UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA CIVIL

MODELO PARA DETECCIÓN DE GRIETAS INCIPIENTES EN VIGAS DE HORMIGÓN ARMADO EN BASE A DEEP LEARNING

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL

RAÚL IGNACIO HENRÍQUEZ SANDOVAL

PROFESOR GUÍA:

RUBÉN BOROSCHEK KRAUSKOPF

MIEMBROS DE LA COMISIÓN:

ENRIQUE LOPEZ DROGUETT

RAFAEL AGUILAR VELEZ

SANTIAGO DE CHILE

2020

RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE: INGENIERO CIVIL CON MENCIÓN EN ESTRUCTURAS, CONSTRUCCIÓN Y GEOTECNIA.
POR: RAÚL HENRÍQUEZ SANDOVAL
PROFESOR GUÍA: RUBEN BOROSCHEK KRAUSKOPF

"MODELO PARA DETECCIÓN DE GRIETAS INCIPIENTES EN VIGAS DE HORMIGÓN ARMADO EN BASE A DEEP LEARNING"

La motivación principal de este trabajo de título nace de la necesidad por parte del área de ingeniería civil de incorporar las novedades que trae la *Ciencia de Datos*, ya que con algoritmos de *Aprendizaje Profundo* complementados con principios de *Visión por Computadora* se pueden construir herramientas que pueden llegar a ser de bastante utilidad en el área de diagnóstico de estados de salud de estructuras. Al mismo tiempo, el desarrollo de nuevas cámaras fotográficas de tamaño reducido y alta calidad en la imagen ha impulsado el uso de pequeños vehículos aéreos no tripulados para el monitoreo de la salud estructural de diversas obras civiles, lo cual propicia el desarrollo de nuevas metodologías para los procesos de detección y caracterización de fallas en elementos estructurales.

El principal objetivo de esta tesis es la confección de un software que permita la detección de grietas incipientes en vigas de hormigón armado, para esto, es necesario contar con una gran cantidad de imágenes de vigas de hormigón armado con grietas y sin grietas, es por esta razón que se realizan ensayos destructivos en vigas de hormigón armado en el Laboratorio de Sólidos presente en la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, los cuales son grabados en condiciones lo más cercano a lo ideal, para capturar de manera óptima el fenómeno y de esta forma contar con este set de imágenes. También se busca una correcta implementación de técnicas tales como *Aumento de datos* y *Aprendizaje por Transferencia*.

Para llevar a cabo este trabajo se comienza realizando los ensayos de las vigas de hormigón armado con la finalidad de obtener los registros de estos ensayos, haciendo factible formar un *Conjunto de Datos* balanceado y robusto. Acto seguido, se realiza un preprocesamiento al *Conjunto de Datos* generado, para luego definir la arquitectura del modelo a utilizar en base a *Aprendizaje Profundo*, para posteriormente realizar el entrenamiento e implementación del modelo. Posteriormente, se analizan y verifican los resultados de la clasificación realizada por el modelo. En caso de que estos no sean satisfactorios se procede a cambiar *hiper-parámetros* de la arquitectura del modelo o el modelo en sí mismo y se repite el procedimiento.

La principal conclusión de este trabajo es que se pueden obtener buenos resultados en variedad de modelos convolucionales, particularmente, el modelo convolucional al cual se llega mediante la *Búsqueda de Cuadrícula* es el que posee mejores resultados en cuanto a detección de grietas, contando además dentro de sus características una buena estabilidad, por lo que es considerado el mejor modelo al cual se logró llegar en esta investigación, dejando atrás a modelos que utilizan *Aprendizaje por Transferencia*.

Dedicatoria

A mis padres, a mi hermana, a mi familia, a mis amigas y amigos que marcaron mi camino, por confiar en mí y por su apoyo incondicional.

Agradecimientos

En este trabajo que da fin a mi etapa universitaria no puedo dejar de agradecer a todas las personas que han estado conmigo en este proceso, ya sea con su apoyo y ayuda o simplemente con su compañía para hacer más llevadero el día a día en la universidad.

En primer lugar, quiero agradecerle a mi familia, padres y hermana, que me han apoyado en todo lo que me he propuesto, quienes me incentivaron desde pequeño en mis estudios y a cumplir mis metas, para lo cual han aguantado mis enojos y momentos de estrés durante todos estos años, también agradecerles por su esfuerzo y sacrificio con la finalidad de darme la oportunidad de estudiar en Santiago y bridarme el mejor pasar universitario que pude tener.

También quisiera agradecer a mis amigos, ya que de cierta forma todos han aportado a mi llegada hasta este punto. Comenzar agradeciendo a mis amigos que me han acompañado desde la etapa del colegio y con los cuales sigo manteniendo contacto, a mis Perros Bomba con los que comparto incontables anécdotas. Agradecer a Indio, eterno compañero; a Durlan, uno de mis descarriados favorito; a Tata, mi guía espiritual; a Nicolas, el hijo del diablo; a King, distinguido miembro de la realeza; a Constanza, la loca del pan; a Hugo, mi médico de cabecera; a Panchote, incontrolable personaje; a Zepeda se busca, se encuentra; a Jorge, dueño totalitario de Pucón; a Jerónimo, mi amarillo genio; a Rivas, que mantiene el control del balón; a Diego T compañero de batallas épicas, gracias totales.

Por otra parte, están los amigos que me acompañaron y fueron parte de mi camino dentro de la universidad, comenzando con las amistades que forje en los primeros años formando a Los Perrines de Beauchef, lo que desemboco en la creación del Club Social y Deportivo Porro Seco, posterior a mi ingreso a la especialidad, coseché nuevas y lindas amistades en la comunidad de civil, reafirmando estos nuevos lazos con la formación del Club Social y Deportivo Sapo Diablo.

Dentro de estas dos comunidades de amistad debo agradecer de manera especial a José Pepe Petaca Benavides por su precisión a la hora de pedir cambio; a DC9 por su permanente locura y enfermedad; a Jorge H por hablar en el mismo idioma que yo; a Negrita por sus buenos consejos y linda personalidad; a Tindersihy por su cariño y su tiempo, a Rolex por sus visitas dominicales; a Vicente y su devoción por el Cacique; a Alexis por sus participación en las misas de los viernes con el querido Robertiwi; a Boris, acérrimo fan de Felipe Avello; a las Vicky´s y a Camila por su eterno apañe; a Shakyri, fallido compañero de gimnasio, muchas gracias.

Agradecer los consejos y cariños de mis tíos, tías, primos y primas, agregando a la comunidad Chillaneja en Santiago por su buena y cálida compañía.

Estaré eternamente agradecido de todos los que colaboraron para ser la persona que soy.

Tabla de Contenido

Índice de tablas

Índice de ilustraciones

1. Introducción

1.1. Motivación

En los últimos años la popularidad y el uso de los algoritmos basados en *Aprendizaje Profundo* ha aumentado debido a la multitud de campos en los que es posible aplicarlos y a los buenos resultados que ofrece esta herramienta.

El principal objetivo de los algoritmos de *Aprendizaje Profundo* es conseguir realizar tareas que un humano desempeñaría de forma casi automática pero que llegan a ser complejas para una máquina. Un ejemplo sería, poder detectar e identificar todos los elementos de una imagen. Hoy en día el *Aprendizaje Profundo* está considerado como el mejor clasificador de imágenes y representa el estado del arte en *Visión por Computadora*, siendo estos algoritmos los más utilizados actualmente y el principal objeto de investigación en dicho campo.

En la actualidad se hace uso de *Aprendizaje Profundo* en un gran número de aplicaciones que son usadas en el día a día como por ejemplo el traductor de Google, en asistentes virtuales como Siri, Cortana y Google Assistant, los cuales utilizan algoritmos de *Aprendizaje Profundo* para el reconocimiento de voz, clasificación de correos electrónicos e incluso para sistemas de seguridad que hagan uso de reconocimiento facial. Otra de las áreas donde se aplica *Aprendizaje Profundo*, es en algo tan complejo como los vehículos autónomos, los cuales cada día están más cerca de convertirse en una realidad.

Dada la gran versatilidad de este conjunto de algoritmos de aprendizaje, y la escaza proximidad de la Ingería Civil con este tipo de novedades, es que nace el interés de que, por medio de este trabajo, se incorporen algunos de los avances que trae la *Ciencia de Datos* al área de la construcción, puesto que con la ayuda de *Visión por Computadora* en conjunto con *Aprendizaje Profundo* se pueden lograr cosas que pueden llegar a ser de bastante utilidad en el área de clasificación y diagnóstico de estado de estructuras.

En este trabajo se desarrolla un software que permite la identificación de grietas incipientes en vigas de hormigón armado, material por excelencia de diversas construcciones en el país.

Esta solución nace en respuesta a la problemática de los métodos actuales de caracterización, los cuales son manuales y toman mucho tiempo al operario. Por lo tanto, este trabajo de titulación propone una solución con herramientas de procesamiento de imágenes, la cual necesita de muy poca interacción con el usuario, ahorrando tiempo y costos económicos.

1.2. Objetivos

El objetivo principal del trabajo es la confección de un software que permita la identificación de grietas incipientes en vigas de hormigón armado, para esto, es necesario contar con una gran cantidad de imágenes de vigas de hormigón armado con grietas y sin grietas, es por esta razón que se realizan ensayos destructivos en vigas de hormigón armado en el Laboratorio de Sólidos presente en la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, los cuales son grabados en condiciones lo más cercano a lo ideal, para capturar en video de manera óptima el fenómeno y de esta forma generar este set de imágenes.

También se busca una correcta implementación de técnicas tales como *Aumento de datos* y *Aprendizaje por Transferencia*.

1.3. Metodología

Para llevar a cabo este trabajo se comienza realizando los ensayos de las vigas de hormigón armado con la finalidad de obtener los registros de estos ensayos, haciendo factible formar un *Conjunto de Datos* balanceado y robusto.

Acto seguido, se define la arquitectura del modelo a utilizar en base a *Aprendizaje Profundo* para posteriormente realizar el entrenamiento e implementación del modelo.

Luego, se analizan y verifican los resultados de la clasificación realizada por el modelo. En caso de que estos no sean satisfactorios se procede a cambiar hiper-parámetros de la arquitectura del modelo o el modelo en sí, para luego repetir el procedimiento.

1.4. Alcances por capítulo

1.4.1. Introducción

Se presenta la motivación del presente trabajo, los objetivos, y la metodología usada.

1.4.2. Estado del arte

Se exhiben las características de distintas metodologías actuales para la detección de grietas.

1.4.3. Marco teórico

Se expone la teoría que constituye la base donde se sustentan los algoritmos y técnicas utilizadas.

1.4.4. Experiencia en laboratorio

Se detallan los procedimientos realizados en laboratorio.

1.4.5. Identificación de grietas y métodos

Se presenta detalladamente el paso a paso realizado para la confección del software.

1.4.6. Resultados

Se exponen los resultados de los distintos modelos generados.

1.4.7. Análisis de resultados

Se analizan los resultados de los distintos modelos generados.

1.4.8. Conclusiones

Se presentan las conclusiones.

2. Estado del Arte: Detección de grietas en la actualidad

2.1. Utilidad en el Análisis Estructural

La segmentación de grietas en concreto es una tarea importante en temas de estructuras, debido a que estas pueden provocar problemas graves sin la debida supervisión. Gran parte de los trabajos previos del mismo tema orientan su enfoque a detección de grietas de manera manual y no automática, por lo que se plantea como solución utilizar una red neuronal de aprendizaje automático, de manera de realizar una clasificación y segmentación de grietas con imágenes recibidas desde la industria. Como principal motivación se tiene que la detección de grietas es una parte importante del trabajo de mantenimiento de puentes, de manera de poder evaluar su estado de salud y garantizar su seguridad. Normalmente este trabajo se realiza por personal de ingeniería mediante inspección visual, un trabajo de gran dificultad y que requiere mucho tiempo, se debe agregar a esto la subjetividad que agrega la participación de personal humano.

2.2. Inspección Visual

Este método es utilizado como base de referencia para los otros métodos más sofisticados. Siendo así, el más empleado por su sencillez y economía. Es no destructivo por excelencia, ya que el agente físico es la luz, la cual no produce ningún daño a los materiales evaluados. Este método permite evaluar la cantidad de fisuras, su tamaño, su forma o configuración, su acabado superficial, su color, etc.

El procedimiento que se sigue en este método consta de la iluminación del objeto, la inspección por visión ocular directa y/o ocular con ayuda de medios auxiliares.

Las desventajas que conlleva este método son la dependencia de la habilidad y la experiencia del operario, además, de la duración de la inspección, la cual depende de la fisuración y su posición en la obra civil o elemento estructural.

Por otro lado, no se requiere de gran entrenamiento para poder realizar una inspección visual correcta.

2.3. Inspección por Líquidos Penetrantes

El objetivo principal de este método es detectar discontinuidades, aberturas en la superficie como fisuras, porosidades y traslapes.

Para llevar a cabo este ensayo se requiere un solvente limpiador, un tinte o líquido penetrante y un revelador (suspensión de polvo en líquido). Estos agentes varían según el líquido penetrante (LP).

El procedimiento de este método comienza con la limpieza inicial de la zona de interés, que permite la eliminación de agentes extraños a la medición. Luego, se aplica el LP y se debe esperar el tiempo de penetración para poder remover el exceso de LP sin extraer el líquido que se encuentra dentro de la anomalía.

Finalmente, se aplica el revelador para comenzar la inspección y evaluación.

La desventaja de este método recae en los agentes, ya que deben ser los adecuados puesto que en caso contrario se puede dañar el material. Además, es un método invasivo [8].

2.4. Métodos Automatizados

Dada la premisa de que las fisuras pueden convertirse en anomalías que amenacen la integridad estructural de una edificación, existen diversos estudios en el monitoreo continuo y a tiempo real de las fisuras.

En la literatura científica, el número de artículos publicados recientemente sobre detección de grietas y caracterización del tipo de grietas muestra un interés creciente en esta área. En el último tiempo, algunos autores han propuesto formas de organizar / estructurar los algoritmos de detección y caracterización de grietas existentes.

A continuación, se presenta un resumen de las técnicas de procesamiento de imágenes disponibles en [9], organizado en cuatro etapas de procesamiento:

(i) **Preprocesamiento**, basado principalmente en técnicas de estiramiento de contraste y ecualización de histograma, esta etapa viene a reducir los efectos de las sombras y otros objetos igualando las variaciones en la textura del concreto y mitigando el contraste entre las áreas húmedas y secas de la superficie.

(ii) **Segmentación de la imagen**, basada en operaciones de umbral de entropía fija o difusa, así como en los coeficientes espaciales de la transformación Wavelet [16], esta etapa tiene como objetivo separar la información de grietas del resto de la imagen.

(iii) **Postprocesamiento**, basado principalmente en operaciones de búsqueda morfológica y de conectividad [17], esta etapa es para reducir la cantidad de grietas falsas previamente detectadas, así como para vincular regiones de grietas para formar grupos de píxeles conectados de intensidades más oscuras que su entorno.

(iv) **Extracción de grietas**, emplea técnicas como la transformación de Hough [18] o redes neuronales, entre otras, para identificar grietas y ubicarlas en imágenes.

En [10], se discuten los enfoques de detección de grietas semiautomáticos y automáticos existentes, y se organizan en cinco etapas:

(i) **Análisis de histograma**, utilizando técnicas de umbral (adaptativas o locales) siguiendo hipótesis gaussianas [19], se realiza para distinguir entre píxeles de grietas y el fondo de la imagen.

(ii) **Herramientas de morfología matemática**, generalmente adoptadas para aliviar el problema de las falsas detecciones de grietas, imponiendo la continuidad espacial entre grupos de componentes conectados detectados como grietas.

(iii) **Fase de aprendizaje**, utiliza, por ejemplo, una red neuronal entrenada u otra técnica de aprendizaje automático.

(iv) **Filtrado de imágenes**, incluida la detección de bordes, filtrado adaptativo, contornos, métodos basados en filtros Gabor, filtrado de respuesta de impulso finito o técnicas de filtrado basadas en ecuaciones diferenciales parciales, entre otros [20], explorando el conocimiento de que las grietas corresponden a desviaciones de una textura de imagen regular.

(v) **Enfoques basados en modelos**, que generalmente exploran propiedades locales, globales o multiescala de grietas en imágenes, basadas en propiedades fotométricas, geométricas o de frecuencia.

Otra propuesta para estructurar algoritmos de detección y caracterización de grietas se presenta en [14], que consta de cuatro etapas principales:

(i) **Preprocesamiento de imágenes**, generalmente aplicado para mejorar el contraste entre las regiones de grietas y el fondo de la imagen, para suavizar las imágenes que exhiben una textura altamente aleatoria, o para reducir el ruido que corrompe la imagen y dificulta la detección de grietas.

(ii) **Extracción de características**, cálculo del valor de las características seleccionadas para ser utilizadas por un sistema de reconocimiento de patrones para la identificación de grietas, explotando las propiedades fotométricas, geométricas o de frecuencia de los píxeles de grietas.

(iii) **Detección y clasificación de grietas**, utilizando técnicas como umbralización, redes neuronales, k-NN, potenciación o soporte de clasificadores de máquinas de vectores.

(iv) **Caracterización del tipo de grietas**, utilizando otro sistema de reconocimiento de patrones para clasificar las grietas según sus formas [11].

3. Marco teórico

En este capítulo, se presentan al lector los antecedentes teóricos subyacentes a esta tesis. Al principio, se explica el Análisis de Componentes Principales (PCA), que es posiblemente el método más conocido para realizar una reducción de dimensionalidad de datos. Luego, se presenta una breve introducción al aprendizaje automático y, en particular, los tipos de problemas de clasificación que existen dentro del área para contextualizar al lector en el tema del diagnóstico de los estados de salud y clasificación. A continuación, la discusión se centra en las Redes Neuronales Artificiales (ANN) y las Redes Neuronales Convolucionales (CNN), ambos algoritmos de Aprendizaje Automático que han demostrado un considerable éxito en las tareas de clasificación dentro de las áreas de visión computacional, reconocimiento de voz, procesamiento de lenguaje natural e incluso la fiabilidad y el mantenimiento basado en la condición (CBM).

