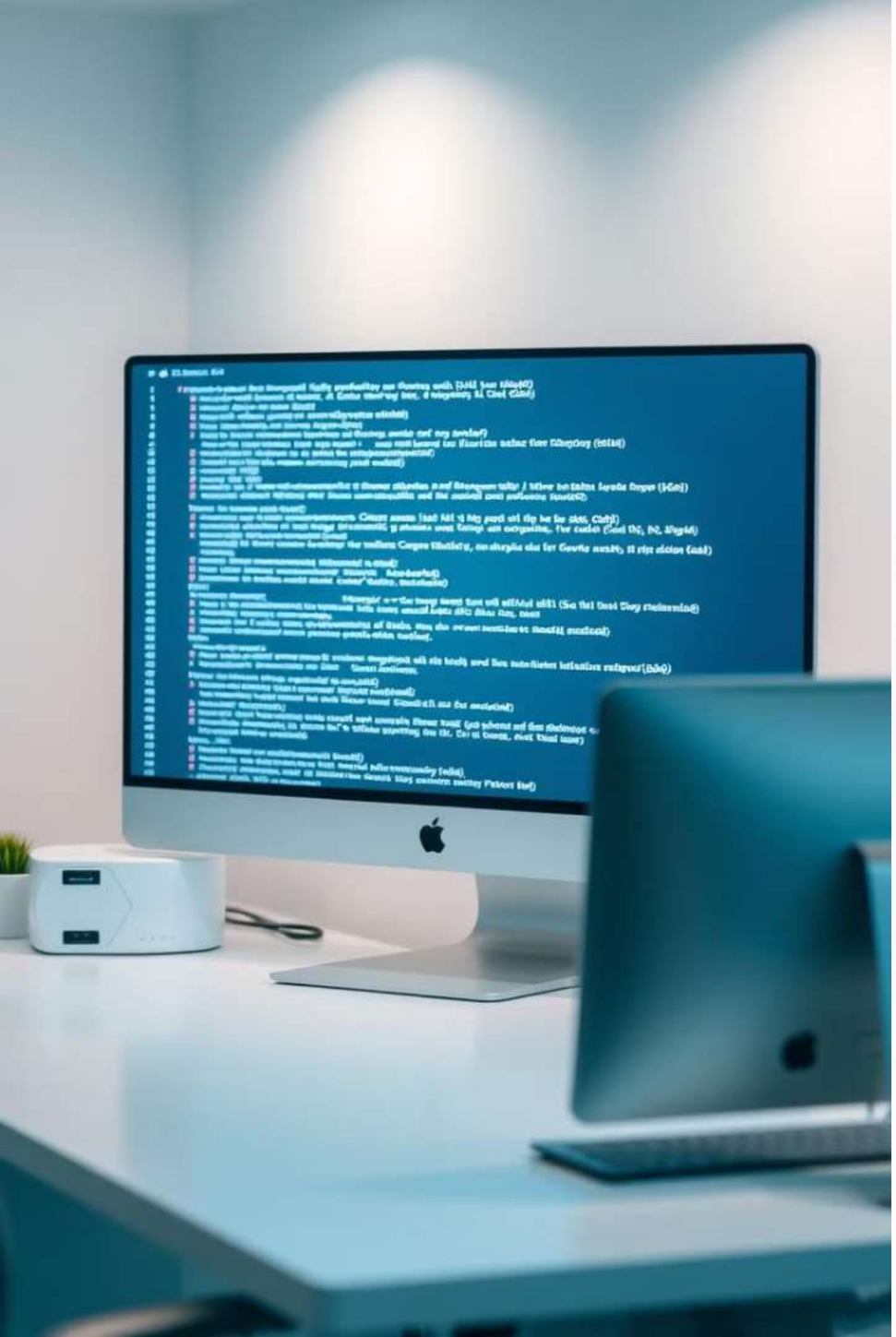
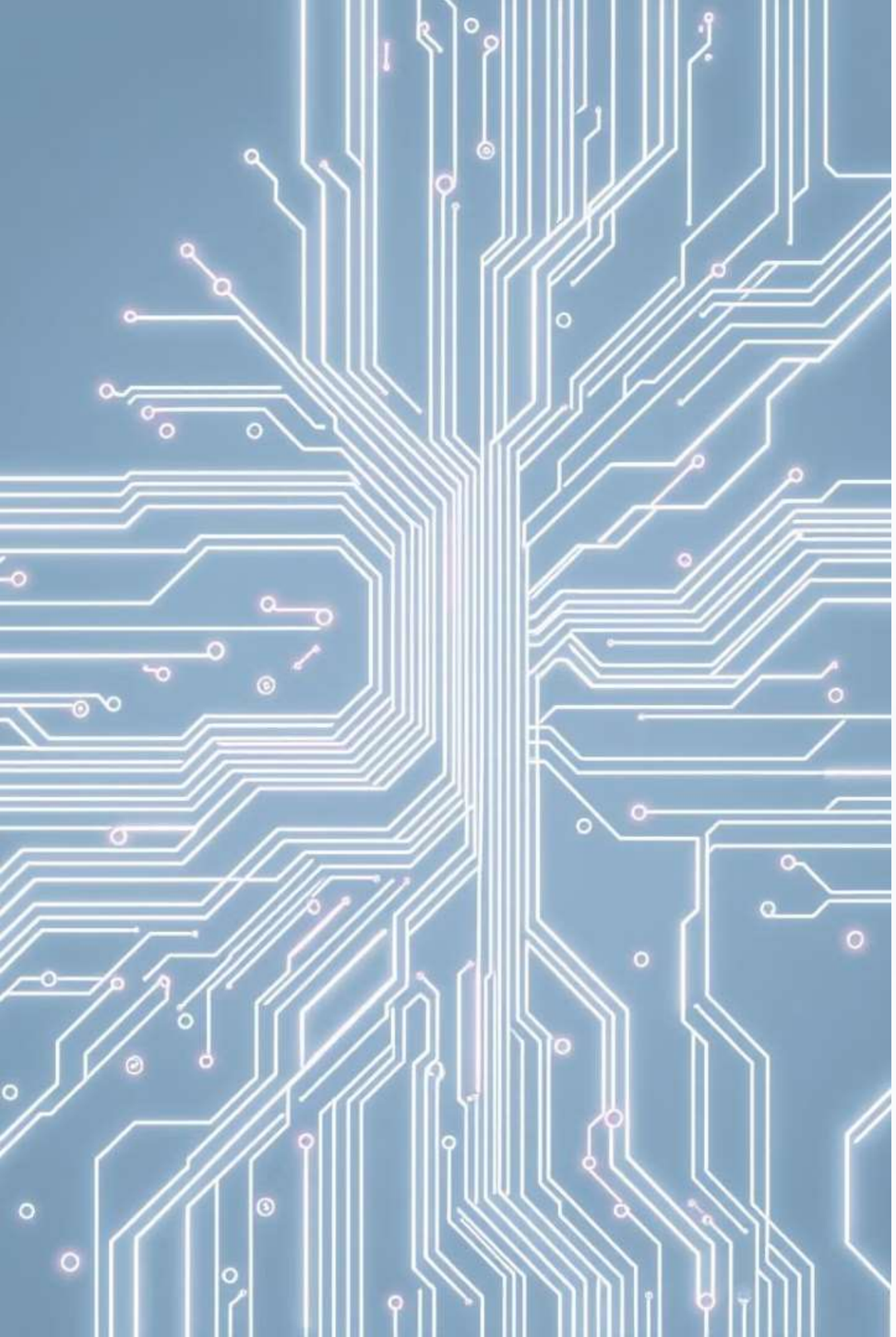


