UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL

TÍTULO DE INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

TEMA

"DISEÑO DE UN SISTEMA DE RECONOCIMIENTO AUTOMÁTICO DE VEHÍCULOS MEDIANTE EL USO DE REDES NEURONALES PROFUNDAS (DNN)"

AUTOR: CYNTHIA PAMELA PINEDA PASQUEL

DIRECTOR: MSC. VLADIMIR ISRAEL GARCÍA SANTOS

IBARRA - ECUADOR

2018

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art.144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DEL CONTACTO			
CÉDULA DE IDENTIDAD:	1004424956		
APELLIDOS Y NOMBRES:	Pineda Pasquel Cynthia Pamela		
DIRECCIÓN:	Vicente Rocafuerte e Inés Hernández		
E-MAIL:	cppineda@utn.edu.ec		
TELÉFONO FIJO:	062-511-711	**TELÉFONO MÓVIL:**	0987601250
DATOS DE LA OBRA			
TÍTULO:	DISEÑO DE UN SISTEMA DE RECONOCIMIENTO AUTOMÁTICO DE VEHÍCULOS MEDIANTE EL USO DE REDES NEURONALES PROFUNDAS (DNN)		
AUTOR:	Pineda Pasquel Cynthia Pamela		
FECHA: DD/MM/AAAA	12/02/2019		
PROGRAMA:	■ PREGRADO ☐ POSGRADO		
TÍTULO POR EL QUE OPTA:	Ingeniera en Electrónica y Redes de Comunicación		
ASESOR/DIRECTOR:	MSc. Vladimir García		

CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra,

EL AUTOR:

.....

Pineda Pasquel Cynthia Pamela

CI: 1004424956

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

MAGISTER VLADIMIR ISRAEL GARCÍA SANTOS, DIRECTOR DEL PRESENTE TRABAJO DE TITULACIÓN CERTIFICA:

Que, el presente trabajo de Titulación "DISEÑO DE UN SISTEMA DE RECONOCIMIENTO AUTOMÁTICO DE VEHÍCULOS MEDIANTE EL USO DE REDES NEURONALES PROFUNDAS (DNN)". Ha sido desarrollado por la señorita CYNTHIA PAMELA PINEDA PASQUEL bajo mi supervisión.

Es todo cuánto puedo certificar en honor a la verdad.

Ing. Vladimir Israel García Santos M.Sc.

060402526-2

DIRECTOR

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

DEDICATORIA

Dedicado con todo mi amor y cariño a mi Madre, por todo lo que sembró en mí y por ser mi fuente de motivación, es el espejo en el que me quiero reflejar por su cariño, humildad y gran corazón. Tu recuerdo palpita en mí.

A mi padre y a mis hermanos con inmenso amor.

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

AGRADECIMIENTO

Quiero expresar mi agradecimiento a cada una de las personas que hicieron posible este trabajo:

A Dios por darme la fuerza, sabiduría y fe para salir adelante a pesar de las circunstancias que se han presentado en mi vida, gracias a él tengo la oportunidad de que cada mañana puedo empezar de nuevo y despertar no solo con vida sino también con salud y empeño.

A mis padres por haberme forjado como la persona que soy en la actualidad, y a mis hermanos por el amor, la dedicación y la paciencia con la que cada día se preocupaban por mi avance y desarrollo, por confiar y creer en mi en todo momento.

A mis tutores, PhD. Lorena de los Ángeles Guachi y Msc. Vladimir García. Por darme la oportunidad de cumplir mi objetivo mediante su enseñanza, confianza, apoyo y consejo, ha sido un privilegio poder contar con su guía y ayuda.

.

RESUMEN

El presente trabajo es un estudio sobre detección de objetos con el diseño de un sistema de reconocimiento automático de vehículos mediante el uso del aprendizaje profundo para facilitar el desarrollo de una aplicación de seguridad vehicular.

El sistema propuesto en este trabajo se basa en un detector de objetos desarrollado con técnicas de aprendizaje supervisado y planteado con redes neuronales artificiales convolucionales. Haciendo uso de la plataforma Caffe y sus librerías, así como también de librerías en Opencv y scripts desarrollados en Python.

En este trabajo se estudian también algoritmos para el reconocimiento automático de objetos, necesarios para comprender su proceso y su amplia gama de implementación en distintas aplicaciones.

Este documento demuestra el estudio realizado para diseñar una arquitectura de red y seleccionar los conjuntos de datos destinados tanto para la fase de entrenamiento como de validación, además presenta la correcta preparación de los scripts para el entrenamiento de la red.

ABSTRACT

The present work is a study about object detection with the design of an automatic vehicle recognition system through the use of deep learning to facilitate the development of a vehicular safety application.

The system proposed in this work is based on an object detector developed with supervised learning techniques and it is posed with convolutional artificial neural networks. Making use of the Caffe platform and its libraries, as well as Opencv libraries and scripts developed in Python.

In this work algorithms for the automatic recognition of objects are study too, necessary to understand its process and its wide range of implementation in different applications.

This document demonstrates the study carried out to design a network architecture and select the data sets intended for both the training and validation phase, it also present the correct preparation of the scripts for network training.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

CAPÍTULO I: ANTECEDENTES

El capítulo I presenta una breve introducción al lector sobre las razones que impulsaron al desarrollo de este proyecto. A continuación, se presenta la problemática, los objetivos planteados tanto general como específicos, así como también el alcance y la justificación que fundamenta este proyecto.

1.1. Problema

OMS (2017) describe que cada año mueren cerca de 1,3 millones de personas en las carreteras del mundo entero. Con el incremento de adquisición de vehículos, el tráfico en las carreteras ha aumentado considerablemente, según el Telégrafo, en el año 2017 se comercializó 3.978 unidades adicionales, en comparación con el año 2016.

Las causas de accidentes de tránsito pueden ser varias, en las que influyen distintos factores, en su mayoría pueden ser controladas con facilidad. INEC (2017) afirma que, en el Ecuador, los accidentes de tránsito se presentan en un elevado número, especialmente en las carreteras interprovinciales. Siendo la cuarta causa de muerte en el País y la principal causa de siniestros de transito es la perdida de atención al conducir, en el mes de diciembre del 2018, el 24,13% de accidentes era por la causa mencionada y en enero del 2019 la cifra aumenta a 29,2% (Estadísticas de transporte terrestre y seguridad vial, 2017).

Un conductor, con el pasar del tiempo, cree adquirir experiencia en conducir. Sin embargo, la complejidad y riesgo de conducir incrementa mientras un conductor guía su vehículo, debido a que, a la par realiza diversas actividades que también requieren atención. Entre estas actividades se encuentra conversar con el pasajero, utilizar el teléfono celular, buscar un objeto dentro del vehículo, escuchar o manipular la radio y comer o beber (Castillo & Herrera, 2013) (Montes, 2016).

La alarmante cifra de accidentes que acontecen en las carreteras debido muchos de ellos por la distracción de un conductor afectan a la sociedad en gran medida; por lo que puede cobrar la vida de varias personas, costos crecientes en años y calidad de vida como en términos socioeconómicos. Con los antecedentes antes mencionados, se puede notar que la distracción es un problema presente en la sociedad y en nuestro diario vivir, el cual no ha sido atendido ni tomado conciencia de la mejor manera por la población.

En los últimos años, la industria automotriz ha realizado importantes esfuerzos para alcanzar el objetivo de cero muertes por accidentes automovilísticos mediante la introducción de mecanismos de seguridad vial, como es el caso del Sistema Avanzado de Asistencia a la Conducción (ADAS), el cual ha sido concebido como un asistente seguro de conducción. El funcionamiento de un sistema ADAS se conforma por varios sistemas de sensores y dispositivos que permiten seguridad al conductor. El sistema de mitigación de impactos y el sistema de control de crucero adaptativa, son sistemas fundamentales que forman parte de un ADAS; en los cuales una de las tareas principales consiste en identificar los vehículos que se aproximan o que se encuentran delante del vehículo conducido.

En base a esto y considerando la principal causa de siniestros de tránsito se plantea la creación de un sistema de reconocimiento automático de vehículos, el cual detectará la presencia de vehículos en ambientes controlados. El sistema será entrenado y testeado con conjuntos de datos existentes (dataset de imágenes) con fines de investigación, con el objetivo de proporcionar una verificación funcional y asequible en ambientes de laboratorio. La adaptabilidad a ambientes de vías y carreteras ecuatorianas se logrará proporcionando los respectivos conjuntos de datos en la fase de prueba adaptativo de la red. Es importante notar que el sistema propuesto ayudará a minimizar el riesgo de colisionar con otros vehículos y beneficiará la calidad de vida de la sociedad que se encuentra tras el volante, sin embargo, no eliminará los accidentes de tránsito. La ejecución del proyecto brindará al conductor la

detección del vehículo de frente con antelación con la finalidad de reducir el riesgo de un posible impacto.

1.2. Objetivos

1.2.1. Objetivo General:

Diseñar un sistema de reconocimiento automático de vehículos mediante el uso de redes neuronales profundas (DNN) a partir de una secuencia de imágenes para aplicaciones de seguridad vehicular.

1.2.2. Objetivos Específicos:

- Revisar sistemáticamente la literatura referente a redes neuronales profundas con el uso de repositorios digitales para determinar las bases teóricas necesarias.
- Determinar los requerimientos tanto de software como de hardware que necesita el diseño para su fase de entrenamiento y ejecución.
- Definir la selección y tratamiento del dataset para evitar datos erróneos en el aprendizaje de la red neuronal.
- Definir la arquitectura de red neuronal profunda y analizar su procesamiento con el dataset seleccionado para alcanzar considerables valores de distancia de ubicación del objeto y altos valores de precisión en el proceso de detección.
- Realizar las pruebas funcionales de reconocimiento automático mediante una secuencia de imágenes de test.

1.3. Alcance

En el transcurso de seis meses se desarrollará un sistema de reconocimiento automático de vehículos, mediante el uso de redes neuronales profundas que detectará la presencia de objetos categorizados como “vehículos”, orientado a aplicaciones de seguridad vehicular como un

sistema de asistencia al conductor o sistemas de parqueo, para viajes largos en carreteras o espacios cortos de estacionamiento.

Para obtener las bases teóricas requeridas en la revisión bibliográfica, se realizará una revisión minuciosa de literatura especialmente en artículos y estudios científicos realizados que se encuentren en repositorios.

El módulo de preparación de datos para las fases, tanto de entrenamiento como de pruebas, comprende el pre-procesamiento de los dataset de imágenes disponibles en la web con fines de investigación, que contiene más de miles de imágenes entre vehículos de transporte público y privado, previamente seleccionados y organizados en carpetas; a los cuales se aplicarán transformaciones y redimensionamiento (Data augmented) antes de generar el formato de archivo más adecuado y permitido como entrada a la red neuronal. Se utilizará los datasets existentes en lugar de imágenes capturadas en un ambiente particular, con la finalidad de entrenar a la red con grandes volúmenes de imágenes que contengan diferentes dimensiones desde varios puntos de vista, buscando reducir los problemas de aprendizaje (*Overfittin*g = entrenar una red con datos para los que se conoce el resultado deseado, generando así un bajo error. *Underfitting* = entrenar una red con datos dispersos y diferentes a los datos con los que se va a validar, generando así un alto error). Esto además facilitará las pruebas funcionales del sistema en un ambiente de laboratorio sin la necesidad de recurrir a ambientes reales como carreteras. Es importante mencionar que ésta alternativa ha sido seleccionada luego de haber realizado un estudio sobre las fuentes de datos de entrenamiento de sistemas basados en el aprendizaje, los cuales exigen grandes cantidades de datos que incluyan la mayoría de las posibilidades que puedan darse en un ambiente.

El componente software se basará en una red neuronal profunda con sus fases de entrenamiento y pruebas. La fase de entrenamiento contemplará los módulos generales:

preparación de datos, entrenamiento de la red. El módulo de preparación de datos viene dado por la elección adecuada de las imágenes de test adaptables a nuestro ambiente y para el módulo de entrenamiento de la red neuronal profunda se analizará las técnicas más adecuadas de procesamiento de imágenes con sus respectivos factores que influyen para una convergencia entre tamaño de la red neuronal, tales como el número de capas, tamaño de los filtros, parametrización por capa, dimensión de las imágenes de entrada y las limitaciones que la neurona debe tener para adaptarse a las condiciones reducidas y en ambientes ideales. La fase de pruebas abarcará los módulos generales: preparación de datos, uso de la red entrenada, identificación y extracción de los objetos de tipo vehículo y post-procesamiento para su detección. Los módulos de esta fase hacen referencia a las pruebas que se realizan con la red neuronal ya entrenada y usando varias secuencias de datasets para determinar una matriz de confusión y así analizar los errores y aciertos del sistema.

1.4. Justificación

Tomando en cuenta el objetivo número seis del Plan Nacional del Buen Vivir, el cual se enfoca en garantizar la seguridad integral y la calidad de vida de las personas, reducir la tasa de mortalidad tanto por accidentes como por homicidios, así como aumentar la tasa de jueces, fiscales y defensores. El presente trabajo tiene como finalidad facilitar creación de aplicaciones que permitan disminuir la tasa de accidentes de tránsito causadas por distracción del conductor, mediante alertas emitidas con antelación al conductor, sobre la presencia de vehículos detectados automáticamente con el uso de técnicas basadas en la visión por computador.

Proyecto que será una parte fundamental para un futuro proyecto de implementación de un sistema de asistencia al conductor (ADAS), el cual es un mecanismo que permite mejorar la seguridad del automovilista en el momento de la conducción, y poder así minimizar el riesgo de sufrir un accidente vial o colisionar con otros vehículos. Un sistema ADAS es un conjunto

de soluciones que se adaptan al vehículo para ofrecer seguridad, entre ellas: control de cruce de carril, estacionamiento automático, detección de punto ciego, adaptación de velocidad, control de luces, entre otros; las cuales actúan dependiendo del escenario en el que se encuentre.

El reconocimiento automático de objetos dentro del campo de visión por computadora permite abrir un mundo de aplicaciones que mejoren la calidad de vida en diversas áreas, tales como, la agricultura, la ganadería, la industrial, la medicina, la seguridad, entre otros; es por ello que los científicos tienen enfocado el punto de estudio en ello. Es así que, para garantizar la seguridad de los conductores, en los últimos años han capturado el interés investigativo en aplicaciones enfocadas a seguridad avanzada como son los sistemas ADAS y adaptarlos a sistemas embebidos que cuentan con capacidades reducidas.

La propuesta está planteada en base a las necesidades que vive nuestra sociedad, el sistema de reconocimiento automático de vehículos se adaptará a aplicaciones de seguridad vehicular y su detección se aplicará en situaciones de ambientes sin variaciones del clima, es decir, detecta los vehículos cuando no existe neblina, lluvia, variaciones de luminosidad, entre otros.

1.5. Descripción del trabajo

El resto de este escrito está organizado de la siguiente manera: El capítulo 2 presenta un estudio esencial teórico de la inteligencia artificial, especificando la visión por computador y estado del arte del reconocimiento automático de objetos, sus pasos generales, técnicas y algoritmos. También se presenta un contraste entre redes neuronales artificiales y las redes neuronales profundas, así mismo sus aplicaciones y arquitecturas. Seguido de una revisión de varios trabajos de investigación basados en Redes Neuronales Convolucionales para reconocimiento automático de objetos y el análisis de las posibles opciones de Redes Neuronales Convolucionales para dispositivos con recursos limitados.

El capítulo 3 presenta el método de aprendizaje explorado y su diseño para entrenar a una Red Neuronal Convolucional, capaz de lograr un desempeño de alto nivel con parámetros que eviten el consumo excesivo de recursos computacionales.

El capítulo 4 presenta los resultados logrados con el diseño propuesto, analizando de manera cuantitativa y cualitativa las métricas obtenidas. Y finalmente, en la última sección se presenta algunas conclusiones y recomendaciones para posibles extensiones de trabajos a futuro con el desarrollo del mismo.

CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se dará a conocer aspectos sobre la visión por computador para comprender el reconocimiento automático de objetos, sus aplicaciones, algoritmos, procedimiento y las técnicas que se pueden utilizar. Con ello, se introduce a un contraste entre redes Neuronales Artificiales y Redes Neuronales Profundas, sus aplicaciones y arquitecturas, detallando las Redes Neuronales Convoluciones que presenten valores de precisión altos y bajo consumo de recursos computacionales.

2.1. Visión por computador

Dentro de la Inteligencia Artificial (IA), uno de los campos con un extenso y novedoso número de aplicaciones es el campo de visión artificial. El cual, provee una amplia gama de soluciones reales y eficientes para procesar, analizar e interpretar secuencias de entrada tanto de video como de imágenes, dando así origen al procesamiento de imágenes como una disciplina especializada. Particularmente, el procesamiento de imágenes tiene como finalidad "comprender" e interpretar las características o patrones en secuencias de imagen/video para incrementar la calidad de los datos recolectados y lograr una "comprensión automática" de ellos.

Entre las aplicaciones más sobresalientes dentro del procesamiento de imágenes existen:

- Reconocimiento de patrones: Se define como "El reconocimiento de características únicas que identifican un sujeto de los demás de la misma especie" (Vega, Cortez, Alarcón, & Romero, 2009). El reconocimiento de patrones es una técnica de la IA y es empleada por tecnologías para el procesamiento de visión computacional. Su uso se ha visto reflejado en terapias de lenguaje mediante el reconocimiento de voz o en ultrasonidos donde se desarrolla técnicas para mejorar la identificación y seguimiento

de formas, también tiene su aplicación en cirugías plásticas de donde se puede generar una reconstrucción tridimensional de la parte del cuerpo a intervenir.

- Seguimiento de objetos: Es una de las aplicaciones más utilizadas hoy en día, debido a que permite encontrar la ubicación de un objeto en el tiempo. Para el seguimiento de objetos se utilizan técnicas de reconocimiento de objetos, lo cual aumenta la complejidad del proceso. Su interés es amplio en vigilancia e inteligencia de negocios, en juegos interactivos, aplicaciones médicas, aplicaciones vehiculares y medios de comunicación.
- Reconstrucción: La reconstrucción de objetos es posible gracias a la IA, a partir de una secuencia de imágenes, que contengan el objeto deseado, es posible procesar una imagen incompleta para que pueda ser reconocida y clasificada. Se aplica en el análisis y/o estudios de restos arqueológicos.
- Extracción de características: Se basa en la identificación y selección de rasgos principales y de interés contenidos en la imagen. Una imagen puede poseer varios rasgos que sean de utilidad, por ejemplo, para clasificar y detectar un objeto.
- Segmentación: La segmentación se define como "la división de la imagen en sus partes constituyentes u objetos" (Caballero H. , 2001). La segmentación permite obtener los objetos o las áreas de interés, seleccionando las zonas con mayor probabilidad de aparición del objeto y rechazando las zonas con ruido. El tipo de segmentación a utilizarse depende de la aplicación, ya sea en el campo de reconocimiento, seguimiento o clasificación de objetos.

Actualmente, la amplia gama de avances tecnológicos permite obtener gran cantidad de datos a través de imágenes de computador y escáner, cámaras digitales o teléfonos móviles, los mismos que deben ser procesados para obtener información útil para aplicaciones de visión por computador que se expanden constantemente en distintos campos como filtrado, interacción

humana, captura de movimiento óptico, control de tráfico, agricultura, medicina, seguridad, militar, control de calidad, robótica (De la Escalera, 2017).

2.2. Reconocimiento automático de objetos

La identificación de objetos es muy importante en el reconocimiento automático de objetos (Corrales & Rivas, 2007), detección en multitudes (Sánchez, 2014), conteo de personas (Paci, Brunelli, & Benini, 2014), identificación de acciones (Alvear, 2016), identidades (Caballero E. , 2017) y productos (Clint, 2015). Generalmente, el reconocimiento de objetos es una tarea de pre-procesamiento en muchos sistemas de visión por computador, la cual debe permitir una identificación eficiente y precisa del (o de los) objeto(s) de interés para reducir la clasificación errónea, falsas alarmas, y positivos perdidos conocidos también como falsos positivos; al mismo tiempo que debe proporcionar una rápida ejecución y adaptación en diversos escenarios. Una correcta detección debe identificar de forma precisa la forma, bordes y posición del objeto; al mismo tiempo la detección debe ser lo suficientemente rápida para identificar varios objetos, patrones normales y eventos inusuales.

2.2.1. Pasos generales para el reconocimiento de objetos

De forma general, el reconocimiento automático de objetos se obtiene mediante los siguientes pasos:

- Pre – procesamiento: En esta sección se aplica técnicas que permitan ajustar a la imagen para los siguientes pasos, de esta manera se adecua una imagen para una aplicación específica. Entre las técnicas más utilizadas se menciona el mejoramiento del contraste, reducción de ruido, realce de características, normalización del brillo, transformación del espacio de color, entre otros.
- Etapa de parametrización: Dentro de esta etapa se define los descriptores, en muchas ocasiones también conocidos como patrones, los cuales permitirán discriminar a un

objeto, se puede obtener descriptores que tengan información general del objeto o descriptores con información específica del mismo. Los descriptores generales son aquellos que contienen información básica, como el color (Salomón, Misller, & Miranda, 2015), el movimiento (Martínez, Gómez, & Romero, 2009), la forma (Sanz, 2008), la localización o la textura (Ciriza, Albizua, & Gonzales, 2009) (Olveres, Nava, & Escalante, 2016), entre otros. Por otro lado, los descriptores específicos no son fáciles de extraer ya que contienen información sobre los objetos o los eventos de la imagen, entre ellos están atributos topológicos (Malavé, Márquez, & Lezama, 2015), perímetro y área (Bergamini & Kamlofsky, 2015), distancias, tamaño y orientación (Solera, Moreira, & Hernández, 2014).

- Construcción del clasificador: Esta etapa es importante dentro del reconocimiento de objetos, debido a que, haciendo uso de los algoritmos y métodos de detección de objetos, se construye un modelo del sistema que sea capaz de dar significado a las imágenes entrantes de acuerdo a los parámetros escogidos de la etapa anterior, de esta manera se agrupa a los objetos que contengan los mismos descriptores con el fin de discriminar, separar a los diferentes y obtener una respuesta. Es así como el clasificador no tendrá problemas en tratar distintas entradas con varios descriptores, este proceso permite que el reconocimiento automático de objetos sea posible y aprenda el comportamiento correcto que el clasificador debe tener.
- Reconocimiento del objeto: Una vez definido el clasificador, este tiene la habilidad de identificar objetos de interés en una entrada, para lo cual cada imagen de entrada pasa por el clasificador y este detecta los descriptores seleccionados; dando como resultado una respuesta de salida con el o los objetos de interés detectados. De esta manera se completa el ciclo de detección del objeto, el clasificador es capaz de detectar el objeto en cualquier entrada.

2.2.2. Técnicas de aprendizaje

Dentro del reconocimiento automático de objetos, se encuentra las técnicas que conllevan al aprendizaje de ciertos patrones para conseguir su objetivo. Estos pueden ser de aprendizaje supervisado o no supervisado.

2.2.2.1. Aprendizaje supervisado

"En el aprendizaje supervisado la red dispone de los patrones de entrada y de salida que se quiere obtener para esa entrada, y en función de ellos se modifican los pesos de las sinopsis para ajustar la entrada a la salida" (Galán & Martínez, 2015).

Con este aprendizaje se busca obtener características/patrones en un espacio con mínimo error, pero con ello se tiene el riesgo de perder la capacidad de generalización, es decir, no reconoce aquellos datos que se encuentran distorsionados pero que sean correctos. Este aprendizaje necesita un humano que mida el funcionamiento del sistema.

El aprendizaje supervisado puede ser de tres tipos (Sistemas Expertos e Inteligencia artificial, 2016): (i) Aprendizaje por corrección de errores, en el cual su estructura cuenta con un comparador que permite diferenciar cada salida obtenida con cada salida deseada y de esta manera minimizar el error entre ellos. (ii) Aprendizaje por refuerzo, este tipo de aprendizaje es más lento que el anterior debido a que solo se conoce una salida deseada en general, únicamente proporciona una sugerencia de corrección de salida en la red. (iii) Aprendizaje estocástico en el que se realizan cambios aleatorios hasta obtener una salida que sea aceptada.

