UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
Facultad de Ingeniería, Arquitectura y Diseño
Programa de Maestría y Doctorado en Ciencias e Ingeniería

SISTEMA DE DETECCIÓN DE COMPONENTES DE ZINC EN UNA LÍNEA DE ENSAMBLE MANUAL BASADO EN REDES NEURONALES CONVOLUCIONALES

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN INGENIERÍA

presenta:

EDGAR RENE RAMOS ACOSTA

Director de tesis

Dr. Everardo Inzunza González

Ensenada, Baja California, México. Junio de 2022.

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ingeniería, Arquitectura y Diseño

SISTEMA DE DETECCIÓN DE COMPONENTES DE ZINC EN UNA LÍNEA DE ENSAMBLE MANUAL BASADO EN REDES NEURONALES CONVOLUCIONALES

TESIS

que para obtener el grado de MAESTRO en INGENIERÍA presenta:

EDGAR RENE RAMOS ACOSTA

Y aprobada por el siguiente comité:

Dr. Everardo Inzunza González
Director de Tesis

Dr. Enrique Efrén García Guerrero
Co-Director de Tesis

Dra. Eloísa del Carmen García Canseco
Miembro del Comité

Dr. Oscar Roberto López Bonilla
Miembro de Comité

Dr. Ulises Jesús Tamayo Pérez
Miembro del Comité

Junio de 2022

RESUMEN de la tesis de **Edgar Rene Ramos Acosta**, presentada como requisito parcial para obtener el grado de MAESTRO EN INGENIERÍA, del programa de Maestría y Doctorado en Ciencias e Ingeniería de la UABC. Ensenada, B. C. México, Junio de 2022.

SISTEMA DE DETECCIÓN DE COMPONENTES DE ZINC EN UNA LÍNEA DE ENSAMBLE MANUAL BASADO EN REDES NEURONALES CONVOLUCIONALES

Resumen aprobado por:

Dr. Everardo Inzunza González
Director de Tesis

Dr. Enrique Efrén García Guerrero
Co-director de Tesis

El uso de la Inteligencia Artificial (IA) en la industria ha aumentado en los últimos años debido a la cuarta revolución industrial, donde la IA ha sido la nueva estrategia para resolver los problemas más complejos relacionados con el *big data*, el reconocimiento de imágenes, la detección de objetos y otros problemas de clasificación y predicción. En este trabajo de tesis de maestría se presenta el desarrollo e implementación de un sistema de detección de componentes de zinc en una línea de ensamble manual dentro de un proceso de manufactura, utilizando técnicas de Aprendizaje Profundo (DL, por sus siglas en inglés). El objetivo de este trabajo es entrenar y evaluar cinco algoritmos de Aprendizaje Profundo (DL, por sus siglas en inglés) para detectar cinco componentes zincados y una fixtura de ensamble, bajo distintas condiciones de luz ambiental y distintos acabados. Se crea un conjunto de datos personalizado, cuya caracteristica inicial es de 1542 imágenes distribuidas entre seis clases diferentes que corresponden a los componentes de montaje. Mismas, que fueron procesadas posteriormente mediante una técnica de aumentación de datos de imágen de manera aleatoria para cada imágen. Se entrenan y evalúan bajo la técnica de aprendizaje por transferencia y una proporción 80-20, las arquitecturas InceptionResNetV2, Xception, InceptionV3, ResNet101V2 y ResNet152V2, con seis neuronas artificiales en la capa de salida. De los resultados obtenidos, es el modelo InceptionV3 el que arroja la mayor exactitud en las pruebas, con un 99.3% y una precisión del 97.9%, mientras que InceptionResNetV2 arroja un 96% en exactidud y una precisión del 87.9%. Se concluye que estos algoritmos pueden ser adoptados como buenos modelos clasificadores para la detección de componentes zincados en una línea de producción industrial.

Palabras clave: Aprendizaje profundo, Visión artificial, Visión profunda por computadora, Aprendizaje automático, Red Neuronal Convolucional, MobileNetV2, Convnet, Xception, Inception, ResNet, Clasificiación de imágenes.

ABSTRACT of the thesis of **Edgar Rene Ramos Acosta**, presented as a partial requirement to obtain the degree of MASTER IN ENGINEERING, of the program of MSc and PhD in Sciences and Engineering of UABC. Ensenada, B. C., Mexico, June 2022.

ZINC COMPONENT DETECTION SYSTEM FOR A MANUAL ASSEMBLY LINE BASED ON CONVOLUTIONAL NEURAL NETWORKS.

Abstract approved by:

Dr. Everardo Inzunza González
Thesis Supervisor

Dr. Enrique Efrén García Guerrero
Thesis Co-Supervisor

The use of Artificial Intelligence (AI) in industry has increased in recent years due to the fourth industrial revolution, where AI has been the new strategy to solve the most complex problems related to big data, image recognition, object detection and other classification and prediction problems. In this master thesis work, we present the development and implementation of a zinc component detection system in a manual assembly line within a manufacturing process, using Deep Learning techniques. The objective of this work is to to train and evaluate five Deep Learning (DL) algorithms for detecting the five zinc plated components and one assembly fixture, under different ambient light conditions and different finishes. A customized dataset is created. The initial characteristic is 1542 images distributed among six different classes corresponding to the assembly components. The images were then processed using a randomized image data augmentation technique for each image. They were trained and evaluated under the transfer learning technique and an 80-20 ratio over the architectures InceptionResNetV2, Xception, InceptionV3, ResNet101V2 and ResNet152V2, with six artificial neurons in the output layer. From the results obtained, it is the InceptionV3 model which scored with the highest accuracy in the tests, with 99.3% and an accuracy of 97.97%, while InceptionResNetV2 showed 96% accuracy and an accuracy of 87.9%. It is concluded that these algorithms can be adopted as good classifier models for the detection of zinc plated components in an industrial production line.

Keywords: Deep-Learning, artificial vision, deep computer vision, Machine Learning, Convolutional Neural Network, MobileNetv2, Convnet, Xception, Inception, ResNet, Image classification.

A mi familia

Agradecimientos

*Al **Dr. Everardo Inzunza González, Dr.Enrique Efrén García Guerrero**, por haberme brindado su apoyo, guía y revisión durante mis estudios de maestría*

*Al **Dr. Oscar Roberto López Bonilla**, por ser uno de mis apreciables maestros que sembró una semilla de pasión por la excelencia, siempre con su gran amabilidad, vocación, profesionalismo y sobre todo mucha alegría.*

*A **Dra. Eloísa del Carmen García Canseco, Dr. Ulises Jesús Tamayo Pérez**, por sus excelentes retroalimentaciones para mi desarrollo profesional.*

*A mi madre **María Eva Acosta Godínez**, por toda su vida entregada a mí y mis hermanos, convirtiéndose en una gran maestra de la vida, que con su sabiduría, logró empujarme al mágico mundo del aprendizaje.*

*A mi esposa **Dra. Lidia Yolanda Ramírez Rios**, por estar a mi lado, apoyándome, guiándome y educándome para lograr mayor objetividad en este estudio.*

*A mis compañeros de posgrado **Eduardo y Alfonso**, por las experiencias y amistad brindada a lo largo de estos años de estudio.*

*Al **CONACyT**, por apoyarme financieramente en mis estudios de maestría en ingeniería.*

*A nuestra alma mater: **Universidad Autónoma de Baja California**, que tanto me ha dado académicamente y profesionalmente.*

A todas las personas que de alguna u otra manera estuvieron involucradas conmigo a lo largo de mis estudios de maestría.

Tabla de Contenido

Página

Tabla de Contenido (Continuación)

Tabla de Contenido (Continuación)

Página

Lista de figuras

Lista de figuras (Continuación)

Lista de tablas

Capítulo I

Introducción

I.1 Antecedentes

El uso de la inteligencia artificial (IA) en la industria, se ha incrementado en los últimos años, debido a que ésta ha resultado ser una gran estrategia para la solución de problemas complejos relacionados con el *big data*, reconocimiento de imágenes, detección y clasificación, así como problemas de predicción, entre otros. Estos problemas surgen a medida que las empresas transicionan hacia la industria 4.0. La visión por computadora, también conocida como visión artificial, ha jugado un papel preponderante para posicionarse como pieza clave en la resolución de dichas problemáticas.

Antes de la IA y el aprendizaje profundo, el método clásico de visión artificial podía distinguir características básicas en imágenes adquiridas con cámaras digitales. Sus procedimientos eran relativamente sencillos dado a que, como sucede en un código QR, la forma de las imágenes debían ser simples, predecibles y con un patrón. A principios de los años sesenta surge el denominado *"Summer Vision Project"* (Papert, 1966), con el cual, los sistemas de visión podían detectar bordes, identificar cambios de color, tama´ no y distinguir los puntos de pixeles conectados que indicaban un objeto. Sin embargo, dichos sistemas clásicos estaban imposibilitados para leer bases de datos MNIST de dígitos escritos a mano (Deng, 2012), diferenciar acabados o tonalidades de piezas vistas desde diversos ángulos o condiciones ambientales de iluminación, entre otros aspectos, que la aplicación de la visión artificial puede ofrecer.

En los últimos veinte años, la visión artificial profunda ha desempeñado un papel

importante en el reconocimiento de imágenes y la detección de objetos (Gupta *et al.*, 2019). El objetivo primordial de un sistema de visión por computadora es detectar, segmentar, localizar y reconocer objetos concretos a partir de una o varias imágenes bidimensionales (Quintana *et al.*, 2012). Existen técnicas para el reconocimiento de objetos, las cuales se diversifican progresivamente en clasificación de imágenes, detección de objetos, segmentación semántica y segmentación de instancias. La primera técnica se refiere a la clasificación de toda la imagen en función de sus características intrínsecas. La segunda, tiene la particularidad de detectar objetos dentro de la imagen a partir de la similitud de los píxeles. La tercera, realiza la segmentación de cada objeto detectado en la imagen, agrupándolo según características similares; dicha segmentación se refiere al relleno de píxeles de cada objeto. Al igual que la anterior, la cuarta técnica consigue enumerar el número de objetos por clase (Sharma y Mir, 2020).

La aplicación de la visión artificial en la industria conlleva a la optimización de procesos, principalmente aquellos en los que tenga que ver la inspección. Un sistema de inspección con visión artificial permite verificar la conformidad de una parte respecto a los requerimientos o especificaciones, tales como dimensiones, números de serie, la presencia de componentes, errores de fabricación, entre otros. En términos generales, la visión artificial aplicada en la industria permite automatizar las tareas de inspección repetitivas que los operadores realizan para generar los controles de calidad en los procesos, los cuales no se podrían verificar por los métodos clásicos. Lo anterior, se puede expresar como la digitalización de la administración clásica de la calidad (Carvalho *et al.*, 2021).

La mayoría de los sistemas de visión artificial en la industria, pueden medir el nivel de luz, analizar colores, detectar bordes y formas. Este es el potencial de la visión artificial clásica, en la que estos sistemas deben ser programados lógicamente en función del usuario. Sin embargo, la tendencia de los últimos años es migrar hacia

la visión artificial profunda, que puede autoprogramarse en función de las entradas y salidas mostradas previamente por el usuario. La extracción de las características de una imagen empleando visión artificial profunda proporciona mejores resultados versus la visión artificial clásica.

Uno de los trabajos más recientes sobre visión artificial profunda en la industria tuvo como tema principal de investigación el reconocimiento de la acción. A partir de un conjunto de datos de vídeo y un modelo de entrenamiento YOLO, se consiguió clasificar en tiempo real las diferentes secuencias de ensamble (Zamora-Hernández *et al.*, 2021).En otro estudio reciente, clasificaron diferentes patrones de defectos en una superficie metálica utilizando redes neuronales convolucionales (Yun *et al.*, 2020). Mientras que en otros trabajos han conseguido detectar puntos de soldadura defectuosos en un proceso de la industria automotriz utilizando la versión cinco del modelo YOLO (por sus siglas en inglés *You Only Look Once*) (Dai *et al.*, 2021).

Algunas investigaciones han desarrollado modelos para la detección de objetos y el reconocimiento de imágenes. Las técnicas de detección de objetos se clasifican en detectores clásicos y de aprendizaje profundo. Otros han propuesto técnicas como YOLOv3 (Redmon y Farhadi, 2018), el cual es un algoritmo de detección de objetos que divide, en primer lugar, una imagen en una cuadrícula. En cada celda de la cuadrícula se estima la colocación de un cierto número de cuadros delimitadores alrededor de los objetos que obtienen una buena puntuación en las clases predefinidas. Cada cuadro delimitador tiene una puntuación de confianza que indica el grado de precisión de la predicción, y cada cuadro delimitador identifica sólo un objeto. Para determinar las formas y los tamaños más comunes, los recuadros delimitadores se crean agrupando las dimensiones de los recuadros de verdad del conjunto de datos original. En el caso de R-CNN (*Region-based Convolutional Neural Networks*), se realiza una búsqueda selectiva, extrayendo la región de origen, que se alimenta a una red neuronal

para producir características que alimentan un clasificador de máquina de vectores de soporte (SVM por sus siglas en inglés) para averiguar el objeto dentro de la imagen (Girshick *et al.*, 2014). La red SPP (*Spatial Pyramid Pooling*) es una arquitectura neuronal convolucional que utiliza la agrupación de pirámides espaciales para superar la limitación del tamaño fijo de la red. Encima de la capa convolucional final, se aplica una capa SPP. La capa SPP agrega las características y proporciona salidas de longitud fija que se envían a las capas totalmente conectadas (u otros clasificadores) (He *et al.*, 2015). R-CNN, está dividida en tres partes, la primera genera 2000 propuestas de regiones para pasarlas al siguiente módulo, que es una gran red neuronal convolucional que extrae un vector de características de longitud fija de cada región. Al tercer módulo le corresponde un conjunto de clasificadores lineales SVM (Girshick *et al.*, 2014). *Fast* R-CNN es muy similar al trabajo anterior, pero en este caso, la imagen se alimenta a una capa CNN para generar un mapa de características (Girshick, 2015). En el caso de las redes neuronales rápidas, se trata de una versión mejorada de la R-CNN rápida, que evita el uso de la búsqueda selectiva, dejando que la red neuronal aprenda las características de las imágenes (Ren *et al.*, 2015). *Feature Pyramid Networks* (FPN) (He *et al.*, 2015) y Mask R-CNN (He *et al.*, 2020), también han sentado las bases para la generación de imágenes de conjunto, como PASCAL VOC 2007 (Everingham *et al.*, 2007), PASCAL VOC 2012 (Everingham *et al.*, 2010), MSCOCO (Lin *et al.*, 2014), VIS Drone 2018 (Zhu *et al.*, 2021), KITTI (Geiger *et al.*, 2013) y LVIS (Gupta *et al.*, 2019).

Hay muchas arquitecturas publicadas durante la última década para problemas de reconocimiento de imágenes. Una de ellas es Xception, una variante de la arquitectura Inception que utiliza convoluciones separables en profundidad en lugar de los módulos Inception habituales. Se divide en tres bloques de un grupo de capas: flujo de entrada, flujo medio y flujo de salida (Chollet, 2017). Inceptionv3 cuenta con capas de convolución, reducción media, agrupación máxima, concatenaciones, abandonos y

capas totalmente enlazadas. La normalización por lotes se realiza en las entradas de activación y se utiliza con frecuencia en todo el modelo, mientras que softmax se utiliza para calcular la pérdida (Szegedy *et al.*, 2016). ResNet101v2 y Resnet152v2 (He *et al.*, 2016a) son redes residuales diseñadas para converger sin problemas de degradación de la precisión debido a la profundidad de la arquitectura de la red en sí. InceptionResNetv2 es una red neuronal convolucional basada en la familia de arquitecturas Inception, pero con conexiones residuales (Szegedy *et al.*, 2017).

En el caso del presente trabajo, se muestra la aplicación de la visión artificial profunda utilizando la técnica de aprendizaje de transferencia para detectar componentes de zinc en una línea de ensamble manual dentro de un proceso de manufactura de la empresa Schlage de México, ubicada en la ciudad de Ensenada, Baja California.

I.2 Motivación

Durante la última década, las redes neuronales para el aprendizaje profundo por computadora han sido perfeccionadas. Éstas han venido a solucionar problemas más complejos, los cuales no se habían podido resolver mediante la visión clásica. Lo anterior, es debido a que los actuales sensores de visión tienen el reto de mitigar la variación, tanto de la luz ambiental como la variación de coloración o acabado de los objetos a detectar o clasificar.

De la problemática anterior, se desprende la motivación para el desarrollo de ésta investigación. La forma en que se aborda el problema de estudio y su solución, se presenta en éste trabajo de tesis de maestría, mediante el uso de técnicas de aprendizaje profundo y transferencia de aprendizaje. Los modelos de redes neuronales convolucionales son un tipo de red neuronal que se utilizan para la tarea de clasificación de imágenes y el reconocimiento de objetos. Aprenden sobre las imágenes en el entrenamiento y pudiesen aprender sobre las características principales del objeto en

cuestión, sin importar la luz ambiental o variación entre pieza y otra.

I.3 Planteamiento del problema

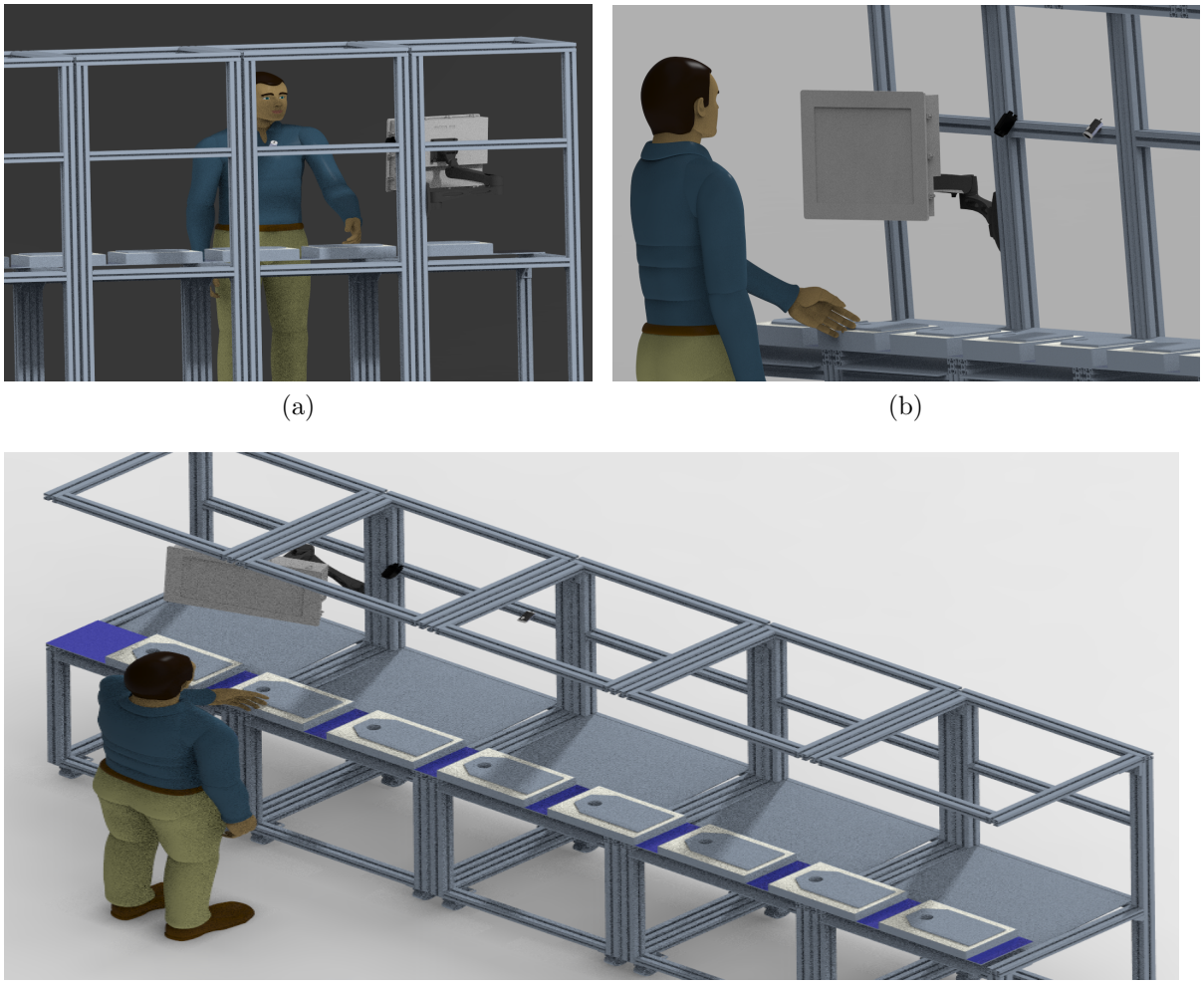
La empresa Schlage de México, planta Ensenada, en los últimos meses, ha reportado problemas con un sistema clásico de visión debido a fallos en la detección de componentes de zinc en una de sus líneas de ensamble manual.Dicho sistema consiste en un sensor de visión clásico, al cual se le programan patrones de detección por cada clase diferente de objeto que se quiere detectar. Los patrones que se pueden configurar para la detección son filtros de detección de borde, color RGB y forma. Para cada clase existe un programa específico con dichos filtros programados. Una vez que se coloca en modo correr, el sistema espera un disparo de inicio para tomar la fotografía y realizar el procesamiento, si se detecta que el objeto cumple con los filtros preprogramados, entonces, retorna un verdadero como resultado, de lo contrario, regresa un falso como resultado.

Este tipo de sistema ha tenido numerosos problemas, debido a la variación de luz ambiental y variación del acabado de los objetos a detectar. Por lo anterior, el presente trabajo de investigación aborda dicha problemática para desarrollar una propuesta de solución mediante la aplicación de la estrategia de aprendizaje automático profundo.