Cómo funciona YOLO para detección de objetos

YOLO (You Only Look Once) es un algoritmo revolucionario de detección de objetos en tiempo real. Su objetivo es detectar objetos en una imagen, localizándolos con un cuadro delimitador (bounding box) y clasificándolos en categorías predefinidas.. Este enfoque único permite una identificación precisa y rápida de múltiples objetos en una sola pasada. Fue introducido por Joseph Redmon



Características YOLO

1 **Enfoque en una sola red**: YOLO procesa toda la imagen en una sola pasada (en contraste con métodos como R-CNN que procesan regiones de interés de manera separada).

2 **Velocidad y precisión**: Está diseñado para ser rápido sin sacrificar demasiado la precisión.

3 **División en cuadrículas**: La imagen de entrada se divide en una cuadrícula S×S Cada celda de la cuadrícula es responsable de predecir: Uno o más cuadros delimitadores (cada uno con un puntaje de confianza). Las probabilidades de que el objeto detectado pertenezca a las clases definidas.

4 **Predicción en paralelo**: Utiliza una sola red convolucional para predecir cuadros delimitadores y clasificaciones de objetos.

Arquitectura del modelo YOLO

Etapas del Modelo

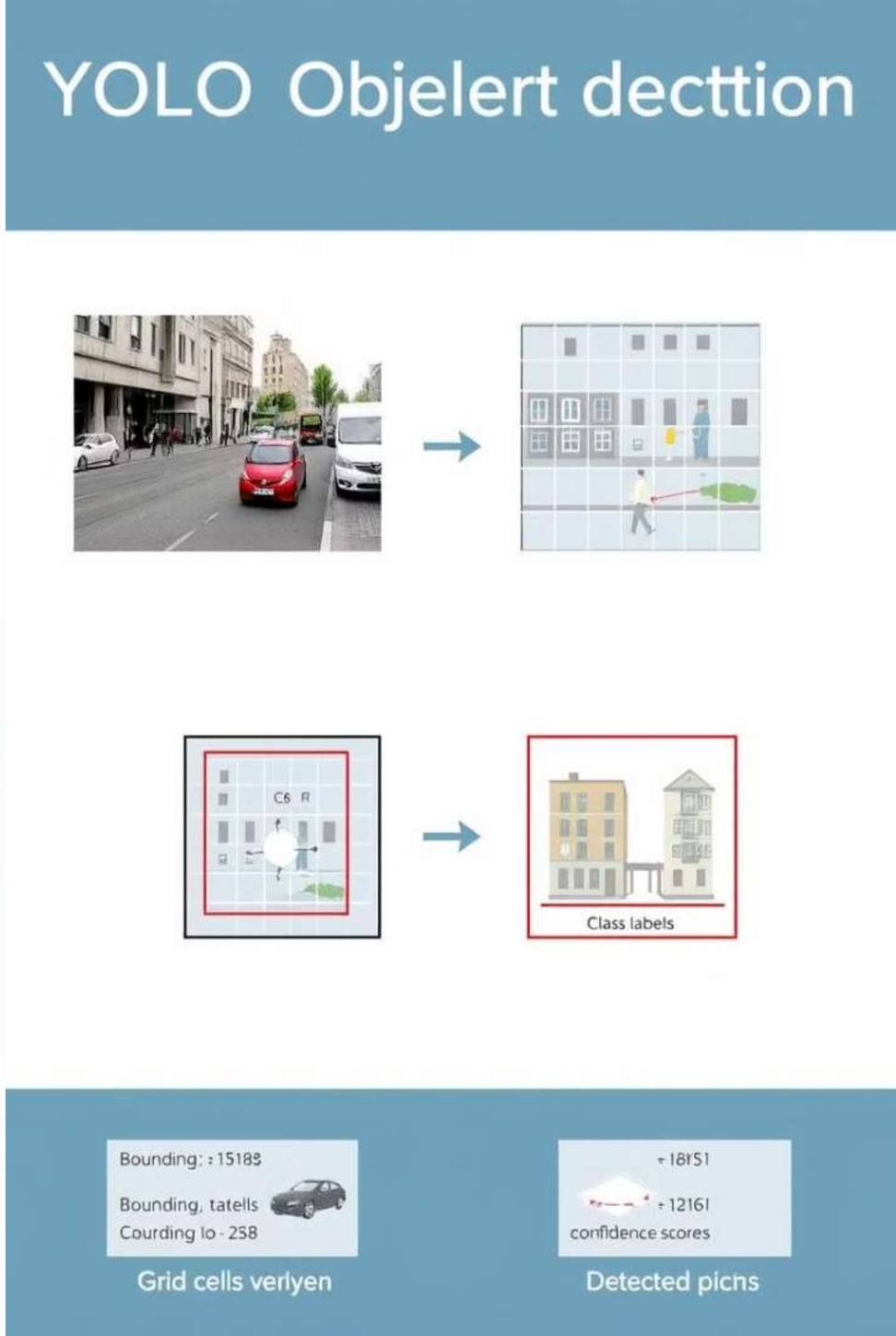
YOLO divide la imagen en una cuadrícula de celdas, cada una de las cuales genera predicciones de cuadros delimitadores y probabilidades de clase.

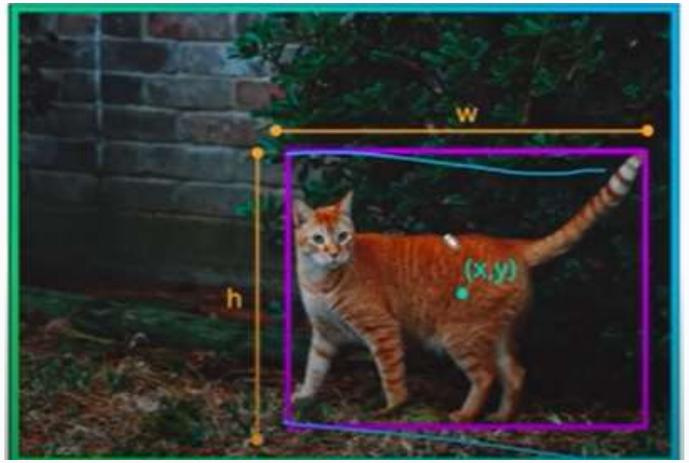
Backbone CNN

El modelo utiliza una red convolucional profunda como backbone, que extrae características visuales de la imagen de entrada.

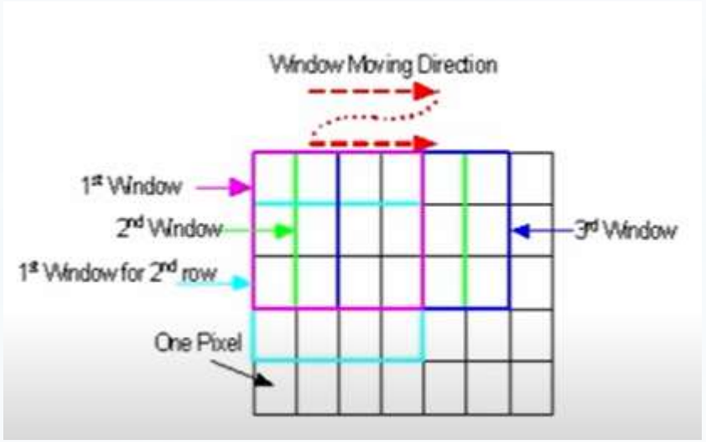
Predicciones Finales

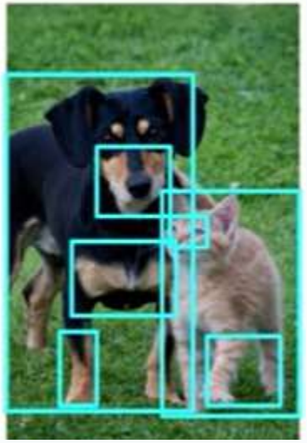
La etapa final combina las predicciones de las celdas para generar las detecciones finales de objetos.