Los sistemas de aprendizaje utilizan algoritmos que le permiten entrenar y realizar su función correctamente. En los últimos años, en la literatura se han presentado varios algoritmos de aprendizaje supervisado, como se muestran en la Tabla 1:

Tabla 1 Algoritmos de aprendizaje supervisado

Algoritmo	Descripción / Función	Aplicaciones	Ventajas	Desventajas
Árbol de decisión	Busca clasificar los elementos mediante el uso de una serie de variables y posibilidades	-Medicina, identificación de patrones de lesiones fatales (Timarán, Calderón, & Hidalgo, 2017). -Industrial, detección de niveles de agua (Rivas, 2008).	Llega a una conclusión lógica presentando las posibles consecuencias y eventos fortuitos, construyendo un modelo.	En cada elección considera únicamente una hipótesis por lo que lo hace un proceso lento.
K vecinos más cercanos (k-NN)	Se centra en criterios de vecindad, dado un conjunto de ejemplos clasificados, se clasifica uno nuevo a la clase que contenga la mayor cantidad de vecinos más cercanos.	-Industrial, clasificador del producto para predicción de precios (Troncoso, 2007). -Medicina, identificador de señales respiratorias para clasificar pacientes (González & Giraldo, 2015).	Es un algoritmo sencillo debido a que no es necesario construir un modelo explicito, ni hacer suposiciones sobre los datos.	Lentitud en la clasificación cuando existen muchos datos, dependencia de la función distancia, gran influencia del ruido en los datos.
Naïve Bayes	Es capaz de clasificar un nuevo elemento que no tenga grupo perteneciente, haciendo uso de elementos ya conocidos.	-Industria automotriz, como coches autónomos. -Informática, localización de correos y clasificación de artículos.	Toma decisiones óptimas razonando con probabilidades, las nuevas instancias se clasifican combinando hipótesis.	Elevado coste computacional en el proceso de actualización de probabilidades, requiere conocer gran número de elementos.
Regresión ordinaria por mínimos cuadrados	Utiliza el método matemático llamado regresión lineal con la estrategia de mínimos cuadrados ordinarios, para estimar los valores que no se conocen.	-Medicina, como el reconocimiento de patrones de comportamiento (Pacheco & Díaz, 2005). -Climatología, análisis de los patrones del comportamiento del clima.	Permite realizar predicciones a partir de datos parciales, lo que le hace un algoritmo de funcionamiento rápido.	No es apto para el manejo de variables aleatorias continuas.
Regresión logística	Manera estadística de representar un resultado binomial con variables explicativas, estima la probabilidad de ocurrencia de un suceso.	-Medicina, identificación de variables endocrinólogas (Domínguez & Padilla, 2001). -Académico, reconocimiento de patrones de rendimiento estudiantil (Reyes, Escobar, Duarte, & Ramírez, 2007).	Se utiliza en escenarios de grandes volúmenes de datos por su eficiencia, además el resultado del modelo se puede interpretar como una probabilidad.	Resultados satisfactorios en caso de que los atributos y clases sean linealmente separables.
Support Vector Machines (SVM)	Es de clasificación binaria, con dos grupos de distintos tipos, el SVM buscará la manera de separarlos mediante líneas rectas estando lo más lejos posible entre ellos.	Reconocimiento de sitios de empalme humano, detección de genero basado en imágenes y clasificación de imágenes a gran escala.	Predice el grupo al que pertenece una nueva entrada, basándose en una sola variable. Permite minimizar el error en la separación, utilizando pocos parámetros a estimar.	Es un clasificador binario y no está diseñado para identificar los atributos importantes que construyen la regla discriminante.

Redes neuronales	Simulan el comportamiento de un cerebro humano, conectando varios procesadores entre sí.	-Telecomunicaciones, reconocimiento de patrones, filtrado de señales, reconocimiento de caracteres (Galán & Martínez, 2015). -Robótica, identificación de movimientos (Moya, Herrero, & Guerrero, 1998). -Psicología, identificación de características para diagnóstico (Montaño, 2002).	Son capaces de aprender de la experiencia, crea su propia organización y tiene un bajo porcentaje de fallos o errores.	Complejidad de aprendizaje para grandes tareas, por su gran cantidad de datos, requieren de una gran capacidad de hardware.

Fuente: Autoría propia

2.2.2.2. Aprendizaje no supervisado

El aprendizaje no supervisado es aquel en el que no se conoce los datos deseados de salida por lo que lo hace más complejo, en este aprendizaje se arman grupos dependiendo de los patrones que se busque en los datos. Estos grupos pueden ser varios, los cuales contienen máximas o mínimas diferencias entre ellos.

Según Cáceres (2010), el aprendizaje no supervisado que basa en clustering; el clustering o agrupamiento es el proceso mediante el cual se divide en subclases a un conjunto de datos. La clasificación de estos agrupamientos es no supervisada, es decir, las etiquetas de las clases no están definidas y tampoco se conoce el número de grupos creados.

Un cluster presenta características como: escalabilidad, formas arbitrarias, es capaz de manejar diferentes tipos de atributos, puede añadir restricciones, maneja con facilidad el ruido, funciona con alta dimensionalidad, son independientes del orden de los datos y los clusters deben ser interpretables y fácilmente utilizables (Himani, Darshak y Udesang, 2014).

De la misma manera que en la técnica anteriormente mencionada, el aprendizaje no supervisado cuenta con varios algoritmos mostrados en la Tabla 2:

Tabla 2 Algoritmos de Aprendizaje no supervisado

Algoritmo	Descripción / Función	Aplicaciones	Ventajas	Desventajas
k-means	Agrupa los elementos que más parecido tengan entre sí, basándose en una característica concreta.	-Segmentación de imágenes (Pinto, 2015). -Geoestadísticas (Honarkhah & Caers, 2010). -Medicina, detección de patrones de agrupamiento de mortalidad de cáncer (Ortega, Reyes, Cruz, & Pozos, 2007).	Es un algoritmo simple y trabaja muy eficientemente gracias a que tomo decisiones rápidas con la característica concreta.	Es muy sensible al ruido y está limitado a encontrar clusters globulares.
K-medoids	Es una partición clásica de agrupación a un conjunto de datos de n objetos en k grupos conocidos, selecciona los puntos centroides y mediante un matriz arbitraria las distancias entre ellos.	-Agricultura, Agrupar genes y proteínas (Gravano & Mislej, 2015). -Telecomunicaciones, detección para la distribución de una red hibrida (Amoroso & Avila, 2015)	Aporta una mayor precisión diferencial entre grupos, además es un algoritmo más robusto frente al ruido.	Requiere indicar el número de clusters que se van a crear, además las agrupaciones resultantes varían con la asignación inicial de los centroides.
Análisis de Componentes Principales (PCA)	Algoritmo de reducción de dimensionalidad, es invariante para escalar los datos.	-Ecología, Identificación de diagnóstico socioambiental (Olivares, 2014). -Teledetección, detección de objetos terrestres (Ruiz, 2013). -Educación, determinación de la influencia de enseñanza (González, Diaz, Torres, & Garnica, 1994).	Presenta mayor robustez cuando las variables tienen unidades distintas, además son fáciles de interpretar sus resultados.	Utiliza los vectores propios de la matriz de covarianza y considera que los datos se distribuyen de forma gaussiana, no considera otra forma.
Análisis de Componentes Independientes (ICA)	A diferencia del PCA, pretende aprender una base linealmente independiente para los datos.	-Medicina, determinación de región cerebral estimulada (Sandoval, 2014), detección de ruido en señales electrocardiográficas (Muñoz, Rivera, & Duque, 2008). -Extracción de característica de imágenes (Álvarez & Giraldo, 2008)	Aprende no solo cualquier base linealmente independiente, sino una base ortonormal para los datos.	No se obtiene información sobre la varianza de la señal original, además no se puede reponer el orden de los componentes.

Fuente: Autoría propia

2.2.3. Algoritmos para el reconocimiento automático de objetos

El reconocimiento automático de objetos es un tema que permite realizar varias aplicaciones que solucionen problemas, los cuales necesitan de alguna actividad humana. Para que esto sea posible, varios científicos han estudiado la manera de lograr que un sistema computacional se comporte o actúe de acuerdo a cada aplicación. En un principio se encontró la manera de obtener máquinas que estaban limitadas a realizar cálculos predeterminados y de esta manera solucionar un problema, estas máquinas tuvieron una gran acogida en el mundo científico y tecnológico, pero, por muchas aplicaciones y publicaciones que se realizaron, estas solo se basaban en razonamiento lógico. Es así como, con el pasar del tiempo, se obtuvieron programas que aprendían de su propia experiencia, sistemas capaces de razonar con sentido común, sistemas con razonamiento analógico de geometría, sistemas con demostración de teoremas, sistemas con reconocimiento visual (Romero, Dafonte, & Gómez, 2007).

El reconocimiento automático de objetos se lo realiza con el uso de algoritmos, estos pueden ser básicos, no estadísticos, estadísticos y técnicas avanzadas (Teledetección: clasificación y detección, 2003). Sin embargo, también existen varios algoritmos catalogados como híbridos ya que combinan diversas técnicas para incrementar la precisión en el reconocimiento.

2.2.3.1. Algoritmos básicos

Son métodos que se caracterizan por la baja complejidad computacional en sus operaciones, lo cual a menudo genera un bajo nivel de precisión. A esta categoría pertenecen los siguientes algoritmos:

- Diferenciación de fotogramas: Consiste en tomar dos imágenes consecutivas, para determina la información que ha variado en un fotograma inicial y los fotogramas siguientes. De esta manera el fotograma de referencia posee toda la información del objeto y los fotogramas siguientes únicamente la

información que ha variado. El cálculo de este método es simple y fácil de implementar, puede ser muy útil en entornos que varían dinámicamente, sin embargo, es complejo conseguir el contorno completo del objeto cuando se encuentra en movimiento debido a que pueden existir vacíos en la imagen lo que hacen que la detección del objeto no sea preciso.

- Flujo óptico: Es una técnica específicamente para video, en la cual, en una imagen del video se tiene puntos de esta y se compara con la imagen siguiente, entonces se puede decir que el punto se ha movido de un lado a otro lo que implica su flujo óptico. Su procedimiento es mediante vectores de movimiento entre imágenes secuenciales, su ventaja es que permite comprender la información contenida en el video y así evaluar la escena, la posición y la cantidad. Por otro lado, requiere de una gran cantidad de cálculos lo que le hace una técnica lenta e inconveniente para aplicarla en tiempo real.
- Sustracción de fondo: Es una técnica ampliamente utilizada para detectar objetos en movimiento, este proceso es una serie de métodos que distingue entre las zonas estáticas y las zonas dinámicas de la imagen. El modelado de fondo es el núcleo del algoritmo de sustracción de fondo, se asigna un modelado de referencia con el cual se compara cada secuencia del video para determinar la posible variación, cuando existe un cambio entre ellos, se dice que el objeto se encuentra en movimiento.

2.2.3.2. Algoritmos Estadísticos

El reconocimiento de objetos con métodos estadísticos se basa en cálculos de probabilidades condicionadas, permite interpretar datos cuyo carácter esencial es la variabilidad. Estos métodos pueden utilizarse para confirmar hipótesis o explorar conjunto de datos sin hipótesis, entre ellos son:

- Máxima verosimilitud: Este método asume que los datos tienen una función de distribución normal para asignar la probabilidad de que cualquier descriptor pertenezca a cada una de las clases, de este modo se le asigna a la clase que sea más probable que pertenezca. Este método puede utilizarse de forma automática o puede asignarse las clases dependiendo de la superación de un umbral determinado.
- Método bayesiano: En este método se calcula la probabilidad de cada hipótesis y, a partir de estos valores, se predice los datos desconocidos. Permite hacer predicciones y reconocimiento de objetos basándose en métodos estadísticos.
- Regresión de soporte vectorial: Realiza una regresión en un espacio de variables continuas, el método genera la línea de regresión haciendo uso de un margen de distancias, mediante el cual reconoce el objeto clasificándolo a la clase correspondiente.

2.2.3.3. Algoritmos no estadísticos

Dentro de los algoritmos no estadísticos, se puede mencionar a los siguientes (Teledetección: clasificación y detección, 2003):

- Árbol de decisiones: El método plantea una serie de cuestiones al respecto de la clase a la que pertenece el objeto dependiendo de sus características, estas cuestiones planteadas hacen referencia a los diferentes valores que puede tomar, de modo que trazan fronteras entre clases. Al terminar todo el proceso, el objeto que identificado en su clase.
- Mínima distancia: Cada pixel de la imagen es calculado su distancia a las distintas clases por lo que el objeto es asignado a la clase a la que se encuentre con una distancia mínima, pero este método ha sido tomado como de muy baja precisión debido a que puede clasificar objetos sin ninguna garantía.

- Método por paralelepípedos: Es un método sencillo de utilizar, en el cual, se definen una serie de rectángulos que cumplen como fronteras de cada clase. El problema de este método es que pueden aparecer descriptores en varias clases y el reconocimiento es confuso.

2.2.3.4. Algoritmos de detección avanzada

El desarrollo de la tecnología respecto a los ordenadores, ha permitido implementar métodos más robustos que los mencionados con anterioridad. El problema de métodos anteriores es que no se conoce si cumplen con la función o no, como es una hipótesis, una clase o una distancia, es así como nacen las redes neuronales artificiales.

Las redes neuronales artificiales tratan de simular el procesamiento del cerebro, se estructuran como numerosos procesadores conectados entre sí. Matich (2001) define a una red neuronal artificial como un sistema para el tratamiento de la información, cuya unidad básica de procesamiento está fundamentada en la célula principal del sistema nervioso humano, es decir, la neurona. Una neurona artificial es un objeto lógico (se trata de software no de hardware) que recibe diversas entradas, hace una suma ponderada de las mismas y produce una salida a partir de la aplicación de una función umbral a la media ponderada.

Un ejemplo es el trabajo realizado por García (2013), donde se realiza un Reconocimiento de imágenes utilizando Redes neuronales artificiales, teniendo como objetivo el reconocimiento de cualquier objeto. Para el reconocimiento de un objeto en una imagen se utiliza previamente un entrenamiento de la red, la cual, para este trabajo, es realizada en un servidor y almacenada en una base de datos, en donde las neuronas artificiales aprenden características del objeto. Mediante el uso de un teléfono celular, el usuario realiza una captura de imagen del objeto que desea reconocer y se ingresa la imagen a la red neuronal entrenada por su capa de entrada, se procesa en las neuronas de la capa oculta, utilizando las características aprendidas, cada neurona aporta con una salida obteniendo una respuesta de reconocimiento de la imagen en la capa de

salida. Las limitaciones que presenta este trabajo se deben a la cantidad de imágenes con distintos objetos que deberían entrenar para cumplir con su objetivo.

Las redes neuronales artificiales han llegado a ser utilizadas para la detección de objetos en tiempo real, como se observa en el trabajo de Moralejo y Storni (2018) en donde se entrena imágenes del objeto en una red neuronal que obtiene principales características de la misma, mediante la utilización de dos cámaras como entrada, envía la imagen a la red entrenada y utilizando las características aprendidas, detecta el objeto permitido o no en una industria. La arquitectura de una red neuronal artificial cuenta con una única capa oculta que procesa las imágenes, esto hace que las características aprendidas de un objeto sean pocas y es por ello que se ve comprometida la precisión de detección.

El desarrollo de modelos con mayor capacidad de discriminación ha permitido superar las limitaciones mencionadas de una red neuronal artificial, es así como nace el estudio de una nueva técnica de aprendizaje automático: El Aprendizaje Profundo o Deep Learning. El cual, permite realizar tareas de clasificación directamente a partir de imágenes, texto o sonido, obteniendo altos niveles de precisión con la arquitectura de redes neuronales profundas.

2.2.3.5. Algoritmos Híbridos

Los algoritmos híbridos son aquellos que combinan dos o más algoritmos antes mencionados para mejorar y complementar deficiencias que tengan entre ellos y así solucionar un mismo problema. De esta manera se combinan las mejores características de cada uno de ellos a fin de que el global sea mejor que sus componentes individuales.

"La técnica híbrida suministra mejores resultados que las técnicas individuales" (Hernández & Rodríguez, 2016),el trabajo presenta una comparación de los distintos algoritmos, obteniendo mejores resultados con las características, debilidades, similitudes y fortalezas, se define una técnica híbrida con el uso de K-medias y SOM (KMediasSom).

Un clasificador usando los algoritmos Genéticos y Máquinas de Vectores de Soporte (Ramírez, 2010), presentan un mejor rendimiento frente a una Red Neuronal artificial. Como una de las aplicaciones de algoritmos híbridos se encuentra la detección de obesidad infantil (Suca, Córdova, Condori, Cayra, & Sulla, 2016), donde se combina un árbol de decisión con regresión en la selección de variables y Naive Bayes para la clasificación. Otra aplicación se presenta utilizando un Algoritmo PSO-Híbrido para solucionar el problema de ruteo de vehículos con entrega y recolección simultáneas (Lamos, Galvan, Gonzalez, & Cruz, 2013). Otro ejemplo presenta López (2011), Sistemas de reconocimiento de patrones híbrido inteligente.

2.3. Redes Neuronales profundas

A medida que se estudiaba la manera de lograr resolver problemas usando reglas, redes semánticas y modelos probabilísticos, aparecieron problemas más complejos como el gran número de variables, aprendizaje autónomo o auto organización de información, los cuales no se podían procesar usando un tipo de razonamiento lógico. Es por ello que se buscó la manera de realizar una solución como lo hace la inteligencia humana, con neuronas.

Las redes neuronales artificiales son procesadores interconectados con organización jerárquica, la cual, consta de una unidad procesadora de 4 elementos funcionales: receptor, sumador, activador y salida. La interconexión de estas neuronas forma la estructura básica de una red neuronal artificial las cuales se dividen en tres capas generales llamadas: capa de entrada, cuyas neuronas se conectan a las neuronas de la capa oculta y estas conectadas a la capa de salida. La capa de entrada es la que recibe directamente las señales de fuentes externas, mientras que la capa de salida es la que transfiere la información obtenida en la red neuronal hacia el exterior para ser analizada. La capa oculta puede contener varios niveles internos y estas no tiene contacto con el exterior, esta capa contiene neuronas procesadoras que pueden

estar conectadas de distintas maneras, de acuerdo a la topología que se plantee de red neuronal. Esta estructura básica es demasiado simple para obtener una red optima, es por ello que incrementar el número de capas a una red neuronal permite obtener mayor información de los datos de entrada. Las denominadas Redes Neuronales Profundas son aquellas que tienen una o más capas intermedias entre la entrada y la salida, estas redes son la base del aprendizaje profundo (López M. , 2016).

Las redes neuronales profundas utilizan una jerarquía de características en cada capa, lo que le permite aprender y clasificar, no únicamente el objeto deseado sino también las similitudes de todos los objetos presentes en la imagen. Estas redes utilizan una combinación de aprendizaje supervisado y no supervisado en sus diferentes capas ocultas internas, de esta manera reconstruyen la imagen de entrada aprendiendo características propias en cada una de ellas.

Las redes neuronales profundas a diferencia de las redes neuronales artificiales, son capaces de extraer un gran número de características de la imagen completa de entrada, es decir, extrae características de todos los objetos presentes en la imagen y esto permite agrupar, de cada objeto, las características principales. Esto es posible debido a que una red neuronal profunda es capaz de tener en su estructura una gran profundidad en su capa oculta haciendo que los datos pasen por varios procesos de reconocimiento de patrones, en cambio las redes neuronales artificiales cuentan con una estructura sencilla, en términos de número de capas.

El aprendizaje profundo se realiza mediante el uso de un gran número de datos etiquetados y la arquitectura de la red neuronal que, sin necesidad de realizar una extracción manual de características, aprende directamente de los datos. Gracias a la cantidad de capas ocultas que esta red puede tener, es posible obtener mayor información del objeto de entrada y tomar un resultado final más preciso, de esta manera, a medida que se avanza en la red, cada capa oculta obtiene características más complejas de la imagen de entrada (Jones, 2018).

Para su respectivo entrenamiento los parámetros de la red comienzan con una suposición y, al final de pasar los datos por toda la red, se obtiene el índice de error y la influencia de cada parámetro; con esta información el entrenamiento continúa actualizando sus parámetros. Unas ves entrenada la red, es capaz de realizar su tarea con datos no etiquetados, los cuales la red no había visto antes.

2.3.1. Aplicaciones

Gracias al tratamiento y resultado que ofrece las redes neuronales profundas, han tenido gran acogida en aplicaciones, tal como un sistema automático de análisis de imágenes para el reconocimiento de un conjunto limitado de objetos (Gamarra & Ríos, 2018). Así como también en el tránsito vehicular para la Detección de vehículos en imágenes satelitales (Jiang, Cao, Cheng, Wang, & Li, 2015), para el reconocimiento automático de matrículas de vehículos (Nuñez, 2016) (Diaz, 2006) o detección de peatones (Vizcaya, 2018).

El reconocimiento facial es una de las aplicaciones más desarrolladas en el mundo de las redes neuronales profundas (Aplicaciones reales de Deep Learning, 2016), utilizando una red para detectar rasgos faciales.

La detección de posturas humanas utilizando redes neuronales profundas (Toshev & Szegedy, 2013) es una propuesta interesante que interactúa con las articulaciones del cuerpo humano.

De acuerdo a López (2016) las redes neuronales profundas se utilizan para detectar pinturas artísticas, de donde se ingresa una imagen a la entrada y basándose en rasgos artísticos, identifica a un pintor en particular. Otra aplicación se presenta en el desarrollo de una red neuronal profunda aplicada en el campo de la medicina para la detección de objetos presentes, por ejemplo, en tomografías.

El mundo de la realidad virtual y videojuegos también se ha visto beneficiada con el uso de redes neuronales profundas, debido a que ayudan al videojuego a aprender del usuario y actuar según una base de aprendizaje constante.

2.3.2. Arquitecturas

Las diferentes arquitecturas de las redes neuronales profundas han permitido ampliar la cantidad de soluciones a problemas que una red neuronal puede proveer, como la extracción de características, precisión, actualización de parámetros, capacidad limitada, entre otros; de esta manera, mientras más datos se usen para entrenar a una red, mejor nivel de precisión se obtendrá mediante diferentes estructuras profundas de redes. Recopilando las arquitecturas más usadas (Restrepo, 2015) y (Jones, 2018), se encuentra:

2.3.2.1. Redes neuronales recurrentes

Una Red neuronal recurrente (RNN) es una de las arquitecturas fundamentales debido a que, a partir de esta, se construye otras arquitecturas de aprendizaje profundo. Una red multicapa se diferencia de una red recurrente en sus conexiones, una RNN tiene conexiones de retroalimentación entre capas o a la misma capa, lo cual permite una memoria de entradas pasadas, es por esto que requieren de mayor tiempo de procesamiento (Oropeza, 2007).

Las RNN constan de un amplio conjunto de arquitecturas, todas con su característica principal de retroalimentación y tienen un comportamiento variante en el tiempo, lo cual permite resolver problemas con no – linealidades temporales significativas, es decir, son ideales

para la identificación y clasificación de patrones secuenciales que tengan diferentes probabilidades de suceder en el tiempo.

2.3.2.2. Redes recurrentes de memoria corta y larga

Las Redes recurrentes de memoria corta y larga (LSTM) son parte de las tradicionales RNN, estas redes mejoraban el pre-entrenamiento con la introducción de celdas de memoria, las cuales permitían recordar lo valores importantes a corto o largo plazo.

Las celdas de memoria forman la capa oculta de la red en una LSTM, la estructura de una celda de memoria está compuesta por tres puertas que controlan su funcionamiento. La primera puerta es la de entrada, la cual controla cuando fluye información a la memoria. La segunda puerta es la de olvido, es la que permite que la célula recuerde los datos antes de olvidarla. La tercera puerta es la de salida, controla cuando la información contenida es usada en la salida. Cada puerta tiene sus respectivos pesos, los cuales son actualizados en base a la función de error obtenida en la red resultante (Pérez, 2002).

2.3.2.3. Redes neuronales convolucionales

Estas redes neuronales de aprendizaje profundo se crearon inspiradas en la estructura del sistema visual humano, las primeras capas reconocen características, como bordes, y las capas posteriores combinan estas características en atributos de nivel superior de la entrada, es por ello que su arquitectura es muy útil en aplicaciones de procesamiento de imágenes.

La estructura de estas redes se basa en dos tipos de capas: las capas convolucionales y las submuestreadas, cada neurona relacionada con su peso. Este tipo de redes tienen mejor desempeño que una arquitectura de red neuronal profunda tradicional, debido a que, aun cuando

se utiliza pesos aleatorios en las primeras capas de entrenamiento, sus pesos se van adaptando, obteniendo buenos resultados con la red convolucional completamente entrenada.

2.3.2.4. Deep Belief Network

El Deep Belief Network (DBN) es una arquitectura de red típica, pero incluye un novedoso algoritmo de entrenamiento. El DBN es una red multicapa en la que cada par de capas conectadas es una máquina Boltzmann restringida (RBM). Su entrenamiento depende de la aplicación, el aprendizaje no supervisado se utiliza para reconstrucción de entradas o datos de entrenamiento y el aprendizaje supervisado se utiliza para aplicaciones de clasificación (Restrepo, 2015).

En el DBN, la capa de entrada representa las entradas sensoriales primas, y cada capa oculta aprende representaciones abstractas de esta entrada. La capa de salida, que se trata de manera diferente a las otras capas, implementa la clasificación de red.

2.3.2.5. Autoencoders

Un autoencoder tiene la característica de lo mismo que se tiene a la entrada, obtener a la salida, pasando por un proceso de codificación. Una forma de limitar a las neuronas es con el autoencoder, estas se especializan en una sola sección de entrada, teniendo en su capa oculta pocas neuronas con valores altos, es decir, que se activen.

Existen Autoencoders apilados, los cuales tienen varios autoencoders apilados y permiten tomar el resultado de la primera codificación para utilizarlo en la siguiente codificación y así sucesivamente hasta obtener el resultado.

