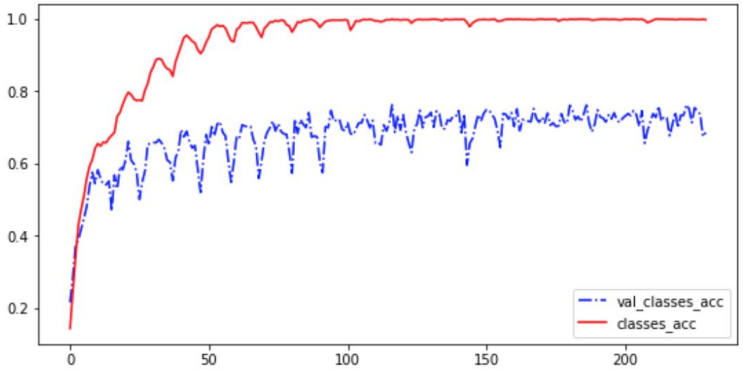
Aplica MSE a todas las salidas

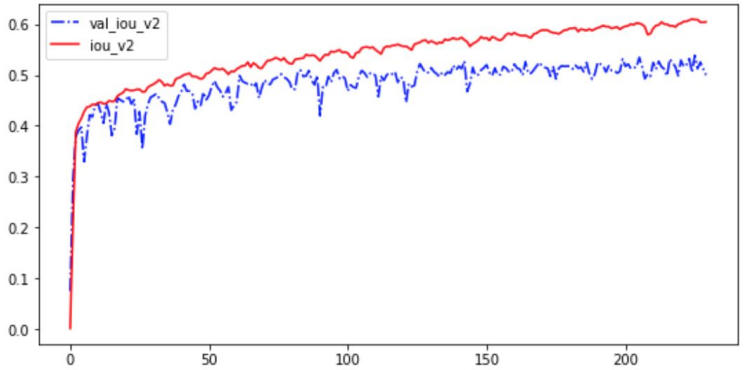
```
def mse_custom_loss(y_true, y_pred):
    indexes_pos = tf.where(K.equal(y_true[:,-1], K.ones_like(y_true[:,-1])))[:,0]
    indexes_neg = tf.where(K.equal(y_true[:,-1], K.zeros_like(y_true[:,-1])))[:,0]
    y_true_pos = tf.gather(y_true, indexes_pos)
    y_pred_pos = tf.gather(y_pred, indexes_pos)
    y_true_neg = tf.gather(y_true, indexes_neg)
    y_pred_neg = tf.gather(y_pred, indexes_neg)
    classes_mse = K.mean(K.square(y_true_pos[:,:n_classes] - y_pred_pos[:,:n_classes]), axis=-1)
    centers_mse = K.mean(K.square(y_pred_pos[:,n_classes:n_classes+2] - y_true_pos[:,n_classes:n_classes+2]), axis=-1)
    width_height_mse = K.mean(K.square(K.sqrt(y_pred_pos[:,n_classes+2:n_classes+4]) - K.sqrt(y_true_pos[:,n_classes+2:n_classes+4])), axis=-1)
    confidence_pos_mse = K.mean(K.square(y_pred_pos[:,n_classes+4:] - y_true_pos[:,n_classes+4:]), axis=-1)
    confidence_pos_neg = K.mean(K.square(y_pred_neg[:,n_classes+4:] - y_true_neg[:,n_classes+4:]), axis=-1)
    return K.mean(classes_mse) + 5*K.mean(centers_mse) + 5*K.mean(width_height_mse) + K.mean(confidence_pos_mse) + 0.5*K.mean(confidence_pos_neg)
```