I.3.1 Propuesta de solución

Para dar solución al problema planteado, se propone el entrenamiento de cinco redes neuronales convolucionales, teniendo como objeto de entrenamiento un conjunto de imágenes que se generan en la línea de manufactura. Las imágenes pertenecen a 6 diferentes clases, las cuales representan a los 5 modelos disintos de componentes de zinc y 1 fixtura de ensamble que no contiene ningún componente. Ésta última clase se incorpora con la finalidad de modelar todas las posibles combinaciones de ensamble:

componente 1 (clase 1) sobre fixtura de ensamble (clase 6), componente 2 (clase 2) sobre fixtura de ensamble (clase 6), componente 3 (clase 3) sobre fixtura de ensamble (clase 6), componente 4 (clase 4) sobre fixtura de ensamble (clase 6), componente 5 (clase 5) sobre fixtura de ensamble (clase 6) y fixtura de ensamble (clase 6). La Figura 1 muestra un escenario de aplicación en una línea de manufactura empleando una cámara web y computadora personal.



(a) (b)

(c)

Figura 1: Manufacturing line with webcam. (**a**) View A, (**b**) View B and (**c**) View C.

I.4 Objetivos del trabajo de tesis

I.4.1 Objetivo General

La presente investigación tiene por objetivo general diseñar, desarrollar, evaluar e implementar un sistema inteligente de detección de componentes de zinc para ensamble de producto en un proceso de manufactura, mediante el uso de técnicas del aprendizaje profundo.

I.4.2 Objetivos específicos

Los objetivos específicos planteados para este trabajo de investigación se enlistan a continuación:

- Crear una nueva propuesta de conjunto de datos para su preprocesamiento a partir de la técnica de aumentación de datos de imágenes.
- Entrenar y evaluar diferentes algoritmos de clasificación de imágenes para la detección de componentes de zinc bajo diversas condiciones de acabado y factores ambientales.
- Generar modelos entrenados de redes neuronales para la clasificación de diferentes clases de componentes de zinc con alto grado de confiabilidad.
- Documentar diagrama de flujo para implementar solución mediante aprendizaje automático.
- Evaluar el funcionamiento y desempeño de los modelos.
- Usar modo de inferencia para detección en tiempo real.

I.5 Organización del manuscrito

El presente trabajo de tesis está compuesto por cinco capítulos. En el **capítulo II** se presenta el marco teórico referente a la manufactura, ensamble manual, proceso de galvanizado, metodología six sigma como herramienta fundamental para aproximaciones a soluciones, aprendizaje automático y generalidades de redes neuronales artificiales. En el **capítulo III** se describen los pasos que se siguieron en este trabajo para poder diseñar, desarrollar, evaluar y validar los diferentes algoritmos de aprendizaje automático. En el **capítulo IV** se describen los resultados obtenidos así como las discusiones. Finalmente, en el **capítulo V** se presentan las conclusiones generales y trabajos futuros.

Capítulo II

Marco Teórico

II.1 Cincado (galvanizado)

El cincado o galvanización, consiste en la deposición de una fina capa de aluminio sobre un componente metálico, cuyo objetivo es proporcionar protección. La superficie exterior del revestimiento pasa por un proceso de oxidación para formar óxido de zinc, lo que da lugar a un acabado mate de color plateado. El cincado se aplica a menudo a piezas de hierro o acero cuya superficie se oxidaría al exponerse al aire o al agua (Kristoff, 2017).

II.1.1 Preparación del cincado

La pieza a cubrir se limpia a fondo antes del revestimiento para eliminar las partículas, la grasa y los óxidos que pudieran haberse acumulado en su superficie. Este proceso suele consistir de un baño en una solución alcalina para eliminar las partículas de la superficie, seguido de un baño en una solución ácida débil para grabar la superficie y eliminar los óxidos. Si las partículas u óxidos permanecen en la superficie de la pieza, podrían crear huecos en la capa de cincado, lo que daría lugar a manchas en la pieza que quedarían sin protección.

II.1.2 Métodos de cincado

Un método para aplicar una capa de zinc a una pieza metálica es la galvanización en caliente. La pieza se sumerge en una tina de zinc fundido que tiene una capa de fundente flotando sobre el zinc. El fundente suele ser una solución de cloruro de amonio y zinc.

Esto permite que la superficie de la pieza se recubra de fundente antes de entrar en el zinc fundido. posteriormente, se retira la pieza del baño y se deja secar la capa de zinc. También se puede utilizar el proceso de galvanización en seco para revestir una pieza con zinc. En este caso, la pieza se recubre sólo con fundente y se deja secar antes de sumergirla en una cuba de zinc fundido. En cualquiera de los dos métodos, la capa de zinc forma una apariencia cristalina, llamada lentejuela. El aspecto de la lentejuela puede controlarse en función de la velocidad de enfriamiento.

II.1.3 Transformación de la capa de zinc

El zinc forma una unión con la pieza de acero, lo que da lugar a una capa de transición de aleación de zinc y acero entre los metales. La capa de zinc no puede despegarse como una capa de pintura porque está integrada atómicamente con el acero. Tras uno o dos días de exposición a la atmósfera, la superficie exterior de la capa de zinc se convierte en óxido de zinc. Esta transformación aumenta la protección que proporciona la capa de zinc. Tras un largo periodo de exposición al medio ambiente, el óxido de zinc se convierte en carbonato de zinc, que también actúa como capa protectora. El anodizado crea una capa dura de óxidos metálicos en la superficie del aluminio, que lo protege de la corrosión o la abrasión. En ocasiones, los revestimientos anodizados se dañan. Al reparar el aluminio anodizado, primero hay que determinar si el metal base está dañado o si sólo lo está el revestimiento de óxido metálico. Si el revestimiento de metal base está dañado, tendrá que desanodizar la superficie y luego reparar el metal base lijando o puliendo. Si sólo está dañado el revestimiento de óxido metálico, puede repararlo mediante el anodizado con cepillo, un proceso de anodizado portátil.

II.2 Ensamble manual

El ensamble manual es un tipo de manufactura, en el cual dos o mas partes son unidas para formar una nueva entidad (Groover, 2013). Por otro lado, Swift y Booker (2013), definen el ensamble manual como la constitución de componentes y/o subensambles previamente fabricados en un producto completo o en una unidad de un producto, realizado principalmente por operadores humanos que utilizan su destreza, habilidad y juicio inherente. El operador puede estar en una estación de trabajo (banco) o formar parte de un sistema de transferencia que mueve el producto a medida que se ensambla. El ensamblaje manual puede estar asistido por sistemas mecanizados o automatizados para las operaciones de alimentación, manipulación, ajuste y control, tal como se ilustra en la Figura 2.

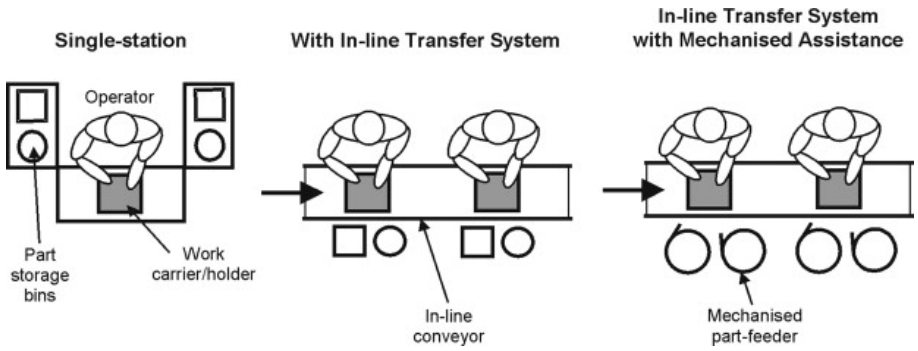


Figura 2: Configuraciones comunes del sistema de ensamble manual. Fuente: (Swift y Booker, 2013).

II.2.1 Elementos de un proceso manual

A continuación, se enlistan y describren los distintos elementos que conforman un proceso manual según Swift y Booker (2013):

- **Alimentación:** presentación de un componente al equipo de manipulación en la orientación correcta mediante una variedad de métodos. Las piezas se extraen manualmente de los depósitos de almacenamiento y luego son orientadas por

el operario. La orientación puede lograrse mediante alimentadores de cubeta vibratorios/centrífugos, piezas ya orientadas en forma de paletas/magazines/tiras o el uso de mecanismos de escape de alimentación de piezas.

- **Manipulación:** reunir los componentes y/o subconjuntos de manera que pueda producirse la composición posterior. Utilización de las manos, medios de elevación simples y plantillas, así como dispositivos de fijación.

- **Montaje:** pueden utilizarse diversas configuraciones de colocación/ubicación de piezas o métodos de fijación/unión, por ejemplo, "clavija en agujero", ajuste a presión, soldadura, remachado, unión adhesiva, estacado o atornillado utilizando diversas herramientas manuales o mecanizadas/eléctricas.

- **Comprobación:** detección de componentes ausentes, incorrectos, deformados o mal orientados mediante la capacidad de detección y comprobación de alto nivel de los operarios y las ayudas mecánicas/eléctricas. También detección de cuerpos extraños, fallos en las piezas y mal funcionamiento de las máquinas.

- **Transferencia:** distintas operaciones de montaje necesarias que se realizan en estaciones separadas, normalmente acumuladas en un portapiezas, palé o soporte. Se requiere un sistema para transferir los conjuntos parcialmente completados de una estación de trabajo a otra. En general, los distintos tipos de sistema de estación de trabajo/transferencia para el montaje manual son:

- **Estación única:** montaje en una estación de trabajo o banco donde se realiza una operación específica o varias operaciones.

- **Continuo:** el carro de trabajo fluye sin detenerse utilizando transportadores (en línea, giratorios o carrusel), carril aéreo o sirga.

- **Intermitente:** sincrónico/indexado (se desplaza con un tiempo de ciclo fijo) o no sincrónico/transferencia libre (se desplaza según las necesidades o cuando el operario finaliza la operación/montaje) utilizando sistemas en línea o giratorios.

II.3 Metodología Six Sigma

Definir a la metodología Six Sigma, siempre resulta un tanto complejo, dada las diversas descripciones que existen acerca de ella. Podemos conceptualizarla como una forma de medir el nivel de variabilidad de cualquier proceso, el rendimiento de una compañía, una estrategia para la solución de problemas basados en datos, hasta decir que es toda una filosofía que cambia la vida (Brook, 2010). Para Pyzdek y Keller (2010), Six Sigma es la aplicación rigurosa de un conjunto de principios y técnicas de calidad cuya eficacia ha sido probada, dado a que incorpora elementos del trabajo de varios estadistas considerados los pioneros de la administración de la calidad.

El despliegue de la metodología Six Sigma permite mejorar y asegurar los procesos de calidad. Lo anterior, conlleva a alzancar, mantener y maximizar las ganancias, incrementar el rendimiento de la organización y centrarse en las necesidades de los consumidores. Es por éstas razones que las industrias como la automotriz, eléctrica y electrónica han obtenido grandes beneficios a través de su implementación, logrando mejorar sus servicios y grandes ahorros en costos (Ľubica Simanová *et al.*, 2019).

El origen de ésta metodología data del año 1980, surgiendo como una estrategia de la compañía Motorola para ahorrar costos mediante la mejora de sus procesos de manufactura, centrándose en la eliminación de defectos con la misma fuerza de trabajo, tecnología y diseños.

Recopilando los trabajos de diversos autores, se puede concluir que Six Sigma se refiere a la combinación de métodos estadísticos, la correcta comprensión de los requerimientos del cliente y la reducción de la variabilidad en los procesos con la

finalidad de mejorarlos y perfeccionarlos, para acercarse a 3.4 errores por millón de oportunidad (Ľubica Simanová *et al.*, 2019).

II.3.1 DMAIC

Dentro del conjunto de herramientas y técnicas que son empleadas en Six Sigma, el modelo DMAIC resultan ser uno de los más importantes para la mejora de productos, procesos o servicios. La metodología DMAIC es un acrónimo en inglés que define las 5 fases que integran el modelo: Definir, Medir, Analizar, Mejorar y Controlar *(Define, Measure, Analyze, Improve and Control).* En la primera fase, se forma un equipo de trabajo y se definen los objetivos que conducen a la mejora deseada, los cuales, por lo general, se plasman en un "*project charter*", el cual es un documento que contiene toda la información clave de un proyecto con la finalidad de darle formalidad y un eje de acción. En la segunda fase, se establecen métricas válidas y confiables que ayuden a monitorear el curso que sigue el proceso actual y medir su progreso respecto a los objetivos definidos en la fase previa. La tercera fase realiza un análisis explotarorio y descriptivo de datos para su estudio mediante herramientas estadísticas. Posteriormente se identifican las formas en que se puede eliminar la brecha entre el funcionamiento y rendimiento actual y el objetivo deseado. La cuarta fase tiene por objetivo mejorar al sistema mediante la creatividad y la búsqueda de nuevas maneras de realizar las actividades, de forma rápida y al menor costo. Para ello, se emplean herramientas de planificación y administración de proyectos para la implementación del nuevo enfoque y métodos estadísticos para validar la mejora. Por último, la quinta fase busca controlar el nuevo sistema, mediante la utilización de herramientas estadísticas para monitorear la estabilidad del mismo. Para lo anterior, es común apoyarse de estándares y normativas para la generación de documentación correcta, así como de políticas, procedimientos, instrucciones de operaciones, entre otros aspectos (Pyzdek y Keller, 2010).

II.3.2 Diseño de experimentos como base fundamental del aprendizaje automático

El diseño de experimentos (DOE, por sus siglas en inglés), es una metodología que representa un rango amplio de técnicas experimentales, en la cual, el proceso es experimentado de una manera controlada, asi como los resultados observados y posteriormente analizados (Brook, 2010). Dicha técnica intenta modelar un fenómeno físico o proceso según las entradas y salidas observadas, tal como se realiza en la metodología del aprendizaje automático, dado a que una de las fases iniciales reside en la identificación de las entradas de un fenómeno o proceso, para posteriormente, etiquetar cada muestra con una salida esperada.

En un DOE, el objetivo es identificar las entradas importantes del proceso y entender sus afectaciones sobre la salida. La matemática detrás de un DOE es similar a una regresión, sin embargo la diferencia entre ambas reside en que:

- Las técnicas de regresión son generalmente utilizadas para analizar datos históricos, tomados del proceso en "modo normal".

- El diseño de experimentos es creado para analizar los datos en tiempo real, que son tomados del proceso en "modo experimental". Esto puede tomar horas o días.

El diseño de experimentos va de la mano con proyectos six sigma, que están orientados a la mejora de procesos donde se pueden controlar sus entradas en tiempo real (Brook, 2010). Para lo anterior, se pone el ejemplo de una planta que esté buscando controlar el grosor de una lámina de acero que emerge de un proceso de rolado. Para este proceso existen muchos factores de entrada, tal como grosor de entrada, presión de rolado, velocidad de rolado, lubricante, temperatura del acero, entre otros, los cuales pueden afectar el grosor a la salida de la lámina de acero. Un experimento está planeado para encontrar cuál de los factores de entrada es el más importante y cuantificar su

efecto. Éste y muchos otros ejemplos cotidianos de un DOE en la manufactura, se iniciarán con la identificacion de la variable de salida del proceso, luego, se identificarán las entradas del proceso que pudiesen afectar la salida, para posteriormente diseñar el experimento, correrlo y finalmente analizarlo (Brook, 2010). Cabe señalar que los factores de entrada, representan las entradas X's (x1, x2, x3, ...xn) mientras que la variable de salida, en este caso, el grosor de la lámina , representaría la Y (Brook, 2010).

II.4 Aprendizaje automático y sus características

El aprendizaje automático está bioinspirado en el aprendizaje humano, el cual requiere de un periodo de tiempo para dicho aprendizaje, tal como sucede con muchos de los procesos del desarrollo psicomotriz del ser humano: aprender a sentarse, caminar, correr, hablar y escribir. Como mecanismo incorporado a la psique humana, no se necesitan de muchos ejemplos para aprender sobre algo. Por ejemplo, con sólo mirar dos o tres casas en la carretera, se puede aprender fácilmente a reconocer una casa. También a diferenciar fácilmente entre un coche y una moto con sólo ver unos cuantos de éstos. Así como diferenciar fácilmente un gato de un perro. Aunque parece muy fácil e intuitivo para los seres humanos, para las máquinas puede ser una tarea de extrema dificultad (Singh, 2019).

El aprendizaje automático es el mecanismo por el que se intenta que las máquinas aprendan sin ser programadas. Ejemplo a ello, pudiése ser una máquina a la cual se le muestran imágenes de gatos, y aprende lo suficiente para que la misma, distinga la diferencia entre un gato y otro animal. El reto al que se enfrentan las máquinas es aprender todo el patrón, abstracción de características o rasgo de abstracción sólo a partir de pocas imágenes (Singh, 2019). Para resolver dicho reto, necesitarían suficientes ejemplos para aprender tantas características como sean posibles, debido a que el

modelo generado deberá tender a la generalización.

II.5 Aprendizaje supervisado

Esta es la categoría principal del aprendizaje automático que impulsa muchas aplicaciones y valor para las empresas. En el aprendizaje supervisado, se entrenan los modelos en los datos etiquetados. Por etiquetados, se entiende que tienen las respuestas o resultados correctos para los datos (Singh, 2019). Por ejemplo, para ilustrar el aprendizaje supervisado, se puede considerar a una planta manufacturera que quiere saber si los nuevos operarios contratados I.

Tabla I: Conjunto de datos.

Edad	**Género**	**Salario**	**¿Préstamo pagado?**
25	Hombre	$35K	No
36	Mujer	$78K	Si
34	Hombre	$120K	Si
58	Mujer	$65K	No

En el aprendizaje supervisado, el modelo aprende de los datos de entrenamiento que también tienen una columna de etiqueta/resultado/objetivo y la utiliza para hacer predicciones sobre los datos no vistos. En el ejemplo anterior, las columnas como edad, sexo y salario, se conocen como atributos o características, mientras que la última columna (préstamo no pagado) se conoce como el objetivo o la etiqueta que el modelo intenta predecir para los datos no vistos. Un registro completo con todos estos valores se se conoce como una observación. El modelo necesitaría una cantidad suficiente de observaciones para ser entrenado y luego hacer predicciones sobre un tipo similar de datos. Debe haber al menos una característica/atributo de entrada para que el modelo se entrene junto con la columna de salida en el aprendizaje supervisado. La razón por la que la máquina es capaz de aprender de los datos de entrenamiento, es

debido a la suposición subyacente de que algunas de estas características de entrada, individualmente o en combinación, tienen un impacto en la columna de salida (préstamo no pagado) (Singh, 2019).

II.6 Aprendizaje no supervisado

El aprendizaje no supervisado, implica que el conjunto de datos que se esta manejando, no está etiquetado como en el caso del aprendizahe supervisado, es decir, se busca algun patrón escondido dentro del mismo conjunto de datos, el cual pudiese tender a agrupar ciertos datos, tal como se realiza con el *clustering*. (Neapolitan y Jiang, 2019). Por ejemplo, algún conjunto de imágenes que no están etiquetadas, se pasan por un proceso de entrenamiento que luego daría como resultado un conjunto de subgrupos (*clustering*). Se pudiese pensar en un conjunto de fotografiás de animales o distintas especies donde el resultado es la agrupación de estos mismos.

II.7 Aprendizaje semi-supervisado

Como su nombre lo indica, el aprendizaje semi-supervisado se encuentra entre el aprendizaje supervisado y el no supervisado. De hecho, utiliza ambas técnicas. Este tipo de aprendizaje es principalmente relevante cuando se trata de un conjunto de datos mixto, es decir, que contiene tanto datos etiquetados como no etiquetados. A veces son sólo datos sin etiquetar por completo o sólo algunos etiquetados manualmente. El aprendizaje semi-supervisado puede utilizarse en una pequeña porción de datos etiquetados para entrenar el modelo y luego utilizarlo para etiquetar el resto de los datos o parte de ellos, que luego se puede utilizar para otros fines. Esto también se conoce como pseudo-etiquetado, ya que etiqueta los datos no etiquetados. Por citar un ejemplo sencillo, se tiene el caso de una gran cantidad de imágenes de diferentes marcas de redes sociales, las cuales, en su mayoría no están etiquetadas. Utilizando el aprendizaje

semi-supervisado, se puede etiquetar algunas de estas imágenes manualmente y luego entrenar el modelo en las imágenes etiquetadas. A continuación, se utiliza las predicciones del modelo para etiquetar las imágenes restantes y transformar los datos no etiquetados en datos completamente etiquetados. El siguiente paso en el aprendizaje semi-supervisado es volver a entrenar el modelo en el conjunto de datos etiquetados. La ventaja que ofrece es que el modelo se entrena en un conjunto de datos más grande y, contrario a lo que era antes, ahora es más robusto y con mejores predicciones. Otra ventaja es que el aprendizaje semi-supervisado ahorra mucho esfuerzo y tiempo. La contrapartida de realizar esto es que es difícil conseguir un alto rendimiento del pseudo-etiquetado, ya que utiliza una pequeña parte de los datos etiquetados para hacer las predicciones. Sin embargo, sigue siendo una mejor opción que etiquetar manualmente los datos, dado a lo costoso que puede resultar y el timepo invertido para ello (Singh, 2019).

II.8 Aprendizaje reforzado

Es el cuarto y último tipo de aprendizaje, el cual es una gran área de investigación. Difiere al resto en cuanto al uso de los datos y sus predicciones. En cuanto a los datos, se necesitan, principalmente, datos históricos para entrenar los modelos. El aprendizaje por refuerzo funciona con un sistema de recompensas, es decir, se trata principalmente de una toma de decisiones basada en determinadas acciones que el agente realiza para cambiar su estado, intentando maximizar las recompensas (Singh, 2019).