2.4. Redes Neuronales convolucionales (CNN)

En los últimos años, las redes neuronales convolucionales han sido un enfoque de estudio para varias aplicaciones que resuelven problemas complejos que, gracias a su precisión, son

comúnmente utilizados en reconocimiento y procesamiento de imágenes. El modelo de una red convolucional se introdujo en 1998 con la red LeNet (Lecun, 2015), la cual se basaba en la clasificación de dígitos y eran utilizadas en los bancos para reconocer los escritos a mano de un cheque. El número de capas de esta red eran de ocho, se limita por recursos disponibles informáticos, debido a que requiere largas capas convolucionales para procesar imágenes con mayor resolución.

Motivados por los altos niveles de precisión en clasificación, segmentación y reconocimiento de objetos mediante el uso de redes convolucionales profundas. En los últimos años varias arquitecturas de redes convolucionales profundas han sido presentadas en la literatura, demostrando que el nivel de precisión alcanzado está relacionado con la profundidad de la red y con los métodos tanto de aprendizaje como de optimización.

Así, en el 2012, se presentó la red Alexnet (Lampert, 2017), la cual, en el desafío de reconocimiento visual de gran escala ImageNet, superó a todas las redes anteriores gracias a su reducción de error al 15,3 %. Su arquitectura es similar a la de LeNet, pero contiene más filtros por capa y capas convolucionales apiladas, lo que hace una red más profunda. AlexNet contiene ocho capas, las primeras cinco son capas convolucionales y las tres últimas son capas conectadas por completo, con 60 millones de parámetros. Usa la función de activación de ReLU no saturante, que muestra un rendimiento de entrenamiento mejorado sobre tanh y sigmoide. AlexNet ha tenido un gran impacto en el campo del aprendizaje automático, específicamente en la aplicación del aprendizaje profundo a la visión artificial.

GoogLeNet (Günel, 2016) aparece en el 2014 y muestra un rendimiento muy cercano al nivel humano, con una tasa de error de 6,67% para el reconocimiento visual. Esta red utiliza una CNN basada en Lenet, pero implementando un elemento llamado módulo de inicio, el cual, realiza varias convoluciones muy pequeñas con el fin de aumentar la precisión y reducir el

número de parámetros. Su arquitectura contiene una CNN de 22 capas de profundidad y utiliza 5 millones de parámetros.

En el mismo año se obtiene la red VGGNet (Li, Johnson, & Yeung, 2017), la cual consta de 16 capas convolucionales y es muy atractivo debido a su arquitectura muy uniforme. Esta red obtuvo el primer lugar en localización y el segundo lugar en clasificación, en el desafío de reconocimiento visual de gran escala ImageNet. Actualmente es la opción más preferida en la comunidad para extraer características de imágenes. La configuración de peso de VGGNet está a disposición del público y se ha utilizado en muchas otras aplicaciones y desafíos como extractor de características básicas. Sin embargo, VGGNet consta de 138 millones de parámetros, que pueden ser un poco difíciles de manejar.

Resnet (Prakash, 2017) se introdujo en el año 2015, logrando un índice de error de 3,57% en la tarea de clasificación, destacándose en varios concursos de clasificación y detección, tales como ILSVRC'15 y COCO'15. Este modelo tiene 152 capas de profundidad y hace uso de conexiones residuales, estas conexiones de omisión también se conocen como unidades con compuerta o unidades recurrentes con compuerta y tienen una gran similitud con los elementos exitosos recientes aplicados en las RNN. ResNet está enfocado en resolver el problema de precisión saturada con el aumento de la profundidad de la red.

2.4.1. Estructura de una red neuronal convolucional

Una red neuronal convolucional (CNN o ConvNet) es un tipo especial de redes neuronales multicapa diseñadas para reconocer patrones visuales directamente desde imágenes de píxeles con un preprocesamiento mínimo. El nombre de la red se deriva de su funcionamiento y su arquitectura. Generalmente su arquitectura se organiza en capas convolucionales, de Pooling y fully connected, con dos grandes propósitos: (i) Extraer características que poseen los datos de entrada o entrenamiento (usualmente imágenes), formando los mapas de características (feature

maps), los cuales van aumentando en número y disminuyendo en tamaño a medida que haya más capas ocultas en esta etapa. Aquí se encuentran la capa convolucional y la capa de pooling, las dos capas van en cascada y, por lo general, no se suelen separar. (ii) Una vez que los mapas de características han sido extraídos, una red fully connected como clasificadora se encarga de separar y clasificar cada característica. A pesar de que las CNNs varían en como las capas convolucionales y de pooling son realizadas, tal y como muestra la Figura 1:

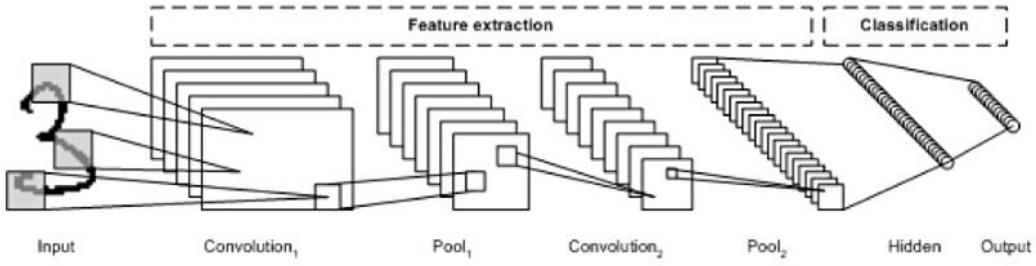


Figura 1. Estructura Red Neuronal Convolucional

Fuente: (Jones, 2018)

2.4.1.1. Capa de entrada:

La primera capa de una red convolucional es la entrada a la red, en la cual, normalmente se tiene un conjunto de imágenes previamente seleccionadas, que contengan los objetos que se desea reconocer. Estas imágenes deben estar anticipadamente con sus respectivas etiquetas para realizar un aprendizaje supervisado.

2.4.1.2. Capa convolucional:

Las imágenes naturales poseen la propiedad de ser estacionarias, esto quiere decir que las características o rasgos que existen en alguna parte determinada de la imagen, pueden ser los mismos que otros que se encuentran en otra zona totalmente distinta, es por ello que la capa convolucional su principal propósito es extraer características de una imagen. Consiste de un conjunto de filtros entrenables que realizan producto punto con los valores de la capa precedente, es decir, se aplica un mismo filtro a todos los conjuntos que se formen en la imagen

y se obtiene una imagen convolucionada, esto permite reducir el número de conexiones y el número de parámetros a entrenar en comparación con una red multicapa full-connected, tal y como se muestra en la Figura 2:

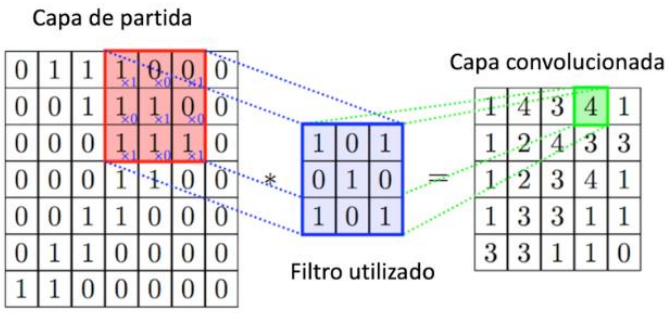


Figura 2. Proceso de una capa convolucional

Fuentes: (Calvo, 2017)

En la práctica, los valores de los filtros son aprendidos para su activación al encontrar ciertas características. Al ser colocados en cascada se obtienen diferentes niveles de abstracción.

2.4.1.3. Función de activación

La función de activación se usa para limitar la salida de una neurona y también para introducir no linealidades a las activaciones lineales generadas por una capa de convolución. La entrada a tales funciones es típicamente la salida de una capa de convolución. Hay varias funciones de activación usadas en redes neuronales convolucionales, algunos ejemplos populares son:

- Función sigmoidal: Con esta función de activación, la neurona permanece en cero hasta recibir una señal de entrada de donde empieza a ser asíntota gradualmente. Dicha función logística es la más usada debido a sus características de derivación y es recomendable para resolver los problemas de predicción, el rango de valores de salida de esta función es de 0 a 1. Sin embargo, de acuerdo a la investigación de Llano, Hoyos, Arias y Velásquez

(2007), la función sigmoidal no es práctica al tener una sola capa y al incrementar más capas, esta se vuelve lenta.

- Función tangente hiperbólica: La función de activación tangente hiperbólica es parecida a la función logística, el rango de valores de esta función es de -1 a 1. Esta función resulta ser muy utilizada en redes multicapa, su ventaja es que hace posible expresar polaridades, también la función de activación es comúnmente utilizada en aprendizaje supervisado como es el entrenamiento por retropropagación del error.
- ReLU (unidad lineal rectificada): Esta es la función de activación recomendada para redes neuronales convolucionales, esta función de activación se calcula como f (x) = max (0, x). Se ha demostrado que los algoritmos de entrenamiento convergen más rápido que con las funciones de tangente hiperbólica y sigmoidal, debido a que ReLU es lineal. Sin embargo, algunas neuronas nunca se activan, es decir, solo producen 0, reduciendo la capacidad de representación de la red

2.4.1.4. Capa pooling

El siguiente paso es ver en qué zona de la imagen se tiene esos rasgos predominantes y para realizarlo es posible calcular la media o buscar el máximo valor de una característica a lo largo de una región de la imagen. La salida de esta capa depende de la elección del usuario, debido a que se puede decidir qué operación realizar. Estas operaciones son: Promediar, maxpooling o elegir el máximo valor y subsampling que consiste en utilizar una función de tangente hiperbólica. La Tabla 3 presenta las funciones aplicables en esta capa:

Tabla 3 Funciones de la capa pooling

Tipo de pooling	Implementación	Descripción
Max-pooling	$\max_{rxr}(x_i)$	Máximo de una región cuadrada de tamaño rxr.
Promedio (Average) o downsampling	$\text{mean}_{rxr}(x_i)$	Promedio de una región cuadrada de tamaño rxr.
Subsampling	$\tanh(\beta \sum_{rxr} \frac{x_i}{r^2} + b)$	Promedio de la entrada de una región cuadrada de tamaño rxr y multiplicado por un escalar entrenable (β) y sumado un bias entrenable (b), calculando el resultado a través de una función no lineal.

Fuente: (Restrepo, 2015)

La matriz resultante es de dimensiones considerablemente menores que la matriz de características obtenida en la capa de convolución. El objetivo de esta capa es disminuir aún más la carga computacional del sistema y ayuda a la carnetización de la imagen obteniendo rasgos predominantes en ella.

2.4.1.5. Capa full connected

Cada una de las neuronas de esta capa está conectada con todos y cada una de los elementos de las matrices de la capa anterior. Se suele utilizar capas completamente conectadas en la que cada pixel se considera como una neurona separada al igual que en un perceptrón multicapa.

2.4.1.6. Capa de clasificación

La última capa de la red neuronal convolucional es la capa de clasificación de esta red, la cual determina la clase a la que pertenece la imagen de entrada. La salida de esta capa es la probabilidad que tiene la imagen de entrada de pertenecer a una determinada clase. Su objetivo es transformar todas las activaciones de red en la capa de salida final a una serie de valores que puede interpretarse como valores de vector de probabilidad entre 0 y 1. Softmax es la más utilizada para la clasificación en las redes neuronales convolucionales (Guachi, y otros, 2018). Esta aplica una distribución de probabilidad categórica basada en una función exponencial,

proporcionando un valor cercano a uno para la entrada máxima y un valor cercano a cero para el resto de las entradas.

La salida de nuestra red neuronal puede tener muchos valores y diferentes distribuciones, dependiendo de la función de activación; ReLu no tiene un límite superior por lo que puede tomar valores bastante grandes, es por ello que, el objetivo de la capa de clasificación de Softmax es simplemente transformar todas las activaciones de red en la capa de salida final en una serie de valores que puedan interpretarse como probabilidades.

2.4.1.7. Función de pérdida

En el contexto de las redes neuronales artificiales, la Loss Function son funciones diferenciables que caracterizan la similitud de una etiqueta con algunas características predichas por la red dado una entrada, es decir, a medida de que la entrada interactúa con cada neurona, con cada peso y con cada bias, al final en la salida se compara la salida con la respuesta deseada y se genera un margen de error, restando estos dos valores y realizando una operación de multiplicación y sumas hacia atrás para determinar cuánto contribuye cada neurona al error y así modificar sus pesos de acuerdo a la contribución de cada neurona para obtener el resultado. Este margen de error se lo realiza tantas veces sea el entrenamiento de la red, de este error la red aprende y actualiza los pesos y las ganancias de cada neurona.

Para el entrenamiento de detección de objetos, normalmente las redes son optimizadas para una tarea múltiple que incluye elementos de clasificación y regresión. Al cuantificar el rendimiento de clasificación, el método de entropía cruzada es una de las funciones de pérdida más comunes, la cual, entre dos distribuciones de probabilidad mide la media de bits necesarios para identificar un evento de un conjunto de posibilidades. Además, la entropía cruzada es una métrica que puede utilizarse para reflejar la precisión de los pronósticos probabilísticos y está estrechamente vinculada con la estimación por máxima verosimilitud. En la regresión, la función de pérdida varía de una red a otra.

2.4.2. Redes neuronales convolucionales para dispositivos con recursos limitados

Debido a los limitados recursos con los que se cuenta en sistemas embebidos utilizados para distintas aplicaciones de redes neuronales convolucionales, se ha estudiado arquitecturas alternativas que generen un bajo coste computacional y se ejecuten en plataformas con limitados recursos de memoria y procesamiento. Entre las arquitecturas más adecuadas, para este tipo de sistemas, se destacan:

2.4.2.1. Faster R-CNN

Faster R-CNN explota la salida del extractor de funciones para la clasificación y regresión, usando esta información para generar propuestas regionales a través de la Region Proposal Network (RPN). De esta manera, se genera un único conjunto de mapas de características a partir de la imagen de entrada, que luego se pasan a dos redes: (a) La RPN que genera propuestas regionales, (b) Una red convolucional rápida para la clasificación y la regresión.

La red de detección de objetos Faster R-CNN realiza esta tarea en varios pasos, por lo cual puede ser lento de ejecutar y también difícil de optimizar, ya que cada componente individual debe ser entrenado por separado (Menegaz, 2018).

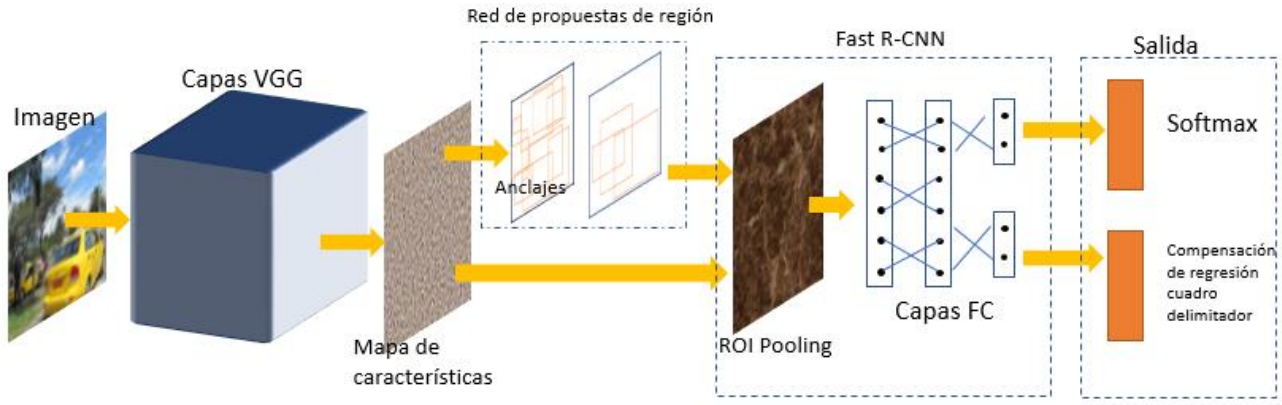


Figura 3. Estructura de red Faster R-CNN

Fuente: Autoría propia

La Figura 3 muestra la estructura básica de una Faster R-CNN, en ella se observa las capas VGG como una red base, donde las imágenes son previamente procesadas utilizando una CNN preparada para la clasificación, la cual permite obtener un mapa de características de la imagen, utilizando la salida de una capa intermedia.

Seguida de anclajes para el mapa de características, donde se utiliza la salida de la capa intermedia anterior para encontrar propuestas de regiones de interés mediante anclajes, los cuales son cajas de delimitación fijas que se colocan en toda la imagen con diferentes tamaños y proporciones utilizadas como referencia al predecir las ubicaciones de los objetos.

Continua con una Red de Propuestas de Región (RPN), la cual, toma todas las anclas o cuadros de referencia y genera un conjunto de buenas propuestas, estas se obtienen del análisis de las dos salidas diferentes en cada anclaje. La primera es la probabilidad de obtener un ancla el objeto, es decir, cualquier objeto que no sea fondo, y la segunda salida es la regresión del cuadro delimitador para ajustar los anclajes al tamaño del objeto que se predice. Para obtener los mapas de características se utiliza la función de activación ReLU, pero para la detección del objeto se utiliza mapas de activación, los cuales se obtienen con la RPN utilizando el ROI Pooling (Liao, 2018).

Para el entrenamiento, se toma todos los anclajes y se clasifican en dos categorías diferentes. Aquellos que se superponen con un objeto de verdad con una Intersección sobre Unión (IoU) mayor que 0.5 se consideran "primer plano" y aquellos que no se superponen con ningún objeto o tienen menos de 0,1 IoU se consideran "fondo ".

El RPN usa todos los anclajes para calcular la pérdida de clasificación utilizando la entropía cruzada. Luego, usa solo los anclajes de primer plano para calcular la pérdida de regresión.

Para poder asignar una clase al gran número de anclas sin etiquetar, se hace la reutilización del mapa de características convolucionales existente. Esto se hace extrayendo mapas de

características de tamaño fijo, a través de ROI Pooling, para cada propuesta usando la agrupación de la región de interés. Se necesitan mapas de características de tamaño fijo para la R-CNN a fin de clasificarlos en un número fijo de clases.

A diferencia de Max-Pooling que tiene un tamaño de corrección, ROI Pooling divide el mapa de características de entrada en un número fijo (por ejemplo, k) de regiones aproximadamente iguales, y luego aplica Max-Pooling en cada región. Por lo tanto, la salida de ROI Pooling siempre es k independientemente del tamaño de la entrada.

La red neuronal convolucional basada en la región (R-CNN) es el paso final de Faster R-CNN. Al obtener el mapa de características convolucional de la imagen, usado para obtener propuestas de objetos con el RPN y extrayendo características para cada una de esas propuestas, finalmente se necesita usar estas características para la clasificación. R-CNN intenta imitar las etapas finales de CNN de clasificación donde se utiliza una capa totalmente conectada para generar un puntaje para cada clase de objeto posible.

La red tiene dos vectores de salida: probabilidades de Softmax y compensaciones de regresión de cuadro delimitador por clase, es decir, una produce estimaciones de probabilidad sobre clases de objetos K más una clase de fondo y otra genera 4 valores codificados de posiciones refinadas del cuadro delimitador para una de las clases K.

2.4.2.2. You Only Look Once (YOLO)

La red YOLO se inspiró en el hecho de que, en el mundo real, los ojos humanos están disponibles para detectar objetos de un vistazo, por lo que es más natural tratar el problema como una regresión pura y predecir directamente los cuadros delimitadores, así como las clases correspondientes y niveles de confianza con una sola red.

YOLO utiliza funciones de toda la imagen para predecir cada cuadro delimitador. También predice todos los cuadros delimitadores en todas las clases para una imagen simultáneamente.

Esto significa que nuestra red tiene una razón global sobre la imagen completa y todos los objetos en la imagen. El diseño YOLO permite entrenamientos de extremo a extremo y velocidades en tiempo real, manteniendo una alta precisión promedio.

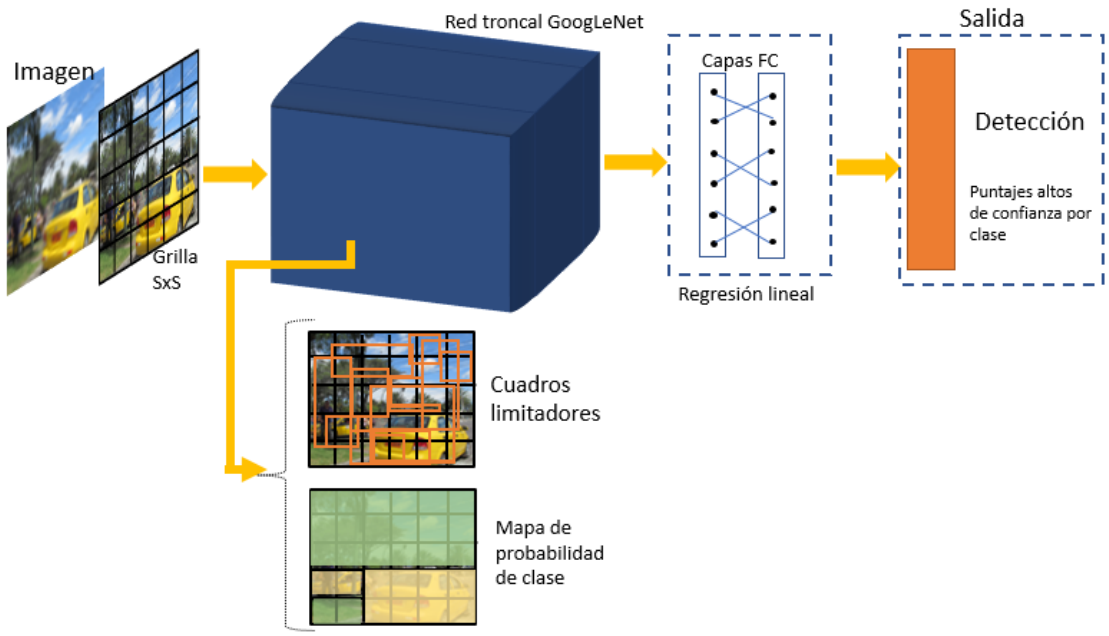


Figura 4. Estructura de red YOLO

Fuente: Autoría propia

La estructura de una red neuronal convolucional YOLO se presenta en la Figura 4, la imagen de entrada se divide en una cuadrícula de celdas S x S. Para cada objeto que está presente en la imagen, la celda donde cae el centro del objeto, será la responsable de predecirlo. Cada celda predice cuadros de límites B y puntajes de confianza para esos cuadros y cada cuadro delimitador consta de 5 predicciones: x, y, w, h y confianza. Las coordenadas (x, y) representan el centro del cuadro en relación con los límites de la celda, el ancho (w) y la altura (h) se predicen en relación con la imagen completa.

Formalmente se define la confianza como Pr (Probabilidad de Objeto) * IOU (Intersección Sobre Unión). Si no existe ningún objeto en esa celda, el puntaje de confianza debería ser cero. De lo contrario, el puntaje de confianza será igual a la intersección sobre unión entre el cuadro

predicho y la verdad sobre el terreno, se debe tomar en cuenta que la confianza refleja la presencia o ausencia de un objeto de cualquier clase.

La red YOLO tradicional está diseñada en base a la red troncal de GoogLeNet, con 24 capas convolucionales y capas max pooling, seguidas por 2 capas completamente conectadas, pero existe Fast YOLO la cual únicamente usa 9 capas convolucionales y lo demás es igual a YOLO tradicional. La capa final puede usar una función de activación lineal, pero todas las otras capas usan un RELU. Algunas capas de convolución usan capas de Pooling de 1x1 alternativamente para reducir la profundidad de los mapas de características. Después de una capa convolucional, las dimensiones de altura y ancho generalmente se reducen a favor de un crecimiento de la profundidad de los volúmenes de activación, que luego representarán los puntajes de clasificación. En un GPU Titan X, con una resolución de la imagen de entrada de 448 x 448, YOLO tradicional procesa 45 frames por segundo y YOLO rápido procesa 155 fotogramas por segundo (Redmon, Divvala, Girshick, & Farhadi, 2016). La salida de YOLO consta en predicciones de clase C, de donde, a partir de la grilla S x S, se tiene B cajas delimitadoras en cada una de las grillas y cada caja tiene 5 predicciones, es por ello que el vector de salida es S x S x (B * 5 + C).

2.4.2.3. Single Shot Detectors (SSD)

SSD fue lanzado a finales de noviembre de 2016 y alcanzó nuevos registros en términos de rendimiento y precisión para tareas de reconocimiento de objetos (Forson, 2017). SSD adopta una estructura completamente convolucional, lo que garantiza que las capas de salida implican la información de las ubicaciones, además el diseño de la estructura permite manejar predicciones a múltiples escalas.

