UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CARRERA DE INGENIERÍA CIVIL

DETECCIÓN Y MEDICIÓN AUTOMÁTICA DEL DESCASCARAMIENTO SUPERFICIAL DE ESTRUCTURAS DE CONCRETO ARMADO MEDIANTE APRENDIZAJE PROFUNDO

TESIS

Para optar por el título profesional de Ingeniero Civil

AUTOR

Luis Alexander Espinola Diaz (ORCID: 0009-0007-0183-0996)

ASESOR

Luis Alberto Bedriñana Mera (ORCID: 0000-0001-5839-0636)

Lima – Perú

2024

DECLARACIÓN JURADA

Yo, Luis Alberto Bedriñana Mera, identificado con DNI N° 42815379, en mi condición de persona responsable de validar la autenticidad de los trabajos de investigación y tesis de la Universidad de Ingeniería y Tecnología (en adelante, la Universidad), **declaro bajo juramento** lo siguiente:

Que el trabajo de investigación/tesis denominado: “DETECCIÓN Y MEDICIÓN AUTOMÁTICA DEL DESCASCARAMIENTO SUPERFICIAL DE ESTRUCTURAS DE CONCRETO ARMADO MEDIANTE APRENDIZAJE PROFUNDO” ha sido elaborado, bajo mi asesoría, por Luis Alexander Espinola Diaz, identificado con DNI N° 72735957, para obtener el grado académico o título profesional de título profesional de Ingeniero Civil.

El trabajo de investigación/tesis antes mencionado ha sido sometido a los mecanismos de control y sanciones anti plagio previstos en la normativa interna de la Universidad, encontrándose un porcentaje de similitud de 0%.

En fe de lo cual firmo la presente.

En Barranco, el 11 de septiembre de 2024

_____

Firma del asesor

TABLA DE CONTENIDO

Pág.

ÍNDICE DE TABLAS

Pág.

ÍNDICE DE FIGURAS

Pág.

ÍNDICE DE ANEXOS

Pág.

RESUMEN

En Perú, los puentes juegan un papel esencial en la infraestructura vial, pero a menudo reciben poca atención. Esta falta se refleja en la escasa información disponible sobre estos puentes, dado que sólo se ha inspeccionado una cuarta parte de los puentes registrados. Los métodos convencionales para la inspección de puentes requieren personal especializado y una inversión considerable, lo cual resulta inviable para la frecuencia necesaria en el contexto peruano. El objetivo de esta tesis es proponer un sistema que permita acelerar la inspección preliminar de puentes, facilitando la recolección e identificación del descascaramiento en concreto. El método propuesto utiliza redes neuronales convolucionales para la segmentación semántica de imágenes de descascaramiento en superficies de concreto. Para ello se evaluaron dos arquitecturas, YOLOv8 y Unet, entrenados con 200 imágenes de estructuras civiles. Los resultados indican que el modelo YOLOv8 alcanzó un F1-score de 0,68, mientras que el modelo Unet obtuvo un F1-score de 0,89 en la segmentación del descascaramiento. En cuanto a la predicción del diámetro según su F1-score, YOLOv8 tuvo un 52%, y Unet un 61%. Se observó que la predicción empeoraba con el aumento del diámetro y la variabilidad geométrica del daño, lo que se atribuye a la definición imprecisa del diámetro en las guías de inspección y a la plantilla utilizada. El análisis de las imágenes en escala RGB mostró que el daño en la superficie de concreto puede identificarse mediante diferencias en la intensidad de los colores, aunque no se encontró una correlación clara entre la profundidad del daño y los valores de intensidad.

PALABRAS CLAVES: Descascaramiento; Red convolucional; Computer Vision; evaluación de daños; detección de daños; puentes

ABSTRACT

AUTOMATIC SEGMENTATION AND MEASUREMENT OF SURFACE CONCRETE SPALLING FOR STRUCTURAL MEMBERS

In Peru, bridges play an essential role in road infrastructure, but often receive little attention. This neglect is reflected in the limited information available on these bridges, as only a quarter of the registered bridges have been inspected. Conventional methods for bridge inspection require specialized personnel and considerable investment, which is unfeasible for the frequency required in the Peruvian context. The objective of this thesis is to propose a system to accelerate the preliminary inspection of bridges, facilitating the collection and identification of concrete spalling. The proposed method uses convolutional neural networks for the semantic segmentation of images of spalling on concrete surfaces. Two architectures, YOLOv8 and Unet, trained on 200 images of civil structures, were evaluated. The results indicate that the YOLOv8 model achieved an F1-score of 0.68, while the Unet model obtained an F1-score of 0.89 in the spalling segmentation. In terms of diameter prediction according to its F1-score, YOLOv8 had 52%, and Unet 61%. It was observed that prediction worsened with increasing diameter and geometric variability of the defect, which is attributed to the imprecise definition of diameter in the inspection guides and the template used. Analysis of the RGB scale images showed that damage to the concrete surface can be identified by differences in colour intensity, although no clear correlation was found between damage depth and intensity values.

KEYWORDS:

Concrete structures; spalling; convolutional network; Computer Vision;damage assessment; damage detection; bridges

INTRODUCCIÓN

Presentación del tema de investigación

Históricamente los pueblos que se ubicaban en zonas de fácil acceso eran favorecidos por el paso de las caravanas y redes de comercio, entablando una fuerte conexión entre el avance de su infraestructura vial y el desarrollo económico [1]. La extensión de la red vial requirió de la creación de los puentes para cruzar ríos de manera eficiente. Es así como dentro del contexto de la red vial, los puentes representan el eslabón más frágil pero importante de todo el sistema. Por ello es necesario mantenerlos en condiciones óptimas de uso o esto pondría en riesgo la economía del país y las vidas de cientos de sus habitantes.

Para obtener información sobre la condición de la infraestructura vial, se elaboró la guía de inspección de puentes del Ministerio de Transporte y Comunicaciones (MTC) [2], el cual especifica la frecuencia, el tipo y que cosas se deben evaluar de la estructura. Inspeccionar un puente de acuerdo a esta guía puede tomar a un grupo de ingenieros y personal técnico calificado alrededor de 3 días para que planifique y realice manualmente la inspección del puente. Sin embargo, este proceso es completamente dependiente de la experiencia de los inspectores a cargo. Con el paso del tiempo la frecuencia en la que la estructura debe ser inspeccionada aumenta. Considerando que en Estados Unidos el costo para la inspección bianual de sus puentes es 2.7 “billones” (9 ceros) de dólares y ciclos mayores de inspección incrementaría bastante su costo [3].

Por ello la propuesta elaborada en esta tesis busca mejorar las herramientas que involucran al método tradicional. Logrando ahorrar tiempo y dinero, pero más importante asegurando las vidas de los usuarios.

Descripción de la situación problemática

Actualmente, en el Perú se tiene en inventario un total de 3705 puentes, entre definitivos, modulares y artesanales. Sin embargo, solo 1055 de estos han pasado por un proceso de inspección. Se tiene poca o nula información del gran número de puentes construidos, según el portal del MTC de Provias Nacional que se evidencia en la Figura 1 [4]. En el programa de puentes del 2012-2020 reportaron que dentro de los entonces 2227 puentes existentes en el país, un 54% de los puentes son calificados como estructuras artesanales y alrededor de un 34% necesitaban cambiarse por completo o pasar por una urgente rehabilitación en su estructura [5].

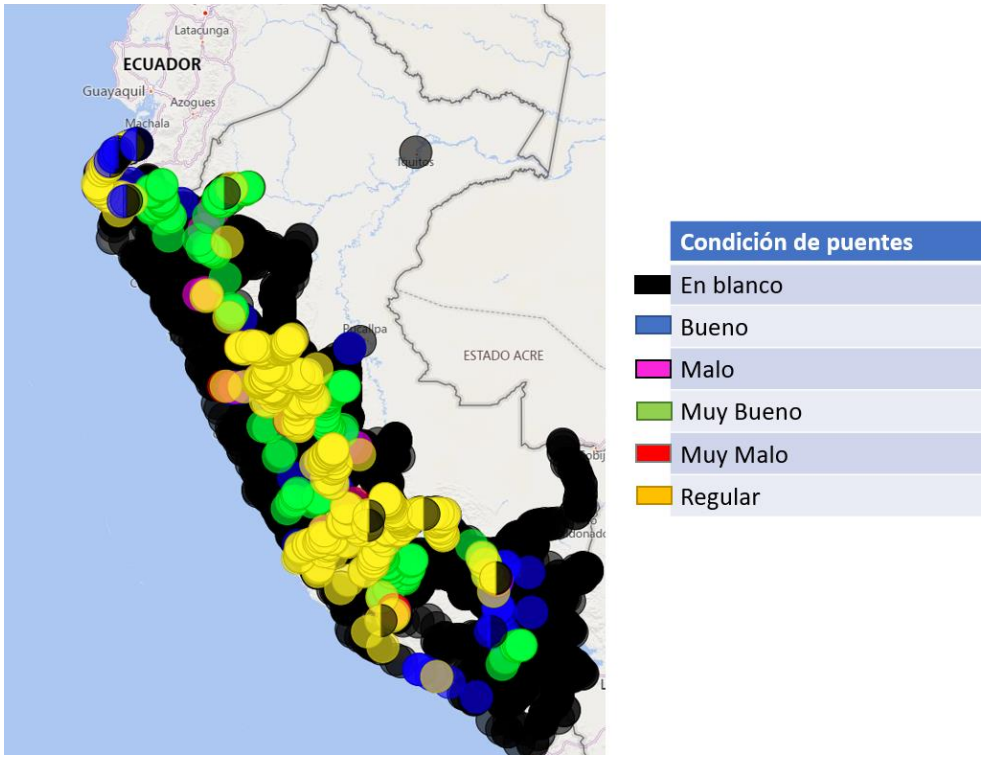


Figura 1: Mapeo del inventario de puentes del Perú (Adaptado de [4])

Además, desde el 2010 el inventario de puentes del Perú incluye en su mayor parte puentes construidos en los 70s y 80s que si bien han contribuido al transporte vial en sus localidades, a la fecha ya no cumplen con la normativa vigente o con los estándares internacionales de serviciabilidad [6]. A la fecha se tienen 2650 puentes (71.50%) que no han sido inspeccionados y del porcentaje que sí se han inspeccionado se tiene tan solo 951 (25.67%) en condición "Regular" o mejor [4]. El manual de inspección de puentes del MTC exige una frecuencia de inspección de una vez al año para puentes en servicio

y de cada tres años para sus elementos sumergidos [2]. Esto quiere decir que no se está cumpliendo los requerimientos de mantenimiento y que se están poniendo en riesgo a los elementos estructurales de los puentes ante sus distintas patologías.

Para los puentes de concreto armado sus patologías pueden ser de características superficiales o internas. Las patologías superficiales más comunes son las grietas y el descascaramiento (*spalling*). El descascaramiento son las áreas del concreto que se han de laminado de su capa superficial, causado por congelamiento, por exposición extensa al fuego, a los efectos expansivos de la reacción *Alkali Silica* o a la corrosión en las barras de refuerzo de acero [7]. La formación de grietas ocurre por múltiples razones, ya sean por malas prácticas constructivas o de diseño, por corrosión, cambios bruscos de temperatura o reacciones químicas [8].

Figura 2: Ejemplo de descascaramiento y grietas [7]

Formulación del problema

Los esfuerzos en el país para inspeccionar y evaluar la condición de los puentes se han realizado de manera tradicional, entonces, se ha vuelto una tarea que carece de eficiencia para representar de manera precisa las dimensiones y formas de todas las patologías de la estructura. Además, está reportado que a un ingeniero inspector le puede demorar 720 minutos para detectar visualmente un total de 500 patologías en el concreto [9]. Dado que la forma tradicional requiere demasiados recursos, la industria ha estado evaluando soluciones más eficientes en los últimos 20 años.

El estudio por Luo et al. [10] realizó una revisión literaria extensa en septiembre del 2023. En este se resaltan métodos de *Computer Vision* para la detección automática de defectos superficiales, medición vibracional e identificación de parámetros vehiculares para la inspección y monitoreo de puentes. A partir de este estudio se resaltan dos puntos

importantes: “Falta de *datasets* que sean *open-source* para defectos superficiales de puentes, requiriendo realizar un trabajo laborioso y lento para preparar los datos.” Y “La mayor parte de las investigaciones actuales utilizan algoritmos avanzados para identificar defectos superficiales de puentes, mientras que la evaluación de su daño requiere mayor investigación.” Con ello se puede analizar la diferencia del estado de detección y evaluación de daño entre grietas y descascaramiento.

Para la detección de grietas en base a imágenes el estudio hecho por Mohan et al. [11] en el 2017 realizó un análisis crítico del estado del arte. En este se detallan múltiples tipos de investigaciones dependiendo del tipo de imagen utilizada, cámara digital, cámara infra roja, cámara ultrasónica, difracción por tiempo de vuelo e imagen laser. En la detección de descascaramiento se resalta al autor German et al. [12] que logró implementar un método para identificar el largo de la región descascarada en la dirección longitudinal y la distancia expuesta en barras de esfuerzo. Este trabajo forma parte de las primeras investigaciones aportando al área de detección de descascaramiento, con lo que el autor elaboró la Figura 3 para resaltar el estado del arte en detección de daño automático en el año 2012.

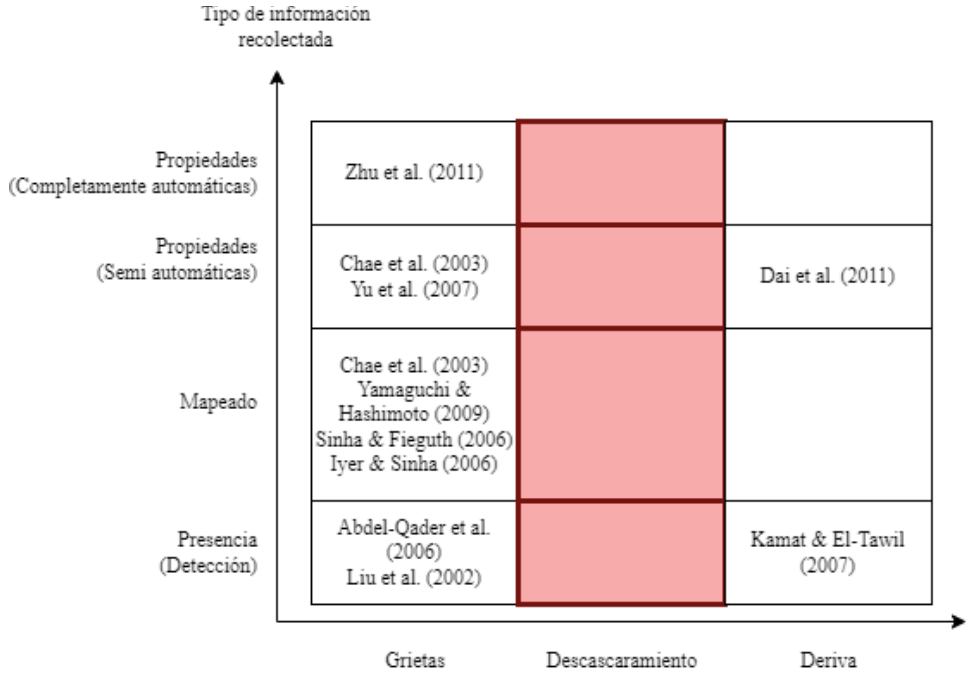


Figura 3: Estado del arte en detección de daño automático en el año 2012 (Adaptado de [12])

En el siguiente capítulo de la tesis se entrará más en detalle en el estado del arte actual, pero es importante resaltar la falta de información que hay en la cuantificación del daño para grietas y descascaramiento en concreto armado. Es por ello que se puede plantear las siguientes preguntas ¿será posible tener un método que haga mediciones cuantitativas del daño en descascaramiento, usando sólo imágenes? Y si es así ¿Cuán preciso puede ser comparado con los métodos tradicionales, que tanto aumentaría su eficiencia?

Objetivos de investigación

El objetivo principal del trabajo de estudio es proponer un sistema que permita la inspección y evaluación automática del descascaramiento utilizando imágenes e implementando Deep Learning en puentes de concreto armado. De esta forma los objetivos específicos de la tesis son:

1. Establecer una base de datos confiable para fines de estudio en descascaramiento para estructuras de concreto
2. Plantear una metodología para la cuantificación automática del daño local en términos del diámetro y la profundidad de descascaramiento
3. Evaluar el desempeño de Unet y YOLO para segmentar el descascaramiento y su capacidad de extraer el daño a través de imágenes
4. Clasificar automáticamente el nivel de daño superficial en estructuras de concreto en base a la segmentación de imágenes

Justificación

Como se ha resaltado antes, en el Perú se desconoce casi en su totalidad la condición de los puentes. No se cuenta con una red conocida de inspección de puentes que permita a las entidades correspondientes o al público general conocer fácilmente la situación real de sus puentes en comparación al resto del país. A pesar que se reconozca que la inspección estructural de los puentes es una herramienta valiosa que asegura la condición óptima de ellos, no se está buscando alternativas locales que nos den una solución más eficiente y económica.

En países como Estados Unidos, su sistema de monitoreo e inspección de todos sus puentes se acerca a un total de 2.7 “billones” (9 ceros) de dólares [3], por lo que siempre están invirtiendo en la investigación de parte de su academia para nuevas formas de volver más eficiente su sistema. Es así como la presente tesis cumplirá este rol en el área de inspección de existentes y futuros del Perú. Además, el presente estudio sería uno de los pioneros en un área donde no hay un consenso claro en la detección y cuantificación del daño para descascaramiento, al cual se le dará el enfoque para estructuras de concreto.

Alcance y limitaciones

En la presente tesis la propuesta de análisis y detección automática de descascaramiento o *spalling* en base a Deep Learning se están evaluando puentes de concreto armado bajo cargas de servicio sin la necesidad de equipos especializados y de alto costo. Por lo tanto, los siguientes puntos no se están considerando en su elaboración:

- Se están omitiendo los puentes elaborados en otros materiales, ya sea de madera, acero, etc.
- No se están considerando otras patologías presentes en los puentes de concreto armado, ya sean grietas, oxidación, etc.
- Solamente se están estudiando los daños superficiales, omitiendo el daño interno estructural.
- No se están realizando análisis o pruebas en campo que involucren *hardware* o instrumentación especial.

CAPÍTULO I

REVISIÓN CRÍTICA DE LA LITERATURA

Como se resaltó en la sección del problema, es limitada la literatura que evalúa el nivel de descascaramiento en base a imágenes; sin embargo, los estudios a continuación representan avances con espacios de mejora que se esperan.

1.1. Detección de patologías en base a imágenes

1.1.1. Descascaramiento

German et al. [12] presentó una de las primeras investigaciones relacionadas con inspección del descascaramiento. Este trabajo consiste de dos etapas, la primera consiste en la detección del daño de descascaramiento y la segunda en poder extraer algunas de sus propiedades, su longitud y su profundidad. Para detectar el descascaramiento, se adaptó la umbralización en base a los conceptos de la entropía de información (detalle de este método se puede leer en la sección 2.4.4.1 del marco teórico) debido a que en cada imagen de descascaramiento la variación local de los pixeles es alta, logrando una distinción entre las regiones dañadas y sanas. A la par elaboraron un algoritmo capaz de detectar el acero de refuerzo longitudinal, con lo que tuvieron en cuenta la gama de colores CMYK y la textura en espiral del acero para utilizar el método de umbralización y de *template matching*. Después extrajeron la longitud vertical donde se ubica el descascaramiento en la imagen y la "profundidad" del descaramiento. Para la longitud vertical obtuvieron un mapa de descascaramiento al que le aplicaron un algoritmo de etiquetado a los pixeles identificados como zona dañada. De esta manera consiguen la región real del descascaramiento y para poder extraer su longitud, primero dibujan un cuadrado alrededor de esta región y la miden. Para identificar la profundidad del descascaramiento emplearon un método más directo, compararon los resultados del algoritmo de *template matching* en el que identificaba si existía o no el acero longitudinal con las guías de inspección de puentes para que un ingeniero estructural determine la profundidad manualmente.

Finalmente, estos resultados son determinados apropiados para ser utilizados en conjunto con un ingeniero estructural especializado en inspección; sin embargo, el método puede sobre estimar la región de descascaramiento detectada, las grietas grandes son consideradas como extensión del descascaramiento. En la Figura 4, se tiene en la izquierda el resultado a partir de su algoritmo y a la derecha el resultado objetivo para futuros trabajos. Los autores consideran necesario aumentar la data con la que trabajaron, fotos que cuenten con áreas de daño y fotos que no la tengan. Además, para poder recuperar una mayor cantidad de sus propiedades, establecen la necesidad de detectar el refuerzo transversal mediante una relación entre la exposición del refuerzo transversal y longitudinal que permita estimar la profundidad de descascaramiento.

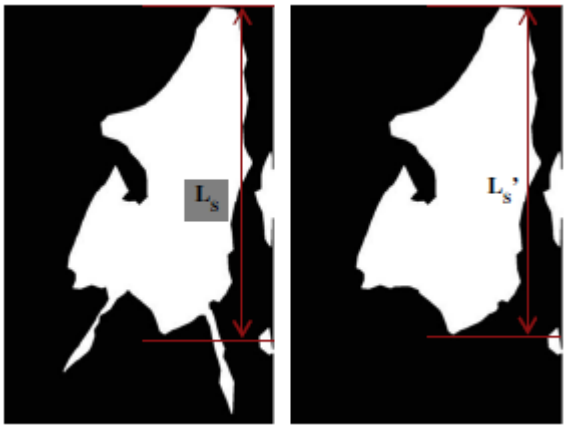


Figura 4. Resultado real y resultado deseado (Tomado de [12])

En Dawood et al. [13] se desarrolló un modelo integrado en base a técnicas de procesamiento de imágenes y machine learning para automatizar la detección constante de descascaramiento y la representación numérica del peligro en redes de subterráneo. Emplearon tres etapas para poder lograrlo, en primer lugar, capturaron imágenes 2D del descascaramiento con una cámara digital. En segundo lugar, las imágenes fueron preprocesadas para remover el ruido y aumentar su resolución. En tercer lugar, emplearon un análisis regresional en la data preprocesada para predecir la profundidad del descascaramiento.

