



# Introducción a la Ciencia de Datos

## Informe de Tarea 3

### Introducción

La tragedia del colapso del [Champlain Towers South Condo](#), un edificio de 12 pisos frente al mar en el suburbio de Surfside en Miami, puso en evidencia la importancia de monitorear y mantener en buen estado los edificios. Incluso un pequeño trozo de fachada al desprenderse, puede provocar heridas fatales en un peatón.

Mientras tanto, en Montevideo, las imponentes edificaciones de estilo Neoclásico, Gótico y Art Nouveau, se roban las miradas curiosas de cruceristas que pasean por la ciudad vieja y la avenida 18 de Julio. En su mayoría, hijos de *La belle époque* entre finales del siglo XIX y principios del siglo XX. Estos edificios de prominentes fachadas, en donde predominan los ornamentos y la mampostería, se construyeron utilizando materiales como hormigón, ladrillo y hierro. Lamentablemente, muchos de ellos se encuentran en un estado de conservación pobre, carentes de mantenimiento y con visibles signos de deterioro.

Por otro lado, la inspección preventiva de una fachada suele implicar, al igual que un mantenimiento o reparación, trabajos en altura con personal calificado y equipo especializado (andamios, cuerdas y arneses). A su vez, el despliegue de operarios suele causar inconvenientes que pueden entorpecer el funcionamiento de un edificio, muchas veces con fines comerciales o turísticos (museos, tiendas, edificios de oficinas, etc.). Por esta razón no es habitual que se hagan inspecciones, detectando los problemas de forma tardía cuando se produce un desprendimiento y el daño es mayor.

En este contexto, el avance en técnicas de visión por computadora y en particular de modelos de aprendizaje profundo como [YOLO](#) o [Mobile Nets](#), que logran resolver tareas de detección de objetos con igual o mejor tasa de aciertos que un ser humano, así como el abaratamiento de drones, ofrecen una nueva perspectiva a este problema combinando software y hardware para resolver el desafío de inspeccionar fachadas de una forma innovadora y de bajo coste.

En este informe nos proponemos analizar [Crack Segmentation Dataset](#), un conjunto de imágenes de grietas en superficies de hormigón, anotadas tanto para detección como segmentación. En base a este dataset nos planteamos experimentar con diferentes técnicas de aprendizaje profundo para visión por computadora vistas en el curso, con el objetivo final de reconocer grietas en fachadas de edificios históricos en Montevideo.

## Búsqueda de un Dataset

El [Crack Segmentation Dataset](#) contiene 4209 imágenes a color de 416 x 416 píxeles de grietas en paredes y calles, con sus respectivas etiquetas. Las etiquetas a su vez incluyen máscaras de segmentación (anotaciones poligonales) y bounding boxes. Por un lado, los bounding boxes son un rectángulo que rodea el objeto a detectar, siendo una anotación no muy precisa pero simple y efectiva para detección de objetos. Por otro lado, las máscaras de segmentación emplean un contorno poligonal exacto alrededor del objeto, lo que detalla su forma específica. Esta anotación se emplea en la segmentación de objetos, en donde a cada píxel se le asigna una categoría específica, generando un área delimitada por un contorno poligonal que incluye al objeto a detectar.

El dataset viene previamente separado en los conjuntos *training*, *testing* y *validation*, con las siguientes cardinalidades respectivamente: 3717, 112 y 200 imágenes.