II.9 Aprendizaje por transferencia

A continuación, se describe la técnica de aprendizaje por transferencia, mencionada por Singh (2019), misma que fue utilizada en el presente trabajo.

El aprendizaje por transferencia intenta “transferir” el conocimiento aprendido

en las tareas de origen y lo aplica para mejorar el aprendizaje en las tareas de destino, las cuales suelen estar relacionadas con las de origen. El aprendizaje puede mejorarse mediante el aprendizaje por transferencia desde dos aspectos. Los modelos de aprendizaje automático entrenados con el conocimiento transferido suelen tener un mejor rendimiento y la cantidad de tiempo de entrenamiento se reduce considerablemente (Singh, 2019).

Este método es un enfoque útil para tratar el problema de la insuficiencia de datos de entrenamiento. Intenta transferir el conocimiento del dominio de origen al dominio de destino dejando de lado los requisitos de datos de entrenamiento y de prueba idénticos. Esto tendrá un impacto positivo significativo en una serie de dominios que son difíciles de desarrollar debido a la falta de datos de entrenamiento (Tan *et al.*, 2018).

Para transferir modelos de aprendizaje profundo se aplican dos estrategias comunes, el ajuste fino y la extracción de características a los modelos de aprendizaje profundo preentrenados en conjuntos de datos de origen. El ajuste fino, se suele utilizar para tareas de clasificación en las que las redes base se modifican ligeramente para adaptarlas a las tareas de clasificación. Posteriormente, las redes modificadas se entrenan con los conjuntos de datos de destino para afinar todos los parámetros de las capas aprendibles de las redes. Tras el ajuste, los conocimientos aprendidos en las redes base se transfieren a las nuevas redes, que también aprenden nuevos conocimientos de los conjuntos de datos objetivo. En consecuencia, las redes ajustadas requieren el mismo tiempo de entrenamiento que las redes con las mismas arquitecturas pero entrenadas desde cero. Sin embargo, el rendimiento de las redes transferidas es mucho mejor que el de las redes entrenadas desde cero en términos de precisión y representación de características. Para la extracción de características, los parámetros de todas las capas, excepto las superiores, permanecen inalterados. Sólo las capas superiores se reentrenan para actualizar los parámetros. En este caso, las capas superiores pueden ser las capas

superiores originales de las redes base o capas recién añadidas en relación con las tareas de clasificación. Cuando se utiliza la técnica de extracción de características para el aprendizaje por transferencia, las redes base tienen que ser modelos de aprendizaje profundo universales y potentes, que puedan extraer características representativas para que las capas superiores puedan ser bien entrenadas en las tareas de clasificación de una manera relativamente económica. La diferencia entre el ajuste fino y el extractor de características, es en esencia, el número de parámetros de entrenamiento durante las sesiones de entrenamiento en dos estrategias. El ajuste fino se utiliza, más a menudo, cuando el tamaño del conjunto de datos objetivo es grande y puede garantizar el buen rendimiento de los modelos transferidos. Cabe señalar que los modelos transferidos podrían estar sobreajustados si el número de parámetros a entrenar supera el número de muestras del conjunto de datos de destino. En este caso, la extracción de características puede ser una alternativa para reutilizar los modelos preentrenados y producir eficientemente un modelo. Sin embargo, es demasiado impreciso calibrar el tamaño de un conjunto de datos, ya que es difícil definir lo que es grande, mediano y pequeño. Por lo tanto, se sugiere desplegar dos estrategias en los conjuntos de datos objetivo antes de encontrar el mejor modelo adquirido mediante las mejores estrategias (Singh, 2019).

II.10 Redes neuronales artificiales

El estudio de las redes neuronales artificial en diveras áeras de la ciencia tiene años de investigación. Su comienzo data desde 1958, cuando la primera arquitectura de red neuronal artificial (ANN, por sus siglas en inglés) fue propuesta por el psicólogo Frank Rosenblatt, un psicólogo. El nombre de esta red neuronal artificial fue perceptrón (Rosenblatt, 1958). Es aquí donde comienza una retórica de tres cuestionamientos, teniendo en cuenta a organismos superiores (si los hay):

- ¿Cómo es la información del ambiente físico detectado sensorialmente por un sistema biológico?
- ¿Cómo se guarda ésta información?
- ¿Cómo influye dicha información guardada en el comportamiento y reconocimiento?

Lo anterior, vendría a desencadenar la propuesta de dicha arquitectura, como se muestra en la Figura 3.

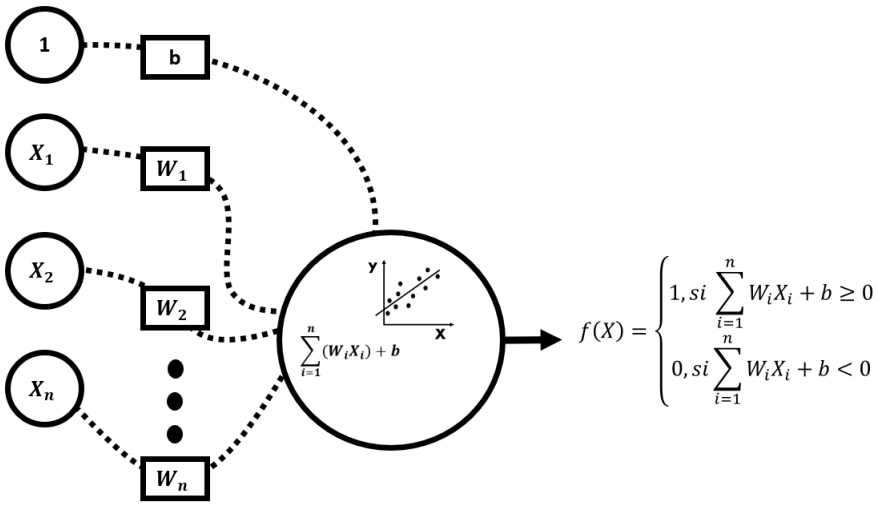


Figura 3: Representación de un perceptron. Fuente: Modificado de Vega (2018)

Sin embargo, tres años antes, dos científicos médicos descubrieron la forma en que la retina detecta el movimiento, el contraste y la orientación (Mountcastle, 1957). David Hubel y Torsten Wieselbut fueron reconocidos 26 años más tarde, ya que en 1981 fueron galardonados con el Premio Nobel de Medicina por sus trabajos sobre el sistema visual, algo importante para esa época. Un año antes, Kunihiko Fukushima habia publicado lo que hasta hoy ha sido la mejor red neuronal para reconocimiento de imágenes, la primera red neuronal convolucional primitiva, llamada Neocognitron (Fukushima, 1988).

II.11 Red neuronal convolucional

La red neuronal convolucional es un perceptrón multicapa considerada como red neuronal profunda, utilizada para identificar información de im

'agenes bidimensionales. Esta red consiste en varias capas de neuronas, dónde la primer capa oculta consiste en extraer un mapa de características de la imagen de entrada, a través de un filtro de tamaño ***n x n*** que pasa sobre la imágen según el *stride*, es decir, el paso que dará el filtro cada vez que realice la operación matemática del producto entre matrices (ver figura 4).

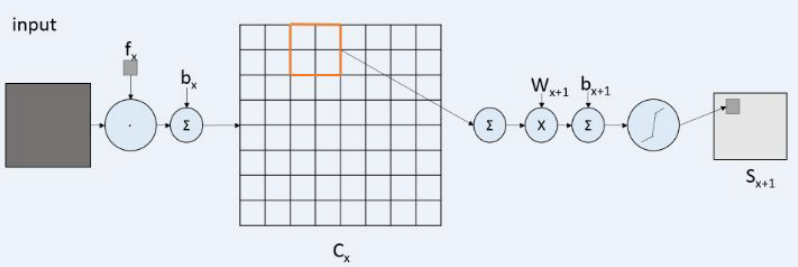


Figura 4: Representación gráfica de una Red neuronal convolucoonal. Fuente: Liu *et al*, *"Implementation of Training Convolutional Neural Networks".*

Una vez que se genere la primer capa de características, se inicia con un proceso de muestreo, que consiste en extraer los pixeles promedio representativos para generar de nuevo un nuevo mapa de características. Este proceso continúa dependiendo el número de capas ocultas contenidas en la red neuronal convolucional. Al final, previo a la cabeza de la red neuronal, se contiene una capa plana para posteriormente, unirse con la capa de salida (Liu *et al.*, 2015), (Fukushima, 1988).

Esto se repite cada época de entrenamiento y propagando hacia atrás el error en la diferencia de la salida desaeada y la calculada con la finalidad de que la red neuronal convolucional pueda aprender (Rumelhart *et al.*, 1986).

II.12 Visión profunda por computadora

La visión profunda por computadora ha ampliado los límites de lo que era posible en el ámbito del procesamiento digital de imágenes. Sin embargo, eso no quiere decir que las técnicas tradicionales de visión por computadora que habían experimentado un desarrollo progresivo en años anteriores al auge del DL se hayan quedado obsoletas. Dicha metodología consiste en abordar el problema de reconocimiento de imágenes u objetos, utilizando las técnicas de aprendizaje profundo, que en su mayoría, utilizan las redes convolucionales con más capas ocultas.

Las tareas objetivo del aprendizaje profundo por computadora se clasifican en cuatro categorías:

- **Clasificación de imágenes**: Se refiere a la clasificación de toda la imagen a partir de sus características intrínsecas.
- **Detección de objetos**: Tiene la particularidad de detectar objetos dentro de la imagen a partir de la similitud de los píxeles, teniendo como salida, coordenadas de un rectangulo y objeto detectado.
- **Segmentación semántica**: Realiza la segmentación de cada objeto detectado en la imagen, agrupándolo según características similares; dicha segmentación se refiere al relleno de píxeles de cada objeto.
- **Segmentación por instancias**: Similar a la segmentación semántica, logra identificar y enumerar la cantidad de objetos por clase.

En la siguiente figura 5 se muestra de manera mas descriptiva, las diferentes tareas objetivo de la visión artificial.

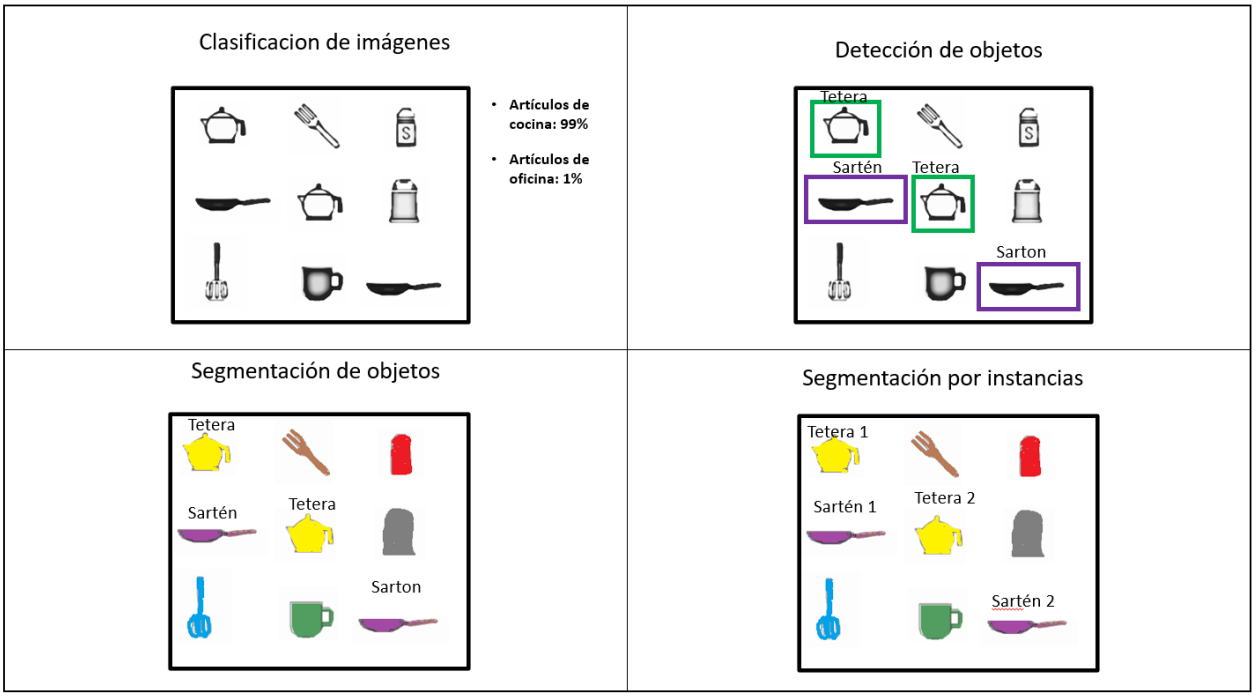


Figura 5: Representación gráfica de las tareas objetivo de la visión artificial. Fuente: Iconos tomados de www.dreamstone.com

En la última década se han publicado varias arquitecturas para problemas de reconocimiento de imágenes. A continuación se describirán cada una de las arquitecturas utilizadas en este trabajo:

Una de ellas es Xception, una variante de la arquitectura Inception que utiliza convoluciones separables en profundidad en lugar de los módulos Inception habituales. Se divide en tres bloques de un grupo de capas: flujo de entrada, flujo medio y flujo de salida (Chollet, 2017). Por su parte, InceptionV3 tiene capas de convolución, reducción media, agrupación máxima, concatenaciones, abandonos y capas totalmente enlazadas. La normalización por lotes se realiza a las entradas de activación y se utiliza con frecuencia en todo el modelo, mientras que softmax se utiliza para calcular la pérdida (Szegedy *et al.*, 2016). Mientras que, ResNet101V2 y Resnet152V2 (He *et al.*, 2016b), son redes residuales diseñadas para converger sin problemas en cuanto a la

degradación de la precisión debido a la profundidad de la propia arquitectura de la red. Por último, InceptionResNetV2 es una red neuronal convolucional basada en la familia de arquitecturas Inception, pero con conexiones residuales (Szegedy *et al.*, 2017).

II.12.1 InceptionV3

Inception V3 es un modelo de reconocimiento de imágenes que alcanzó una exactitud de 78.1% con el conjunto de datos ImageNet. El modelo está compuesto por componentes básicos simétricos y asimétricos, que incluyen oepraciones convolucionales, reducción promedio, agrupación máxima, concatenaciones, dropout y capas completamente conectadas. La normalización por lotes se usa ampliamente en todo el modelo y se aplica a las entradas de activación. La pérdida se calcula mediante Softmax.

En la figura 6 se muestra un diagrama de alto nivel del modelo:

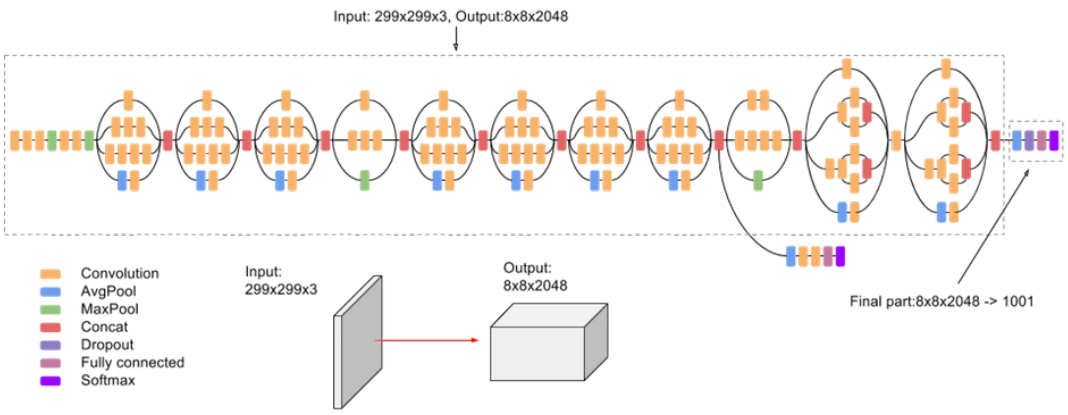


Figura 6: Arquitectura InceptionV3.

II.12.2 Xception

Como se puede apreciar en la figura 7, esta arquitectura contiene 36 capas convolucionales que forman la base para la extracción de características de esta red. En su fase experimental, se toma como objetivo la clasificación de imágenes, por lo que la base de convolución es seguida por una capa de regresión logística. Las 36 capas convolucionales

se componen de 14 módulos, todos los módulos tienen residuos lineales (la conexión de cada bloque pequeño utiliza una conexión residual que es el signo + en la figura) y conexión alrededor de ellos, excepto el primer y último modulo (Chollet, 2017).

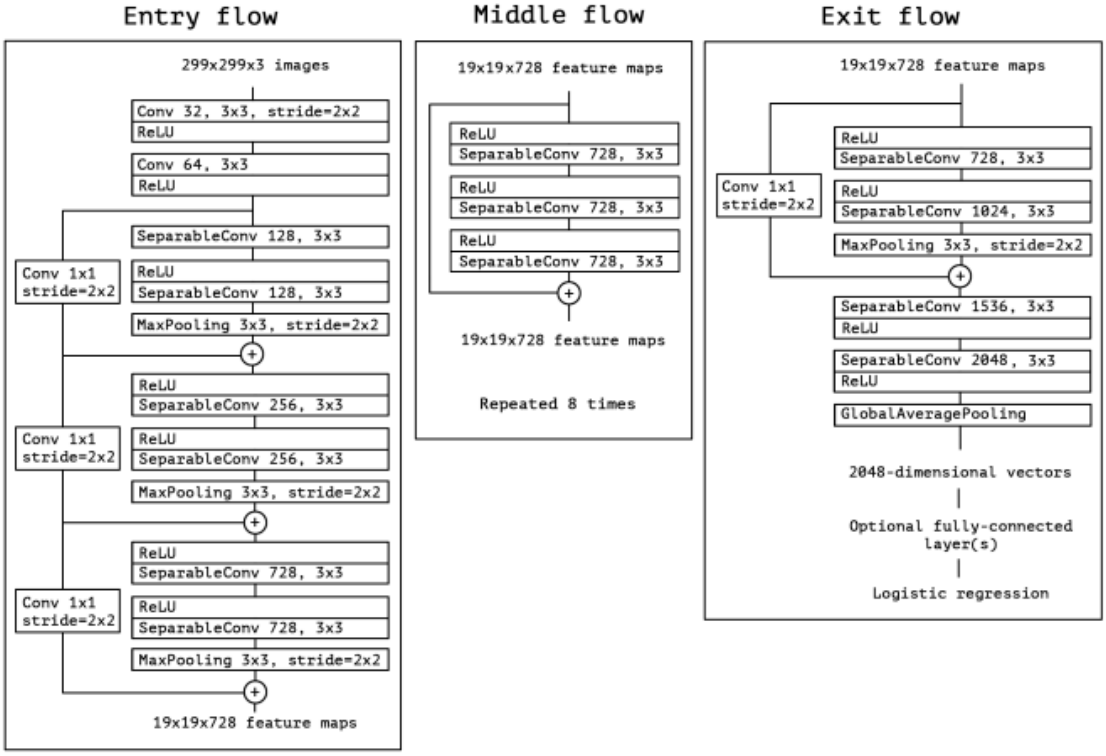


Figura 7: Arquitectura Xception.

II.12.3 ResNet101V2 y ResNet152V2

ResNet, se refiere a Red Residual y es un tipo de red neuronal que fue introducida en 2015 (He *et al.*, 2016a) en un artículo titulado *"Deep Residual Learning for Image Recognition"*. Los modelos ResNet han tenido un gran éxito en competiciones:

- Primer puesto en la competición de clasificación ILSVRC 2015, con una tasa de error top-5 del 3,57%.
- Primer puesto en la competición ILSVRC y COCO 2015, en detección de ImageNet, localización de ImageNet, detección de Coco y segmentación de Coco.

- Sustituyendo las capas VGG-16 en Faster R-CNN por ResNet-101, se observan mejoras de aproximadamente 28%, entrenando eficientemente redes con 100 capas y también con 1000 capas.

Específicamente, ambos modelos tienen 101 y 152 capas en su arquitectura, ver tabla 9

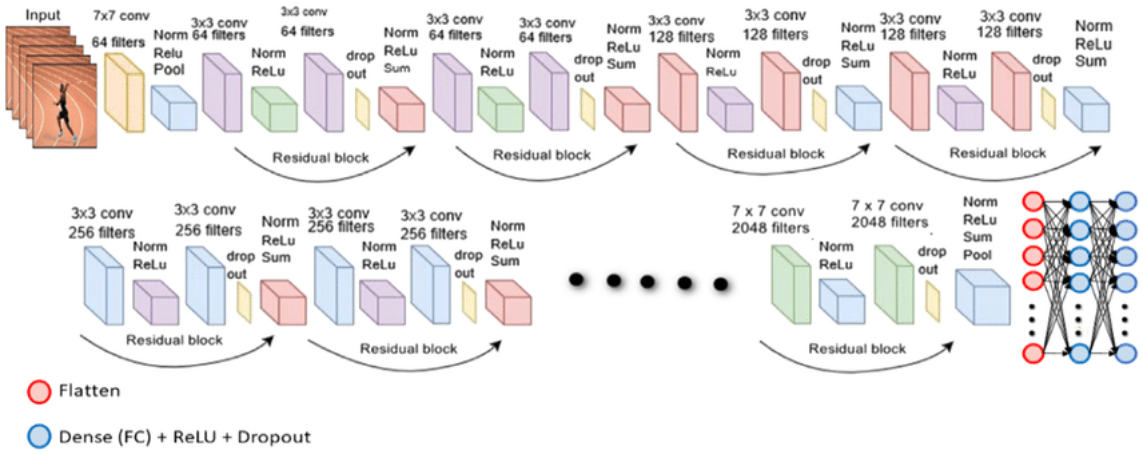


Figura 8: Arquitectura ResNe101V2 y ResNet152V2 .