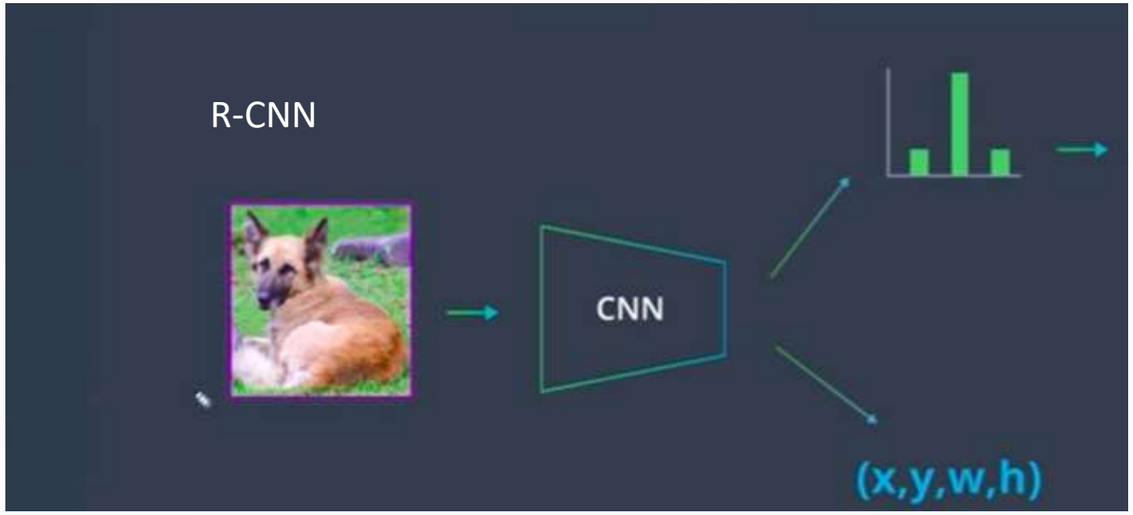


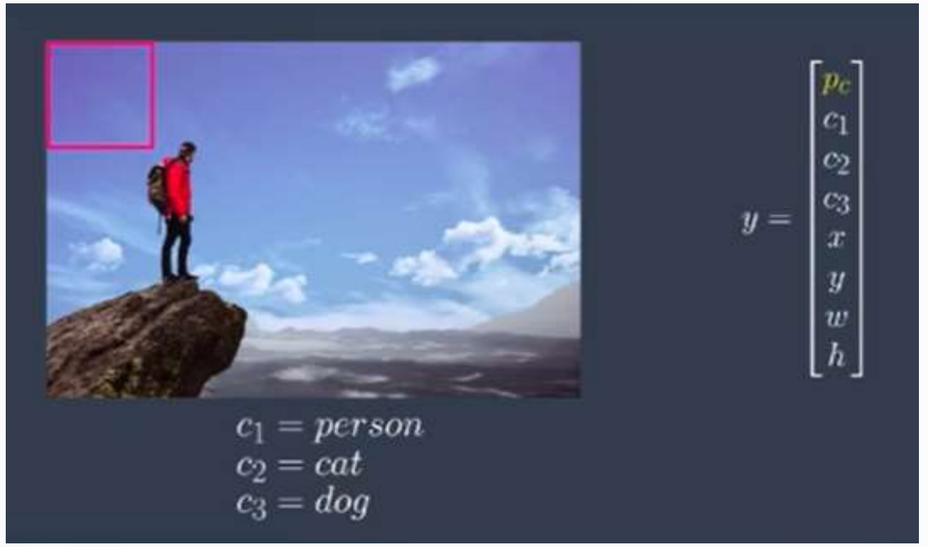

w
h
(x,y)


Cat
Feature Vector
Class Scores
Predicted Class
(x,y,w,h)
Predicted bounding box coordinates


Window Moving Direction
1st Window
2nd Window
3rd Window
1st Window for 2nd row
One Pixel