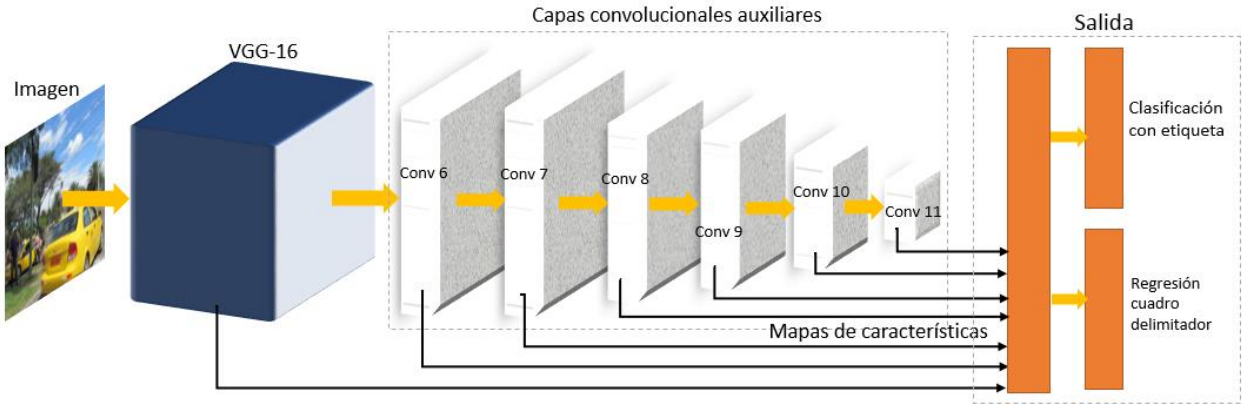


Figura 5. Estructura de red SSD

Fuente: Autoría propia

La detección de objetos SSD se compone de 2 partes: extraer mapas de características y aplicar filtros de convolución para detectar objetos. Como se observa en la Figura 5, SSD extrae mapas de características, basada en la venerable arquitectura VGG-16, pero descarta las capas totalmente conectadas.

La razón por la que VGG-16 se usó como red base se debe a su gran desempeño en tareas de clasificación de imágenes de alta calidad y su popularidad para problemas donde el aprendizaje por transferencia ayuda a mejorar los resultados. En lugar de las capas originales totalmente conectadas de VGG, se agrega un conjunto de capas convolucionales auxiliares (desde conv6 en adelante), lo que permite extraer características a múltiples escalas y disminuir progresivamente el tamaño de la entrada a cada capa subsiguiente. Es así como, conv4_ 3 es responsable de detectar los objetos más pequeños mientras que conv11_2 es responsable de los objetos más grandes. SSD se reserva una clase "0" para indicar que no tiene objetos.

Los MultiBox son como los llamados anclajes en YOLO, son una colección de recuadros superpuestos sobre la imagen en diferentes ubicaciones espaciales, escalas y relaciones de aspecto que actúan como puntos de referencia en las imágenes de verdad del suelo. Gracias al uso de mapas de funciones de escala múltiple, detectan objetos de forma independiente. A

medida que CNN reduce la dimensión espacial gradualmente, la resolución de los mapas de características también disminuye. SSD utiliza capas de menor resolución para detectar objetos de mayor escala.

En MultiBox, se creó lo que se llama prior, que son casillas de cálculo de tamaño fijo predefinidas que coinciden estrechamente con la distribución de las casillas de verdad terrestres originales. De hecho, esos antecedentes se seleccionan de tal manera que su relación Intersección sobre Unión (también conocida como IoU, y algunas veces referida como índice de Jaccard) es mayor que 0.5. Cualquier cuadro predeterminado con un IoU mayor que 0.5 se considera una coincidencia. La función de pérdida de MultiBox también combinó dos componentes críticos: (i) Pérdida de confianza, la cual mide cuán segura es la red de la objetividad del cuadro delimitador calculado. La entropía cruzada categórica se usa para calcular esta pérdida. (ii) Pérdida de ubicación, la cual mide cuán lejos están los cuadros de delimitación previstos de la red de los puntos de verdad del conjunto de entrenamiento.

Algunos de sus aspectos principales son los mapas de características de múltiples capas, donde, las capas superiores de redes convolucionales contienen buena calidad de detalle de la imagen. Por otro lado, las capas inferiores son mejores en la información de ubicación. Como resultado, SSD integra capas inferiores y superiores como mapas de características. En diferentes capas de entidades, las imágenes se dividen en diferentes tamaños de celdas de cuadrícula, y cada celda de cuadrícula es responsable de una ubicación correspondiente de objetos en la imagen.

Otro aspecto principal es la predicción Offset, la cual, divide las imágenes en celdas de la grilla en los mapas de características y resultados. Sin embargo, en lugar de predecir casillas de delimitación directamente, las predicciones de SSD se basan en "casillas predeterminadas". La resolución de la imagen de entrada para SSD es de 300 x 300, de donde el primer mapa de

características obtiene una profundidad de 512 canales (Gao, 2018), para cada celda de la imagen se le asigna un numero de cajas limitadoras de distintas escalas y en cada celda del mapa de características, se predice los desplazamientos relativos a las formas de caja predeterminadas en la celda, así como los puntajes por clase que indican la presencia de una instancia de clase en cada uno de esos cuadros, es decir, para cada caja k en una ubicación determinada, se calcula puntajes de clase c y los 4 desplazamientos relativos a la caja original por defecto, obteniendo (c + 4) k*m*n de salidas para un mapa de características m × n (Liu, y otros, 2016).

CAPÍTULO III: DISEÑO DEL SISTEMA

En este capítulo se presenta el análisis y diseño del sistema propuesto, incluyendo una exploración de los requerimientos tanto de software como de hardware, y el diagrama de bloques perteneciente a las dos fases del sistema. La fase de entrenamiento abarca dos módulos generales: la preparación de datos, comprende la selección y estudio del conjunto de imágenes (dataset) que proveerán el conocimiento a la red, los cuales son procesados para generar la entrada adecuada a la fase de entrenamiento; y el entrenamiento de la red, donde se analizan los factores y parámetros de configuración (número de capas, tamaño de los filtros, parametrización por capa, dimensión de las imágenes de entrada, entre otros) para alcanzar una convergencia entre tiempo de procesamiento y precisión en la detección. Mientras que la segunda fase es la validación del modelo entrenado, para comprobar el funcionamiento del sistema propuesto.

3.1. Características del sistema

Antes de analizar el diseño del sistema, es necesario identificar las características que este debe tener para su desarrollo. En esta sección se presenta aspectos importantes que se debe tener en cuenta para la parte del software y hardware del sistema, tales como sus requerimientos y su selección.

3.1.1. Requerimientos del sistema

De acuerdo a Alvear (2016), los requerimientos del sistema se definen partiendo del estándar ISO / IEC / IEEE 29148: 2011, el cual, expone como elaborar un buen requisito con características de aplicación duradera a lo largo del ciclo de vida del sistema. El proyecto se enfoca principalmente en el diseño de un sistema de reconocimiento automático, el cual necesita un software para desarrollarlo y un hardware que permita su despliegue, En la Tabla 4 se define los requerimientos de arquitectura del diseño:

Tabla 4 Requerimientos de Software

Requerimientos de arquitectura				
# SRSH	Requerimiento	Prioridad		
		Alta	Media	Baja
	Requerimientos de Software			
1	Requiere un sistema operativo de distribución libre.		x	
2	Requiere un software de tratamiento de un gran bloque de imágenes y archivos de texto.	x		
3	Requiere un software de programación compatible con el software de tratamiento de imágenes.	x		
4	Requiere un Dataset compatible con la arquitectura de red neuronal convolucional.	x		
5	Requiere una arquitectura de red neuronal convolucional con bajo consumo computacional.	x		
6	Requiere un software de tratamiento de imágenes para representar la salida del detector	x		
	Requerimientos de Hardware			
7	Se necesita un equipo que contenga la máquina virtual con todo el software requerido.	x		

Fuente: Autoría propia

3.1.2. Selección de software y hardware

La selección del software para el desarrollo del proyecto, se realiza en base a los requerimientos mencionados en la Tabla 4. Para el primer requerimiento SRSH 1, la elección de sistema operativo de código abierto es Ubuntu, el cual cuenta con varias ventajas, entre ellas, es totalmente gratuito, realiza tareas con mayor eficiencia que Windows, es un sistema operativo seguro y no tiene restricciones de uso. (Blanco, 2013).

Cumpliendo con el requerimiento SRSH 2, el software para el tratamiento de imágenes que se utiliza en este proyecto es Caffe, un framework de aprendizaje profundo, de código abierto, escrito en C++ y provee interfaces para Python y Matlab. El framework seleccionado ha sido desarrollado considerando la expresión, la velocidad y la modularidad. La arquitectura

expresiva permite fomentar la aplicación y la innovación, su uso puede alternarse entre CPU y GPU estableciendo una única bandera; la velocidad de Caffe ha sido altamente aceptable para uso en experimentos de investigación, ya que puede procesar más de 60 millones de imágenes de 375 x 500 píxeles de tamaño promedio por día con una sola GPU NVIDIA K40 * (Yangqing, y otros, 2014). El lenguaje de programación seleccionado es Python debido a su compatibilidad con Caffe, cumpliendo así con el requerimiento SRSH 3.

La eficiencia de una red neuronal convolucional depende de su arquitectura y parametrización de la red, así como también de la cantidad de datos utilizados en el entrenamiento. Los cuales deben contener un gran número de imágenes del objeto de interés, incluyendo una variedad de condiciones en las que el objeto de interés puede encontrarse, tales como diferente escala, iluminación, sombreado, localización, orientación, entre otros.

Para el requerimiento SRSH 4, varios trabajos afirman que el Dataset Pascal VOC presenta objetos más centrados en las imágenes, facilitando un gran volumen de información de cada etiqueta en detección de objetos frente a otros dataset (Hui, Detección de objetos: comparación de velocidad y precisión , 2018) (Gauen, Dailey, Laiman, Zi, & Asokan, 2017). Cuenta con conjunto de datos publicados desde el 2005 hasta el 2012, obteniendo en cada uno mayor número de imágenes y clases, llegando a 20 clases (autos, personas, gatos, caballos, entre otros) y alrededor de 20 000 imágenes en RGB para entrenamiento y validación con sus respectivos cuadros delimitadores. La unión más común de datos de Pascal es el uso de trainval 2007 y trainval 2012 para entrenamiento y test 2007 para validación, además de utilizar el test 2012 para las pruebas (Guanghan, 2015) (Redmon & Farhadi, 2018) (Hui, Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3, 2018). En este diseño se utilizará el dataset del año 2007 y 2012, de acuerdo a Hui (27 de marzo 2018), VOC Pascal 2007 no contiene la cantidad de información que contiene la versión 2012, presentando un mejor desempeño que el de 2012.

Para atender al requerimiento SRSH 5, se evaluaron las alternativas de arquitecturas de red para detección de objetos en tiempo real que generen un bajo coste computacional y se ejecuten en plataformas con recursos limitados de memoria y procesamiento, de donde el "Solo se ve una vez" es una opción óptima ya que procesa una sola vez la imagen, obteniendo los cuadros delimitadores, así como las clasificaciones de categorías de objetos. Mediante un proceso de extremo a extremo sin etapas separadas y así optimizar el error de detección de manera conjunta, es decir, realiza un único procedimiento sin procesos en paralelo obteniendo un resultado general de error de detección, de tal manera que no solo se entrena el modelo para una mejor precisión sino también para mejorar la velocidad de detección. Es por ello que, de las redes neuronales para detección de objetos mencionadas en el capítulo anterior, YOLO es la elección de este diseño.

El requerimiento SRSH 6 se atiende tomando en cuenta que este software debe ser una biblioteca libre, compatible con el sistema operativo y que sea una herramienta dedicada a la visión artificial. Opencv cumple con ello, siendo una multiplataforma de software de visión de computadora y de aprendizaje automático de código abierto, que abarca más de 2500 algoritmos como opciones para procesos de visión artificial y cuenta con interfaz de Python (OpenCV 3.4.1 , 2018).

Para el requerimiento de hardware, es necesario el cálculo de memoria RAM y de Disco duro que necesita la máquina virtual donde se desarrollará la investigación. El valor de memoria RAM se encuentra haciendo uso de la ecuación 3.1, según (Rojas, 2018).

$$\mathrm{RAM}(\mathrm{MV}) = \mathrm{RAM}(\mathrm{OS}) + \mathrm{RAM}(\text{Por cada hilo}) \qquad \text{(Ec. 3.1)}$$

La cantidad de RAM necesaria para la máquina virtual, $\mathrm{RAM}(\mathrm{MV})$, depende de la cantidad de RAM requerida por el sistema operativo, $\mathrm{RAM}(\mathrm{OS})$, y la RAM de cada hilo, $\mathrm{RAM}(\text{Por cada hilo})$.

De la misma manera, es importante conocer el valor de disco duro que debe tener la Máquina virtual para desarrollar el diseño. De acuerdo a Rojas (2018), la ecuación 3.2 se utiliza para este cálculo, donde se requiere el tamaño de disco que necesita el sistema operativo y cada uno de los hilos utilizados.

$$\mathrm{HD}(\mathrm{MV}) = \mathrm{HD}(\mathrm{OS}) + \mathrm{HD}(\text{Por cada hilo}) \qquad \text{(Ec. 3.2)}$$

La Tabla 5, muestra la RAM y Disco duro necesario del sistema operativo y de cada software a utilizar.

Tabla 5 Memoria RAM y Disco duro requerido

Requerimientos	RAM	Disco duro
Ubuntu 16.04	1GB	1.5GB
Caffe	2GB	16.8MB
Opencv	1GB	80 MB
Dataset Pascal VOC	-	4.20GB

Fuente: Autoría propia

Para satisfacer el requerimiento de hardware siguiendo los lineamientos del software requerido, es necesario un equipo que tenga asignado una memoria de 4 GB de RAM y un disco duro de 6GB como recursos mínimos.

3.1.2.1. Instalación de software de tratamiento de imágenes

La instalación de Caffe se realiza siguiendo las instrucciones de su página principal, seleccionando el sistema operativo Ubuntu. El siguiente link se dirige a la página oficial de Caffe:

http://caffe.berkeleyvision.org/install_apt.html

Una vez instalado las dependencias necesarias, se procede a descargar y compilar el repositorio de Caffe que contenga las herramientas requeridas para el entrenamiento de la red y el uso posterior del modelo entrenado. El siguiente repositorio es el utilizado en esta investigación:

https://github.com/yeahkun/caffe-yolo.git

3.2. Sistema Propuesto

El diseño propuesto, de manera general, se desarrolla a partir de tres elementos principales: (i) Datos de entrada, (ii) La red neuronal convolucional y (iii) Datos de salida, como se presenta en la Figura 6:

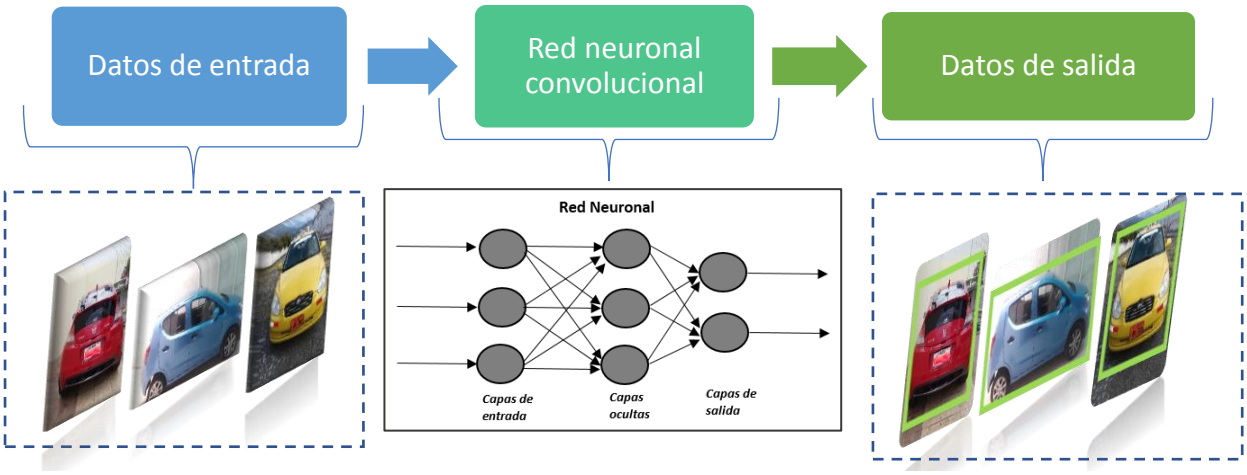


Figura 6 Elementos principales del sistema propuesto

Fuente: Autoría propia

El primer elemento del diagrama del sistema propuesto comienza con la entrada de datos, este bloque representa las imágenes con distintos vehículos, de diferentes colores y ángulos de vista. Los cuales deben ser previamente seleccionados y adecuados para la entrada a la red, proporcionando la información necesaria como tamaños, etiquetas, clases, entre otros.

La salida de datos del diagrama hace referencia a la detección del objeto de acuerdo a su clase, es decir, se obtiene el reconocimiento del objeto deseado dentro de la imagen mediante un cuadro delimitador alrededor del mismo. Este bloque visualizará en pantalla el resultado del diseño propuesto para comprobar su funcionamiento.

La red neuronal convolucional es el elemento central del sistema, debido a que, de su diseño depende el procedimiento que los datos realicen para obtener la detección del objeto. Este bloque contiene la arquitectura de la red neuronal convolucional YOLO con sus respectivos valores de activación y parámetros de cada capa.

3.2.1. Arquitectura de la red neuronal

YOLO tradicional está estructurada por una secuencia de 24 capas convolucionales, dentro de las cuales se utiliza capas de reducción con filtros de 1 × 1 seguidas de capas convolucionales con filtros de 3 × 3, y capas de pooling que extraen las características elementales de la imagen (formas, contornos, texturas, colores, entre otros), seguidas por 2 capas totalmente conectadas, las cuales predicen las probabilidades y coordenadas del cuadro delimitador (x, y, w, h) como salida.

La arquitectura está basada en la red convolucional YOLO presentada por Redmon, Divvala, Girshick, & Farhad (2016), la cual facilita su disponibilidad de uso. El escrito compara el rendimiento que alcanza la red y demuestra la precisión que tiene frente a otra arquitectura. Además, YOLO muestra ventajas en el aprendizaje de representaciones generales del objeto, lo que le hace menos probable que prediga falsos positivos en el fondo, aunque presenta algunos errores de localización.

También sigue la estrategia de Khan (2018), quien explica detalladamente el proceso a seguir en el modelo de red con las celdas y cuadros delimitadores, aplicando el diseño de red para la detección de vehículos en una secuencia de video. El autor consigue el entrenamiento

de la red desde cero y la detección de vehículos en videos nunca antes vistos, sin embargo, presenta algunos problemas con los objetos lejanos y detecciones bruscas en los cuadros delimitadores.

La arquitectura de la red YOLO se resume en la Figura 7:

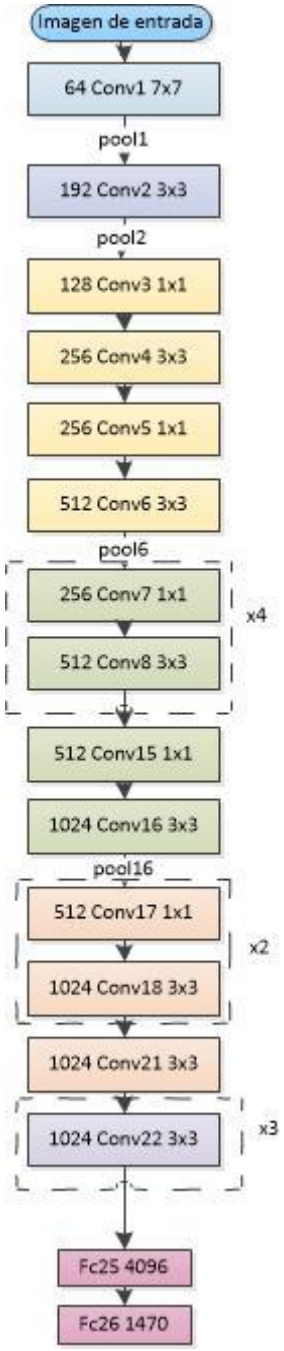


Figura 7. Arquitectura YOLO

Fuente: Autoría propia

La arquitectura de YOLO utiliza filtros de 64, 192, 128, 256, 512 y 1024 en sus capas convolucionales y de reducción, dentro de las cuales, cada una de ellas realiza operaciones requeridas de sumas-restas y multiplicaciones-divisiones para obtener las activaciones y parámetros de la red. La Tabla 7 presenta la cantidad de operaciones que realiza cada capa convolucional en los diferentes cambios de resolución de la imagen, obtenidas con la herramienta web:

http://dgschwend.github.io/netscope/#/editor

Tabla 6 Número de parámetros y operaciones de las capas convolucionales

Nombres de las capas convolucionales	**# de capas**	**Resolución de la imagen**	**# de Parámetros**	**Operaciones (SR y MD)**
Conv1	1	224 x 224	9.47k	472.06M
Conv2	1	112 x 112	110.78k	1.39G
Conv3, Conv4, Conv5, Conv6	4	56 x 56	1.565M	4.906G
Conv7, Conv8, Conv9, Conv10, Conv11, Conv12, Conv13, Conv14, Conv15, Conv16	10	28 x 28	10.228M	8.015G
Conv17, Conv18, Conv19, Conv20, Conv21, Conv22	6	14 x 14	29.365M	4.367G
Conv23, Conv24	2	7 x 7	18.876M	924.844M

Fuente: Autoría propia

La cantidad de operaciones que realiza la red neuronal YOLO, influye en la cantidad de costo computacional que el sistema requiera, es por ello que, a medida que la imagen disminuye su tamaño de resolución, tanto el número de parámetros como el número de operaciones disminuyen también.

La reducción progresiva del tamaño espacial de los datos permite la reducción del número de parámetros, el uso de recursos computacionales y la cantidad de cómputo en la red, es por ello que se controla el sobreajuste con las capas de Pooling, presentadas en la Tabla 8, con sus respectivas activaciones.

Tabla 7 Activaciones de las capas Pooling

Nombres de las capas pooling	**# de capas**	**Resolución de la imagen**	**Activaciones**
Pool1	1	224 x 224	802.82k
Pool2	1	112 x112	602.112k
Pool6	1	56 x 56	401.408k
Pool16	1	28 x 28	200.704k

Fuente: Autoría propia

En la Figura7, se observa que la arquitectura a la entrada recibe una imagen de 448 x 448 pixeles, lo que significa que necesita un total de 602112 neuronas en la primera capa, y continua con 24 capas convolucionales (Conv1, Conv2, Conv3……Conv24) con funciones de activación ReLU, acompañada por cuatro capas de pooling. El cálculo total de activación de neuronas se lo realiza con la ecuación 3.3:

$$\text{Activaciones} = \text{resolución de la imagen} * \text{filtros} \qquad \text{(Ec. 3.3)}$$

$$\text{Activaciones} = (448*448) * (3) = 602.11\text{k}$$

Regiones de 49 neuronas de entrada están conectadas a las neuronas de la primera capa convolucional mediante una ventana de 7x7 pixeles, desplazada horizontalmente por la imagen con un paso de 2 píxeles y con relleno de 3 para determinar características en la imagen de entrada. Esta capa está definida por 64 filtros, cada uno con su matriz de 49 pesos y bias, lo que permite identificar características en cada filtro, útiles para la detección.

El resultado de la primera capa convolucional (Conv1) es de 64 volúmenes de 224 x 224 píxeles, es decir, 3.211M neuronas. De donde, aplicando la capa de Pooling, se obtiene 64 volúmenes de 112 x 112 píxeles, esta reducción de dimensiones de cada volumen se debe al tamaño de bloque de 2x2 que tiene la capa.

Todas las demás capas de la arquitectura de red siguen la misma estructura descrita con la diferencia de que cada una tiene su parametrización. En la Tabla 8 se presenta de cada capa convolucional el número de filtros con su respectivo tamaño de ventana, el número de activaciones obtenidas y el tamaño de la imagen de salida de la capa, además se detalla el tipo y el tamaño de ventana de las capas de pooling con sus activaciones y dimensiones de tamaño de salida.

Tabla 8 Parametrización de cada capa

Capa	Filtros	Output convolucional	Pooling	Output pooling
Conv1	64 7x7	3.21 M 224x224	MAX 2x2	802.82k 112x112
Conv2	192 3x3	2.41M 112x112	MAX 2x2	602.11k 56x56
Conv3	128 1x1	401.41k 56x56	-	-
Conv4	256 3x3	802.82k 56x56	-	-
Conv5	256 1x1	802.82k 56x56	-	-

Conv6	512 3x3	1.61M 56x56	MAX 2x2	401.41k 28x28
Conv7	256 1x1	200.7k 28x28	-	-
Conv8	512 3x3	401.41k 28x28	-	-
Conv9	256 1x1	200.7k 28x28	-	-
Conv10	512 3x3	401.41k 28x28	-	-
Conv11	256 1x1	200.7k 28x28	-	-
Conv12	512 3x3	401.41k 28x28	-	-
Conv13	256 1x1	200.7k 28x28	-	-
Conv14	512 3x3	401.41k 28x28	-	-
Conv15	512 1x1	401.41k 28x28	-	-
Conv16	1024 3x3	802.82k 28x28	MAX 2x2	200.7k 14x14
Conv17	512 1x1	100.35k 14x14	-	-
Conv18	1024 3x3	200.7k 14x14	-	-
Conv19	512 1x1	100.35k 14x14	-	-
Conv20	1024 3x3	200.7k 14x14	-	-
Conv21	1024 3x3	200.7k 14x14	-	-
Conv22	1024 3x3	50.18k 7x7	-	-
Conv23	1024 3x3	50.18k 7x7	-	-
Conv24	1024 3x3	50.18k 7x7	-	-

Fuente: Autoría propia

El conjunto de estas capas que conforman la red es considerado un "sistema de extracción de características entrenable" (Nuñez, 2016), debido a que durante el entrenamiento se aprende los pesos y bias para las distintas características que definen el objeto que se va a detectar.

