Detección de defectos con Deep Learning en imágenes industriales tomadas con rayos X

TFG, Ingeniería Electrónica de Comunicaciones

Pablo Suárez[1,2]
[1]Universidad Complutense de Madrid
[2]Facultad de Ciencias Físicas, [3]DACyA

Supervisores: Maria José Gómez Silva[1,2,3] y Jesús Chacón Sombría[1,2,3]

21 de abril de 2024

Índice

1. Introducción

¿**Finalidad** de este **TFG**?

- Detección de **defectos** → ¿dónde? → en imágenes de **radiografías** de piezas industriales.

¿**Cómo**?

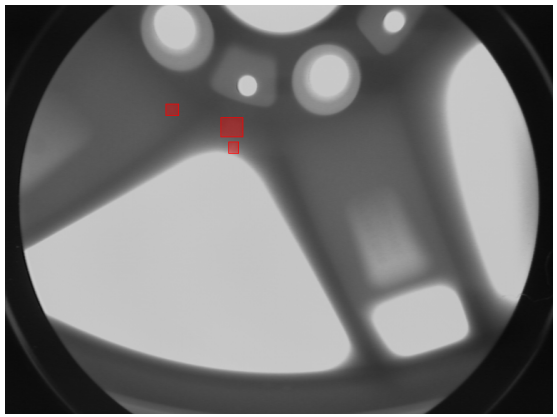
- Con la ayuda del *Deep Learning.*

Herramientas:

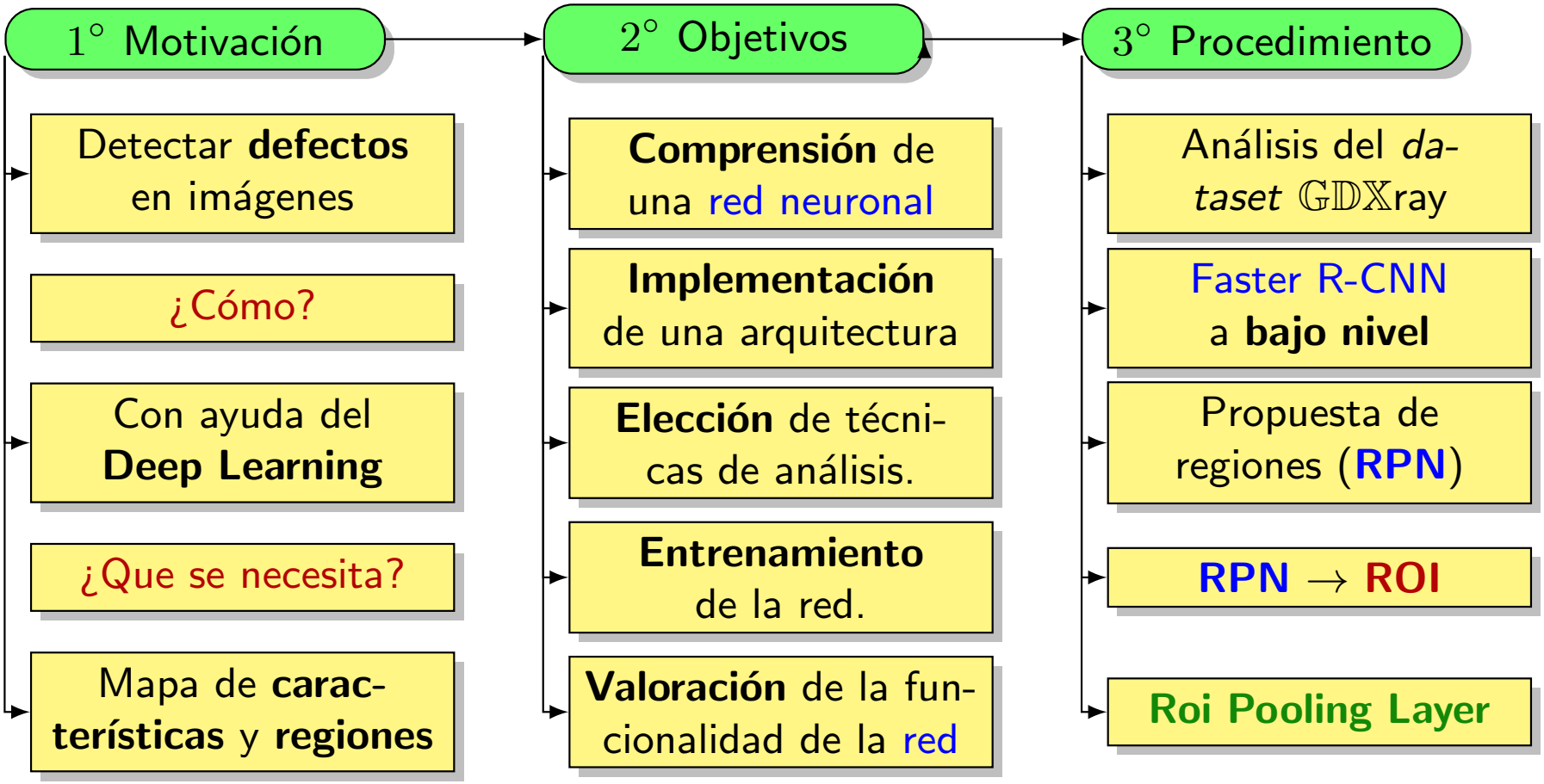
- `Python` → `pandas`, `numpy`, `torchvision`, `PIL`, `matplotlib`, `TensorFlow` y `Keras`.

¿Por qué es **relevante**?

- Puede ahorrar tiempo y dinero.
- Rapidez.
- Precisión.