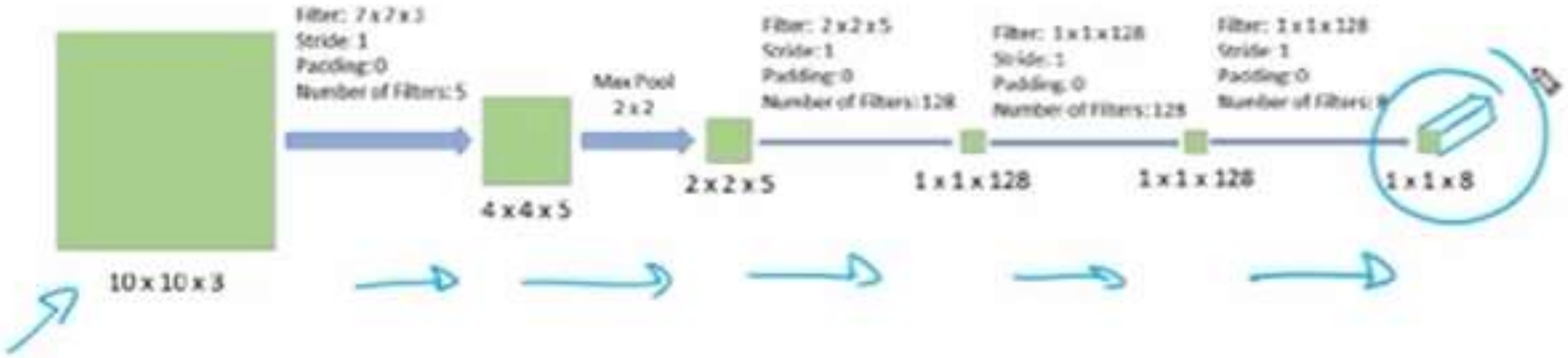

R-CNN
CNN
(x,y,w,h)



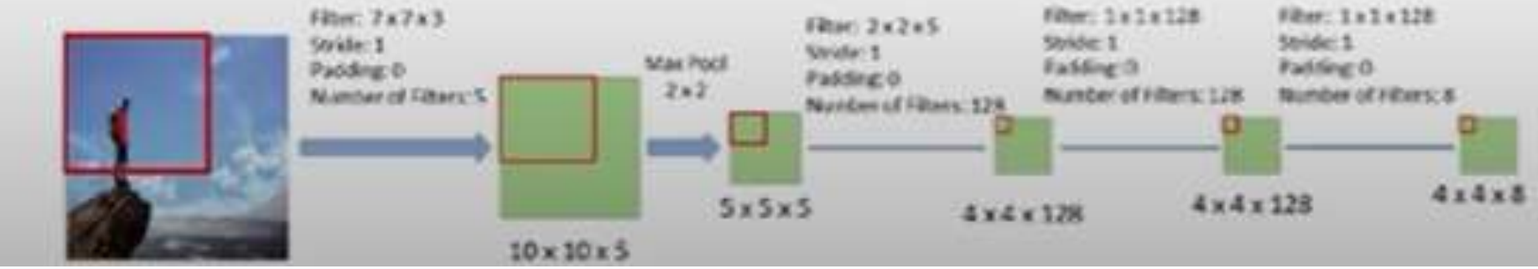
16 x 16 x 3

16 x 16 x 3

- Passing the window through a CNN



- Whole image through the CNN



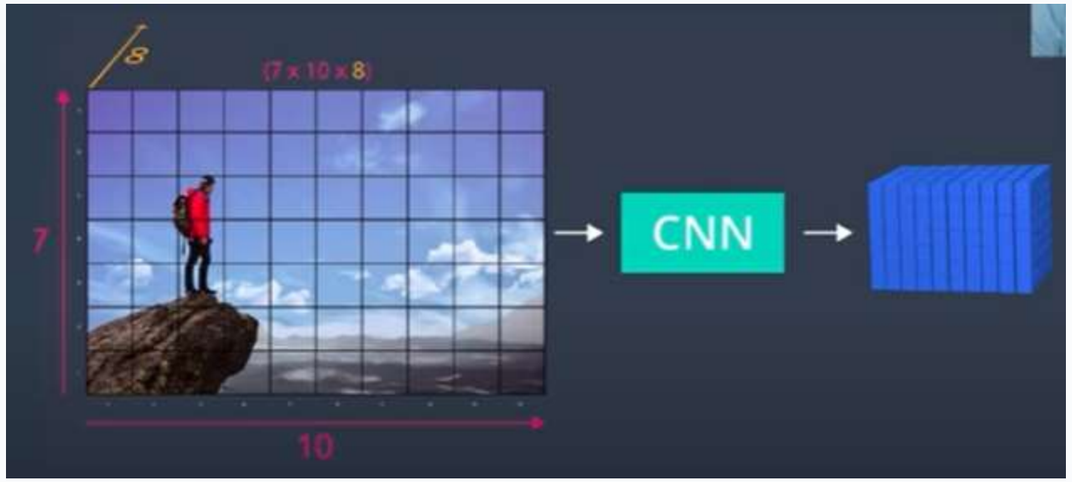
A una región en la imagen original, le corresponde una región en el mapa de características

Cada celda tiene asignado un vector de características

$$g_n = \begin{bmatrix} p_c \\ c_1 \\ c_2 \\ c_3 \\ x \\ y \\ w \\ h \end{bmatrix}$$