3.1. Análisis de Componentes Principales

El Análisis de Componentes Principales (PCA) es una herramienta estándar en el análisis de datos moderno, utilizado para la reducción de dimensionalidad, compresión de información, extracción de características y visualización de datos.

El objetivo principal del algoritmo PCA es reducir la dimensionalidad de un conjunto de datos, mientras se conserva la mayor cantidad de información posible. Esto va guardando únicamente los componentes más importantes dentro de los datos que repercuten en mayor grado a los resultados al encontrar una transformación ortogonal **P** que transforma los datos originales en una nueva representación latente, con la esperanza de que este proceso filtre el ruido y revele relaciones ocultas en los datos. A continuación, se define una nomenclatura y luego se explica el algoritmo general.

Primero, los datos se representarán como una matriz **X** que pertenece al espacio $\mathbb{R}^{N \times D}$. En **X**, las filas representan diferentes puntos de datos y las columnas representan las diferentes características de los datos. La transformación **P** se puede definir como otra matriz, que pertenece a un espacio diferente $\mathbb{R}^{D \times k}$ donde k es el número de componentes principales que se elige incluir. Entonces, los datos reducidos pueden expresarse como una matriz $Z \in \mathbb{R}^{N \times k}$. El desafío es encontrar los coeficientes indicados de **P** que produzcan una representación reducida óptima. Esta reducción óptima se obtiene cuando se maximiza la varianza de los datos proyectados. Se puede demostrar [1] que esto sucede cuando la transformación P se construye de la siguiente manera:

1. Se calcula la matriz de covarianza de **X**, $\mathbf{S}_X$.
2. Se determinan el vector propio y valor propio de $\mathbf{S}_X$.
3. Para una representación reducida de la dimensión *k*, se seleccionan los *k* valores propios más grandes, junto con sus vectores propios correspondientes.
4. La transformación se construye colocando los vectores propios seleccionados como columnas en una nueva matriz. Esa matriz corresponde a **P**.

Para realizar realmente la reducción de dimensión del conjunto de datos, se calcula la siguiente expresión.

$$\mathbf{Z} = \mathbf{XP} \tag{3-1}$$

3.2. Aprendizaje Automático

El aprendizaje automático se puede definir como un conjunto de métodos que pueden detectar automáticamente patrones y estructuras en los datos, y luego usar esas características aprendidas para realizar predicciones o tareas de toma de decisiones como la clasificación. En general, el aprendizaje automático es un subconjunto del campo de la inteligencia artificial, que se basa en gran medida en la teoría de la probabilidad, las técnicas estadísticas y las ciencias de la computación. Los modelos dentro del aprendizaje automático intentan aprender de los datos, por lo que no es necesaria una programación explícita para resolver problemas complejos. Los tipos de problemas que se pueden resolver con el aprendizaje automático se pueden dividir en tres categorías diferentes: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje de refuerzo.[2]

- **Aprendizaje Supervisado:** En este tipo de problemas, el objetivo principal es aprender un mapeo de las entradas *x* a las salidas *y* dado un conjunto de datos que contiene un conjunto etiquetado de N pares de entrada-salida $D = \{(x_i, y_i)\}_{i=1}^{N}$. La existencia de la etiqueta y_i para cada entrada *x* es fundamental en este tipo de enfoque, ya que todo el proceso de capacitación se basa en encontrar una relación $f : x \to y$ que minimice la discrepancia entre las salidas deseadas y las salidas que el modelo está produciendo.

  En general, cada entrada x_i se puede representar como un vector D-dimensional. Si los datos de entrada son una imagen o una señal de vibración, el proceso de expresarlos como vectores es natural, pero para casos más complejos, como en el reconocimiento de mensajes o textos escritos, puede ser necesario algún procesamiento previo.

  La forma de las salidas y_i depende principalmente del tipo de problema que se está resolviendo. Para problemas de clasificación, donde el objetivo es asignar cada entrada x_i a una o más clases de un conjunto $C = \{c_1, c_2, \dots c_k\}$, la forma de las salidas es categórica. Eso significa que cada y_i es igual al código de la clase a la que pertenece x_i. Por ejemplo, si x_i pertenece a la clase c_2 de 4 clases posibles, es recomendable trabajar con una salida escrita de la siguiente manera: $y_i = [0, 1, 0, 0]^T$. Esta representación a menudo se llama representación codificada en caliente en la literatura.

  En el contexto de la confiabilidad, un ejemplo de un problema de clasificación podría ser el diagnóstico de modos de falla en elementos mecánicos o la detección

de la ubicación de una falla determinada. En la sección 3.3 se presenta al lector una discusión sobre los diferentes tipos de problemas de clasificación.

- **Aprendizaje No Supervisado:** para este tipo de problemas, la base de datos solo consiste en un conjunto de entradas $D = \{x_i\}_{i=1}^{N}$, y el objetivo no es predecir o clasificar nada, sino encontrar patrones o estructuras dentro de los datos. Por esa razón, este subconjunto de aprendizaje automático también se llama *descubrimiento de conocimiento*. Este tipo de problemas están mucho menos definidos y son mucho más difíciles en general, porque nada les dice a los modelos qué tipo de patrones deben buscar, ni si existe una métrica de error para usar (en el aprendizaje supervisado, la comparación entre la predicción y el resultado deseado es lo que guía el algoritmo hacia buenas soluciones).

  Los ejemplos más comunes para este tipo de problemas son los algoritmos para el agrupamiento, cuyo objetivo principal es agrupar los diferentes ejemplos dentro de la base de datos en clases, sin tener conocimiento previo sobre sus categorías reales, o si estas existen.

- **Aprendizaje Por Refuerzo:** este se centra en los procesos de aprendizajes reglamentados, en los que se proporcionan algoritmos de aprendizaje automáticos con un conjunto de acciones, parámetros y valores finales. Al definir las reglas, el algoritmo de aprendizaje automático intenta explorar diferentes opciones y posibilidades, monitoreando y evaluando cada resultado para determinar cuál es el óptimo. En consecuencia, este sistema enseña a través del proceso de ensayo y error. Aprende de experiencias pasadas y comienza a adaptar su enfoque en respuesta a la situación para lograr el mejor resultado posible.

Para efectos de esta tesis, dentro del contexto del aprendizaje supervisado, las redes neuronales se utilizarán ampliamente como modelos capaces de realizar tareas de clasificación.

3.3. Problemas de Clasificación

Dentro de los problemas de clasificación, existen principalmente cuatro categorías que dependen del número y la estructura de las clases presentes en el problema [3].

1. **Clasificación Binaria:** en este tipo de problemas, la entrada debe clasificarse en una de dos clases no superpuestas: C_1 o C_2. Los ejemplos de este problema incluyen la clasificación del estado de falla / no falla en los mecanismos, o el proceso de toma de decisiones de elegir si se debe realizar un mantenimiento o no. Para esta tesis, este es el tipo de problema que el modelo de clasificación tendrá como objetivo resolver dentro del contexto de clasificar las imágenes con o sin grieta.

2. **Clasificación de Clases Múltiples:** el objetivo de este tipo de clasificación es clasificar cada entrada como una, y solo una clase de un conjunto de *k* clases no superpuestas. Los ejemplos de este tipo de problemas incluyen el reconocimiento facial o el reconocimiento de voz en el campo de la biometría o, en el campo de la confiabilidad, el diagnóstico de modos de falla en un elemento mecánico de una lista de más de dos fallas posibles que no se presentan en el mismo tiempo.

3. **Clasificación con Etiquetas Múltiples:** en los problemas con etiquetas múltiples, el objetivo es clasificar cada entrada en una o varias clases no superpuestas. Este tipo de problemas son más complejos que los de varias clases ya que las entradas pueden pertenecer a más de una clase, por lo tanto, el número de posibilidades para cada ejemplo aumenta en gran medida. En los sectores industriales, este tipo de problemas son importantes para los elementos que pueden fallar por más de una razón al mismo tiempo.

4. **Clasificación jerárquica:** la entrada para este tipo de problemas debe clasificarse en una, y solo una clase, que podría dividirse en subclases o agruparse en diferentes superclases. Una característica de estos problemas es que se supone que la jerarquía es estable durante la tarea de clasificación. Un ejemplo de este tipo de problema de clasificación es la falla general de una caja de engranajes, donde la primera categoría podría ser binaria (falla o no falla), entonces si el primer resultado es que la caja de cambios falló, la tarea de clasificación pasa a un segundo paso para determinar qué elemento falló (rodamientos, engranajes, pernos) y luego a un tercer paso para identificar el modo de falla que ocurrió en ese elemento específico.

Para todos los tipos de problemas de clasificación discutidos anteriormente, se puede definir una gran variedad de métricas para medir el rendimiento de un determinado clasificador. En esta tesis, la métrica utilizada para comparar los modelos implementados entre ellos es la *precisión de la clasificación*, que se puede entender como el número de predicciones correctas dividido por el número total de predicciones. Si y_i es la salida categórica deseada para la entrada x_i e $\hat{y}_i$ es la salida obtenida del clasificador, entonces la precisión para el conjunto de datos $D = \{(x_i, y_i)\}_{i=1}^{N}$ puede definirse como:

$$Accuracy(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i) \qquad (3\text{-}2)$$

Donde $\mathbb{1}$ es igual a 1 si la condición en el argumento es verdadera, y 0 en caso contrario. N representa el número total de muestras que se clasificaron, correcta o incorrectamente.

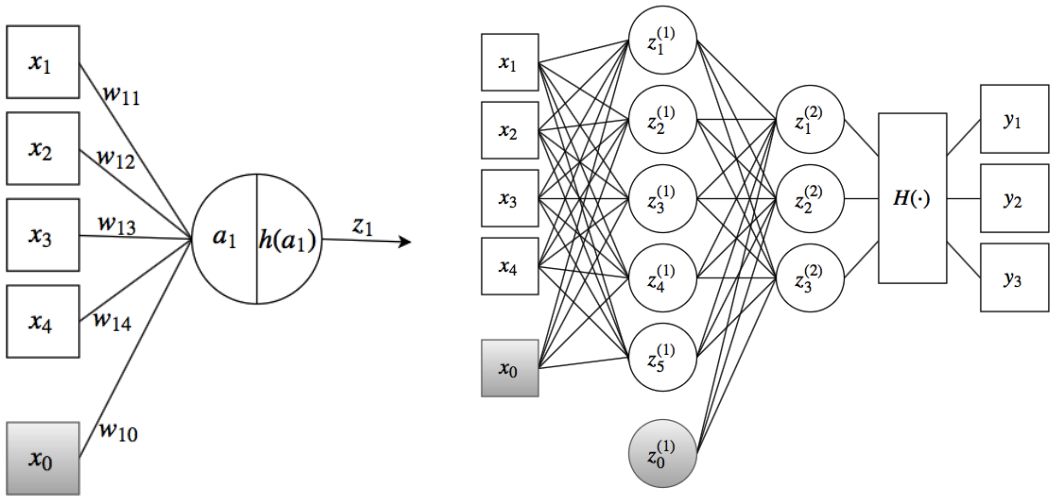
3.4. Redes Neuronales Artificiales

Las redes neuronales artificiales (ANNs) son un tipo de modelo dentro del campo del aprendizaje automático cuyo objetivo principal es aproximar funciones complejas para expresar. Una de las aplicaciones más populares para las ANNs incluye la resolución de problemas de aprendizaje supervisados, en particular, tareas de clasificación. Para esto, las ANNs combinan una base de funciones no lineales, donde cada función que pertenece a esa base es, en sí misma, otra función no lineal de una combinación lineal de las entradas, donde los parámetros que controlan esa combinación son parámetros adaptativos que pueden ser optimizados hacia un objetivo [1].

Un vector de entradas para las ANN se define como un vector $\vec{x} = \{x_1, \dots, x_D\}$. Las ANN en sí, consisten en *C* capas diferentes apiladas una encima de la otra, donde cada capa consta de unidades c_k o neuronas. En la primera capa de la ANN, el vector de entrada $\vec{x}$ se combina linealmente con un primer conjunto de parámetros como se expresa en la ecuación 3-3:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{3-3}$$

donde $j = 1, \dots, c_1$ es un subíndice que indica la neurona respectiva de la primera capa y el superíndice (1) indica que el parámetro correspondiente pertenece a la primera capa de la ANN. Los parámetros w_{ji} se conocen como los *pesos* de la red y w_{j0} como *sesgos*. La cantidad a_j se conoce como la *activación* de la neurona *j*. Con el fin de hacer que la redacción de las ecuaciones sea más clara y concisa, generalmente en la literatura el parámetro de sesgo se incluye como un peso extra.



(a) Unidad de neurona. (b) Red neuronal.

Figura 3.1 Representación gráfica de (a) una unidad neuronal con dimensión de entrada D = 4 y (b) red neuronal con dimensión de entrada D = 4, 3 unidades de salida y 2 capas. El número de unidades en la primera y segunda capas son $c_1 = 5$ y $c_2 = 3$ respectivamente.

Para esto, es necesario definir un nuevo componente del vector de entrada, x_0, cuyo valor siempre es igual a 1. Con esta nueva entrada, la ecuación 3-3 puede reescribirse como:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{3-4}$$

Donde el parámetro de polarización sigue siendo $w_{j0}^{(1)}$ y el componente x_0 es igual a 1. Luego, cada una de las activaciones a_j se transforma mediante una función no lineal diferenciable $h(\cdot)$, como sigue:

$$z_j = h(a_j) \tag{3-5}$$

Como ejemplo, la figura 3.1a muestra una representación gráfica de una unidad neuronal que tiene como entrada un vector $\vec{x}$ con dimensionalidad $D = 4$.

Las cantidades z_j se conocen como salidas de las neuronas. En un segundo paso, el proceso se repite nuevamente, pero con la siguiente capa de neuronas y utilizando como entrada las salidas de la capa anterior, $\{z_1, \ldots, z_j, \ldots, z_{c1}\}$, que tiene una dimensionalidad igual al número de neuronas de la primera capa, c_1. Nuevamente, se define un nuevo componente del vector de entrada actual, z_0, como 1 y el parámetro de sesgo se incluye en los pesos para la nueva capa. En este caso, las activaciones de las segundas capas se pueden escribir como:

$$a_l = \sum_{j=0}^{c_1} w_{lj}^{(2)} z_j \tag{3-6}$$

Donde ahora, el subíndice $l = 1, \ldots, c_2$ indica la unidad neuronal de la segunda capa. Es importante notar que los pesos y sesgos de la ANN pertenecen a una capa específica dentro de la red, es por eso que ahora el superíndice es (2). De nuevo, estas activaciones se transforman con la función $h(\cdot)$, en la forma:

$$z_l = h(a_l) \tag{3-7}$$

En este punto, si la ANN tiene más capas, el proceso se repite nuevamente hasta que se calculan las salidas de la última capa. Una vez que esto sucede, se les aplica una última función de activación $H(\cdot)$ para generar la salida global de la ANN, el vector $\vec{y}$.

$$\vec{y} = H(\vec{z}) \tag{3-8}$$

Las expresiones matemáticas para las salidas de todas las capas de la red se pueden combinar para formular una expresión general. Para el caso donde la ANN solo consiste en dos capas, esa expresión se puede escribir como:

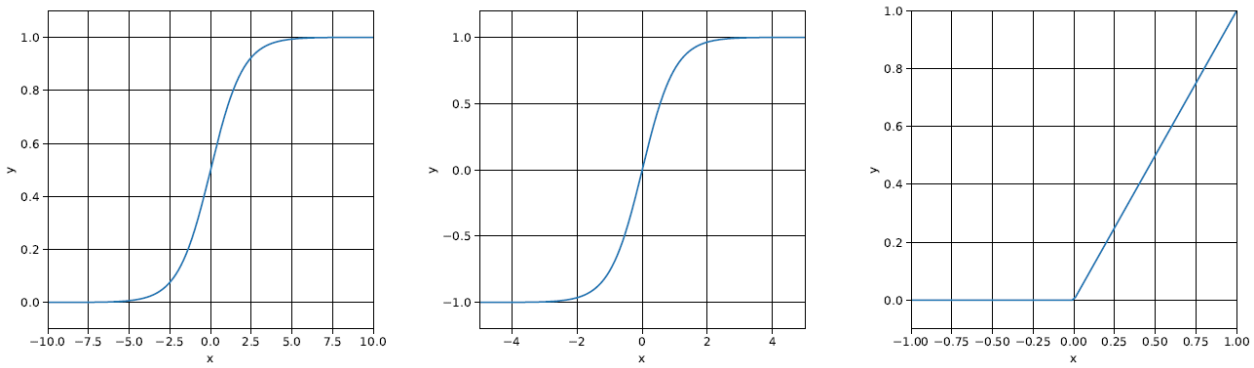
$$y_l(\vec{x}, \vec{w}) = \sum_{j=1}^{c_1} w_{lj}^{(2)} h(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}) + w_{l0}^{(2)} \tag{3-9}$$

En la ecuación 3-9, $\vec{w}$ es un vector que contiene todos los pesos y sesgos de la ANN. El proceso general de evaluación de la expresión anterior se puede interpretar como el paso directo de los vectores de entrada a través de la red. La ecuación 3-9 también muestra que el modelo de ANN es solo una función no lineal controlada por un conjunto de parámetros adaptativos representados por el vector w. En la figura 3.1b se muestra una representación gráfica de una red neuronal.

La forma explícita que tomará la función $H(\cdot)$ depende principalmente del tipo de problema que la ANN está diseñado para resolver. Como en esta tesis el uso de ANNs está restringido a problemas de clasificación entre dos clases para la detección de grietas incipientes en imágenes de vigas de hormigón armado, en la sección 3.4.2 se explica la forma más utilizada para $H(\cdot)$ en ese tipo de problemas. A continuación, se presentan las funciones de activación utilizadas para las capas intermedias de la red.

3.4.1. Funciones de activación para las capas intermedias

En las capas intermedias de la red, el uso de funciones que no son lineales y diferenciables en casi todos los puntos de su dominio es lo que permite a la ANN aproximar funciones difíciles de expresar o calcular. Una discusión más detallada acerca de cómo se puede usar una suma de funciones no lineales para aproximar casi todas las funciones está en [4] y [5].