Finalmente, la red neuronal convolucional concluye con 2 capas totalmente conectadas, la primera con 4096 neuronas y la segunda con 1470 neuronas. Seguidas por una capa de "Loss Detection" para calcular, a partir de las características extraídas, la probabilidad de clase, la probabilidad de objeto y sus cuadros delimitadores del objeto en la imagen ingresada.

3.2.2. Diagrama de bloques de YOLO

El reconocimiento automático, se divide en 2 etapas. La primera es el entrenamiento de la red y la segunda etapa es su validación.

- Bloques de entrenamiento:

La etapa de entrenamiento del sistema es aquella que permite al sistema aprender características de los objetos que debe reconocer, su proceso se presenta en la Figura 8.

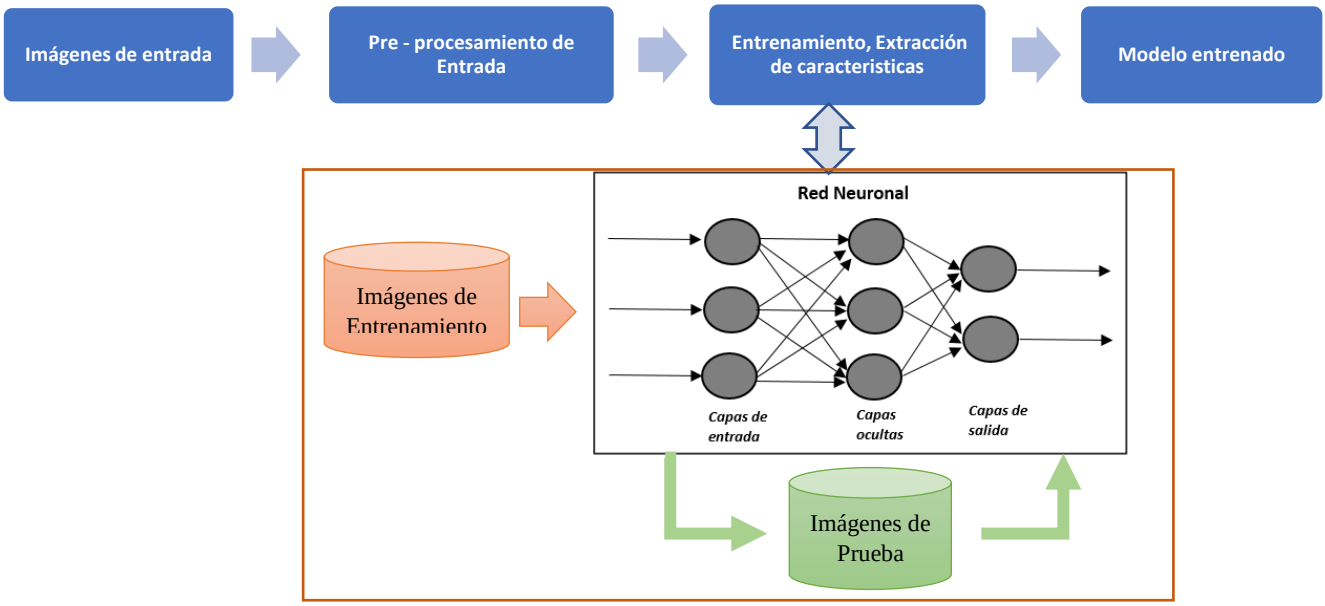


Figura 8. Diagrama de Bloques del entrenamiento

Fuente: Autoría propia

El primer bloque corresponde al dataset VOC Pascal, el cual contiene imágenes divididas y exclusivas para esta fase de entrenamiento.

Cada imagen de entrada pasa por la etapa de Pre- Procesamiento en donde la imagen se etiqueta, se extrae sus coordenadas de cuadros delimitadores, redimensiona, cambia de formato y se calcula la media del grupo de imágenes de entrenamiento.

El bloque de Entrenamiento se realiza mediante el uso de la red neuronal convolucional YOLO, para analizar y extraer las características (formas, figuras, colores, densidad, entre otros) de los objetos de cada una de las imágenes de entrada, las cuales son aprendidas durante el proceso mediante el uso de las capas iniciales de la red. Las imágenes de prueba permiten calcular el porcentaje de error que la red está teniendo con las características extraídas y sus parámetros, de esta manera corrige los parámetros y crea mapas de características correctos que permitirán detectar los objetos de interés (vehículos) en nuevas imágenes que usen el modelo entrenado.

El modelo entrenado es la salida que se obtiene del entrenamiento de la red, este contiene la información necesaria para la detección de los objetos (vehículos), tales como los pesos y bias de las neuronas, el número de neuronas en cada capa y el mapa de características del objeto.

- Bloques de validación:

La etapa de validación del sistema, es aquella que permite poner a prueba la red para verificar la detección automática de vehículos. Su proceso se presenta en la Figura 9:

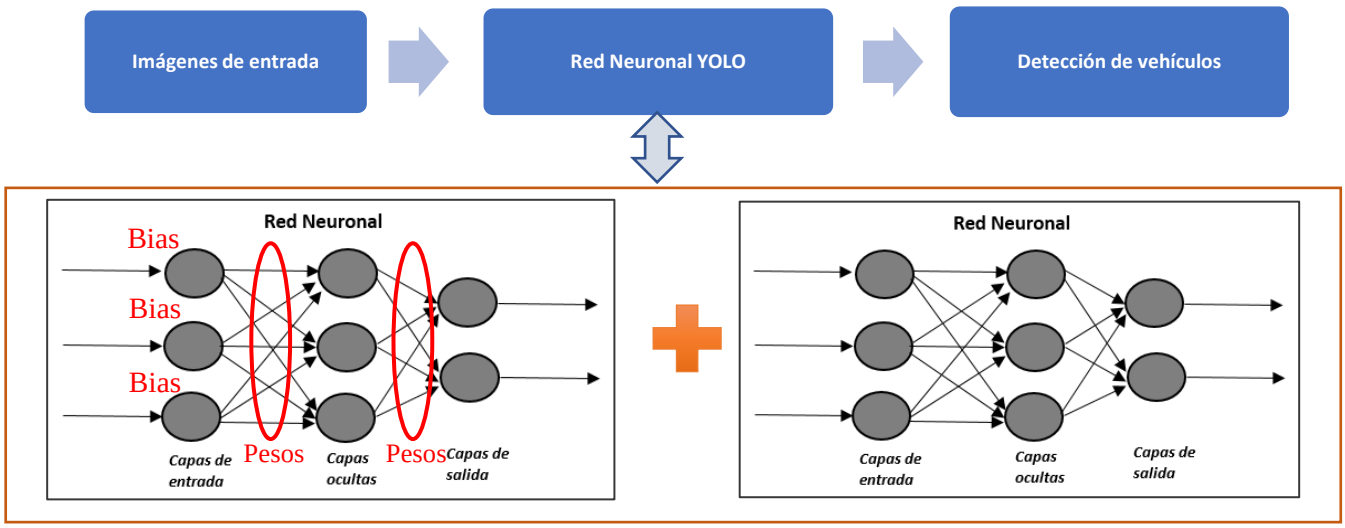


Figura 9 Diagrama de bloques de validación

Fuente: Autoría propia

La validación del diseño hace uso de un nuevo grupo de imágenes tomadas en la ciudad de Ibarra, las cuales se encuentran en RGB y formato .JPG, en ellas se realiza el proceso de detección de vehículos.

El bloque de red neuronal YOLO cuenta con el archivo del modelo entrenado, el cual contiene los parámetros aprendidos en la fase de entrenamiento, y a estos se agrega la arquitectura de la red YOLO.

La salida del sistema será la imagen de entrada con su respectivo cuadro limitador que corresponde al (a los) vehículo(s) detectado(s).

3.2.3. Conjunto de datos de entrenamiento

Las versiones VOC2007 y VOC2012, que se utilizan en el diseño, se encuentran subdivididas en varias carpetas como se muestra en la Figura 10, de las cuales se utilizan tres para la detección de objetos, estos son: Imagesets, JPEGImages y Annotations.

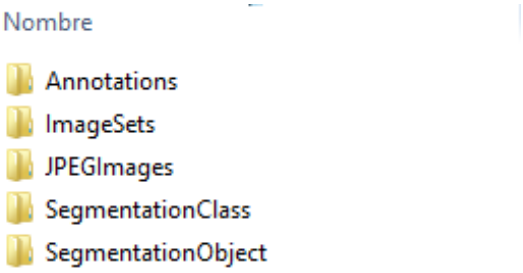


Figura 10. Subcarpetas de Pascal VOC

Fuente: Autoría propia

Como se muestra en la Figura 11, en Pascal VOC dentro de la carpeta Annotations, se encuentran archivos con extensión .xml que contienen sus respectivos parámetros del cuadro delimitador (x, y, h, w) y etiquetas de cada imagen (Everingham M. , Van Gool, Williams, Winn, & Zisserman, 2009).

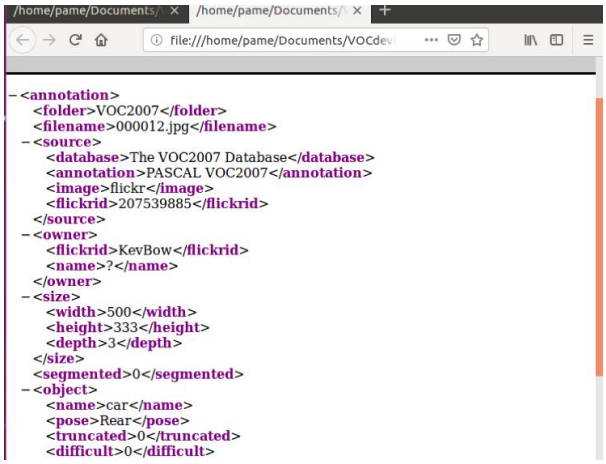


Figura 11 Etiquetas y parámetros de VOC Pascal

Fuente: Autoría propia

Estos archivos contienen información importante para esta investigación, los cuales se describen a continuación:

- *Filename:* Especifica el nombre de la imagen con su extensión.
- *Size:* Es el tamaño de la imagen.

- *Class:* Representa la clase del objeto que contenga la imagen, es importante recordar que una imagen puede tener más de una clase.
- *Pose:* Se refiere a la vista del objeto, este puede ser: frontal, trasero, izquierda o derecha. Las vistas se marcan subjetivamente para indicar la vista del "volumen" del objeto.
- *Truncated:* El objeto puede tener o no marcada esta opción, pues, un objeto marcado como 'truncado' indica que el cuadro delimitador, especificado a continuación, no contiene la extensión completa del objeto, es decir, no se encuentra el objeto completo en la imagen. Por ejemplo, una imagen que contenga una persona desde la cintura para arriba.
- *Difficult:* Un objeto marcado como "difícil" indica que el objeto es considerado complejo de reconocer, es decir, un objeto que no es claramente visible.
- *Bounding box:* Representa las coordenadas del cuadro delimitador, el cual es un rectángulo alineado alrededor del objeto.

En la carpeta Imagesets se encuentran archivos de texto que contienen y especifican las imágenes destinadas al entrenamiento, prueba y validación, también incluye archivos de texto que separan las imágenes dependiendo de su clase, como se muestra en la Figura 12. Y, por último, la carpeta JPEGImages contiene las imágenes nombradas con su respectiva etiqueta.

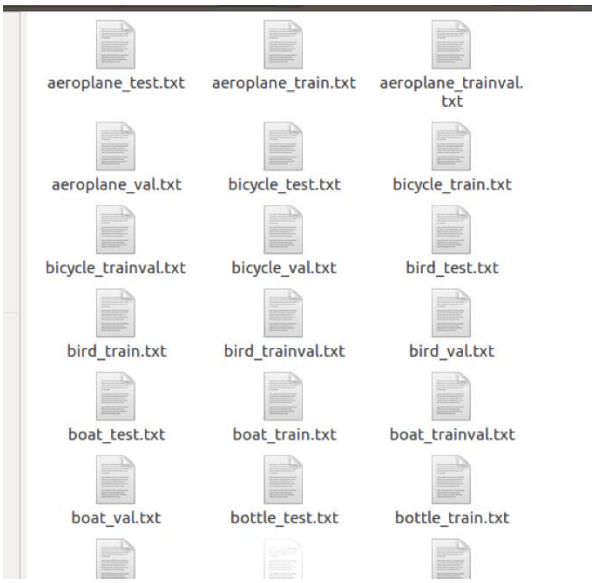


Figura 12. Dataset dividido por clase

Fuente: Autoría propia

Las imágenes se encuentran en formato .JPG y su modelo de color es RGB con un tamaño aproximado de 350x500 píxeles, además para descargar todo este kit de datos, se necesita 4.20 GB de espacio en el disco. En este proyecto se utilizó una velocidad de descarga de 5.602 Mb/s, así se presenta los tiempos de descarga y las características del dataset en la Tabla 10:

Tabla 9 Características del Dataset

Nombre del archivo	Tamaño	# de imágenes	Tiempo de descarga (min)
VOCtrainval_06-Nov-2007.tar	439 MB	2501	12
VOCtest_06-Nov-2007.tar	430 MB	2510	14
VOCtrainval_11-May-2012.tar	1.9 GB	5717	77
VOC2012test.tar	1.4 GB	5823	49

Fuente: Autoría propia

Organizando el Dataset en una sola carpeta llamada VOCdevkit, se realiza la descarga directa del conjunto de datos completo ejecutando los siguientes comandos:

wget http://pjreddie.com/media/files/VOC2012test.tar

wget http://pjreddie.com/media/files/VOCtrainval_11-May-2012.tar

wget http://pjreddie.com/media/files/VOCtest_06-Nov-2007.tar

wget http://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar

Posteriormente se extrae cada carpeta del conjunto de datos, en la misma carpeta llamada VOCdevkit con los siguientes comandos:

tar -xvf VOCtest_06-Nov-2007.tar

tar -xvf VOCtrainval_06-Nov-2007.tar

tar -xvf VOCtrainval_11-May-2012.tar

tar -xvf VOC2012test.tar

3.3. Ejecución

La fase de ejecución se divide en tres partes fundamentales para la red neuronal, la primera es el tratamiento de imágenes mediante el cual se adapta el conjunto de datos a la red YOLO, la segunda sección se enfoca en la realización del entrenamiento de la red y la tercera sección es la validación de la red entrenada para medir su eficiencia.

3.3.1. Tratamiento de los datos

Para iniciar con el tratamiento de los datos es necesario recopilar la información del dataset (nombre, clase, coordenadas, entre otros), además de la conversión del formato para conjuntos grandes de datos y por último el cálculo de la media de la imagen, la cual se restará de cada imagen para mejorar el rendimiento de la red.

3.3.1.1. Recopilación de información del Dataset

Para empezar a utilizar el dataset, es necesario recopilar la información necesaria del conjunto de datos, tales como, nombre de la imagen, parámetros del cuadro delimitador y clase. El módulo “Os” de Python se utiliza para realizar tareas como buscar el nombre del directorio de trabajo actual, cambiar el directorio de trabajo actual, verificar si existen ciertos archivos o

directorios en una ubicación, crear nuevos directorios, eliminar archivos o directorios existentes, recorrer un directorio y realizar operaciones en cada archivo en el directorio. El script lista.py de nuestro diseño, crean archivos con listas y accede a los archivos, tanto de VOC 2007 como VOC 2012, previamente descargados.

Los archivos creados son trainval.txt, test_2007.txt y test_2012.txt, dentro de los cuales se recolecta la información presentada por el dataset como muestra en el código Anexo A. En el resultado del script lista.py, se obtiene los tres documentos de texto mencionados, cada uno con listas que contienen ordenadamente el nombre de la imagen, la etiqueta del objeto y las coordenadas del cuadro delimitador, información necesaria para nuestra red artificial convolucional.

3.3.1.2. Conversión de formato del dataset

Cuando Caffe usa grandes volúmenes de datos, trabaja con un formato llamado Lightning Memory-Mapped Database (LMDB), la cual es una librería de software que proporciona un dataset en forma de almacén de valores claves de alto rendimiento. Caffe contiene la herramienta llamada convert_box_data, la cual permite convertir a lmdb un conjunto de imágenes y su lista de archivos, obtenidos con el script anterior, asociando la primera imagen con el primer archivo. Es importante tener siempre organizado tanto las imágenes como las subcarpetas del VOCdevkit y los archivos de texto que se van creando, de esta forma no existen errores al asociar los documentos con las imágenes.

La red Neuronal YOLO utiliza como resolución de entrada 448x448, es por ello que el script convertir.sh, mostrado en el código anexo A, realiza la creación del lmdb con la herramienta de Caffe, el redimensionamiento de la imagen, las etiquetas por clase y el trainval.txt de la sección de recopilación de información.

Este script se ejecuta tanto para la carpeta destinada a test como la de trainval y el resultado de este script se puede visualizar en las carpetas lmdb, como muestra la Figura 13:

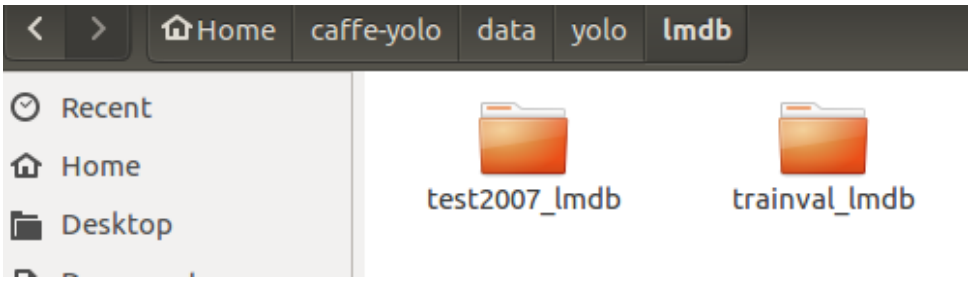


Figura 13. Carpetas lmdb

Fuente: Autoría propia

3.3.1.3. Cálculo de la media de la imagen

Es necesario calcular la media de la imagen de nuestro conjunto de datos para poder utilizarla durante el entrenamiento, debido a que, es una forma de normalización básica de los datos, restando esta media a cada una de las imágenes para que se encuentren "centradas".

Caffe proporciona una manera de calcular la media de la imagen directamente, utilizando la herramienta de compute_image_mean y haciendo uso del lmdb, tal como se muestra en el código del script media.sh anexo A. El resultado de este script es visible como un archivo binario, tal y como muestra la Figura 14:

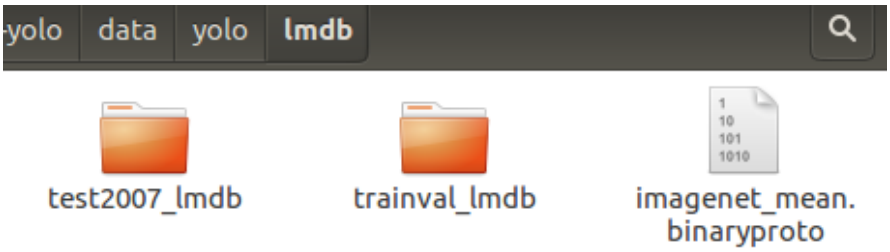


Figura 14. Archivo Imagenet_Mean

Fuente: Autoría propia

3.3.2. Entrenamiento de la red

El entrenamiento de esta red neuronal convolucional YOLO, se basa en la utilización del algoritmo de propagación hacia atrás, el cual permite la optimización de la misma. En este proceso se buscan los valores adecuados de los pesos y bias para que el valor de la función de error sea mínimo. En este sistema se hace uso del método de descenso del gradiente estocástico, el cual, permite calcular iterativamente el gradiente para un subconjunto de los datos de entrenamiento, actualizando los pesos de la red mediante la propagación de errores hacia atrás, sin necesidad de procesar todo el conjunto de datos completo.

El algoritmo de propagación hacia atrás calcula cómo varía el error a medida que cambia los pesos en las conexiones, mientras que el algoritmo de descenso de gradiente optimiza el error de la función de costo. El uso de este método de descenso de gradiente en el sistema, está basado en la estrategia utilizada por (Nuñez, 2016), para disminuir el coste computacional debido a que ofrece una convergencia más rápida.

3.3.2.1. Inicialización

Partiendo de la arquitectura de red presentada en el apartado 3.4., es necesario iniciar agregando a la capa de entrada las dos fases de datos: la primera de entrenamiento y la segunda de prueba. Para ello, se incluye las carpetas de imágenes tanto de entrenamiento como de prueba y a su vez la media de la imagen, calculada en el apartado anterior. En ambas carpetas de imágenes de entrada se asignan valores de subconjunto, los cuales representan la cantidad de imágenes que la red debe tomar de la carpeta de entrenamiento para calcular los parámetros de la red y la cantidad de imágenes que debe tomar de prueba para pasar al siguiente paso.

En el script entrenamiento_yolo.prototxt, mostrado en el anexo B, se describe cada una de las capas de la red, tales como, capas convolucionales con su tamaño de ventana, filtros, pasos

entre pixeles y funciones de activación, capas de Pooling con su tamaño de ventana y tipo, capas totalmente conectadas con funciones de activación y salidas.

Para este diseño de red, se ha optado por utilizar la técnica de transferencia de aprendizaje, la cual consiste en utilizar un modelo entrenado en un conjunto de datos diferente e inicializar los parámetros de la red con este modelo, esta técnica permite disminuir el costo computacional del procesamiento de la red debido a que evita que la red comience con valores de cero.

3.3.2.2. Entrenamiento

Una vez definido los parámetros de la red, se procede a su entrenamiento y para ello es necesario concretar los valores que permiten aplicar el método de descenso del gradiente estocástico.

Caffe permite entrenar la red con el uso de Caffe train, la cual aprende modelos desde cero, reanuda el aprendizaje a partir de instantáneas guardadas y ajusta los modelos a nuevos datos. El script train.sh es para el entrenamiento y uso de esta herramienta, mostrado en el código anexo B, en él se especifica el archivo que contiene los parámetros de entrenamiento de la red y el modelo pre-entrenado para la transferencia de aprendizaje y así evitar las ponderaciones iniciales al azar, haciendo más óptimo el entrenamiento.

La ejecución del proceso de entrenamiento se lo realiza haciendo uso del script solver.prototxt mostrado en el código anexo B, el cual es un archivo de configuración que se utiliza para que Caffe tenga el conocimiento de cómo se desea entrenar la red. Los parámetros especificados son los presentados en la Tabla 11:

Tabla 10 Parámetros de ejecución del entrenamiento

Nombre del parámetro	**Descripción**	**Valor seteado**
net:	Ubicación de la red planteada, es decir, el archivo que contiene toda la parametrización presentada en el apartado de inicialización	gnet_train.prototxt
average_loss:	Frecuencia con la que se presentará los resultados en pantalla, haciendo un recuento de iteraciones. Muestra la perdida promedio de las ultimas iteraciones y evita el gasto de recursos al mostrar la perdida en cada iteración.	50
lr_policy:	Permite tener control sobre el cambio de la tasa de aprendizaje, con velocidad constante en intervalos.	"step"
stepsize:	Frecuencia con la que se realiza el cambio de la tasa de aprendizaje en intervalos constantes.	25
max_iter:	Número total de iteraciones al que va a llegar la red para dejar de entrenar, manteniendo un buen porcentaje de cada peso anterior para el nuevo cálculo de pesos en cada iteración.	10000
snapshot:	Caffe genera un modelo de la red entrenada para evitar problemas en el entrenamiento como cortes o interrupciones en la red, guardándolos en su respectiva carpeta.	50
solver_mode:	Modos en los que puede resolver la red GPU o CPU	CPU

Fuente: Autoría propia

3.3.2.3. Prueba

Como se menciona en el apartado de inicialización, la arquitectura de red tiene asignada su capa de prueba, la cual, utiliza un conjunto diferente a las imágenes que entrenan la red. El entrenamiento incluye esta fase de prueba, por lo que, después de cada iteración se puede comprobar el margen de error y actualizar los pesos, adaptándose a lo aprendido en la iteración.

La sección de prueba evita que se produzca un sobre-entrenamiento (*overfitting*) en la red, debido a que permite controlar la cantidad de imágenes que se entrenan antes de actualizar los pesos y bias, es decir, ayuda a prevenir un ajuste excesivo de parámetros, lo que puede provocar que una vez entrenada la red no sea capaz de clasificar correctamente imágenes distintas a las analizadas durante el aprendizaje.

Los parámetros para la sección de prueba se encuentran en solver.prototxt mostrado en el código anexo B, en cual, se especifica 500 para el tamaño del conjunto y 500 para el número de iteraciones después de las cuales prueba la red capacitada. También se desactiva la prueba para la primera iteración debido a que evita el ajuste innecesario de los parámetros en su comienzo.