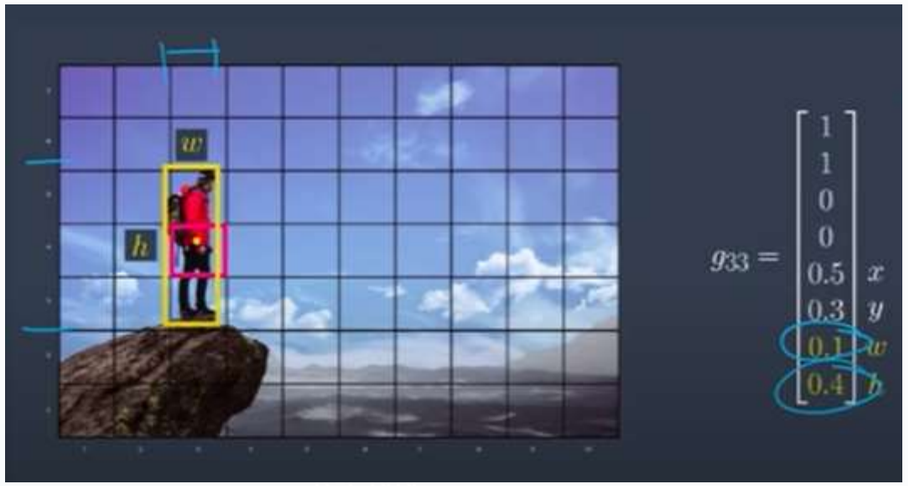
$$c_1 = person \quad c_2 = cat \quad c_3 = dog$$

R-CNN



Etiquetado

A cada una de las celdas asignar la posición del objeto
y el tamaño del cuadro que encapsula al objeto



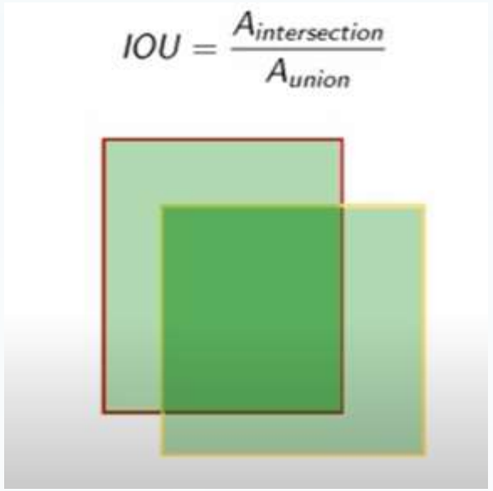
W y h es proporcional a toda la imagen en porcentaje

Varias celdas pueden tener un objeto
sin embargo solo se etiqueta la celda que tiene
el centro del objeto.

¿Cual es la celda que etiqueto?

Problemas: Tenemos múltiples regiones con los mismos objetos,
En este caso personas

Solución:
Non Maximum Supression



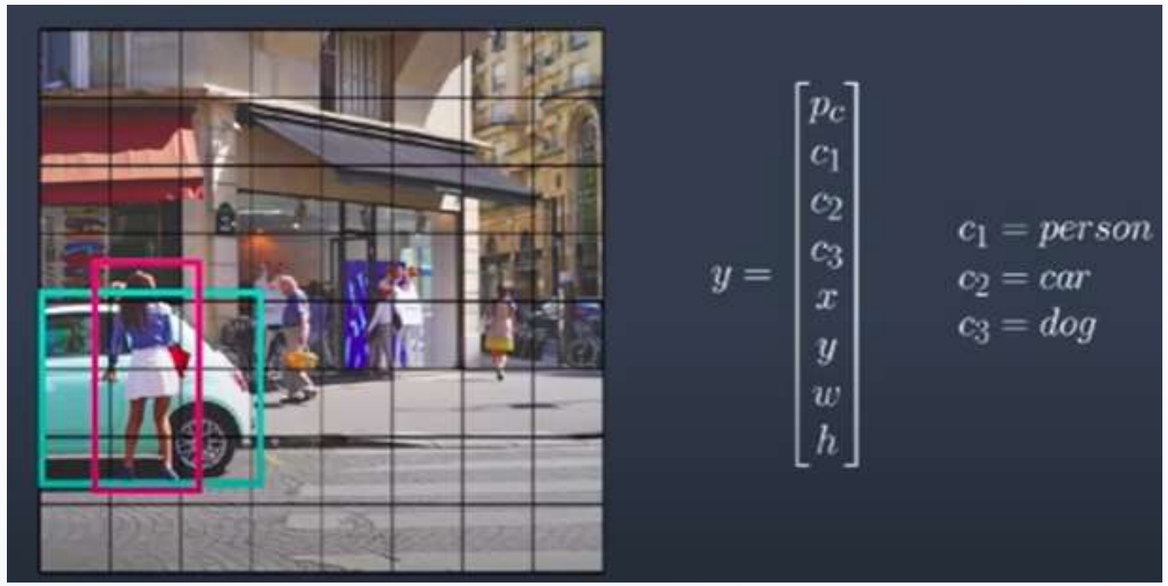
Cuando las regiones son diferentes
El resultado es 0
Sino 1