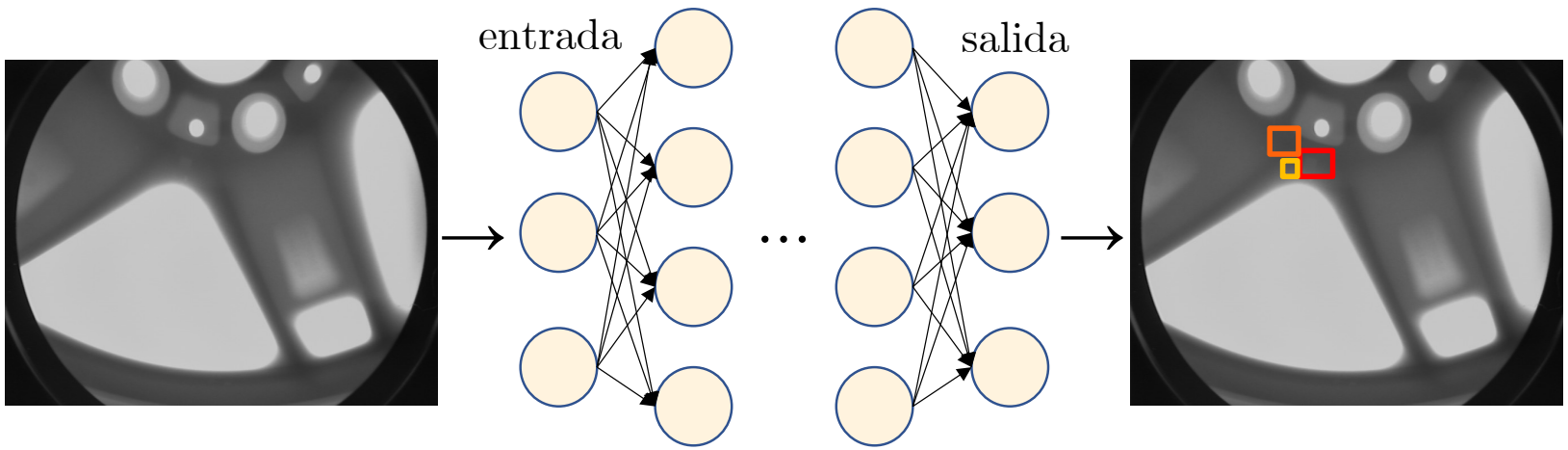
2. Motivación, objetivos y procedimiento



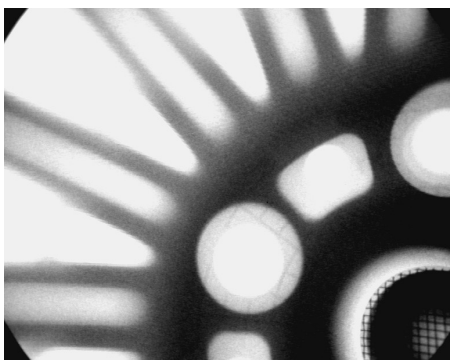
3.1. El *Deep Learning* para la detección de defectos

1. ¿Qué es el *Deep Learning*?
   - Técnica de **IA** que utiliza **redes neuronales** → analizar imágenes y localizar defectos/objetos.
   - *Deep Learning* ⊂ *Machine Learning*.
   - ¿Cómo aprende? → reconociendo patrones y características en las imágenes.

2. Cualidades del *Deep Learning*:
   - Herramienta **óptima** para → **TAREAS REITERATIVAS.**
   - Tras un **largo entrenamiento** con **muchos datos** → **precisión.**



- Varios **tipos** de redes neuronales.
- ¿Por qué una red convolucional? $\rightarrow$ extracción de su **MAPA DE CARACTERÍSTICAS**.
  - ¿**Cómo**? con la convolución $\rightarrow p(x) = \mathbf{w} * \mathbf{X} = \sum_{a=-s}^{s} \mathbf{w}(a)\mathbf{X}(x-a)$
  - $\mathbf{X} \in \mathbb{R}^{n \times m}$ representa la imagen de entrada, $\mathbf{w} \in \mathbb{R}^{i}$ representa los pesos y s viene determinada por el tamaño del filtro y representa el *stride*



$\mathbb{R}^{572\times768}$ $\rightarrow$

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

$\mathbf{X}$ $*$

0	1
2	3

$\mathbf{K}$ $=$

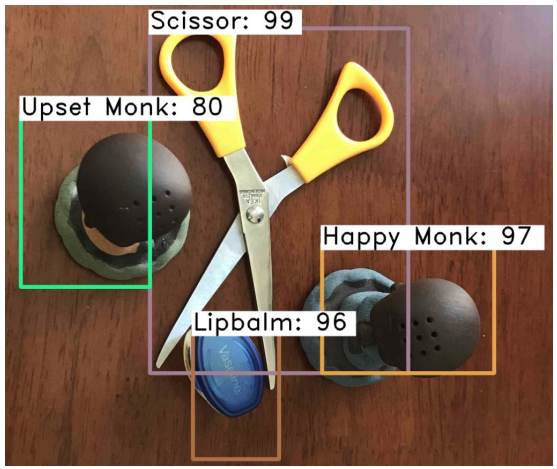
0	8
6	8

$\mathbf{X} * \mathbf{K}$

4. Diferencia entre objetos y defectos.

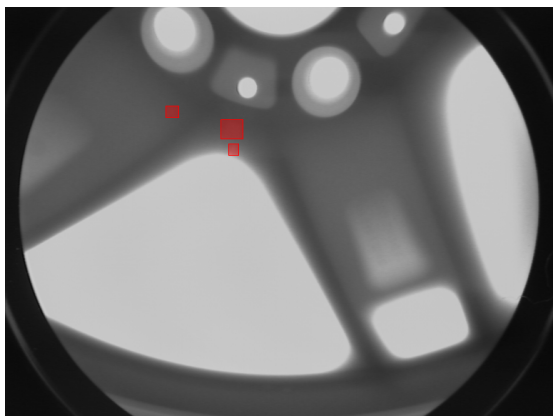
Objetos

- **Grandes** proporciones, $\frac{\texttt{size(objeto)}}{\texttt{size(imagen)}} \in (0{,}4 - 1)$
- Siluetas o formas características.
- **Policromáticos**.
- **Gran** variedad de modelos **pre-entrenados** y bases de datos.



Defectos

- Suelen ser **pequeños**, $\frac{\texttt{size(objeto)}}{\texttt{size(imagen)}} \in (0{,}01 - 0{,}15)$
- **Amorfos**.
- **Monocromáticos**.
- **Pocos** modelos y **pocas** bases de datos.
- Clasificación **BINARIA.**



Similitudes: ambas tareas de detección $\rightarrow$ **COMPLEJAS** $\therefore$ esto requiere de estructuras de red **MÁS COMPLEJAS.**

5.1. Faster R-CNN, introducción

¿Qué implica extraer **características** y **proponer regiones**?