(a) Función sigmoidea (b) Función tangente hiperbólica (c) Función ReLU

Figura 3.2 Funciones de activación para las capas intermedias de la ANN.

En el contexto de una ANN, existen tres funciones de activación muy populares: la función sigmoidea, la función tangente hiperbólica y la función ReLU. En la Figura 3.2 se representan gráficas para todos ellos.

La expresión y la gráfica de la función sigmoidea se muestran en la ecuación 3-10 y la figura 3.2a, respectivamente. Como se puede ver en el gráfico, la función sigmoide transforma la entrada x del dominio inicial $\mathbb{R}$ al espacio contenido en el intervalo (0,1). Esta función de activación es una opción popular para problemas de clasificación binaria, donde se necesita expresar las salidas de las unidades neuronales como probabilidades de que la entrada pertenezca a una de dos clases.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3-10}$$

La expresión para la función tangente hiperbólica se muestra en la ecuación 3-11. Como se puede ver en la Figura 3.2b, esta función exhibe un comportamiento similar al de la función sigmoidea, donde la única diferencia radica en el espacio al que se transforman las entradas. En este caso, ese espacio es el intervalo (-1,1) en lugar de (0,1).

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3-11}$$

Finalmente, la expresión para la función ReLU se presenta en la ecuación 3-12. Esta función es hoy en día la elección típica como función de activación en la mayoría de las redes neuronales. La gráfica para la función ReLU se muestra en la figura 3.2c.

$$ReLU(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{3-12}$$

3.4.2. Función de activación para la capa de salida

La elección de la función de activación utilizada en la capa de salida dependerá principalmente del tipo de problema para el que está diseñado la ANN. Como se mencionó anteriormente, la función $H(\cdot)$ se aplica a las salidas de las neuronas de la última capa para generar la salida global de la red neuronal.

Para las redes neuronales de clasificación de varias clases, la idea general es generar un vector $\vec{y}$ que represente en su componente *k* la probabilidad de que el vector de entrada pertenezca a la clase *k*. Para hacer esto, la función de activación más utilizada para la capa de salida es la función *Softmax*.

Para un problema con *K* diferentes clases, la expresión para la función *Softmax* aplicada en el vector de salida de la última capa $\vec{z}$ se muestra en la ecuación 3-13. Como se puede ver en esa expresión, el vector de la capa de salida se *aplasta* en el intervalo (0,1) como se ve en la figura 3.3, de manera que la suma del vector transformado suma 1. Por lo tanto, pueden interpretarse como probabilidades.

$$\vec{y} = H(\vec{z}) = \left\{ \frac{e^{zk}}{\sum_{i=1}^{K} e^{zk}} \right\}_{k=1,\dots K} \tag{3-13}$$

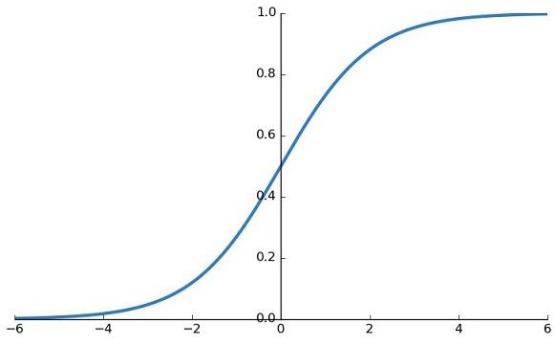


Figura 3.3: Gráfico de la función de activación Softmax.

La siguiente sección continúa la discusión sobre las redes neuronales, específicamente sobre el tema de su entrenamiento.

3.4.3. Entrenamiento de redes neuronales artificiales

Como se mencionó en la sección 3.4, un ANN es una función no lineal controlada por un conjunto de parámetros adaptativos denotados por el vector $\vec{w}$ (que contiene los pesos y los sesgos del modelo). El objetivo principal de cada red neuronal es aproximar una función que es muy difícil de expresar o calcular. Esa función modela alguna situación de interés. Por ejemplo, podría modelar la relación existente entre las emisiones acústicas de un determinado componente mecánico y la presencia de un determinado fallo en él. Como las redes neuronales son modelos que se basan en una base de datos para aprender, la idea general detrás de su entrenamiento es minimizar alguna noción de error o discrepancia entre la respuesta de la red neuronal y la salida deseada para una determinada entrada $\vec{x}$. Este proceso de minimización se realiza variando los parámetros adaptativos contenidos en $\vec{w}$ hasta que se alcanza un mínimo local.

La noción de error será una función tanto de los parámetros como de los datos de entrada, $E(\vec{x},\vec{w})$. Su forma explícita dependerá del tipo de problema a resolver. La metodología que utilizan las redes neuronales para adaptar sus parámetros para minimizar la función *E* se basa en una técnica numérica de optimización llamada *Descenso de gradiente*. A continuación, se explica el descenso en gradiente.

Descenso de Gradiente

En la ecuación 3-14 se muestra el problema de optimización que presenta el entrenamiento de redes neuronales.

$$\vec{w}^* = arg_{\vec{w}} min E(\vec{w},\vec{x}) \qquad (3\text{-}14)$$

En muchos casos, en los que se incluyen las redes neuronales, no es práctico resolver una optimización práctica al encontrar una solución analítica. Para esas situaciones, es necesario recurrir a un método iterativo. El más común de estos métodos es el descenso

por gradiente [6]. Si E($\vec{w}$) es una función diferenciable que se desea minimizar con respecto a un vector de parámetros $\vec{w} \epsilon \mathbb{R}^D$, como en 3-14, siempre es posible calcular un vector que contenga todas las derivadas parciales de la función *E* con respecto a cada parámetro, como se muestra en la ecuación 3-15:

$$\frac{\partial E}{\partial \vec{w}} = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_D}\right]^T \tag{3-15}$$

En el descenso de gradiente, el proceso de iteración comienza con valores iniciales aleatorios para $\vec{w}$. Luego, en cada iteración, se calcula el vector de gradiente, y luego cada parámetro se actualiza en la dirección opuesta del gradiente[1]:

$$w_d \rightarrow w_d - \eta \frac{\partial E}{\partial w_d} \tag{3-15}$$

Donde η es un hiperparámetro llamado *tasa de aprendizaje* y determina cuánto moverse en esa dirección en cada paso. Con este proceso, $E(\vec{w}, \vec{x})$ se minimizará hasta que se alcance un mínimo global o local, donde el gradiente sea cero y, por lo tanto, el proceso se detenga. En la práctica, el proceso se detiene cuando el gradiente alcanza un valor cercano a cero con algún tipo de tolerancia definida previamente por el programador. Para las redes neuronales, a medida que el número de parámetros aumenta considerablemente con el número de neuronas y capas, se desarrolló un método llamado *Back-Propagation* (BP) para realizar el descenso del gradiente cuando la dimensionalidad de $\vec{w}$ resulta en un problema de capacidad. En pocas palabras, el algoritmo BP hace uso de la regla de la cadena de multiplicación para propagar el error de la capa *k* a la capa *k-1*, haciendo que el proceso se ejecute más rápido y más eficientemente. Se pueden encontrar más detalles sobre este tema en [1]. A continuación, se introducen las redes neuronales convolucionales (CNN), ya que representan una variedad interesante de redes neuronales que han ganado popularidad en los últimos años por sus resultados de vanguardia en el campo de la visión por computadora.

3.4.4. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN) son otro modelo de aprendizaje profundo, muy similar al ANN tradicional (descrito en la Sección 3.4). De hecho, las CNNs también tienen pesos y sesgos que pueden optimizarse mediante el descenso de gradiente estocástico y el algoritmo de propagación hacia atrás para resolver un problema de clasificación o regresión, también están formados por unidades neuronales, donde cada uno de ellos recibe una entrada, realiza un producto punto y luego lo sigue con una no

[1] Si se desea maximizar la función E, el parámetro se actualiza en la dirección positiva del gradiente.

linealidad e incluso tienen una función de puntuación que representa qué tan bien el modelo está aprendiendo la relación entre las entradas y las salidas deseables.

La principal diferencia entre CNN y ANN tradicional está en el tratamiento topológico de la entrada [7]. CNN asume explícitamente que las entradas tienen dependencias espaciales entre sus diferentes componentes (por ejemplo, imágenes o series de tiempo), por lo que el modelo puede aprovechar la relación espacial entre las características de los datos. En palabras simples, las ANNs convierten las entradas en vectores antes de pasar por el modelo, perdiendo todo tipo de información relacionada con la ubicación espacial de cada componente con respecto a los demás. Las CNNs no realizan esa transformación inicial de la entrada, por lo que tienden a funcionar mejor con datos como imágenes, por ejemplo.

Un modelo CNN tradicional está formado por tres tipos de capas apiladas que trabajan juntas para extraer información útil de los datos y realizar la tarea de clasificación o regresión. Estas capas se describen brevemente a continuación:

1. **Capa Convolucional:** este tipo de capa realiza una operación matemática llamada *convolución* entre la entrada y un conjunto de parámetros que se pueden ir modificando en cada aplicación para formar una estructura llamada *filtro*. Este filtro se desliza a través de la entrada realizando convolución, extrayendo mapas de características en el proceso.

2. **Capa de Agrupación:** este tipo de capa realiza un muestreo descendente de su entrada, lo que reduce la dimensión de los datos que fluyen a través de la red, lo que lo hace más manejable y, con suerte, ayuda en el proceso de extracción de información útil de la entrada inicial. La capa de agrupación más común es la *capa de agrupación máxima*, que desliza una ventana de tamaño $p \, x \, p$ a través de la entrada e informa a la siguiente capa solo el valor máximo de dicha ventana.

3. **Red de Avance de Alimentación:** en general, una CNN tendrá n capas convolucionales intercaladas con m capas de agrupación. Se espera que este proceso funcione como un proceso de extracción de características, obteniendo al final un vector de características útiles de la entrada. La última etapa de la CNN es siempre una red de alimentación directa, que es la que realiza la clasificación de la tarea de regresión utilizando como entrada aquellas características extraídas por la combinación de capas convolucionales y de agrupación.

4. Experiencia en laboratorio

En este capítulo, se presentan al lector los procedimientos y ensayos que se llevan a cabo en el Laboratorio de Sólidos presente en la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, con el objetivo de recoger los datos necesarios para el posterior entrenamiento de la red neuronal.

4.1. Procedimientos para Ensayo y Disposición

Programa de actividades en laboratorio de sólidos:

- En colaboración con los encargados de laboratorio, se coordina el almacenamiento de 2 vigas de hormigón armado cuya confección fue realizada con anterioridad, estas tienen 0,2 [m] de alto, 0,15 [m] de ancho y 1,9 [m] de largo.

- En primer lugar, antes de la realización de cualquier ensayo, se buscan las condiciones ideales del laboratorio, esto tiene como objetivo capturar en video de manera óptima la aparición y propagación de las grietas en las vigas de hormigón armado, para esto se debe encontrar una iluminación optima, una distancia correcta entre cámaras y de estas con la viga, como se ve en la figura 4.4, por último, un zoom adecuado para cada cámara.

- Para la medición de la deformación de la viga en el punto en donde se realiza la carga, se instala un sensor U-GAGE S18U con salida análoga, descrito en la sección 4.5, el cual es posicionado en el tren de carga.

- Luego, se realiza una verificación de que la tarjeta de adquisición de datos que está disponible en el Laboratorio de Sólidos esté funcionando correctamente, ya que es indispensable en la tarea de la medición de carga aplicada y la deformación de la viga en tiempo real.

- Posteriormente, a cada viga se le realiza un ensayo de flexión por medio de una carga controlada en la mitad del elemento estructural, este proceso se monitorea por las 3 cámaras ya correctamente configuradas, las características de estas cámaras están presentes en la tabla 4.1.

- Por último, todos los videos e imágenes generadas se almacenan en Google Drive en el momento en que terminan los ensayos, es gracias al uso de esta plataforma que se evita cualquier perdida de datos.

4.2. Cámaras

A continuación, se presentan las características de cada una de las cámaras utilizadas en los ensayos realizados para el desarrollo de esta tesis:

Tabla 4.1 Características de cámaras.

Cámara	Modelo	Resolución	Fotogramas/segundo
Sony	FDR AXP35	XAVC S HD: 1920 x 1080	60
Panasonic	AG-AC30	AG-AC30PJ/PB: 1920 x 1080	60
Sony	W800	NTSC: 1280 x 720	30

Figura 4.1 Cámara Handycam® 4K AXP35 posicionada a la izquierda.

Figura 4.2 Cámara Panasonic AG-AC30 posicionada a la derecha.

Figura 4.3 Cámara Sony W800 posicionada al centro.

4.3. Ubicación de Cámaras

La ubicación de las tres cámaras es fundamental para obtener buena calidad en la imagen y lograr capturar de manera óptima el fenómeno de aparición de la grieta y su posterior propagación, al tener tres cámaras disponibles, se dispone de un par de ellas para dividir en dos la región de interés, que para efectos de esta tesis, se define como región de interes todo el largo y alto de la viga, optimizando la calidad de la imagen con la que se trabajará posteriormente, por otro lado, la tercera cámara tiene como objetivo entender fenómenos globales, la curvatura de la viga como un todo es un buen ejemplo.

A continuación, se presenta la ubicación en planta de cada cámara para el monitoreo del ensayo, cabe mencionar que en la figura 4.4 todas las medidas están en milímetros:

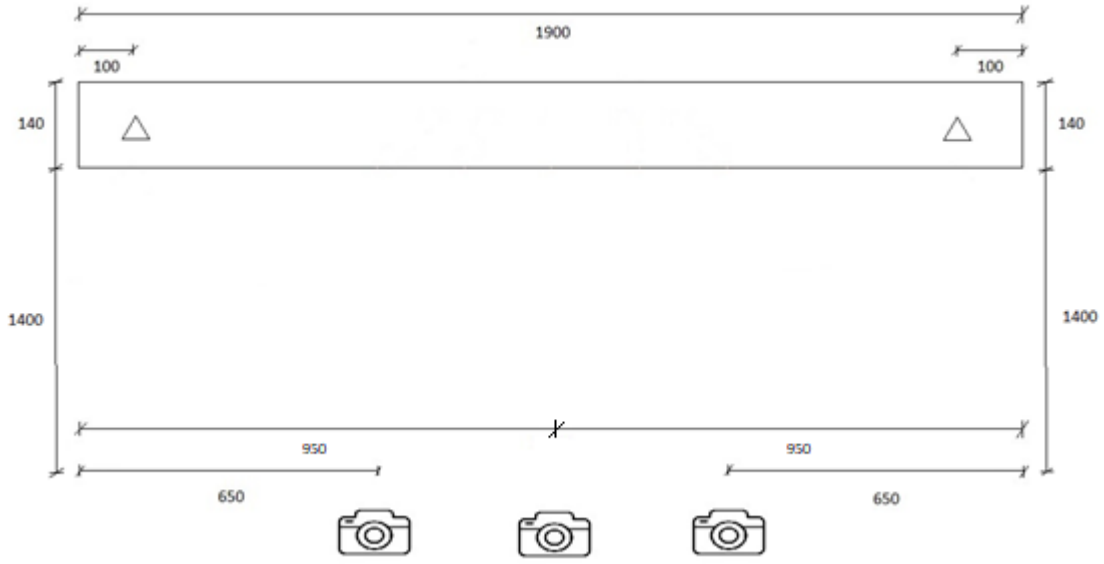


Figura 4.4 Plano en planta de las cámaras en vigas de 1,9 [m].

4.4. Modos de Falla Seleccionados

En vista y considerando que esta tesis busca entrenar una red neuronal que tenga la capacidad de detectar la presencia de una grieta incipiente en imágenes de vigas de hormigón armado. Es de vital importancia entregar y nutrir a esta red neuronal con distintos tipos de grietas, que tengan distintas direcciones, varios anchos y distintos largos, esto se logrará variando el refuerzo presente en cada una de las vigas para ensayar.

Ambas vigas disponibles cuentan con refuerzo a flexión y refuerzo a corte, a continuación, se describen los refuerzos presentes en cada una de las vigas:

- Viga 1: refuerzo a flexión de 4 barras Φ= 8 [mm], su refuerzo a corte consta de estribos Φ= 8 [mm] a una separación de 15 [cm] entre ellos.

- Viga 2: refuerzo a flexión de 2 barras superiores Φ= 8 [mm] y 2 barras inferiores Φ= 6 [mm], su refuerzo a corte consta de estribos Φ= 8 [mm] a una separación de 20 [cm] entre ellos.

Con esto, se logra abarcar una gran variedad de grietas y distintas distancias entre ellas. Ahora se presentan esquemas de cómo se espera, se presenten las grietas en los distintos tipos de configuración.

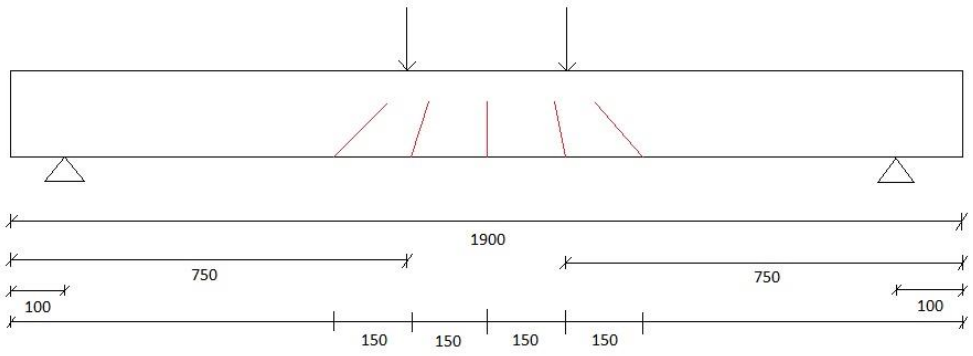


Figura 4.5 Grietas en viga con estribos cada 15[cm].