1. Observamos el vector de cada celda
2. Eliminamos todos los Bbox con un valor de probabilidad menor o igual a

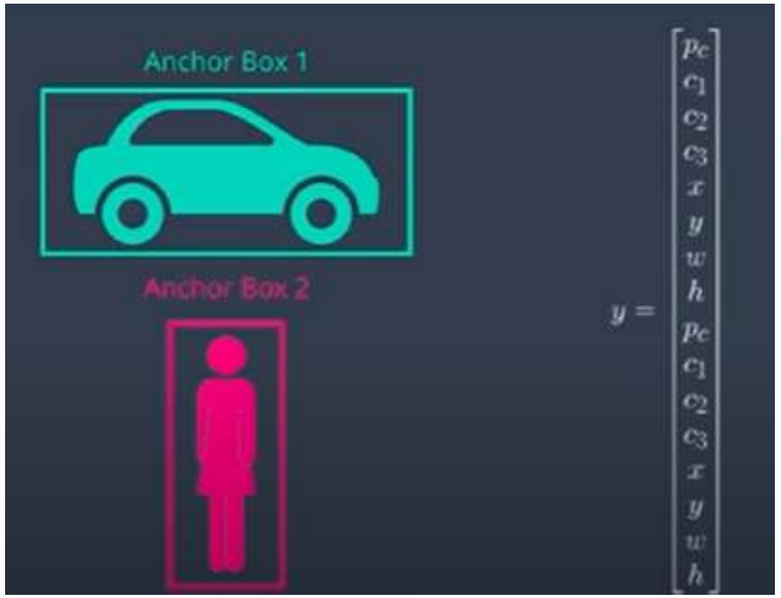
un cierto umbral.

1. Seleccionamos el Bbox con el máximo valor de probabilidad
2. Eliminamos todos los Bbox con el máximo valor de IoU (duplicidad)

Mismo centro de celda (múltiples salidas para una celda)



R-CNN



Ventajas y Desventajas de YOLO

Ventajas

Rapidez, precisión en tiempo real, capacidad de detectar múltiples objetos simultáneamente.

Desventajas

Puede ser menos preciso que otros enfoques en objetos pequeños o yuxtapuestos.

Ejemplo Práctico de Implementación de YOLO

Recolección de Datos

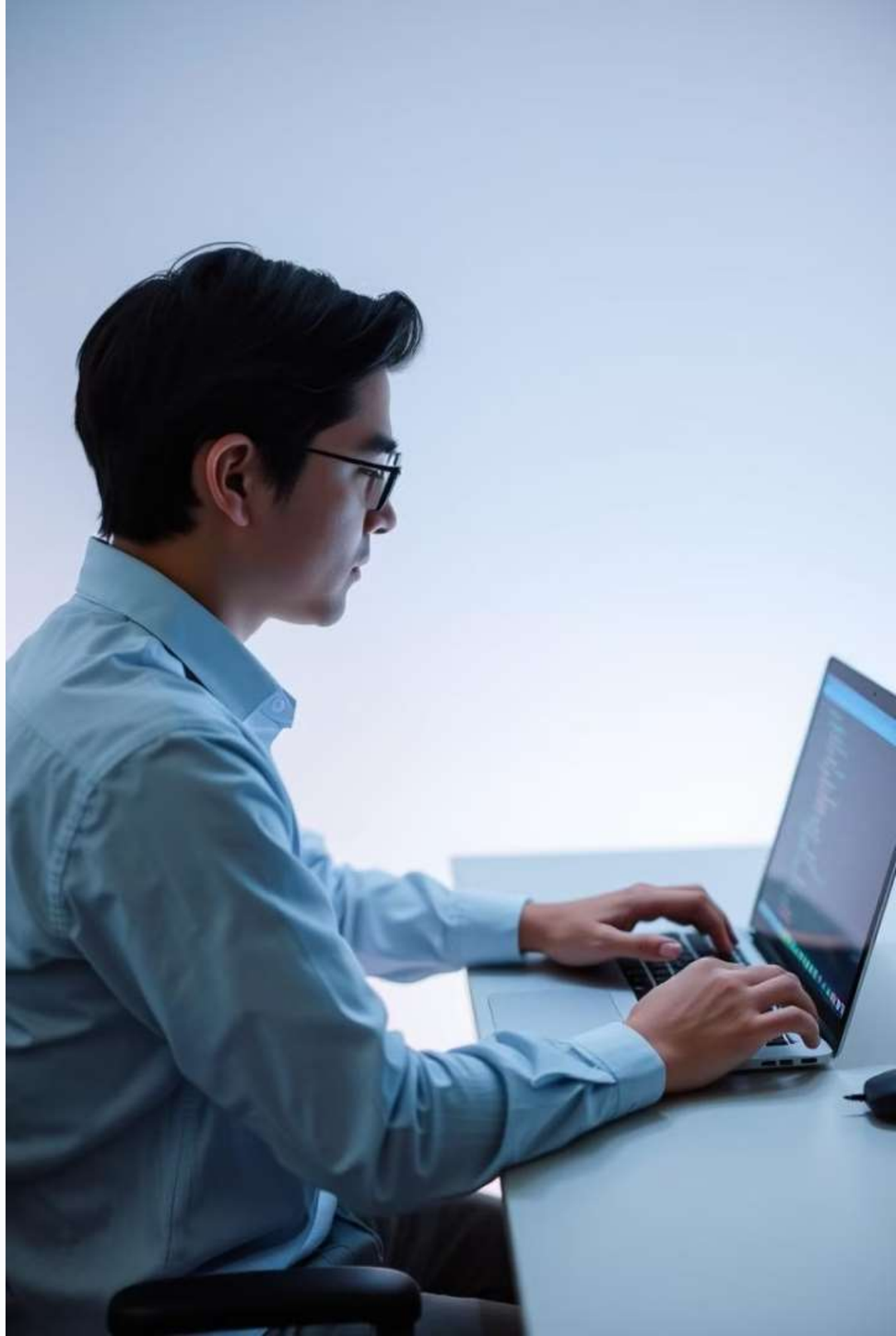
Reunir un conjunto de datos de imágenes anotadas con objetos a detectar.