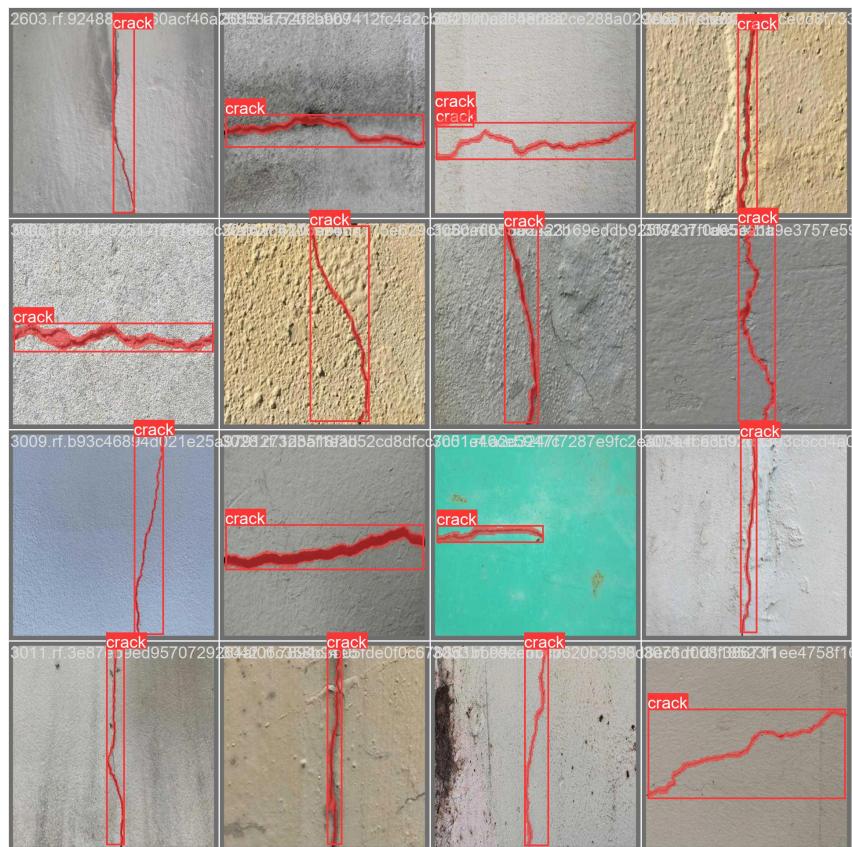


Figura 1: Ejemplos de imágenes en el dataset con sus respectivas anotaciones (imagen extraída de [Crack Segmentation Dataset](#))

En la figura anterior se pueden observar diversos ejemplos de grietas en dataset con sus respectivas anotaciones o etiquetas (bounding boxes y máscaras de segmentación). Se puede ver la diversidad de escenas en las que se encuentran las grietas, cambiando la ubicación el ambiente y las densidades. Observamos además que las grietas se encuentran en el centro de

la imagen, algo que favorece el entrenamiento de modelos de detección y segmentación. Esto a su vez, evidencia cierto grado de preprocesamiento en el dataset.

### Problemas de Calidad

Tras un pequeño relevamiento del dataset encontramos algunos potenciales problemas de calidad. Varias imágenes de entrenamiento parecen ser la misma grieta con una rotación o flip. Esto nos hace suponer que el dataset de entrenamiento se construyó utilizando técnicas de data augmentation y la cantidad de imágenes reales de grietas es menor a la cantidad de imágenes en el dataset. Por otro lado, el dataset no contiene imágenes de superficies sin grietas lo cual podría ser una limitante al momento de entrenar ciertos modelos.

## Definición del Problema

De la sección anterior queda prácticamente determinado el problema a resolver: detectar y segmentar grietas en superficies de hormigón a partir de imágenes. Para esto, pretendemos utilizar el dataset anterior como insumo para experimentar con diferentes modelos de visión por computadora y particularmente modelos de aprendizaje profundo. Para ir de menos a más proponemos experimentar con las siguientes técnicas vistas en el curso:

1. **Redes Convolucionales:** Utilizando la arquitectura de redes convolucionales (CNN) proponemos entrenar un modelo baseline para la detección y segmentación de grietas. En particular podemos combinar diferentes ideas planteadas en trabajos como [Mask R-CNN](#) o [U-Net: Convolutional Networks for Biomedical Image Segmentation](#).
2. **Aprendizaje por Transferencia:** Entrenar modelos profundos requiere de muchos datos y sobre todo de hardware potentes como GPUs en la nube. Sin embargo, modelos entrenados para tareas generales como detección de objetos, se pueden reutilizar en nuevas tareas, haciendo que el reentrenamiento requiera de menos datos y menos tiempo. A esta técnica se la conoce como aprendizaje por transferencia. Para esto proponemos probar haciendo aprendizaje por transferencia con [YOLOv8](#) y/o [MobileNet](#) y comparar los resultados.
3. **Modelos Pre-Entrenados:** Existen algunos trabajos en el área de detección de grietas con los que puede ser interesante compararse. Uno de estos trabajos es [YOLOv8-crack-seg](#), un modelo basado en YOLOv8 entrenado sobre el Crack Segmentation Dataset por el equipo de OpenSistemas. Este trabajo no publica mucha información sobre la evaluación pero el modelo se encuentra disponible en huggin face así que podemos descargarlo y probarlo para obtener un punto de comparación adicional.
4. **Fine-Tuning:** Ya sea usando un modelo pre-entrenado como [YOLOv8-crack-seg](#) o haciendo aprendizaje por transferencia, al pasar de predecir en imágenes del Crack Segmentation Dataset a imágenes reales por ejemplo de edificios históricos de Montevideo, podemos notar una caída drástica de la performance. De hecho, es muy probable que esto suceda ya que las imágenes pueden diferir bastante de las empleadas para entrenamiento. Para solventar este problema tenemos dos opciones al

menos. La primera es volver al entrenamiento incorporando imágenes de edificios de Montevideo en el dataset de entrenamiento. La segunda es hacer fine tuning en los pesos de las últimas capas de un modelo pre-entrenado, utilizando las nuevas imágenes.

El objetivo final es tener un benchmark con los resultados de evaluación empleando diferentes técnicas y así poder seleccionar la que de los mejores resultados. Por otro lado, muchas de estas técnicas tienen hiper parámetros que debemos ajustar para obtener los mejores resultados posibles. Para esto utilizaremos el *validation* dataset.

### ¿Cómo medir el rendimiento del modelo?

Para detección o segmentación de objetos en imágenes suele utilizarse IoU (Intersection-over-Union). Esta es una medida de semejanza en imágenes entre un par: etiqueta y predicción. Para ello usa el área de la intersección entre ambas bounding boxes (etiqueta y predicción) y lo divide por el área de la unión de las mismas. De forma muy similar se puede definir también IoU para máscaras de segmentación. En librerías como PyTorch se puede utilizar [box\\_iou](#) o [JaccardIndex](#).