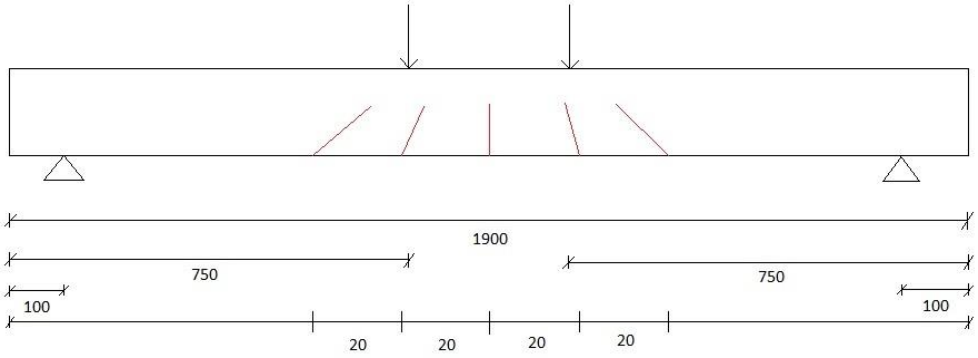


Figura 4.6 Grietas en vigas con estribos cada 20 [cm].

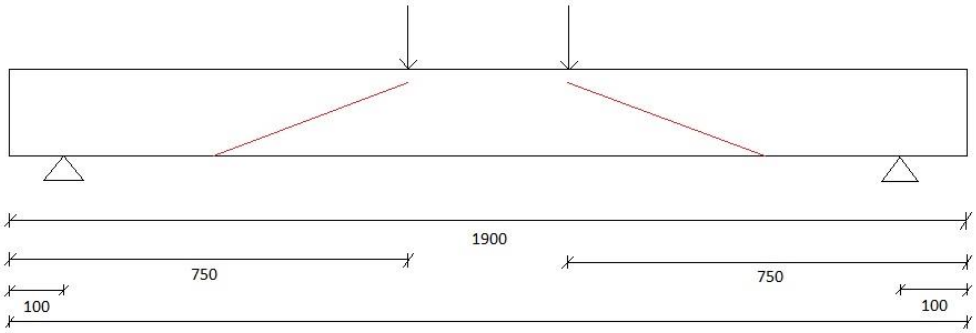


Figura 4.7 Grietas en vigas por falla de esfuerzo cortante.

4.5. Sensor de Medición

Para la medición de la deformación de la viga en el punto en donde se realiza la carga, se utiliza el sensor U-GAGE S18U con salida análoga, el cual es posicionado en el tren de carga, este sensor posee las siguientes características:

4.5.1. Características y propiedades del sensor:

• Programación rápida en TEACH-Mode y fácil de usar; sin ajustes de potenciómetro.

• Zona muerta corta.

• La salida escalable distribuye automáticamente la señal de salida sobre el ancho de la ventana de detección programada.

• Dos LED de estado bicolores.

• Diseño encapsulado resistente a entornos hostiles.

• Cable de 9 metros, y un conector QD estilo europeo de 5 pines.

• Amplio rango de funcionamiento de −20 ° C a +60 ° C (−4 ° F a +140 ° F).

• Carcasa recta o en ángulo recto.

• Compensación de temperatura.

• Tiempos de respuesta seleccionables de 2.5 [ms] o 30 [ms].

• Modelos analógicos con salida de 0 [V] a 10 [V] CC o de 4 [mA] a 20 [mA].

4.5.2. Indicadores de estado

Tabla 4.2 Estado de LED de encendido/ apagado.

El estado del LED de encendido / apagado	Indica
Apagado	El poder está apagado
En rojo	El objetivo es débil o está fuera del rango de detección
En verde	El sensor funciona normalmente, el objetivo es bueno

Tabla 4.3 El estado del LED de salida.

El estado del LED de salida	Indica
Off	El objetivo está fuera de los límites de la ventana
En amarillo	El objetivo está dentro de los límites de la ventana.
En rojo	El sensor está en modo ENSEÑAR y está esperando el primer límite
Rojo parpadeante	El sensor está en modo ENSEÑAR y está esperando el segundo límite

4.5.3. Programación de sensores:

Se usa uno de los dos métodos TEACH para programar el sensor:

• Se enseña límites mínimos y máximos individuales.

• Se usa la función ventana automática para centrar una ventana de detección alrededor de la posición enseñada.

El sensor se puede programar mediante su botón o mediante un interruptor remoto. La programación remota también se puede usar para deshabilitar el botón, evitando que personal no autorizado ajuste la configuración de programación. Para acceder a esta función, se conecta el cable gris del sensor a 0 [V] CC a 2 [V] CC, con un interruptor de programación remota entre el sensor y el voltaje. En [21] se puede encontrar más información acerca del sensor.

4.5.4. Dimensiones

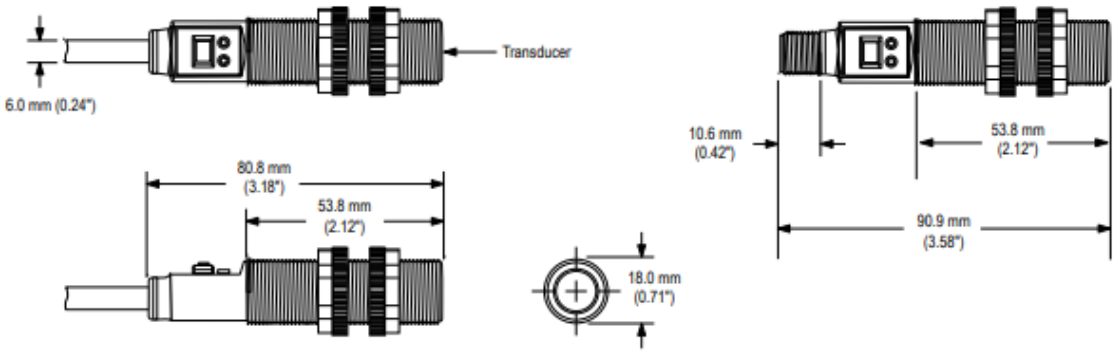


Figura 4.8 Vista frontal.

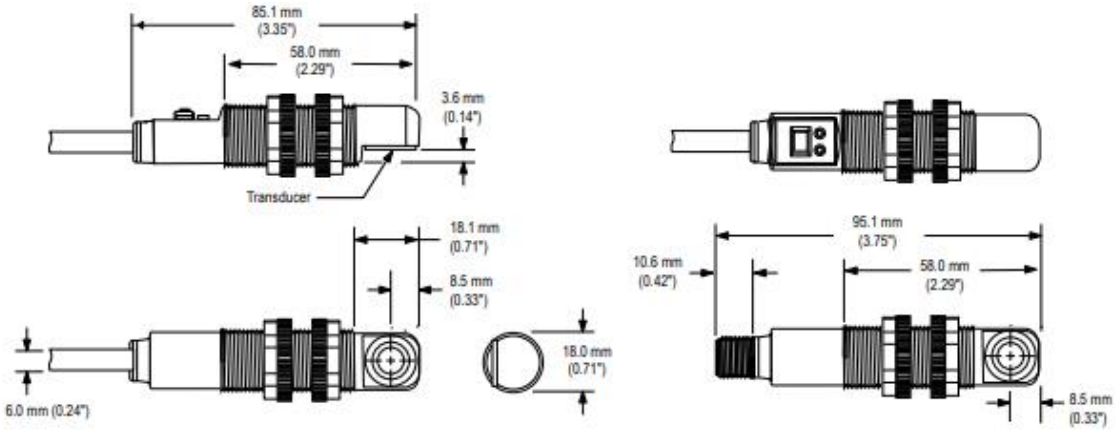


Figura 4.9 Vista de perfil.

Cabe destacar que este sensor va conectado a una tarjeta de adquisición de datos que está disponible en el Laboratorio de Sólidos. Gracias a esto, es posible trabajar con los datos posteriormente como se estime conveniente. En [21] se puede encontrar más información acerca del sensor.

Por último, se presenta la posición que ocupa el sensor de medición en el marco de carga representado en la figura 4.10.

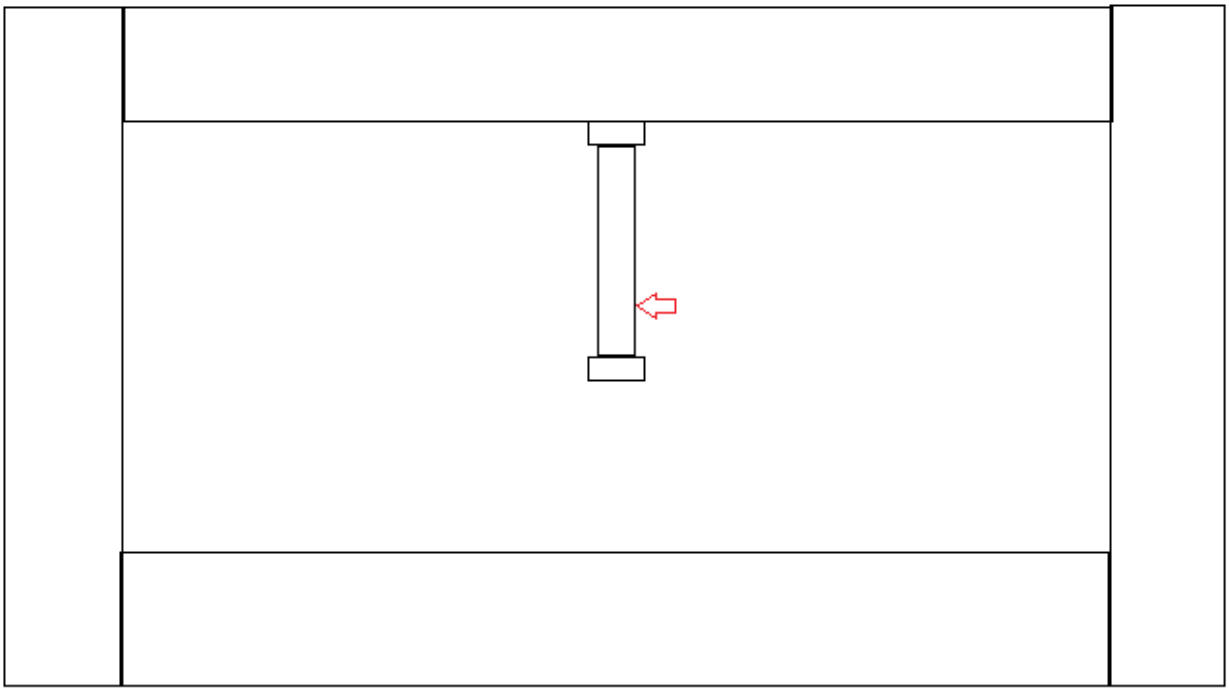


Figura 4.10 Posición del sensor de medición en el marco de carga.

5. Identificación de grietas y métodos

Como fue mencionado anteriormente, el objetivo principal de esta tesis es la creación de un software que sea capaz de detectar grietas incipientes en vigas de hormigón armado con *Aprendizaje Profundo*, para este fin, se entrena una red neuronal desde cero con las imágenes que se extraen de los videos obtenidos de los ensayos, así como también se utiliza una práctica denominada *Aprendizaje por Transferencia*, en donde se toma una red neuronal preentrenada para reconocimiento y clasificación de otro tipo de objetos y se entrena con los datos que se tienen disponibles.

5.1. Creación del Conjunto de Datos

Para todo lo anteriormente mencionado, es necesario contar con un set de datos con los cuales entrenar a la red neuronal, en consecuencia, se deben obtener imágenes adecuadas para enseñarle a la red como son las imágenes de vigas que contienen grietas y como son las imágenes que no contienen grietas, es así como de los videos generados con las tres cámaras de los ensayos de ambas vigas, se extraen determinados frames que contienen distintos tipos de grietas incipientes, variadas en largo, posición, orientación, para así de esta forma obtener la mayor variedad de grietas que se pudiesen presentar en caso de falla de estos elementos estructurales.

Luego de extraer los frames considerados valiosos de todos los videos, se procede a extraer imágenes de 128x128 [pixeles] de cada frame mencionado anteriormente hasta alcanzar la suma de 1400 imágenes en total, 700 imágenes enfocadas en grietas y 700 imágenes sin grieta, las cuales se distribuyen de la siguiente manera en 3 grupos de datos:

- Entrenamiento: Consta de 500 imágenes con grieta y 500 imágenes sin grietas.
- Validación: Consta de 100 imágenes con grieta y 100 imágenes sin grietas.
- Testeo: Consta de 100 imágenes con grieta y 100 imágenes sin grietas.

5.2. Uso de Google Colaboratory

Posterior a tener los sets definidos, estos se suben a una nube de Google Drive, puesto que, para la confección y entrenamiento de la red neuronal, se usa un cuaderno en *Google Colaboratory*, esta es una herramienta de Google en la nube para ejecutar códigos Python y crear modelos de aprendizaje de máquinas a través de la nube de Google.

La principal ventaja que ofrece esta herramienta es que libera al computador personal de tener que llevar a cabo un trabajo demasiado costoso tanto en tiempo como en potencia, incluso permite realizar ese trabajo si la máquina que se está utilizando no cuenta con

recursos suficientemente potentes, y todo de forma gratuita. Otro de los beneficios que tiene lo indica el propio nombre, *Colaboratory*, es decir, colaborativo, ya que permite realizar tareas en la nube y compartir cuadernos si se necesita trabajar en equipo.

En cuanto a *Google Colaboratory*, este es un entorno gratuito de *Jupyter Notebook* que no requiere configuración y que se ejecuta completamente en la nube. *Python* permite su utilización para diversos paradigmas de programación, como programación orientada a objetos o programación funcional, pero quizá la característica más importante en este contexto sea que también se usa como lenguaje de *scripting*, además de ser un lenguaje interpretado.

Por lo que se refiere a *Python*, este trae consigo un «modo interactivo» que con un intérprete de línea de comandos permite ejecutar sentencias y obtener los resultados, pero sus funcionalidades no fueron suficiente para los desarrolladores, por lo que surgió *IPython*, que más tarde evolucionaría en *Jupyter*.

Jupyter es un entorno interactivo que permite desarrollar código *Python* de manera dinámica. *Jupyter* se ejecuta en local como una aplicación cliente-servidor y posibilita tanto la ejecución de código como la escritura de texto, favoreciendo así la interactividad del entorno y que se pueda entender el código como la lectura de un documento.

Con respecto a los cuadernos anteriormente mencionados, un cuaderno es un documento que contiene código ejecutable (por ejemplo, *Python*) y también elementos de texto enriquecido (links, figuras, etc). Mas que nada es un entorno de trabajo en *Colaboratory*, este está compuesto por celdas, en donde una celda es la unidad mínima de ejecución dentro de un cuaderno, es decir, es donde se incluye el código y se ejecuta.

Cabe destacar que cada celda es independiente, pero todas las celdas en un cuaderno utilizan el mismo *kernel*. El *kernel* es el motor de computación que está por debajo y que se encarga de ejecutar el código y devolver el resultado para mostrarlo en la celda. El estado del *kernel* persiste durante el tiempo, con lo que, aunque cada celda sea independiente, las variables declaradas en una celda se pueden utilizar en las demás celdas.

Cuando se crea un nuevo cuaderno, este es estático, es decir, se ve su contenido, pero no se está conectado a ningún entorno de ejecución. El cuaderno se conecta a una máquina virtual de *Google Compute Engine* (la infraestructura de máquinas virtuales de Google en la nube) cuando se ejecuta una celda o se pulsa sobre el botón de “Conectar”. Al hacerlo, el cuaderno toma un momento en conectarse y después muestra, de ahí en adelante, el espacio de RAM y disco que se está consumiendo. La máquina en un inicio cuenta con 12 GB de RAM y 107 GB de almacenamiento en disco disponibles para el uso.

Por otro lado, desde el menú de “Entorno de ejecución” se puede reiniciar el estado del entorno de ejecución para liberar toda la memoria que se haya utilizado o interrumpir cualquier ejecución lanzada.

La duración de la máquina virtual a la que se conecta, vale decir, el tiempo máximo que se puede estar conectado a una misma máquina desde un cuaderno es de 12 horas. Aquí se debe tener cuidado, sobre todo si se están llevando a cabo ejecuciones que toman mucho tiempo: si se pasa más de 90 minutos sin utilizar un cuaderno, el entorno se desconecta. Para que no se desconecte basta con dejar la ventana del navegador abierta o celdas ejecutándose. La ventaja que tiene para estos casos es que, si se deja una celda ejecutándose y se cierra el navegador, la ejecución continuará y si posteriormente se abre el navegador, se tendrá el resultado.

Por defecto, el entorno al que se conecta *Colaboratory* utiliza un *kernel* con *Python 3* y no permite la ejecución con GPU. En un inicio, Colaboratory utilizaba *Python 2*, pero ahora permite crear nuevos cuadernos tanto en *Python 3* como en *Python 2*, desde el menú de “Archivo”.

De esta forma, se crea un cuaderno utilizando la versión de *Python* que se desee. No obstante, podría ocurrir que se necesite cambiar la versión de *Python* una vez creado el cuaderno o utilizar la GPU en lugar de la CPU (que es la que se usa por defecto) para realizar ejecuciones más potentes, como la creación de modelos de *Aprendizaje Profundo*, tal cual como es necesario para efectos de esta tesis. *Colaboratory* permite cambiar los ajustes del entorno para utilizar una GPU de forma gratuita. Para ello, se puede ir al desplegable de “Entorno de ejecución” y se selecciona “Cambiar tipo de entorno de ejecución”.

Aquí se puede cambiar la versión entre Python 3 o Python 2 y, lo más importante, cambiar el “Acelerador por hardware” de “None” a “GPU”. La otra opción es “TPU”, que son unidades de procesamiento de tensores, específicas de Google para ejecutar modelos de aprendizaje de máquinas.

Cuando se pulsa “Guardar”, *Colaboratory* guarda los ajustes y cambia de entorno de ejecución para dar una máquina con esos ajustes. Es posible que por temas de limitación de recursos no se pueda conectar a una máquina con GPU. Esto puede ocurrir si hay demasiados usuarios al mismo tiempo, además, Google limita el uso que podemos hacer de estas GPU para evitar que, por ejemplo, se minen criptomonedas. Por esto, puede pasar que, si se está realizando una ejecución muy prolongada y potente, el entorno se desconecte.