En base a su recolección de data plantean una serie de pautas a considerar para optimizar la adquisición de data. Las imágenes deben ser tomadas de forma ortogonal a la superficie con suficiente superposición para reducir el error de

proyección. Al mismo tiempo, hay que tener en cuenta que existen múltiples formas en la que la intensidad de los pixeles puede ser afectada, por iluminación y por sombras. Para evitar esto utilizaron el flash de la cámara para mantener coherencia de iluminación entre las imágenes, de la misma forma proponen que en caso no se hubiese podido emplear esto se pudieron haber aplicado técnicas como normalización de color y *Multi-Scale Retinex (MSR)* [14].

Su preprocesamiento inicia con la lectura y ajuste de las imágenes, primero separaron las capas de colores de las imágenes RGB en tres distintas, utilizando la capa del color rojo. Lo hicieron para poder obtener una representación en escala de grises que las prepararía para los pasos de mejoramiento de imagen y detección del descascaramiento. Primero realizaron la aplicación de *image blurring* [15] para reducir el ruido de la imagen sin la necesidad de sacrificar información de la región de interés (ROI). A la par, utilizaron el método de difusión anisotrópica [16], que permite realizar operaciones más suaves en la parte interna del ROI sin cruzar bordes entre los pixeles de intensidad fuerte.

Después emplean el método de umbralización para segmentar el descascaramiento del fondo de la imagen, alterando los niveles de gris en la imagen. Por ello le emplean un filtro 3D de *Gaussian Blur* [17] para reducir el ruido y mejorar la resolución de la imagen. Para finalmente utilizar el filtro *color transformer* para filtrar los pixeles de las imágenes y resalten los pixeles que se encuentren cerca de un color en particular, esto revela variaciones en los niveles de intensidad más que variaciones en el color de la imagen.

Una vez definido esto, recuperan la data real de la profundidad del descascaramiento con cinta de medir desde la "cima" hasta el fondo del daño. Esta data la correlacionan con la data de intensidades de pixeles obtenidas en los pasos anteriores, emplearon análisis linear cuadrático y cúbico para determinar la que mejor se correlaciona con la data real. Con todo esto obtuvieron resultados de 91.7% de recall, 94.8% de precisión y 89.3% de accuracy para el algoritmo de detección del descascaramiento y para el método de detección de su profundidad se concluyó el modelo cúbico es el mejor por su $R^2 = 0.9567$ por lo que ambos cumplen su objetivo. Sin embargo, los autores reconocen que las condiciones no

son las comunes en un escenario en campo, comúnmente no se toman imágenes cercanas, ortogonales a la superficie con apropiada superposición o tomadas con un buen flash que asemeje la iluminación a lo largo de la imagen. Además, no tomaron en cuenta la descoloración de la superficie del concreto. Todos estos factores pueden llevar a resultados erróneos en caso no se cumpla con alguno.

En el futuro los autores plantean que es necesario revisar otros métodos de filtro para la data como filtros Notch [18] y otros métodos de detección para el descascaramiento como métodos de agudizamiento de imagen [19]. Pero principalmente tienen planeado incorporar la misma metodología en otras patologías de los elementos de concreto armado.

1.1.2. Grietas

En Fan et al. [20] propusieron un método para evaluar la superficie del pavimento en base a una red de conjunto, detalle de este método se puede leer en la sección 2.4.3 del marco teórico, en una red neuronal convolucional, diseñado para descartar capas puestas en común para mejorar el *accuracy* y obtuvieron las propiedades de longitud y ancho de la grieta a partir de un *crack map*.

Con el objetivo de obtener el *crack map* los autores realizaron tres etapas principales, primero llenaron los huecos pequeños, descartaron los pixeles ruidosos y le colocaron un *label* a los pixeles de grietas. Para llenar los huecos emplearon la operación morfológica de cerrado que se ve en la ecuación (1), donde la cruz y la raya en medio son las operaciones de dilatación y erosión respectivamente y los valores de f y ψ son la imagen agrietada y el elemento estructural respectivamente. Con lo que la primera operación incrementa las regiones de pixeles de grietas y la segunda borra las regiones de borde de las grietas de pixeles. Al descartar los pixeles ruidosos emplearon la operación de abierto que es similar a la operación cerrado, pero en términos adversos como se nota en la ecuación (2) para finalmente colocar un *label* de grietas en las imágenes segmentadas.

$$closing: ((f \oplus \psi) \ominus \psi) \quad (1)$$

$$opening: ((f \ominus \psi) \oplus \psi) \quad (2)$$

El método propuesto por Fan et al. [20] no pudo realizar la detección de grietas de fin a fin, pudiendo solo detectar grietas en imágenes estáticas. A pesar de este limitante, consiguieron resultados de precisión, *recall* y F1 de 0.955, 0.952 y 0.953 respectivamente en la base de datos del CFD. Confirmando que este método es apropiado para poder realizar mediciones de las grietas en longitud y ancho en sus distintos tipos.

Zhang et al. [21] propusieron un modelo basado en Unet (ver sección 2.4.1.4) llamado CrackUnet que aplica una función de *loss* capaz de detectar grietas a nivel de pixel. Iniciaron poniendo a prueba distintas versiones de CrackUnet, cada versión era una cantidad distinta de bloques de convolución en la arquitectura de Unet. Agregaron dos capas de activación de tipo *Rectified Linear Unit* (ReLU, ver sección 2.4.3.4), dos capas de normalización de batches (ver sección 2.4.3.3) y una capa de *dropout* para ignorar ciertos nodos en una capa de forma aleatoria, promoviendo la generalidad del modelo y por ende reducir *overfitting*. Dentro los modelos propuestos, el que obtuvo los mejores resultados fue el que contenía la mayor cantidad de bloques de convolución. Se observó que tendía a pasar por *overfitting* cuando se entrenaba con un número reducido de imágenes. Cuando al modelo se le entrenaba con un mayor número de imágenes, mejor fue su capacidad de aprender los parámetros característicos de las grietas y menor la probabilidad de *overfitting*. De esta forma obtuvo un valor de 0.90 en sus resultados de F1-Score (ver sección 2.4.4.6), a diferencia del mismo modelo con menores bloques de convolución que obtuvo 0.75 en F1-Score.

Flah et al. [22] diseñaron una CNN basado en un grupo de capas, primero con una capa de entrada para captar data de 227x227x1 pixeles, una capa convolucional que reduce sus dimensiones a 1x1x64, una *pooling* layer, una capa de normalización de batches, una capa de activación, una capa de *dropout* y finalmente capas de salida *softmax* para predecir el tipo de clasificación del pixel. El primer tipo de clasificación es sólo para saber si el pixel es o no una grieta, el segundo tipo es para predecir su orientación y el clasificador final abarca ambos clasificadores en uno. Para poder determinar si el pixel es o no una grieta, se implementó la técnica de *thresholding* para *image segmentation*, consiste en ubicar las variaciones en los niveles de grises de una imagen para determinar sus

valores máximos. Estos valores máximos usualmente representan grietas de concreto debido al cambio abrupto con sus regiones aledañas. De esta forma tomaron un solo valor de intensidad de nivel de gris para categorizar a todo valor menor a este como no grieta y todo valor mayor a este como grieta.

Posterior a ello se comenzó con el cálculo de sus dimensiones, en este estudio se midió el largo, el ancho y el ángulo de la grieta. Para determinar la longitud de la grieta trazaron un rectángulo delimitador que la cubra, de esta forma se puede tomar su distancia máxima entre las esquinas opuestas para indicarlo como el largo. El ancho fue determinado primero clasificando las grietas entre grietas verticales y horizontales, para que en el caso que la grieta sea vertical se mida la diferencia entre la altura de borde superior e inferior de cada columna de pixel y si fuese una grieta horizontal se mediría su ancho. En ambos casos el valor máximo sería clasificado como el ancho de la grieta. El ángulo de la grieta se obtuvo de los puntos utilizados para la longitud de la grieta, se trazó una línea horizontal con la cual realizar la fórmula de coseno usando la diferencia en alturas y anchos de cada punto. Con estos métodos implementados se obtuvo porcentajes de error menores al 3% para la longitud de grieta, menor al 10% para el ancho de grieta y menor al 4% para el ángulo de grieta.

CAPÍTULO II

MARCO TEÓRICO

2.1. Patologías presentes en puentes de concreto armado

2.1.1. Grietas

Son fracturas menores lineales presentes en el concreto, que según su tipo pueden extenderse parcial o completamente a lo largo del miembro estructural. Para medir las grietas se mide su ancho, largo y ubicación con el uso de una regla de anchura de grietas en cuatro distintas categorías (*Hairline, Narrow, Medium, Wide*). Es común la aparición de grietas de baja dimensiones (*Hairline*) y no representan un problema mientras que no se encuentre evidencia de oxidación u otro tipo de deterioro.

De la misma forma las grietas se clasifican dependiendo de su origen, pueden ser grietas estructurales o no estructurales. La primera es causada por los esfuerzos generados por las cargas muertas y cargas vivas. Las grietas no estructurales son causadas por efectos como los ambientales. Pero cuando son de mayor tamaño deben ser clasificadas según su origen para poder solucionarlas:

- Grietas estructurales: Flexión y Corte.
- Grietas no estructurales: Temperatura y Contracción.

Las grietas también pueden ser clasificadas según su orientación para entender cómo es que se comporta con las cargas vivas y muertas que interactúa. Por ello se clasifica en las siguientes categorías [23]:

- Grietas transversales.
- Grietas longitudinales.
- Grietas diagonales.
- Patrón de grietas.
- Grietas aleatorias.

2.1.2. Descascaramiento

Se originan de la separación o la pérdida de una porción del recubrimiento del concreto, por lo que se evidencia una fractura paralela a la superficie. Puede ocurrir por un reforzamiento corroído, por fricción en movimientos térmicos y sobre esfuerzo. Usualmente se puede ver el acero de refuerzo en el descascaramiento. El manual de referencia para la inspección de puentes los clasifica de dos formas [23]:

- Pequeños descascaramientos: Profundidad no mayor a 25 mm con un diámetro alrededor de 150 mm.
- Grandes descascaramientos: Profundidad mayor a 25 mm con un diámetro mayor a 150 mm.

2.1.3. Peladuras

Es la perdida gradual y continua de mortero y agregado en la superficie de un área de concreto debido a un daño químico, por lo que se acelera cuando es expuesto a un ambiente malo [23].

Se clasifica según las siguientes categorías:

- Escala menor: Pérdida de mortero de hasta 6 mm de profundidad.
- Escala media: Pérdida de mortero de entre 6 a 13 mm de profundidad.
- Escala grande: Pérdida de mortero de entre 13 a 25 mm de profundidad.
- Escala severa: La pérdida de mortero es mayor a 25 mm y el refuerzo de acero está expuesto.

2.2. Guías de inspección de puentes

Las condiciones con las que se mide cuantitativamente el daño en los puentes de concreto armado es similar a lo largo de distintas fuentes. A continuación se presenta en la Tabla 1 una compilación de distintos criterios para medir el nivel del daño en función del descascaramiento [24]–[26] y en la Tabla 2 está la descripción de condición del puente según los daños encontrados en la inspección acorde al MTC [2].

Tabla 1. Matrices de condición por manual

Manual	Matriz de condición			
	1 (Bueno)	**2 (Leve)**	**3 (Pobre)**	**4 (Severo)**
AASHTO [24]	Ninguno	Desprendimiento < 25 mm profundidad o < 150 mm diámetro	Desprendimiento > 25 mm y > 152 mm de diámetro	La condición supera los límites establecidos en el estado de condición tres (3) y/o justifica una revisión estructural para determinar la resistencia o la capacidad de servicio del elemento o del puente.
FHWA [26]	Ninguno	< 25 mm profundidad o 75 -150 mm en diámetro	>25 mm de profundidad o > 150 mm en diámetro	--
Caltrans Bridge Element Inspection Manual [25]	Ninguno	< 25.4 mm de profundidad o < 152.4 mm de diámetro	> 25 mm de profundidad o > 150 mm en diámetro	La condición de descascaramiento requiere una revisión estructural más detallada para determinar el efecto que el daño tiene en la condición de serviciabilidad de la estructura.

Nota. Matriz de condición para el descascaramiento del 1 al 4 con calidades de bueno hasta severo. Adaptado de [24]–[26].

Tabla 2. Cuadro de descripción de la condición

Calificación	Condición	Descripción de la Condición
0	Muy Bueno	No se observa problemas
1	Bueno	Hay problemas menores. Algunos elementos muestran deterioro sin importancia.
2	Regular	Regular: Los elementos primarios están en buen estado
3	Malo	La pérdida de sección, deterioro o socavación afectan seriamente a los elementos estructurales primarios. Hay posibilidad de fracturas locales, pueden presentarse rajaduras en el concreto o fatigas en el acero.
4	Muy Malo	Avanzado deterioro de los elementos estructurales primarios. − Grietas de fatiga en acero o grietas de corte en el concreto − La socavación compromete el apoyo que debe dar la infraestructura. − Conviene cerrar el puente a menos que este monitoreado.
5	Pésimo	Gran deterioro o pérdida de sección presente en elementos estructurales críticos. − Desplazamientos horizontales o verticales afectan la estabilidad de la estructura − El puente se cierra al tráfico, pero con acciones correctivas se puede restablecer el tránsito de unidades ligeras.

Nota. Cuadro de descripción de la condición de los puentes según el MTC. Tomado de [2].

2.3. Inspección tradicional

2.3.1. Tipos de inspección de puentes

En la Figura 5 se tiene una infografía de los métodos tradicionales de inspección de puentes, este mapeo se basó en la inspección de puentes del MTC [2] y el manual de referencias de inspección de puentes de la Administración Federal de Carreteras Americana (FHWA) [23]. La presente tesis toma lugar en la modernización de los procesos de la Inspección rutinaria y la Inspección de Daño que se describen a continuación.

Figura 5. Tipos de inspección de puentes (Adaptado de [2], [23])

2.3.2. Planificación

En el manual de referencia para inspectores de puentes [23] se detalla que el éxito de una inspección de un puente es altamente dependiente de las labores y el esfuerzo que se le dé a su preparación y planificación. De esta forma se logran realizar trabajos eficientes, ordenados y sistemáticos. La Figura 6 muestra los criterios a tener en cuenta para la planificación de una adecuada inspección en campo de puentes.

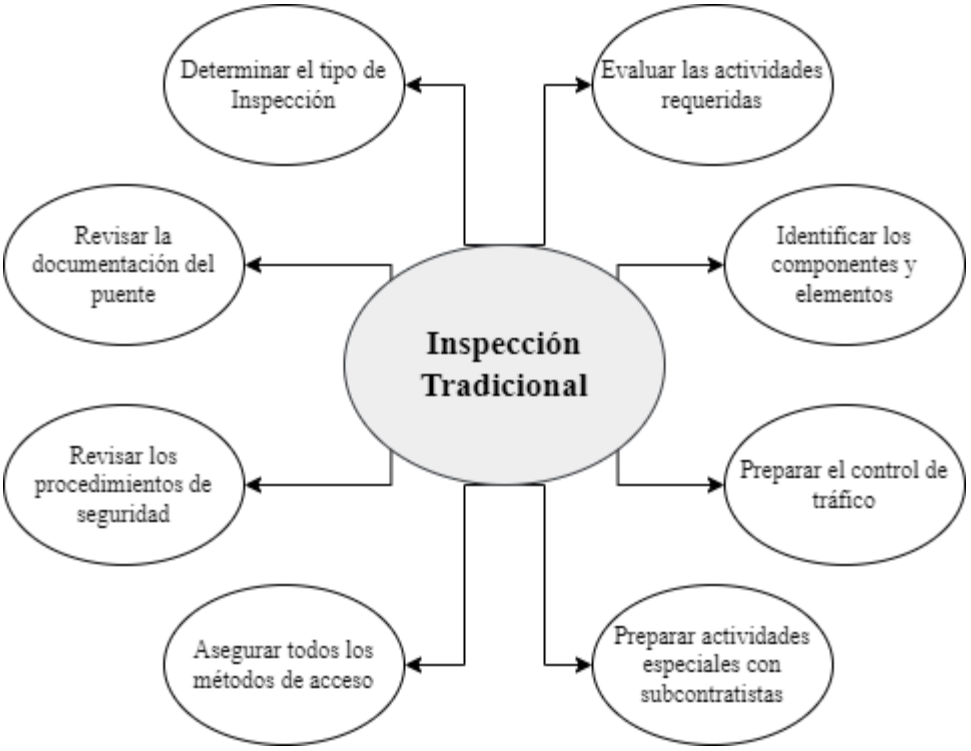


Figura 6. Gráfico de la inspección tradicional (Adaptado de [23])

2.3.3. Equipamiento

El equipamiento requerido para poder realizar las tareas de inspección inicia con los Equipos de Protección Personal (EPPs) generales de obra, pero pueden aumentar según las tareas a realizar. Por ejemplo, en el *Bridge Inspector Reference Manual (BIRM)* [23] detallan el equipamiento de inspección y cuando se debe utilizar para cada caso de medición, documentación, seguridad, etc. Para más información se requerirá revisar las secciones 3.2 y 3.4 del *Bridge Inspector Reference Manual (BIRM)* para el equipamiento de inspección y seguridad correspondientemente. En la Figura 7 se tiene un ejemplo de los EPPs a usar durante la inspección de un puente que esté construido sobre un río, el inspector tendrá que contar con un chaleco salvavidas, un chaleco reflector, casco y un arnés.

Figura 7. Inspector con casco, chaleco salvavidas, chaleco reflector y arnés (Tomado de [23])

2.4. Fundamentos de Deep Learning

2.4.1. Redes Neuronales Convolucionales (CNNs)

Las redes neuronales convolucionales (CNNs) están diseñadas para procesar arreglos estructurados de datos, como es el caso de imágenes. Contiene muchas capas para procesar los datos y cada capa es capaz de procesar formas más sofisticadas. Uno puede resumir los CNNs en etapas de capas convolucionales, seguidos de funciones de activación y de una capa de *pooling* (*downscaling*) que se repiten múltiples veces para obtener más y más detalles de la imagen. Esto lo vuelve una red muy buena para encontrar patrones complejos en una imagen, ya sea líneas, gradientes, círculos, etc [27]. En la Figura 8 se puede notar la arquitectura de la red neuronal de YOLO, compuesta de capas convolucionales y completamente conectadas. En general las redes siguen este tipo de estructura con su propio tipo de configuración para cumplir la tarea esperada [28]. Además las redes neuronales son capaces de operar directamente de una imagen sin preprocesamiento [27].

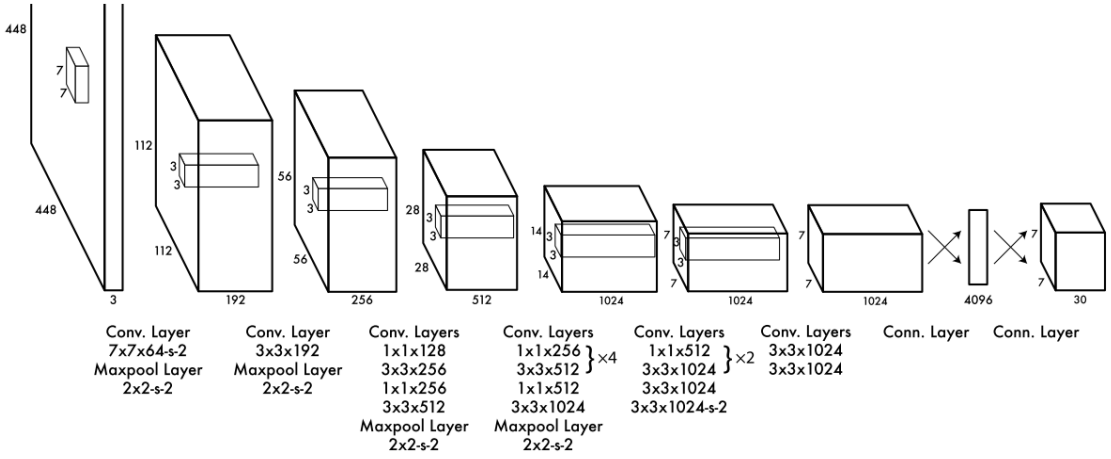


Figura 8. Arquitectura YOLO (Tomado de [28])

2.4.1.1. Capas convolucionales

Expresado de forma matemática, las capas convolucionales son matrices de pesos llamados kernels convolucionales, que escanean la imagen y buscan patrones que se asemejen al patrón que tenga el kernel. Cuando lo encuentra, el kernel tiene como output un valor positivo y cuando no lo encuentra el output es 0 o un valor cercano a él. Un ejemplo de las capas convolucionales se ve en la arquitectura YOLO de la Figura 8 [28].

2.4.1.2. Funciones de activación

Después que una imagen pasa por una capa convolucional necesita pasar por una función de activación, las más comunes son la función sigmoid y la función de unidad rectificada linear (ReLU), que se pueden visualizar en la ecuación (3) y la ecuación (4) respectivamente. El objetivo de utilizar estas funciones es el de agregar la no linealidad al CNN, que lleva a la eliminación del ruido de los detalles no requeridos de la figura y de mejorar el contraste de estos detalles con el fondo.

$$S(x) = \frac{1}{1+e^{-x}} \tag{3}$$

$$f(x) = \max(0, x) \tag{4}$$

2.4.1.3. Arquitectura YOLO

YOLO fue elaborado en el 2016 como una nueva propuesta para la detección de objetos, reutilizando clasificadores para realizar la detección como un problema de regresión entre cuadros delimitadores y probabilidades de clases. Conocido por sus siglas de *You Only Look Once*, es una arquitectura neuronal que se caracteriza por ser extremadamente rápida gracias al realizar la detección como un problema de regresión en toda la imagen a la vez. En la Figura 8, se mostró su arquitectura basada en 24 capas convolucionales, seguidas de 2 capas completamente conectadas para finalmente duplicar la resolución de la imagen para su detección [28].

2.4.1.4. Arquitectura Unet

La arquitectura neuronal de Unet ha sido desarrollada con el objetivo de realizar segmentación de imágenes. Su estructura está detallada en la Figura 9 y consiste en dos caminos, el primero es representado por *encoder blocks* para realizar el análisis y rescatar la información de clasificación y el segundo camino representa los *decoder blocks* con los que sintetiza la información recolectada en base a *up-convolutions* y de concatenaciones con las convoluciones de los *encoder blocks* a su mismo nivel. Esto representa un nivel de información por pixel más potente que los tradicionales de etiquetado por objeto, debido a esto también se le puede llamar *pixel based* [29].