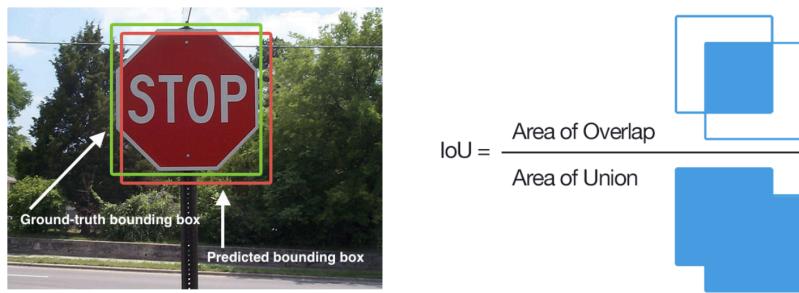


Figura 2 - Ejemplo de IoU para etiqueta y predicción en un cartel de pare (imagen extraída de [Mean Average Precision \(mAP\) Explained](#))

Es fácil ver que IoU es un valor entre 0 y 1: cuanto más cerca de 1 más parecidas son la etiqueta y la predicción y cuanto más cerca de 0 más diferentes. A partir de IoU podemos calcular medidas como Precision y Recall para medir los resultados sobre el Test dataset.

Por otro lado, IoU supone un nuevo parámetro a considerar en la evaluación: el *threshold*. Con un *threshold* bajo estamos maximizando el recall del modelo (dando más detecciones como buenas) pero reducimos la precisión. Por otro lado, un *threshold* alto logra una mayor precisión a costa de un menor recall. Por ello suele utilizarse además mAP (mean-Average-Precision) la cual mide para cada clase el AP (Average-Precision) utilizando diferentes valores de *thresholds* (e.j. 0.4, 0.5, 0.6) y luego calcula la media de AP en todas las clases. Cuanto más cerca de 1 se encuentra el mAP mejores son los resultados.

Al igual que se grafica el valor de la loss function por epoch durante el entrenamiento de un modelo de deep learning, para asegurarse que está convergiendo al óptimo, en este tipo de

problemas se suele graficar el mAP por epoch, procurando que se vaya acercando a 1 a medida que aumentan las epochs de entrenamiento.

### ¿Cómo presentar los resultados?

Una técnica que podría generar resultados interesantes es mostrar la evolución de una grieta a lo largo del tiempo. Para esto precisamos una colección de imágenes de la misma grieta, tomadas a intervalos regulares, por ejemplo cada 15 días y luego calculamos las máscaras utilizando el modelo. Luego con la función [FuncAnimation](#) de Matplotlib podemos construir una animación que muestre la evolución y la eventual propagación de la grieta en el tiempo. Esto tiene el desafío técnico adicional de identificar la misma grieta entre imágenes de  $t$  y  $t + 1$ , así como alinearlas perfectamente, lo cual podría ser un problema técnico en sí mismo. Pese a ello, creemos que es una forma muy efectiva de mostrar el deterioro de una fachada.

También podemos incluir sobre la imagen el valor del área de la máscara para tener información más precisa. A su vez, para una misma fachada en el tiempo podemos mostrar indicadores generales como: cantidad de grietas encontradas en toda la fachada, superficie de área promedio de grietas (área de las máscaras), largo promedio, ancho promedio así como también mostrar máximos, mínimos e incluso el TOP N de grietas más grandes. Estos resultados se pueden mostrar tanto para un muestreo de imágenes (relevamiento de la fachada en  $t$ ) como en función del tiempo para los relevamientos a intervalos regulares de tiempo.

## Siguientes líneas de trabajo

Identificamos las siguientes líneas de trabajo que podrían extender el alcance original:

- El objetivo de este trabajo es aplicar técnicas de computer vision en fachadas de edificios históricos de Montevideo. Para esto precisamos obtener imágenes reales de edificios y así ver resultados en imágenes reales. A su vez, sería bueno etiquetar algunas de estas imágenes para comparar de forma precisa los resultados, calculando Precision, Recall y mAP para un nuevo test dataset.
- El [Crack Segmentation Dataset](#) no contiene imágenes de fachadas sin grietas. Por lo tanto es posible que cualquier modelo de detección tienda a generar falsos positivos en imágenes normales (sin grietas). Para solventar este problema, podemos entrenar un clasificador binario de grietas (imagen tiene o no tiene grieta). Luego usamos el modelo de detección y segmentación de grietas solamente para las imágenes que dieron positivo en el clasificador. Para entrenar este clasificador necesitaríamos incorporar imágenes normales (sin grietas).
- Existen otros defectos en superficies de hormigón, que también nos interesa detectar: agrietamientos (crazing), delaminación (delamination), desconchado (spalling), eflorescencias (efflorescence), entre otros. Para esto, podríamos extender el [Crack Segmentation Dataset](#) combinándolo con otros datasets públicos e incluso elaborando un dataset propio.