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 9: Arquitectura de redes tipo residual.

II.12.4 InceptionResNetV2

La arquitectura Inception ha tenido una proporción con buen rendimiento con requerimiento computacional relativamente bajo. Recientemente, la introducción de conexiones residuales, junto con una arquitectura más tradicional, ha dado lugar a un rendimiento de vanguardia en el desafío ILSVRC de 2015; su rendimiento fue similar a la red Inception-v3 de última generación.

Se dan pruebas empíricas claras de que el entrenamiento con conexiones residuales de la Incepcion acelera el entrenamiento de las redes de Incepcion significativamente. También, hay pruebas de que las redes de Inception residuales superan a las redes de Inception sin conexiones residuales por un estrecho margen.

Se presentan varias arquitecturas nuevas y simplificadas para las redes de Incepcion residual y no residual. Estas variaciones mejoran el rendimiento del reconocimiento de un solo fotograma en la tarea de clasificación del ILSVRC 2012 de forma significativa. Se demuestra además cómo una escala de activación adecuada estabiliza el entrenamiento de redes de Incepción residual muy amplias. Con un conjunto de tres redes residuales y una Inception-v4, se logra un error del 3,08% en el conjunto de pruebas del reto de clasificación de ImageNet (CLS) (Szegedy *et al.*, 2017).

II.13 Python

Python es un lenguaje de alto nivel, fácil de entender y aprender, tal como menciona H.Bhasin (2019). Fue escrito en C a finales de los 80's por Guido Van Rossum. En la uĺtima década ha tenido un creciente uso en diferentes ámbitos de la ingeniería y ciencia de datos. Existen numerosas aplicaciones que se pueden desarrollar con Python, las cuales se pueden categorizar en las siguientes:

- Desarrollo web

- Desarrollo de software
- Interfaces gráficas GUI
- Aplicaciones científicas

La manera como se ejecuta es sencilla, pues se corre tal como un *script*. Por otro lado, su sintaxis es sencilla, debido a que no contiene llaves ni paréntesis y su identación es automática (H.Bhasin, 2019).

Otra de las características principales de Python, es su portabilidad, debido a que puede correr en cualquier sistema operativo.

II.14 Datetime

El módulo *datetime* proporciona clases para manipular fechas y horas. Aunque se admite la aritmética de la fecha y la hora, la implementación se centra en la extracción eficiente de atributos para el formato de salida y la manipulación.

II.14.1 OpenCV

OpenCV fue iniciado en Intel en 1999 por Gary Bradsky, y la primera versión salió en el año 2000. Vadim Pisarevsky se unió a Gary Bradsky para dirigir el equipo ruso de software OpenCV de Intel. En 2005, OpenCV se utilizó en Stanley, el vehículo que ganó el Gran Desafío de DARPA de 2005. Posteriormente, su desarrollo activo continuó bajo el apoyo de Willow Garage con Gary Bradsky y Vadim Pisarevsky al frente del proyecto. En la actualidad, OpenCV es compatible con multitud de algoritmos relacionados con la visión por ordenador y el aprendizaje automático, y se amplía día a día.

OpenCV es compatible con una gran variedad de lenguajes de programación como C++, Python, Java, entre otros, y está disponible en diferentes plataformas como

Windows, Linux, OS X, Android e iOS. También, se están desarrollando activamente interfaces para operaciones de alta velocidad en la GPU basadas en CUDA y OpenCL.

OpenCV-Python es la API de Python para OpenCV, que combina las mejores cualidades de la API C++ de OpenCV y el lenguaje Python.

II.14.2 OpenCV-Python

OpenCV-Python es una librería de enlaces de Python diseñada para resolver problemas de visión por ordenador. Incluye un amplio conjunto de algoritmos de visión por ordenador y de aprendizaje automático, tanto clásicos como de última generación. Estos algoritmos pueden utilizarse para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en vídeos, seguir los movimientos de la cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D a partir de cámaras estereoscópicas, unir imágenes para producir una imagen de alta resolución de toda una escena, encontrar imágenes similares a partir de una base de datos de imágenes, eliminar los ojos rojos de las imágenes tomadas con flash, seguir los movimientos de los ojos, reconocer paisajes y establecer marcadores para superponerlos a la realidad aumentada (Bradski, 2000).

OpenCV-Python hace uso de Numpy, que es una librería altamente optimizada para operaciones numéricas con una sintaxis al estilo de MATLAB. Todas las estructuras de array de OpenCV se convierten a y desde arrays de Numpy. Esto también facilita la integración con otras librerías que utilizan Numpy como SciPy y Matplotlib (Bradski, 2000).

II.15 Tensorflow library

TensorFlow es una interfaz para expresar algoritmos de aprendizaje automático y una implementación para ejecutar dichos algoritmos. Un cálculo expresado mediante

TensorFlow puede ejecutarse sin apenas cambios en una amplia variedad de sistemas heterogéneos, desde dispositivos móviles como teléfonos y tabletas, hasta sistemas distribuidos a gran escala de cientos de máquinas y miles de dispositivos de cálculo como tarjetas GPU. El sistema es flexible y puede utilizarse para expresar una gran variedad de algoritmos, incluidos los de entrenamiento e inferencia para modelos de redes neuronales profundas. Se ha utilizado para realizar investigaciones y poner en producción sistemas de aprendizaje automático en más de una docena de áreas de la informática y otros campos, para el reconocimiento del habla, la visión por computadora, la robótica, la recuperación de información, el procesamiento del lenguaje natural, la extracción de información geográfica y el descubrimiento computacional de fármacos (Abadi *et al.*, 2015). En la siguiente Figura, se puede observar lo se que se conoce como grafo.

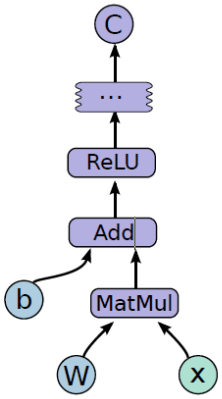


Figura 10: Ejemplo de fragmento de código de TensorFlow. Fuente: tensorflow-white paper.

II.16 Librería Scikit-learn

El proyecto *Scikit-learn* proporciona una librería de aprendizaje automático de código abierto para el lenguaje de programación python. (Pedregosa *et al.*, 2011). El objetivo de esta librería es dar acceso a las herramientas de aprendizaje automático de una manera sencilla, para los no expertos. Es una librería que proporciona modismos de

aprendizaje automático a un lenguaje de alto nivel de uso general. Entre otras cosas, incluye algoritmos clásicos de aprendizaje, herramientas de evaluación y selección de modelos, así como procedimientos de preprocesamiento. La librería se distribuye bajo la licencia BSD simplificada, lo que fomenta su uso tanto en el ámbito académico como en el comercial. *Scikit-learn* es una colección de clases y funciones que los usuarios importan a los programas de Python. El uso de *Scikit-learn* requiere, por tanto, conocimientos básicos de programación en Python. No se ofrece ninguna interfaz de línea de comandos ni una interfaz gráfica para los usuarios que no son programadores. Sin embargo, el uso interactivo es posible gracias al intérprete interactivo de Python y a su sustituto mejorado IPython Pérez y Granger (2007), que ofrece un entorno de trabajo similar a Matlab, diseñado específicamente para el uso científico. La librería ha sido diseñada para enlazar con el conjunto de paquetes numéricos y científicos centrados en las librerías NumPy y SciPy. NumPy (Walt *et al.*, 2011) aumenta Python con un tipo de datos de matrices numéricas contiguas y primitivas de cálculo de matrices rápidas, mientras que SciPy (Virtanen *et al.*, 2020) lo amplía con operaciones numéricas comunes, ya sea implementándolas en Python/NumPy o envolviendo las implementaciones existentes de C/C++/Fortran. Sobre la base de esta pila, se creó una serie de librerías llamadas scikits, para complementar SciPy con conjuntos de herramientas específicas del sector. En la actualidad, las dos más populares y completas que se encarga del procesamiento de imágenes son *scikit-learn* y *scikit-image*.

Scikit-learn fue desarrollado a principios de 2007 por un equipo internacional de más de una docena de desarrolladores, en su mayoría investigadores de diversos campos (informática, neurociencia, astrofísica, entre otros). Actualmente, el proyecto se beneficia de colaboradores ocasionales que proponen pequeñas correcciones de errores o mejoras. El desarrollo se lleva a cabo en GitHub, una plataforma que facilita enormemente este tipo de colaboración. Debido al gran número de desarrolladores,

se hace hincapié en que el código debe seguir unas directrices de calidad específicas, como la consistencia del estilo y la cobertura de las pruebas unitarias. Los cambios importantes deben ser revisados por al menos dos desarrolladores de código que no participen en la implementación del cambio propuesto (Pedregosa *et al.*, 2011).

II.17 Aumentos de datos basados en manipulaciones básicas de la imagen

En esta sección se describe diferentes aumentos basados en transformaciones geométricas y muchas otras funciones de procesamiento de imágenes. La clase de aumentos que se discute a continuación podría caracterizarse por su facilidad de implementación. La comprensión de estas transformaciones proporcionará una base útil para seguir investigando las técnicas de aumento de datos.

Cabe destacar que en la etapa de aumentación de datos, es importante conocer la seguridad de la aplicación. La seguridad de un método de aumento de datos se refiere a su probabilidad de preservar la etiqueta después de la transformación. Por ejemplo, las rotaciones y los giros suelen ser en los retos de ImageNet, como la clase gato frente a la clase perro, pero no son seguros para tareas de reconocimiento de dígitos, como 6 frente a 9. Una transformación que no preserve la etiqueta podría dar una respuesta que indique que no tiene confianza en su predicción. Sin embargo, para conseguirlo sería necesario refinar las etiquetas después de la transformación (Bagherinezhad *et al.*, 2018). Si la etiqueta de la imagen después de una transformación sin preservación de la etiqueta es algo así como como [0,5 0,5], el modelo podría aprender predicciones de confianza más robustas. Sin embargo, construir etiquetas refinadas para cada aumento de datos no seguro, es un proceso computacionalmente caro.

A continuación se despliegan las aumentaciónes de datos de imágen basados en transformaciones geométricas según Shorten y Khoshgoftaar (2019):

II.17.1 Girado

La fijación del eje horizontal es mucho más común que la del eje vertical. Este aumento es uno de los más fáciles de aplicar y ha demostrado su utilidad en conjuntos de datos como como CIFAR-10 e ImageNet. En conjuntos de datos de reconocimiento de texto como MNIST o SVHN, no se trata de una transformación que conserve la etiqueta.

II.17.2 Espacio del color

Los datos de las imágenes digitales suelen codificarse como un tensor de la dimensión (altura x ancho x color canales). Realizar aumentos en el espacio de los canales de color es otra estrategia que resulta muy práctica de aplicar. Los aumentos de color muy sencillos consisten en aislar un solo canal de color, como R, G o B. Una imagen puede convertirse rápidamente en su representación en un canal de color aislando esa matriz y añadiendo 2 matrices cero de los otros canales de color. Además, los valores RGB pueden manipularse fácilmente con simples operaciones matriciales para aumentar o disminuir el brillo de la imagen. Más aumentos de color, más avanzados, provienen de la derivación de un histograma de color que describe la imagen. El cambio de los valores de intensidad en estos histogramas da lugar a alteraciones de la iluminación como las que se utilizan en las aplicaciones de edición fotográfica.

II.17.3 Recorte de imagen

El recorte de imágenes puede utilizarse como un paso práctico de procesamiento para datos de imágenes con dimensiones mixtas altura y anchura mezcladas, recortando una parte central de cada imagen. Además, el recorte aleatorio también puede utilizarse para proporcionar un efecto muy similar al de las traslaciones. El contraste entre el recorte aleatorio y las traslaciones es que el recorte reducirá el tamaño de la entrada, como (256,256)$\rightarrow$(224, 224), mientras que las traslaciones conservan las las dimensiones

espaciales de la imagen. Dependiendo del umbral de reducción elegido para recorte, puede que no sea una transformación que conserve la etiqueta.

II.17.4 Rotación

Los aumentos de rotación se realizan girando la imagen a la derecha o a la izquierda en un eje entre 1° y 359°. La seguridad de los aumentos de rotación está fuertemente determinada por el parámetro del grado de rotación. Las rotaciones ligeras, como las comprendidas entre 1 y 20 o entre -1 y -20, podrían ser útiles en tareas de reconocimiento de dígitos como MNIST, pero ha cierto grado de rotación, la etiqueta de los datos ya no se conserva después de la transformación.

II.17.5 Traslación

Desplazar las imágenes hacia la izquierda, la derecha, arriba o abajo puede ser una transformación muy útil para evitar sesgo posicional en los datos. Por ejemplo, si todas las imágenes de un conjunto de datos están centradas, lo que es común en los conjuntos de datos de reconocimiento de rostros, requerirían que el modelon se pruebe también con imágenes perfectamente centradas. Como la imagen original se traslada en una dirección, el espacio restante puede rellenarse con un valor constante como 0 s o 255 s, o bien, con ruido aleatorio o gaussiano. Este relleno preserva las dimensiones espaciales de la imagen después de la aumentación.

II.17.6 Inyección de ruido

La inyección de ruido consiste en inyectar una matriz de valores aleatorios extraídos normalmente de una distribución gaussiana. La inyección de ruido ha sido probada en nueve conjuntos de datos del repositorio de la UCI (Asuncion y Newman, 2007). Añadir ruido a las imágenes puede ayudar a las CNN a aprender características más robustas (Moreno-Barea *et al.*, 2018).

II.17.7 Filtros Kernel

Los filtros de Kernel son una técnica muy popular en el procesamiento de imágenes para afinar y desenfocar imágenes. Estos filtros funcionan deslizando una matriz n $\times$n por una imagen con un filtro de desenfoque gaussiano, que dará lugar a una imagen más borrosa o con un filtro de bordes verticales u horizontales de alto contraste, que permitirá una imagen más nítida en los bordes. Intuitivamente, el desenfoque de imágenes para el aumento de datos podría conducir a una mayor resistencia al desenfoque por movimiento durante pruebas. Además, la nitidez de las imágenes para el aumento de datos podría dar lugar a la encapsulación de más detalles sobre los objetos de interés. La nitidez y el desenfoque son algunas de las formas clásicas de aplicar filtros de núcleo a imágenes.

II.18 Computación de alto rendimiento

La Computación de alto rendimiento (HPC, por sus siglas en inglés) es un campo de trabajo que se refiere a todas las facetas de la tecnología, la metodología y la aplicación asociadas a la consecución de la mayor capacidad de computación posible en cualquier momento y tecnología. Se trata de una clase de máquinas digitales electrónicas denominadas "supercomputadoras" para darle solución a una amplia gama de problemas o y realizar diversas "aplicaciones" computacionales (alternativamente, "cargas de trabajo") tan rápido como sea posible. La acción de realizar una aplicación en un supercomputadora se denomina ampliamente "supercomputación" (Sterling *et al.*, 2017).

- **Clúster:** Un clúster esta conformado por mas de un CPU, donde el primero representa el nodo 0, y los demas CPUś son nodos 1, 2...N. Cada nodo puede tener uno o varios núcleos interconectados mediante un bus de comunicación. Los nodos reciben órdenes desde el nodo cero.

II.19 Computación de frontera

La computación de frontera, es donde se procesan los datos cerca de los sensores o en la proximidad de los dispositivos moviles Kalyani y Collier (2021). El cómputo de frontera es una tecnología que viene a resolver problemas de latencia y acceso de recursos de cómputo al usuario final, es decir, mientras que el cómputo en la nube tiene el mayor poder respecto a recursos, motores de alojamiento y procesamiento, resulta un retardo en el momento de las llamadas desde usuario final hasta la nube.

En los últimos veinte años se ha reducido la separación entre un CPU y un sistema embedido, tal como lo es un microcontrolador o microprocesador. Actualmente, se cuentan con microprocesadores con una muy baja potencia de consumo, memoria RAM, conectividad a la red WiFi, bluetooth de baja potencia, entre otros periféricos. Las características de estos dispositivos han facilitado la adopción de la nueva revolución industrial, conocida como Industria 4.0, mediante la llegada del "Internet de las Cosas", debido a que estos microprocesadores pueden conectarse a la nube mediante la red inalámbrica Wifi y el uso de protocolos estandar como MQTT para envío y recepción de mensajes de telemetría.

Por otro lado, en el campo del aprendizaje automático, también existen dispositivos que pueden alojar algún modelo de red neuronal artificial. A continuación se enlistan algunos de ellos:

- Xavier AGX
- Jetson nano 2GB
- Raspberry Pi4
- Coral

II.20 Computación en la nube

El cómputo en la nube es una de las tecnologías emergentes en todos los campos y se denomina como el tipo de computación retransmitida en internet. Las principales funciones de la computación en nube son el alojamiento, la entrega de diversas aplicaciónes y servicios a través de internet. Estos servicios van desde servidores SQL hasta servidores de inferencia, mediante el uso de GPU, dedicados a la predicción de modelos de aprendizaje automático. Dependiendo de las demandas y requisitos del usuario, la computación en nube ofrece como principales recursos informáticos a los usuarios como el tener un mayor espacio de almacenamiento, un servidor de alto rendimiento, varios sistemas operativos para diversas plataformas y una red. La demanda de estos recursos por parte del usuario aumenta día a día, sin embargo, esto trae consigo el inconveniente de la seguridad, que se considera un problema muy grave en la computación en nube.

El cómputo en la nube, provee infraestructura, plataforma y software como servicio (Yousefpour *et al.*, 2019) el cual se define como un paradigma bien establecido para construir sistemas centralizados Kalyani y Collier (2021).

II.21 Revisión del estado del arte

Conforme se ha avanzado, en los últimos veinte años, respecto al tema de redes neuronales artificiales, se ha venido mejorando la exactitud de resolución a problemas categorizados como difíciles de aboradr desde el punto de vista del cómputo. En la siguiente imagen podemos ver las tendencias respecto al avance en la detección de objetos mediante el uso de redes neuronales artificiales.

Cabe destacar, que el uso de las redes neuronales convolucionales, han venido resolviendo el problema del reconocimiento y detección de objetos dada su naturaleza

en entornos industriales. A continuación se presentan los trabajos relacionados a la detección o clasificación de objetos o defectos dentro de un ambiente de manufactura.

En 2018, se publica un artículo titulado *"Detection and segmentation of manufacturing defects with convolutional neural networks and transfer learning"* (Ferguson *et al.*, 2018), que propone un sistema para la identificación de defectos de fundición en imágenes de rayos X. Este trabajo utiliza la arquitectura de CNN basada en regiones de máscara (RCNN, por su acrónimo en inglés). El sistema de detección de defectos propuesto realiza simultáneamente la detección de defectos y la segmentación en las imágenes de entrada, lo que lo hace adecuado para una serie de tareas de detección de defectos. Se demuestra que el entrenamiento de la red para realizar simultáneamente la detección de defectos y la segmentación de instancias de defectos da como resultado una mayor precisión en la detección de defectos que el entrenamiento en la detección de defectos solamente. Se aprovecha el aprendizaje de transferencia para reducir la demanda de datos de entrenamiento y aumentar la precisión de predicción del modelo entrenado. En concreto, el modelo se entrena primero con dos grandes conjuntos de datos de imágenes de libre acceso antes de afinarlo con un conjunto de datos de rayos X de fundición de metales relativamente pequeño. La precisión del modelo entrenado supera el rendimiento de la base de datos de imágenes de rayos X del Grupo de Inteligencia de Máquina (GDXray) Castings y es lo suficientemente rápido como para ser utilizado en un entorno de producción. El sistema también rinde bien en el conjunto de datos GDXray Welds. Se realizan varios estudios en profundidad para explorar cómo el aprendizaje por transferencia, el aprendizaje multitarea y el aprendizaje multiclase influyen en el rendimiento del sistema entrenado.

Un año mas tarde, en 2019, se publica un trabajo relacionado a la identificación de los defectos de la soldadura (Bacioiu *et al.*, 2019). Dicho trabajo consistió en un sistema para evaluar la soldadura con gas inerte de tungsteno (TIG) utilizando una cámara de

alto rango dinámico (HDR) con la ayuda de redes neuronales artificiales (ANN) para el procesamiento de imágenes. Este estudio propone un nuevo conjunto de imágenes del proceso de soldadura TIG en el espectro visible con un contraste mejorado, similar al que vería normalmente un soldador, y un modelo para computar una etiqueta que identifique la imperfección de la soldadura. Se presenta de forma exhaustiva el progreso (precisión) alcanzado con el nuevo sistema en distintos grados de complejidad de la categorización.

En 2021, se publica otro trabajo que a diferencia de los anteriores, se utiliza un conjunto de imágenes artificiales, es decir, se generan de manera computarizada, debido a la dificultad de obtener imágenes de defectos. Este trabajo fue titulado *"Spot Welding Defect Detection Using Synthetic Image Dataset on Convolutional Neural Networks"*, donde se utilizaron técnicas de aprendizaje profundo mediante redes neuronales convolucionales (ConvNet) para detectar defectos de soldadura. Este estudio propone el uso de imágenes sintéticas que imitan la característica de la soldadura por puntos defectuosos como conjunto de datos de entrada. Reproduciendo el anillo de la zona afectada por el calor (HAZ), el anillo de la zona de fusión (FZ) y el anillo de fusión en diferentes tamaños, colores y formas, se pueden generar imágenes de soldadura por puntos anormales y típicos utilizando un programa de procesamiento de imágenes escrito en Python. Estas imágenes se utilizaron para entrenar dos niveles diferentes de clasificación ConvNet. Los resultados mostraron que utilizando dos mil imágenes artificiales, la ConvNet puede clasificar la soldadura por puntos defectuosos con una precisión superior al 98% (Tantrapiwat, 2021).