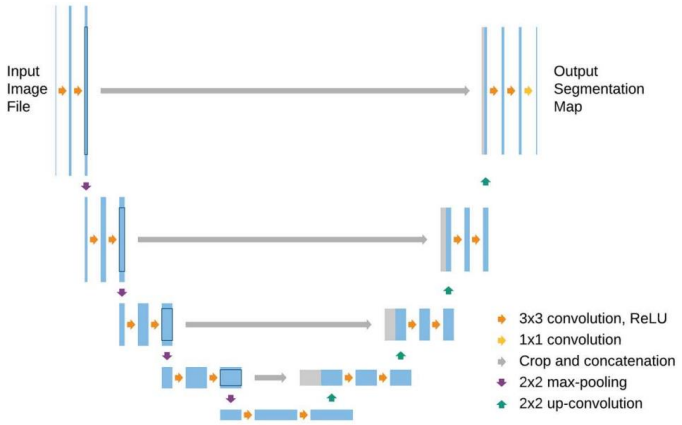


Figura 9. Arquitectura Unet (Tomado de [29])

2.4.2. Método de aprendizaje ensamblado

Las redes ensambladas (*ensemble networks*) son algoritmos que generan múltiples modelos para cumplir la tarea en cuestión. La red contiene *k* modelos *m1, ..., mk* con los que tienen un output de probabilidades $p(x = y_i|m_1)$ con un output x que se clasifica como y. La predicción se puede representar en la ecuación (5):

$$p(x = y_i|m_1, \dots, m_k) = \frac{1}{k}\sum_{j=1}^{k} p(x = y_i|m_j) \quad (5)$$

2.4.3. Estrategias de optimización

2.4.3.1. Métodos de optimización estocástica (SGD)

Es utilizado para obtener mejoras en el entrenamiento de un modelo, minimizando el *loss*. Con SGD el entrenamiento procede en pasos y cada uno de ellos se realiza en mini *batches* usado para aproximar la gradiente de la función *loss* en relación a los hiper parámetros del algoritmo. La ventaja está en la reducción en la computación que se necesita para efectivamente entrenar un modelo; sin embargo, las capas necesitan adaptarse continuamente a las nuevas distribuciones. Esto lo vuelve vulnerable al *covariate shift* que ocurre cuando la distribución del input del aprendizaje de un modelo cambia [30].

2.4.3.2. Métodos de optimización estocástica (Adam)

Similar al método SGD, el método Adam [31] (*adaptive moment estimation)* es un método de optimización estocástica que busca solucionar los problemas de escalares de parametrización mediante maximización o minimización. La diferencia está en que este método se basa en combinar las ventajas de otros dos métodos populares; AdaGrad [32], que trabaja bien con gradientes dispersas, y RMSProp [33] que trabaja bien con configuraciones no estacionares y en línea.

2.4.3.3. Normalización de Batches

Es un mecanismo para acelerar el entrenamiento en Deep Learning, realiza un paso de normalización para corregir los promedios y variaciones en los inputs, adicionalmente reduce la dependencia de la gradiente en la escala de los hiper parámetros de sus valores iniciales. Por lo que previene que pequeños cambios en los hiper parámetros se amplifiquen en cambios subóptimos durante la activación de las gradientes, es decir, evita que la red se atasque en modos saturados mediate el uso de las no linealidades saturantes. En la Figura 10, está el algoritmo para poder implementar esta herramienta de optimización elaborada por Ioffe [30].

Input: Red N con parámetros entrenables Θ; subconjunto de activaciones $\{x^{(k)}\}_{k=1}^{K}$

Output: Red normalizada por *batches* para interferencias N_{BN}^{inf}

1: $N_{\mathrm{BN}}^{tr} <- N$ // Entrenando de red BN
2: **para:** $k = 1 \ldots K$ **hacer**
3: Agregar transformación $y^{(k)} = BN_{\gamma^{(k)}\beta^{(k)}}(x^{(k)})$ a N_{BN}^{tr} (Alg. 1)
4: Modificar cada capa en N_{BN}^{tr} con input $x^{(k)}$ para tomar $y^{(k)}$
5: **terminar para:**
6: Entrenar N_{BN}^{tr} para optimizar los parámetros $\{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: N_{BN}^{inf} <-- N_{BN}^{tr} // Interferencia de la red BN con parámetros congelados
8: **para** $k = 1 \ldots K$ **hacer**
9: // Para Claridad, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathrm{B}}^{(k)}$, etc.
10: Proceso de entrenamiento múltiple en mini-batches B, cada uno de tamaño m, y promediar sobre ellos $\mathrm{E}[x] <- \mathrm{E}_{\mathrm{B}}[\mu_{\mathrm{B}}]$

$$\mathrm{Var}[x] <- \frac{m}{m-1}\,\mathrm{E}_{\mathrm{B}}[\sigma_{\mathrm{B}}^2]$$

11: En N_{BN}^{inf} , reemplazar la transformada $y = \mathrm{BN}_{\gamma,\beta}(x)$ con

$$y = \frac{\gamma}{\sqrt{Var[x] + e}} * x + \left(\beta - \frac{\gamma \mathrm{E}[x]}{\sqrt{Var[x] + e}}\right)$$

12: terminar para

Figura 10. Algoritmo para entrenar una red de normalización en *batches*. Adaptado de [30].

2.4.3.4. Rectified Linear Units (ReLU)

Es una función de activación utilizada en CNNs usada como una función de clasificación para la función Softmax. Su función es limitar los valores de la penúltima capa de convolución a que siempre sean mayores o iguales a 0. De esta forma se promueve la etapa de entrenamiento en modelos de Deep Learning [34].

2.4.4. Medidores de desempeño

2.4.4.1. Coeficiente de determinación

El coeficiente de determinación mide lo bien que un modelo de regresión se ajusta a los datos reales. También conocido como el R^2, este se mide en una escala de 0 a 1; donde 1 indica un modelo que predice de forma perfecta los valores [35]. En la ecuación 6 se tiene la fórmula para obtener este coeficiente, dividiendo el cuadrado de la diferencia entre el valor real y el valor predicho con el cuadrado del valor real con el promedio de las predicciones.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\overline{y_i} - y_i)^2}{\sum_{i=1}^{n}(\overline{y_i} - y_{mean})^2} \quad (6)$$

2.4.4.2. Matriz de confusión

La matriz de confusión es una matriz de NxN usada para evaluar el desempeño de un modelo de clasificación, donde N es el número de clases objetivo. En otras palabras, es una tabla resumen de las predicciones correctas e incorrectas producidas por un modelo de clasificación en tareas binarias [36].

En la Figura 11, se tiene un ejemplo gráfico de una matriz de confusión, el lado de las filas representa cada clase predicha por el modelo y por el lado de las columnas representa cada clase según sus valores reales. Por ello, se ver como hay 50 objetos que representan dos clases en términos binarios, la clase 1 es positivo y la clase 0 negativo. A raíz de este ejemplo se definen las siguientes clasificaciones de predicción según los resultados de la matriz de confusión.

		Reales	
		1	0
Predichas	1	10	50
	0	40	0

Figura 11. Matriz de Confusión

2.4.4.2.1. Clasificación de predicción

a) Verdaderos Positivos (TP)

Volviendo a la Figura 11, los verdaderos positivos serían representados en el recuadro que une la predicción de los 10 objetos de clase 1 con la fila que representa la clase 1.

b) Falsos Positivos (FP)

Aquí es donde el modelo clasificó como clase 1 50 objetos que realmente son clase 0. Es decir, su clasificación debió ser negativa pero el modelo lo predijo como positivo.

c) Verdaderos Negativos (TN)

En el ejemplo de la Figura 11, el modelo representó 0 objetos correctamente como negativos. Es decir, el valor debió ser negativo y el modelo lo predijo como negativo.

d) Falsos negativos (FN)

En la Figura 11 el modelo clasificó a 40 objetos como negativo de clase 0, cuando en realidad deberían ser positivos de clase 1. El valor debió ser asignado como positivo pero el modelo lo predijo como negativo.

2.4.4.3. Precisión

De la ecuación 7, se interpreta que de todas las clases predichas como verdaderas, cuántas son en realidad positivas [37].

$$Precisión = \frac{TP}{TP+FP} \quad (7)$$

2.4.4.4. Exhaustividad (Recall)

De la ecuación 8, se interpreta que de todos los casos positivos, cuales son los predichos correctamente [37].

$$Recall = \frac{TP}{TP+FN} \tag{8}$$

2.4.4.5. Exactitud (Accuracy)

Sirve para medir que tan frecuente el modelo predice correctamente positivo [37].

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{9}$$

2.4.4.6. F1-Score

Sirve para medir el valor de exhaustividad y precisión al mismo tiempo. Es usado principalmente para comparar dos clasificadores y determinar cuál produce mejores resultados [37].

$$F - measure = \frac{2*Recall*Precisión}{Recall+Precisión} \tag{10}$$

CAPÍTULO III

MARCO METODOLÓGICO

Esta tesis se caracteriza por ser una investigación cuantitativa de carácter experimental, por lo que se ha divido en base a cuatro módulos que pueden visualizarse en el diagrama de flujo de la Figura 12. Primero se llevó a cabo un proceso de recolección de data basado en las guías de inspección de puentes de la sección 2.2 y se le identifica como el módulo uno. Después se cuantificó el daño de la data recolectada en base a las matrices de condición de la Tabla 1 y a la par se elaboraron algoritmos capaces de detectar el daño a través de imágenes. Estos dos últimos módulos se desarrollaron de forma paralela para agilizar su proceso.

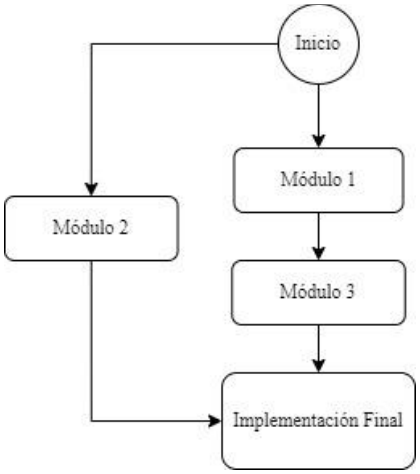


Figura 12. Metodología de trabajo

3.1. Módulo 1: Recolección de data

En este módulo se estableció el protocolo para la recolección de datos en campo que se puede resumir en la Figura 14. Los materiales que se utilizaron son un aro de luz que iluminó el área con daño, un celular con el que se tomaron las fotos y un cuadrado de 4 cm^2 como plantilla. Toda la data que se recolectó cuenta con tres elementos, una fotografía del descascaramiento, una medición de su diámetro y una medición de su profundidad. Con estos parámetros se analizaron las imágenes para verificar si su uso es apto o no para entrenar los modelos indicados en el módulo 3. Una vez validada se empezó con el etiquetado de las imágenes con los programas de LabelStudio [38] y Roboflow [39]. Finalmente se presentó la data validada en un histograma para poder resumir su contenido. A continuación, se describe los pasos que se tomaron para la

recolección de imágenes en campo. En el Anexo 1 también se puede observar una secuencia gráfica que ilustra este protocolo de recolección de datos en campo.

- Debido a que es probable que se encuentre el descascaramiento en zonas oscuras, se iluminó el área dañada con un aro de luz de 10'' con una potencia de 12.5W. En caso la iluminación natural haya sido suficiente, no se empleó el aro de luz.
- Una vez ubicada la zona dañada, se limpió la zona de interés para poder facilitar pegar la plantilla (cuadrado) que sirvió de escala.
- En la Figura 13 se muestra un ejemplo de una de las fotos que se tomaron, para ello se cubrió toda la región dañada incluyendo una cantidad generosa del fondo que lo acompaña. La distancia a la que se tomaron las fotos fue de 500 mm aproximadamente de la patología y se tomaron lo más ortogonales posibles.
- Una vez tomada la foto se midió el diámetro promedio con una wincha o cinta, trazando un círculo que cubra lo que más se pueda de la región dañada. En caso un círculo no sea suficiente para cubrir todo el daño, se trazaron múltiples círculos y se reportó el promedio de estos círculos. También se midió su profundidad, midiendo desde el punto más profundo del descascaramiento hasta una altura cercana a un punto "0" de la superficie en la que se encuentra el daño.
- Siguiendo estos pasos se removió la escala y se repitió.

Figura 13. Toma de foto en la noche de descascaramiento

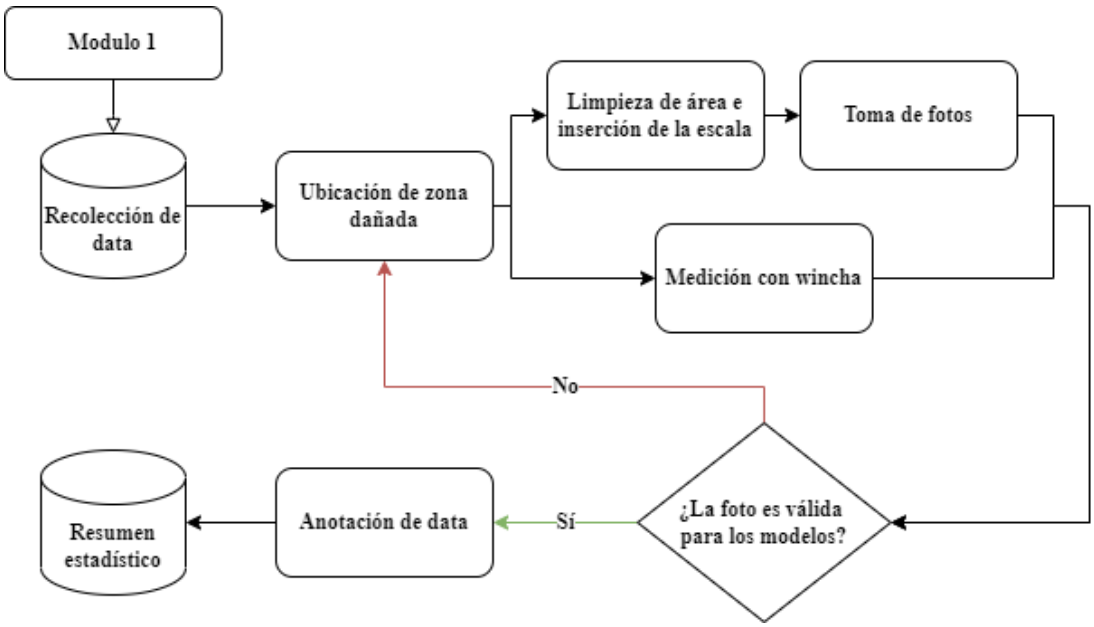


Figura 14. Protocolo para la recolección de datos en campo

3.2. Módulo 2: Cuantificación local del daño

En la Figura 15 se tiene el esquema del módulo 2, el cual describe los indicadores numéricos de daño para el descascaramiento, esto se realizó tomando como referencia las matrices de daño establecidas en la Tabla 1 del marco teórico. Los valores numéricos para la profundidad y el diámetro del descascaramiento se le da el nombre de índice de daño (ID). Este índice sirvió como la métrica principal para comparar la información recolectada en campo con la información predicha en los modelos presentados en el módulo 3.

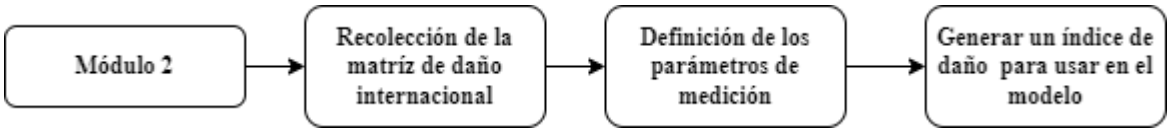


Figura 15. Módulo 2 – Cuantificación local del daño

3.3. Módulo 3: Segmentación y medición del daño

Este módulo entra en desarrollo después de finalizar la etapa de recolección de datos, por lo que se pasó a una etapa de aumentación de data con el uso de la librería de Python Imgaug [40]. Esta librería dio la facilidad de utilizar múltiples filtros que modifiquen la data original y brinden mayor variedad a la etapa de entrenamiento y validación. El primero es un filtro de traslación que mueve la posición relativa de la imagen unos cuantos pixeles en sus coordenadas x e y. El segundo es un filtro de rotación

que rota la posición relativa de la imagen. El tercero es un filtro que genera borrosidad en los pixeles, similar a una cámara mal enfocada. El cuarto es un filtro que remueve algunos pixeles, dejándolos como vacíos o negros. El quinto es un filtro que voltea en su eje vertical. Toda la data aumentada junto con la original fue partida en 2 partes, la primera consiste en la data utilizada para el entrenamiento del modelo que consiste de un 80% del total de imágenes, la segunda es un 20% del total de imágenes para la validación de los resultados.

Una vez dividida la data se entrenó un modelo en Unet y en YOLO que fuera capaz de detectar de forma precisa los pixeles en donde se encuentren la plantilla de la Figura 16. Teniendo en cuenta que sus dimensiones reales son de 2 cm x 2 cm, se relacionó sus dimensiones en unidad pixel vs cm, obteniendo una escala dinámica en cada una de las imágenes recolectadas.

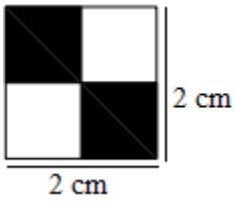


Figura 16. Plantilla cuadrada

El diagrama de flujo de la Figura 17 describe los pasos desarrollados para el modelo que detecte el descascaramiento y cuantifique su nivel de daño de acuerdo a la Tabla 3. Para la segmentación del descascaramiento se puso a prueba dos métodos, el primero hace uso de YOLO (punto 2.4.1.3 del marco teórico) y el segundo hace uso de Unet (punto 2.4.1.4 del marco teórico). YOLO habilita la capacidad de realizar *Object Detection* e *Image Segmentation* a la par, basándose en cuadros delimitadores en la imagen original y las probabilidades de clase entre cada cuadro. Por ello se utilizó el cuadro delimitador de mayor confiabilidad para poder delimitar donde el daño realmente está.

Para Unet se agregaron capas de Normalización de Batches (punto 2.4.3.3 del marco teórico) en cada bloque de convolución para acelerar el entrenamiento. Esta arquitectura es más conocida por su alto rendimiento en tareas de *Image Segmentation*, al costo de una mayor exigencia computacional. Toma como *input* máscaras donde indica la ubicación de las áreas de interés y su *output* se basa en máscaras en formato PNG con los pixeles categorizados como descascaramiento.

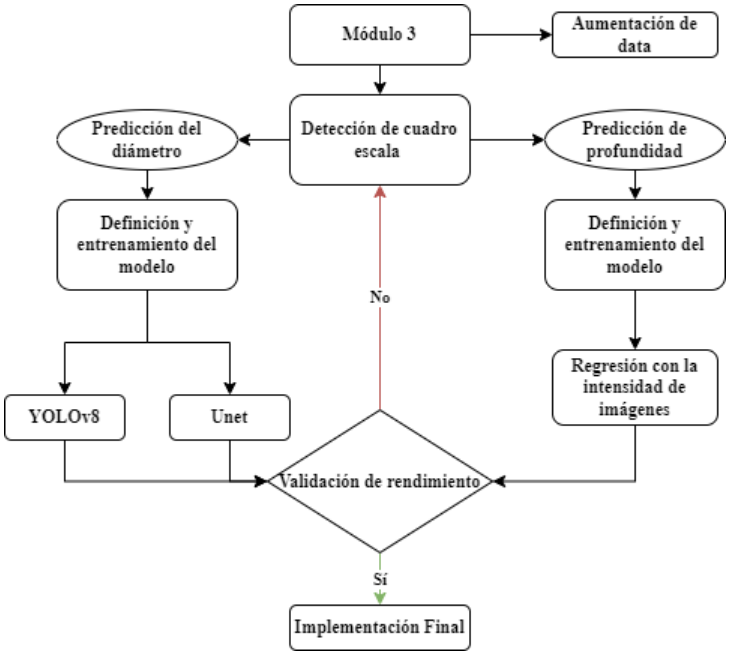


Figura 17. Segmentación y medición del daño

En ambos casos, los pixeles categorizados mediante *Image Segmentation* fueron representados como el área de un círculo. Esta área utilizó la escala calculada previamente para tener un estimado real de cm2 con la que se calculó el diámetro con la fórmula del diámetro en un círculo. En la Figura 18 se puede ver un diagrama del proceso. De esta forma ambos modelos fueron comparados según su capacidad de segmentación del descascaramiento según sus métricas de *F1-score y accuracy* y su correspondiente predicción de diámetro en comparación con la data real recolectada en el módulo 1.

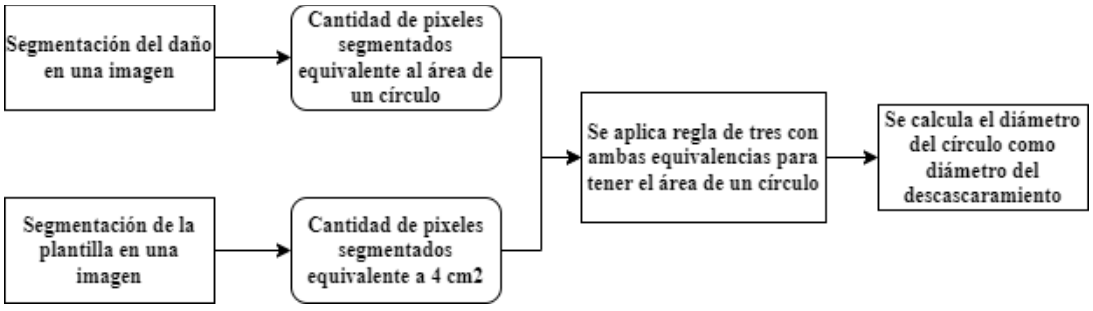


Figura 18. Proceso de cálculo del diámetro

Para la estimación de la profundidad máxima de descascaramiento se está revisando el estudio realizado por Dawood et al. [13] donde se realizó un método de regresión lineal en base a un perfil de data real de la profundidad y un perfil con los valores de intensidad con los que se asocian estos pixeles. De esta forma es que en esta tesis primero se analizó la relación de sus pixeles en la escala RGB con el área segmentada

representada por el daño. De tal forma que las zonas dañadas y segmentadas representen valores distintos a la zona no dañada. Para realizar esto se tomó la media geométrica de los pixeles en cada fila de la imagen según su escala de grises. La fila de pixeles donde se ubique el valor calculado de intensidad más pequeño será la ubicación del punto más profundo de la imagen. De esta forma se obtuvo el perfil de la imagen según sus valores de intensidad en una escala de grises en base a sus niveles de luminosidad. Además, se creó una imagen de contorno de las intensidades de la imagen para analizar su relación con el producto de la imagen segmentada con el daño. Segundo, se plantea una correlación entre la diferencia de los valores de intensidad de la imagen máximos y mínimos. Se compararán los resultados para su escala de grises y cada uno de sus canales en la escala RGB, con los valores reales medidos. Debido a que puede haber varias imágenes que cuentan con la misma profundidad, se promedió su valor de intensidad para evitar conflictos en el modelo de regresión.