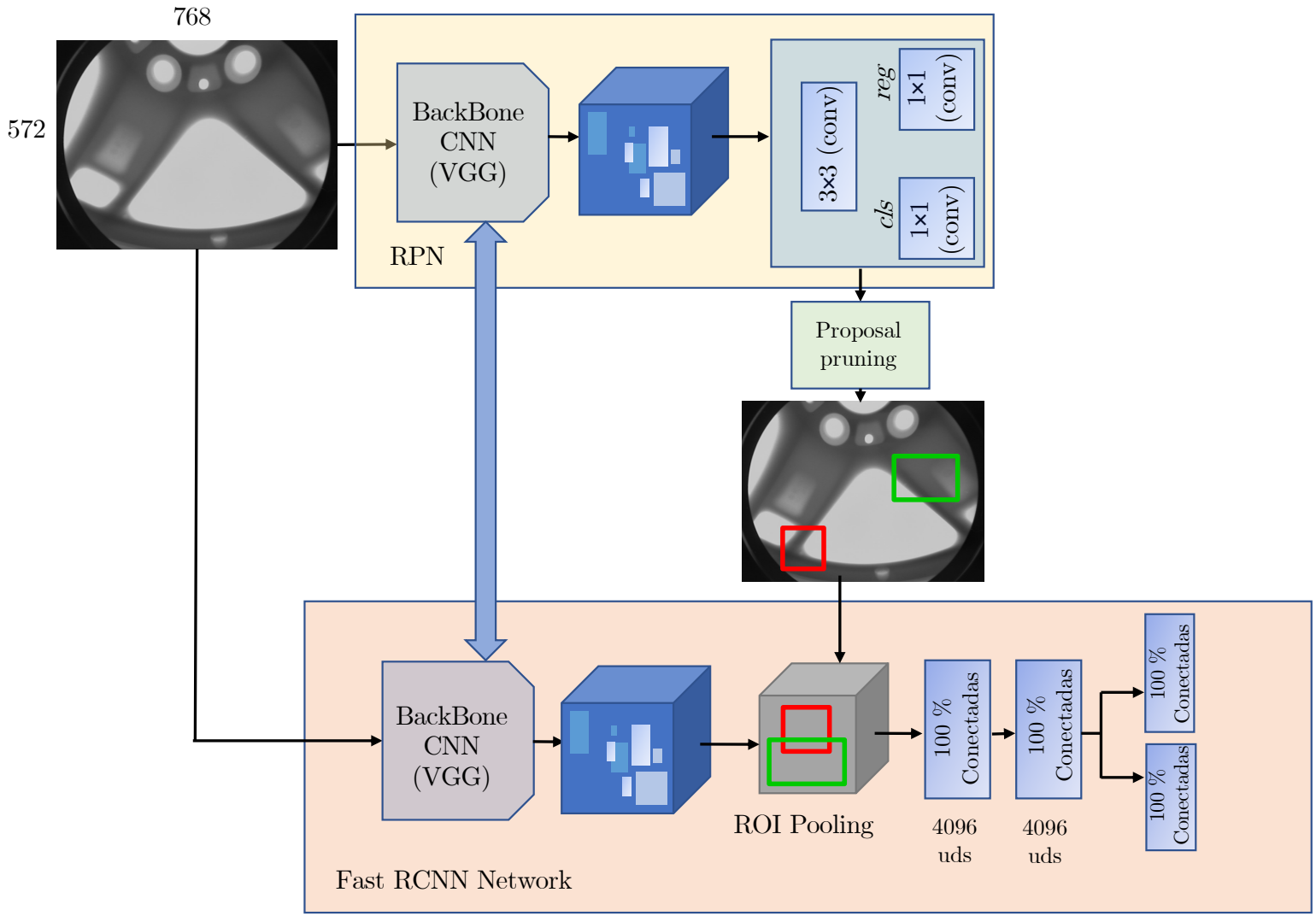
- Debido a su **complejidad** $\rightarrow$ dividir la tarea en **capas**.
- ($\uparrow$ **capas**) $\rightarrow$ ($\uparrow$ **complejidad** de la arquitectura) $\rightarrow$ **más control**.
- ¿**Solución**? $\rightarrow$ Faster R-CNN

¿Por qué la **Faster R-CNN**?

- Arquitectura orientada a la detección de objetos $\rightarrow$ **MUY COMÚN.**
- En [RHGS15] $\rightarrow$ **mAP** $\in (75 - 99)$ [%]
- **Mayor** $\boldsymbol{\eta}$ respecto a la **R-CNN** y a la **Fast R-CNN**.
- Esta arquitectura extrae **características** y **propone regiones.**

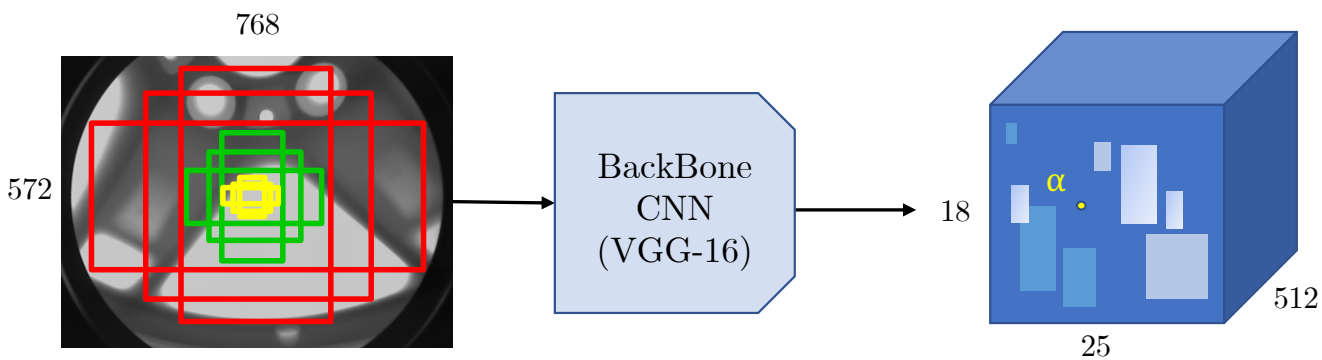
5.2. Componentes de la *Faster R-CNN*

La arquitectura de la **Faster R-CNN** es la siguiente:



5.3. El *backbone*: la VGG-16

- Es el **corazón** de la **Faster R-CNN** $\rightarrow$ **VGG-16.**
- ¿Por qué la **VGG-16**? $\rightarrow \varepsilon_{cls} = 7,32\,\%$ y $\varepsilon_{loc} = 25,32\,\%$.
- **stride = 16.**
- Encuadrada dentro de la **RPN.**
- Función **principal** $\rightarrow$ extracción del mapa de características.
- ¿**Cómo**? $\rightarrow (\mathbb{R}^{300\times400} * \mathbb{R}_i^{3\times3})|_{i=(1,..,512)} \rightarrow$ característica de menor dimensión.
- **M.C** $\in \mathbb{R}^{18\times25\times512}$



5.3. *Anchors*

- Disposición de **anchors** para cada punto del mapa de características.
- ¿Para qué? → Para saber si hay o no defectos.
- ↑ **stride** → $\#_{anchors} \downarrow$
- Si **pos** ∈ **M.C** | **pos** $\in \mathbb{R}^{18\times25\times9}$ → 4050 **anchors** en total.
- Inicialmente → **anchors negativos.**
- Dependiendo de la **IoU**:
  - **anchor negativo.**
  - **anchor positivo.**
  - **anchor descartado.**

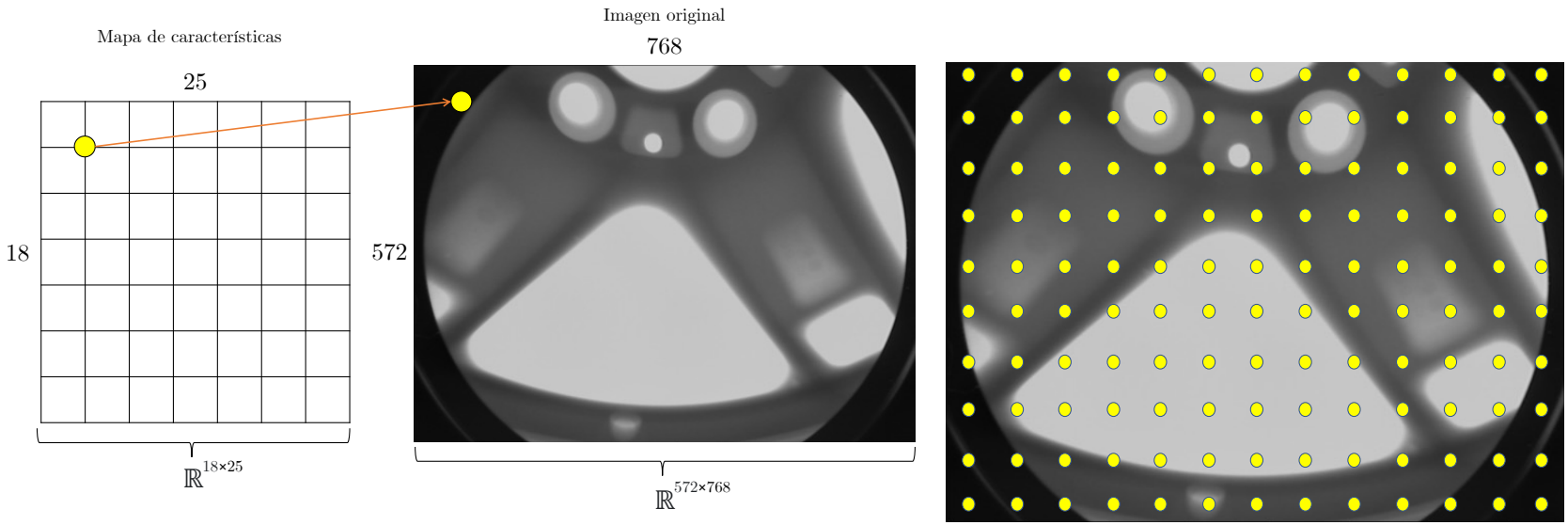
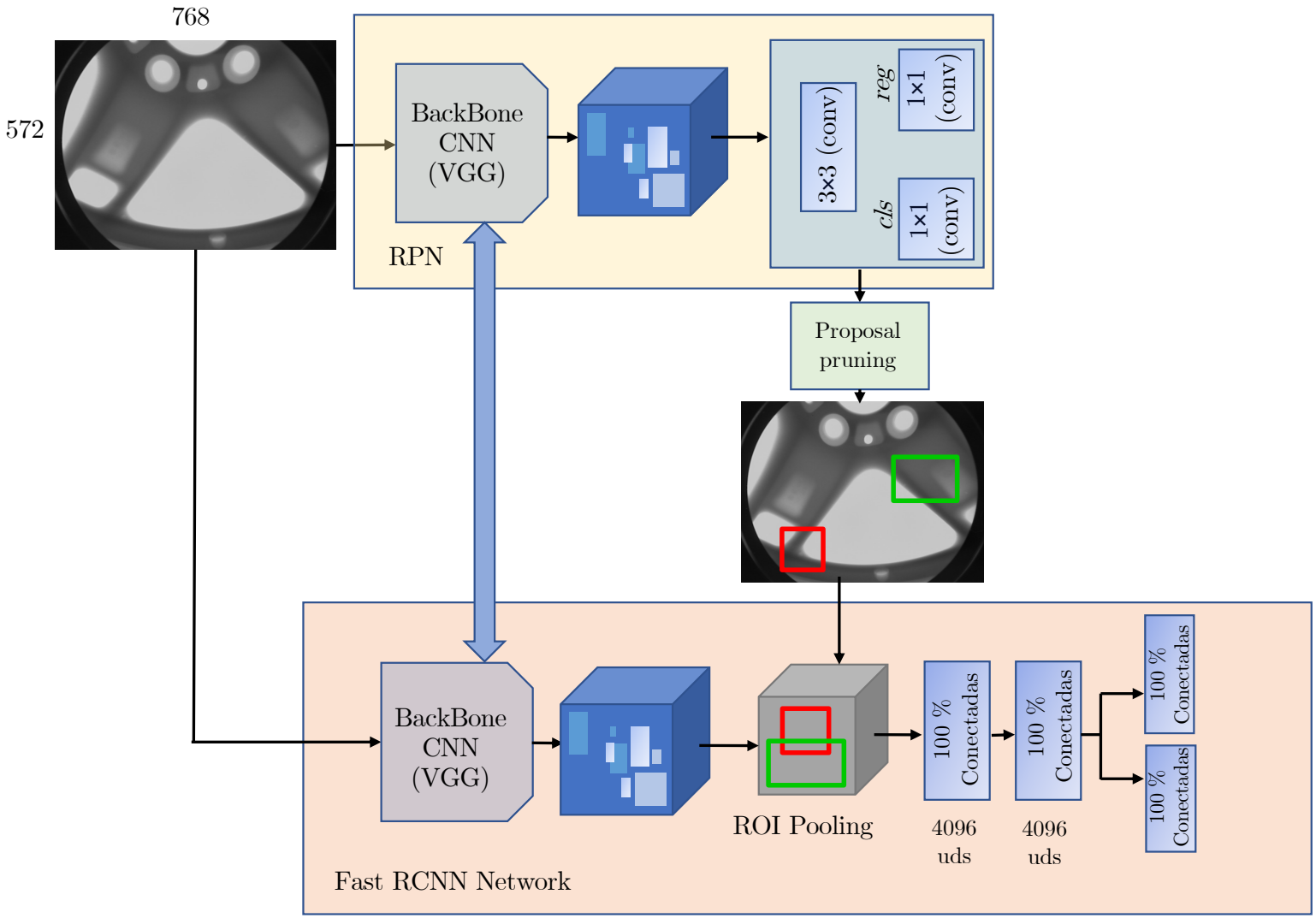