Si, por alguno de estos u otros motivos, no se puede conectar a un entorno, *Colaboratory* dará un aviso diciendo que no hay entornos con GPU disponibles.

Colaboratory también permite conectar con un servidor de Jupyter que se tenga configurado en local si se pulsa el botón de “Conectar” y “Conectar a un entorno de ejecución local”. Esto permite acceder a un sistema de archivos local, con las versiones que se estime conveniente y estén instaladas en local. La desventaja es que el hardware que se estaría utilizando es el de la máquina local y se perdería la capacidad de ejecutar el código sobre una GPU de Google y perdería también el uso compartido del cuaderno, pues si un usuario se conecta a local con un cuaderno, el resto de los usuarios se conectarían a un entorno en la nube.

Por último, *Google Colaboratory* permite utilizar archivos que estén almacenados en Google Drive. Existen varias opciones, cómo utilizar la API *REST* de Drive o la librería de *Python PyDrive*, pero sin duda la que más facilita las tareas es montar Google Drive localmente en la máquina.

La ventaja de usar esta opción no es únicamente que se logra tener archivos alojados en Google Drive y acceder fácilmente a ellos, sino que cuando se trabaja con *Aprendizaje de Máquinas* o *Ciencia de Datos*, en la mayoría de las ocasiones se cuenta con archivos enormes de datos. Si cada vez que se conecta a un entorno distinto se tienen que subir estos archivos, se pierde demasiado tiempo. Por ello, si se tiene estos archivos alojados en Drive y se monta Drive en la máquina, se puede acceder a ellos como si estuvieran en local, se puede encontrar más información relacionada con *Google Colaboratory* en [12].

5.3. Creación de archivos NPZ

Ya claro esto, se procede a montar Google Drive en *Google Colaboratory* con la finalidad de cargar las imágenes y crear archivos *NPZ* para cada uno de los grupos de datos, *NPZ* es un formato de archivo de *numpy* que proporciona almacenamiento de datos de matriz mediante compresión gzip. Este complemento admite datos de cualquier forma y también admite múltiples imágenes por archivo, por lo que es ideal para guardar las imágenes en forma de matrices para posteriormente poder acceder a ellas en forma de vectores con las imágenes ordenas, ya que primero se adjuntan al vector las imágenes con grieta y posteriormente las sin grieta.

5.4. Implementación de Aumento de Datos

Luego, con la finalidad de contar con un conjunto de datos robusto y bien nutrido para el entrenamiento de la red neuronal, se implementa una técnica denominada *Aumento de Datos,* esta es una técnica de regularización que consiste en generar nuevas instancias de entrenamiento a partir de las existentes, aumentando artificialmente el tamaño del conjunto de entrenamiento y reduciendo el sobreajuste. El truco es generar instancias de entrenamiento realistas; idealmente, un humano no debería poder decir qué instancias se generaron y cuáles no. Además, agregar simplemente ruido blanco no ayudará; las modificaciones que se apliquen deben poder aprenderse (el ruido blanco no lo es).

Por ejemplo, si un modelo está destinado a clasificar imágenes de grietas, puede cambiar ligeramente, rotar y cambiar el tamaño de cada imagen en el conjunto de entrenamiento en varias cantidades y agregar las imágenes resultantes al conjunto de entrenamiento (ver figura 5.1). Esto obliga al modelo a ser más tolerante con la posición, orientación y tamaño de las grietas en la imagen. Si se desea que el modelo sea más tolerante a las condiciones de iluminación, se pueden generar de manera similar muchas imágenes con varios contrastes. Suponiendo que se pueden presentar grietas en una posición simétrica

a la grieta de la imagen con la cual se cuenta, también se puede voltear las imágenes horizontalmente. Al combinar estas transformaciones, se puede aumentar considerablemente el tamaño del conjunto de entrenamiento.

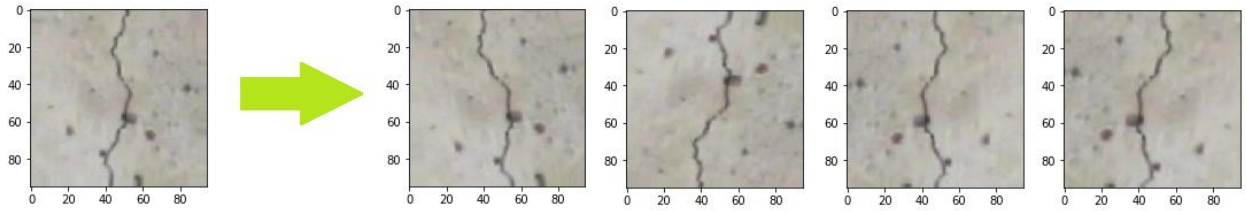


Figura 5.1 Generando nuevas instancias de entrenamiento a partir de las existentes.

A menudo, es preferible generar instancias de entrenamiento sobre la marcha durante el entrenamiento en lugar de desperdiciar espacio de almacenamiento y ancho de banda de red. *TensorFlow* [22] ofrece varias operaciones de manipulación de imágenes, como la transposición (desplazamiento), rotación, cambio de tamaño, volteo y recorte, así como el ajuste del brillo, el contraste, la saturación y el tono (consultar en [13] para obtener más detalles). Esto facilita la implementación del aumento de datos para conjuntos de datos de imágenes.

Es así como con la ayuda de la herramienta *ImageDataGenerator*, importada de la documentación de *TensorFlow* se realiza el aumento de datos a las imágenes de entrenamiento, volteando vertical y horizontalmente la imagen original y luego rotando en 20 grados estas 2 imágenes nuevas, se determinó este ángulo con la finalidad de que no se generen grietas artificiales, con lo que se consiguen 4 instancias nuevas de entrenamiento, en consecuencia, se cuenta con 2500 imágenes con grietas y 2500 imágenes sin grieta para entrenamiento.

5.5. Codificación de los datos

Además, para comprender como se realiza el entrenamiento de la red neuronal, se debe entender en primer lugar el concepto de dato categórico, estos son variables que contienen valores de etiqueta en lugar de valores numéricos. El número de valores posibles a menudo se limita a un conjunto fijo. Las variables categóricas a menudo se denominan nominales.

Algunos ejemplos incluyen:

Una variable de "color" con los valores: "rojo", "verde" y "azul".

Una variable de "lugar" con los valores: "primero", "segundo" y "tercero".

Cada valor representa una categoría diferente. Algunas categorías pueden tener una relación natural entre sí, como un orden natural. La variable "lugar" anterior tiene un orden natural de valores. Este tipo de variable categórica se llama variable ordinal.

Algunos algoritmos pueden trabajar con datos categóricos directamente. Por ejemplo, un árbol de decisión puede aprender directamente de datos categóricos sin necesidad de transformación de datos (esto depende de la implementación específica). Muchos algoritmos de aprendizaje automático no pueden funcionar directamente con los datos de la etiqueta, estos requieren que todas las variables de entrada y salida sean numéricas, en general, esto es principalmente una restricción de la implementación eficiente de algoritmos de aprendizaje automático en lugar de limitaciones estrictas en los algoritmos mismos, esto significa que los datos categóricos deben convertirse a una forma numérica. Si la variable categórica es una variable de salida, es posible que también desee convertir las predicciones del modelo a una forma categórica para presentarlas o usarlas en alguna aplicación.

Es así como para variables categóricas donde no existe tal relación ordinal, la codificación entera no es suficiente, de hecho, el uso de esta codificación y permitir que el modelo asuma un orden natural entre categorías puede dar como resultado un rendimiento deficiente o resultados inesperados (predicciones a medio camino entre categorías).

Por ende, a los conjuntos de datos se le aplica una codificación *One-Hot Encoding*, en el ejemplo de la variable "color", hay 3 categorías y, por lo tanto, se necesitan 3 variables binarias. Se coloca el valor "1" en la variable binaria para el color correspondiente y valores "0" para los otros colores, como se ve en la tabla 5.1:

Tabla 5.1 Ejemplo de One-Hot Encoding.

Rojo	1	0	0
Verde	0	1	0
Azul	0	0	1

Para efectos de esta tesis, la codificación es como se presenta en la tabla 5.2:

Tabla 5.2 One-Hot Encoding aplicado en esta tesis.

Con Grieta	0	1
Sin Grieta	1	0

En consecuencia, se procede a la creación de vectores que contengan la etiqueta de su imagen correspondiente, esto quiere decir que para los tres grupos de datos, se les asigna su propio vector “Y” con las etiquetas “[0, 1]” si es que la imagen contiene grieta y [1, 0] en el caso de que la imagen no contenga grietas, en consecuencia los vectores “Y” tienen el mismo largo que su conjunto de datos análogo, con la primera mitad completada por [0, 1] y su segunda mitad completada con [1, 0], quedando así cada conjunto apareado con su vector Y_entrenamiento, Y_validacion e Y_testeo según corresponda.

5.6. Construcción del modelo

Ahora bien, el tipo de modelo que se usa es el modelo secuencial, este modelo es la forma más fácil de construir un modelo en Keras [23], ya que permite construir un modelo capa por capa. Las capas de diferente tipo tienen algunas propiedades en común, específicamente su método de inicialización de peso y funciones de activación. *Keras* admite una gama de funciones de activación neuronal estándar, como: *softmax*, *ReLu*, *tanh* y *sigmoide*. Normalmente se especifica el tipo de función de activación utilizada por una capa en el argumento de activación, que toma un valor de cadena.

5.6.1. Batch Normalization

La primera capa en el modelo es una Batch Normalization, este es un método que normaliza cada lote de datos (bath_size), ya que es necesario para evitar que se tengan distancias muy diferentes entre ellos, en una imagen a color se pueden tener valores de 0 hasta 255. Normalizando los datos, las distancias de los datos van de 0 a 1, esto ayuda a la red neuronal a tener menos problemas, pero cuando se normaliza los datos, solo la capa de entrada se beneficia de esto, conforme los datos pasan por otras capas, esta normalización se va perdiendo y, si tenemos una red neuronal con muchas capas, podemos tener problemas con el entrenamiento. El método de Batch Normalization normaliza los datos antes de que pasen por la función de activación en cada capa de la red neuronal, de esta manera siempre tendremos los datos normalizados.

5.6.2. Capas Convolucionales

Posterior a la capa de Batch Normalization, se usan capas convolucionales en dos dimensiones, siendo estas las más indicadas para la confección de un clasificador de imágenes dado sus características presentes en la sección 3.4.4, posterior a esta capa, en vez de usar capas de agrupación, se usan capas convolucionales en dos dimensiones pero con zancada doble, llamadas convoluciones estriadas. Aplicar la convolución significa deslizar un kernel sobre una señal de entrada que genera una suma ponderada donde los pesos son los valores dentro del kernel. La zancada es el paso deslizante. En el caso de usar una convolución con un paso de dos por dos (zancada doble), el paso es dos en ambas direcciones x e y, el resultado de una convolución con paso dos, divide a la mitad el ancho y la altura de la entrada, con esto se renuncia a cierta información al hacerlo, es un compromiso entre el consumo de recursos (ya sea CPU o memoria) y la información recuperada.

En 2014, se publicó un artículo [15] que demostró que reemplazar las capas de agrupación por convoluciones estriadas puede aumentar la precisión en algunas situaciones. De esta forma se reemplazan las típicas capas de agrupación máxima por capas de convolución con zancada doble utilizando el mismo tamaño de filtro para reducir el tamaño del modelo y mejorar la precisión de la CNN. Cabe destacar que la capa de

agrupación no tiene parámetros, análogamente, la capa de convolución estriada tiene pesos y sesgos para optimizar. Luego, la CNN puede aprender a resumir los datos.

5.6.3. Capa de Agrupación

Este tipo de capa realiza un muestreo descendente de su entrada, lo que reduce la dimensión de los datos que fluyen a través de la red, lo que lo hace más manejable y, con suerte, ayuda en el proceso de extracción de información útil de la entrada inicial. La capa de agrupación más común es la *capa de agrupación máxima*, que desliza una ventana de tamaño $p \; x \; p$ a través de la entrada e informa a la siguiente capa solo el valor máximo de dicha ventana.

5.6.4. Aplastar

Entre las capas convolucionales y las necesarias capas densas, hay una capa de nombre 'Aplastar'. Esta capa transforma una matriz bidimensional de características en un vector que puede alimentar o, en otras palabras, ser la entrada de un clasificador de red neuronal denso.

5.6.5. Capas Densas

Luego vienen capas densas, en la que los resultados de las capas convolucionales se alimentan a través de dos capas neurales para generar una predicción, la cantidad de neuronas de la última capa densa depende de la cantidad de clases entre las cuales tiene que decidir nuestro clasificador que, para efectos de esta tesis, son dos.

5.7. Compilación del modelo

Una vez ya definido el modelo, se debe compilar, esto crea las estructuras eficientes utilizadas por el backend subyacente (TensorFlow) para ejecutar eficientemente el modelo durante el entrenamiento. Se compila el modelo utilizando la función *compile(),* que debe recibir tres atributos importantes:

- Modelo optimizador.
- Función de pérdida.
- Métrica.

El optimizador es la técnica de búsqueda utilizada para actualizar los pesos en el modelo, se puede crear un objeto optimizador y pasarlo a la función de compilación a través del

argumento optimizador. Esto permite configurar el procedimiento de optimización con argumentos propios, como la tasa de aprendizaje, sin embargo, se utiliza el optimizador Adam que está dentro de los optimizadores predeterminados de *TensorFlow.* Adam es un algoritmo de optimización de la tasa de aprendizaje adaptativo que ha sido diseñado específicamente para entrenar redes neuronales profundas.

Por otro lado, la función de pérdida, también llamada función objetivo, es la evaluación del modelo utilizado por el optimizador para navegar por el espacio de los pesos, se puede especificar el nombre de la función de pérdida para usar en la función de compilación mediante el argumento de pérdida, en la presente tesis, se obtuvieron mejores resultados trabajando con 'Categorical crossentropy'.

Por último, las métricas son evaluadas por el modelo durante el entrenamiento, por el momento la métrica admisible es la *precisión,* ya que se está realizando una clasificación. Cabe destacar que para efectos de esta tesis el modelo se entrena en matrices *NumPy* utilizando la función *fit().*

5.8. Entrenamiento del modelo

En el entrenamiento, además de entregarle al modelo las imágenes obtenidas de *Aumento de Datos* y sus respectivas etiquetas, se especifica la cantidad de épocas para entrenar y el tamaño de los lotes.

- Las épocas es el número de veces que el modelo está expuesto al conjunto de datos de entrenamiento, en el modelo a entrenar se ocupan 200 épocas, pero esto va variando según el comportamiento del modelo.
- El tamaño de lote es el número de instancias de entrenamiento que se muestran al modelo antes de realizar una actualización de peso, en el modelo a entrenar se ocupa 40 como tamaño de lote.

En el entrenamiento se puede establecer el valor *validation_split* para retener una fracción del conjunto de datos de entrenamiento para que la validación del modelo se evalúe cada época, o se proporciona una dupla *validation_data* de (x, y) de datos para evaluar, como es el caso de esta tesis. Ajustar el modelo devuelve un objeto de historial con detalles y métricas calculadas para el modelo en cada época. Esto se puede usar para graficar el rendimiento del modelo.

Una vez entrenado el modelo, se puede usar para hacer predicciones sobre datos de prueba o datos nuevos. Hay varios tipos de salida diferentes que se pueden calcular a partir del modelo entrenado, cada uno calculado utilizando una llamada de función diferente en el objeto modelo.

5.9. Verificaciones al rendimiento del modelo

Para conocer que tan bien está realizando la clasificación el modelo, se utilizan las herramientas disponibles en *Keras*, esta es una biblioteca bastante útil en *Python* que proporciona una interfaz limpia para crear modelos de aprendizaje profundo y envuelve los *backends* más técnicos de *TensorFlow*. *Keras* ofrece la capacidad de registrar devoluciones de llamada cuando se entrena un modelo de aprendizaje profundo.

Una de las devoluciones de llamada predeterminadas que se registra al entrenar a todos los modelos de aprendizaje profundo es la devolución de llamada de *Historial*. Registra métricas de entrenamiento para cada época. Esto incluye la pérdida o error y la precisión (para problemas de clasificación) en el entrenamiento, así como la pérdida o error y la precisión en el conjunto de datos de validación, si se establece uno.

Contando con estos datos, se puede crear un gráfico de comparación entre la pérdida del modelo asociado a entrenamiento y pérdida asociada a validación, así como también se puede crear un gráfico de comparación entre la precisión de entrenamiento y la precisión de validación.

A partir de la gráfica de precisión, se puede evaluar si es que el modelo puede entrenarse un poco más, si es que la tendencia de precisión en ambos conjuntos de datos sigue aumentando durante las últimas épocas. También se puede ver si es que el modelo aún no ha superado al conjunto de datos de entrenamiento, lo que indicaría una habilidad comparable en ambos conjuntos de datos, en la figura 5.2 se muestra un ejemplo de grafica de precisión a medida que se avanza en las épocas.

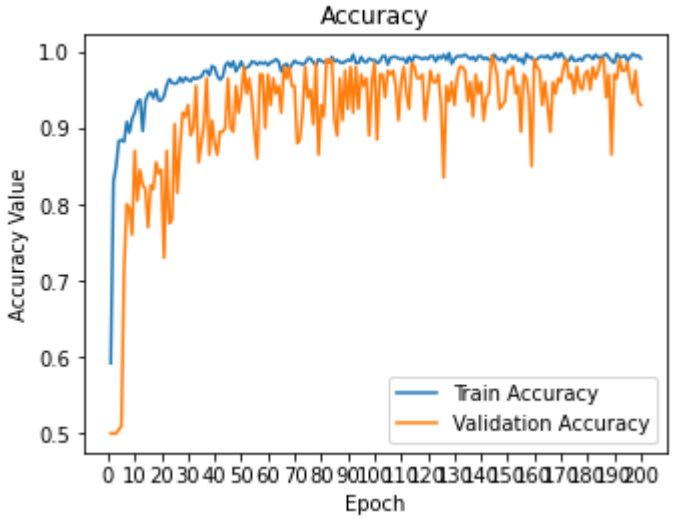


Figura 5.2 Ejemplo de grafica de precisión.