Una vez culminada la selección de los métodos de detección y extracción de propiedades. Se entrenaron los modelos con los que se evaluaron distintas métricas de sus resultados de entrenamiento de *accuracy*, *loss y F1-score*, también tomaron en cuenta los resultados de sus predicciones. En algunos casos los resultados de entrenamiento se reiteraron con distintos hiper parámetros de razón de entrenamiento, número de *epochs*, incremento de data, entre otros para encontrar el mejor resultado.

3.4. Validación e implementación del modelo final

Finalmente, con el objetivo de validar ambas metodologías, se utilizó como punto de comparación el set de imágenes recolectados y sus propiedades de diámetro y profundidad, a ellos se les asignó un nivel de daño según la tabla elaborada en el Módulo 2. A los resultados predichos por los modelos de la sección 3.3 se validaron en base a un porcentaje de error aceptable a ±25% en comparación con el valor medido, posterior a ello se graficaron sus valores de tal forma que se tenga en el eje "y" su valor predicho y en el eje x su valor medido. La diagonal de esta gráfica nos dio una predicción perfecta, por lo que al calcular el valor de R^2 se obtuvo que tanto varía la predicción hecha de una predicción real.

Además, los modelos finales escogidos se compilaron en un solo programa con el cuál se pueda procesar data de descascaramiento de forma dinámica. La Figura 18 detalla el flujo que sigue esta interfaz. Primero se da la opción al usuario a escoger la carpeta donde se encuentren las fotos que sean de 192x256 pixeles y contengan la plantilla dentro de la foto, después se muestran las fotos y se empiezan a procesar con los modelos de Unet y YOLO. Una vez procesadas se muestra la foto original, la foto segmentada por Unet, la foto procesada por YOLO y debajo de estas dos últimas su diámetro predicho y su nivel de daño según la tabla del módulo 2.

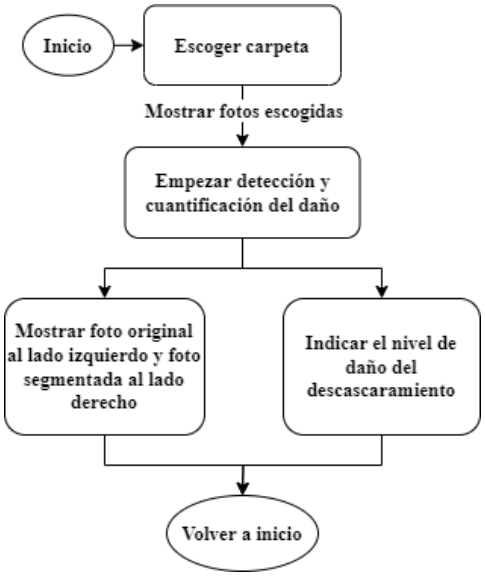


Figura 19. Implementación Final

CAPÍTULO IV

RESULTADOS

4.1. *Dataset* y cuantificación del daño

Con las matrices de condición de daño de la Tabla 1, se elaboró los índices de daño de la Tabla 3, de esta forma se puede categorizar cada imagen recolectada en el módulo 1 según un nivel de daño para su diámetro y profundidad. Dicho esto, en la Tabla 4 se tiene el nivel de daño según la profundidad y diámetro de los descascaramientos de todas las imágenes recolectadas. La Tabla 4 muestra que la mayoría de imágenes recolectadas corresponden a un nivel de daño leve (69.00% para profundidad y 45.50% para el diámetro), seguido de un menor porcentaje de imágenes con daño moderado o severo. Cabe mencionar que los valores para diámetro son mayores a la cantidad total de imágenes recolectadas, esto se debe a que en algunos casos había presentes múltiples descascaramientos en una sola imagen.

Tabla 3. Índices de daño

	1 (Despreciable	2 (Leve)	3 (Moderado)	4 (Severo)
Profundidad	0 mm	< 25 mm	> 25 mm	> 30 mm
Diámetro		< 150 mm	> 150 mm	> 250 mm

Tabla 4. Nivel de daño de data recolectada

	Profundidad	Diámetro
Leve	138	91
Moderado	39	74
Severo	23	35

Además, con el fin de elaborar un modelo lo más general posible, se recolectó data de múltiples elementos estructurales (Vigas, losas, columnas y parapetos), bajo múltiples niveles de iluminación. De esta forma se recolectaron 200 imágenes, tomadas en la ciudad de Lima; sin embargo, se evidenció que en un gran número de casos de descascaramiento, la geometría del daño requirió la medición de más de un diámetro. El problema se acentúa debido a que las guías internacionales de inspección de puentes no dan una definición clara sobre cuál es el diámetro del descascaramiento, en este estudio se tuvo que promediar entre un diámetro de valor mayor y uno de valor menor para reportar el promedio como el diámetro del descascaramiento. Un caso como este se muestra en la Figura 20 donde se hicieron dos mediciones con la intención de cubrir la

mayor parte del área dañada. Con esto en mente, se categorizaron las imágenes según este diámetro y profundidad medido y la Figura 21 muestra la distribución de los datos medidos.

Figura 20. Ejemplo de medición en descascaramiento

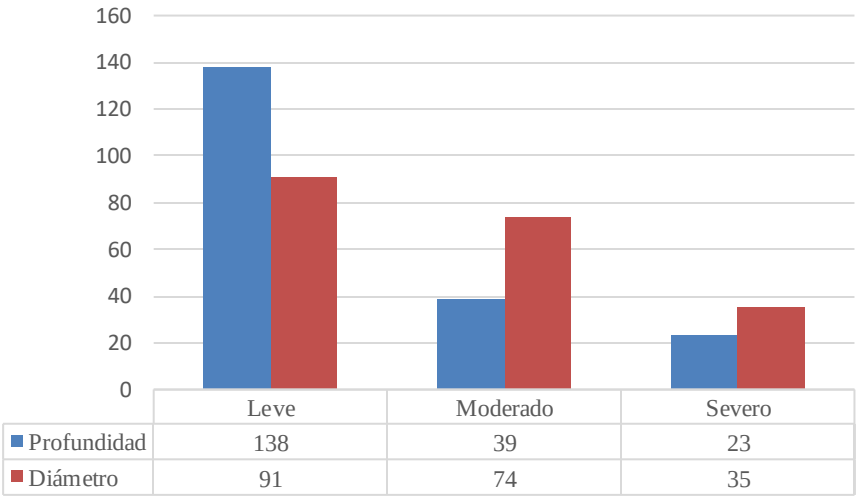


Figura 21. Nivel de daño según su profundidad

En las Figuras 22 y 23 se tiene dos descascaramientos de profundidades de 25 mm y 5 mm, con diámetros de 300 mm recolectados en losa y viga, respectivamente. Ambas fotos son ejemplos de los casos en el que se tuvo que agregar iluminación con el aro de 10'' descrito en metodología. Todas las fotos se tomaron con la cámara trasera de un celular Samsung Galaxy A21s (48 megapíxeles f/2.2) con una alta resolución de 3000 x 4000 pixeles. Además, estas dos imágenes son ejemplos claros de la irregularidad de la forma del descascaramiento encontrado en la mayoría de imágenes.

Figura 22. Descascaramiento en una losa

Figura 23. Descascaramiento en una viga

4.2. Segmentación y medición del daño

Todos los trabajos de procesamiento y entrenamiento de modelos se realizaron en una Workstation (Intel ® Core (™) i5-10400 CPU @2.90 GHz 16 GB RAM, sistema operativo de 64-bits). Los modelos de *Image Segmentation* fueron realizados con las arquitecturas de Unet y YOLOv8 (versión YOLOv8.0.28). La arquitectura Unet requiere el *downsizing* de sus imágenes según la cantidad de niveles de bloques de convolución más el puente, en cada nivel se reduce su tamaño en un factor de 2. En este estudio se mantuvo su cantidad original de 4 para el número de niveles de bloques de convolución. Esto quiere decir que el tamaño de las imágenes en x e y necesita ser divisible entre 32. Por ello, se escogieron las dimensiones de 256x192 pixeles para las imágenes pues cumplían con este requisito y al ser de tan reducido tamaño, también evita el consumo excesivo de memoria del computador.

4.2.1. Aumentación de data

Previo al inicio del entrenamiento de los modelos, se realizó la aumentación de data, utilizando la librería de Imgaug [40]. De esta forma se le aplican 5 filtros distintos al *dataset* original de 200 imágenes para obtener un total de 1200 imágenes con las cuales poder entrenar el modelo. En la Figura 24 se tiene los resultados de la aplicación de cada filtro descrito en la sección 3.3.

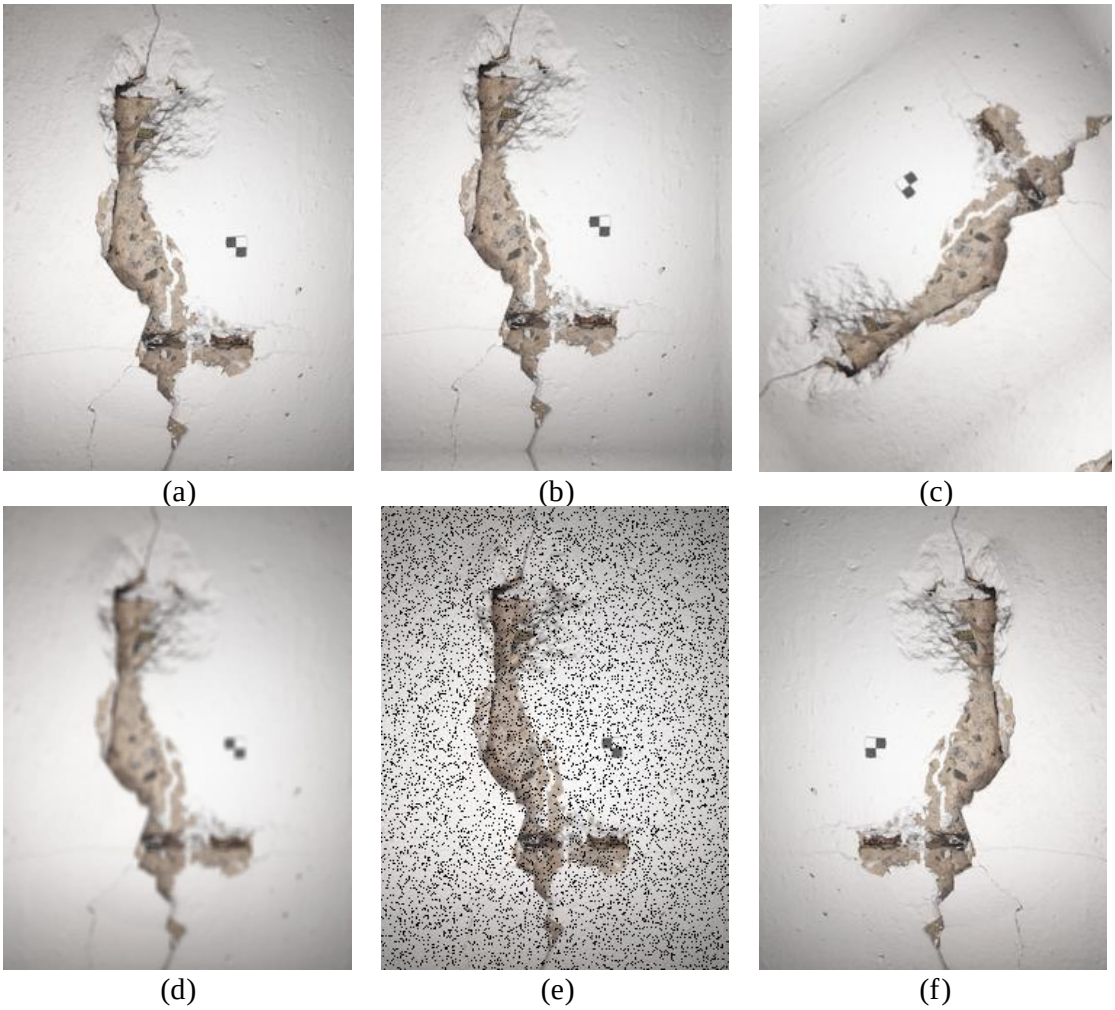


Figura 24. Filtros aplicados con Imgaug: A) Imagen Original B) Imagen Trasladada C) Imagen Rotada D) Imagen Borrosa E) Imagen con Dropout F) Imagen con Flip

4.2.2. Detección de cuadro escala

El objetivo principal de este paso es poder obtener una relación exacta de pixel vs cm, elaborándose un modelo en Unet con los hiper parámetros de entrada de la Tabla 5 y el *dataset* de 200 imágenes más las imágenes aumentadas, pero etiquetando solo las escalas. Finalmente, en la Figura 25 se tiene la predicción hecha con el modelo junto a la imagen real que se puede evidenciar buenos resultados que van acorde al valor de F1-score obtenido.

Tabla 5. Hiper parámetros y resultados - Cuadro de escala

Learning rate	**Batch Size**	**Epoch Size**	**Accuracy (%)**	**F1-Score**
0.001	102	200	98.40	0.90

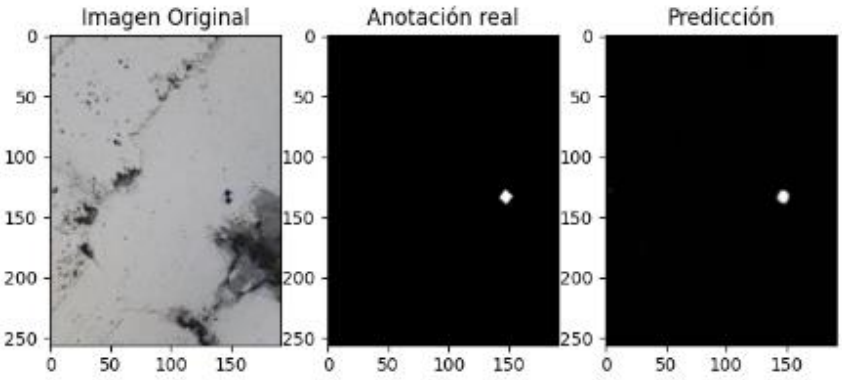


Figura 25. Predicción – Cuadro de escala

4.2.3. Segmentación y medición del daño

Por un lado, en la Tabla 6, se muestran los diferentes hiper parámetros usados para entrenar el modelo de Unet. El primero de estos hiper parámetros es la razón de aprendizaje (*Lnr*) de cada bloque, seguido del *batch size* y los *epochs*. Cada uno de estos fueron manualmente modificado a partir de múltiples iteraciones en las que se compararon sus resultados, optando por valores múltiplos del total de imágenes para su *batch size*. De la misma forma, en la tabla también se muestran los valores de *accuracy* y el *loss* de entrenamiento obtenido según el optimizador escogido, observándose que el optimizador "Adam" [31] brinda los mejores rendimientos. Por otro lado, en la Tabla 7, están los hiper parámetros de entrada que se le dieron al modelo YOLO, con el que utilizó el modelo base llamado yolov8l.seg. El primero de estos hiper parámetros es la razón de

aprendizaje (*Lnr*) de cada bloque, seguido del *batch size* y los *epochs*. Cada uno de estos fueron manualmente decididos a partir de múltiples iteraciones en las que se compararon sus resultados, manteniendo constante el uso del optimizador "Adam" y realizando un entrenamiento multi instancia para la segmentación del cuadro escala a la par.

Tabla 6. Hiper parámetros de entrada y resultados de entrenamiento - Unet

N°	Lnr	Batch Size	Epochs	Accuracy (%)	Loss (%)	F1-score val
1	.00010	9	50	94.93	9.86	0.50
2	.00075	42	100	95.48	4.60	0.61
3	.00010	102	100	96.70	3.14	0.59
4	.00010	60	100	96.70	1.77	0.80
5	.00010	102	100	33.60	27.49	0.50
6	.00010	102	150	43.00	55.00	0.51
7	.00010	102	200	96.98	1.35	0.77
8	.00050	102	200	96.92	1.51	0.87
9	.00025	102	350	96.96	2.77	0.80
10	.00035	170	300	96.99	0.99	0.90

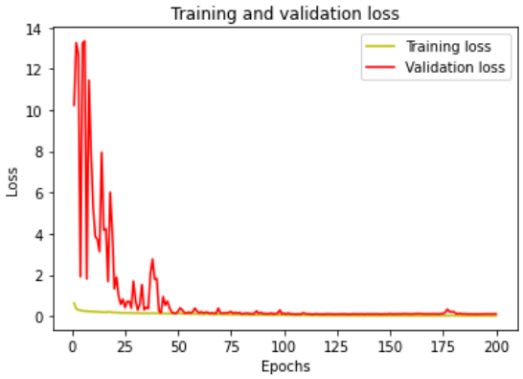
Tabla 7. Hiper parámetros de entrada y resultados de entrenamiento - YOLO

N°	Lnr	Batch Size	Epochs	mAP50	Loss (%)	F1-score val
1	.00010	11	288	0.63	2.70	0.53
2	.00025	102	200	0.65	2.85	0.63
3	.00010	85	250	0.64	2.50	0.60
4	.00100	85	271	0.66	2.30	0.64
5	.00100	85	320	0.59	2.66	0.68
6	.00100	60	350	0.20	10.00	0.17
7	.00100	102	308	0.59	2.78	0.65
8	.00050	85	158	0.61	2.70	0.66
9	.00035	170	178	0.66	3.25	0.68

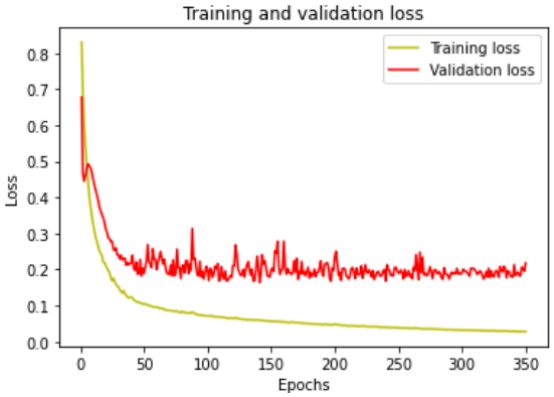
En Unet se obtuvieron los resultados finales de las gráficas en las Figuras 26 y 27 como parte de sus últimas 3 iteraciones. En estas figuras se representa como fue variando su valor de *accuracy* y de *loss* a lo largo de cada epoch, en rojo está su valor de validación y en amarillo su valor de entrenamiento. En la iteración 9 el valor de *loss* alcanzó niveles muy altos y se mantuvieron alrededor del 2% por lo que esta iteración está realizando *overfitting*. En las iteraciones 8 y 10 tienen valores de 1.51% y 0.99% de loss respectivamente, estos son valores al usar un *batch size* más alto que redujo la posibilidad de *overfitting* y representan valores mucho más óptimos para poder escoger un modelo final para Unet. Por otro lado, en las gráficas de *accuracy* de las iteraciones 8 y 10 resultan haber valores máximos muy repentinos en los *epochs* iniciales, para después estabilizarse en su valor final. Múltiples fuentes [41], [42] explican que algunas de las posibles razones de esto están relacionadas al valor asignado a *batch size,* a la cantidad de data total y a la variante de decrecimiento de gradiente dentro del optimizador "Adam". Tomando en cuenta que el *batch size* en la iteración 10 es mucho mayor a la iteración 8 (170 > 102), se confirma esta primera razón. En el futuro se tendrá que realizar más iteraciones con una mayor cantidad de data. Finalmente, en la Tabla 8 se tiene una comparación del F1-score de las iteraciones de Unet, con el F1-Score de la iteración N°10 siendo el mejor con 89.47%.