Respecto a la detección o reconocimiento de defectos en metales, se encontró un artículo relacionado a la detección de defectos sobre acero (Ali *et al.*, 2021). Dicho artÍculo fue publicado bajo el nombre *"Detection of Steel Surface Defects Using U-Net with Pre-trained Encoder"*, donde se propuso utilizar la red de aprendizaje profundo

U-Net para la detección de defectos. U-Net es un tipo de red neuronal convolucional que realiza la segmentación semántica de imágenes, fue originalmente desarrollada para lograr la segmentación de imágenes biomédicas, ya que U-Net requiere conjuntos de entrenamiento más pequeños en comparación con las CNN típicas. Se determinó que la segmentación puede realizarse con resultados prometedores cuando se utiliza el modelo U-Net con un codificador preentrenado, ya que el aprendizaje por transferencia aumenta la precisión y la estabilidad de los modelos de detección de objetos.

Asi pues, hemos encontrado trabajos donde el uso de la red neuronal convolucional sigue siendo la principal en emplearse. También cabe destacar el trabajo relacionado al reconocimiento de acciones humanas (*"Human Action Recognition"*).

II.22 Resumen

En este capítulo se presentan los conceptos básicos de la metodología six sigma, proceso de ensamble manual y sus variantes, asi como el aprendizaje automático y sus diferentes ramas.

Además, se dan a conocer diferentes conceptos del aprendizaje automático mediante el uso de redes neuronales artificiales profundas (y sus variantes) para el reconocimiento y detección de problemas durante procesos de manufactura.

También, se da a conocer una descripción del lenguaje python y sus librerias, utilizadas en éste trabajo, las cuales también son utilizadas para el preprocesamiento de los datos.

Se describen las diferentes técnicas para aumentación de datos de imágenes que son ultilizadas para evitar sobreentrenamiento de la red neuronal artificial.

Además, se presentan los recursos que se utilizan para un entrenamiento, los cuales pueden variar según el tiempo que se tarde para dicho objetivo, pues existe el cómputo de alto rendimiento, dónde en los últimos años el uso de GPU, ha cobrado popularidad,

dado a que son mayormente rápidos que el uso de múltiples CPUs.

Finalmente, se realiza una revisión del estado del arte, donde se describen los trabajos realizados en el ámbito de reconocimiento de imágenes, asi como la detección de objetos como tarea objetivo de la visión artificial profunda.

Capítulo III

Materiales métodos

III.1 Materiales

Los materiales empleados para el desarrollo metodológico del presente trabajo de tesis, se enlistan a continuación:

- **Webcam USB LifeCam Studio Microsoft:** Interfaz USB 2.0, micrófono incorporado, máxima resolución de video: 1920 x 1080 Pixeles; longitud de cable: 1,829 m.
- **Laptop Dell:** Modelo *precision* 5550 Intel(R) Core(TM) i7-10850H CPU @ 2.70GHz 2.71 GHz. 64GB RAM. GPU dedicado 12GB. 500GB Disco Duro.
- **Componentes galvanizados o plateados en zinc:** Se seleccionaron 5 tipos de componentes llamados *"baseplates"* y una fixtura de ensamble, esto con la finalidad de obtener la sexta clase con la fixtura sin componente.
- **Cómputo en la nube:** *Google Colaboratory* es un laboratorio virtual para poder programar en la nube mediante *notebooks*. También llamado Colab, es un producto de *Google Research*. Colab permite que se pueda escribir y ejecutar código arbitrario de Python en el navegador. Es ideal para aplicarlo en proyectos de aprendizaje automático, análisis de datos y educación. En términos técnicos, Colab es un servicio de *notebooks* de Jupyter alojados que no requiere instalación para usarlo y brinda acceso sin costo a recursos computacionales, incluidas GPU y TPU.

III.2 Método propuesto

Este trabajo se basa en una estrategia de aprendizaje automático tal como se muestran en la Figura 11, Figura 12, Figura 13, Figura 14, and Figura 15.

La fase cero consiste en los criterios de selección del método de aprendizaje automático o six sigma, mediante la definición del problema y su disección, es decir, tratar de conocer la naturaleza de las probables causas raíces para determinar si se debiera emplear un método de aprendizaje automático o técnicas Six Sigma.

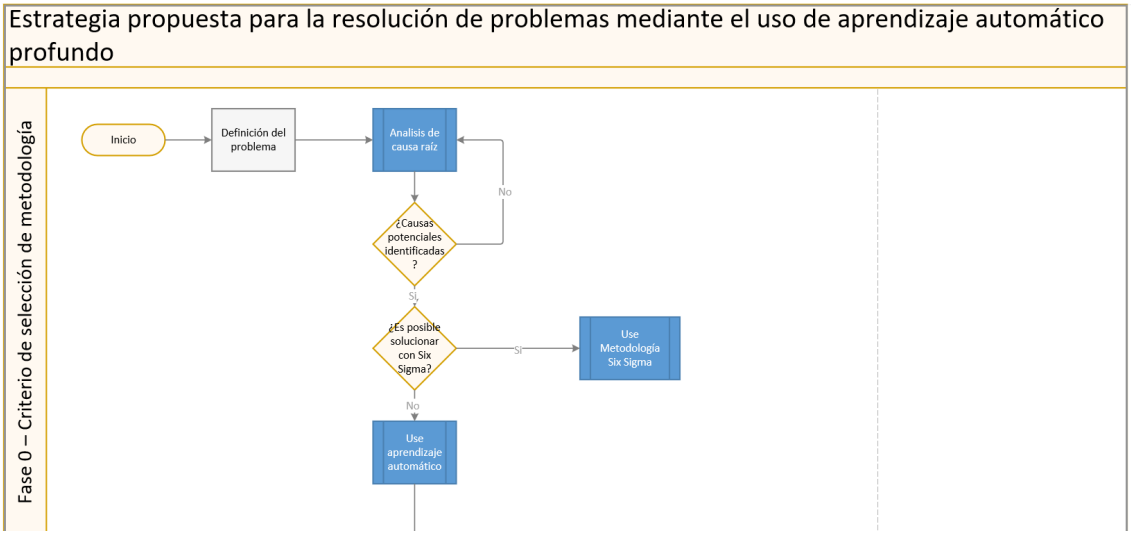


Figura 11: Fase 0 - Diagrama de flujo propuesto.

La fase uno, consiste en la generacion del conjunto de imágenes, asi como su aumentacion de dato, tal como se muestra en la Figura 12.

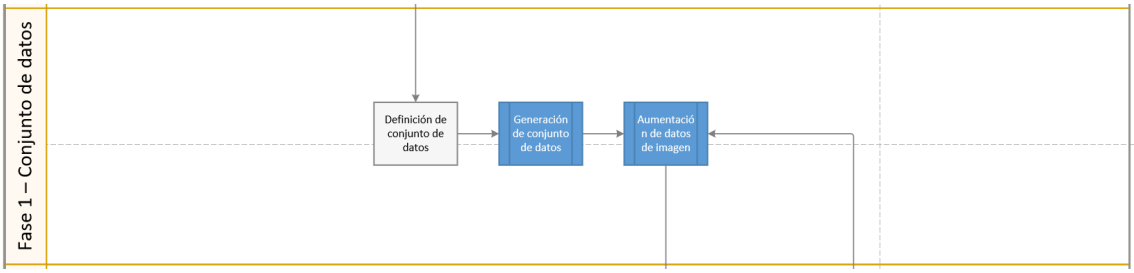


Figura 12: Fase 1 - Diagrama de flujo propuesto.

Durante la fase dos, se lleva a cabo el entrenamiento y validación (ver Figura 13).

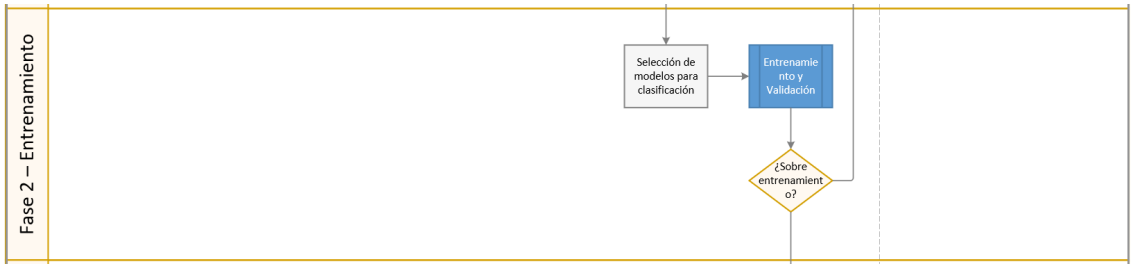


Figura 13: Fase 2 - Diagrama de flujo propuesto.

La evaluación y pruebas del modelo, se lleva a cabo durante la fase tres, aqui es donde se generan los reportes de rendimiento, a traves de las curvas ROC y Matriz de confusión (ver Figura 14).

Figura 14: Fase 3 - Diagrama de flujo propuesto.

La cuarta y última fase, se refiere al modo de inferencia, es decir el modelo se coloca en modo predicción, tal como indica la Figura 15.

Figura 15: Fase 4 - Diagrama de flujo propuesto.

III.2.1 Criterio de selección de metodología

Dentro del ámbito industrial, una de las herramientas más interesantes y antiguas de análisis de causa raíz, es el diagrama causa-efecto, también conocido como "Diagrama de Ishikawa" (Voelkel y Ishikawa, 1989) o el "Análisis de Árbol de Fallos" (Kritzinger, 2017). En la fase cero, se utilizaron dichos métodos, basados en la metodología six sigma, para encontrar la causa raíz del problema de investigación.

El proceso metodológico comenzó con el uso de la herramienta de análisis de causa raíz conocida como Análisis de Árbol de Fallos para identificar las posibles causas raíz del problema como se muestra en la Figura 16. A partir de éste análisis, se determinó que las principales causas raíces del comportamiento del sensor de visión actual, reside en la variación de la luz ambiental y de las diferentes tonalidadades del zincado de los componentes, ya que una misma clase (modelo de componente) puede ser diferente desde el punto de vista del contraste. Además, la variación de la luz ambiental en cada estación de trabajo provoca una mala detección del color RGB.

A partir de lo anterior, se generó la oportunidad de aplicar una técnica de *Deep Learning* para entrenar un modelo que clasifique los diferentes componentes de zinc y la ausencia de este componente en la fixtura de ensamble. Esto debido a que dichas

causas potenciales son variables que no se pueden controlar por su naturaleza.

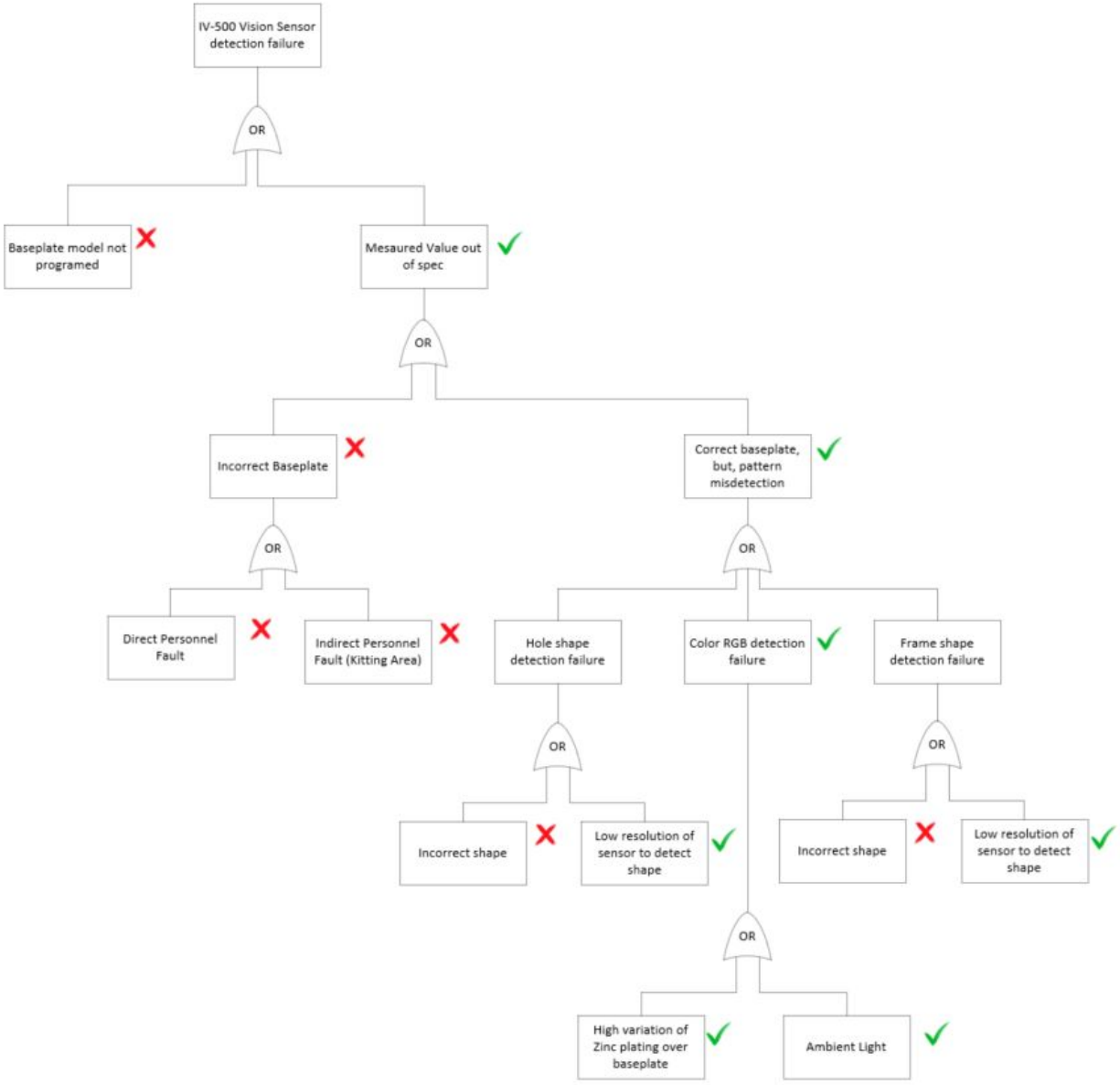


Figura 16: Análisis de árbol de fallas para identificar posibles causas del problema. Fuente: Elaboración propia

III.2.2 Generación del conjunto de datos

Se generaron dos carpetas para almacenar las imágenes de los diferentes componentes de zinc y la fixtura de ensamble. El entrenamiento y prueba estuvieron subdivididos en

seis clases: cada una de las cinco primeras clases representa un componente diferente, y la última, se utiliza para identificar la ausencia de un componente de zinc.

Esta fase comenzó con la creación de un conjunto de imágenes adquiridas directamente en la línea de fabricación a través de una cámara USB 2.0 y la posterior generación de 1542 imágenes (véase la Figura 17) mediante un código escrito en lenguaje Python 3.8.

Para este paso, fué necesario instalar openCV, una librería utilizada para poder accesar a captura de imágenes utilziando alguna cámara conectada a la computadora.

A continuación se muestra el código fuente utilizado para poder generar las imágenes para cada clase. Para la captura de éstas imágenes se utilizó una laptop Dell, cuyas características fueron descritas previamente.

```
/* USER CODE BEGIN 1 */
#Author: Edgar Ramos
#University Autonomous of Baja California
#Date: Apr-2022
# importing OpenCV library & Datetime
import cv2 as cv
from datetime import datetime

port=0
cam = cv.VideoCapture(port)

while(True):

# Read input from webcam
    result, image = cam.read()
    if(result):
        #Display the image captured on the window
        cv.imshow("Webcam", image)

        #Generate unique tag name based on the timestamp
        date = datetime.now().strftime("%Y_%m_%d-%I%M%S_%p")
        print(f"Class1_{date}")

        #Save image to specific class name, in this case class1 is by
   default
        cv.imwrite("./Dataset/Class1/Class1_"+str(date)+".png", image)

        #Wait for a ESC key pressed with 100ms delay, if not pressed,
   capture will continue
        c = cv.waitKey(100)
        if c == 27:
```

```
            break
        cv.destroyWindow("Webcam")

    # If input corrupted jump to else!
    else:
        print("input issues detected. Check webcam")
/* USER CODE END 1 */
```

Figura 17: Conjunto de batch de 16 imágenes de los componentes de zinc. Fuente: Elaboración propia.

III.2.3 Selección del entorno de programación

Se seleccionó Google Colab (Bisong, 2019) debido a que éste laboratorio de programación virtual cuenta con hardware de última generación, como GPU y TPU, que puede ser utilizado para entrenar el modelo más rápido que la CPU, cuyo lenguaje de codificación fué Python 3 (van Rossum y Drake, 2009), TensorFlow 2, Keras como

librerías principales para el entrenamiento (Abadi *et al.*, 2015), sci-kit learn (Pedregosa *et al.*, 2011) como librería para el reporte de métricas, y finalmente, wandb para el seguimiento y monitorización del proceso de entrenamiento (Biewald, 2020a).

III.2.4 Aumentación de imágenes

Este paso de preprocesamiento se realizó para aumentar la robustez del propio conjunto de datos. Se aplicaron algunas operaciones al conjunto de datos de entrenamiento con una proporción de 80/20, dejando fuera el 20% para la validación durante las épocas de entrenamiento.

III.2.5 Configuración de aumentación de datos para el conjunto de imágenes de componentes de zinc

Se utilizó la librería TensorFlow (Abadi *et al.*, 2015) para producir generadores de entrenamiento con estas imágenes. Esta etapa de preprocesamiento consistió en generar un conjunto de datos aleatorios con parámetros específicos. Estos parámetros se describen a continuación:

- Reescalado: Cada píxel fué normalizado, dividiendo por 255.
- Rango de rotación: La rotación aleatoria se fijó en un 40
- Rango de cizallamiento: La deformación de la imagen se fijó en un 20%.
- Rango de zoom: Zoom aleatorio dentro del rango de 20%.
- Rango de desplazamiento de la anchura: Imagen desplazada horizontalmente dentro del rango del 20%.
- Rango de brillo: El brillo aplicó de forma aleatoria dentro de un rango de 0 a 1.

- Rango de desplazamiento de la altura: La imagen se desplazó verticalmente dentro de un rango de 20%.
- Voltear horizontalmente: La imagen se volteó horizontalmente y de forma aleatoria.
- División de la validación: La carpeta del subconjunto de entrenamiento se dividió con una proporción de 80/20.

En el siguiente código, se muestra dicha configuración:

```
/* USER CODE BEGIN 2 */
#Author: Edgar Ramos
#University Autonomous of Baja California
#Date: Apr-2022
datagen_kwargs = dict(rescale=1./255,rotation_range=40,
        shear_range=0.2,
        zoom_range=0.2,
        width_shift_range=0.2,
        brightness_range=[0,1],
        height_shift_range=0.2,
        horizontal_flip=True,validation_split=0.2)
/* USER CODE END 2 */
```

Como se puede apreciar en la figura 18, diferentes imágenes correspondientes al conjunto de imágenes original, fueron transformadas aleatoriamente, esto con la finalidad de evitar sobreentrenamiento y aumentar el tamaño de características del conjunto de datos original.

Figura 18: Imágenes después del proceso de aumentación de datos. Fuente: Elaboración propia.

III.2.6 Etapa de entranemiento utilizando de transferencia de aprendizaje

En esta fase se importaron cinco modelos de redes neuronales profundas, que son públicas, y que se muestran en la Tabla II. Se utilizó la arquitectura de cada uno de ellos con el objetivo de entrenar el modelo con el 80% de las imágenes y el 20% para la validación. Se eliminó su capa de salida para conectar nuestra capa de salida de seis neuronas y se deshabilitó la bandera de entrenamiento para mantener los pesos y las pérdidas del entrenamiento.

Por otro lado, se configuraron los siguientes hiperparámetros para el entrenamiento

de cada modelo:

- ***Learning rate:*** 0.001
- ***Optimizador:*** El optimizador Adam fue el elegido debido a que es el que representa menor coste de entrenamiento.
- ***Dropout:*** 20% de las neuronas se apagaron durante cada época, esto con la finalidad de evitar sobreentrenamiento.

Una vez realizado el entrenamiento, se compararon los resultados de cada modelo y se determinó cuál era el mejor para implementar en la detección de los componentes de zinc.

Tabla II: Keras API public model.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.79	0.945	22910480	126
InceptionV3	92 MB	0.779	0.937	23851784	159
ResNet101V2	171 MB	0.772	0.938	44675560	-
ResNet152V2	232 MB	0.78	0.942	60380648	-
InceptionResNetV2	215 MB	0.803	0.953	55873736	572

```
/* USER CODE BEGIN 3 */
#Author: Edgar Ramos
#University Autonomous of Baja California
#Date: Apr-2022

def train_model(TL_Model,w,h):
  wandb.init(project=TL_Model.__name__, entity="computervision",
  config = {
    "learning_rate": 0.001,
    "epochs": 30,
  })
  config = wandb.config

  train_generator,valid_generator,dataset_labels=
   Create_Images_for_training(w,h)

  pre_trained_model=TL_Model(input_shape=(w,h,3),
                             include_top=False,
                             weights='imagenet')
```

```
  for layer in pre_trained_model.layers:
    layer.trainable = False

  model = tf.keras.Sequential([pre_trained_model,
                               tf.keras.layers.
   GlobalAveragePooling2D(),
                               tf.keras.layers.Dropout(0.2),
                               tf.keras.layers.Dense(6, activation="
   softmax")
                              ])


  optimizer=keras.optimizers.Adam(learning_rate=config.learning_rate)

  model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=METRICS)

  steps_per_epoch = np.ceil(train_generator.samples/train_generator.
   batch_size)
  val_steps_per_epoch = np.ceil(valid_generator.samples/
   valid_generator.batch_size)

  history = model.fit(train_generator,
         epochs=config.epochs,
         verbose=1,
         steps_per_epoch=steps_per_epoch,
         validation_data=valid_generator,
         validation_steps=val_steps_per_epoch,callbacks=[WandbCallback
   ()])
  model.save('Master_Baseplates'+'_'+TL_Model.__name__+str(dt.datetime
   .now())+'.h5')
  wandb.save('Master_Baseplates' + '_'+ TL_Model.__name__+str(dt.
   datetime.now())+'.h5')

  wandb.finish()
  return history,model
/* USER CODE END 3 */
```

III.2.7 Evaluación del modelo

Para la evaluación del modelo, se utilzaron 240 imágenes, que no formaron parte del entrenamiento ni validación. Para ello, se generaron métricas para comparar los cinco modelos diferentes entrenados previamente.