Entrenamiento









Detección de Objetos

Custom Data Generator

Custom DataGenerator with confidence

```
next(train_generator_multiple_outputs)[0].shape
```

```
(4, 320, 320, 3)
```

```
next(train_generator_multiple_outputs)[1].shape
```

```
(4, 13)
```

```
print(next(train_generator_multiple_outputs)[1])
```

```
[[ 0.          0.          0.          0.          1.          0.
   0.          0.          0.65        0.47443182  0.564       0.60227273
   1.        ]
 [ 0.          1.          0.          0.          0.          0.
   0.          0.          0.355       0.51951952  0.706       0.95495495
   1.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.         -0.         -0.         -0.         -0.
   0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.         -0.         -0.         -0.         -0.
   0.        ]]
```

Custom DataGenerator - YOLO or SSD

```
next(train_generator)[0].shape
```

```
(60, 320, 320, 3)
```

```
next(train_generator)[1].shape
```

```
(60, 10, 10, 1, 13)
```

Predicciones (8 + 4 + 1)

Anchor boxes

Grid Cells

Batch size

```
batch_pred = next(train_generator)[1]
for j in range(10):
    for i in range(10):
        print(i, j, batch_pred[0, i, j, 0, :])
```

```
3 4 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
4 4 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
5 4 [1.         0.         0.         0.         0.         0.
 0.         1.         0.         0.97663544 0.7173914  2.71028015
 5.00000023]
6 4 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Modelo YOLOv2 (1)

```
def get_YOLO_V2_NN(IMAGE_H, IMAGE_W, BOX, CLASS, GAP=False):
    input_image = Input(shape=(IMAGE_H, IMAGE_W, 3))

    # Layer 1
    x = Conv2D(32, (3,3), strides=(1,1), padding='same', name='conv_1', use_bias=False)(input_image)
    x = BatchNormalization(name='norm_1')(x)
    x = LeakyReLU(alpha=0.1)(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    # Layer 2
    x = Conv2D(64, (3,3), strides=(1,1), padding='same', name='conv_2', use_bias=False)(x)
    x = BatchNormalization(name='norm_2')(x)
    x = LeakyReLU(alpha=0.1)(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
                    ..........
                    ..........
                    ..........

   # Layer 13
   x = Conv2D(512, (3,3), strides=(1,1), padding='same', name='conv_13', use_bias=False)(x)
   x = BatchNormalization(name='norm_13')(x)
   x = LeakyReLU(alpha=0.1)(x)
   skip_connection = x

   x = MaxPooling2D(pool_size=(2, 2))(x)

   # Layer 14
   x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_14', use_bias=False)(x)
   x = BatchNormalization(name='norm_14')(x)
   x = LeakyReLU(alpha=0.1)(x)
```

Modelo YOLOv2 (2)

```
# Layer 20
x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_20', use_bias=False)(x)
x = BatchNormalization(name='norm_20')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 21
skip_connection = Conv2D(64, (1,1), strides=(1,1), padding='same', name='conv_21', use_bias=False)(skip_connection)
skip_connection = BatchNormalization(name='norm_21')(skip_connection)
skip_connection = LeakyReLU(alpha=0.1)(skip_connection)
skip_connection = Lambda(space_to_depth_x2)(skip_connection)

x = concatenate([skip_connection, x])

# Layer 22
x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_22', use_bias=False)(x)
x = BatchNormalization(name='norm_22')(x)
x = LeakyReLU(alpha=0.1)(x)

# Layer 23
x = Conv2D(BOX * (4 + 1 + CLASS), (1,1), strides=(1,1), padding='same', name='conv_23')(x)
GRID_H_ = x.shape[1]
GRID_W_ = x.shape[2]

if GAP:
    output = GlobalAveragePooling2D(name='concatenated_outputs')(x)
    if BOX>1:
        output = Reshape((BOX, 4 + 1 + CLASS))(output)
else:
    output = Reshape((GRID_H_, GRID_W_, BOX, 4 + 1 + CLASS))(x)