Tabla 8. Resultados de F1-score, Precision y Recall para Unet

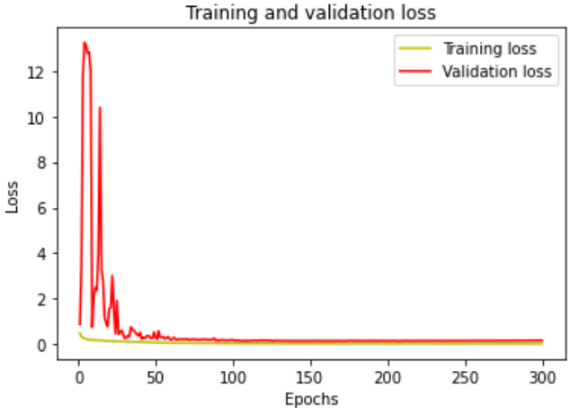
N°	**F1-score val**	**Precision val**	**Recall val**
1	49.64%	71.21%	38.11%
2	61.21%	69.85%	54.47%
3	59.67%	81.49%	47.06%
4	80.07%	86.92%	74.22%
5	49.87%	40.61%	64.58%
6	51.36%	43.41%	62.89%
7	77.45%	91.13%	67.34%
8	86.79%	92.71%	81.58%
9	79.79%	82.62%	77.14%
10	89.47%	89.74%	89.19%



Iteración #8

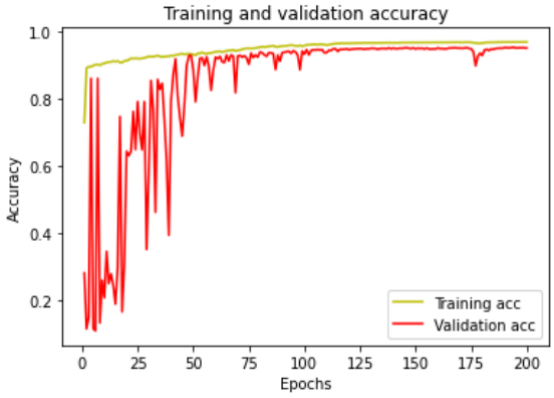


Iteración #9

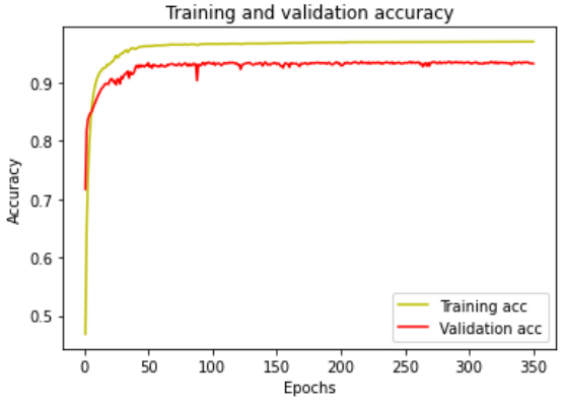


Iteración #10

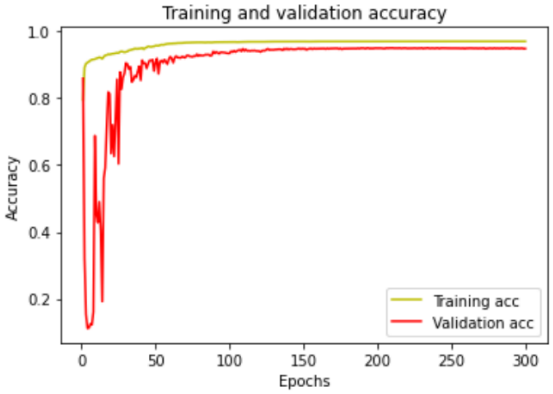
Figura 26. Loss de entrenamiento y validación de descascaramiento – Unet



Iteración #8



Iteración #9



Iteración #10

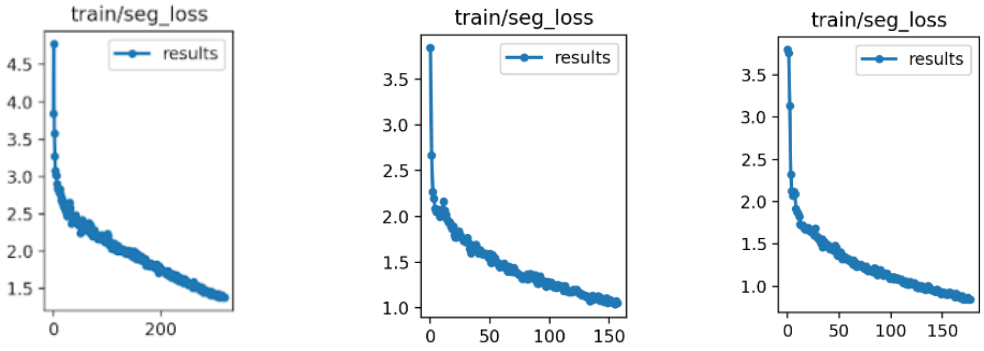
Figura 27. Accuracy de entrenamiento y validación de descascaramiento - Unet

Con el modelo YOLOv8 se obtuvieron los resultados finales de las gráficas en las Figuras 28 y 29 como parte de sus las iteraciones número 5, 8 y 9 respectivamente. En estas figuras se representa como fue variando su valor de mAP (Comparación entre el cuadro predicho y el cuadro como parte del *ground truth*) y su *loss* durante el entrenamiento en cada epoch. Por un lado, se puede evidenciar como el valor de *loss* fue establemente decreciendo hasta valores altos en comparación con Unet (alrededor del 2%) sería necesario aumentar la data para poder reducir este valor. Por otro lado, notamos como en todos los casos la métrica de mAP converge alrededor de 0.60 con algunas fluctuaciones a lo largo del modelo que fueron mitigadas con un valor mayor de *batch size*. A pesar de esto, los resultados en la iteración número 9 cuentan con un valor de F1-score más alto que los otros (F1-score = 0.68) con mayor tendencia al *optimal-fitting* pues su gráfica de mAP50 de la Figura 28 es más estable.

Si bien la variable del costo computacional no ha sido analizada en las Tablas 6 y 7, es importante brevemente resaltar la diferencia en ambos modelos. El tiempo promedio de entrenamiento para Unet ha sido de dos días, mientras que el de YOLO fue de alrededor de dos horas. Esto debido a la cantidad de capas y al proceso en cada convolución que es más complejo en Unet a comparación de YOLO.

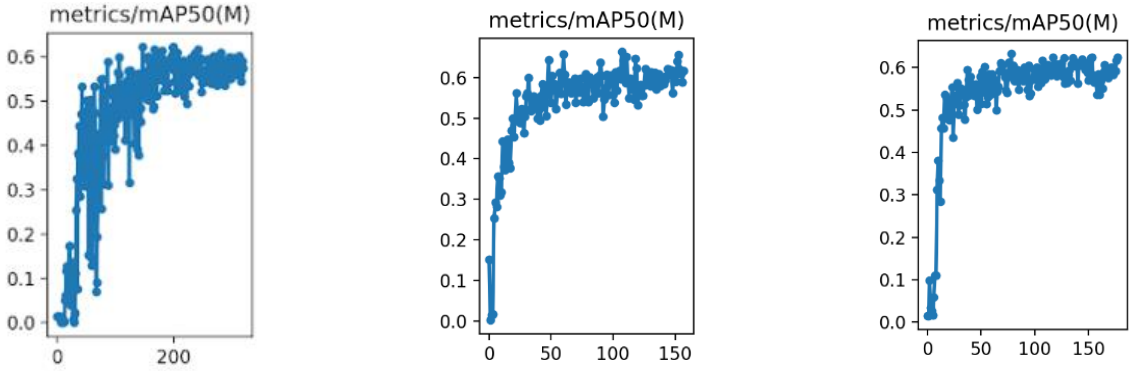
Tabla 9. Resultados de F1-score, Precision y Recall para YOLO

N°	F1-score val	Precision val	Recall val
1	53.01%	68.86%	43.10%
2	63.25%	67.70%	59.33%
3	59.60%	66.40%	54.07%
4	64.12%	73.42%	56.78%
5	67.40%	72.05%	63.28%
6	17.86%	18.95%	16.86%
8	66.00%	70.50%	61.20%
9	68.00%	73.24%	64.00%



Iteración #5 Iteración #8 Iteración #9

Figura 28. Loss de entrenamiento de descascaramiento – YOLOv8



Iteración #5 Iteración #8 Iteración #9

Figura 29. mAP50 de entrenamiento de descascaramiento– YOLOv8

4.2.3.1. Comparación de predicciones entre modelos de segmentación

Por un lado, se realizaron predicciones con los cuatro últimos modelos iterados utilizando la arquitectura Unet con los hiper parámetros presentados en la Tabla 7. En la Figura 30 está la primera columna con la imagen original a testear, en estas el acero se encuentra expuesto y el descascaramiento se encuentra a similares niveles de iluminación. En la segunda columna está la máscara real donde se ubica la patología y en las últimas columnas está las predicciones hechas en cada iteración del modelo. De esta forma se puede notar como fue mejorando el modelo según variaban los hiper parámetros escogidos, además se puede notar como la iteración con el valor de F1-score más alto (iteración #10) obtiene los mejores resultados, en el caso de la última fila se puede notar como la predicción es más fina que la de la anotación real, con lo que el modelo evitó predecir algunas marcas incorrectamente etiquetadas como descascaramiento. Es posible que resulte de un menor valor de *Lnr*, al estar utilizando el método de optimización de gradiente descendente con el que cada paso se volvería más pequeño y resulta en una mejor aproximación de la derivada verdadera. Cabe mencionar que escoger un *Lnr* muy pequeño, cae en la posibilidad de converger a un óptimo local en vez de un óptimo global.

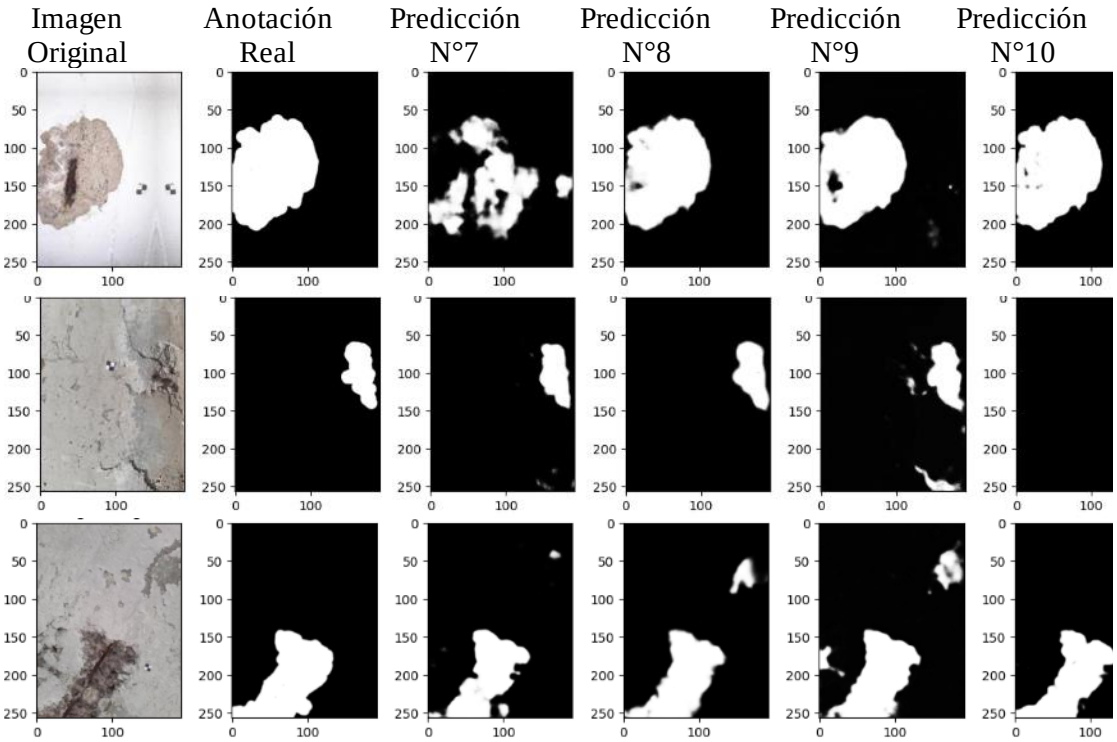


Figura 30. Predicción del modelo Unet

Por otro lado, se realizaron predicciones de multi instancia (se realizó predicciones de múltiples clases, *spall*, *square* y *rectangle*) con las iteraciones número 5, 8 y 9 del modelo utilizando la arquitectura YOLO con los hiper parámetros presentados en la Tabla 8. En la Figura 31 se realizó la predicción sobre las mismas tres imágenes mostradas previamente, en estos resultados se puede notar cómo solo en una de las tres imágenes se logró una predicción adecuada en las tres mejores iteraciones. Por otro lado, en la primera imagen ninguna iteración logró una segmentación adecuada del descascaramiento, pero sí de la escala, en la tercera imagen se encuentra resultados similares, pero también se segmentó un área erróneamente como descascaramiento. De esta forma se descartar la iteración número 9 y tomando en cuenta su valor de F1-score detallado en la Tabla 7 la iteración número 5 será utilizada en esta primera versión.

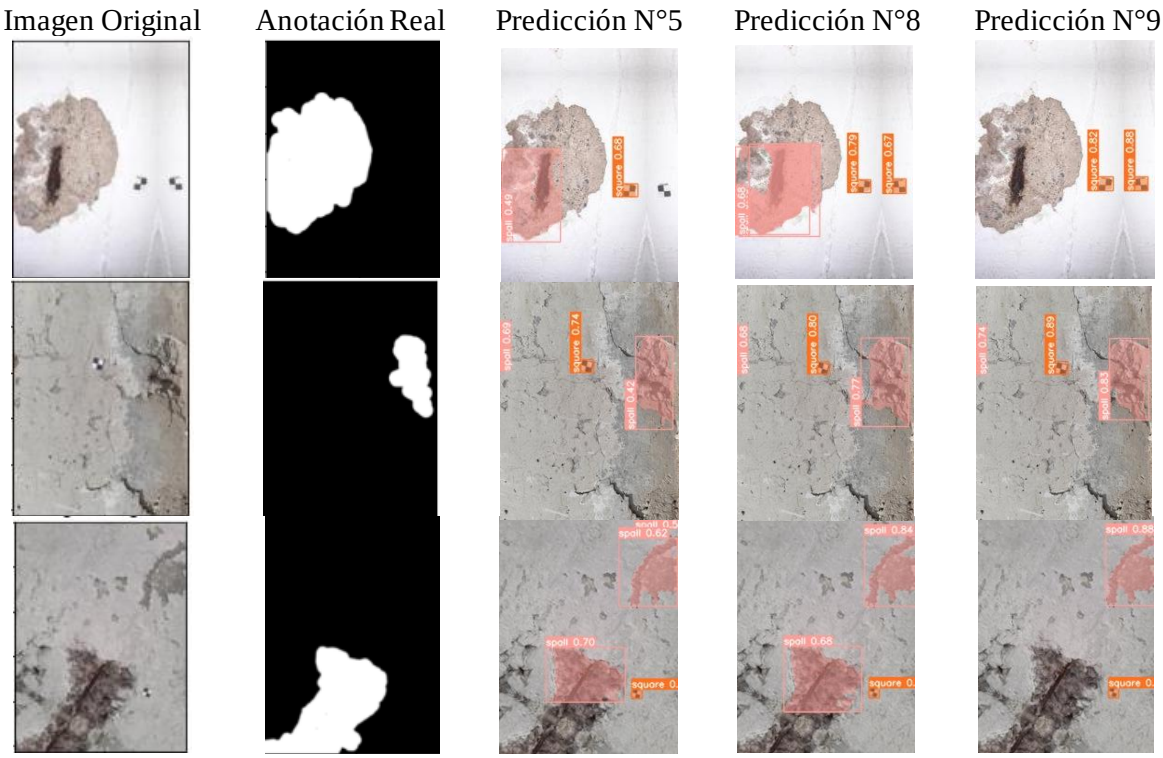


Figura 31. Predicción del modelo YOLOv8

Finalmente, a partir de estos resultados el modelo elaborado con la arquitectura de Unet ha demostrado ser el mejor para esta aplicación, superando en la tarea de segmentación del descascaramiento a YOLO según el nivel de precisión para determinar y evitar erróneamente segmentar el daño. También se evidencia esto según sus resultados de *F1-score* (89.47% para Unet y 68% para YOLO). Es importante resaltar que ambas arquitecturas se entrenan de manera distinta en función de sus hiper parámetros y se necesita un mayor nivel de detalle para sus ajustes en el modelo de YOLO a comparación de Unet. Sin embargo, este nivel mayor de detalle daría la opción a mejoras en los resultados para ambas arquitecturas, ajustando valores como su momento (SGD) o los valores beta (Adam). En lo general está claro que para mejorar los resultados en ambos casos es necesario un *dataset* mayor que permita entrenar ambos modelos en casos más generales y además evitar el *overfitting* presente en todos los modelos.

4.2.3.2. Estimación del diámetro

Con los modelos finales escogidos para Unet y YOLO, se testeó la metodología propuesta de la sección 3.3 con el uso de la relación escala vs pixel para poder estimar el diámetro y comparar cómo es que varía según su valor real. Es importante mencionar que se estará considerando un rango aceptable de error de ±25%. De esta forma se obtienen los resultados de las Figuras 32 y 33, que muestran los valores de diámetro real y predicho en cada modelo. La línea diagonal indica una predicción perfecta del diámetro en cada caso. Adicionalmente, se calculó el coeficiente de determinación ($\boldsymbol{R^2}$) entre las predicciones con su valor real como se describe en la sección 2.4.4.1.

Para el modelo de Unet se obtuvo un valor de $R^2 = 0.64$ con casos atípicos principalmente por encima de la diagonal. Sin embargo, se puede evidenciar como conforme el diámetro aumenta al modelo se le dificulta poder determinar correctamente su diámetro debido principalmente a la falta de cantidad de imágenes con diámetros mayores a 30.

Para el modelo de YOLO se obtuvo un valor de $R^2 = 0.26$, esto quiere decir que el modelo es poco eficaz para la determinación del diámetro en base a imágenes. En todos los casos hay una dispersión considerable que se acentúa conforme el daño es de mayor diámetro. Esto guarda relación con los ejemplos presentados en la Figura 31, donde las predicciones de identificación del descascaramiento son bastante reducidas e incompletas. A diferencia del caso de Unet donde las predicciones fueron bastante precisas.

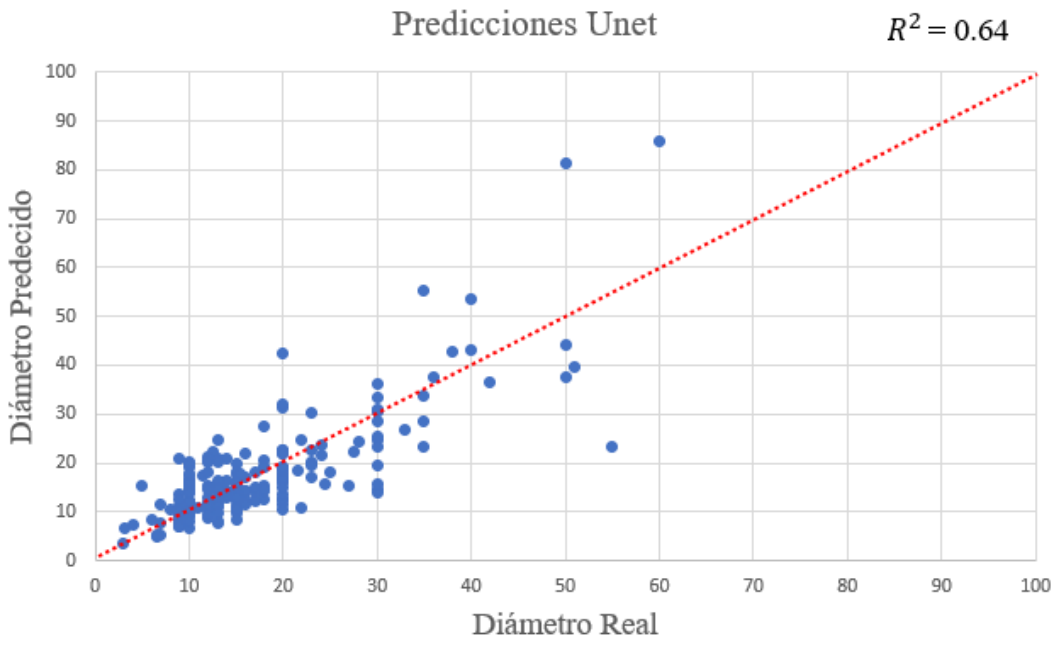


Figura 32. Coeficiente de determinación en base a las predicciones de Unet

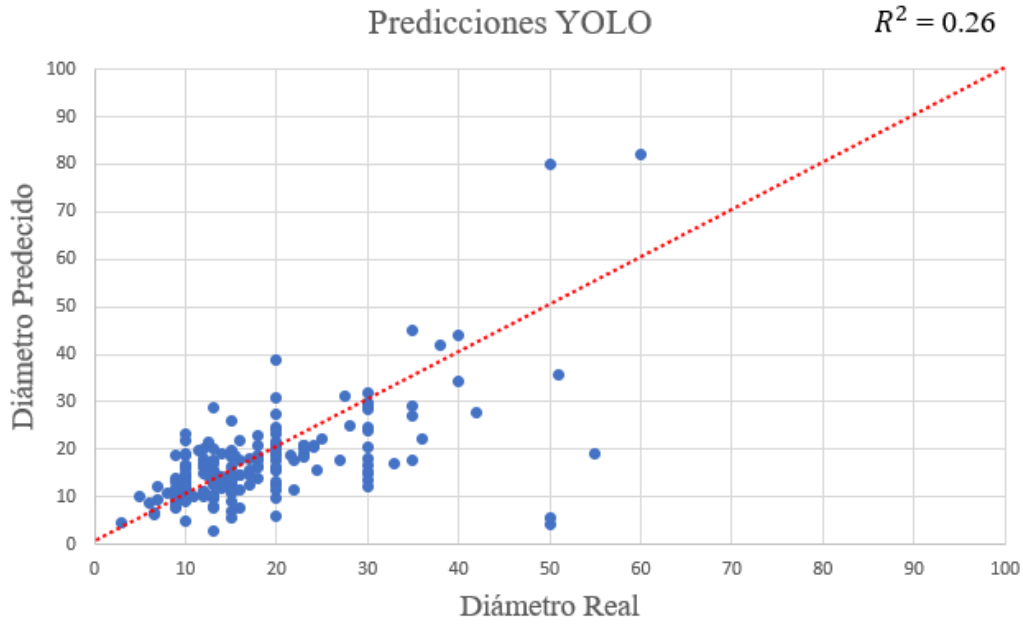


Figura 33. Coeficiente de determinación en base a las predicciones del modelo YOLO

Con la cantidad de pixeles de cada modelo se puede hacer uso de la escala pixel vs cm obtenida con el modelo de segmentación del cuadro escala para obtener una medición real del daño. En la Figura 34 se pueden ver dos ejemplos de la diferencia en las predicciones de YOLO y Unet, el primer caso se puede evidenciar como el modelo de Unet predijo un diámetro menor al real y en el segundo caso como YOLO predijo muy por debajo del valor real. Tomando en cuenta las predicciones de la Figura 30 y 31, es razonable asumir que la disparidad en los resultados para la predicción del diámetro entre los modelos de YOLO y Unet no es por su capacidad de detectar dónde está el daño sino cómo se calcula su diámetro.