Las métricas y puntuación de evaluación fueron: verdadero positivo, verdadero negativo, falso positivo, falso negativo, precisión, puntuaci ón F1, *Recall*, Kappa de

Cohen, perdida de Hamming y Área bajo la curva (AUC, por sus siglas en inglés).

Por otro lado, se graficaron las curvas ROC y matrices de confusión de cada modelo durante cada corrida. Es por ello, que se denominan matrices de confusión multi corrida (multi-run en inglés).

III.3 Resumen

El procedimiento que se definió para el desarrollo de ésta metodología se compuso de cinco fáses. La fase cero, consistió en el análisis de causa raíz del problema, dónde se utilizó una de las herramientas de *Six Sigma*, la cuál fué el Árbol de fallos. A partir de lo anterior, se identificaron dos factores ponderantes que impactan en la detección de componentes de zinc con el actual sistema de visión clásico. Dichos factores fueron: la variación luz ambiental y la variación en el acabado de cada componente de zinc. Es por ello que se consideró pertinente la aplicación del aprendizaje automático como estrategia innovadora para abordar dicho problema.

En la fase uno, se creó el conjunto de datos de imágenes, cuyas herramientas utilizadas fueron: cámara webcam USB, laptop, aplicación python y la librería OpenCV para grabar las imágenes. Todas ellas, con un tamaño de 640 por 480 píxeles, color RGB y formato PNG. Durante ésta fase se realizó la aumentación de datos de imágenes para poder mejorar el entrenamiento de la red neuronal.

Durante la fase dos, se realizó el entrenamiento de cinco modelos, los cuales son los mayormente utilzados para clasificación. Dicho entrenamiento consistió en treinta épocas y diez corridas por modelo. El monitoreo fue realizado utilizando la API de WandB (Biewald, 2020b).

Posteriormente, en la fase 3, se mostraron los pasos de la validación del modelo, donde se eligieron métricas y puntuaciones de cada modelo para su respectiva comparación. Una vez que se tuvo la tabla de comparación, se eligió el modelo que

tuvo mayor puntuación en cuanto a precisión y *recall*.

Finalmente, en la fase 4, una vez que guardaron los modelos en formato **.h5*, estos fueron guardados en la nube, para su posterior uso como modo inferencia, es decir, en modo predictivo, donde cada vez que se le muestró una imágen, este modelo pasó la imágen en un sólo sentido para mostrar la predicción según los pesos obtenidos en toda la red neuronal artificial.

Capítulo IV

Resultados y discusiones

IV.1 Métricas de rendimiento

Para evaluar el rendimiento de los modelos, se basó en las métricas de la biblioteca scikit-learn (Pedregosa *et al.*, 2011). Las métricas utilizadas son verdadero positivo, verdadero negativo, falso positivo, falso negativo, precisión, puntuación F1, recuerdo, Kappa de Cohen, pérdida de Hamming y AUC.

Se utilizaron 240 muestras de prueba equilibradas diferentes para probar los distintos modelos. Los parámetros de puntuación se muestran en la tabla III para ver cómo el modelo DL es bueno para clasificar. Para los resultados mostrados, los modelos se entrenaron diez veces y se calculó la media de cada métrica.

Tabla III: Rendimiento medio de los algoritmos de aprendizaje profundo basado en 30 épocas y 10 ejecuciones de entrenamiento.

Scoring Parameters(Avg)	InceptionV3	ResNet101V2	Xception	ResNet152V2	InceptionResNetV2
True positive	235	234	230	221	211
True negative	1195	1194	1190	1181	1171
False positive	5	6	10	19	29
False negative	5	6	10	19	29
Accuracy	99.3%	99.2%	98.6%	97.4%	96.0%
Recall	97.9%	97.5%	95.8%	92.1%	87.9%
Precision	97.9%	97.5%	95.8%	92.1%	87.9%
F1 Score	97.9%	97.5%	95.8%	92.1%	87.9%
Cohen's Kappa	97.6%	97.4%	95.3%	90.8%	85.5%
Hamming Loss	0.02	0.022	0.04	0.077	0.121
AUC	96.9%	93.8%	97.4%	94.6%	96.0%

La métrica de rendimiento intuitiva más sencilla es la exactitud, que no es más que la relación entre las observaciones predichas correctamente y todas las observaciones. La

precisión (1) es una métrica útil, pero sólo cuando los conjuntos de datos son simétricos y los valores de falsos positivos y falsos negativos son casi iguales. En este trabajo en particular, estos valores se muestran en la tabla III.

$$Accuracy = \frac{TP + TN}{(TP + FP + TN + FN)} \tag{1}$$

La relación entre las observaciones positivas predichas con exactitud y todas las observaciones de la clase real se conoce como Recall (2), tasa de aciertos, sensibilidad o simplemente tasa de verdaderos positivos. Esta puntuación puede considerarse una métrica de rendimiento de calidad.

$$Recall = \frac{TP}{(TP + FN)} \tag{2}$$

La relación entre las observaciones positivas predichas con exactitud y el total de observaciones positivas esperadas se conoce como precisión o valor predictivo positivo (3).

$$Precision = \frac{TP}{(TP + FP)} \tag{3}$$

La media ponderada de Precisión y Recuperación es la puntuación $F1$-$Score$ (4). Como resultado, esta puntuación considera tanto los falsos positivos como los falsos negativos.

$$F1 - Score = \frac{2 \times Recall \times Precision}{(Recall + Precision)} \tag{4}$$

La Exactitud, el Recall, la Precisión y la puntuación $F1$-$Score$ se aproximan entre sí porque los datos de prueba están equilibrados. No obstante, los resultados sugieren que estos modelos son buenos: InceptionV3 obtiene un 97,9% de precisión, ResNet101V2 un 97,5% y el modelo Xception un 95,8%, mientras que el resto de los modelos, ResNet152V2 e InceptionResNetV2, obtienen un 92,1% y un 87,9% respectivamente.

Por otro lado, para una pérdida de Hamming baja, se consigue el modelo de mejor ajuste, InceptionV3 y ResNet101V2 muestra una pérdida de Hamming muy baja (0,02) y un Kappa de Cohen alto (coeficiente que mide la concordancia entre los valores predichos y los reales) con un 97,6% y un 97,4% respectivamente.

En la siguiente sección se muestran las curvas ROC de varias ejecuciones por modelo y la matriz de confusión de varias ejecuciones, que son útiles para analizar los resultados gráficamente. La curva de características operativas del receptor, o curva ROC, es una métrica para evaluar el rendimiento de un modelo clasificador. La curva ROC ilustra la tasa de verdaderos positivos en relación con la tasa de falsos positivos, destacando la sensibilidad del modelo clasificador. Una matriz de confusión es un resumen de los resultados de la predicción del problema de clasificación. El número de predicciones exitosas y no exitosas se totaliza y se desglosa por clase utilizando valores de conteo. La matriz de confusión (Pearson, 1904) ilustra las diferentes formas en que un modelo de clasificación se desconcierta al hacer predicciones.

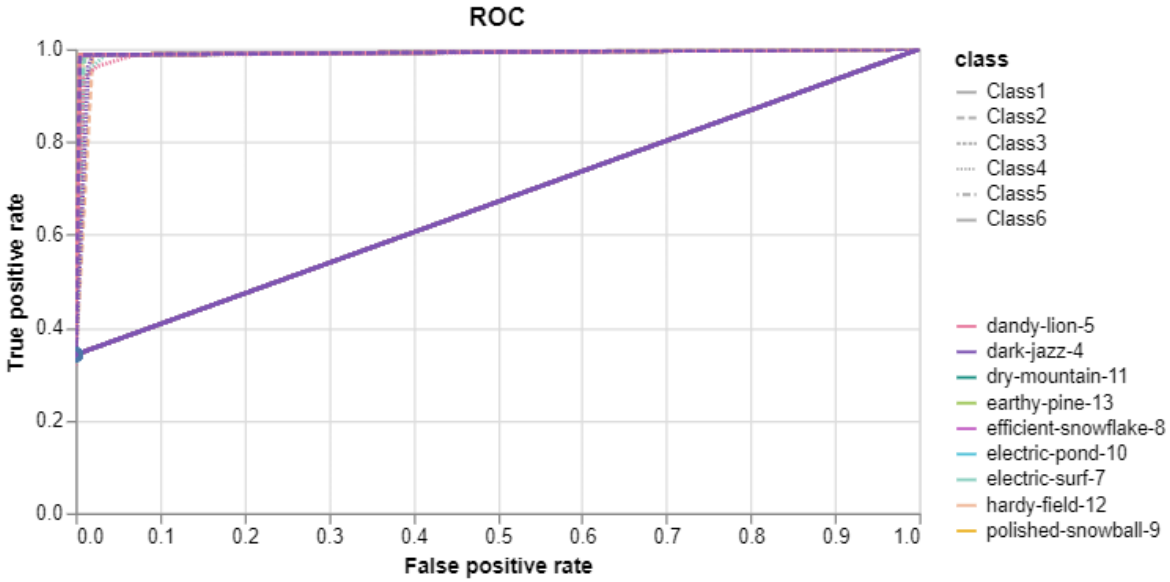


Figura 19: InceptionV3 ROC Curva.

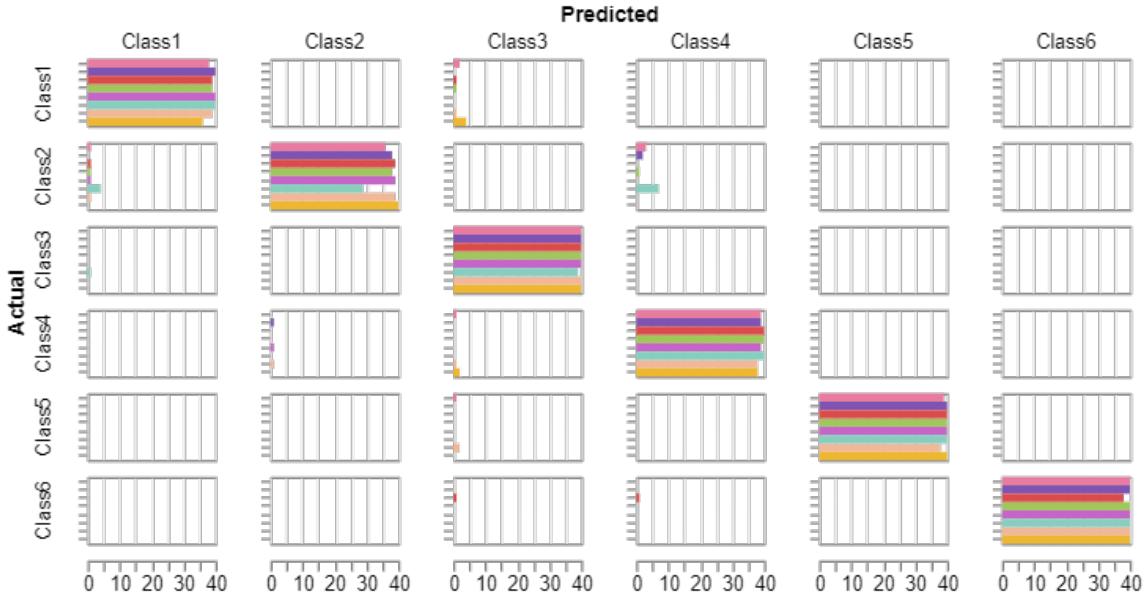


Figura 20: Inception Matriz de confusión de varias ejecuciones.

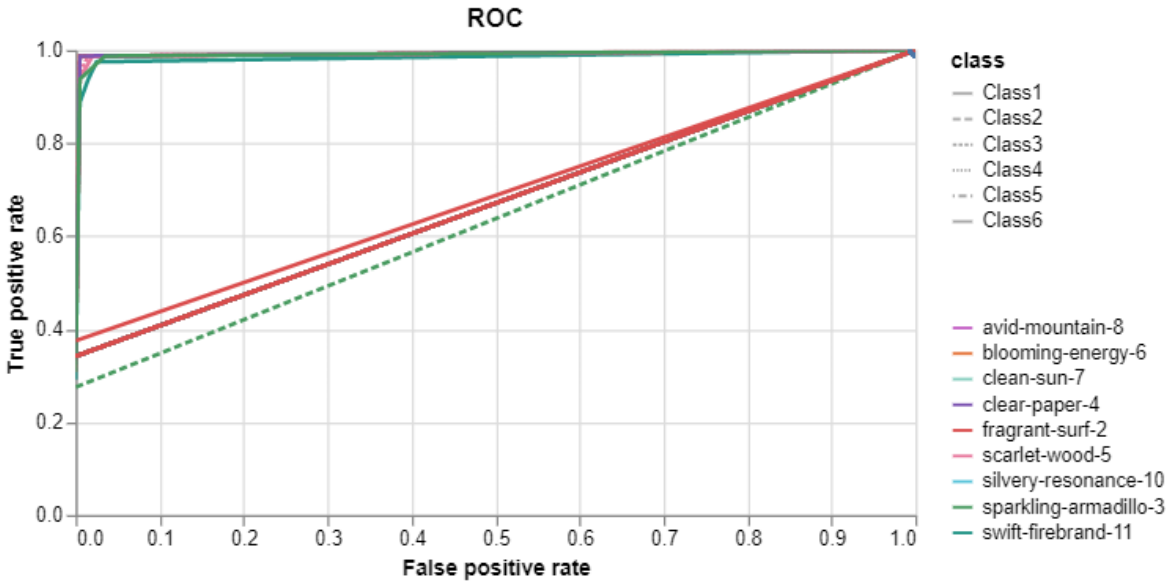


Figura 21: ResNet101V2 ROC Curva.

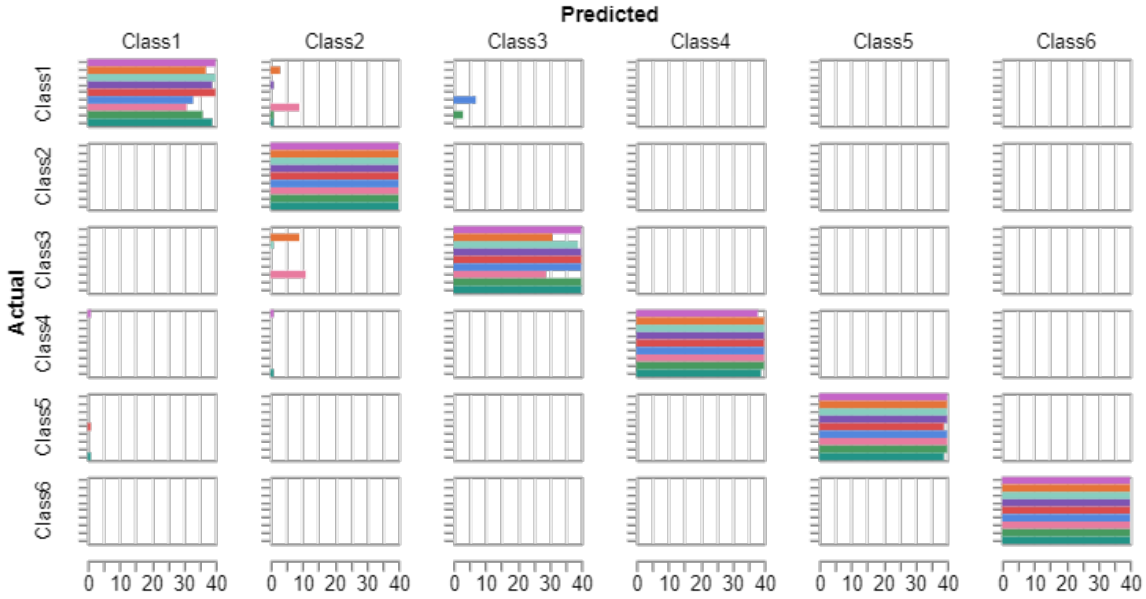


Figura 22: ResNet101V2 Matriz de confusión de varias ejecuciones.

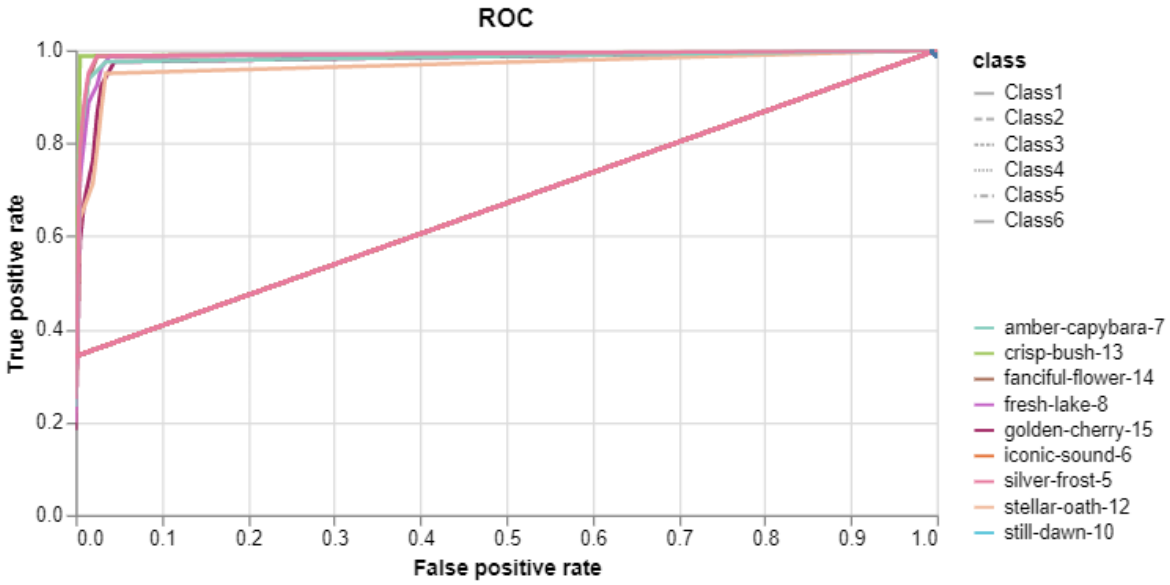


Figura 23: Xception ROC Curva.

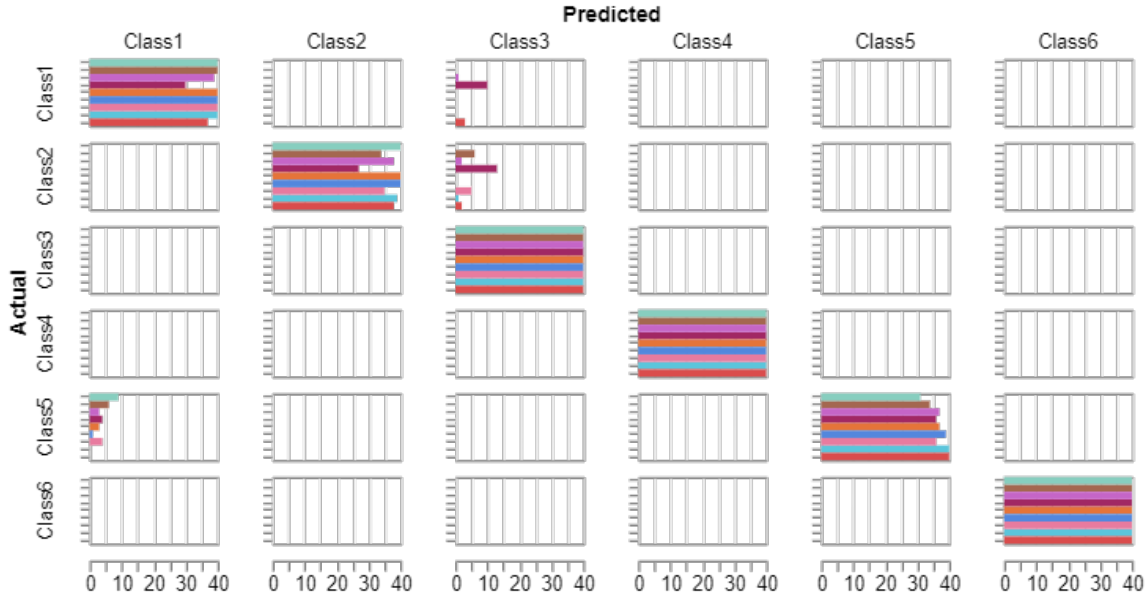


Figura 24: Xception Matriz de confusión de varias ejecuciones..