Figura: Dinámica de disposición de los *anchors*.

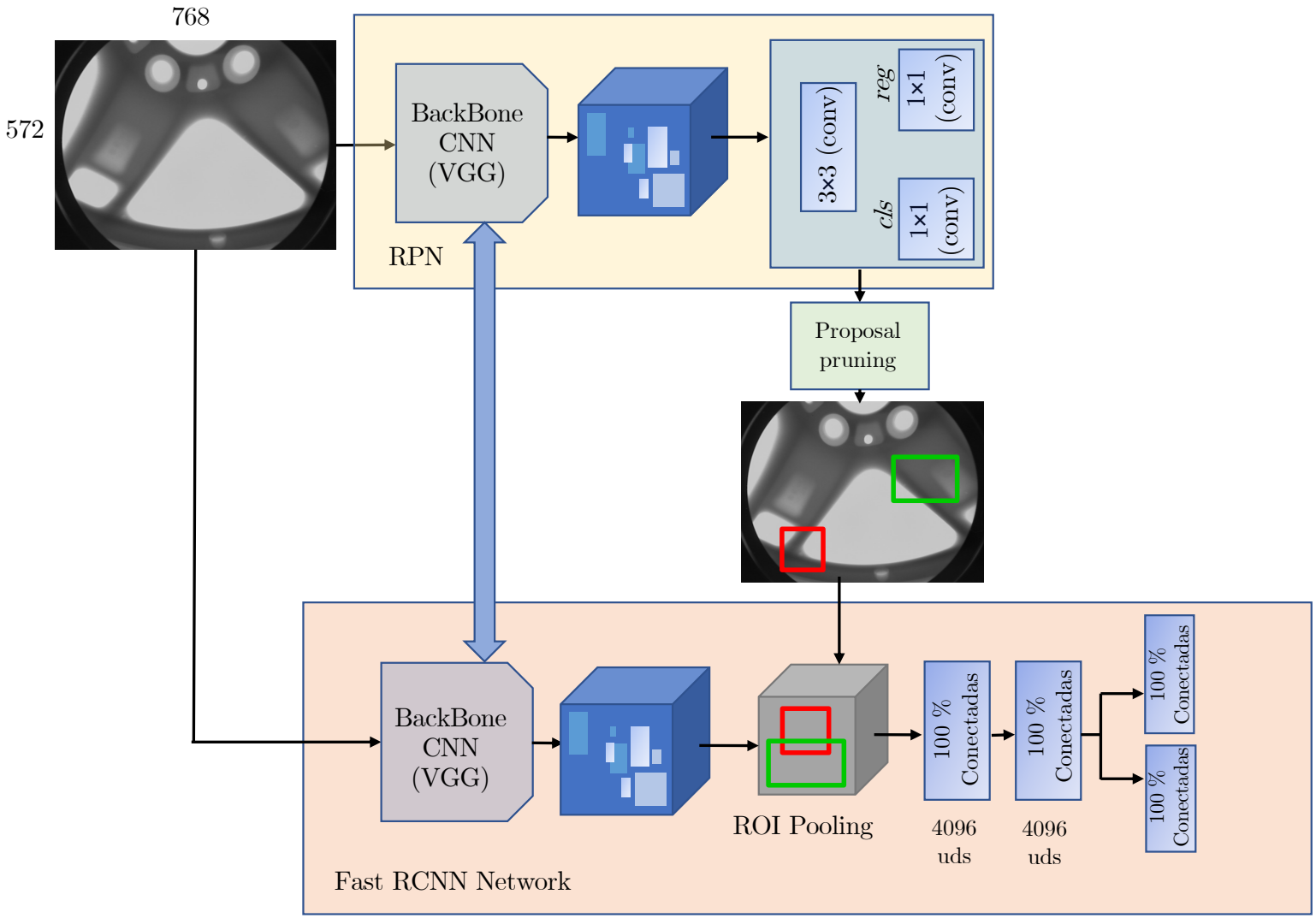
Arquitectura de la *Faster R-CNN*



5.4. Propuesta de regiones (RPN)

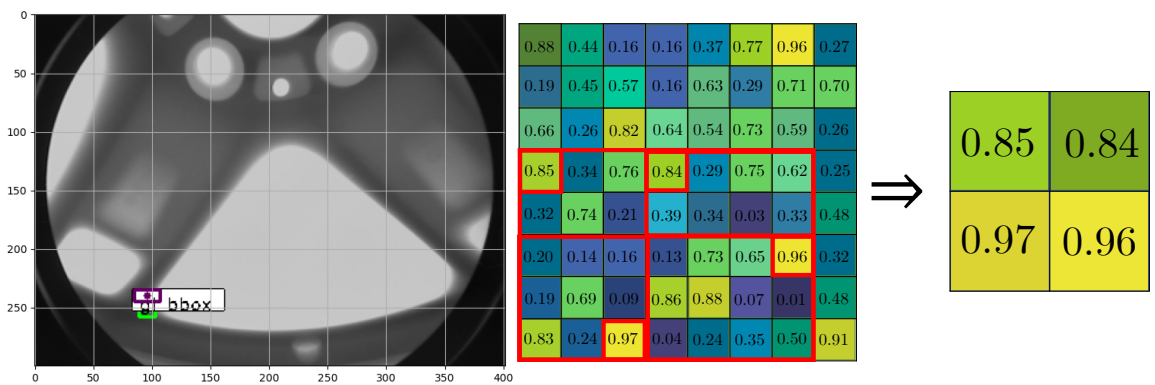
1. Una vez se tienen:
   - El mapa de características.
   - Los **anchors** posicionados.
2. Cada **posición** de la ventana móvil $\rightarrow$ PROPUESTAS DE REGIONES
3. La red se desplaza por cada **píxel** del **M.C** $\rightarrow$ defecto ✓ o defecto ×
4. ¿Cómo se comprueba? $\rightarrow$ IoU $= J(\mathbf{A},\mathbf{B}) = \frac{|\mathbf{A} \cap \mathbf{B}|}{|\mathbf{A} \cup \mathbf{B}|}$
5. Capas de **reg** y **cls**:
   - Etapa **reg** $\rightarrow$ perfecciona las coordenadas.
     - `y_rpn_regr` $\in \mathbb{R}^{1 \times 18 \times 25 \times 72}$.
   - Etapa **cls** $\rightarrow$ $Prob$(defecto ✓)
     - `y_rpn_cls` $\in \mathbb{R}^{1 \times 18 \times 25 \times 18}$.