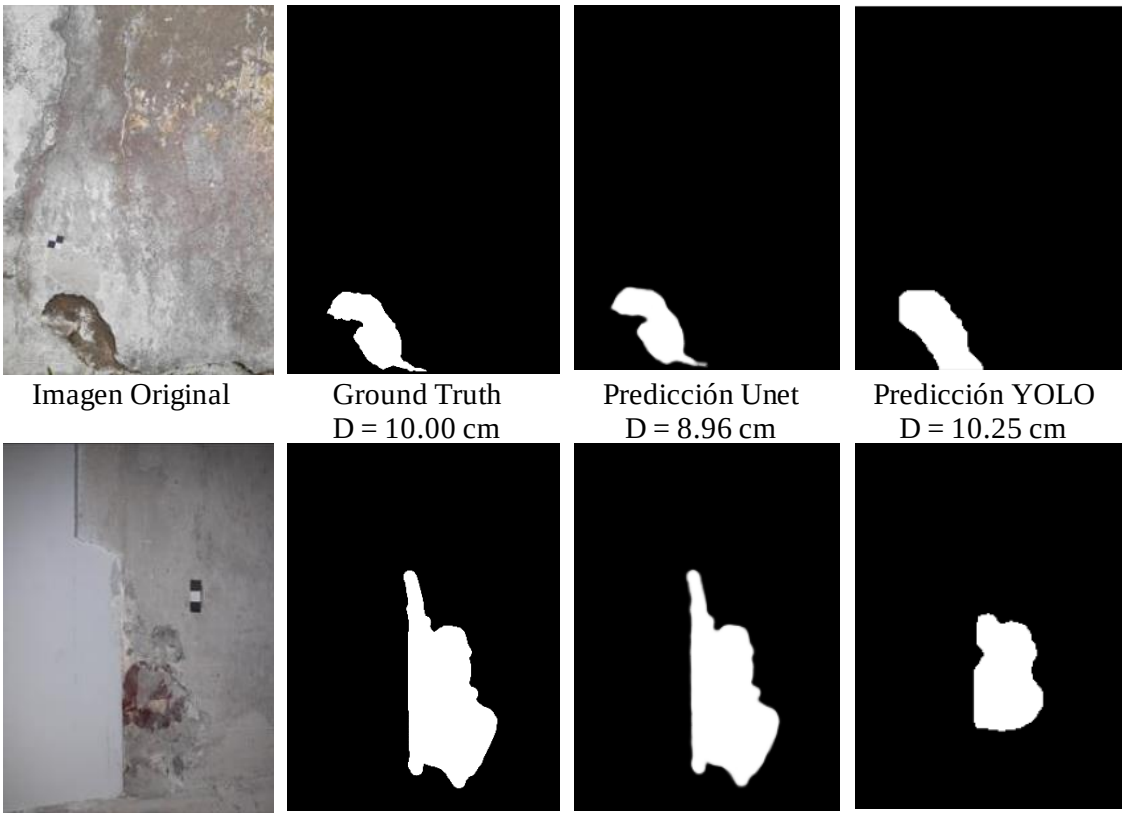


Figura 34. Ejemplo de resultados entre YOLO y Unet

Como se describió en la sección 4.1 junto con la Tabla 1, no hay una definición exacta sobre cuál es el diámetro en las guías internacionales para el descascaramiento. En este estudio se promedió entre un diámetro de valor mayor y uno de valor menor para reportar el promedio como el diámetro real (*ground truth*) del descascaramiento. Esto quiere decir que ambos modelos dependen de la geometría del daño y del modelo de detección del cuadro escala para indicar su diámetro. La geometría del daño es imposible

de controlar, pues este proviene de condiciones naturales por el que superficies de concreto son expuestas. El modelo de detección del cuadro escala sufre de la necesidad de tener que ser bastante preciso con los pixeles que detecta. En cualquiera de las figuras presentadas donde se encuentre este cuadro, se puede notar que es una pequeña parte de la imagen total. Considerando que las imágenes han sido redimensionadas de 3000x4000 pixeles a 256x192 es razonable decir que se está perdiendo información en la forma de pixeles con las que el modelo pueda detectar de forma perfecta este cuadro y así tener una relación de pixeles vs cm precisa.

A pesar de estas limitaciones en la estimación del diámetro se tiene las Tablas 10, 11, 12 y 13 para la cuantificación del índice de daño de los 200 casos de descascaramiento. Las Tablas 10 y 12 son matrices de confusión donde cada fila representa los valores reales del diámetro y las categorías en cada columna representan los valores predichos según los modelos escogidos.

En la Tabla 10 se presenta la matriz de confusión de YOLO, se comprueba que el modelo tiende a subestimar el daño. Estos son categorizados principalmente en la categoría inferior de "Leve" con un F1-score de 0.69, para la categoría "Moderado" se obtuvo un F1-score de 0.54 y para la categoría "Severo" un F1-score de 0.58. Confirmando su capacidad de detectar el daño en rangos de diámetro bajo, es posible que esto se deba a que a menor diámetro la variación de la geometría disminuya y, por lo tanto, la ambigüedad de la definición del diámetro en las guías internacionales toma un rol menos importante. Este modelo consiguió predecir correctamente el diámetro en 52.50% de los 200 casos presentados, considerando un rango de error del 25%.

En la Tabla 12 se reporta la matriz de confusión de Unet, se evidencia una mejora en la predicción en cada uno de los niveles de daño a comparación del modelo de YOLO. Su F1-score para la categoría "Leve" es de 0.66, para la categoría "Moderado" es de 0.51 y la categoría "Severo" es de 0.70. Al igual que con YOLO, persiste el grado de dificultad para identificar correctamente los casos moderados como se puede evidenciar en la Tabla 11 y 13, teniendo resultados similares para la detección correcta en la categoría "Leve". Pero a diferencia de YOLO, el modelo de Unet tuvo una alta capacidad de detectar correctamente los casos "Severos", esta observación se debe tomar con precaución pues la cantidad de imágenes en esta categoría es muy limitada y es necesario un mayor número

de data para mejorar su representatividad. Los casos de predicción correcta para este modelo son de 61% sobre los 200 casos recolectados.

Tabla 10. Matriz de confusión - Predicción del daño en función del diámetro y el modelo YOLO

YOLO	**Leve**	**Moderado**	**Severo**
Leve	67	23	1
Moderado	28	42	4
Severo	6	13	16

Tabla 11. Resultados de matriz de confusión - Predicción YOLO

YOLO	**Leve**	**Moderado**	**Severo**
Predecido	101	78	21
Real	91	74	35
F1-score	0.70	0.55	0.57

Tabla 12. Matriz de confusión - Predicción del daño en función del diámetro y el modelo Unet

Unet	**Leve**	**Moderado**	**Severo**
Leve	63	28	0
Moderado	30	39	5
Severo	4	10	21

Tabla 13. Resultados de matriz de confusión - Predicción Unet

Unet	**Leve**	**Moderado**	**Severo**
Predecido	97	76	26
Real	91	74	35
F1-score	0.67	0.52	0.70

4.2.3.3. Estimación de la profundidad

Primero se realizó el análisis de la intensidad y su relación con la zona dañada real en la imagen. Para esto en la Figura 35 se puede revisar tres imágenes que detallan un caso de descascaramiento, su predicción según el modelo de Unet elaborado previamente y una tercera imagen que representa su contorno según sus valores de intensidad. La línea de color rojo indica la ubicación del punto más profundo para este caso determinado en base al promedio geométrico de los valores de intensidad de la imagen en su escala de grises. Esto confirma la relación que tiene los valores de intensidad con la zona dañada y más profunda de descascaramiento.

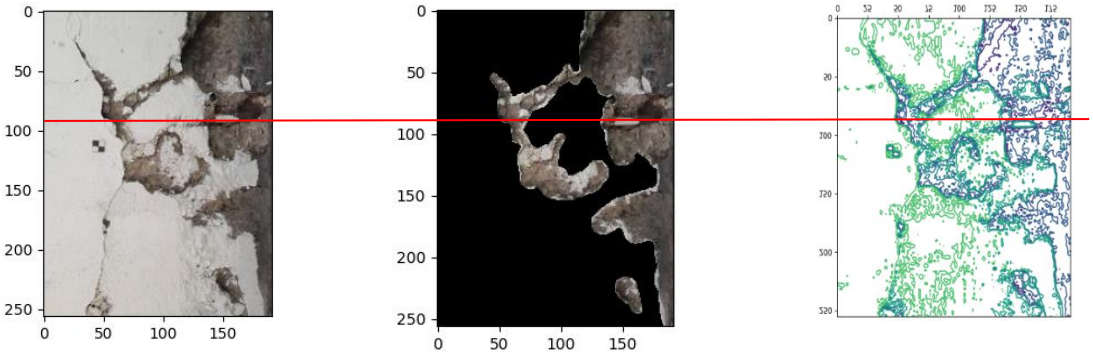


Imagen Original Predicción Análisis de intensidad

Figura 35. Análisis de intensidad

Con este análisis se puede observar la Figura 36 donde se muestra un perfil de los valores de intensidad por encima de la línea roja mostrada en la figura previa. Aquí nuevamente se puede notar los valores más intensos de profundidad como la ubicación donde se encuentra el daño por descascaramiento. Para poder dar razón a un modelo estadístico que pueda predecir la profundidad en cualquier imagen de descascaramiento, se propone que la diferencia entre los valores máximos y mínimos de intensidad representen la profundidad máxima medida. De esta forma se elaboraron los gráficos en la Figura 37, donde se muestra una regresión polinomial de segundo orden para cada valor de profundidad medida. Se realizaron cuatro funciones de regresión para los distintos canales de una imagen, escala de grises, rojo, verde y azul de los casos de descascaramiento. Se observa que las funciones propuestas no cuentan con un adecuado coeficiente de determinación para hacer mediciones precisas de la profundidad de descascaramiento.

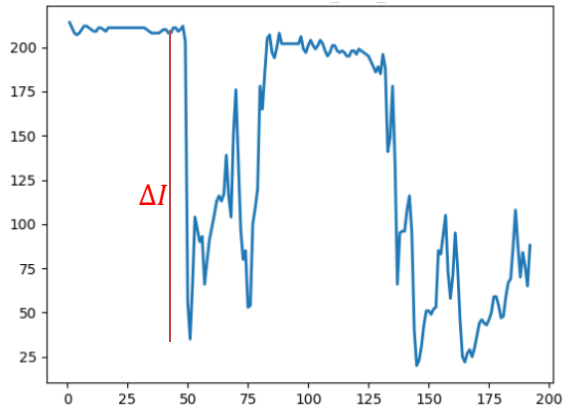
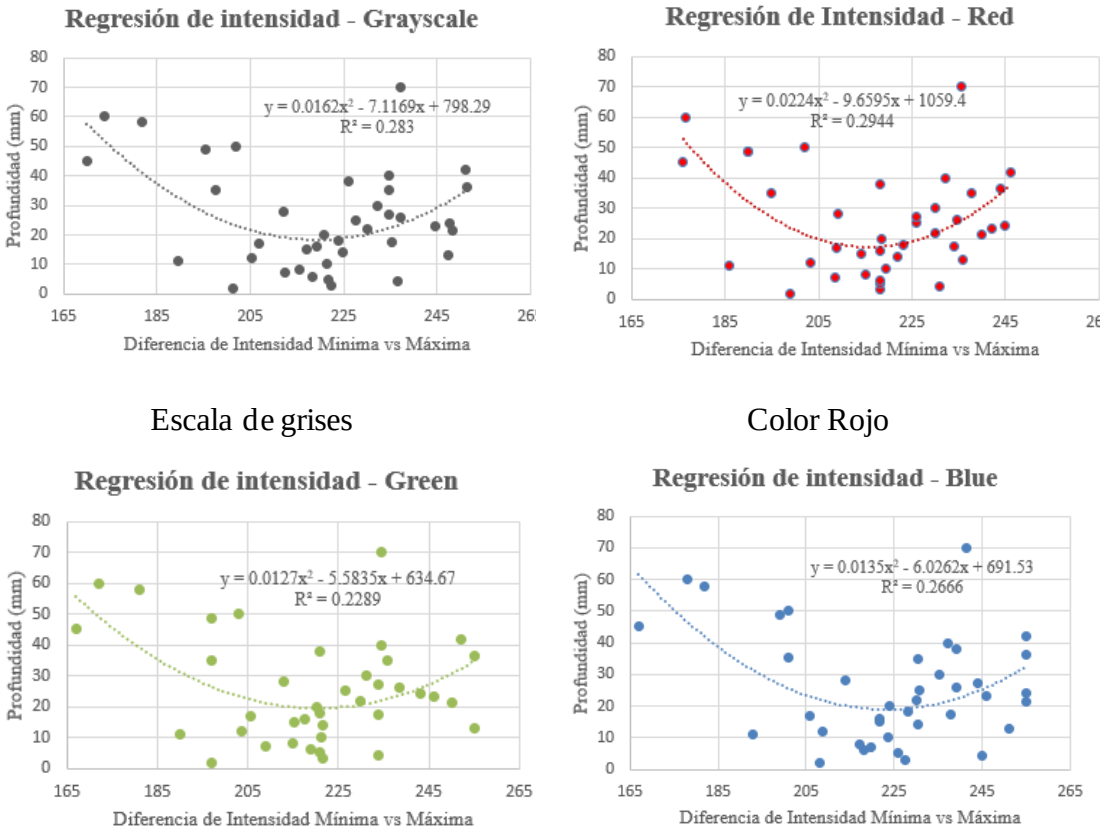


Figura 36. Perfil de la fila de intensidad más profunda

Los mejores resultados fueron obtenidos por el canal de intensidad rojo para la imagen en escala RGB y su escala de grises, con valores de 0.29 y 0.28 respectivamente. Estos modelos de regresión dificultan la predicción de los casos de profundidad de categoría “Leve”, en el canal de intensidad roja para la escala RGB resultó que para las imágenes con profundidades menores a 25 cm se predicen 94% por encima de su valor real, mientras que para las imágenes con profundidades mayores a 25 cm se estimaron 10% por debajo de su valor real. Se pueden evidenciar en las gráficas diferentes valores atípicos, que no se ajustan a la curva planteada, y que afectan los resultados finales. Estos valores atípicos se identificaron como los casos donde hubo la mayor dificultad para medir su diámetro y profundidad debido a que el daño no se encontraba expuesto en su totalidad u ocupaba un área muy extensa en comparación con el tamaño de la imagen.



Escala de grises Color Rojo

Color Verde Color Azul

Figura 37. Regresión de intensidad según su canal de color

Los resultados encontrados en este análisis resaltan el reto más grande para la inspección y evaluación automática de descascaramiento. La necesidad de imágenes de alta calidad tomadas bajo condiciones similares y que sean capturadas por equipos

fotográficos que tengan la capacidad de tomar en formato RGB-D, este tipo de formato incluye información de su profundidad. De esta forma se pueden emplear herramientas de CNN que demuestran una alta eficiencia en la detección de la profundidad de daños de concreto. Uno de ellos es Marigold [43], un modelo de difusión y asociado a un ajuste fino para estimación monocular de profundidad. Este modelo tiene como objetivo de aprender de centenas de datos muestreados para lograr una alta capacidad de predicción de profundidad. Para ello recurrieron a cerca de 70 mil imágenes RGB-D elaboradas de forma sintética y obteniendo resultados de cerca del 95% para la métrica de *accuracy* en múltiples *datasets* como NYUv2 [44]. Por ello, se resalta la necesidad de incrementar considerablemente el *dataset* actual para lograr un modelo específico para la estimación de la profundidad o intentar conseguir data sintética como se empleó en Marigold.

4.2.3.4. Implementación final

Como se describió en la sección 3.4, se realizó una interfaz que permite a cualquier usuario procesar sus imágenes de descascaramiento con los modelos escogidos y requiere la data este en formato jpg, de dimensiones 192x256 pixeles y contenga la plantilla utilizada en la Figura 16. Esta interfaz es capaz de cuantificar su diámetro en base a los modelos de Unet y YOLO escogidos en la sección 4.2.3.2. La interfaz se elaboró con la librería de *tkinter* [45] en Python. Primero, en la Figura 38 se tiene la pantalla de bienvenida del interfaz del software en la que se indica los requerimientos mínimos que las fotos necesitan tener, junto con un botón que permite al usuario escoger la carpeta donde se encuentren las fotos.

Figura 38. Pantalla de inicio de interfaz de implementación

Segundo, en la Figura 39 se da la opción de previsualizar las fotos escogidas para verificar que son las correctas. En la parte inferior hay un botón con el que se analizan las imágenes o volver a inicio a escoger una nueva carpeta donde se ubiquen las imágenes.

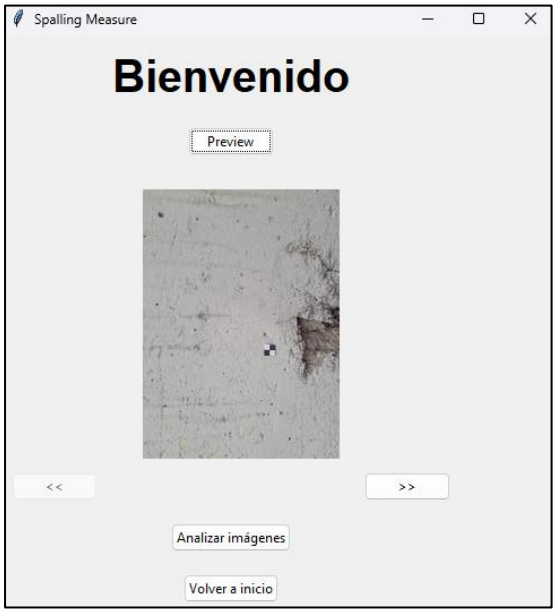


Figura 39. Previsualización de la data

Finalmente, se tiene la Figura 40 con todas las imágenes procesadas en los modelos de Unet y YOLO. Debajo de cada imagen se tiene su diámetro en milímetros y su categoría de ID. En el anexo 3 se presenta el código y ejemplos adicionales de imágenes analizadas con el interfaz.

Figura 40. Data procesada – Categoría de daño y diámetro

CAPÍTULO V

Futuros Estudios

En un estudio futuro se espera cubrir diferentes opciones de mejora para una inspección eficiente y de bajo costo para puentes. Primero se sugiere cambiar el círculo equivalente de los pixeles segmentados por un óvalo equivalente, cómo se mencionó en el módulo uno. Dentro de los 200 casos de descascaramiento, se evidenció que su geometría es más similar a la de un óvalo que a la de un círculo, por lo que se disminuiría el error estimado en los modelos de Unet y YOLO para los diámetros medidos. En la Figura 41, se muestra un ejemplo gráfico de los dos diámetros que se obtienen de un óvalo, el mayor de estos sería utilizado para categorizarlo en base a las guías de inspección actuales y el análisis de ambos sería utilizado para proponer una nueva forma en cómo se categorizan los índices de daño en las guías internacionales.

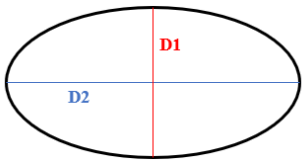


Figura 41. Diámetros encontrados en un óvalo

Una de las limitaciones para la implementación del presente estudio es, ¿cómo es que se utilizará el cuadro escala en lugares de más difícil acceso? La metodología propuesta requiere a alguien que físicamente coloque el cuadro escala y tome una foto a una distancia reducida con el fin que un modelo de segmentación la pueda detectar. Una alternativa de mejora es utilizando un cuadro escala como AprilTags que cuenta con su propia librería para detectarlo, ver Figura 42. En el estudio por Cesar et al [46] compararon la efectividad de este cuadro escala con otros similares para distintos casos de distancia y turbidez, mostrando que diferentes niveles de turbidez resultan en distintas calidades de fotos para la escala. De esta forma es cómo se evidenció los mejores resultados a comparación de otros a distancias máximas de 220 cm y ángulos máximos de fotos de 85°. De esta forma se podría evitar el entrenamiento de un modelo de segmentación solo para el cuadro escala y mantendrá la seguridad de su aplicación para los inspectores de campo.

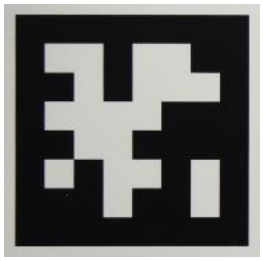


Figura 42. Cuadro de la librería AprilTags (Tomado de [46])

Adicionalmente, otra alternativa que optimice la segmentación de descascaramiento es la aplicación de *Segment Anything* (SAM) elaborado por Meta AI [47]. Propone la capacidad de segmentar cualquier objecto de una imagen de forma automática. Dando la oportunidad de acelerar la obtención de los resultados en grandes cantidades de imágenes. Otra aplicación importante de arquitecturas como SAM es que pueden agilizar y homogeneizar el proceso de anotación. En el estudio hecho por Wu et al [48] entran en extenso detalle con arquitecturas similares a esta sobre su posibilidad de utilizarse como input de entrenamiento de otros sistemas de Machine Learning y otras formas en las que se puede automatizar la anotación de imágenes. Pero esto no significa que es una solución definitiva, el estudio argumenta que una anotación completa y consistente es un problema de alto grado tanto para computadoras como para humanos. Lo bueno del uso de estas arquitecturas es que se evita el uso de una extensa cantidad de tiempo que se le tiene que aplicar a la anotación y a su homogenización en el etiquetado. En la presente tesis se evidenció que al pedir ayuda a otras personas para cumplir con esta labor la homogenización era el punto más frágil que requirió de mayor atención. La forma de mitigación en este estudio fue el de revisar con detalle los resultados y recalcar repetidas veces los detalles del objeto de estudio.

Se espera poder comparar el objetivo de la segmentación del descascaramiento con más modelos de segmentación semántica. Uno de ellos es con el modelo de segmentación de Mask R-CNN [49] desarrollado en el 2018, su propósito es obtener segmentación por instancias con la correcta detección de las clases al igual que segmentar cada instancia de forma paralela. Otro es el uso de DeepLab [50] elaborado en el 2017, esta red neuronal reutiliza una CNN entrenada en clasificación de imágenes y es dado un nuevo propósito en segmentación semántica. Primero transforman todas las capas completamente conectadas a capas convolucionales. Después aumentan la resolución de captación de características con el uso de capas convolucionales dilatadas para finalmente

emplear una interpolación bilineal que obtenga una muestra superior por un factor de 8 y alcanzar la resolución original de la imagen. Ambos modelos han demostrado ser capaces de tareas similares a las de Unet y YOLO, por lo que un nuevo punto de comparación daría una posible conclusión distinta a la de este estudio.

Esta tesis sólo tomó en cuenta la recolección de imágenes RGB y un solo punto para poder determinar su profundidad. Sin embargo, Huang et al. [51] analizaron la efectividad de fusionar data 2D en escala de grises e imágenes RGB con su determinada data de profundidad. Determinó que hacer uso de data RGB junto a la data real de profundidad categorizada como RGB-D es mucho más confiable para el entrenamiento de modelos de predicción. Además, hicieron una revisión breve de otros estudios que permitan estimar la profundidad en base a modelos *encoder-decoder*. Este estudio realizó su análisis para baches en carreteras, pero su metodología y alternativas pueden ser extendidas para ser revisadas con descascaramiento.