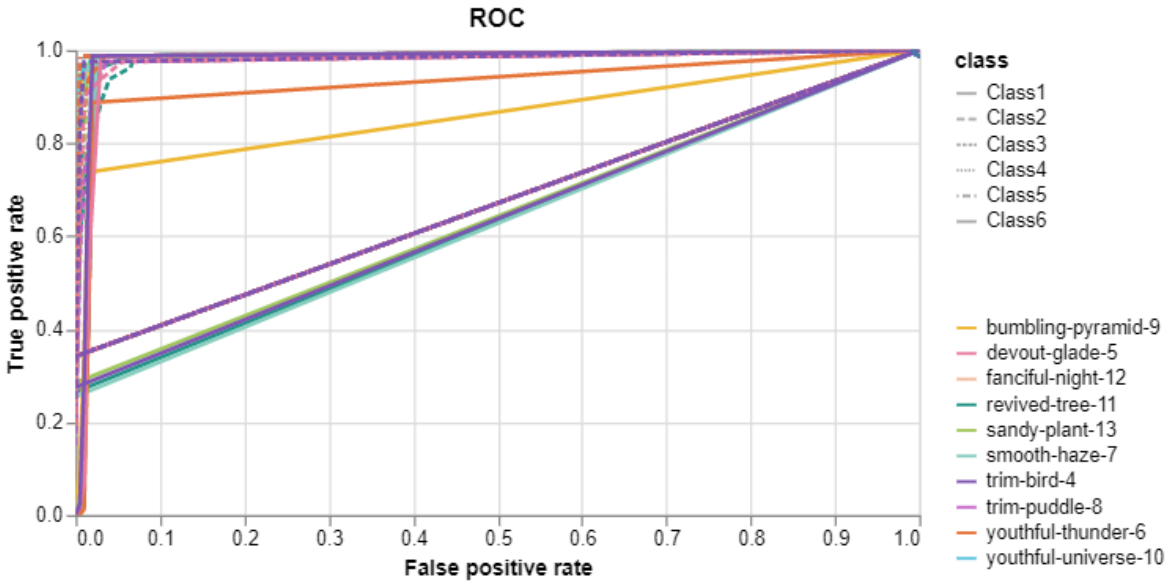


Figura 25: ResNet152V2 ROC Curva.

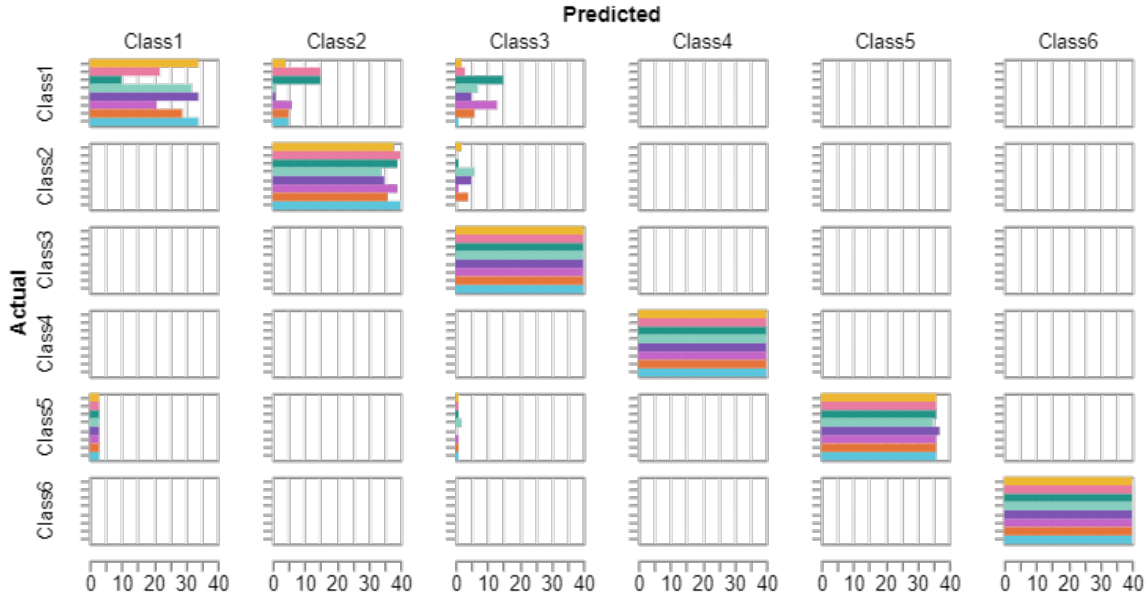


Figura 26: ResNet152V2 Matriz de confusión de varias ejecuciones.

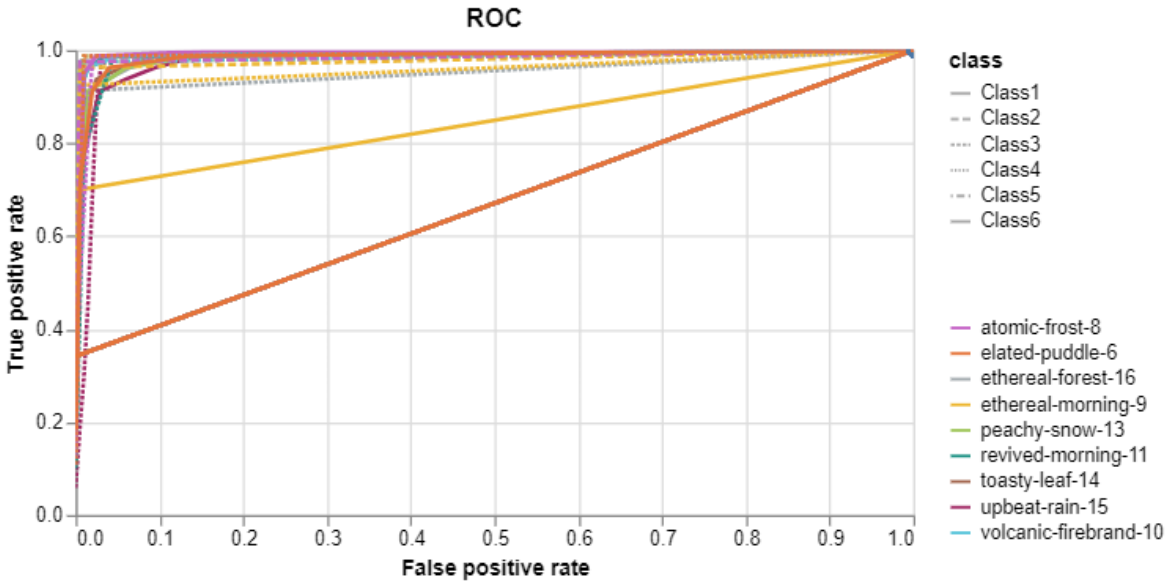


Figura 27: InceptionResnetV2 ROC Curva.

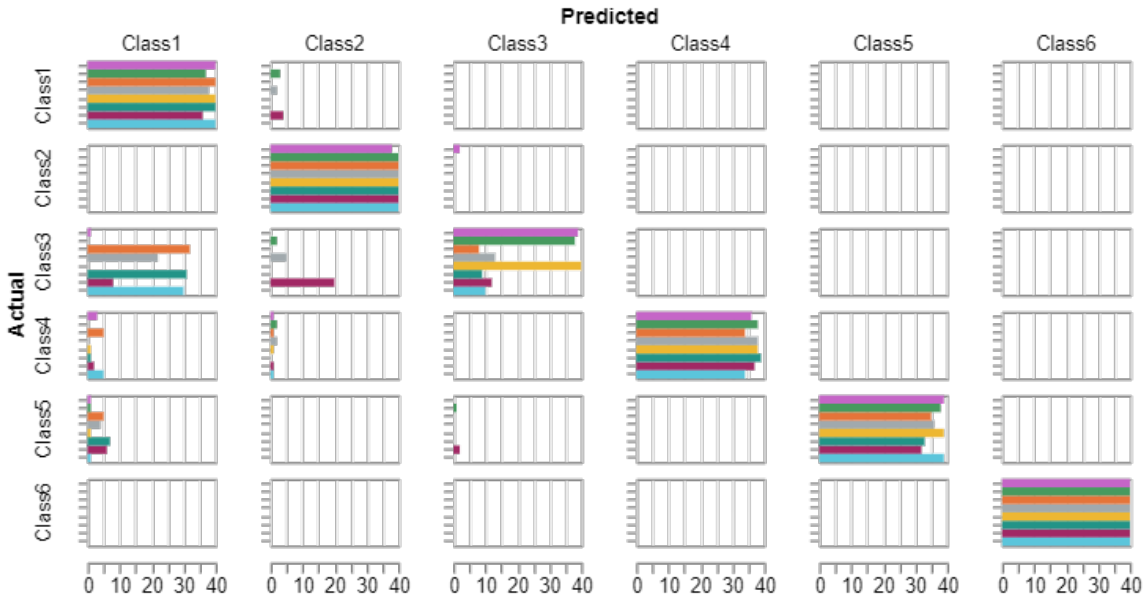


Figura 28: InceptionResnetV2 Matriz de confusión de varias ejecuciones.

IV.2 Tiempo de ejecución del entrenamiento de ML

La fase de entrenamiento utilizó la GPU Tesla K80. La figura 29 muestra el tiempo de ejecución de cada modelo entrenado. Los modelos ResNet fueron los más rápidos de entrenar, con 0.44 horas en la parte inferior, mientras que InceptionV3, Xception e InceptionResNetV2 lograron más de media hora: 0.54, 0.60 y 0.66, respectivamente.

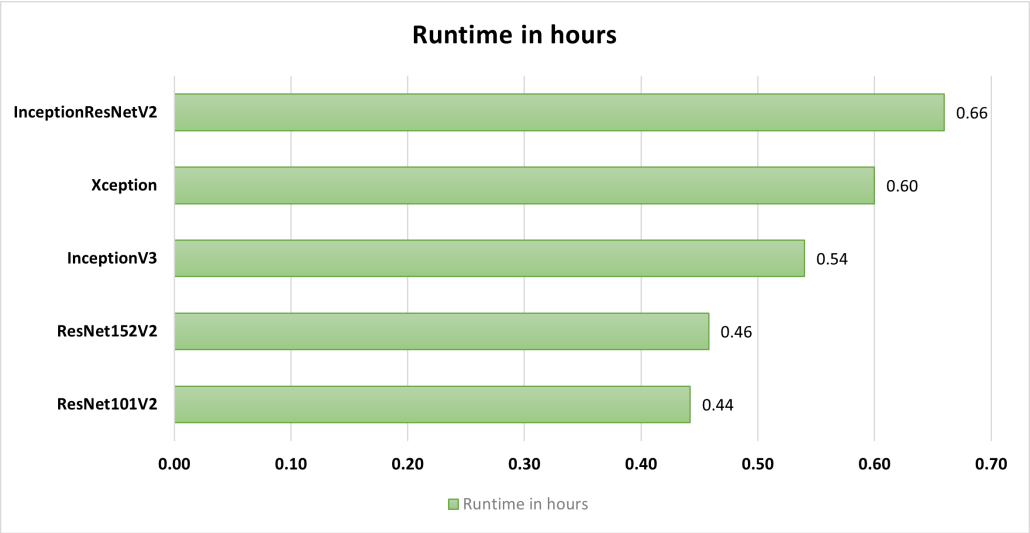
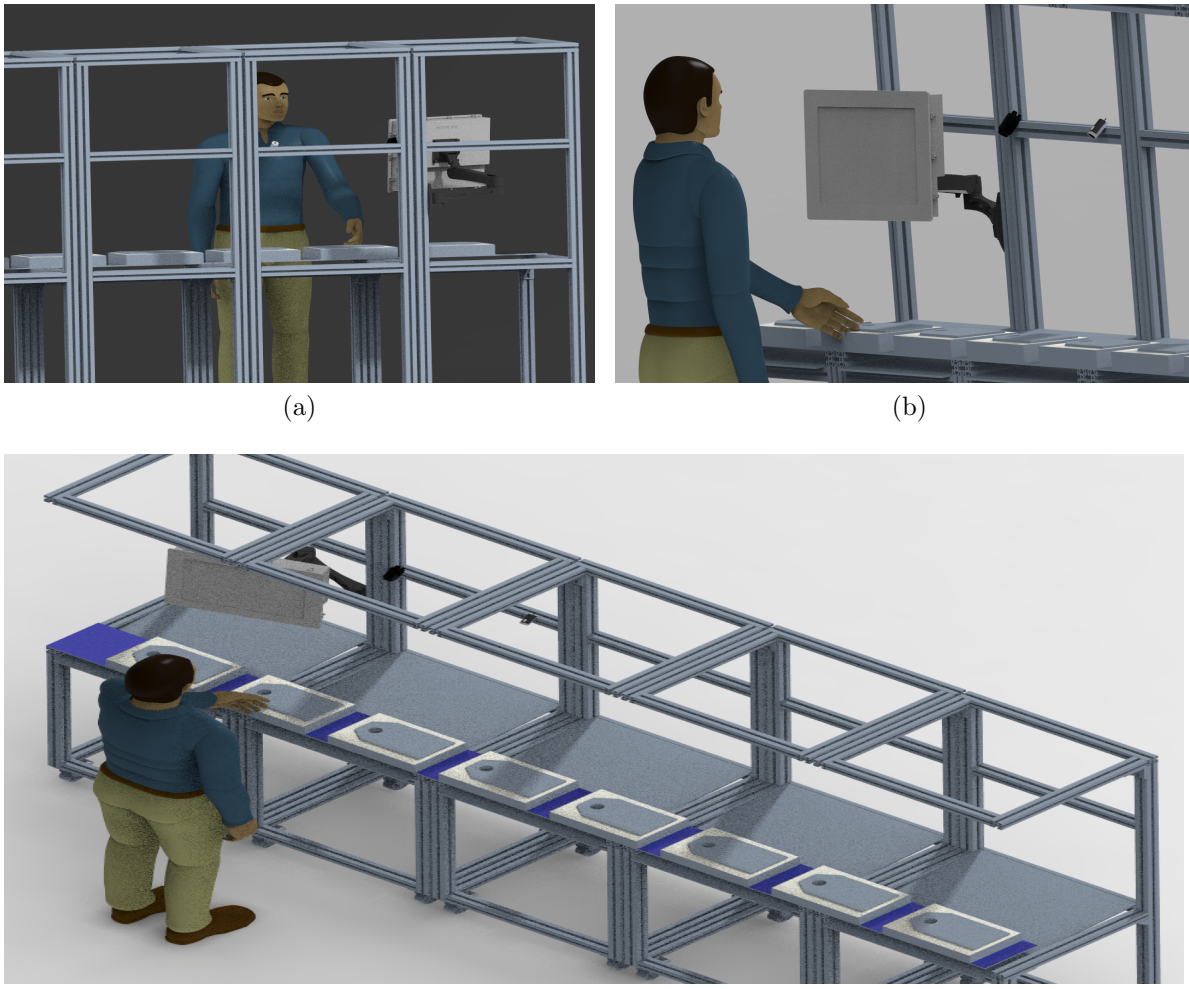


Figura 29: Tiempo de ejecución del entrenamiento con *Google Colaboratory* y la GPU Tesla K80 como hardware.

IV.3 Situación de uso

El objetivo principal fué validar este trabajo dentro de la línea de fabricación actual y la máquina de PC, cargando el modelo en la máquina PC y sirviendo como API para vincular las aplicaciones MES actuales; esta fase se denomina modo de inferencia.



(a) (b)

(c)

Figura 30: Línea de manufactura con *webcam*. (**a**) Vista A, (**b**) Vista B y (**c**) Vista C.

IV.4 Discusiones

Se describió la metodología que se siguió, desde las bases fundamentales de aumentación de datos mediante transformaciones geométricas, donde se pudo lograr mejorar el entrenamiento. Por otro lado, se muestró cómo se utilizan los modelos mediante el uso de la API de Keras, eliminando su ultima capa de salida, para poder conectar la capa objetivo, que en este caso fueron las 6 neuronas de salida para la identificación de la clase perteneciente de los componentes.

Los resultados muestran que cualquiera de los tres modelos principales pueden utilizarse para clasificar componentes de ensamblaje galvanizados durante el proceso de fabricación. Esta técnica de aprendizaje profundo demuestra que los modelos pueden detectar cualquier problema, independientemente de lo elevada que sea la variación del chapado o la luz ambiental. Esto es impresionante en comparación con los sensores de visión clásicos actuales, que requieren una linterna para compensar la falta de esta. Sin embargo, debería considerarse un procedimiento de aprendizaje activo para volver a etiquetar todas las clases de falsos positivos o falsos negativos para generar un mejor conjunto de datos con el que reiniciar la fase de entrenamiento, vea figura 31. Este es uno de los aspectos más importantes del éxito de la metodología del aprendizaje automático.

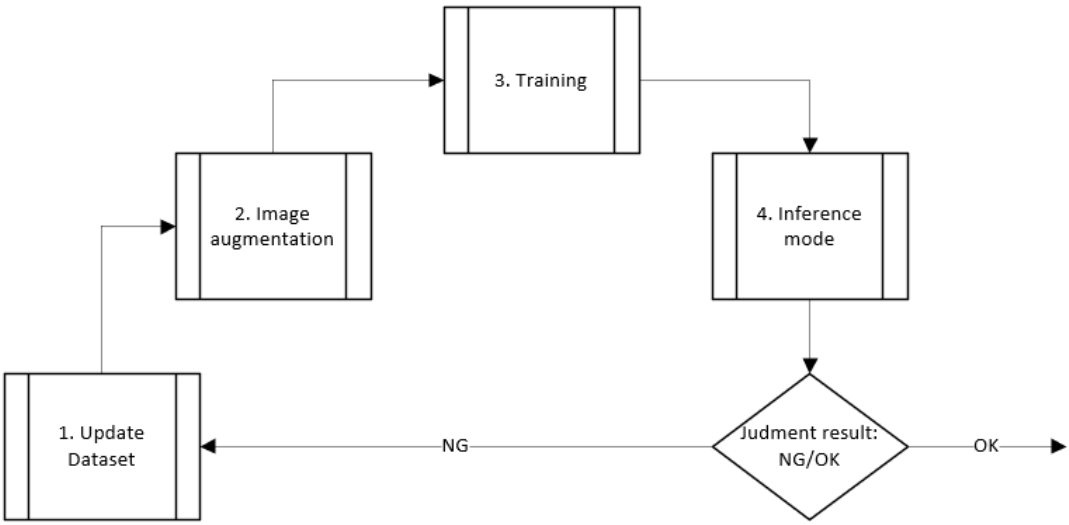


Figura 31: Proceso de aprendizaje activo.

Otro elemento a tener en cuenta es el costo, dado a que en este proyecto sólo se gastaron 20 dólares para resolver un reto difícil, mientras que un sensor de visión estándar cuesta más de 3.000 dólares. Esto representa un ahorro del 0.06 porciento del gasto actual, que puede utilizarse para financiar todos los nuevos controles de calidad

que se implementarán durante el lanzamiento de un nuevo producto o para actualizar los controles de calidad existentes que han quedado obsoletos debido a la Industria 4.0 y a las estrategias de calidad 4.0.

IV.5 Resumen

En este capítulo se mostraron los resultados del entrenamiento de los cinco diferentes modelos, utilizando Google Colaboratory para el entrenamiento, esto por la disponibilidad del recursos de cómputo de alto rendimiento.

Google Colaboratory, tiene GPU disponibles, es por esto que se decide entrenar en esta plataforma.

Por otro lado, se publican resultados por cada época en la página Wandb, debido a que posibilita el uso de diferentes métricas y gráficas de rendimiento, tales como ROC, Accuracy, Precision, Recall, $F1\text{-}Score$ y las métricas de error.

Se muestra también los resultados por modelo, esto con la finalidad de obtener una idea mas clara para entender cual modelo tuvo mejor resultado desde el punto de vista de exactitud, precision y pérdida.

Finalmente, se muestran las gráficas multicorrida, para ver el comportamiento de cada modelo versus curvas ROC y matrices de confusión.

Capítulo V

Conclusiones

V.1 Conclusiones

Sobre la base del desarrollo de éste trabajo de investigación, se concluye que el aprendizaje autómatico puede ser observado como una gran estrategia a emplear en la resolución de problema complejos en la industria, sobre todo, cuando la visión por ordenador tradicional no proporciona la funcionalidad deseada.

Respecto a los modelos de redes neuronales empleados en la metodología de este trabajo, se puede afirmar que todos ellos muestran una capacidad óptima para la clasificación, sin embargo, teniendo en cuenta los resultados arrojados por las métricas, los tres modelos con mejores rendimientos fueron: 1. InceptionV3, 2. ResNet101V2 y 3. Xception; todos ellos alcanzan una precisión y un *recall* mayor al 95%.

Este trabajo es un primer intento de emplear redes neuronales convolucionales para reconocer objetos en procesos de manufactura, quedando demostrado que la visión artificial profunda aplicada a dichos procesos, es una gran área de oportunidad para generar diversas líneas de investigación, como lo fué, en este caso, el reconocimiento de imágenes.

Finalmente, se concluye que el trabajo generado cumple con el objetivo de investigación planteado, dado a que se diseñó y desarrolló un sistema para darle solución, mediante técnicas de aprendizaje profundo, a un problema complejo en un proceso de manufactura, donde los objetos a reconocer eran difícilmente detectables mediante la visión artificial clásica.

V.2 Trabajos futuros

Dadas las limitaciones señaladas en el planteamiento del problema, el presente trabajo sirve de base para futuros proyectos relacionados con la clasificación y reconocimiento de imágenes en procesos de manufactura.

Los tipos de componentes que fueron objeto de estudio, representan un reto mayor cuando existe diversidad en sus acabados, por tal motivo, la técnica de aprendizaje profundo queda como guía metodológica a emplear cuando se presenten componente con similares característica.

A continuación se enlistan los posibles trabajos futuros que pudiesen ser objeto de investigación:

- Publicación de un artículo científico.
- Modelo para detección de acabados en componentes.
- Modelo de detección de condición de mano en componentes de ensamble.
- Sistema autónomo inteligente para ensamble manual mediante uso de atornilladores inteligentes.
- Sistema autónomo inteligente de reconocimiento de acciones de ensamble manual.
- Cobot haciendo uso de modelos de detección de objetos.
- Inferencia mediante uso de servidores de reconocimiento de imagen en la nube.
- Optimización de modelos.
- Inferencia mediante uso de computo de frontera.
- Inferencia mediante uso de computo de neblina.

- Sistema inteligente mediante uso de 3D CNN.

- Implementación de flujo de trabajo para aprendizaje automático profundo.