Arquitectura de la *Faster R-CNN*



5.5. *ROI pooling*

- Una vez se tiene:
  - **Propuesta de regiones** ✓
  - **Probabilidades** correspondientes ✓
- ¿Siguiente paso? → transformar las **propuestas de regiones** en **ROIs.**
- ¿Para qué? → de los **ROIs** → elementos representables → **bounding boxes.**
- ¿Cómo? → `rpn_to_roi`
  - **Re-escalado** de *anchors.*
  - **Regresión.**
- ROIpooling
  - Procesa **ROIs** → tamaño específico, con → `max pooling`.
  - Etapa **cls** → **tamaño fijo.**
  - **coordenadas** ∈ **M.C** → **coordenadas** ∈ **img**
  - ¿**Salida**? → clases y mejores coordenadas.



5.6. Entrenamiento

- ¿**Conjunto muestral**? → repositorio $\mathbb{GDX}$ray → 2727 imágenes.
- Parámetros más influyentes:
  - ↑**profundidad** de la red → ↓ **resolución M.C.**
  - ↑$\#_{imágenes}$ → ↓ **sobre-ajuste**.
  - ↑$\#_{rois}$ → ↑ **desbalance**.
  - ↑$\#_{max,BB}$ → ↓ **filtrado**.
  - ↑**resolución** → ↑ **representación de defectos**.

- **Métricas**
  - **4 funciones de pérdidas.**
    - `loss_rpn_cls` → resultado de clasificación del modelo RPN.
    - `loss_rpn_regr` → resultado de la regresión del modelo RPN.
    - `loss_class_cls` → resultado de clasificación del modelo clasificador.
    - `loss_class_regr` → resultado de regresión del modelo clasificador.
  - **mAP:**
    - ¿Cuánta precisión? → **IoU**
    - mAP = cmp(umbral(**IoU**), **IoU**).

6.1. Resultados del entrenamiento

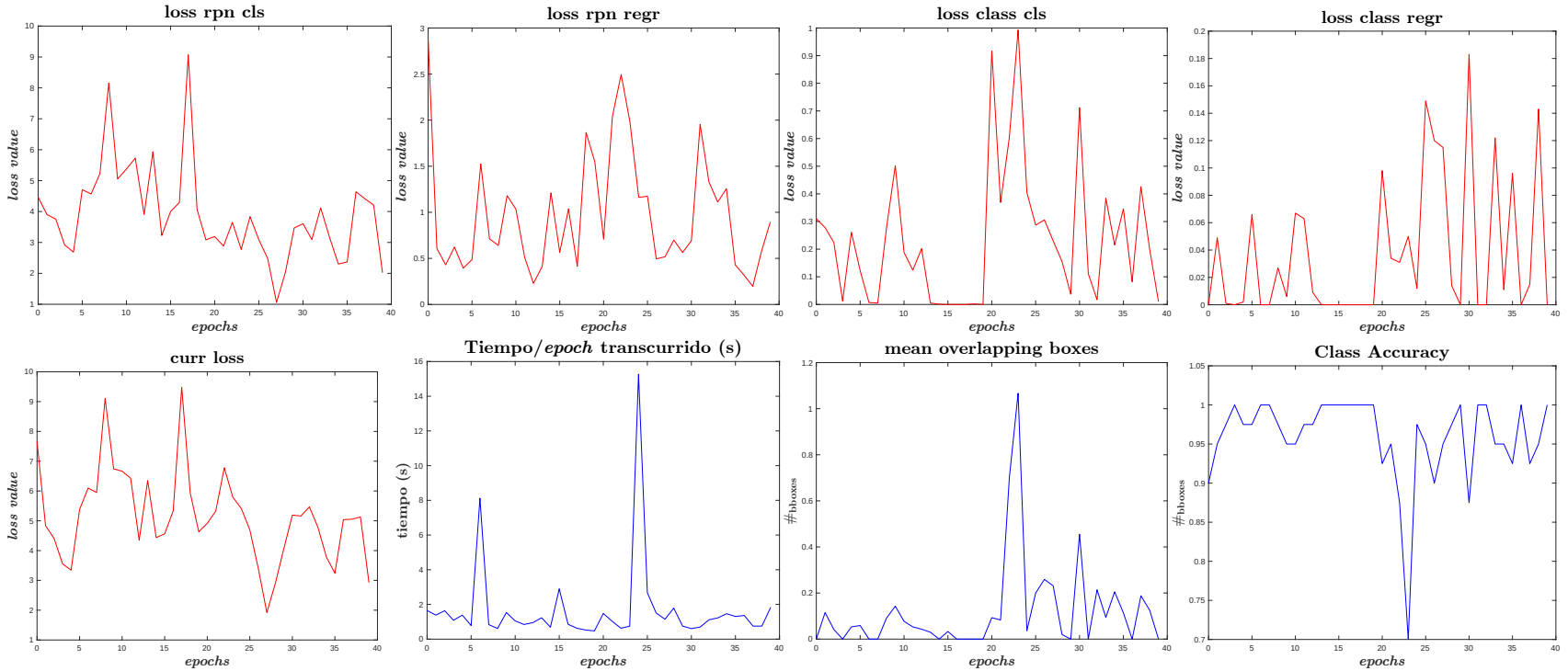


Figura: Resultados del primer entrenamiento.