Cómo se resaltó en Luo et al. [10] recolectar data para estudios de defectos en puentes sigue siendo bastante complicado. Hay un vacío en la academia de *open-source datasets* bastante grande, en la presente tesis se recolectó data de superficies de concreto de distintas estructuras. Pero es importante resaltar que a mayor financiación está la posibilidad de mapear los puentes que no han sido inspeccionados por Provias Nacional [4] de esta forma se podrá elaborar un *dataset* lo suficientemente extenso, con múltiples patologías en puentes de distintos materiales y tipos a lo largo de todo el Perú, ver Figura 1. La dificultad principal de esto es el tema de la seguridad, los permisos a las municipalidades locales y regionales y la financiación para viajar a cada ciudad y puente mapeado. Siguiendo esta idea, la metodología propuesta no se limita a ser utilizada solamente para edificaciones construidas con concreto armado. Cómo se detalló en el marco teórico, existen múltiples metodologías creadas con el objetivo de detectar daño superficial en concreto armado, pavimentos, etc. Las metodologías son una mejora de una propuesta anterior o una son propuestas nuevas aplicadas en casos específicos que son capaces de ser replicadas y su uso en diferentes casos dará oportunidades de mejora por obstáculos no previstos con bajo concreto armado. Por ello, se incentiva a poner a prueba la metodología de este estudio en materiales distintos al de concreto armado para confirmar su aplicabilidad y mejorarla.

CONCLUSIONES

1. Se ha recolectado un total de 200 imágenes de descascaramiento en superficies de concreto armado bajo distintos tipos de iluminación. El *dataset* cuenta con información del promedio de los diámetros medidos para cada caso y su profundidad máxima. Sin embargo, esta data se caracteriza por ser desbalanceada debido a que gran parte de sus 200 imágenes son de nivel "Leve".
2. Las guías de inspección de puentes actualmente no cuentan con una descripción clara de cuál es el diámetro en el descascaramiento. La recolección de imágenes del estudio nos da a entender que se requiere una definición más detallada de las propiedades a extraer para describir los casos reales de descascaramiento. Recolectar la data de dos diámetros es necesaria para brindar una descripción visual más precisa del daño presente en la estructura.
3. El uso de una escala física (cuadro de etiqueta) ha demostrado ser eficaz para extraer propiedades físicas y daño de imágenes en términos cuantitativos, logrando predicciones acertadas del diámetro del 61% del *dataset* según el modelo de segmentación de Unet y 51.50% de predicciones acertadas para el modelo de YOLO. Sin embargo, una reducción excesiva del tamaño de la imagen original evidenció una pérdida en la capacidad de su detección, generando falsos negativos al momento de detectar su existencia en las fotos utilizadas.
4. La técnica propuesta para medir la profundidad confirmó la relación de intensidad en la imagen con la zona de interés, sirviendo como una alternativa para segmentar el daño. Sin embargo, esta técnica produjo resultados inexactos de medición de las profundidades. Obtuvo mejores resultados en las escalas de grises y RGB con la intensidad del canal rojo donde se obtuvieron predicciones que son 94% mayores a las reales y 10% menores a las reales para las imágenes con profundidades menores a 25cm y mayores a 25cm respectivamente. La predicción de la profundidad también deberá evaluarse con el uso de CNNs, por ello es necesario un *dataset* de entrenamiento mucho más grande que cuente con información de su profundidad, es decir, en su formato RGB-D.

5. La arquitectura que presentó los mejores resultados para segmentación del descascaramiento en imágenes fue Unet con un valor de F1-score de 0.89, a diferencia del modelo de YOLOv8 que consiguió un valor de F1-score de 0.68. Además, Unet presentó resultados similares de F1-score para la estimación del diámetro en las categorías de Leve y Moderado (0.67 y 0.52) que YOLO (0.70 y 0.55), en cambio en la categoría de Severo obtuvo mucho mejores resultados (0.70) que YOLO (0.57). Por lo tanto, se recomienda usar el modelo propuesto en base a Unet para la detección de descascaramiento y medición automática del nivel de daño.
6. El presente estudio logró hacer un *proof of concept,* es decir, se evidencia la posibilidad de elaborar alternativas con costo mínimo que sean capaces de optimizar el trabajo de inspectores de puentes. Se espera que sirva como complemento a las tecnologías de alta tecnología actualmente en el mercado, logrando descartar zonas de interés por zonas con información más importante.
7. La data utilizada cuenta con un formato rectangular con una ratio de 3:4 y un tamaño de 192x256, esto se realizó con el objetivo de optimizar la velocidad de entrenamiento debido a la limitante de los equipos utilizados. En el caso de YOLO esto afectó las predicciones de los modelos entrenados en gran capacidad, el modelo base de yolov8l.seg que utiliza un rato de 1:1 y un tamaño de 640x640. En Unet esta limitante de un modelo pre-entrenado no está presente, dando mejores resultados en su capacidad de predicción.
8. Finalmente, la interfaz es prueba de la rapidez con las que se puede calcular las propiedades de las patologías en el concreto con el uso de imágenes. Agilizando el proceso de recolección de data para la inspección de puentes.

RECOMENDACIONES

1. Durante la recolección de data del presente estudio, se dificultó encontrar data que cuente con todos los casos mapeados, causando el sesgo de los resultados. Por ello se recomienda extender el *dataset* de este estudio para incluir imágenes con más casos de nivel de daño moderado y severo.
2. La recomendación principal para la recolección de data para el descascaramiento es poder solicitar acceso a las obras donde ingenieros estructurales estén realizando inspecciones estructurales. En caso esto no sea posible, se deberá recolectar data de edificaciones que se encuentren en zonas de bastante humedad.
3. Se recomienda analizar a mayor profundidad la inspección tradicional. Esto es necesario para optimizar la inspección tradicional que, en base a las ambigüedades notadas en la descripción del diámetro y profundidad para el descascaramiento resulta necesario.
4. En esta tesis se redujo el tamaño de la imagen de 3000x4000 pixeles a 192x256 (ver sección 4.2 para más información), esto ocasionó la perdida de información al momento de entrenar los modelos de segmentación, perjudicando la estimación de las propiedades de la patología. Por ello se recomienda multiplicar las dimensiones de la data por un valor mayor a 6 a las utilizadas en la presente tesis. Esto daría mejores resultados para analizar las imágenes sin tener que sacrificar mucho en cuestión de cuánto tiempo demorará en entrenar y procesar los modelos.
5. Para estimar de forma más precisa la profundidad, se recomienda hacer una medición de múltiples puntos del descascaramiento. Una forma de hacerlo es con fotografías con cámaras que den un output para RGB-D, o haciendo medición exacta de una línea de la patología presente.
6. Se recomienda utilizar librerías de aumentación de data para proyectos de Machine Learning. Esto dará mayor generalidad y un mayor número de data para ingresar los modelos.
7. En Machine Learning, la variable más demandante es el costo computacional. Esto se debe a las múltiples iteraciones que se tienen que realizar con el fin de ir probando el mejor ajuste de sus hiper parámetros para modelo testeado. Por ello, es importante revisar este aspecto con detalle y buscar de forma activa formas de optimizar el modelo. Formas de realizarlo es disminuyendo el tamaño del parámetro de entrada o

quitándole complejidad al modelo en base al número de capas o a su tipo de capas dependiendo de los hiper parámetros que se puedan modificar.

8. Se debe tener cuidado con el etiquetado de la data ya que en trabajos de Machine Learning es común contratar servicios de etiquetado de la data a terceros debido a la cantidad de data que se debe etiquetar. No sirve de nada un modelo de Machine Learning si es que la data está mal etiquetada. Por ello es importante etiquetar un grupo inicial de la data de entrenamiento para servir como base para el resto de la data a etiquetar por terceros.
9. La variable más sensible en un método que haga uso de imágenes es la luminosidad, por ello es importante intentar mantener constante la cantidad de luz en todo el *dataset* utilizado. Una forma de lograrlo es utilizando el flash del celular o una lámpara externa.
10. Se incentiva poder continuar con el estudio de la cuantificación del daño tanto para descascaramiento como para grietas. Esta área de estudio sigue siendo bastante reducida con potencial de impactar la industria de mantenimiento y operaciones de infraestructura civil a nivel mundial.

REFERENCIAS BIBLIOGRÁFICAS

[1] H. Cain y L. Arcos, "Evaluación Estructural y Funcional del Puente Cebadas, ubicado en el kilómetro 32 del tramo Guamote-Macas (Ruta E46), Aplicando la Metodología del Sistema de Administración de Estructuras de Puentes (S.A.E.P.), para su rehabilitación y conservación", 2016.

[2] MTC, "Guia para la inspeccion de puentes", *Guia para la Insp. puentes*, p. 20, 2006.

[3] A. Zulfiqar, "Design of a Bridge Inspection System (BIS) to Reduce Time and Cost", pp. 1–44, 2014.

[4] MTC, "Sistema de Puentes - Provías Nacional", 2018. https://www.pvn.gob.pe/puentes/spuentes/

[5] O. Vargas, "Provias nacional - Programa de Puentes 2012-2020", 2012.

[6] P. Nacional, "Programa Nacional de Puentes Primera etapa 2012-2016", 2016.

[7] CP-TECH, "Concrete Spalling: Causes, Effects, and Repair", 2021. https://cp-tech.co.uk/concrete-spalling-causes-effects-and-repair/

[8] A. Olukayode Bankole, "8 causes of Cracks in Concrete you should know", 2017. https://dailycivil.com/causes-cracks-in-concrete-1/

[9] I. Canon, "Detecting cracks with Ai Technology", 2019. https://global.canon/en/technology/crack2019.html

[10] K. Luo, X. Kong, J. Zhang, J. Hu, J. Li, y H. Tang, "Computer Vision-Based Bridge Inspection and Monitoring: A Review", *Sensors*, vol. 23, núm. 18, 2023, doi: 10.3390/s23187863.

[11] A. Mohan y S. Poobal, "Crack detection using image processing: A critical review and analysis", *Alexandria Eng. J.*, vol. 57, núm. 2, pp. 787–798, 2018, doi: 10.1016/j.aej.2017.01.020.

[12] S. German, I. Brilakis, y R. Desroches, "Rapid entropy-based detection and properties measurement of concrete spalling with machine vision for post-earthquake safety assessments", *Adv. Eng. Informatics*, vol. 26, núm. 4, pp. 846–858, oct. 2012, doi: 10.1016/J.AEI.2012.06.005.

[13] T. Dawood, Z. Zhu, y T. Zayed, "Machine vision-based model for spalling detection and quantification in subway networks", *Autom. Constr.*, vol. 81, núm. December 2016, pp. 149–160, 2017, doi: 10.1016/j.autcon.2017.06.008.

[14] R. Kimmel y M. Elad, "A variational Framework for Retinex", *Int. J. Comput. Vis.*, vol. 52, núm. 1, pp. 7–23, 2003, doi: 10.1023/A:1022314423998.

[15] Z. Wang y H. Wang, “Image smoothing with generalized random walks: Algorithm and applications”, *Appl. Soft Comput.*, vol. 46, pp. 792–804, sep. 2016, doi: 10.1016/J.ASOC.2016.01.003.

[16] P. Perona y J. Malik, “Scale Space and Edge Detection Using Anisotropic Diffusion”, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, núm. 7, pp. 629–639, 1990, [En línea]. Disponible en: http://authors.library.caltech.edu/6498/1/PERieeetpami90.pdf

[17] R. Gonzalez y R. Woods, *Digital Image Processing*, vol. 3. 2018.

[18] P. Moallem, M. Behnampour, P. Noise, O. N. Filter, y R. Growing, “Adaptive Optimum Notch Filter for Periodic Noise Reduction in Digital Images”, núm. 1, pp. 1–7, 2010.

[19] S. Kaur y M. Kaur, “Image Sharpening Using Basic Enhancement Techniques”, *Int. J. Res. Eng. Sci. Manag.*, vol. 1, núm. 12, pp. 122–126, 2018, [En línea]. Disponible en: https://www.ijresm.com/Vol_1_2018/Vol1_Iss12_December18/IJRESM_V1_I12_30.pdf

[20] Z. Fan *et al.*, “Ensemble of deep convolutional neural networks for automatic pavement crack detection and measurement”, *Coatings*, vol. 10, núm. 2, pp. 1–15, 2020, doi: 10.3390/coatings10020152.

[21] L. Zhang, J. Shen, y B. Zhu, “A research on an improved Unet-based concrete crack detection algorithm”, *Sage Journals*, vol. 20, núm. 4, 2020, doi: https://doi.org/10.1177/1475921720940068.

[22] M. Flah, A. R. Suleiman, y M. L. Nehdi, “Classification and quantification of cracks in concrete structures using deep learning image-based techniques”, *Cem. Concr. Compos.*, vol. 114, núm. January, p. 103781, 2020, doi: 10.1016/j.cemconcomp.2020.103781.

[23] FHWA - Federal Highway Administration, “Bridge Inspector’ s Reference Manual”, vol. 1, p. 1754, 2006.

[24] AASHTO, “AASHTO Bridge Element Inspection Guide Manual”, *Aashto*, p. 172, 2011, [En línea]. Disponible en: papers2://publication/uuid/ADF09968-5E9E-45CB-8842-8D5AD3A01307

[25] D. et al. Johnson, M.B. Casey, W. O’Donnell, L. Allec, P. Allen R Marshall, A.R. Soden, “Caltrans Bridge Element Inspection Manual”, núm. December, p. 340, 2017.

[26] US FHWA, “Highway and Rail Transit Tunnel Inspection Manual”, 2005.

[27] T. Wood, “Definición de Convolutional Neural Networks”. https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network

[28] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, “You only look once: Unified,

real-time object detection", *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[29] N. Siddique, S. Paheding, C. P. Elkin, y V. Devabhaktuni, "U-net and its variants for medical image segmentation: A review of theory and applications", *IEEE Access*, pp. 1–28, 2021, doi: 10.1109/ACCESS.2021.3086020.

[30] S. Ioffe y C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 1, pp. 448–456, 2015.

[31] D. P. Kingma y J. Lei Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION", *Iclr*, pp. 1–15, 2015, [En línea]. Disponible en: https://arxiv.org/pdf/1412.6980.pdf %22 entire document

[32] J. C. Duchi, P. L. Bartlett, y M. J. Wainwright, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Proc. IEEE Conf. Decis. Control*, vol. 12, pp. 5442–5444, 2012, doi: 10.1109/CDC.2012.6426698.

[33] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures", *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7700 LECTU, pp. 437–478, 2012, doi: 10.1007/978-3-642-35289-8_26.

[34] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)", núm. 1, pp. 2–8, 2018, [En línea]. Disponible en: http://arxiv.org/abs/1803.08375

[35] IBM, "R2", 2023. https://www.ibm.com/docs/es/cognos-analytics/11.1.0?topic=terms-r2

[36] R. Nisar, "(Visually) Interpreting the confusion-matrix", 2020. https://medium.com/analytics-vidhya/visually-interpreting-the-confusion-matrix-787a70b65678

[37] S. Narkhede, "Understanding Confusion Matrix", 2018. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

[38] M. Tkachenko, M. Malyuk, A. Holmanyuk, y N. Liubimov, "Label Studio: Data Labeling Software". 2020. [En línea]. Disponible en: https://github.com/heartexlabs/label-studio

[39] B. Dwyer, J. Nelson, y J. Solawetz, "Roboflow", 2022, [En línea]. Disponible en: https://roboflow.com/

[40] A. Jung, "Imgaug", 2018. https://github.com/aleju/imgaug

[41] Dk14, "Why is the validation accuracy fluctuating?" https://stats.stackexchange.com/questions/255105/why-is-the-validation-accuracy-fluctuating

[42] I. T. WARRIOR, "Reasons for Fluctuations in Loss During Training", [En línea]. Disponible en: https://indiantechwarrior.com/why-does-the-loss-accuracy-

fluctuate-during-the-training/

[43] B. Ke, A. Obukhov, S. Huang, N. Metzger, R. C. Daudt, y K. Schindler, "Repurposing Diffusion-Based Image Generators for Monocular Depth Estimation", 2023, [En línea]. Disponible en: http://arxiv.org/abs/2312.02145

[44] N. Silberman, D. Hoiem, P. Kohli, y R. Fergus, "Indoor segmentation and support inference from RGBD images", *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7576 LNCS, núm. PART 5, pp. 746–760, 2012, doi: 10.1007/978-3-642-33715-4_54.

[45] R. Holzer, "Tk-Tutorial-Readthedocs-Io-En-Latest". 2020. [En línea]. Disponible en: https://github.com/rasql/tk-tutorial/blob/master/docs/index.rst

[46] D. B. Dos Santos Cesar, C. Gaudig, M. Fritsche, M. A. Dos Reis, y F. Kirchner, "An evaluation of artificial fiducial markers in underwater environments", *MTS/IEEE Ocean. 2015 - Genova Discov. Sustain. Ocean Energy a New World*, núm. May, 2015, doi: 10.1109/OCEANS-Genova.2015.7271491.

[47] A. Kirillov, E. Mintun, N. Ravi, H. Mao, y C. Rolland, "SAM", 2023. https://segment-anything.com/

[48] S. Lee, L. M. Chang, y M. Skibniewski, "Automated recognition of surface defects using digital color image processing", *Autom. Constr.*, vol. 15, núm. 4, pp. 540–549, jul. 2006, doi: 10.1016/J.AUTCON.2005.08.001.

[49] M. R-cnn, P. Doll, y R. Girshick, "Mask R-CNN".

[50] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, y A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, núm. 4, pp. 834–848, 2018, doi: 10.1109/TPAMI.2017.2699184.

[51] Y.-T. Huang, M. R. Jahanshahi, F. Shen, y T. G. Mondal, "Deep Learning–Based Autonomous Road Condition Assessment Leveraging Inexpensive RGB and Depth Sensors and Heterogeneous Data Fusion: Pothole Detection and Quantification", *J. Transp. Eng. Part B Pavements*, vol. 149, núm. 2, pp. 1–18, 2023, doi: 10.1061/jpeodx.pveng-1194.