Bibliografía

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., y Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Ali, A. A., Chramcov, B., Jasek, R., Katta, R., Krayem, S., y Kadi, M. (2021). Detection of steel surface defects using u-net with pre-trained encoder. *Lecture Notes in Networks and Systems*, **232 LNNS**.

Asuncion, A. y Newman, D. J. (2007). Uci machine learning repository: Data sets. *University of California Irvine School of Information*, **2008**.

Bacioiu, D., Melton, G., Papaelias, M., y Shaw, R. (2019). Automated defect classification of aluminium 5083 tig welding using hdr camera and neural networks. *Journal of Manufacturing Processes*, **45**.

Bagherinezhad, H., Horton, M., Rastegari, M., y Farhadi, A. (2018). Label refinery: Improving imagenet classification through label progression. *CoRR*, **abs/1805.02641**.

Biewald, L. (2020a). Experiment Tracking with Weights & Biases. *Software available from wandb. com.*

Biewald, L. (2020b). Experiment tracking with weights and biases. Software available from wandb.com.

Bisong, E. (2019). *Google Colaboratory BT - Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Apress, Berkeley, CA. ISBN 978-1-4842-4470-8.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Brook, Q. (2010). Lean six sigma and minitab. *International Journal of Lean Six Sigma*, **4**.

Carvalho, A. V., Enrique, D. V., Chouchene, A., y Charrua-Santos, F. (2021). Quality 4.0: An Overview. *Procedia Computer Science*, **181**: 341–346.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. En *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, Vol. 2017-Janua.

Dai, W., Li, D., Tang, D., Jiang, Q., Wang, D., Wang, H., y Peng, Y. (2021). Deep learning assisted vision inspection of resistance spot welds. *Journal of Manufacturing Processes*, **62**: 262–274.

Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, **29**(6).

Everingham, M., van Gool, L., Williams, C., Winn, J., y Zisserman, A. (2007). The PASCAL visual object classes challenge 2007 (VOC2007) results.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., y Zisserman, A. (2010). The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, **88**(2).

Ferguson, M., Ak, R., Lee, Y. T. T., y Law, K. H. (2018). Detection and segmentation of manufacturing defects with convolutional neural networks and transfer learning. *Smart and Sustainable Manufacturing Systems*, **2**.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, **1**.

Geiger, A., Lenz, P., Stiller, C., y Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*, **32**(11).

Girshick, R. (2015). Fast R-CNN arXiv1504.08083v2-1. *arXiv*.

Girshick, R., Donahue, J., Darrell, T., y Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. En *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Groover, M. P. (2013). Fundamentals of modern manufacturing material, processes, and systems, 5th edition. *Journal of Chemical Information and Modeling*.

Gupta, A., Dollar, P., y Girshick, R. (2019). Lvis: A dataset for large vocabulary instance segmentation. En *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 2019-June, páginas 5351–5359. ISBN 9781728132938.

H.Bhasin (2019). *Python Basics: A Self-Teaching Introduction*. Mercury Learning and Information.

He, K., Zhang, X., Ren, S., y Sun, J. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **37**(9).

He, K., Zhang, X., Ren, S., y Sun, J. (2016a). Identity mappings in deep residual networks. En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 9908 LNCS.

He, K., Zhang, X., Ren, S., y Sun, J. (2016b). Identity mappings in deep residual networks. En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 9908 LNCS.

He, K., Gkioxari, G., Dollar, P., y Girshick, R. (2020). Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **42**(2): 386–397.

Kalyani, Y. y Collier, R. (2021). A systematic survey on the role of cloud, fog, and edge computing combination in smart agriculture.

Kristoff, S. (2017). *Zinc Plating Process*. Sciencing.

Kritzinger, D. (2017). 4 - Fault tree analysis. En D. Kritzinger, editor, *Aircraft System Safety*, páginas 59–99. Woodhead Publishing. ISBN 978-0-08-100889-8.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., y Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 8693 LNCS.

Liu, T., Fang, S., Zhao, Y., Wang, P., y Zhang, J. (2015). Implementation of training convolutional neural networks. *CoRR*, **abs/1506.01195**.

Moreno-Barea, F., Strazzera, F., Jerez, J., Urda, D., y Franco, L. (2018). Forward noise adjustment scheme for data augmentation.

Mountcastle, V. B. (1957). Modality and topographic properties of single neurons of cat's somatic sensory cortex. *Journal of neurophysiology*, **20**.

Neapolitan, R. E. y Jiang, X. (2019). *Artificial Intelligence With an Introduction to Machine Learning*, Vol. 53. CRC Press Taylor & Francis Group.

Papert, S. (1966). The summer vision project. páginas 1–6.

Pearson, K. (1904). On the thoery of contingency and its relation to association and normal correlation. `http://ia800207.us.archive.org/16/items/cu31924003064833/cu31924003064833.pdf`.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., y Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, **12**: 2825–2830.

Pyzdek, T. y Keller, P. (2010). *The Six Sigma handbook: a complete guide for green belts, black belts, and managers at all levels*.

Pérez, F. y Granger, B. E. (2007). Ipython : A system for. *IEEE Journals & Magazines*, **9**.

Quintana, A. F. L., Herrera, D. A. M., y Salazar, L. E. M. (2012). Sistema de visión artificial para conteo de objetos en movimiento. *El hombre y la máquina*, páginas 87–101.

Redmon, J. y Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *Computer Science ¿ Computer Vision and Pattern Recognition*.

Ren, S., He, K., Girshick, R., y Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, **65**.

Rumelhart, D. E., Hinton, G. E., y Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, **323**.

Sharma, V. y Mir, R. N. (2020). A comprehensive and systematic look up into deep learning based object detection techniques: A review. *Computer Science Review*, **38**: 100301.

Shorten, C. y Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*.

Singh, P. (2019). *Machine Learning with PySpark*.

Sterling, T., Anderson, M., y Brodowicz, M. (2017). *High performance computing: Modern systems and practices*. Morgan Kaufmann.

Swift, K. y Booker, J. (2013). Chapter 10 - assembly systems. En K. Swift y J. Booker, editores, *Manufacturing Process Selection Handbook*, páginas 281–289. Butterworth-Heinemann, Oxford. ISBN 978-0-08-099360-7.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., y Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. En *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 2016-Decem.

Szegedy, C., Ioffe, S., Vanhoucke, V., y Alemi, A. A. (2017). Inception-v4, inception-ResNet and the impact of residual connections on learning. En *31st AAAI Conference on Artificial Intelligence, AAAI 2017*.

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., y Liu, C. (2018). A survey on deep transfer learning. En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 11141 LNCS.

Tantrapiwat, A. (2021). Spot welding defect detection using synthetic image dataset on convolutional neural networks. *2021 7th International Conference on Engineering, Applied Sciences and Technology, ICEAST 2021 - Proceedings*.

van Rossum, G. y Drake, F. L. (2009). *Python 3 Reference Manual.*. nature.

Vega, C. S. (2018). *¿Qué es una Red Neuronal? Parte 1 : La Neurona*.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., y SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, **17**: 261–272.

Voelkel, J. G. y Ishikawa, K. (1989). Guide to Quality Control. *Technometrics*, **31**(2).

Walt, S. V. D., Colbert, S. C., y Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science and Engineering*, **13**.

Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., y Jue, J. P. (2019). All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, **98**: 289–330.

Yun, J. P., Shin, W. C., Koo, G., Kim, M. S., Lee, C., y Lee, S. J. (2020). Automated defect inspection system for metal surfaces based on deep learning and data augmentation. *Journal of Manufacturing Systems*, **55**: 317–324.

Zamora-Hernández, M.-A., Castro-Vargas, J. A., Azorin-Lopez, J., y Garcia-Rodriguez, J. (2021). Deep learning-based visual control assistant for assembly in Industry 4.0. *Computers in Industry*, **131**: 103485.

Zhu, P., Wen, L., Du, D., Bian, X., Fan, H., Hu, Q., y Ling, H. (2021). Detection and Tracking Meet Drones Challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Ľubica Simanová, Sujová, A., y Gejdoš, P. (2019). Improving the performance and quality of processes by applying and implementing six sigma methodology in furniture manufacturing process. *Drvna Industrija*, **70**.

Apéndice A

Publicaciones derivadas del trabajo de tesis

Artículo a someter en JCR

E. Ramos-Acosta, O. R. López-Bonilla, E. E. García-Guerrero, E. Tlelo-Cuautle, D. López-Mancilla, C. Hernández-Mejía, E. Inzunza-González. 2022. *Assembly zinc plated components detection system based on convolutional neural networks: A case study.* MDPI `https://www.overleaf.com/6326798283ttffrsgxxqfj`

Presentación en Congreso Internacional

E. Ramos-Acosta, O. R. López-Bonilla, E. E. García-Guerrero, E. Tlelo-Cuautle, D. López-Mancilla, C. Hernández-Mejía, E. Inzunza-González, 2021. *Assembly zinc plated components detection system based on convolutional neural networks: A case study*, 9$^{\text{th}}$ International Workshop on Numerical and Evolutionary Optimization.

Presentaciones digitales colaborativas

E. R. Ramos, E. E. Contreras Lujan,A. Navarro-Espinoza, E. Inzunza-González, E. E. García-Guerrero, O. R. López-Bonilla, 2021. *Machine learning for kids.* XXVIII Jornadas de Ingeniería, Arquitectura y Diseño. `https://youtu.be/JkNhe0fkGtI`

E. E. Contreras Lujan, A. Navarro-Espinoza, **E. R. Ramos Acosta**, E. Inzunza-González, E. E. García-Guerrero, O. R. López-Bonilla, 2021. *Machine learning aplicado*

al diagnóstico de enfermedades. XXVIII Jornadas de Ingeniería, Arquitectura y Diseño. https://www.youtube.com/watch?v=lzkowEpeYis

E. R. Ramos Acosta, A. Navarro-Espinoza, E. E. Contreras Lujan, E. Inzunza-González, E. E. García-Guerrero, O. R. López-Bonilla, 2021. *Aprendizaje Profundo Aplicado en la Industria.* XXVIII Jornadas de Ingeniería, Arquitectura y Diseño. https://www.youtube.com/watch?v=zdOKr_dT3e8

Apéndice A

Código fuente

A continuación se muestra el código fuente utilizado para poder generar el conjunto de datos, aumentación de datos, definición de métricas a utilizar asi como el uso de aprendizaje transferido:

```
/* USER CODE BEGIN 2 */
#Se instala wandb, para poder subir las metricas durante el
   entrenamiento y validaci\'on
!pip install wandb
!pip install tensorflow-addons==0.8.3

#Importaci\'on de librear\'ias
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import matplotlib.text as mpl_text
import datetime as dt

from sklearn.metrics import classification_report,confusion_matrix,
   accuracy_score, f1_score, precision_score, recall_score,
   hamming_loss, zero_one_loss, cohen_kappa_score
from sklearn.model_selection import train_test_split

from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Flatten, Conv2D, Dense,
   MaxPooling2D, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras.applications import NASNetLarge,
   InceptionResNetV2, Xception, ResNet152V2, InceptionV3, DenseNet201,
    ResNet101V2, ResNet152, ResNet101, DenseNet169, MobileNetV2
import tensorflow_addons as tfa

# Import W&B
import wandb
from wandb.keras import WandbCallback

#1. Dataset Loading
from google.colab import drive
drive._mount('/content/drive')
cd '/content/drive/MyDrive'
```

```
#%%
TRAINING_DATA_DIR = 'sample_data/Baseplates/Training' #'sample_data/
    trainingImages_original'
#'sample_data/Baseplates/Training'#
VALIDATION_DATA_DIR='sample_data/Baseplates/Validation'
TESTING_DATA_DIR= 'sample_data/Baseplates/Test'

#1.2 Image augmentation
datagen_kwargs = dict(rescale=1./255,rotation_range=40,
        shear_range=0.2,
        zoom_range=0.2,
        width_shift_range=0.2,
        brightness_range=[0,1],
        height_shift_range=0.2,
        horizontal_flip=True,validation_split=0.2)

##1.2.1 Images for training:
def Create_Images_for_training(shape_size_W,shape_size_H):

  IMAGE_SHAPE=(shape_size_W, shape_size_H)
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(**
   datagen_kwargs)
  train_generator = train_datagen.flow_from_directory(
  TRAINING_DATA_DIR,
  batch_size=32,
  subset="training",
  seed=42,
  shuffle=True,
  target_size=IMAGE_SHAPE)
  for image_batch, label_batch in train_generator:
    break
  print(image_batch.shape, label_batch.shape)
  image_batch_train, label_batch_train = next(iter(train_generator))
  valid_datagen = tf.keras.preprocessing.image.ImageDataGenerator(**
   datagen_kwargs)
  valid_generator = valid_datagen.flow_from_directory(
  TRAINING_DATA_DIR,
  subset="validation",
  shuffle=False,
  target_size=IMAGE_SHAPE
  )
  val_image_batch, val_label_batch = next(iter(valid_generator))
  print("Image batch shape: ", image_batch_train.shape)
  print("Label batch shape: ", label_batch_train.shape)
  dataset_labels = sorted(train_generator.class_indices.items(), key=
   lambda pair:pair[1])
  dataset_labels = np.array([key.title() for key, value in
   dataset_labels])
  print(dataset_labels)

  print(label_batch[0:5])
```

```
  print(train_generator.class_indices)
  labels = '\n'.join(sorted(train_generator.class_indices.keys()))
  with open('labels.txt', 'w') as f:
    f.write(labels)
  return train_generator,valid_generator,dataset_labels

  train_generator,valid_generator,dataset_labels=
   Create_Images_for_training(299,299)

class_names = dataset_labels
image_batch, label_batch = next(iter(train_generator))

plt.figure(figsize=(40, 40))
for i in range(32):
  ax = plt.subplot(4, 8, i + 1)
  plt.imshow(np.array(image_batch[i]))
  label = np.argmax(label_batch[i])
  plt.title(class_names[label])
  plt.axis("off")

  #2 Metrics Definition
  METRICS = [
  tf.keras.metrics.TruePositives(name='tp'),
  tf.keras.metrics.FalsePositives(name='fp'),
  tf.keras.metrics.TrueNegatives(name='tn'),
  tf.keras.metrics.FalseNegatives(name='fn'),
  tf.keras.metrics.Precision(name='precision'),
  tf.keras.metrics.Recall(name='recall'),
  tf.keras.metrics.CategoricalAccuracy(name='acc'),
  tf.keras.metrics.AUC(name='auc'),
  tfa.metrics.CohenKappa(name='cohen_kappa',num_classes=6),
  tfa.metrics.HammingLoss(name='hamming_loss',mode='multiclass'),
  tf.keras.metrics.SpecificityAtSensitivity(0.5),
  tf.keras.metrics.SensitivityAtSpecificity(0.5),
]

##3.1 Wandb Initialization

wandb.init(project="Baseplates_Convnet", entity="computervision",
config = {
  "learning_rate": 0.001,
  "epochs": 30
})
config = wandb.config

##3.2 Neural Network Design, en caso de hacer su propia red neuronal
   convolucional:

steps_per_epoch = np.ceil(train_generator.samples/train_generator.
   batch_size)
val_steps_per_epoch = np.ceil(valid_generator.samples/valid_generator.
   batch_size)
```

```
model = Sequential([

  Conv2D(16, (3, 3), activation='relu',input_shape=(299, 299,3)),
  MaxPooling2D(),
  Conv2D(32, (3, 3), padding='same', activation='relu'),
  MaxPooling2D(),
  Conv2D(64,(3,3),padding='same',activation='relu'),
  MaxPooling2D(),
  Conv2D(128,(3,3),padding='same',activation='relu'),
  Flatten(),
  Dense(256,activation='relu'),
  Dropout(0.2),
  Dense(6, activation='softmax')

 ])

 ##3.3 Optimization and Compilation

 optimizer=keras.optimizers.Adam(lr=config.learning_rate)

model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=METRICS)

## 3.4 Training
hist = model.fit(
        train_generator,
        epochs=config.epochs,
        verbose=1,
        steps_per_epoch=steps_per_epoch,
        validation_data=valid_generator,
        validation_steps=val_steps_per_epoch,callbacks=[WandbCallback
   ()]).history

  model.save('Model_baseplates'+'_'+'ConvNet'+str(dt.datetime.now())+'
   .h5')

#5 Results

##5.1 Test images preparation

def testGen(width,height):
  val_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
   rescale=1.0/255.0)
  validation_data_dir=VALIDATION_DATA_DIR
  validationTest_generator = val_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(width,height),
        batch_size=32,
        class_mode='categorical',
        shuffle=False)
```

```
 return validationTest_generator

##5.2 Report preparation
def Classification_report(trained_model, W, H):

 #Confution Matrix and Classification Report
 Images4ValidationGenerator=testGen(W,H)
 class_labels = Images4ValidationGenerator.class_indices
 class_labels = {v: k for k, v in class_labels.items()}
 classes = list(class_labels.values())
 y_true = Images4ValidationGenerator.classes
 Y_pred = trained_model.predict(Images4ValidationGenerator)
 y_pred = np.argmax(Y_pred, axis=1)

 conf_mat = confusion_matrix(Images4ValidationGenerator.classes,
  y_pred)
 conf_mat_normalized = conf_mat.astype('float') / conf_mat.sum(axis
  =1)[:, np.newaxis]
 plt.figure(figsize = (10,10))
 sns.heatmap(conf_mat, annot=True,fmt='d',cmap="Blues")
 plt.ylabel('True label')
 plt.xlabel('Predicted label')

 acc = accuracy_score(y_true, y_pred)
 f1 = f1_score(y_true, y_pred, average='macro')
 prec = precision_score(y_true, y_pred, average='macro')
 rec = recall_score(y_true, y_pred, average='macro')
 hamming = hamming_loss(y_true, y_pred)
 zero = zero_one_loss(y_true, y_pred)
 cohen = cohen_kappa_score(y_true, y_pred)


 TruePositive = np.diag(conf_mat)
 FalsePositive = []
 FalseNegative = []
 TrueNegative = []

 for i in range(6):
     FalsePositive.append(sum(conf_mat[:,i]) - conf_mat[i,i])
     FalseNegative.append(sum(conf_mat[i,:]) - conf_mat[i,i])
     temp = np.delete(conf_mat, i, 0)   # delete ith row
     temp = np.delete(temp, i, 1)  # delete ith column
     TrueNegative.append(sum(sum(temp)))

 TP = sum(TruePositive)
 FP = sum(FalsePositive)
 FN = sum(FalseNegative)
 TN = sum(TrueNegative)

 wandb.log({"conf_mat" : wandb.plot.confusion_matrix(probs=None,
                        y_true=Images4ValidationGenerator.classes,
  preds=y_pred,
```

```
                        class_names=classes),
              "accuracy_score": acc,
              "f1_score": f1,
              "precision_score": prec,
              "recall_score": rec,
              "hamming_loss": hamming,
              "zero_one_loss": zero,
              "cohen_kappa": cohen,
              "TruePositive": TruePositive,
               "*TP": TP,
               "FalsePositive": FalsePositive,
               "*FP": FP,
               "TrueNegative": TrueNegative,
               "*TN" : TN,
               "FalseNegative": FalseNegative,
               "*FN": FN})

  print(acc)
  print(f1)
  print(prec)
  print(rec)
  print(hamming)
  print(zero)
  print(cohen)
  print(TruePositive)
  print(TP)
  print(FalsePositive)
  print(FP)
  print(TrueNegative)
  print(TN)
  print(FalseNegative)
  print(FN)

 Classification_report(model, 299, 299)


#Para transfer learning:
#6 Transfer learning
##6.1 Transfer Learning Models definition

def train_model(TL_Model,w,h):
  wandb.init(project=TL_Model.__name__, entity="computervision",
  config = {
    "learning_rate": 0.001,
    "epochs": 30,
  })
  config = wandb.config

  train_generator,valid_generator,dataset_labels=
   Create_Images_for_training(w,h)

  pre_trained_model=TL_Model(input_shape=(w,h,3),
```

```
                                include_top=False,
                                weights='imagenet')
  for layer in pre_trained_model.layers:
    layer.trainable = False

  model = tf.keras.Sequential([pre_trained_model,
                                   tf.keras.layers.
   GlobalAveragePooling2D(),
                                   tf.keras.layers.Dropout(0.2),
                                   tf.keras.layers.Dense(6, activation="
   softmax")
                                  ])


  optimizer=keras.optimizers.Adam(learning_rate=config.learning_rate)

  model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=METRICS)

  steps_per_epoch = np.ceil(train_generator.samples/train_generator.
   batch_size)
  val_steps_per_epoch = np.ceil(valid_generator.samples/
   valid_generator.batch_size)

  history = model.fit(train_generator,
         epochs=config.epochs,
         verbose=1,
         steps_per_epoch=steps_per_epoch,
         validation_data=valid_generator,
         validation_steps=val_steps_per_epoch,callbacks=[WandbCallback
   ()])
  model.save('Master_Baseplates'+'_'+TL_Model.__name__+str(dt.datetime
   .now())+'.h5')
  wandb.save('Master_Baseplates' + '_'+ TL_Model.__name__+str(dt.
   datetime.now())+'.h5')

  wandb.finish()
  return history,model

 #6.2 Training Stage
 ##6.2.1 ResNet152V2

 model_history,model=train_model(ResNet152V2,224,224)
##6.2.2 InceptionV3
model_history,model=train_model(InceptionV3,299,299)
##6.2.3 InceptionV3
model_history,model=train_model(ResNet101V2,224,224)
##6.2.4 InceptionV3
model_history,model=train_model(Xception,299,299)
##6.2.5 InceptionV3
model_history,model=train_model(InceptionResNetV2,299,299)
```

```
/* USER CODE END 2 */
```

El conjunto de modelos generados por cada corrida, son guardados en la plataforma wandb.ai, asi como el mejor modelo por corrida.