6.1. Resultados del entrenamiento

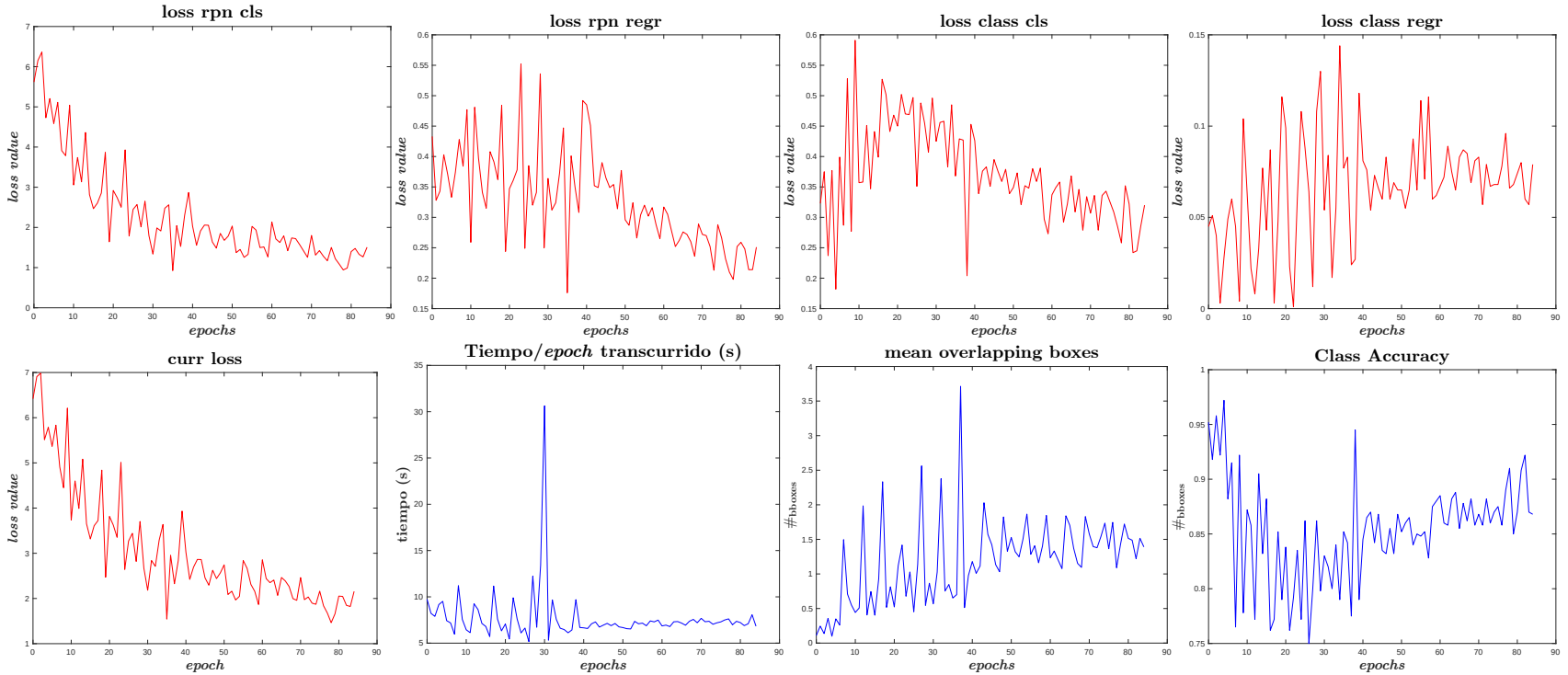


Figura: Resultados del cuarto entrenamiento.