A partir de la gráfica de pérdida, se puede dilucidar si es que el modelo tiene un rendimiento comparable en los conjuntos de datos de entrenamiento y validación (prueba etiquetada). Si estas parcelas paralelas comienzan a separarse consistentemente, podría

ser una señal de dejar de entrenar en una época anterior. En la figura 5.3 se muestra un ejemplo de grafica de perdida a medida que se avanza en las épocas.

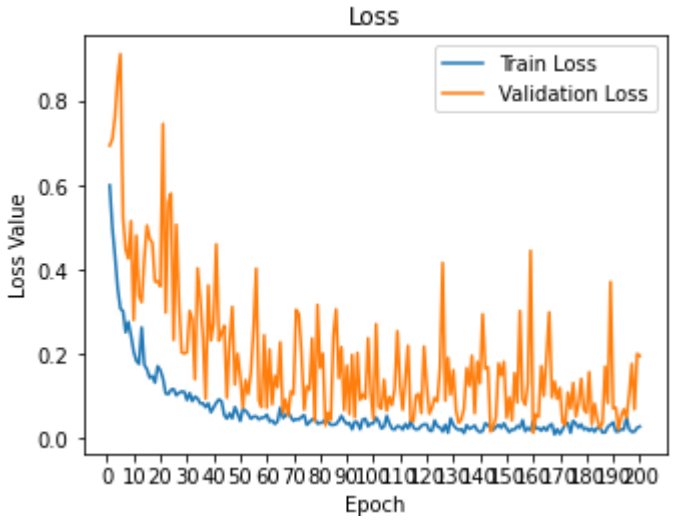


Figura 5.3 Ejemplo de grafica de perdida.

Se debe tener conciencia de que los problemas de clasificación son aquellos en los que el modelo aprende un mapeo entre las características de entrada y una característica de salida que es una etiqueta, como "Con Grieta" y "Sin Grieta". Es así como se puede predecir la clase para nuevas instancias de datos usando el modelo de clasificación finalizado en *Keras* usando la función *predict_classes ()*, posterior a realizar la predicción a todo el set de testeo, se deben procesar levemente los resultados de este, puesto que las predicciones nos devuelven su clasificación en números decimales y no en números enteros como se le fueron entregados, esto ocasionado por las técnicas de optimización utilizadas a lo largo del entrenamiento, este procesamiento se lleva a cabo con *np.argmax*, ya que esto devuelve los índices de los valores máximos a lo largo del eje “y” del vector de entrada, quedando nuevamente en forma categórica, con un 1 si reconoció una grieta y un 0 si es que no reconoció grietas en la imagen.

Ahora bien, para verificar de manera gráfica los resultados, se genera la matriz de confusión asociada al modelo, en el campo del aprendizaje automático y, específicamente, en el problema de la clasificación estadística, una matriz de confusión, también conocida como matriz de error, es una tabla que a menudo se usa para describir el rendimiento de un modelo de clasificación (o "clasificador") en un conjunto de datos de prueba para los que se conocen los valores verdaderos.

Esta permite una fácil identificación de la confusión entre clases, la mayoría de las medidas de rendimiento se calculan a partir de la matriz de confusión. El número de predicciones correctas e incorrectas se resume con valores de conteo y se desglosa por clase. Esta es la clave de la matriz de confusión. Esta muestra las formas en que un modelo de clasificación se confunde cuando hace predicciones. Brinda una idea no solo de los errores que está cometiendo el clasificador, sino más importante aún, de los tipos

de errores que se están cometiendo. A continuación, en la tabla 5.3 se presenta un ejemplo de esta:

Tabla 5.3 Esquema de matriz de confusión.

	Clase 1 Predicción	Clase 2 Predicción
Clase 1 Real	TP	FN
Clase 2 Real	FP	TN

Definición de los términos:

- Positivo (P): la observación es positiva.
- Negativo (N): la observación no es positiva.
- Verdadero positivo (TP): la observación es positiva y se predice que será positiva.
- Falso negativo (FN): la observación es positiva, pero se predice negativa.
- Verdadero negativo (TN): la observación es negativa y se predice que será negativa.
- Falso positivo (FP): la observación es negativa, pero se predice que es positiva.

5.10. Aprendizaje por Transferencia

El aumento de la popularidad y el uso de técnicas de redes neuronales de aprendizaje profundo se remonta a las innovaciones en la aplicación de redes neuronales convolucionales a las tareas de clasificación de imágenes.

Algunas de las innovaciones más importantes surgieron de presentaciones de académicos y líderes de la industria al *Desafío de reconocimiento visual a gran escala de ImageNet* o ILSVRC.

ImageNet es un gran conjunto de datos de fotografías etiquetadas destinadas a investigación en *Visión por Computadora*. El objetivo de desarrollar este conjunto de datos fue proporcionar un recurso para promover la investigación y el desarrollo de métodos mejorados para *Visión por Computadora*.

Basado en estadísticas sobre el conjunto de datos registrado en la página de inicio de ImageNet hay un poco más de 14 millones de imágenes en el conjunto de datos, un poco más de 21 mil grupos o clases, y un poco más de 1 millón de imágenes que tienen anotaciones de cuadro de límite, es decir, cuadros alrededor de los objetos identificados en las imágenes.

Las fotografías fueron etiquetadas por humanos usando plataformas de *crowdsourcing* como el *Amazon Mechanical Turk* de Amazon.

El proyecto para desarrollar y mantener el conjunto de datos fue organizado y ejecutado por una colaboración entre académicos en Princeton, Stanford y otras universidades estadounidenses.

El desafío de reconocimiento visual a gran escala de *ImageNet* (ILSVRC) evalúa algoritmos para la detección de objetos y la clasificación de imágenes a gran escala. Una motivación de alto nivel es permitir que los investigadores comparen el progreso en la detección en una variedad más amplia de objetos, aprovechando el esfuerzo de etiquetado bastante costoso. Otra motivación es medir el progreso de *Visión por Computadora* para la indexación de imágenes a gran escala para su recuperación y anotación.

Lo interesante es que los modelos concursantes con mejores rendimientos de clasificación en el conjunto de datos de validación de ImageNet están disponibles en la documentación de *Keras* junto con sus pesos preentrenados, con esto se puede configurar un modelo propio que aproveche la capacidad de extracción de características de estos modelos.

Al momento de cargar un modelo preentrenado en el cuaderno de Google Colaboratory, se puede establecer que todos sus pesos y parámetros permanezcan intactos durante el entrenamiento o se puede establecer que tengan libertad para adecuarse a los nuevos datos que se le están entregando, luego de corroborar con cual configuración estos modelos presentan mejor desempeño, para efectos de esta tesis se descongelan tan solo las últimas 5 capas de los modelos preentrenados utilizados para entrenamiento, para posteriormente visualizar la precisión de clasificación del modelo resultante con la ayuda de una matriz de confusión.

Cabe destacar que *Aprendizaje por Transferencia* es compatible con la técnica *Aumento de Datos* por lo que también se le está sacando el máximo provecho al set de entrenamiento.

Este proceso se realiza con la finalidad de tener otra opción para crear un modelo detector de grietas, de esta forma se cuenta con resultados comparables con los resultados de clasificación obtenidos de los modelos de confección totalmente propia, en la sección 6.3 se encuentran los resultados de esta metodología.

5.11. Búsqueda de Cuadrícula

El método *Búsqueda de Cuadrícula* corresponde al método tradicional para el ajuste de los hiper-parámetros de la arquitectura y del entrenamiento.

El método es simplemente una búsqueda exhaustiva variando los valores y combinaciones de los hiper-parámetros de forma automática en búsqueda de alguna combinación de hiper-parámetros óptima que minimice la función de costos y en consecuencia, genere la mejor matriz de confusión posible partiendo de un conjunto de parámetros predefinidos en cuanto a límites y pasos.

6. Resultados

En este capítulo se presentan los resultados obtenidos en la búsqueda de un modelo que presente un buen desempeño en clasificación en el set de testeo, partiendo desde un modelo convolucional sin reducción de dimensiones en las imágenes de entrada presente en la sección 6.1, pasando a un modelo en el que se realiza reducción de dimensiones presente en la sección 6.2 y finalizando con modelos en los cuales se utiliza *Aprendizaje por Transferencia* presentes en la sección 6.3.

Además, dado el evidente potencial de mejorar la precisión de clasificación del modelo con reducción de dimensiones en las imágenes de entrada, se realiza una *Búsqueda de Cuadrícula* con este y sus resultados se exhiben en la sección 6.4.

6.1. Modelo Convolucional sin Reducción de Dimensiones

Posterior a una extensa búsqueda de un modelo que posea la menor cantidad de falsos positivos y falsos negativos en su matriz de confusión, trabajando con las imágenes en sus dimensiones originales, 128 pixeles de alto y 128 pixeles de ancho, el modelo presente en la tabla 6.1 es el que presentó mejores resultados.

6.1.1. Arquitectura

Los parámetros de la arquitectura definida para este modelo se presentan en la tabla 6.1.

Tabla 6.1 Parámetros arquitectura del modelo convolucional sin reducción de dimensiones.

Estructura	Hiper-Parámetro	Valor
Normalización por lotes	-	-
Convolución 2D	Número de filtros de salida	8
	Función de activación	relu
	Tamaño de núcleo	3 x 3
Convolución 2D	Número de filtros de salida	8
	Función de activación	relu
	Tamaño de núcleo	3 x 3
	Zancada	2 x 2
Normalización por lotes	-	-
Convolución 2D	Número de filtros de salida	4
	Función de activación	relu
	Tamaño de núcleo	3 x 3
Convolución 2D	Número de filtros de salida	4
	Función de activación	relu
	Tamaño de núcleo	3 x 3
	Zancada	2 x 2
Convolución 2D	Número de filtros de salida	1
	Función de activación	relu
	Tamaño de núcleo	3 x 3
Aplanar	-	-
Completamente conectada	Unidades	64
	Función de activación	relu
Abandono	Razón	0.5
Completamente conectada	Unidades	2
	Función de activación	sigmoid

6.1.2. Parámetros de Entrenamiento

Los parámetros de entrenamiento definidos para este modelo se presentan en la tabla 6.2.

Tabla 6.2 Parámetros entrenamiento del modelo convolucional sin reducción de dimensiones.

Hiper-Parámetro	Valor
Optimizador Adam	0.001
Pérdida	Entropía cruzada binaria
Métrica	Precisión

6.1.3. Historial de Entrenamiento

A continuación, se muestra el gráfico de precisión en los conjuntos de entrenamiento y validación sobre las épocas de entrenamiento y el gráfico de pérdida en los conjuntos de entrenamiento y validación durante las épocas de entrenamiento.

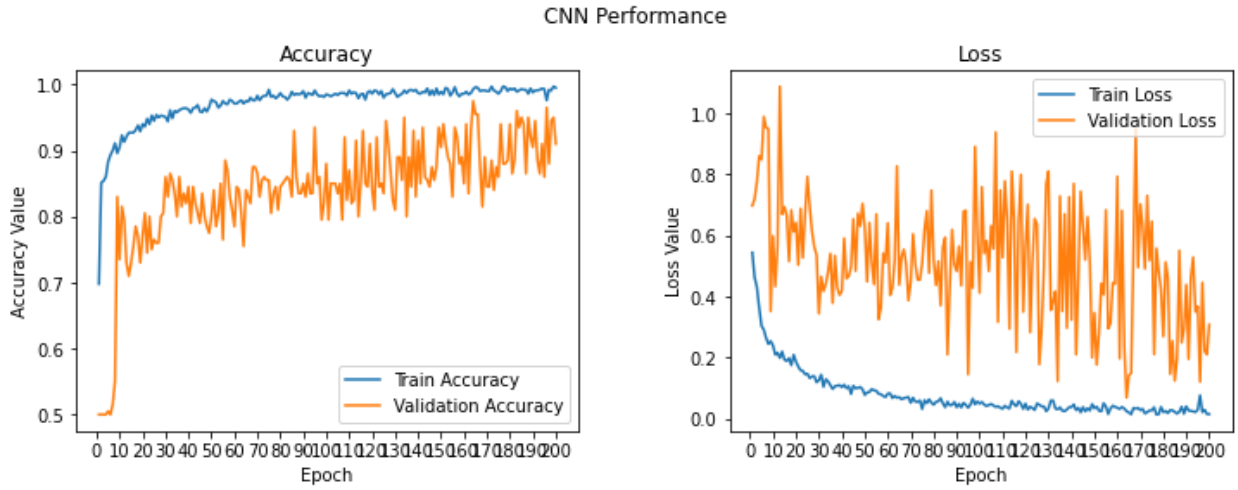


Figura 6.1 Gráficos con el historial de entrenamiento del modelo.

6.1.4. Matriz de Confusión

La matriz de confusión generada con los resultados de la clasificación efectuada en el set de testeo por el modelo definido anteriormente se presenta en la figura 6.2.

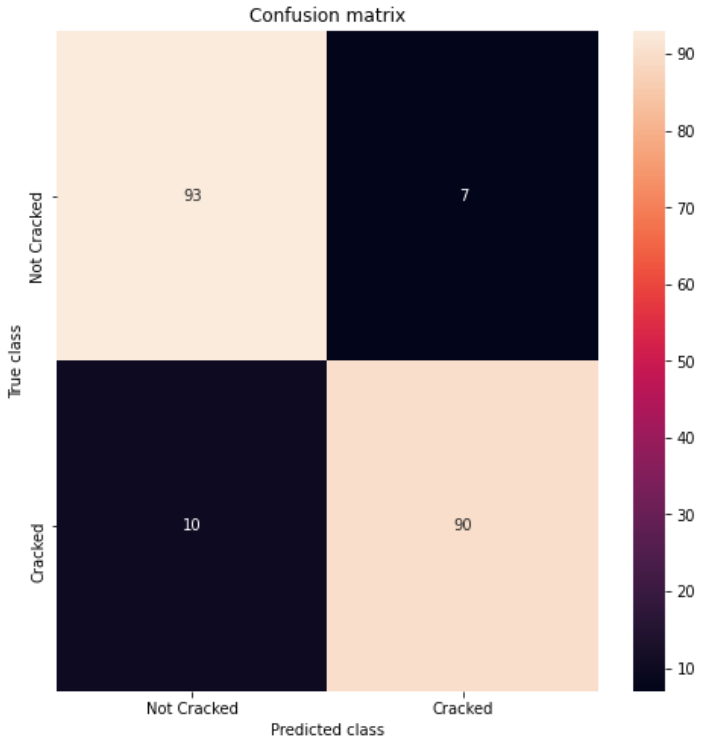


Figura 6.2 Matriz de confusión resultante del testeo del modelo.

6.2. Modelo Convolucional con Reducción de Dimensiones

Posterior a una extensa búsqueda de un modelo que posea la menor cantidad de falsos positivos y falsos negativos en su matriz de confusión, trabajando con las imágenes con sus dimensiones reducidas, 95 pixeles de alto y 95 pixeles de ancho, el modelo presente en la tabla 6.3 es el que presentó mejores resultados.

6.2.1. Arquitectura

Los parámetros de la arquitectura definida para este modelo se presentan en la tabla 6.3.

Tabla 6.3 Parámetros arquitectura del modelo convolucional con reducción de dimensiones.

Estructura	Hiper-Parámetro	Valor
Normalización por lotes	-	-
Convolución 2D	Número de filtros de salida	12
	Función de activación	relu
	Tamaño de núcleo	3 x 3
	Regularizador de núcleo	L2
Agrupación máxima 2D	Tamaño de núcleo	2 x 2
Abandono	Razón	0.25
Convolución 2D	Número de filtros de salida	6
	Función de activación	relu
	Tamaño de núcleo	3 x 3
	Regularizador de núcleo	L2
Convolución 2D	Número de filtros de salida	6
	Función de activación	relu
	Tamaño de núcleo	3 x 3
	Regularizador de núcleo	L2
	Zancada	2 x 2
Abandono	Razón	0.25
Convolución 2D	Número de filtros de salida	1
	Función de activación	relu
	Tamaño de núcleo	3 x 3
Aplanar	-	-
Completamente conectada	Unidades	64
	Función de activación	relu
	Regularizador de núcleo	L2
Abandono	Razón	0.5
Completamente conectada	Unidades	2
	Función de activación	sigmoid

6.2.2. Parámetros de Entrenamiento

Los parámetros de entrenamiento definidos para este modelo se presentan en la tabla 6.4.

Tabla 6.4 Parámetros entrenamiento del modelo convolucional con reducción de dimensiones.

Hiper-Parámetro	Valor
Optimizador Adam	0.001
Pérdida	Entropía cruzada categórica
Métrica	Precisión

6.2.3. Historial de Entrenamiento

A continuación, se muestra el gráfico de precisión en los conjuntos de entrenamiento y validación sobre las épocas de entrenamiento y el gráfico de pérdida en los conjuntos de entrenamiento y validación durante las épocas de entrenamiento.

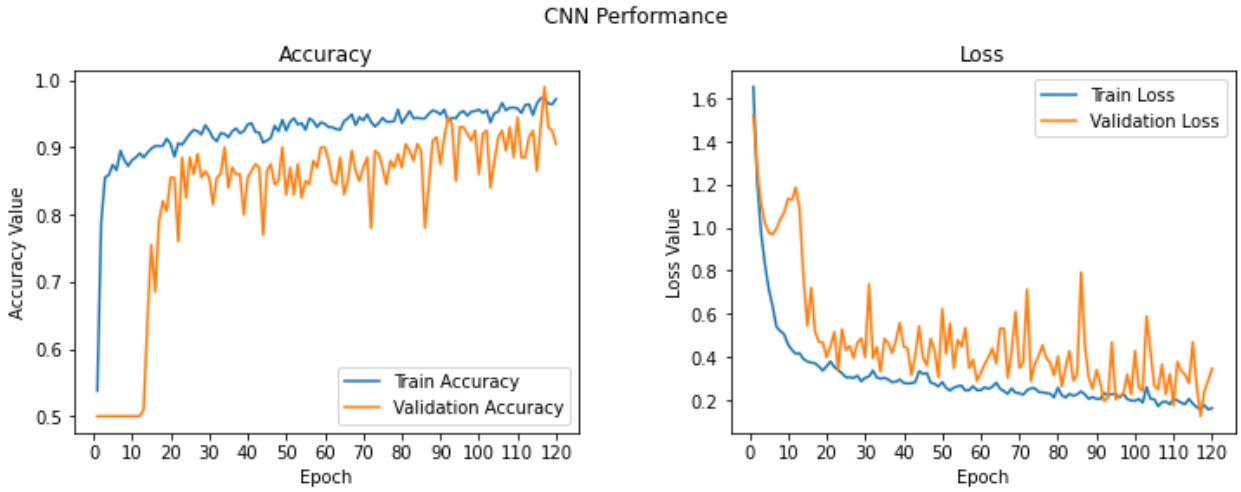


Figura 6.3 Gráficos con el historial de entrenamiento del modelo.