ANEXOS

Anexo 1: Materiales y pasos para la recolección de data

Materiales

- Iluminaria aro 10’’
- Wincha
- Cinta
- Papel
- Cuadro de escala

Paso 1. Ubicación e iluminación de zona dañada

Paso 2. Limpieza y pegado de cuadro de escala

Paso 3. Toma de fotos

Paso 4. Medición con wincha

Anexo 2: Capturas del código principal Sección 4.2.3

```python
import os
from Unet.unet_model_functions import build_unet
from ultralytics import YOLO
from natsort import natsorted
import cv2
from PIL import Image
import numpy as np
from matplotlib import pyplot as plt
import tensorflow as tf
import random
from math import sqrt
import pandas as pd
```

```python
import keras.layers
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate, Conv2DTranspose, BatchNormalization, Dropout, Lambda
from keras.optimizers import Adam
from keras.layers import Activation, MaxPool2D, Concatenate


def conv_block(input, num_filters):...

def encoder_block(input, num_filters):...

def decoder_block(input, skip_features, num_filters):
    s = Conv2DTranspose(num_filters, (2, 2), strides=2, padding="same")(input)
    s = Concatenate()([s, skip_features])
    s = conv_block(s, num_filters)
    return s

def build_unet(input_shape):
    inputs = Input(input_shape)

    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    b1 = conv_block(p4, 1024) #Bridge

    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

    outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(d4)

    model = Model(inputs, outputs, name="U-Net")
    return model
```

```python
from Unet.unet_model_functions import build_unet
import os
from natsort import natsorted
import cv2
from PIL import Image
import numpy as np
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.metrics import Recall, Precision
from keras.losses import BinaryCrossentropy
import tensorflow as tf
from math import sqrt
import datetime
import random

#Actual code

image_directory = 'C:/Users/Usuario/Downloads/drive-download-20230417T051939Z-001/images'
mask_directory = 'C:/Users/Usuario/Downloads/drive-download-20230417T051939Z-001/masks'

image_dataset = []
mask_dataset = []

images = natsorted(os.listdir(image_directory))
```

```python
for i, image_name in enumerate(natsorted(images)):
    if (image_name.split('.')[1] == 'jpg'):
        #print(image_directory+image_name)
        image = cv2.imread(image_directory+"/"+image_name)
        image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        image = Image.fromarray(image)
        w, h = image.size
        if w == 256:
            image = image.transpose(Image.ROTATE_90)
        image = image.resize((192, 256))
        image_dataset.append(np.array(image))

masks = natsorted(os.listdir(mask_directory))
for i, image_name in enumerate(natsorted(masks)):
    if (image_name.split('.')[1] == 'png'):
        image = cv2.imread(mask_directory+"/"+image_name)
        image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        image = Image.fromarray(image)
        w, h = image.size
        if w == 256:
            image = image.transpose(Image.ROTATE_90)
        image = image.resize((192, 256))
        mask_dataset.append(np.array(image))


#Normalize images
image_dataset = np.array(image_dataset)
#D not normalize masks, just rescale to 0 to 1.
mask_dataset = np.array(mask_dataset) /255.

#Dividir la data
X_train, X_test, y_train, y_test = train_test_split(image_dataset, mask_dataset, test_size=0.15, random_state=0)
```

```
image_number = random.randint(0, len(X_test))
print(len(X_test))

a = np.reshape(X_test[image_number], (256, 192, 3))
b = np.reshape(y_test[image_number], (256, 192, 1))

plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(a, cmap="gray")
plt.subplot(122)
plt.imshow(b, cmap="gray")
plt.show()

#INPUT SHAPE

IMG_HEIGHT = image_dataset.shape[1]
IMG_WIDTH = image_dataset.shape[2]
IMG_CHANNELS = image_dataset.shape[3]

input_shape = (IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)

#CALLING FOR THE MODEL
model = build_unet(input_shape)
model.compile(optimizer=Adam(learning_rate=0.001), loss=BinaryCrossentropy(), metrics=["accuracy", Precision(), Recall()])
model.summary()

#Final setup
batch_size = 102

steps_per_epoch = ((len(X_train)) // batch_size)
validation_steps= (len(X_test)// batch_size)

# checkpoint
filepath="PESOS/weights_Square.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

callbacks_list = [checkpoint]
```

```
#Empezar a entrenar
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=batch_size,
                    steps_per_epoch=steps_per_epoch,
                    validation_steps=validation_steps,
                    epochs=200,
                    callbacks=callbacks_list)
model.save("PESOS/SquareDet.h5")


# plot the training and validation accuracy and loss at each epoch

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

acc = history.history['accuracy']
# acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
# val_acc = history.history['val_accuracy']

plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
#########################################################################################################################
# Cargar los pesos de un modelo

model.load_weights('C:/Users/Usuario/Downloads/weights_square.best_2.hdf5')

#cargar un modelo guardado
new_model = tf.keras.models.load_model('Unet/modelo_516_b1_e50')

# Check its architecture
model.summary()

# Evaluate the restored model
loss, acc = model.evaluate(X_test, y_test, verbose=2)

print('Restored model, accuracy: {:5.2f}%'.format(100 * acc))

#Una prediccion

N7 = tf.keras.models.load_model('C:/Users/Usuario/Downloads/SpallDet_0404.h5')
N8 = tf.keras.models.load_model('C:/Users/Usuario/Downloads/SpallDet_1104.h5')
N9 = tf.keras.models.load_model('C:/Users/Usuario/Downloads/SpallDet_1804.h5')
N10 = tf.keras.models.load_model('C:/Users/Usuario/Downloads/SpallDet_2904.h5')

test_img_number = random.randint(0, len(X_test)-1)
test_img = np.reshape(X_test[test_img_number], (1,256, 192, 3))
ground_truth = y_test[test_img_number]

predictionN7 = N7.predict(test_img)
predictionN7 = np.reshape(predictionN7, (256, 192, 1))

predictionN8 = N8.predict(test_img)
predictionN8 = np.reshape(predictionN8, (256, 192, 1))

predictionN9 = N9.predict(test_img)
predictionN9 = np.reshape(predictionN9, (256, 192, 1))

predictionN10 = N10.predict(test_img)
predictionN10 = np.reshape(predictionN10, (256, 192, 1))
```

```
plt.figure(figsize=(14, 3))
plt.subplot(261)
plt.title('Imagen Original')
test_img = X_test[test_img_number]
plt.imshow(test_img, cmap='gray')
plt.subplot(262)
plt.title('Anotación real')
plt.imshow(ground_truth, cmap='gray')
plt.subplot(263)
plt.title('Predicción N7')
plt.imshow(predictionN7, cmap='gray')
plt.subplot(264)
plt.title('Predicción N8')
plt.imshow(predictionN8, cmap='gray')
plt.subplot(265)
plt.title('Predicción N9')
plt.imshow(predictionN9, cmap='gray')
plt.subplot(266)
plt.title('Predicción N10')
plt.imshow(predictionN10, cmap='gray')
plt.show()

#########################################################################################################################

#Hacer una prediccion con una imagen random del repositorio

test_img_number = random.randint(0, len(X_test)-1)
test_img = np.reshape(X_test[test_img_number], (1, 256, 192, 3))

test_img_number = random.randint(0, len(X_test)-1)
test_img2= np.reshape(X_test[test_img_number], (1, 256, 192, 3))
ground_truth = np.reshape(y_test[test_img_number], (256, 192))

prediction = model.predict(test_img)
prediction2 = np.reshape(prediction, (256, 192, 1))

number_of_white_pix1 =(prediction2 > 0.9).sum()   # extracting only white pixels
```

```
Escala_Lado_cuadrado1= sqrt(number_of_white_pix1)

print("Area cubierta por la escala " + str(number_of_white_pix1))
print("Lado equivalente a 2 cm en la realidad " + str(Escala_Lado_cuadrado1))

new_prediction = model.predict(test_img2)
new_prediction2 = np.reshape(new_prediction, (256, 192, 1))

number_of_white_pix2 = (new_prediction2 > 0.9).sum()
Escala_Lado_cuadrado2= sqrt(number_of_white_pix1)

print("Area cubierta por la escala " + str(number_of_white_pix2))
print("Lado equivalente a 2 cm en la realidad " + str(Escala_Lado_cuadrado2))

plt.figure(figsize=(16, 8))
plt.subplot(231)
plt.title('Testing Image')
test_img = np.reshape(X_test[test_img_number], (256, 192, 3))
plt.imshow(test_img, cmap='gray')
plt.subplot(232)
plt.title('Testing Label')
plt.imshow(ground_truth, cmap='gray')
plt.subplot(233)
plt.title('Prediction on test image')
plt.imshow(prediction2, cmap='gray')

plt.subplot(234)
plt.title('Testing Image')
test_img2 = np.reshape(X_test[test_img_number], (256, 192, 3))
plt.imshow(test_img2, cmap='gray')
plt.subplot(235)
plt.title('Testing Label')
plt.imshow(ground_truth, cmap='gray')
plt.subplot(236)
plt.title('Prediction on test image')
plt.imshow(new_prediction2, cmap='gray')
```

```
#out Unet

Outputs = []

for i in range(len(image_dataset)):
    Img= np.reshape(image_dataset[i], (1, 256, 192, 3))
    PEscala_out = np.reshape(M_escala.predict(Img), (256, 192, 1))
    PUnet_out = np.reshape(N10.predict(Img), (256, 192, 1))

    number_of_white_pix_out = (PEscala_out > 0.5).sum()  # extracting only white pixels

    RazonEscala_out = (4 / number_of_white_pix_out)

    number_of_pix_Unet_out = (PUnet_out > 0.9).sum()

    Area_Unet_CM_out = number_of_pix_Unet_out * RazonEscala_out
    D_Unet_CM_out = sqrt(4*Area_Unet_CM_out/3.1416)

    Outputs.append(D_Unet_CM_out)

dict = {'Diametro predecido': Outputs, 'nombre': images}
df= pd.DataFrame(dict)

df.to_csv('Predicciones Unet.csv')
```

```
#out YOLO

Outputs1 = []
Prediccion_F = image_dataset.copy()

for i in range(len(image_dataset)):
    Img= np.reshape(image_dataset[i], (1, 256, 192, 3))
    PEscala_out = np.reshape(M_escala.predict(Img), (256, 192, 1))
    number_of_white_pix_out = (PEscala_out > 0.5).sum()  # extracting only white pixels
    RazonEscala_out = (4 / number_of_white_pix_out)
    results_1 = model(Prediccion_F[i], conf=0.1)
    boxes_1 = results_1[0].boxes

    if len(boxes_1)==0:...

    box = boxes_1[0]  # returns one box
    p1 = int(box.xyxy[0][0].item())
    p2 = int(box.xyxy[0][1].item())
    q1 = int(box.xyxy[0][2].item())
    q2 = int(box.xyxy[0][3].item())

    M1 = int((p1 + q1) / 2)
    M2 = int((p2 + q2) / 2)
    Dist = abs((p1 - q1) / 2)
    pt = (M1, M2)

    # Dibujar el circulo
    im = Prediccion_F[i]
    mask = np.zeros_like(im)
    mask = cv2.circle(mask, pt, int(Dist), thickness=-1, color=(255, 255, 255))
    result = cv2.bitwise_and(im, mask)

    hsv = cv2.cvtColor(result, cv2.COLOR_BGR2HSV)
```

```
    lower = (0, 100, 100)
    upper = (230, 255, 255)
    mask_1 = cv2.inRange(hsv, lower, upper)

    count_YOLO = np.count_nonzero(mask_1)

    Area_YOLO_CM_out = count_YOLO * RazonEscala_out
    D_YOLO_CM_out = sqrt(4*Area_YOLO_CM_out / 3.1416)

    print("Diametro en cm segun YOLO " + ("%.2f" % D_YOLO_CM_out))
    Outputs1.append(D_YOLO_CM_out)

dict1 = {'Diametro predecido': Outputs1, 'nombre': images}
df1= pd.DataFrame(dict1)

df1.to_csv('Predicciones YOLO usando area queso.csv')
```

```
def g_mean(x):
    x_new = [i for i in x if i != 0]
    if len(x_new) == 0:
        return 0
    a = np.log(x_new)
    return np.exp(a.mean())
```

```
#########################################################################################################################
Outputs1 = []
Outputs2 = []

for i in range(len(image_dataset)):
    test_img = np.reshape(image_dataset[i], (1, 256, 192, 3))
    IMG = np.reshape(image_dataset[i], (256, 192, 3))
    Segmentacion_Unet = np.reshape(N10.predict(test_img), (256, 192, 1))
    # Según un grado de confianza en la predicción, se toma como No es spall (0) y Sí es spall (1)
    Segmentacion_Unet[Segmentacion_Unet < 0.8] = 0
    Segmentacion_Unet[Segmentacion_Unet >= 0.8] = 1
    # Necesario para la manipulación con bitwise_not
    Segmentacion_Unet = Segmentacion_Unet.astype('uint8')

    Imagen_segmentada_Prediccion = cv2.bitwise_and(IMG, IMG, mask=Segmentacion_Unet)
    img_gray = cv2.cvtColor(Imagen_segmentada_Prediccion, cv2.COLOR_RGB2GRAY)

    gmeans = []
    gmeans2= []
    gmeans3= 0
    for i in range(len(img_gray)):
        s = g_mean(img_gray[i])
        gmeans.append(s)

    for i in range(len(img_gray)):
        s = np.zeros((256, 192))
        for m in range(len(img_gray[i])):
            if gmeans[i] > img_gray[i][m]:
                s[i][m] = (img_gray[i][m])
        x = g_mean(s[i])
        gmeans2.append(x)

    target = min([i for i in gmeans if i != 0])
    Fila = gmeans.index(target)
    Gray_profile = img_gray_3[Fila]
    m = max([i for i in Gray_profile if i != 0])
    geo_mean = g_mean([m, g_mean(Gray_profile)])
    target_min = min([i for i in Gray_profile if i != 0])
    diff = geo_mean - target_min
```

```
    gmeans3 = g_mean(gmeans2)
    Outputs1.append(gmeans3)
    Outputs2.append(diff)
print(Outputs1)
dict1 = {'Valor de intensidad de la base': Outputs1, "Diferencia entre intensidad mayor y menor: ":Outputs2}
df1= pd.DataFrame(dict1)

df1.to_csv('Valor de intensidad de la base y la diff de intensidades mayores y menores.csv')
```

Anexo 3: Interfaz – Código y otros ejemplos Sección 4.2.3.4

```python
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog
from PIL import Image, ImageTk
from natsort import natsorted
import cv2
import numpy as np
import os
from Unet.unet_model_functions import build_unet
import tensorflow as tf
from ultralytics import YOLO
from math import sqrt
```

```python
image_directory = ""
image_dataset = []
display = {}
npdataset = []

def indexarfotos():
    global image_directory
    global image_dataset
    global npdataset
    global display

    images = natsorted(os.listdir(image_directory))
    for i, image_name in enumerate(natsorted(images)):
        if (image_name.split('.')[1] == 'jpg'):
            # print(image_directory+image_name)
            image = cv2.imread(image_directory + "/" + image_name)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = Image.fromarray(image)
            w, h = image.size
            if w == 256:
                image = image.transpose(Image.ROTATE_90)
            image = image.resize((192, 256))
            image_dataset.append(np.array(image))
    npdataset = np.array(image_dataset)

    for i in range(len(npdataset)):
        display[i]= ImageTk.PhotoImage(image=Image.fromarray(npdataset[i]))
```

```python
def cargarmodelos():
    global image_dataset
    global N10
    global model
    global M_escala
    global npdataset

    # Cargar modelo Unet
    N10 = tf.keras.models.load_model('C:/Users/Usuario/Downloads/SpallDet_2904.h5')

    # Cargar modelo Yolo
    model = YOLO('Yolo/best.pt')

    # Cargar modelo Escala
    IMG_HEIGHT = npdataset.shape[1]
    IMG_WIDTH = npdataset.shape[2]
    IMG_CHANNELS = npdataset.shape[3]

    input_shape = (IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)

    M_escala = build_unet(input_shape)
    M_escala.load_weights('C:/Users/Usuario/Downloads/weights_square.best_2.hdf5')
```

```python
def browse_folder():
    global image_directory
    global N10
    global model
    global M_escala
    global npdataset

    image_directory = filedialog.askdirectory()
    indexarfotos()
    cargarmodelos()
```

```python
class tkinterApp(tk.Tk):

    def __init__(self, *args, **kwargs):

        tk.Tk.__init__(self, *args, **kwargs)

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand=True)

        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        for F in (StartPage, Page1, Page2):
            frame = F(container, self)

            self.frames[F] = frame

            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(StartPage)

    def show_frame(self, cont):
        frame = self.frames[cont]
        frame.tkraise()
```

```python
class StartPage(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        label = ttk.Label(self, text="Bienvenido", font=("Helvetica", 30, "bold"), justify="center")
        label2 = ttk.Label(self, text="Por favor seleccionar su carpeta con las imágenes de descascaramiento que tenga.", font=("Helvetica", 10), justify="left")
        label3 = ttk.Label(self, text="*Las fotos deberán estar en formato jpg y en 192x256.", font=("Helvetica", 10), justify="left")

        label.grid(row=0, column=1, padx=2, pady=10)
        label2.grid(row=1, column=1, padx=2, pady=5)
        label3.grid(row=2, column=1, padx=2, pady=5)

        button1 = ttk.Button(self, text="Escoger Carpeta", command=lambda: [browse_folder(), controller.show_frame(Page1)])

        button1.grid(row=3, column=1, padx=2, pady=10)
```

```python
class Page1(tk.Frame):
    global display
    global npdataset
    global N10  # Unet
    global model  # YOLO
    global M_escala  # Plantilla Unet

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        label = ttk.Label(self, text="Bienvenido", font=("Helvetica", 30, "bold"))
        label.grid(row=0, column=1, pady=10)

        button0 = ttk.Button(self, text="Preview",
                             command=lambda: show_image(0))
        button0.grid(row=1, column=1, padx=10, pady=10)

        canvas = tk.Canvas(self, width=192, height=256)

        def show_image(i):
            canvas.grid(row=2, column=1, padx=10)
            canvas.create_image(20, 20, anchor="nw", image=display[i])

        def forward(i):
            global button_forward
            global button_back

            canvas.create_image(20, 20, anchor="nw", image=display[i-1])
            button_forward = ttk.Button(self, text=">>", command=lambda: forward(i+1))
            button_back= ttk.Button(self, text="<<", command=lambda: back(i-1))

            if i == len(npdataset):
                button_forward = ttk.Button(self, text=">>", state="disabled")

            button_back.grid(row=3, column=0, padx=10, pady=10)
            button_forward.grid(row=3, column=2, padx=10, pady=10)
```

```
    def back(i):
        global button_forward
        global button_back

        canvas.create_image(20, 20, anchor="nw", image=display[i-1])
        button_forward = ttk.Button(self, text=">>", command=lambda: forward(i + 1))
        button_back = ttk.Button(self, text="<<", command=lambda: back(i - 1))

        if i == 1:
            button_back = ttk.Button(self, text="<<", state="disabled")

        button_back.grid(row=3, column=0, padx=10, pady=10)
        button_forward.grid(row=3, column=2, padx=10, pady=10)

    button_back= ttk.Button(self, text="<<", command=back, state="disabled")
    button_forward= ttk.Button(self, text=">>", command= lambda: forward(2))

    button_back.grid(row=3,column=0,padx=10,pady=10)
    button_forward.grid(row=3,column=2,padx=10,pady=10)

    button1 = ttk.Button(self, text="Analizar imágenes",
                         command=lambda: controller.show_frame(Page2))
    button1.grid(row=4, column=1, padx=10, pady=10)

    button2 = ttk.Button(self, text="Volver a inicio",
                         command=lambda: controller.show_frame(StartPage))
    button2.grid(row=5, column=1, padx=10, pady=10)
```

```python
class Page2(tk.Frame):
    global display
    global npdataset
    global N10  # Unet
    global model  # YOLO
    global M_escala  # Plantilla Unet

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        global ImagenAnalizada
        global npdataset
        global N10  # Unet
        global model  # YOLO
        global M_escala  # Plantilla Unet

        ImagenAnalizada = []
        def Implementacion():

            global npdataset
            global N10  # Unet
            global model  # YOLO
            global M_escala  # Plantilla Unet
```

```python
        Results = []
        for c in range(len(npdataset)):
            var = [0,0,0,0]

            test_img = np.reshape(npdataset[c], (1, 256, 192, 3))

            # Plantilla
            PEscala = np.reshape(M_escala.predict(test_img), (256, 192, 1))
            number_of_white_pix = (PEscala > 0.5).sum()  # Pixeles de la plantilla
            RazonEscala_3 = (4 / number_of_white_pix)

            # Unet
            PUnet = np.reshape(N10.predict(test_img), (256, 192, 1))  # Segmentación Unet
            count_Unet = (PUnet > 0.9).sum()
            PUnet = np.squeeze(PUnet, axis=2)
            var[0] = ImageTk.PhotoImage(image=Image.fromarray((PUnet * 255).astype(np.uint8))) #Unet_Seg

            # YOLO
            results = model(npdataset[c], conf=0.008, classes=1)
            yolo_mask = results[0].masks.numpy().masks[0]  # Segmentación YOLO
            count_YOLO2 = np.count_nonzero(yolo_mask)
            var[1] = ImageTk.PhotoImage(image=Image.fromarray((yolo_mask).astype(bool))) #YOLO_Seg

            # Diametro Unet
            count_Unet_areaencm2 = RazonEscala_3 * count_Unet
            var[2] = sqrt(4 * count_Unet_areaencm2 / 3.15) * 10 #D_mm_Unet

            # Diametro Yolo
            count_YOLO_areaencm2_2 = RazonEscala_3 * count_YOLO2
            var[3] = sqrt(4 * count_YOLO_areaencm2_2 / 3.15) * 10 #D_mm_YOLO

            Results.append(var)

        #Resultados = Implementacion()
        #Resultados = [[Unet_seg,YOLO_Seg,D_mm_Unet,D_mm_YOLO],...]
        return Results
```

```
    def AHHH():
        global ImagenAnalizada
        ImagenAnalizada = Implementacion()

    label = ttk.Label(self, text="Bienvenido", font=("Helvetica", 30, "bold"))
    label.grid(row=0, column=1, pady=10)

    button0 = ttk.Button(self, text="Iniciar",
                         command=lambda: [AHHH(),show_image(0,0),button0.destroy()])
    button0.grid(row=1, column=1, padx=10, pady=10)

    canvas = tk.Canvas(self, width=192, height=256)
    canvas2 = tk.Canvas(self, width=192, height=256)
    canvas3 = tk.Canvas(self, width=192, height=256)

    def Indicededano(i,a, U):
        global ImagenAnalizada

        if ImagenAnalizada[i - a][U] < 150:
            damage = "Leve"
        elif ImagenAnalizada[i - a][U] > 250:
            damage = "Severo"
        else:
            damage = "Moderado"
        return damage
```

```
def show_image(i, a):

    global ImagenUnet
    global ImagenYOLO
    global Categoria_UNET
    global Categoria_YOLO

    #Resultados = [[Unet_seg,YOLO_Seg,D_mm_Unet,D_mm_YOLO],...]

    #Unet_seg= Resultados[i][0]
    #YOLO_seg= Resultados[i][1]
    #D_mm_Unet= Resultados[i][2]
    #D_mm_YOLO= Resultados[i][3]

    canvas.grid(row=2, column=0)
    canvas.create_image(20, 20, anchor="nw", image=display[i - a])

    canvas2.grid(row=2, column=1)
    canvas2.create_image(20, 20, anchor="nw", image=ImagenAnalizada[i - a][0]) #Unet_Seg

    canvas3.grid(row=2, column=2)
    canvas3.create_image(20, 20, anchor="nw", image=ImagenAnalizada[i - a][1]) #YOLO_Seg

    ImagenUnet = ttk.Label(self, text="Resultado Unet: " + str(round(ImagenAnalizada[i - a][2],2)) + "mm", font=("Helvetica", 11)) #D_mm_Unet
    ImagenUnet.grid(row=3, column=1, padx=10, pady=10)

    ImagenYOLO = ttk.Label(self, text="Resultado YOLO: " + str(round(ImagenAnalizada[i - a][3],2)) + "mm", font=("Helvetica", 11)) #YOLO_seg
    ImagenYOLO.grid(row=3, column=2, padx=10, pady=10)

    Categoria_UNET = ttk.Label(self, text="Categoria: " + Indicededano(i,a,2), font=("Helvetica", 11)) #Unet
    Categoria_UNET.grid(row=4, column=1, padx=10, pady=10)

    Categoria_YOLO = ttk.Label(self, text="Categoria: " + Indicededano(i,a,3), font=("Helvetica", 11)) #YOLO
    Categoria_YOLO.grid(row=4, column=2, padx=10, pady=10)
```

```
def forward(i):
    global button_forward
    global button_back
    global ImagenUnet
    global ImagenYOLO
    global Categoria_UNET
    global Categoria_YOLO

    canvas.grid_forget()
    canvas2.grid_forget()
    canvas3.grid_forget()
    ImagenUnet.grid_forget()
    ImagenYOLO.grid_forget()
    Categoria_UNET.grid_forget()
    Categoria_YOLO.grid_forget()

    show_image(i, 1)

    button_forward = ttk.Button(self, text=">>", command=lambda: forward(i + 1))
    button_back = ttk.Button(self, text="<<", command=lambda: back(i - 1))

    if i == len(npdataset):
        button_forward = ttk.Button(self, text=">>", state="disabled")

    button_back.grid(row=5, column=0, padx=10, pady=10)
    button_forward.grid(row=5, column=2, padx=10, pady=10)
```

```
def back(i):
    global button_forward
    global button_back
    global ImagenUnet
    global ImagenYOLO
    global Categoria_UNET
    global Categoria_YOLO

    canvas.grid_forget()
    canvas2.grid_forget()
    canvas3.grid_forget()
    ImagenUnet.grid_forget()
    ImagenYOLO.grid_forget()
    Categoria_UNET.grid_forget()
    Categoria_YOLO.grid_forget()

    show_image(i, 1)

    button_forward = ttk.Button(self, text=">>", command=lambda: forward(i + 1))
    button_back = ttk.Button(self, text="<<", command=lambda: back(i - 1))

    if i == 1:
        button_back = ttk.Button(self, text="<<", state="disabled")

    button_back.grid(row=5, column=0, padx=10, pady=10)
    button_forward.grid(row=5, column=2, padx=10, pady=10)
```

```
        ImagenOriginal = ttk.Label(self, text="Imagen Original", font=("Helvetica", 11))
        ImagenOriginal.grid(row=3, column=0, padx=10, pady=10)

        ImagenUnet = ttk.Label(self, text="Unet", font=("Helvetica", 11))
        ImagenUnet.grid(row=3, column=1, padx=10, pady=10)

        ImagenYOLO = ttk.Label(self, text="YOLO", font=("Helvetica", 11))
        ImagenYOLO.grid(row=3, column=2, padx=10, pady=10)

        button_back = ttk.Button(self, text="<<", command=back, state="disabled")
        button_forward = ttk.Button(self, text=">>", command=lambda: forward(2))

        button_back.grid(row=5, column=0, padx=10, pady=10)
        button_forward.grid(row=5, column=2, padx=10, pady=10)

        button2 = ttk.Button(self, text="Volver a inicio",
                             command=lambda: controller.show_frame(StartPage))
        button2.grid(row=6, column=1, padx=10, pady=10)

app = tkinterApp()
tk.Wm.wm_title(app, "Spalling Measure")
app.mainloop()
```