- **mAP** $= 0 \quad \forall$**entrenamientos.**

6.2. Regiones propuestas por la red

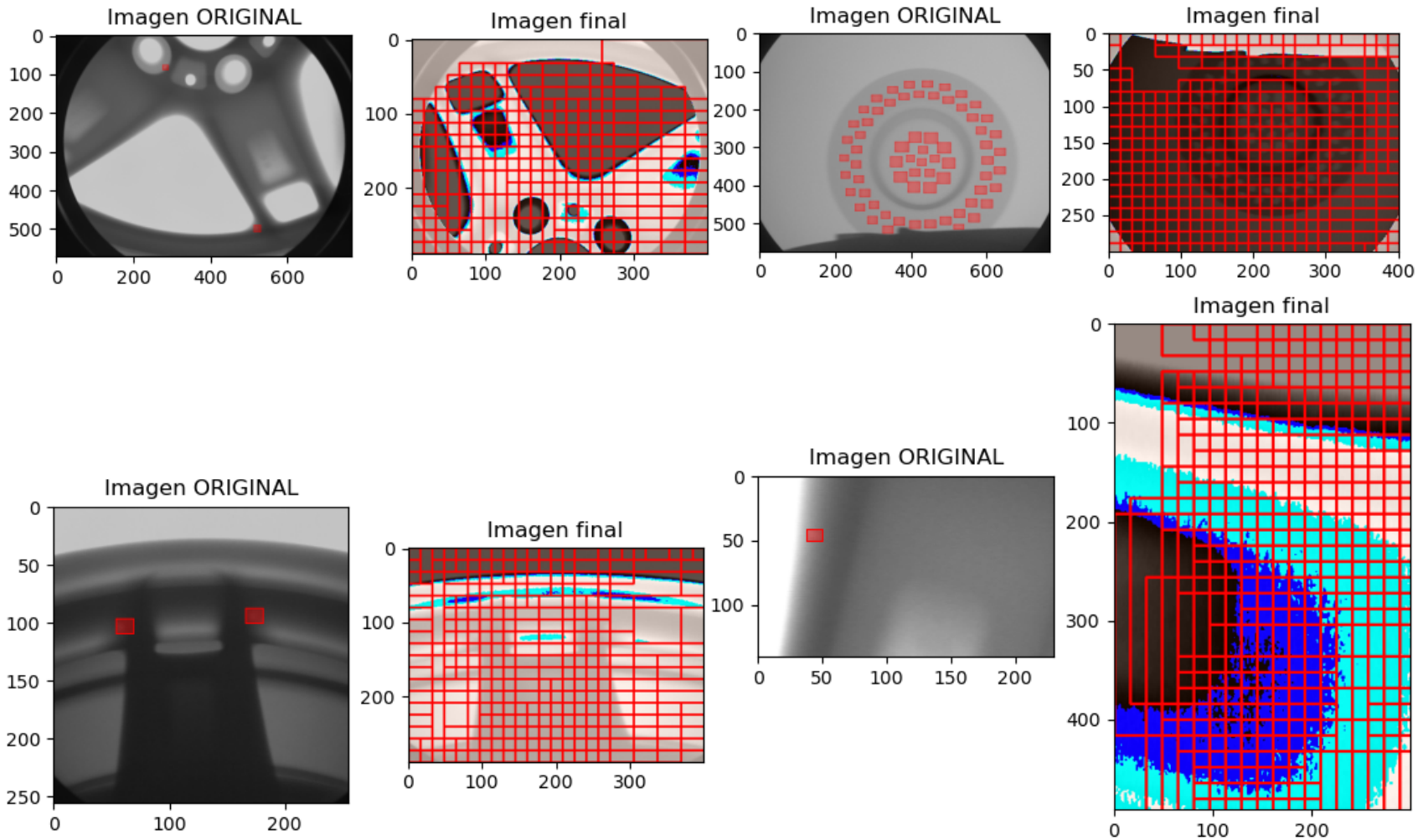


Figura: Algunas propuestas de regiones para algunas imágenes

7.1. Conclusiones

- Una **baja profundidad** $\rightarrow$ ~~asegura~~ **TODOS LOS DEFECTOS.**
- Tarea de naturaleza **desbalanceada** $\rightarrow$ **reg**(× defectos) $>$ **reg**(✓ defectos)
- La red **COLAPSA** $\rightarrow$ **NO APRENDE.**
- **Comprensión** al detalle $\rightarrow$ causas de un mal aprendizaje.
- En **RPN**, los parámetros más importantes $\rightarrow$ **escalas**, **resoluciones**.
- **Buen entrenamiento si:**
  - **Buen mapa de características**.
  - Conjunto muestral **grande.**
  - ~~**Determinismo**~~

- ↑ **Resolución** y ↑ **conjunto muestral.**
- *Transfer Learning* → reciclar **modelos pre-entrenados.**
- **Mask R-CNN** → **(+1) capa de segmentación** → **+PRECISIÓN.**
- **FPN** → múltiples **mapas de características.**

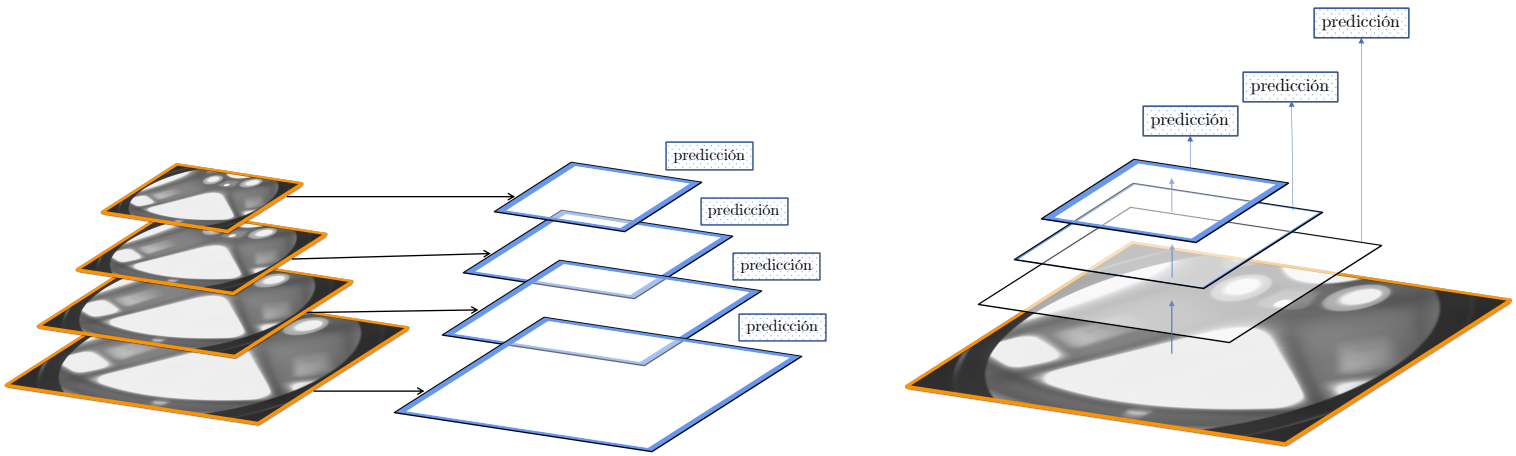


Figura: Dinámica de la **FPN**.

Gracias por su atención.

¿Preguntas?

Pablo Suárez Reyero

pasuar03@ucm.es

`https://github.com/jcsombria/tfg-psr`

Propuesta de regiones (RPN)

1. Una vez se tienen:
   - El mapa de características.
   - Los **anchors** posicionados.
2. Cada **posición** de la ventana móvil $\rightarrow$ PROPUESTAS DE REGIONES
3. La red se desplaza por cada **píxel** del **M.C** $\rightarrow$ defecto ✓ o defecto ×
4. ¿Cómo se comprueba? $\rightarrow$ IoU $= J(\mathbf{A},\mathbf{B}) = \frac{|\mathbf{A} \cap \mathbf{B}|}{|\mathbf{A} \cup \mathbf{B}|}$
5. Inicialmente $\rightarrow$ **anchors negativos.**
   - Si IoU $< 0.3 \rightarrow$ **anchor negativo.**
   - Si IoU $> 0.7 \rightarrow$ **anchor positivo.**
   - Si $0.3 <$ IoU $< 0.7 \rightarrow$ **anchor descartado.**
6. Capas de **reg** y **cls**:
   - Etapa **reg** $\rightarrow$ perfecciona las coordenadas.
   - Etapa **cls** $\rightarrow Prob$(defecto ✓)

Mean Average Precision (**mAP**)

mAP:

- ¿Cuánta precisión? → **IoU**
- umbral(**IoU**) < **IoU** → **TP** → mAP
- umbral(**IoU**) > **IoU** → **FP** → mAP
- ↑ umbral(**IoU**) → **FP**↓ → **precisión**↑ → **recall**↓ (se pierden detecciones)
- ↓ umbral(**IoU**)→ **FP**↑ → **precisión**↓

$$Recall = \frac{TP}{TP+FN}$$