6.2.4. Matriz de Confusión

La matriz de confusión generada con los resultados de la clasificación efectuada en el set de testeo por el modelo definido anteriormente es la presente en la figura 6.4.

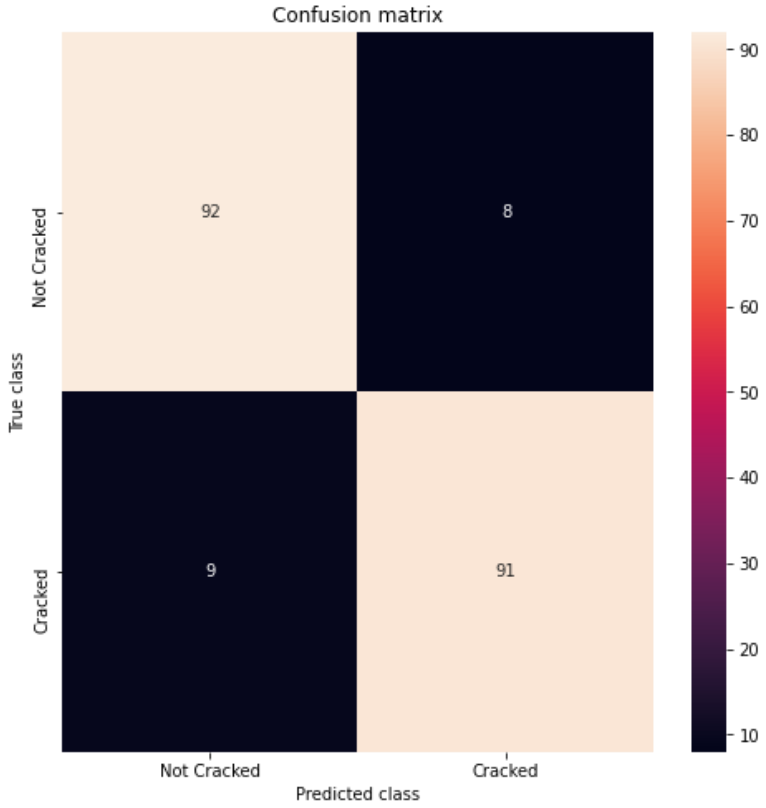


Figura 6.4 Matriz de confusión resultante del testeo del modelo.

6.3. Modelos Convolucionales con Aprendizaje por Transferencia

Posterior a probar todos los modelos preentrenados disponibles en la documentación de *Keras*, se presentan a continuación los dos mejores modelos que poseen la menor cantidad de falsos positivos y falsos negativos en su matriz de confusión, el modelo presente en la tabla 6.5 es el que presentó mejores resultados con VGG16 como modelo preentrenado.

6.3.1. Arquitectura

Los parámetros de la arquitectura definida para recibir la extracción de características del modelo preentrenado se presentan en la tabla 6.5.

Tabla 6.5 Parámetros arquitectura del modelo que continua el modelo preentrenado.

Estructura	Hiper-Parámetro	Valor
Aplanar	-	-
Completamente conectada	Unidades	64
	Función de activación	relu
	Regularizador de núcleo	L2
Abandono	Razón	0.5
Completamente conectada	Unidades	2
	Función de activación	sigmoid

6.3.2. Parámetros de Entrenamiento

Los parámetros de entrenamiento definidos para este modelo se presentan en la tabla 6.6.

Tabla 6.6 Parámetros entrenamiento del modelo con aprendizaje por transferencia.

Hiper-Parámetro	Valor
Optimizador Adam	0.001
Pérdida	Entropía cruzada categórica
Métrica	Precisión

6.3.3. Historial de Entrenamiento

A continuación, se muestra el gráfico de precisión en los conjuntos de entrenamiento y validación sobre las épocas de entrenamiento y el gráfico de pérdida en los conjuntos de entrenamiento y validación durante las épocas de entrenamiento.

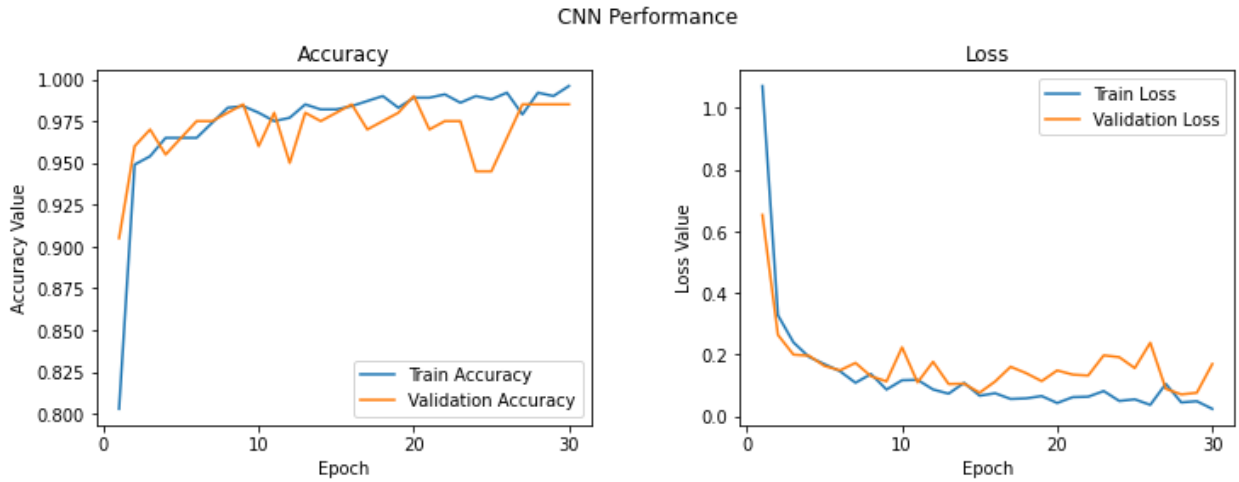


Figura 6.5 Gráficos con el historial de entrenamiento del modelo.

6.3.4. Matriz de Confusión

La matriz de confusión generada con los resultados de la clasificación efectuada en el set de testeo por el modelo definido anteriormente es la presente en la figura 6.6.

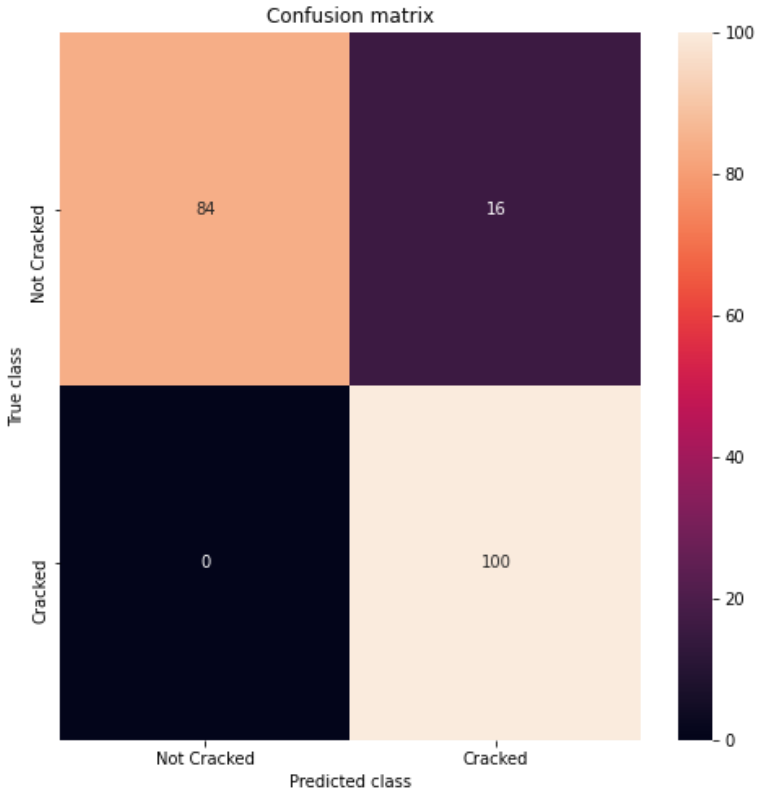


Figura 6.6 Matriz de confusión resultante del testeo del modelo.

Posterior a una extensa búsqueda de modelos que utilicen la VGG19 como modelo preentrenado que posean la menor cantidad de falsos positivos y falsos negativos en su matriz de confusión, el modelo presente en la tabla 6.7 es el que presentó mejores resultados.

6.3.5. Arquitectura

Los parámetros de la arquitectura definida para recibir la extracción de características del modelo preentrenado se presentan en la tabla 6.7.

Tabla 6.7 Parámetros arquitectura del modelo que continua el modelo preentrenado.

Estructura	Hiper-Parámetro	Valor
Aplanar	-	-
Completamente conectada	Unidades	64
	Función de activación	relu
	Regularizador de núcleo	L2
Abandono	Razón	0.5
Completamente conectada	Unidades	2
	Función de activación	softmax

6.3.6. Parámetros de Entrenamiento

Los parámetros de entrenamiento definidos para este modelo se presentan en la tabla 6.8.

Tabla 6.8 Parámetros entrenamiento del modelo con aprendizaje por transferencia.

Hiper-Parámetro	Valor
Optimizador Adam	0.001
Pérdida	Entropía cruzada categórica
Métrica	Precisión

6.3.7. Historial de Entrenamiento

A continuación, se muestra el gráfico de precisión en los conjuntos de entrenamiento y validación sobre las épocas de entrenamiento y el gráfico de pérdida en los conjuntos de entrenamiento y validación durante las épocas de entrenamiento.

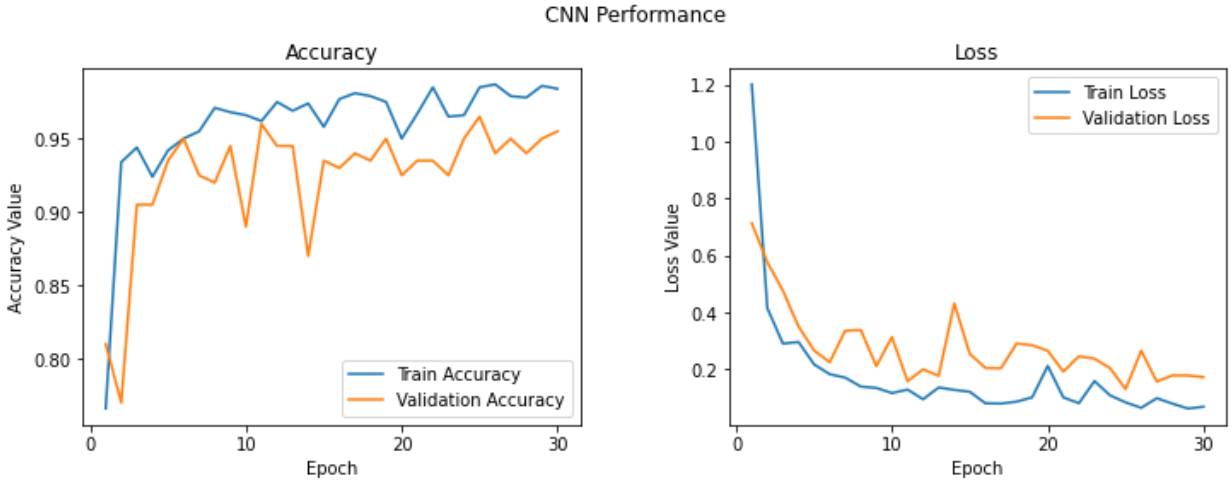


Figura 6.7 Gráficos con el historial de entrenamiento del modelo.

6.3.8. Matriz de Confusión

La matriz de confusión generada con los resultados de la clasificación efectuada en el set de testeo por el modelo definido anteriormente es la presente en la figura 6.8.

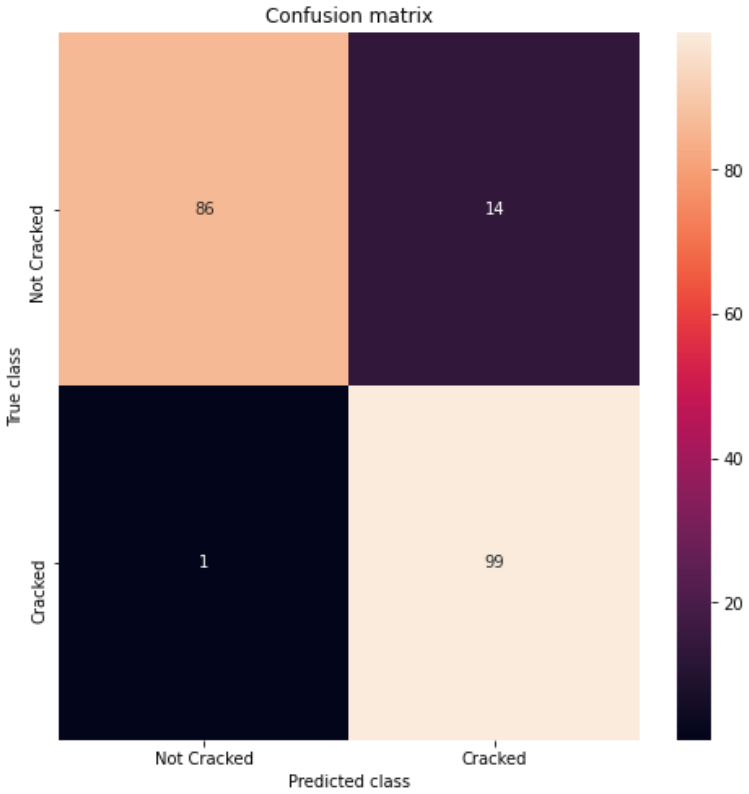


Figura 6.8 Matriz de confusión resultante del testeo del modelo.

6.4. Mejor Modelo Resultante de la Búsqueda de Cuadrícula

Posterior a la *Búsqueda de Cuadrícula*, que se realizó a partir del mejor modelo convolucional con reducción de dimensiones, se determina que el modelo presente en la tabla 6.9 es el que presenta mejores resultados.

6.4.1. Arquitectura

Los parámetros de la arquitectura encontradas para este modelo se presentan en la tabla 6.9.

Tabla 6.9 Parámetros arquitectura del mejor modelo resultante de la búsqueda de cuadrícula.

Estructura	Hiper-Parámetro	Valor
Normalización por lotes	-	-
Convolución 2D	Número de filtros de salida	12
	Función de activación	relu
	Tamaño de núcleo	4 x 4
	Regularizador de núcleo	L2
Agrupación máxima 2D	Tamaño de núcleo	2 x 2
Abandono	Razón	0.35
Convolución 2D	Número de filtros de salida	6
	Función de activación	relu
	Tamaño de núcleo	4 x 4
	Regularizador de núcleo	L2
Convolución 2D	Número de filtros de salida	6
	Función de activación	relu
	Tamaño de núcleo	4 x 4
	Regularizador de núcleo	L2
	Zancada	2 x 2
Abandono	Razón	0.35
Convolución 2D	Número de filtros de salida	1
	Función de activación	relu
	Tamaño de núcleo	4 x 4
Aplanar	-	-
Completamente conectada	Unidades	64
	Función de activación	relu
	Regularizador de núcleo	L2
Abandono	Razón	0.5
Completamente conectada	Unidades	2
	Función de activación	sigmoid

6.4.2. Parámetros de Entrenamiento

Los parámetros de entrenamiento encontrados para este modelo se presentan en la tabla 6.10.

Tabla 6.10 Parámetros entrenamiento del mejor modelo resultante de la búsqueda de cuadrícula.

Hiper-Parámetro	Valor
Optimizador Adam	0.001
Pérdida	Entropía cruzada categórica
Métrica	Precisión

6.4.3. Historial de Entrenamiento

A continuación, se muestra el gráfico de precisión en los conjuntos de entrenamiento y validación sobre las épocas de entrenamiento y el gráfico de pérdida en los conjuntos de entrenamiento y validación durante las épocas de entrenamiento.

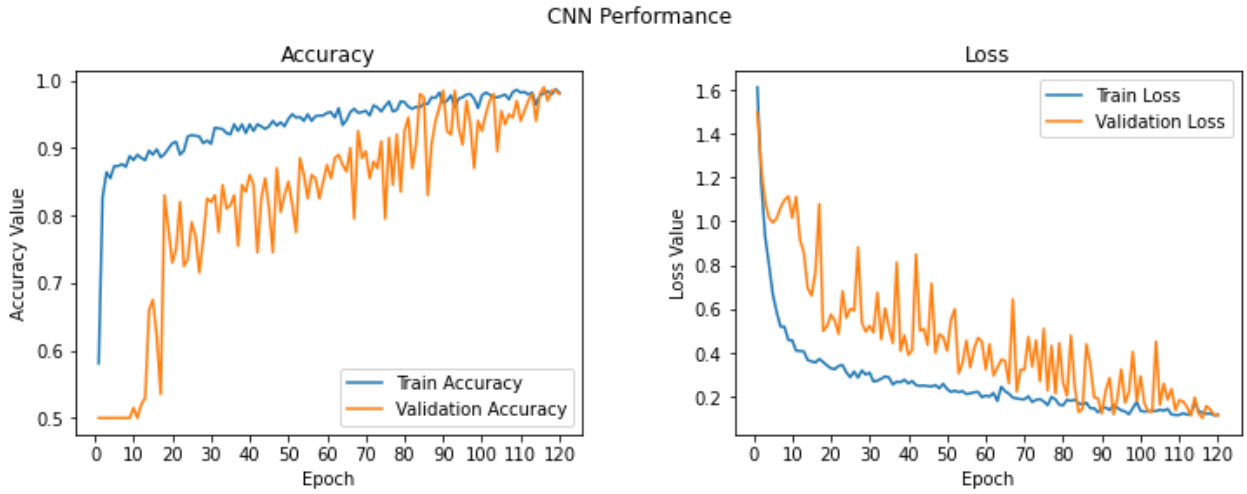


Figura 6.9 Gráficos con el historial de entrenamiento del modelo.