Entrenamiento del Modelo

Utilizar un framework de aprendizaje profundo como TensorFlow o PyTorch para entrenar el modelo YOLO.

Implementación

Integrar el modelo YOLO entrenado en una aplicación o sistema para realizar detecciones en tiempo real.



1. Data set preparation

•Cada imagen tiene un archivo .txt con las etiquetas.

Las anotaciones contienen:

```
<clase> <x_centro> <y_centro> <ancho> <alto>
```

donde los valores están **normalizados** (de 0 a 1) con respecto al tamaño de la imagen.

•Ejemplo de una anotación para una imagen de un gato (clase 0):

```
0 0.5 0.5 0.2 0.3
```

Estructura del directorio

Organiza los archivos de la siguiente manera:

bash

```
dataset/
├── images/
│   ├── train/   # Imágenes para entrenamiento
│   ├── val/     # Imágenes para validación
│   ├── test/    # (Opcional) Imágenes para prueba
├── labels/
│   ├── train/   # Etiquetas correspondientes a las imágenes de entrenamiento
│   ├── val/     # Etiquetas correspondientes a las imágenes de validación
│   ├── test/    # (Opcional) Etiquetas para las imágenes de prueba
```

2. Crear un archivo de configuración de datos

Crea un archivo YAML (por ejemplo, dataset.yaml) con la siguiente estructura:

```yaml
train: dataset/images/train/  # Ruta a las imágenes de entrenamiento
val: dataset/images/val/      # Ruta a las imágenes de validación

# Nombres de las clases
names:
  0: gato
  1: perro
  2: pájaro
```

3. **Descargar YOLOvXX**

Clona el repositorio YOLOvXX desde GitHub:

```bash
git clone https://github.com/ultralytics/yolov5
cd yolov5
pip install -r requirements.txt
```

4. Entrenar el modelo

Ejecuta el comando de entrenamiento:

bash Copiar código

```
python train.py --img 640 --batch 16 --epochs 50 --data dataset.yaml --weights yolov5s.pt
```

•--img 640: Tamaño de las imágenes (640x640 es estándar, pero puedes ajustarlo).
•--batch 16: Tamaño del lote.
•--epochs 50: Número de épocas (iteraciones completas sobre el conjunto de datos).
•--data dataset.yaml: Ruta al archivo de configuración de datos.
•--weights yolov5s.pt: Pesos iniciales del modelo preentrenado (puedes usar yolov5m.pt, yolov5l.pt, etc.).

Si estás comenzando desde cero (sin pesos preentrenados), usa:

bash

```
--weights '' --cfg yolov5s.yaml
```

5. Evaluar el modelo

Después del entrenamiento, los resultados se guardarán en el directorio runs/train/expX/:

- **Gráficos de precisión y pérdida**: results.png
- **Modelo entrenado**: weights/best.pt (el mejor modelo según la métrica mAP).

Puedes probar el modelo entrenado con:

bash Copiar código

```
python detect.py --weights runs/train/expX/weights/best.pt --source ruta_a_tus_imagenes/
```

6. Exportar el modelo

YOLOvXX permite exportar modelos a diferentes formatos para usarlos en otras plataformas:

```
python export.py --weights runs/train/expX/weights/best.pt --include torchscript onnx coreml
```

Consejos para un buen entrenamiento

✓**Aumentación de datos**: YOLOv5 incluye estrategias automáticas de *data augmentation* (como flips, crops, y ajustes de brillo).
✓**Balance de datos**: Asegúrate de que las clases estén balanceadas en el conjunto de entrenamiento.
✓**Validación**: Monitorea la métrica *mAP* (Mean Average Precision) durante el entrenamiento para evitar sobreajuste.
✓**Hardware**: Si tienes acceso a una GPU, usa aceleración CUDA para reducir el tiempo de entrenamiento.

Aplicaciones del Modelo YOLO en la Vida Real

1

Seguridad

Detección de objetos peligrosos en sistemas de vigilancia.

2

Conducción Autónoma

Identificación de obstáculos y peatones en tiempo real.

3

Robótica

Guía de robots industriales y sistemas de manipulación.

Links de interés

- ✓https://www.youtube.com/watch?v=-QWxJ0j9EY8
- ✓https://github.com/computervisioneng/train-yolov8-custom-dataset-step-by-step-guide/tree/master/google_colab
- ✓https://storage.googleapis.com/openimages/web/visualizer/index.html
- ✓https://www.kaggle.com/datasets/bigquery/open-images