model = Model([input_image], output)
return model
```

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 320, 320, 3)	0	
conv_1 (Conv2D)	(None, 320, 320, 32)	864	input_2[0][0]
norm_1 (BatchNormalization)	(None, 320, 320, 32)	128	conv_1[0][0]
leaky_re_lu_23 (LeakyReLU)	(None, 320, 320, 32)	0	norm_1[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, 160, 160, 32)	0	leaky_re_lu_23[0][0]
conv_2 (Conv2D)	(None, 160, 160, 64)	18432	max_pooling2d_6[0][0]
norm_2 (BatchNormalization)	(None, 160, 160, 64)	256	conv_2[0][0]
leaky_re_lu_24 (LeakyReLU)	(None, 160, 160, 64)	0	norm_2[0][0]
max_pooling2d_7 (MaxPooling2D)	(None, 80, 80, 64)	0	leaky_re_lu_24[0][0]
conv_3 (Conv2D)	(None, 80, 80, 128)	73728	max_pooling2d_7[0][0]
norm_3 (BatchNormalization)	(None, 80, 80, 128)	512	conv_3[0][0]
leaky_re_lu_25 (LeakyReLU)	(None, 80, 80, 128)	0	norm_3[0][0]
conv_4 (Conv2D)	(None, 80, 80, 64)	8192	leaky_re_lu_25[0][0]

conv_21 (Conv2D) (None, 20, 20, 64) 32768 leaky_re_lu_35[0][0]

---

leaky_re_lu_41 (LeakyReLU) (None, 10, 10, 1024) 0 norm_19[0][0]

---

norm_21 (BatchNormalization) (None, 20, 20, 64) 256 conv_21[0][0]

---

conv_20 (Conv2D) (None, 10, 10, 1024) 9437184 leaky_re_lu_41[0][0]

---

leaky_re_lu_43 (LeakyReLU) (None, 20, 20, 64) 0 norm_21[0][0]

---

norm_20 (BatchNormalization) (None, 10, 10, 1024) 4096 conv_20[0][0]

---

lambda_2 (Lambda) (None, 10, 10, 256) 0 **leaky_re_lu_43[0][0]**

---

leaky_re_lu_42 (LeakyReLU) (None, 10, 10, 1024) 0 norm_20[0][0]

---

concatenate_2 (Concatenate) (None, 10, 10, 1280) 0 **lambda_2**[0][0], **leaky_re_lu_42**[0][0]

---

conv_22 (Conv2D) (None, 10, 10, 1024) 11796480 concatenate_2[0][0]

---

norm_22 (BatchNormalization) (None, 10, 10, 1024) 4096 conv_22[0][0]

---

leaky_re_lu_44 (LeakyReLU) (None, 10, 10, 1024) 0 norm_22[0][0]

---

conv_23 (Conv2D) (None, 10, 10, **65**) 66625 leaky_re_lu_44[0][0]

---

reshape_2 (**Reshape**) (None, 10, 10, **5, 13**) 0 conv_23[0][0]