3.3.3. Validación

Utilizando el modelo entrenado en el apartado de entrenamiento y haciendo uso de imágenes tomadas en la ciudad de Ibarra en ambientes ideales, como cambios de iluminación sin presencia de neblina o lluvia, se realiza la validación de la red para comprobar su funcionamiento. De acuerdo al código presentado en el anexo C, el script validacion.py necesita:

- Imágenes de prueba nunca antes vista por la red en su entrenamiento.
- El script deploy.prototxt contiene la arquitectura utilizada en el entrenamiento, con la diferencia de cambiar las capas que contienen las fases de entrenamiento y prueba

por una capa de entrada con sus respectivas dimensiones y la eliminación de la última capa de pérdida, como se presenta el código en el anexo C.

- Modelo de la red, el cual fue generado en el apartado de entrenamiento y contiene los pesos aprendidos en la última iteración.

3.4. Métricas de eficiencia

La eficiencia del sistema propuesto se ha medido en base a las siguientes métricas: Intersección Sobre Unión (IoU), Precisión (Pr), Sensibilidad o Recall (Rec), Especificidad (Esp), Confianza y Tiempo de respuesta. Las cuales se obtienen a partir de los índices: verdaderos positivos (VP), verdaderos negativos (VN), falsos positivos (FP) y falsos negativos (FN).

VP es la cantidad de objetos detectados correctamente, VN es el número de objetos verdaderamente detectados como negativos, FP es la cantidad de falsas detecciones positivas, FN es el número de objetos de interés que no fueron detectados.

3.4.1.1. Intersección Sobre Unión

La Intersección Sobre Unión (IoU) controla que el cuadro delimitador encierre en su totalidad al objeto de interés en la imagen y de esta manera el detector encuentra los verdaderos positivos mientras suprime los falsos positivos cercanos. El cálculo del IoU del detector se realiza mediante la fórmula:

$$IoU = \frac{\text{Área de intersección}}{\acute{A}rea\ de\ uni\acute{o}n}$$

En el área de superposición se calcula las coordenadas de intersección de los dos cuadros delimitadores predichos por el detector para cada objeto, tanto de la parte superior-inferior (tb) como izquierda-derecha (lr) para obtener su área, tal y como muestra las ecuaciones 3.4, 3.5 y 3.6:

$$tb = \min(\text{box1[y2]}, \text{box2[y2]}) - \max(\text{box1[y1]}, \text{box2[y1]}) \quad \text{(Ec. 3.4)}$$

$$lr = \min(box1[x2], box2[x2]) - \max(\text{box1[x1]}, \text{box2[x1]}) \quad \text{(Ec. 3.5)}$$

$$Área\ de\ Intersección = \text{tb} * \text{lr} \quad \text{(Ec. 3.6)}$$

Por otro lado, el área de unión se obtiene de la suma de las dos áreas de los cuadros delimitadores, restando la intersección, como se observa en la ecuación 3.7:

$$Área\ de\ unión = \text{áreaBox1} + \text{áreaBox2} - \text{área de intersección} \quad \text{(Ec. 3.7)}$$

El IoU calculado del detector debe ser mayor o igual al valor utilizado de umbral IoU (0,5) para que el cuadro delimitador sea graficado y visualizado en la imagen de salida.

3.4.1.2. Precisión

La precisión mide la fiabilidad del diseño de detección e indica el porcentaje de objetos correctamente detectados, basándose en los verdaderos positivos frente a todas las detecciones marcadas como positivas por el sistema.

La fórmula para el cálculo de la precisión se presenta en la ecuación 3.8:

$$\text{Pr} = \frac{\text{VP}}{\text{VP} + \text{FP}} \quad \text{(Ec. 3.8)}$$

3.4.1.3. Recall

La sensibilidad o Recall es el número de detecciones positivas obtenidas por el sistema, frente a todos los objetos que contiene el dataset, detectados o no, y se calcula mediante la siguiente ecuación 3.9:

$$\text{Rec} = \frac{\text{VP}}{\text{VP} + \text{FN}} \quad \text{(Ec. 3.9)}$$

Su valor representa la capacidad que tiene el diseño para detectar el objeto, haciendo uso de los verdaderos positivos y valores falsos negativos. La gráfica de la curva precisión –

sensibilidad, permite mostrar la compensación entre precisión y sensibilidad del sistema (Massiris, Delrieux, & Fernández, 2018).

3.4.1.4. Especificidad

La especificidad es el número correcto de detecciones negativas obtenidas en la prueba. Por ello también es conocida como Fracción de Verdaderos Negativos (FVN) y se obtiene mediante la ecuación 3.10:

$$\text{FVN} = \frac{\text{VN}}{\text{VN} + \text{FP}} \qquad (\text{Ec. } 3.10)$$

3.4.1.5. Confianza

A partir del IoU, se calcula la confianza de la detección, estos puntajes de confianza determinan que tan correcto y preciso es el cuadro que contiene un objeto. Formalmente la confianza se define de acuerdo a la ecuación 3.11:

$$Conf = \text{Prob}(\text{Objeto}) * IoU \qquad (\text{Ec. } 3.11)$$

Si no existe ningún objeto en esa celda, las puntuaciones de confianza deben ser cero. Con estos valores, se descarta los cuadros delimitadores que tengan un IoU menor a 0.5 y también aquellos que no contengan objetos de detección.

En el proceso de validación, se multiplica las probabilidades de las clases a las cuales un objeto puede pertenecer y las predicciones de confianza de cada cuadro delimitador, de donde se obtiene las puntuaciones de confianza específicas de clase para cada cuadro. Estas puntuaciones codifican la probabilidad de que esa clase aparezca en el cuadro y las correctas dimensiones del cuadro delimitador en el pronosticado objeto, es así como estos datos representan la salida del detector.

3.4.1.6. Tiempo de respuesta

El tiempo de respuesta del sistema corresponde al tiempo de ejecución desde que ingresa la imagen al sistema hasta que se obtiene la imagen de salida detectada. Su cálculo se realiza mediante la ecuación 3.12:

$$t = t_{fin} - t_{inicio} \qquad (Ec. 3.12)$$

Esta métrica permitirá determinar el uso del sistema en aplicaciones de tiempo real.

CAPÍTULO IV: ANÁLISIS DE RESULTADOS OBTENIDOS

En este capítulo se presenta los resultados obtenidos con la red neuronal convolucional YOLO previamente entrenada y validada, además se presenta la eficiencia de este diseño de red, mediante un análisis cualitativo y cuantitativo de métricas en diferentes grupos de imágenes.

Las pruebas experimentales del sistema propuesto han sido ejecutadas en un equipo de 10 Gb de RAM y 500 Gb de Disco duro, procesador Intel core i7 y memoria gráfica de 256 MB.

4.1. Resultados obtenidos

Los resultados obtenidos del apartado anterior, permite observar la detección del objeto en una imagen real. Además de obtener resultados de tiempo de respuesta y la precisión del cuadro delimitador.

4.1.1. Resultados obtenidos del entrenamiento

Al entrenar este diseño de red neuronal artificial, como se menciona en el apartado 3.5.2, después de pasar por sus fases de inicialización, entrenamiento y prueba, se obtiene el modelo entrenado en la carpeta asignada. La salida obtenida es un archivo con extensión .caffemodel, el cual contiene los pesos y bias aprendidos en el proceso de entrenamiento, como se observa en la Figura 15.

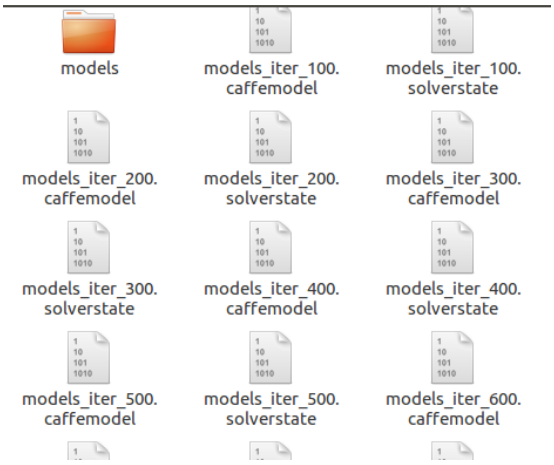


Figura 15. Modelos entrenados

Fuente: Autoría propia

El modelo entrenado, la estructura de la red neuronal y un dataset de prueba, permiten la detección de vehículos con este diseño de red neuronal artificial YOLO.

4.1.2. Resultados obtenidos de la validación

Los resultados obtenidos en la validación se encuentran representados por la salida obtenida de la red neuronal artificial YOLO, los cuales son las coordenadas del cuadro delimitador (x, y, w, h), la clase del objeto detectado (car) y el tiempo de respuesta del detector (t).

Figura 16. Imagen de validación

Fuente: Autoría propia

Utilizando la imagen de la Figura 16, el modelo entrenado, el script de la estructura de la red presentado el código en el anexo C y el script de validación, se obtuvo los resultados

presentados en la consola, como muestra la Figura 17, donde se observa el tiempo de ejecución en detectar el vehículo (t= 6.86 s), el cuadro delimitador con sus coordenadas superior izquierda (x= 115, y=102), el ancho (w=225) y alto (h=109), su valor de confianza de pertenecer a la clase (conf= 0,99) y la clase a la que pertenece la imagen (car).

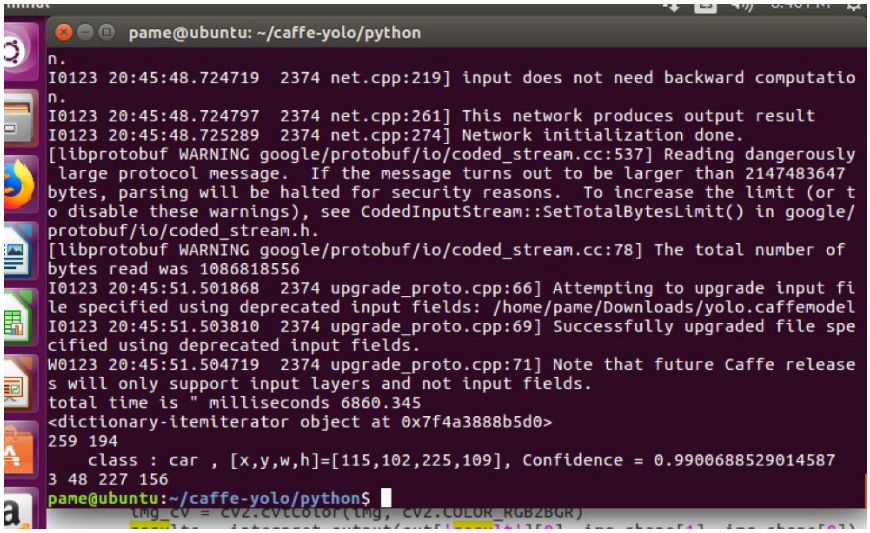


Figura 17. Resultado obtenido prueba 1

Fuente: Autoría propia

La información del cuadro delimitador, la clase y la confianza se visualiza gráficamente en la Figura 18:

Figura 18. Resultado obtenido Prueba 2

Fuente: Autoría propia

4.2. Eficiencia del sistema

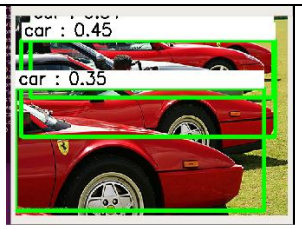
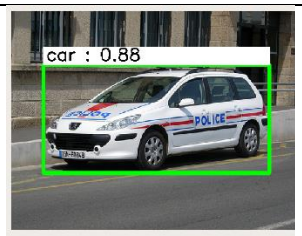
Para comprobar la eficiencia del sistema de reconocimiento de vehículos, se analiza de manera cualitativa y cuantitativa las métricas mencionadas en el apartado 3.4. Haciendo uso de dos grupos, cada una con 100 imágenes: el primer grupo contiene imágenes que no se utilizaron en el entrenamiento de la red neuronal artificial YOLO, pero son parte del dataset PASCAL VOC, el segundo grupo de imágenes son tomadas en el ambiente de la ciudad de Ibarra, es decir, imágenes no vistas antes por la red. Cada imagen RGB, en formato .JPG, contiene diferentes vehículos con diversas perspectivas.

4.2.1. Análisis cualitativo

Los resultados cualitativos obtenidos con el primer grupo de imágenes se presentan en la Tabla 11:

Tabla 11 Resultados Grupo1

Situación	Imagen de entrada	# de vehículos	Imagen de salida	# de vehículos detectados
Vehículos cercanos		1		1
Vehículos lejanos		1		1
Vehículos vista frontal		1		1

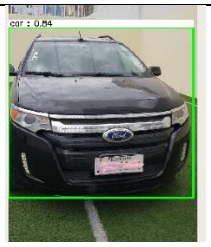
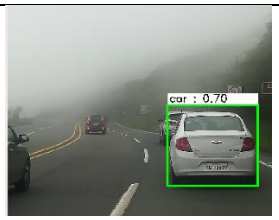
Varios vehículos		5		3
Vehículos vista angular		1		1
Vehículos vista lateral		1		1
Cambios de iluminación		1		1

Fuente: Autoría propia

La tabla presenta los distintos casos de imágenes que se ponen a prueba en la red, la cantidad de vehículos en la imagen de entrada y los resultados obtenidos, como la confianza que se asignó a esa detección y la cantidad de vehículos que se reconocieron en la imagen de entrada.

La Tabla 12 presenta los resultados cualitativos alcanzados en el grupo2:

Tabla 12 Resultados Grupo 2

Situación	Imagen de entrada	# de vehículos	Imagen de salida	# de vehículos detectados
Vehículos cercanos		1		1
Vehículos lejanos		2		1
Vehículos vista frontal		1		1
Vehículos vista angular		3		2
Vehículos vista lateral		1		1
Cambios de iluminación		2		1

Varios vehículos		4		3

Fuente: Autoría propia

En ambos conjuntos de datos se observa la detección vehicular, representada por un cuadro delimitador de color verde alrededor del vehículo. En las Tablas 11 y 12 se visualiza que los vehículos son correctamente detectados en imágenes con vehículos cercanos, vista frontal, lateral y angular, sin embargo, no se detectaron correctamente en situaciones como presencia de neblina o lluvia, vehículos lejanos, varios vehículos o aquellos que se encuentran incompletos por oclusión, con ello, se puede decir que el detector cumple con su función en ciertas situaciones que determinan el ambiente pero es importante conocer que tan preciso resulta el diseño y esto se logra con el análisis cuantitativo.

4.2.2. Análisis cuantitativo

Los resultados obtenidos en VP, VN, FP, FN, Pr, t, Rec y FVN en cada una de las imágenes del grupo 1 y grupo 2, han sido promediadas y se presentan en la Tabla 13:

Tabla 13 Evaluación de métricas

Dataset	**Métricas**							
	VP	**VN**	**FP**	**FN**	**Pr**	**Rec**	**FVN**	**T (s)**
Grupo 1	128	80	2	34	0,9846	0,7901	0,9756	2,59
Grupo 2	82	51	2	60	0,9764	0,5774	0,9622	2,70

Fuente: Autoría propia

Las métricas medidas en la validación presentan un valor mayor de verdaderos positivos y verdaderos negativos para el grupo 1, lo que significa que el diseño es capaz de obtener detecciones correctas, por otro lado, el grupo 2 alcanza un número mayor de objetos de interés no detectados, aumentando así los falsos negativos y los falsos positivos se mantienen en una cifra igualitaria para ambos grupos.

Con estos resultados se observa que la precisión es menor para el segundo grupo debido a la cantidad de verdaderos positivos obtenidos, sin embargo, la diferencia entre ambos grupos es de 0,01. Así mismo, la especificidad calculada para ambos grupos mantiene una diferencia de la misma proporción de la precisión, pero la sensibilidad del detector en el grupo 1 aumenta un 0,2 del grupo 2.

De esta forma, el diseño del detector presenta una convergencia entre precisión, especificidad y sensibilidad, con valores altos para Pr y FVN sin perjudicar el valor de Rec. Logrando un valor de sensibilidad que reduce los falsos positivos para no dificultar la identificación del objeto de interés.

El tiempo de respuesta promedio del detector para el grupo 1 y 2 es de 2, 65 segundos, lo cual, representa 4,4 % de un minuto. La Agencia Nacional de Tránsito afirma que el tiempo máximo de reacción para evitar una colisión debe ser menor al 10% de un minuto, por lo que el diseño de este sistema se considera aceptable para aplicaciones de seguridad vehicular.

El valor de IoU y conf en 4 situaciones reales, se presenta en la Tabla 14:

Tabla 14 Promedio de IoU y Confianzas

Situación	Grupo 1		Grupo 2	
	IoU	**Conf**	**IoU**	**Conf**
Vehículos lejanos	0,75	0,50	0,65	0,42
Vehículos cercanos	0,90	0,89	0,84	0,80
Vehículo ocluido	0,68	0,31	0,61	0,35
Cambios de iluminación	0,79	0,58	0,57	0,30

Fuente: Autoría propia

Los valores de IoU presentados en la Tabla 14, tanto para el grupo de imágenes 1 y 2, varían de acuerdo a las situaciones de cada imagen. El grupo 1 obtiene como valores mínimos promedio 0,68 en IoU y 0,31 en Confianza en ambientes con vehículos ocluidos, así mismo, logra los valores máximos promedio de 0,90 de IoU y 0,89 de Confianza en imágenes con vehículos cercanos y visibles. Por otro lado, El grupo 2 presenta valores mínimos promedio de 0,57 de IoU y 0,30 de Confianza en imágenes con cambios de iluminación, y valores máximos promedio en detecciones de vehículos cercanos de 0,84 en IoU y 0,80 en Confianza.

Por lo tanto, el diseño es capaz de ajustar el cuadro delimitador de manera eficiente con valores de confianza altos en escenarios con vehículos que se encuentren a distancias cortas y presenta dificultades en identificar vehículos obstruidos o incompletos y en ambientes con neblina, lluvia y obscurecimiento.

4.2.2.1. Gráfica Precisión-Recall

Los valores obtenidos en precisión frente a sensibilidad, se presentan en la Figura 22:

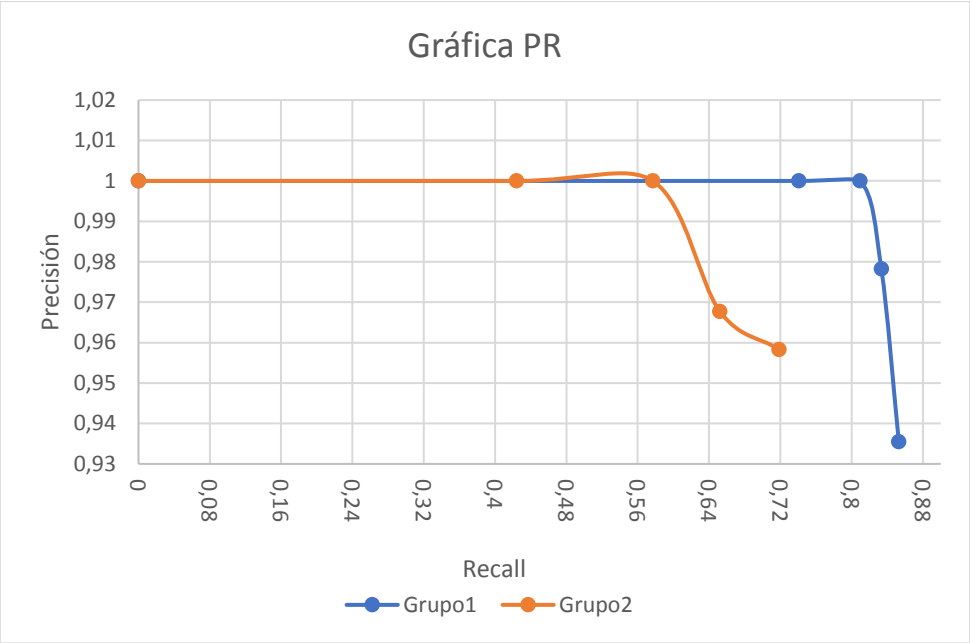


Figura 19 Gráfica Precisión-Sensibilidad

Fuente: Autoría propia

Como se puede observar en las gráficas, la precisión disminuye cuando la sensibilidad aumenta, es por ello que, un diseño con alta capacidad de detectar objetos de interés frente a todos los objetos contenidos en las imágenes, pierde la precisión con la que son reconocidos. El resultado de este diseño muestra una precisión y sensibilidad equilibrada, llegando a tener una precisión entre 93% a 100% con sensibilidad máxima de 86% en sus detecciones.

El área bajo la curva de la gráfica Precisión-sensibilidad, representa el valor de la Precisión Promedio (AP). Cuanto más se acerque este valor a 1, mejor será el rendimiento del diseño del sistema de reconocimiento automático.

El AP del grupo 1 es mayor al del grupo 2, pues se obtiene el valor de 0,84 y 0,71 respectivamente. Logrando un mejor desempeño de la red en detección de vehículos que pertenezcan al Dataset con el que fue entrenado, sin embargo, el valor de la precisión promedio de imágenes en ambientes de la ciudad de Ibarra disminuye apenas en un 0,1, demostrando que para ambos casos el diseño presenta un rendimiento mayor al 50%.

4.2.2.2. Curva ROC

La curva de ROC se obtiene de la relación que existe entre la sensibilidad y los fallos obtenidos en las detecciones (1-especificidad), de modo que, indica la capacidad de detección que tiene el diseño. La curva de ROC obtenida para cada grupo de imágenes se muestra en la Figura 23:

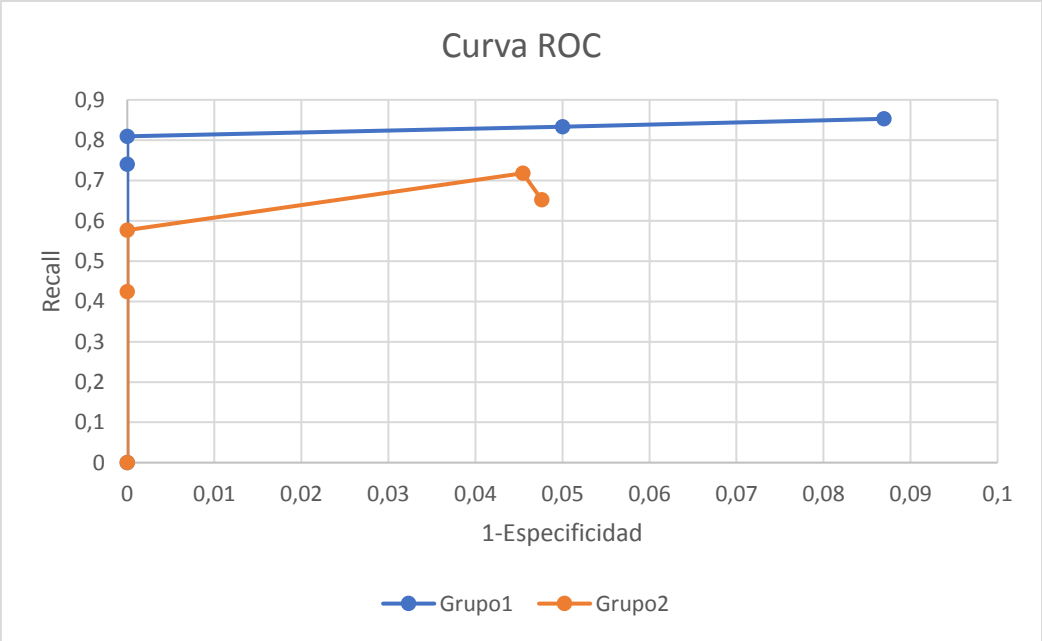


Figura 20 Curva ROC

Fuente: Autoría propia

De acuerdo a López (2011), la curva ROC de un sistema de detección de objetos debe tener sus valores cercanos a la esquina superior izquierda y, dependiendo de ello, se puede valorar la validez y seguridad de las detecciones del diseño. La curva correspondiente al grupo 2 muestra una alta sensibilidad y especificidad, sin embargo, los valores mejoran para el grupo 1.

La distribución entre detecciones positivas y negativas se encuentran bien diferenciadas en la curva ROC del diseño, la cual, es muy próxima al punto (0,1) y representa resultados erróneos bajos.

4.2.3. Limitaciones

En base al análisis cualitativo y cuantitativo se han determinado las limitaciones que presenta el diseño.

- Oclusión:

La oclusión ocurre cuando el objeto de interés se encuentra detrás de otro, presentando una pérdida de datos debido a que únicamente se puede extraer las características de la parte que se puede ver del objeto.

a) *b)*

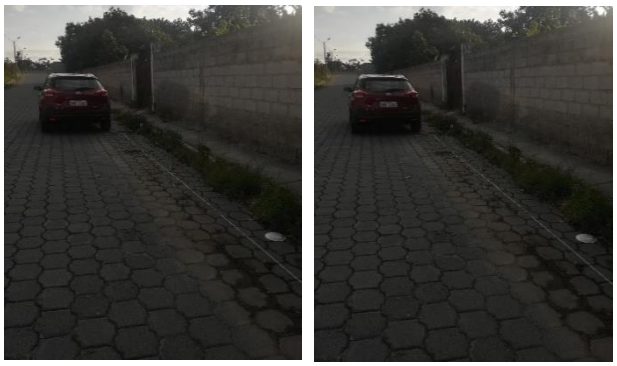
Figura 21: Oclusión a) Imagen de entrada b) Imagen de salida

Fuente: Autoría propia

Como se visualiza en la Figura 24, a entrada (24a) contiene 5 vehículos, de los cuales 3 se encuentran fácilmente visibles y 2 están detrás de los otros; la salida (24b) presenta únicamente 3 vehículos detectados por lo que el detector no reconoce características de los otros 2 para ser identificados. Presentando el diseño errores de detección en imágenes que tengan vehículos con oclusión.