6.4.4. Matriz de Confusión

La matriz de confusión generada por la clasificación efectuada en el set de testeo por el modelo definido anteriormente es la presente en la figura 6.10.

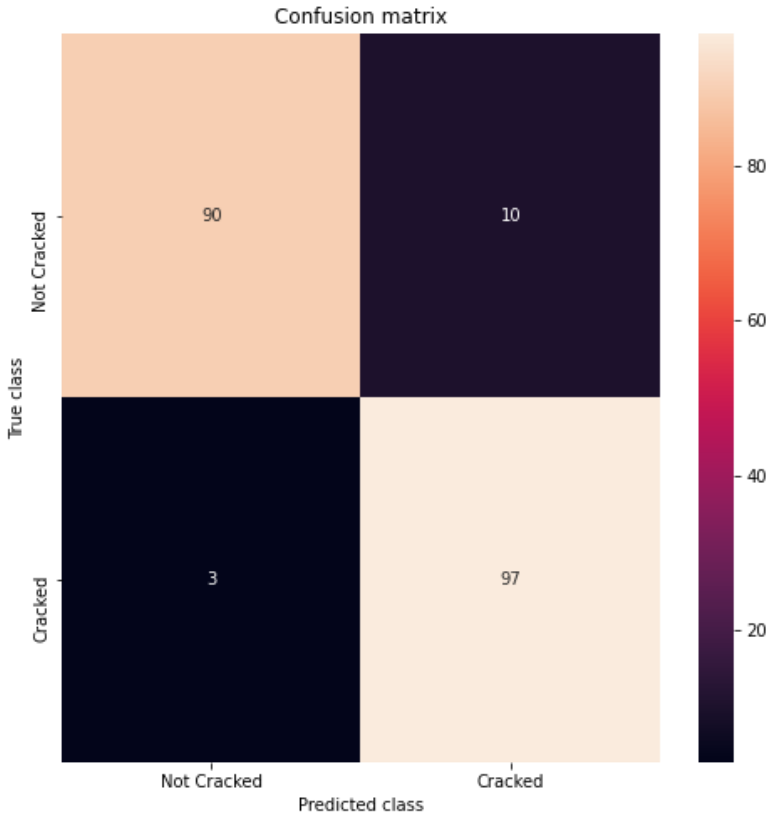


Figura 6.10 Matriz de confusión resultante del testeo del modelo.

7. Análisis de Resultados

Luego de presentar los resultados se procede a realizar un análisis cuantitativo y cualitativo que permita extraer información y conclusiones a partir del trabajo producido.

Cada sección de resultados se analizará en forma particular para luego realizar una comparación de los resultados obtenidos por los modelos.

7.1. Modelo Convolucional sin Reducción de Dimensiones

Durante la búsqueda de un modelo que presentara buenos resultados trabajando con las imágenes en sus dimensiones originales, 128 pixeles x 128 pixeles, se concluye que es difícil hallar una arquitectura con la cual el modelo sea capaz de detectar la grieta. Al ver la figura 6.1 es evidente la inestabilidad que posee el modelo, dado la gran inestabilidad presente en el entrenamiento.

A partir de la gráfica de precisión presente en la figura 6.1, se puede observar que, en el caso de este modelo, no surtiría efecto que fuese entrenado más épocas, ya que la tendencia de precisión en ambos conjuntos de datos permanece casi constante durante las últimas épocas. También se puede ver que el modelo no supera y deja de acercarse al conjunto de datos de entrenamiento, mostrando una habilidad menor al conjunto de entrenamiento en cuanto a precisión, por último, queda en evidencia en la figura 6.1 que el modelo al final de su entrenamiento no llega a una buena precisión.

A partir de la gráfica de pérdida presente en la figura 6.1, se concluye que el modelo no tiene un rendimiento comparable tanto en el conjunto de datos de entrenamiento como de validación, puesto que la pérdida de validación es bastante inestable.

Por otra parte, en la figura 6.2 se logra ver que en la clasificación efectuada con el modelo en el set de testeo, se producen 10 falsos positivos y 7 falsos negativos, los que se interpretan como una precisión del 91.5% por parte del modelo, sin embargo, estos no necesariamente son consecuencia del buen clasificador generado por el modelo, sino que puede estar presentándose el caso de que el modelo está errando en su forma de clasificar, de manera favorable a la clasificación, dado a la gran inestabilidad presente.

7.2. Modelo Convolucional con Reducción de Dimensiones

Durante la búsqueda de un modelo que presentara buenos resultados trabajando con las imágenes en sus dimensiones reducidas, 95 pixeles x 95 pixeles, generando un modelo más pequeño con menos parámetros que optimizar, se concluye que no es dificultoso hallar una arquitectura con la cual el modelo sea capaz de detectar la grieta. Al ver la figura 6.3 se logra identificar una moderada inestabilidad presente en el modelo, también queda en evidencia que el modelo al final de su entrenamiento llega a una buena

precisión, siendo estas las principales razones por las que todos los modelos posteriores son entrenados con las imágenes en sus dimensiones reducidas.

A partir de la gráfica de precisión presente en la figura 6.3, se puede observar que el modelo puede ser entrenado un poco más, ya que la tendencia de precisión en ambos conjuntos de datos sigue aumentando durante las últimas épocas, sin embargo, al momento de entrenar el modelo una mayor cantidad de veces, este pierde su capacidad de detección de grietas obteniéndose malos resultados de clasificación. También queda en evidencia que el modelo aún no supera a el conjunto de datos de entrenamiento en cuanto a precisión, mostrando una habilidad comparable en ambos conjuntos de datos.

A partir de la gráfica de pérdida presente en la figura 6.3, queda claro que el modelo tiene un rendimiento comparable tanto en el conjunto de datos de entrenamiento como de validación.

Por otra parte, en la figura 6.4 se exhibe que en la clasificación efectuada en el set de testeo se producen 9 falsos positivos y 8 falsos negativos, los que se interpretan como una precisión del 91.5% por parte del modelo.

7.3. Modelos Convolucionales con Aprendizaje por Transferencia

7.3.1. Modelo con VGG16

Dada la experiencia acumulada por los numerosos intentos para encontrar la mejor arquitectura, se concluye que no es dificultoso hallar una arquitectura con la cual el modelo sea capaz de detectar la grieta, al ver la figura 6.5 es posible notar una baja sobre-regularización presente en el modelo, también queda en evidencia que el modelo al final de su entrenamiento llega a una buena precisión.

Esta sobre-regularización se puede deber a que la regularización es aplicada durante el entrenamiento, pero no durante la validación, o puede estar dándose el caso que el conjunto de validación sea más fácil de clasificar para el modelo que el conjunto de entrenamiento.

También, a partir de la gráfica de pérdida presente en la figura 6.5, queda claro que el modelo tiene un rendimiento comparable tanto en el conjunto de datos de entrenamiento como de validación.

Por otra parte, en la figura 6.6 se exhibe que en la clasificación efectuada en el set de testeo se producen 0 falsos positivos y 16 falsos negativos, los que se interpretan como una precisión del 92% por parte del modelo.

7.3.2. Modelo con VGG19

Dado la experiencia acumulada por los numerosos intentos para encontrar la mejor arquitectura, se concluye que no es dificultoso hallar una arquitectura con la cual el modelo sea capaz de detectar la grieta, al ver la figura 6.7 es posible notar un leve sub-ajuste que posee el modelo, también queda en evidencia que el modelo al final de su entrenamiento llega a una buena precisión.

También, a partir de la gráfica de pérdida presente en la figura 6.7, queda claro que el modelo tiene un rendimiento comparable tanto en el conjunto de datos de entrenamiento como de validación.

Por otra parte, en la figura 6.8 se exhibe que en la clasificación efectuada en el set de testeo se produce 1 falso positivo y 14 falsos negativos, los que se interpretan como una precisión del 92% por parte del modelo.

7.4. Mejor Modelo Resultante de la Búsqueda de Cuadrícula

Al ver la figura 6.9 es posible notar una baja sobre-regularización presente en el modelo, además de contar con una moderada inestabilidad, queda en evidencia que el modelo al final de su entrenamiento llega a una buena precisión.

Esta sobre-regularización se puede deber a que la regularización es aplicada durante el entrenamiento, pero no durante la validación, o puede estar dándose el caso que el conjunto de validación sea más fácil de clasificar para el modelo que el conjunto de entrenamiento

A partir de la gráfica de precisión presente en la figura 6.9, se puede ver que el modelo probablemente pueda ser entrenado un poco más, ya que la tendencia de precisión en ambos conjuntos de datos sigue aumentando durante las últimas épocas, sin embargo, al momento de entrenar el modelo una mayor cantidad de veces, este pierde su capacidad de detección de grietas obteniéndose resultados deficientes en cuanto a clasificación. También es posible observar que el modelo aún no supera a el conjunto de datos de entrenamiento en cuanto a precisión, mostrando una habilidad comparable en ambos conjuntos de datos.

A partir de la gráfica de pérdida presente en la figura 6.9, queda claro que el modelo tiene un rendimiento comparable tanto en los conjuntos de datos de entrenamiento como de validación.

Por otra parte, en la figura 6.10 se exhibe que en la clasificación efectuada en el set de testeo se producen 3 falsos positivos y 10 falsos negativos, los que se interpretan como una precisión del 93.5% por parte del modelo.

En conclusión, si se habla en términos de estabilidad en el modelo, los modelos que presentan mayor estabilidad son los que se confeccionaron a partir de modelos preentrenados implementando *Aprendizaje por Transferencia*, sin embargo estos modelos no son los que poseen los mejores resultados de clasificación en el set de testeo, el modelo que tiene el mejor rendimiento en este ámbito es el que se obtuvo gracias a la optimización de hiper-parámetros para el modelo convolucional con reducción de dimensiones por medio de una *Búsqueda de Cuadrícula*, logrando obtener tan solo 3 falsos positivos y 10 falsos negativos en su matriz de confusión.

8. Conclusiones

La detección temprana de daños se ha convertido en una tarea esencial en la gestión del estado de sistemas como también en el estado de estructuras, permitiendo a los ingenieros prevenir fallas repentinas en los elementos, reduciendo así los costos al tomar precauciones de seguridad. Sin embargo, no siempre es sencillo analizar los datos disponibles, ya que muchas veces estos tienen desperfectos o parte de ellos no sirve, por lo que se deben eliminar manualmente, requiriendo conocimiento experto y tiempo de procesamiento. Además, un desafío que generalmente se enfrenta al desarrollar herramientas de detección de fallas basadas en técnicas de aprendizaje automático a partir de datos es la baja disponibilidad de estos mismos.

Para el desarrollo de esta investigación, se realizaron ensayos en vigas de hormigón armado preexistentes, con la finalidad de generar el conjunto de datos adecuado para la creación del software que fuese capaz de detectar grietas incipientes en vigas de hormigón armado, dado que no existen datos públicos que se pudiesen utilizar con este fin, de no ser por los ensayos anteriormente mencionados, no habría sido posible llevar a cabo el trabajo de titulación.

A partir del trabajo realizado, los resultados mostrados y el análisis de resultados se procede a enunciar las conclusiones del trabajo.

En primer lugar, se validó la implementación de algoritmos de *Aprendizaje profundo* para la detección de grietas pequeñas o incipientes, puesto que existen trabajos precedentes a este que poseen excelentes desempeños en detección de grietas, pero las grietas que detectan son grietas prominentes, estas ocupan grandes porciones de la imagen con las cuales trabaja, pero no existen hasta el momento investigaciones orientadas a la detección de grietas incipientes.

En cuanto a resultados, se obtuvieron buenos resultados en varios de los modelos entrenados, sin embargo, el modelo convolucional al cual se llegó mediante la *Búsqueda de Cuadrícula* es el que posee mejores resultados en cuanto a detección de grietas contando dentro de sus características una buena estabilidad, por lo que es considerado el mejor modelo al cual se logró llegar en esta investigación.

Por otra parte, es importante mencionar que en todos los modelos entrenados en la búsqueda del poseedor de los mejores resultados, se evidencia inestabilidad al momento de entrenar y validar los modelos, esto se puede solucionar aumentando el tamaño del conjunto de imágenes con los cuales se trabaja, ya que a pesar de contar con un número no menor de imágenes y haber utilizado *Aumento de Datos*, el mejor modelo claramente se puede mejorar, tanto en estabilidad como precisión, por lo que no se puede afirmar que se ha llegado al mejor resultado posible en el área de detección de grietas incipientes en vigas de hormigón armado, lo que sí se puede afirmar es que hasta el momento es un área desatendida y que los buenos resultados obtenidos en este trabajo pueden ser aún mejores.

Con respecto a la precisión obtenida en los diferentes modelos, se pueden establecer dos criterios diferentes para evaluar cual es mejor o peor que otro, uno es el punto de vista matemático en donde solo se evalúa la cantidad de veces que el modelo se equivoca en clasificar una imagen, siendo en este caso el mejor modelo el obtenido mediante la *Búsqueda de Cuadrícula*, en cambio, el otro de punto de vista, tomaría en cuenta en qué casos se está equivocando el modelo en clasificar una imagen, puesto que llevando a producción el software, es mucho más importante que el modelo siempre identifique la presencia de una grieta y se equivoque solo identificando grietas cuando no las hay, que el modelo en ocasiones no detecte las grietas cuando si existen y en otras ocasiones identifique grietas cuando no las hay, siendo en este caso el mejor modelo el obtenido mediante *Aprendizaje por Transferencia* con VGG16 como modelo preentrenado.

Finalmente cabe destacar que, en el área de ingeniería civil, es evidente el potencial y posibles complementos que puede tener este software, dado que el hormigón armado es el material por excelencia en todo tipo de obra civil, además, si al software actual se le agrega, por medio de la implementación de diferentes técnicas, la capacidad de caracterizar las dimensiones de la grieta con tan solo una imagen de esta, se podrían obtener buenas conclusiones en cuanto a la salud estructural del elemento monitoreado.

9. Bibliografía

[1] C. M. Bishop. Pattern recognition and machine learning. Springer, 2006.

[2] K. P. Murphy. Machine learning: a probabilistic perspective. MIT press. 2012.

[3] M. Sokolova y G. Lapalme. A systematic analysis of performance measures for classification tasks. Information Processing & Management, vol. 45, no. 4, pp. 427–437. 2009.

[4] G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems (MCSS), vol. 2, no. 4, pp. 303–314. 1989.

[5] M. A. Nielsen. Neural networks and deep learning. 2015.

[6] E. Alpaydin. Introduction to machine learning. MIT press. 2014.

[7] Cs231n convolutional neural networks for visual recognition. [en línea] https://cs231n.github.io/convolutional-networks/. [consulta: 15 de mayo 2020]

[8] R. Echevarría. Líquidos penetrantes. Universidad Nacional del Comahue. 2003. [en línea] https://web.archive.org/web/20070729110212/http://fain.uncoma.edu.ar/materias/ensayos_no_destructivos/Catedra_END/4-Liquidos%20Penetrantes/LP.pdf. [consulta: 04 de junio 2020].

[9] T. Nguyen y S. Begot. Pavement Cracking Detection Using an Anisotropy Measurement. 11th IASTED International Conference on Computer Graphics and Imaging – CGIM. 17-19 Febrero, Innsbruck, Austria, 2010.

[10] S. Chambon y J. Moliard. Automatic Road Pavement Assessment with Image Processing: Review and Comparison. International Journal of Geophysics, vol. 2011, p. 20. 2011.

[11] H. Oliveira y P. Correia. Identifying and retrieving distress images from road pavement surveys. Workshop on Multimedia Information Retrieval: New Trends and Challenges. 15th IEEE International Conference on Image Processing, ICIP 2008, San Diego, USA, 2008.

[12] Aprende Machine Learning. https://www.aprendemachinelearning.com/machine-learning-en-la-nube-google-colaboratory-con-gpu/. [consulta: 10 de junio 2020].

[13] tf.keras.preprocessing.image.ImageDataGenerator. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator. [consulta: 10 de junio 2020].

[14] A. Marques. Automatic Road Pavement Crack Detection using SVM. Lisboa, Portugal: Dissertation for the Master of Science Degree in Electrical and Computer Engineering at Instituto Superior Técnico. 2012.

[15] J. Springenberg, A. Dosovitskiy, T. Brox y M. Riedmiller. Striving for Simplicity: The All Convolutional Net. 2014.

[16] Gonzales A., J. R. 2014. Transformadas Wavelet impacto fundamental en Procesamiento de señales y compresión de imágenes. Memoria de Licenciado en Matemáticas. Universidad Tecnológica de Pereira. Facultad de Ciencias Básicas. 44p.

[17] Sucar, L. E. y Gomez, G. 2011. Visión Computacional. 100p.

[18] Sucar, L. E. y Gomez, G. 2011. Visión Computacional. 98p.

[19] Métodos de segmentación basados en el análisis del histograma. http://www.varpa.org/~mgpenedo/cursos/Ip/Tema6/nodo6_2.html . [consulta: 07 de septiembre 2020].

[20] Sucar, L. E. y Gomez, G. 2011. Visión Computacional. 35p.

[21] Manual de uso de sensor U-GAGE S18U con salida análoga. http://info.bannerengineering.com/cs/groups/public/documents/literature/110738.pdf. [consulta: 07 de septiembre 2020].

[22] Biblioteca de código abierto para aprendizaje automático. https://www.tensorflow.org/. [consulta: 07 de septiembre 2020].

[23] Biblioteca de redes neuronales de código abierto escrita en Python. https://keras.io/. [consulta: 07 de septiembre 2020].