==========

mAP

Mean Average Precision

https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173
https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2
https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

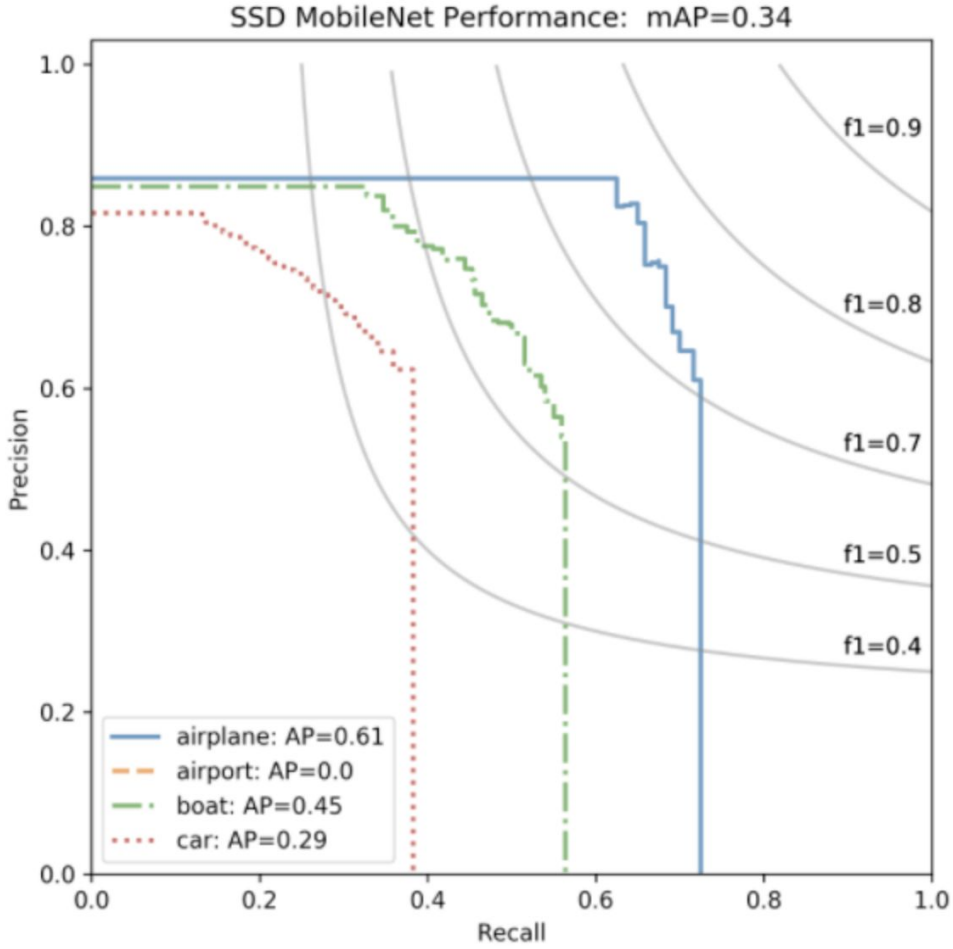
Precision - Recall

¿Por qué es importante un valor única para la métrica?

Precision: porcentaje de predicciones correctas dentro de las positivas: TP/(TP + FP)

Recall: mide que tan bién se encontraron todos los positivos. Por ejemplo, se encontró el 80% de todos los casos posibles en las primeras k predicciones: TP/(TP + FN)

Diferencia con ROC: TPr = TP / (TP + FN) FPr = FP / (FP + **TN**)


SSD MobileNet Performance: mAP=0.34
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
f1=0.9
f1=0.8
f1=0.7
f1=0.5
f1=0.4
airplane: AP=0.61
airport: AP=0.0
boat: AP=0.45
car: AP=0.29

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

= relevant documents for query 1

Ranking #1

Recall	0.2	0.2	0.4	0.4	0.4	0.6	0.6	0.6	0.8	1.0
Precision	1.0	0.5	0.67	0.5	0.4	0.5	0.43	0.38	0.44	0.5

= relevant documents for query 2

Ranking #2

Recall	0.0	0.33	0.33	0.33	0.67	0.67	1.0	1.0	1.0	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.33	0.43	0.38	0.33	0.3

$$average\ precision\ query\ 1 = (1.0 + 0.67 + 0.5 + 0.44 + 0.5)/5 = 0.62$$
$$average\ precision\ query\ 2 = (0.5 + 0.4 + 0.43)/3 = 0.44$$

$$mean\ average\ precision = (0.62 + 0.44)/2 = 0.53$$

	backbone	AP	AP_{50}	AP_{75}
Two-stage methods				
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2
One-stage methods				
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3
RetinaNet [7]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4

COCO for YOLOv3

```
Average Precision (AP):
  AP                  % AP at IoU=.50:.05:.95 (primary challenge metric)
  AP^IoU=.50          % AP at IoU=.50 (PASCAL VOC metric)
  AP^IoU=.75          % AP at IoU=.75 (strict metric)
```