- Cambios de iluminación:

La imagen puede resultar distorsionada y no tan clara debido a los cambios de tonos de luz, las cuales crean sombras o reflejos que impiden una visión idónea de la imagen para procesar. Las imágenes de la Figura 25 presentan la entrada y salida de esta situación:



a) *b)*

Figura 22: Cambios de iluminación a)Imagen de entrda b)Imagen de salida

Fuente: Autoría propia

La entrada (25a) es una imagen en ambiente con poca iluminación y la salida del sistema (25b) no obtiene ninguna detección de vehículos presentes, por lo tanto, el detector presenta limitaciones en cambios de iluminación.

- Distintas distancias y escalas:

La extracción de características para objetos de interés que se encuentren a varias distancias y escalas dependen de la profundidad de la red neuronal artificial, utilizando así mayor cantidad de recursos computacionales.

a) *b)*

Figura 23: Distintas distancias y escalas a) Imagen de entrada b) Imagen de salida

Fuente: Autoria propia

La imagen de entrada (26a) presenta varios vehículos a distintas distancias desde el punto de vista de la cámara estática, lo que conlleva tener vehículos a diferentes tamaños. La imagen de salida (26b) presenta la detección obtenida por el sistema, donde no se identifica los vehículos que tienen escalas muy grandes o muy pequeñas. Esta limitación se presenta en imágenes con objetos que tienen cambios de escala, lo cual, omite la detección de algunos vehículos en los resultados obtenidos.

Para comprobar la distancia de detección que el diseño llega a tener, se realizó pruebas experimentales con vehículos que se encontraban a distintos metros de distancia en ambientes ideales (sin neblina, lluvia, etc.), como se muestra en la Tabla 15:

Tabla 15 Resultados prueba de distancia

# de imágenes	Distancia del vehículo	VP	VN
10	5 m	10	0
10	10 m	10	0
10	15 m	10	0
10	20 m	10	0
20	25 m	11	9
15	30 m	0	15

Fuente: Autoría propia

El diseño presentado consigue detecciones acertadas a 5, 10, 15 y 20 metros de distancia, obteniendo en todas sus pruebas verdaderos positivos. Por otro lado, a 25 y 30 metros de distancia el sistema presenta detecciones erróneas, para el primer caso se adquiere como resultados verdaderos positivos y verdaderos negativos, a diferencia del segundo caso, en el que se pierde la detección completamente.

Partiendo de la velocidad máxima permitida en zona urbana (50 km/h) del Reglamento a Ley de Transporte Terrestre, Tránsito y Seguridad Vial (2012), la distancia mínima permitida entre vehículos, de acuerdo a la regla del cuadrado, es de 25 metros. Por lo tanto, el detector requiere una detección oportuna de vehículos que se encuentren a distancias iguales o menores a 25 metros, así el diseño permite desarrollar este tipo de aplicaciones y reconocer vehículos que se consideran peligrosos frente a posibles colisiones.

Una vez presentado el análisis cualitativo y cuantitativo, el diseño de este sistema de reconocimiento automático con la red neuronal YOLO cuenta con una precisión máxima promedio de 0,85 para detección de vehículos hasta distancias de 25 metros. El cual, resulta ser adecuado para aplicaciones de seguridad vehicular que se basan en el reconocimiento y detección de vehículos que incumplan con la norma de distancia entre vehículos.

CONCLUSIONES

- Las redes neuronales convolucionales han permitido ampliar la gama de aplicaciones en reconocimiento y procesamiento de imágenes debido a su precisión, obteniendo resultados similares a los de arquitecturas que ocupan mayor cantidad de recursos computacionales.
- La capacidad de extracción de características es un punto clave para el aprendizaje de una red neuronal, mientras más profundidad tenga en su capa oculta, mejor será su precisión debido a que los datos pasan por varios procesos de reconocimiento de patrones.
- La utilización de sistemas embebidos en una aplicación permite que esta sea accesible y práctica para el usuario, es por ello que, es importante el análisis y estudio de arquitecturas alternativas que generen un bajo coste computacional y se ejecuten en plataformas con limitados recursos de memoria y procesamiento, tal y como es YOLO.
- Una parte fundamental e indispensable para la detección de objetos en una red neuronal artificial es la elección y preparación del conjunto de datos que se va a utilizar, debido a que de este depende el aprendizaje sobre el objeto, por lo tanto, el conjunto de imágenes debe ser estudiado de acuerdo a su aplicación y diseño.
- La preparación del conjunto de imágenes que se va a procesar en una red neuronal artificial, dependerá de la arquitectura de red que se utilizará, dado que, se debe adaptar a los requisitos de entrada de la misma.
- El dataset VOC PASCAL es un conjunto de datos muy útil para sistemas de detección de objetos con redes neuronales artificiales, puesto que contiene información completa de las características de sus imágenes, lo cual permite desarrollar aplicaciones en distintos ámbitos.

- Utilizar un gran dataset para el entrenamiento permite mejorar la precisión de la detección, es decir, el sistema tendrá errores de reconocimiento si es entrenada con pocas imágenes, por otro lado, el incremento de las mismas requiere de un equipo con altas prestaciones como es un supercomputador.
- Una Red neuronal artificial es procesada en dos fases fundamentales, la primera es su entrenamiento y la segunda es su validación, en ambas se utiliza conjuntos de datos distintos debido a que en su entrenamiento necesita agrupar imágenes para la extracción de características e imágenes para probar su entrenamiento en el proceso y en la validación se necesita poner a prueba a la red entrenada con un conjunto de datos nunca antes visto.
- La utilización del software libre Caffe dentro del aprendizaje profundo, facilita el desarrollo y despliegue de cualquier tipo de investigación por su velocidad y cantidad de procesamiento, además proporciona la información y documentación necesaria del mismo.
- El software de tratamiento de imágenes Caffe cuenta con herramientas utilizables en el proceso del conjunto de datos para su ingreso a la red neuronal artificial, ayudando a cambiar su tamaño, el formato, aumentar datos con variaciones específicas, entre otros.
- El método de descenso del gradiente estocástico mejora la eficiencia del diseño de la red neuronal artificial, debido a que evita el excesivo procesamiento y disminuye el tiempo de respuesta del sistema, los cuales, son partes fundamentales para una correcta detección de objetos
- El uso de la transferencia de conocimiento en redes neuronales ayuda a que el sistema evite realizar iteraciones ineficientes, es decir, evade la actualización de pesos y bias erróneos mediante la inicialización de pesos ya entrenados en otras aplicaciones con la misma arquitectura.

- La forma en la que se utilizan los parámetros y el orden de cada capa de una red neuronal convolucional es lo que permite mejorar su resultado de aprendizaje, ya que de esto depende la cantidad de operaciones en la red, el tamaño de la imagen en cada capa, el número de activaciones de salida, entre otros.
- La utilización de un umbral de intersección sobre unión, para la detección, puede beneficiar como afectar a la precisión del reconocimiento del objeto a causa de que de este umbral depende la eliminación de cuadros delimitadores para que no se reconozca al mismo objeto en dos cuadros distintos.
- La red neuronal convolucional YOLO presenta en sus resultados valores altos de precisión y especificidad, siendo así, una arquitectura que aprovecha el rendimiento de la red en reconocimiento automático de objetos.

RECOMENDACIONES

- Es sustancial la búsqueda, análisis y estudio minucioso del dataset que se va a emplear en un sistema de red neuronal artificial con alguna aplicación específica para cerciorarse que contenga todo lo necesario en la entrada de la red.
- Es importante instalar, revisar y estudiar las herramientas necesarias del Software de tratamiento de imágenes para que cumplan con su correcto funcionamiento y evitar posibles errores en su ejecución.
- Es recomendable el uso de equipos con al menos 8GB de capacidad de memoria RAM para el entrenamiento de un sistema como el planteado en este escrito y así evitar demoras en esta fase.
- Se aconseja estudiar a fondo la estructura de arquitecturas de redes neuronales artificiales para poder entender su funcionamiento y su proceso capa por capa, además de comprender los parámetros que se deben definir en cada tipo de capa.
- Es recomendable usar un gran volumen de datos para el entrenamiento de la red, de esta forma el sistema aprende más características del objeto y sus pesos y bias mejoran, aumentando la precisión de la detección.
- Una red neuronal YOLO presenta altos valores de precisión con objetos cercanos, bajo procesamiento y cortos tiempos de respuesta, por lo cual es una excelente alternativa para aplicaciones de seguridad vehicular, por ejemplo, en un sistema de asistencia al conductor (ADAS), en el que se necesita detectar vehículos cercanos a su alrededor.

REFERENCIAS

Álvarez, D., & Giraldo, E. (2008). ICA aplicado a la extracción de características en imágenes. *Scientia Et Technica*, 43-48.

Alvear, V. (2016). *Sistema electrónico con aplicación IOT para monitoreo facial.* Ibarra, Ecuador: Universidad Técnica del Norte.

Amoroso, M., & Avila, H. (2015). Aplicación de las técnicas de agrupamiento para la distribución Cuasi-Optima de una red Hibrida WDM-TDM/PON en cascada multinivel que da soporte a una Smart City. Ecuador: Universidad Politécnica Salesiana SEDE Cuenca.

Aplicaciones reales de Deep Learning. (21 de julio de 2016). Obtenido de PrNoticias: https://prnoticias.com/comunicacion/prcomunicacion/20154951-6-aplicaciones-reales-del-deep-learning

Bergamini, M., & Kamlofsky, J. (2015). Estudio de descriptores geométricos para el diseño y optimización de algoritmos de reconocimiento de objetos digitales. Argentina: Universidad abierta Interamericana.

Blanco, S. (30 de Octubre de 2013). *Manual básico Ubuntu GNU/Linux*. Obtenido de Universidad Luterana Salvadoreña: http://www.uls.edu.sv/pdf/ubuntu.pdf

Bouza, C., & García, S. (2012). La minería de datos: Arboles de decisión y su aplicación en estudios médicos. En *MODELACIÓN MATEMÁTICA DE FENÓMENOS DEL MEDIO AMBIENTE Y LA SALUD* (págs. 64-78). Cuba: Dirección de Calidad y Enseñanza en Salud.

Caballero, E. (2017). *Aplicación práctica de la visión artificial para el reconocimiento de rostros en una imagen, utilizando redes neuronales y algoritmos de reconocimiento de objetos de la biblioteca opencv.* Bogotá, Colombia: Universidad Distrital Francisco José De Caldas.

Caballero, H. (2001). Modelado y seguimiento de objetos por medio de distribución del color. Mexico.

Cáceres, J. (2010). Reconocimiento de patrones y el aprendizaje no supervisado. España: Universidad de Alcalá.

Calvo, D. (20 de julio de 2017). *Red neuronal Convolucional CNN*. Obtenido de Diego Calvo: http://www.diegocalvo.es/red-neuronal-convolucional-cnn/

Castillo, D., & Herrera, R. (2013). Análisis de los factores que inciden en los accidentes de tránsito del servicio de transportación pública interprovincial en el ecuador. Guayaquil.

Ciriza, R., Albizua, L., & Gonzales, M. (2009). Análisis de la utilidad de los descriptores texturales de haralick para la localización arranques de frutal en ortofoto. *XIII Congreso de la Asociación Española de Teledetección*, (págs. 597-600). España.

Clint, S. (19 de Noviembre de 2015). *Aplicaciones interesantes de detección de objetos*. Obtenido de Quora: https://www.quora.com/What-are-some-interesting-applications-of-object-detection

Corrales, A., & Rivas, R. (2007). Sistema de identificación de objetos mediante RFID para un robot personal. España: Universidad Carlos III de Madrid.

De la Escalera, A. (2017). *Visión por computador.* España: Segunda Edición.

Deepak, K. (17 de Marzo de 2018). *Military Applications Of Artificial Intelligence*. Obtenido de Centre For Land Warfare Studies: http://www.claws.in/1878/military-applications-of-artificial-intelligence-deepak-kumar-gupta.html

Del Valle, A. (2017). Curvas ROC (Receiver-Operating-Characteristic) y sus aplicaciones. España: Universidad de Sevilla.

Diaz, E. (2006). Localización y reconocimiento automático del numero de la placa de un automóvil. Perú: Pontifica Universidad Católica del Perú.

Domínguez, E., & Padilla, D. (2001). Regresión Logística: un ejemplo de su uso de endocrinología. Cuba: Instituto Nacional de Endocrinología.

Estadísticas de transporte terrestre y seguridad vial. (2017). Obtenido de Agencia Nacional de Tránsito: https://www.ant.gob.ec/index.php/noticias/estadisticas

Everingham, M., Van Gool, L., Williams, C., Winn, J., & Zisserman, A. (9 de Septiembre de 2009). *The PASCAL Visual Object Classes (VOC)*. Obtenido de Springer Science+Business Media: http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham10.pdf

Everingham, M., Van Gool, L., Williams, C., Winn, J., & Zisserman, A. (2010). International Journal of Computer Vision. 303-338.

Forson, E. (18 de Noviembre de 2017). *Understanding SSD MultiBox—Real-Time Object Detection In Deep Learning.* Obtenido de Towards Data Science: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab

Galán, H., & Martínez, A. (2015). Inteligencia artificial: Redes neuronales y aplicaciones. España: Universidad Carlos III de Madrid.

Gamarra, C., & Ríos, M. (2018). *Aplicación de técnicas de aprendizaje profundo para la clasificación y reconocimiento de objetos en imágenes.* Colombia: Universidad Santo Tomás.

Gao, H. (6 de Junio de 2018). *Understand Single Shot MultiBox Detector (SSD) and Implement It in Pytorch.* Obtenido de Medium: https://medium.com/@smallfishbigsea/understand-ssd-and-implement-your-own-caa3232cd6ad

García, P. (2013). Reconocimiento de imágenes utilizando redes neuronales artificiales. España: Universidad complutense de Madrid.

Gauen, K., Dailey, R., Laiman, J., Zi, Y., & Asokan, N. (2017). *Comparison of Visual Datasets for Machine.* Estados Unidos: Loyola University Chicago.

González, H., & Giraldo, B. (2015). Diseño de un clasificador para pacientes en proceso de extubación. *ITECKNE*, 131-137.

González, P., Diaz, A., Torres, E., & Garnica, E. (1994). Aplicación del análisis de componentes principales en el área educativa. Venezuela: Instituto de Investigaciones Económicas y Sociales.

González, R. (2007). El Test de Turing: dos mitos. *Revista de Filosofia ,* Volumen 63.

Gravano, A., & Mislej, E. (2015). *Aprendizaje no supervisado*. Obtenido de Universidad de Buenos Aires: http://www.dc.uba.ar/materias/aa/2015/cuat2/clustering

Guachi, L. (2016). Background Subtraction for moving object detection. Italia: Universidad de Calabria.

Guachi, L., Guachi, R., Perri, S., Corsonello, P., Bini, F., & Marinozzi, F. (2018). Automatic Microstructural Classification with Convolutional Neural Network. En TIC.EC, *Information and Communication Technologies of Ecuador* (págs. 170-181).

Guanghan, N. (22 de Diciembre de 2015). *Start Training YOLO with Our Own Data.* Obtenido de http://guanghan.info/blog/en/my-works/train-yolo/

Günel, M. (2016). *GoogleNet*. Obtenido de https://pdfs.semanticscholar.org/0b99/d677883883584d9a328f6f2d54738363997a.pdf

Hernández, C., & Rodríguez, J. (2016). Algoritmo híbrido basado en aprendizaje computacional para el manejo de datos faltantes en aplicaciones OLAP. *Ingeniare. Revista Chilena de Ingeniería*.

Honarkhah, M., & Caers, J. (2010). Stochastic Simulation of Patterns Using Distance-Based Pattern Modeling. En *Mathematical Geosciences* (págs. 487–517). Estados Unidos: Springer.

Hui, J. (27 de Marzo de 2018). *Detección de objetos: comparación de velocidad y precisión* . Obtenido de Medium Corporation: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

Hui, J. (17 de Marzo de 2018). *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3*. Obtenido de Medium Corporation: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

Jiang, Q., Cao, L., Cheng, M., Wang, C., & Li, J. (2015). Deep neural networks-based vehicle detection in satellite images. *International Symposium on Bioelectronics and Bioinformatics (ISBB)*, 184-187.

Jones, T. (27 de Agosto de 2018). *Deep learning architectures*. Obtenido de IBM Developer Technology: https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/

Khan, M. (11 de Abril de 2018). *Implementing YOLO using ResNet as Feature extractor*. Obtenido de Medium.com: https://medium.com/@m.khan/implementing-yolo-using-resnet-as-feature-extractor-5857f9da5014

Lamos, H., Galvan, S., Gonzalez, L., & Cruz, C. (2013). Algoritmo PSO-Híbrido para solucionar el problema de ruteo de vehículos con entrega y recolección simultáneas. *Revista Facultad de Ingeniería, UPTC*, 75-90.

Lampert, C. (2017). *AlexNet*. Obtenido de Institute of Science and Technology Austria: http://cvml.ist.ac.at/courses/DLWT_W17/material/AlexNet.pdf

Lecun, Y. (2015). *LeNet-5: convolutional neural networks*. Obtenido de Yann LeCun´s: http://yann.lecun.com/exdb/lenet/

Li, F.-F., Johnson, J., & Yeung, S. (2017). Lecture 9: CNN Architectures. Estados Unidos: Stanford University .

Liao, M. (1 de Marzo de 2018). *Detecting Objects in (almost) Real-time: FasterRCNN Explained with Code.* Obtenido de Towards Data Science: https://towardsdatascience.com/fasterrcnn-explained-part-1-with-code-599c16568cff

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. (2016). SSD: Single Shot MultiBox Detector. EE.UU.: University of North Carolina.

Llano, L., Hoyos, A., Arias, F., & Velásquez, J. (2007). Comparación del Desempeño de Funciones de Activación en Redes Feedforward para aproximar Funciones de Datos con y sin Ruido. *Revista Avances en Sistemas e Informática*, 80-88.

López, M. (2011). *Sistemas de reconocimiento de patrones híbrido inteligente.* España: Editorial Académica Española.

López, M. (16 de Mayo de 2016). *Una aplicación fantástica de las redes neuronales profundas*. Obtenido de Unocero: https://www.unocero.com/ciencia/una-aplicacion-fantastica-de-las-redes-neuronales-profundas/

Malavé, J., Márquez, E., & Lezama, J. (2015). Cálculo de descriptores moleculares topológicos en flavonoides con actividad ANTI-VIH-1. *SABER. Revista Multidisciplinaria del Consejo de Investigación de la Universidad de Oriente*, 102-109.

Martínez, F., Gómez, F., & Romero, E. (2009). Análisis de vídeo para estimación del movimiento humano. Colombia: Universidad Nacional de Colombia.

Massiris, M., Delrieux, C., & Fernández, Á. (7 de Septiembre de 2018). DETECCIÓN DE EQUIPOS DE PROTECCIÓN PERSONAL MEDIANTE RED NEURONAL CONVOLUCIONAL YOLO. *Actas de las XXXIX Jornadas de Automática.* España: Universidad de Extremadura.

Matich, D. (2001). Redes Neuronales: Conceptos Básicos y Aplicaciones. Argentina: Universidad Tecnológica Nacional.

Menegaz, M. (28 de Marzo de 2018). *Understanding YOLO.* Obtenido de Hackernoon: https://hackernoon.com/understanding-yolo-f5a74bbc7967

Montaño, J. (2002). Redes Neuronales Artificiales aplicadas al análisis de datos. España: Universidad de las Islas Baleares.

Montes, S. (2016). Diferencias individuales y correlatos psicológicos de los errores relacionados con la inatención en conductores. Universidad Nacional de Mar de la Plata, Argentina.

Moralejo, R., & Storni, A. (2018). Plataforma Detección de objetos en tiempo real. *XX Workshop de Investigadores en Ciencias de la Computación.* Argentina: Universidad Nacional de la Plata.

Moya, F., Herrero, V., & Guerrero, V. (1998). La aplicación de Redes Neuronales Artificiales. En P. Cid, *Anuario SOCADI de Documentación e Información* (págs. 147-163). España.

Muñoz, J., Rivera, J., & Duque, E. (2008). Análisis de componentes principales e independientes aplicados a reducción de ruido en señales electrocardiográficas. *Scientia Et Technica, vol. XIV*, 83-88.

Nuñez, F. (2016). Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional. España: Universidad Internacional de Cataluña.

Núñez, F. (2016). Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional. *Universidad abierta de Cataluña, España*.

Olivares, B. (2014). Aplicación del Análisis de Componentes Principales(ACP) en el diagnóstico socioambiental, Sistema de Información Científica. *Red de Revistas Científicas de América Latina y el Caribe*, 364-374.

Olveres, J., Nava, R., & Escalante, B. (2016). Segmentación de Imágenes Médicas Mediante Descriptores de Textura. *Jornadas de Investigación en Sistemas Biomédicos.* México.

OpenCV 3.4.1 . (27 de Febrero de 2018). Obtenido de Opencv Web site: https://opencv.org/about.html

Oropeza, C. (2007). *Modelado y Simulación de un Sistema de Detección de Intrusos Utilizando Redes Neuronales Recurrentes.* Mexico: Universidad de las Américas Puebla.

Ortega, J., Reyes, G., Cruz, L., & Pozos, R. (2007). Mejora al algoritmo de agrupamiento K-means mediante un nuevo criterio de convergencia y su aplicación a bases de datos poblacionales de cáncer. *LAIO.* México.

Pacheco, S., & Díaz, L. (2005). El clasificador Naïve Bayes en la extracción de conocimiento de bases de datos. México.

Paci, F., Brunelli, D., & Benini, L. (2014). *A classroom occupancy monitoring system for smart public buildings.* España: Conferencia en Madrid.

Parekh, H., Thakore, D., & Jaliya, U. (2014). A Survey on Object Detection and Tracking Methods. *International Journal of Innovative Research in Computer and Communication Engineering*, 2970-2978.

Pérez, J. (2002). *Modelos predictivos basados en redes neuronales recurrentes de tiempo discreto.* España: Universidad de Alicante.

Pinto, L. (2015). Análisis de la aplicación de algoritmos de K-means y Continuous Max-Flow a la segmentación de imágenes en color. España: Universidad de Sevilla.

Prakash, J. (7 de Febrero de 2017). *Understanding and Implementing Architectures of ResNet and ResNeXt for state-of-the-art Image Classification*. Obtenido de Medium: https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624

Ramírez, A. (2010). Modelo hibrido de clasificación basado en Algoritmos Genéticos y Máquinas de Vectores de Soporte aplicado a la evaluación crediticia. Colombia: Universidad Nacional de Colombia.

Redmon, J., & Farhadi, A. (8 de Abril de 2018). *YOLOv3: An Incremental Improvement*. Obtenido de Cornell University Library: https://arxiv.org/abs/1804.02767

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection.* Obtenido de University of Washington Computer Science & Engineering community.: https://homes.cs.washington.edu/~ali/papers/YOLO.pdf

Reglamento a Ley de Transporte Terrestre , Tránsito y Seguridad Vial. (25 de junio de 2012). Obtenido de Ministerio de Transporte y Obras Públicas: https://www.obraspublicas.gob.ec/wp-content/uploads/downloads/2018/06/LOTAIP_05_REGLAMENTO-LEY-ORGANICA-DE-TRANSPORTE-TERRESTRE-TRANSITO-Y-SEGURIDAD-VIAL.pdf

Restrepo, G. (2015). Aplicación del aprendizaje profundo ("deep learning") al procesamiento de señales digitales. Colombia: Universidad Autónoma De Occidente.

Reyes, J., Escobar, C., Duarte, J., & Ramírez, P. (2007). Una aplicación del modelo de regresión logística en la predicción del rendimiento estudiantil. Chile: Universidad de Antofagasta.

Rivas, J. (2008). Sistema de Ayuda a la Toma de Decisiones basado en Árboles de Decisión. España: Universidad Complutense de Madrid.

Rodríguez, M. (2017). Detección automática de glóbulos rojos mediante la transformada de Hough. España: Universidad de Málaga.

Rojas, C. (2018). *Despliegue de los servicios telemáticos de la facultad de ingeniería en ciencias aplicadas de la UTN sobre la nube de red CEDIA.* Ibarra, Ecuador: Universidad Técnica del Norte.

Romero, J., Dafonte, C., & Gómez, A. (2007). *Inteligencia Artificial y Computación Avanzada.* Portugal: Fundación Alfredo Brañas.

Rosebrock, A. (7 de Noviembre de 2016). *Intersection over Union (IoU) for object detection*. Obtenido de pyimagesearch: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Ruiz, L. (2013). Análisis de componentes principales: aplicación en Teledetección. España: Universidad Politécnica de Valencia.

Salomón, N., Misller, V., & Miranda, R. (2015). Análisis digital de descriptores de color en trigo. *Revista Internacional de Botánica Experimental*, 306-311.

Sánchez, R. (2014). *Detección jerárquica de grupos de personas.* España: Universidad Autónoma de Madrid.

Sandoval, D. (2014). Análisis de componentes independientes aplicado al estudio de la actividad cerebral. Colombia: Universidad Nacional de Colombia.

Sanz, J. (2008). Reconocimiento de objetos por descriptores de forma. España: Universidad de Barcelona.

Seguridad vial en el mundo. (2017). Obtenido de Organización Mundial de la Salud: http://www.who.int/es/

Sistemas Expertos e Inteligencia artificial. (2016). Obtenido de Universidad Don Bosco: http://www.udb.edu.sv/udb/archivo/guia/informatica-ingenieria/sistemas-expertos-e-inteligencia-artificial/2016/i/guia-9.pdf

Solera, P., Moreira, I., & Hernández, J. (2014). Descriptores botánicos para caracterizar germoplasmas de Ricinus communis de diferentes zonas de Costa Rica. *Revista Tecnología en Marcha*, 37-46.

Suca, C., Córdova, A., Condori, A., Cayra, J., & Sulla, J. (2016). Comparación de algoritmos de clasificación para la predicción de casos de obesidad infantil. Perú: Universidad Nacional de San Agustín.

Teledetección: clasificación y detección. (2003). *Apuntes de la asignatura Teledetección: tema 9*. España: Universidad de Murcia.

Timarán, R., Calderón, A., & Hidalgo, A. (2017). Aplicación de los árboles de decisión en la identificación de patrones de lesiones fatales. Colombia.

Toshev, A., & Szegedy, C. (2013). *DeepPose: Human Pose Estimation via Deep Neural Networks*. Obtenido de https://static.googleusercontent.com/media/research.google.com/es//pubs/archive/42237.pdf

Troncoso, A. (2007). Predicción basada en vecinos. España: Universidad Politécnica de Cataluña.

Vega, H., Cortez, A., Alarcón, L., & Romero, P. (2009). *Vega H., Cortez A, Alarcón Reconocimiento de patrones mediante redes neuronales artificiales.* Perú: Universidad Nacional Mayor de San Marcos.

Vizcaya, R. (2018). Deep Learning para la Detección de Peatones y Vehículos. México: Universidad Autónoma Del Estado De México.

Yangqing, J., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., . . . Darrell, T. (20 de Junio de 2014). *Caffe: Convolutional Architecture for Fast Feature Embedding*. Obtenido de Cornell University Library: https://arxiv.org/abs/1408.5093

ANEXOS A: CÓDIGO FUENTE DEL TRATAMIENTO DE LOS DATOS

A.1. Lista.py: Recopilación de información del dataset

```
import os   #Cargar módulo de Python

trainval_jpeg_list = []
trainval_xml_list = []
test07_jpeg_list = []
test07_xml_list = []
test12_jpeg_list = []

#Recopilar información de cada carpeta del Dataset

for name in ["VOC2007"]:
  voc_dir = os.path.join("VOCdevkit", name)
  txt_fold = os.path.join(voc_dir, "ImageSets/Main")
  jpeg_fold = os.path.join(voc_dir, "JPEGImages")
  xml_fold = os.path.join(voc_dir, "Annotations")
  for t in ["train.txt", "val.txt"]:
    file_path = os.path.join(txt_fold, t)
    with open(file_path, 'r') as fp:
      for line in fp:
        line = line.strip()
        trainval_jpeg_list.append(os.path.join(jpeg_fold,
"{}.jpg".format(line)))
        trainval_xml_list.append(os.path.join(xml_fold,
"{}.xml".format(line)))
        if not os.path.exists(trainval_jpeg_list[-1]):
          print trainval_jpeg_list[-1], "not exist"
        if not os.path.exists(trainval_xml_list[-1]):
          print trainval_xml_list[-1], "not exist"
  if name == "VOC2007":
    file_path = os.path.join(txt_fold, "test.txt")
    with open(file_path, 'r') as fp:
      for line in fp:
        line = line.strip()
        test07_jpeg_list.append(os.path.join(jpeg_fold,
"{}.jpg".format(line)))
        test07_xml_list.append(os.path.join(xml_fold,
"{}.xml".format(line)))
        if not os.path.exists(test07_jpeg_list[-1]):
          print test07_jpeg_list[-1], "not exist"
        if not os.path.exists(test07_xml_list[-1]):
          print test07_xml_list[-1], "not exist"

#Guardar información conjunta en un solo archivo

with open("trainval.txt", "w") as wr:
  for i in range(len(trainval_jpeg_list)):
    wr.write("{} {}\n".format(trainval_jpeg_list[i],
trainval_xml_list[i]))

with open("test_2007.txt", "w") as wr:
  for i in range(len(test07_jpeg_list)):
    wr.write("{} {}\n".format(test07_jpeg_list[i], test07_xml_list[i]))
```

A.2. Convertir.sh: Conversión del formato del Dataset

```
#Adquirir información de directorios
CAFFE_ROOT='/home/pame/caffe-yolo'
ROOT_DIR='/home/pame/Desktop/CAFFEYOLO/'
LABEL_FILE=$CAFFE_ROOT/data/yolo/label_map.txt

# Información de carpetas 2007 y 2012 trainval
#LIST_FILE=$CAFFE_ROOT/data/yolo/trainval.txt
#LMDB_DIR='/home/pame/caffe-yolo/data/yolo/lmdb/trainval_lmdb'
#SHUFFLE=true

# Información de carpeta 2007 test
LIST_FILE=$CAFFE_ROOT/data/yolo/test_2007.txt
LMDB_DIR='/home/pame/caffe-yolo/data/yolo/lmdb/test2007_lmdb'
SHUFFLE=false

#Redimensionamiento de la imagen
RESIZE_W=448
RESIZE_H=448

#conversión de datos con caffe
$CAFFE_ROOT/build/tools/convert_box_data \
      --resize_width=$RESIZE_W --resize_height=$RESIZE_H \
      --label_file=$LABEL_FILE $ROOT_DIR $LIST_FILE $LMDB_DIR \
      --encoded=true \
      --encode_type=jpg \
      --shuffle=$SHUFFLE
```

A.3. Media.sh: Cálculo de la media de la imagen

```
# Calcula la media de la imagen del conjunto de datos de train en
formato lmdb

EXAMPLE=/home/pame/caffe-yolo/data/yolo/lmdb
DATA=/home/pame/caffe-yolo/data/yolo/lmdb
TOOLS=/home/pame/caffe-yolo/build/tools

$TOOLS/compute_image_mean $EXAMPLE/trainval_lmdb \
  $DATA/imagenet_mean.binaryproto

echo "Hecho."
```

ANEXO B: CÓDIGO FUENTE DEL ENTRENAMIENTO DE LA RED

B.1. Entrenamiento_yolo.prototxt: Arquitectura de YOLO

```
name: "gnet_yolo"
layer {
  name: "data"
  type: "BoxData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    #crop_size: 227
    mean_file: "/home/pame/caffe-
yolo/data/yolo/lmdb/imagenet_mean.binaryproto"
  }
  data_param {
    source: "/home/pame/caffe-yolo/data/yolo/lmdb/trainval_lmdb"
    batch_size: 150
    backend: LMDB
  }
}
layer {
  name: "data"
  type: "BoxData"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
    transform_param {
    mirror: false
    #crop_size: 227
    mean_file: "/home/pame/caffe-
yolo/data/yolo/lmdb/imagenet_mean.binaryproto"
  }
  data_param {
    source: "/home/pame/caffe-yolo/data/yolo/lmdb/test2007_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 64
    kernel_size: 7
```

```
    pad: 3
    stride: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
  relu_param{
    negative_slope: 0.1
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}

layer{
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  convolution_param {
    num_output: 192
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
```

```
  relu_param{
    negative_slope: 0.1
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}

layer{
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  convolution_param {
    num_output: 128
    kernel_size: 1
    pad: 0
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
  relu_param{
    negative_slope: 0.1
  }
}


layer{
  name: "conv4"
  type: "Convolution"
  bottom: "conv3"
  top: "conv4"
  convolution_param {
    num_output: 256
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
```

```
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu4"
  type: "ReLU"
  bottom: "conv4"
  top: "conv4"
  relu_param{
    negative_slope: 0.1
  }
}

layer{
  name: "conv5"
  type: "Convolution"
  bottom: "conv4"
  top: "conv5"
  convolution_param {
    num_output: 256
    kernel_size: 1
    pad: 0
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu5"
  type: "ReLU"
  bottom: "conv5"
  top: "conv5"
  relu_param{
    negative_slope: 0.1
  }
}

layer{
  name: "conv6"
  type: "Convolution"
  bottom: "conv5"
  top: "conv6"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
```

```
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu6"
  type: "ReLU"
  bottom: "conv6"
  top: "conv6"
  relu_param{
    negative_slope: 0.1
  }
}
layer {
  name: "pool6"
  type: "Pooling"
  bottom: "conv6"
  top: "pool6"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}

layer{
  name: "conv7"
  type: "Convolution"
  bottom: "pool6"
  top: "conv7"
  convolution_param {
    num_output: 256
    kernel_size: 1
    pad: 0
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu7"
  type: "ReLU"
  bottom: "conv7"
  top: "conv7"
  relu_param{
```

```
    negative_slope: 0.1
  }
}

layer{
  name: "conv8"
  type: "Convolution"
  bottom: "conv7"
  top: "conv8"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu8"
  type: "ReLU"
  bottom: "conv8"
  top: "conv8"
  relu_param{
    negative_slope: 0.1
  }
}

layer{
  name: "conv9"
  type: "Convolution"
  bottom: "conv8"
  top: "conv9"
  convolution_param {
    num_output: 256
    kernel_size: 1
    pad: 0
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu9"
  type: "ReLU"
  bottom: "conv9"
```

```
  top: "conv9"
  relu_param{
    negative_slope: 0.1
  }
}

layer{
  name: "conv10"
  type: "Convolution"
  bottom: "conv9"
  top: "conv10"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu10"
  type: "ReLU"
  bottom: "conv10"
  top: "conv10"
  relu_param{
    negative_slope: 0.1
  }
}

layer{
  name: "conv11"
  type: "Convolution"
  bottom: "conv10"
  top: "conv11"
  convolution_param {
    num_output: 256
    kernel_size: 1
    pad: 0
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu11"
```

```
  type: "ReLU"
  bottom: "conv11"
  top: "conv11"
  relu_param{
    negative_slope: 0.1
  }
}


layer{
  name: "conv12"
  type: "Convolution"
  bottom: "conv11"
  top: "conv12"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu12"
  type: "ReLU"
  bottom: "conv12"
  top: "conv12"
  relu_param{
    negative_slope: 0.1
  }
}


layer{
  name: "conv13"
  type: "Convolution"
  bottom: "conv12"
  top: "conv13"
  convolution_param {
    num_output: 256
    kernel_size: 1
    pad: 0
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
```

```
  }
}
layer {
  name: "relu13"
  type: "ReLU"
  bottom: "conv13"
  top: "conv13"
  relu_param{
    negative_slope: 0.1
  }
}

layer{
  name: "conv14"
  type: "Convolution"
  bottom: "conv13"
  top: "conv14"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu14"
  type: "ReLU"
  bottom: "conv14"
  top: "conv14"
  relu_param{
    negative_slope: 0.1
  }
}

layer{
  name: "conv15"
  type: "Convolution"
  bottom: "conv14"
  top: "conv15"
  convolution_param {
    num_output: 512
    kernel_size: 1
    pad: 0
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
```

```
      value: 0
    }
  }
}
layer {
  name: "relu15"
  type: "ReLU"
  bottom: "conv15"
  top: "conv15"
  relu_param{
    negative_slope: 0.1
  }
}


layer{
  name: "conv16"
  type: "Convolution"
  bottom: "conv15"
  top: "conv16"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu16"
  type: "ReLU"
  bottom: "conv16"
  top: "conv16"
  relu_param{
    negative_slope: 0.1
  }
}

layer {
  name: "pool16"
  type: "Pooling"
  bottom: "conv16"
  top: "pool16"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

```
layer{
  name: "conv17"
  type: "Convolution"
  bottom: "pool16"
  top: "conv17"
  convolution_param {
    num_output: 512
    kernel_size: 1
    pad: 0
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu17"
  type: "ReLU"
  bottom: "conv17"
  top: "conv17"
  relu_param{
    negative_slope: 0.1
  }
}


layer{
  name: "conv18"
  type: "Convolution"
  bottom: "conv17"
  top: "conv18"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu18"
  type: "ReLU"
  bottom: "conv18"
  top: "conv18"
  relu_param{
    negative_slope: 0.1
```

```
  }
}



layer{
  name: "conv19"
  type: "Convolution"
  bottom: "conv18"
  top: "conv19"
  convolution_param {
    num_output: 512
    kernel_size: 1
    pad: 0
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu19"
  type: "ReLU"
  bottom: "conv19"
  top: "conv19"
  relu_param{
    negative_slope: 0.1
  }
}



layer{
  name: "conv20"
  type: "Convolution"
  bottom: "conv19"
  top: "conv20"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
```

```
  name: "relu20"
  type: "ReLU"
  bottom: "conv20"
  top: "conv20"
  relu_param{
    negative_slope: 0.1
  }
}


layer{
  name: "conv21"
  type: "Convolution"
  bottom: "conv20"
  top: "conv21"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu21"
  type: "ReLU"
  bottom: "conv21"
  top: "conv21"
  relu_param{
    negative_slope: 0.1
  }
}


layer{
  name: "conv22"
  type: "Convolution"
  bottom: "conv21"
  top: "conv22"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
```

```
      value: 0
    }
  }
}
layer {
  name: "relu22"
  type: "ReLU"
  bottom: "conv22"
  top: "conv22"
  relu_param{
    negative_slope: 0.1
  }
}



layer{
  name: "conv23"
  type: "Convolution"
  bottom: "conv22"
  top: "conv23"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu23"
  type: "ReLU"
  bottom: "conv23"
  top: "conv23"
  relu_param{
    negative_slope: 0.1
  }
}


layer{
  name: "conv24"
  type: "Convolution"
  bottom: "conv23"
  top: "conv24"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
    weight_filler {
```

```
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu24"
  type: "ReLU"
  bottom: "conv24"
  top: "conv24"
  relu_param{
    negative_slope: 0.1
  }
}



layer{
  name: "fc25"
  type: "InnerProduct"
  bottom: "conv24"
  top: "fc25"
  inner_product_param {
    num_output: 4096
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu25"
  type: "ReLU"
  bottom: "fc25"
  top: "fc25"
  relu_param{
    negative_slope: 0.1
  }
}


layer{
  name: "fc26"
  type: "InnerProduct"
  bottom: "fc25"
  top: "result"
  inner_product_param {
    num_output: 1470
    weight_filler {
```

```
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "det_loss"
  type: "DetectionLoss"
  bottom: "result"
  bottom: "label"
  top: "det_loss"
  loss_weight: 1
  detection_loss_param {
    side: 7
    num_class: 20
    num_object: 2
    object_scale: 1.0
    noobject_scale: 0.5
    class_scale: 1.0
    coord_scale: 5.0
    sqrt: true
    constriant: true
  }
}
```

B.2. Train.sh: Entrenamiento con Caffe

```
CAFFE_HOME=/home/pame/caffe-yolo

#Cargando parámetros y pesos de la red
SOLVER=/home/pame/caffe-yolo/examples/yolo/2.-gnet_solver.prototxt
WEIGHTS=/home/pame/caffe-yolo/data/yolo/bvlc_googlenet.caffemodel

#Entrenamiento con Caffe
$CAFFE_HOME/build/tools/caffe train \
    --solver=$SOLVER --weights=$WEIGHTS
```

B.3. solver.prototxt: Parámetros de entrenamiento

```
#Asignación de valores de los parámetros de entrenamiento
net: "/home/pame/caffe-yolo/examples/yolo/gnet_train.prototxt"
test_iter: 500
test_interval: 500
test_initialization: false
display: 50
average_loss: 50
lr_policy: "step"
gamma: 0.1
stepsize: 2500
max_iter: 50000
momentum: 0.9
weight_decay: 0.0005
snapshot: 5000
snapshot_prefix: "/home/pame/Pictures/models/model"
solver_mode: CPU
```

ANEXO C: CÓDIGO FUENTE DE LA VALIDACIÓN

C.1. Validacion.py: Visualización de resultados

```
#Cargar módulos
import caffe
caffe.set_mode_cpu()
from datetime import datetime
import numpy as np
import sys, getopt
import cv2

#Definir número de clases y salidas obtenidas
def interpret_output(output, img_width, img_height):
      classes = ["aeroplane", "bicycle", "bird", "boat", "bottle",
"bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse",
"motorbike", "person", "pottedplant", "sheep", "sofa",
"train","tvmonitor"]
      w_img = img_width
      h_img = img_height
      print w_img, h_img
      threshold = 0.2
      iou_threshold = 0.5
      num_class = 20
      num_box = 2
      grid_size = 7
      probs = np.zeros((7,7,2,20))
      class_probs = np.reshape(output[0:980],(7,7,20))
#     print class_probs
      scales = np.reshape(output[980:1078],(7,7,2))
#     print scales
      boxes = np.reshape(output[1078:],(7,7,2,4))
      offset =
np.transpose(np.reshape(np.array([np.arange(7)]*14),(2,7,7)),(1,2,0))

      boxes[:,:,:,0] += offset
      boxes[:,:,:,1] += np.transpose(offset,(1,0,2))
      boxes[:,:,:,0:2] = boxes[:,:,:,0:2] / 7.0
      boxes[:,:,:,2] = np.multiply(boxes[:,:,:,2],boxes[:,:,:,2])
      boxes[:,:,:,3] = np.multiply(boxes[:,:,:,3],boxes[:,:,:,3])

      boxes[:,:,:,0] *= w_img
      boxes[:,:,:,1] *= h_img
      boxes[:,:,:,2] *= w_img
      boxes[:,:,:,3] *= h_img

#Filtrar y asignar matrices
      for i in range(2):
            for j in range(20):
                  probs[:,:,i,j] =
np.multiply(class_probs[:,:,j],scales[:,:,i])
      filter_mat_probs = np.array(probs>=threshold,dtype='bool')
      filter_mat_boxes = np.nonzero(filter_mat_probs)
      boxes_filtered =
boxes[filter_mat_boxes[0],filter_mat_boxes[1],filter_mat_boxes[2]]
      probs_filtered = probs[filter_mat_probs]
```

```
      classes_num_filtered =
np.argmax(probs,axis=3)[filter_mat_boxes[0],filter_mat_boxes[1],filter_
mat_boxes[2]]

      argsort = np.array(np.argsort(probs_filtered))[::-1]
      boxes_filtered = boxes_filtered[argsort]
      probs_filtered = probs_filtered[argsort]
      classes_num_filtered = classes_num_filtered[argsort]

#Condicional de valores de IoU
      for i in range(len(boxes_filtered)):
            if probs_filtered[i] == 0 : continue
            for j in range(i+1,len(boxes_filtered)):
                  if iou(boxes_filtered[i],boxes_filtered[j]) >
iou_threshold :
                        probs_filtered[j] = 0.0

      filter_iou = np.array(probs_filtered>0.0,dtype='bool')
      boxes_filtered = boxes_filtered[filter_iou]
      probs_filtered = probs_filtered[filter_iou]
      classes_num_filtered = classes_num_filtered[filter_iou]

      result = []
      for i in range(len(boxes_filtered)):

      result.append([classes[classes_num_filtered[i]],boxes_filtered[i][
0],boxes_filtered[i][1],boxes_filtered[i][2],boxes_filtered[i][3],probs
_filtered[i]])

      return result

#Cálculo de IoU de la red
def iou(box1,box2):
      tb = min(box1[0]+0.5*box1[2],box2[0]+0.5*box2[2])-max(box1[0]-
0.5*box1[2],box2[0]-0.5*box2[2])
      lr = min(box1[1]+0.5*box1[3],box2[1]+0.5*box2[3])-max(box1[1]-
0.5*box1[3],box2[1]-0.5*box2[3])
      if tb < 0 or lr < 0 : intersection = 0
      else : intersection =  tb*lr
      return intersection / (box1[2]*box1[3] + box2[2]*box2[3] -
intersection)

#Representación gráfica de resultados
def show_results(img,results, img_width, img_height):
      img_cp = img.copy()
      disp_console = True
      imshow = True
#     if self.filewrite_txt :
#           ftxt = open(self.tofile_txt,'w')
      for i in range(len(results)):
            x = int(results[i][1])
            y = int(results[i][2])
            w = int(results[i][3])//2
            h = int(results[i][4])//2
            if disp_console : print '    class : ' + results[i][0] + ' ,
[x,y,w,h]=[' + str(x) + ',' + str(y) + ',' + str(int(results[i][3])) +
',' + str(int(results[i][4]))+'], Confidence = ' + str(results[i][5])
            xmin = x-w
```

```
            xmax = x+w
            ymin = y-h
            ymax = y+h
            if xmin<0:
                  xmin = 0
            if ymin<0:
                  ymin = 0
            if xmax>img_width:
                  xmax = img_width
            if ymax>img_height:
                  ymax = img_height
            if  imshow:

      cv2.rectangle(img_cp,(xmin,ymin),(xmax,ymax),(0,255,0),2)
                  #print xmin, ymin, xmax, ymax
                  cv2.rectangle(img_cp,(xmin,ymin-
20),(xmax,ymin),(125,125,125),-1)
                  cv2.putText(img_cp,results[i][0] + ' : %.2f' %
results[i][5],(xmin+5,ymin-7),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,0,0),1)

      if imshow :
            cv2.imshow('YOLO detection',img_cp)
            cv2.waitKey(1000)


#Visualización de la detección
def main(argv):
      model_filename = ''
      weight_filename = ''
      img_filename = ''
      try:
            opts, args = getopt.getopt(argv, "hm:w:i:")
            print opts
      except getopt.GetoptError:
            print 'yolo_main.py -m <model_file> -w <output_file> -i
<img_file>'
            sys.exit(2)
      for opt, arg in opts:
            if opt == '-h':
                  print 'yolo_main.py -m <model_file> -w <weight_file> -i
<img_file>'
                  sys.exit()
            elif opt == "-m":
                  model_filename = arg
            elif opt == "-w":
                  weight_filename = arg
            elif opt == "-i":
                  img_filename = arg
      print 'model file is "', model_filename
      print 'weight file is "', weight_filename
      print 'image file is "', img_filename
      net = caffe.Net(model_filename, weight_filename, caffe.TEST)
      img = caffe.io.load_image(img_filename) # load the image using
caffe io
      inputs = img
      transformer = caffe.io.Transformer({'data':
net.blobs['data'].data.shape})
      transformer.set_transpose('data', (2,0,1))
```

```
      start = datetime.now()
      out =
net.forward_all(data=np.asarray([transformer.preprocess('data',
inputs)]))
      end = datetime.now()
      elapsedTime = end-start
      print 'total time is " milliseconds',
elapsedTime.total_seconds()*1000
      print out.iteritems()
      img_cv = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
      results = interpret_output(out['result'][0], img.shape[1],
img.shape[0])
      show_results(img_cv,results, img.shape[1], img.shape[0])
      cv2.waitKey(10000)

if __name__=='__main__':
      main(sys.argv[1:])
```

C.2. Deploy.prototxt: Arquitectura de YOLO para validación

```
name: "gnet_yolo"
input: "data"
input_shape {
  dim: 1
  dim: 3
  dim: 448
  dim: 448
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 64
    kernel_size: 7
    pad: 3
    stride: 2
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
  relu_param{
    negative_slope: 0.1
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer{
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  convolution_param {
    num_output: 192
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu2"
  type: "ReLU"
```

```
  bottom: "conv2"
  top: "conv2"
  relu_param{
    negative_slope: 0.1
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer{
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  convolution_param {
    num_output: 128
    kernel_size: 1
    pad: 0
    stride: 1
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv4"
  type: "Convolution"
  bottom: "conv3"
  top: "conv4"
  convolution_param {
    num_output: 256
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu4"
  type: "ReLU"
  bottom: "conv4"
  top: "conv4"
  relu_param{
    negative_slope: 0.1
  }
```

```
}
layer{
  name: "conv5"
  type: "Convolution"
  bottom: "conv4"
  top: "conv5"
  convolution_param {
    num_output: 256
    kernel_size: 1
    pad: 0
    stride: 1
  }
}
layer {
  name: "relu5"
  type: "ReLU"
  bottom: "conv5"
  top: "conv5"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv6"
  type: "Convolution"
  bottom: "conv5"
  top: "conv6"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu6"
  type: "ReLU"
  bottom: "conv6"
  top: "conv6"
  relu_param{
    negative_slope: 0.1
  }
}
layer {
  name: "pool6"
  type: "Pooling"
  bottom: "conv6"
  top: "pool6"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer{
  name: "conv7"
  type: "Convolution"
  bottom: "pool6"
```

```
  top: "conv7"
  convolution_param {
    num_output: 256
    kernel_size: 1
    pad: 0
    stride: 1
  }
}
layer {
  name: "relu7"
  type: "ReLU"
  bottom: "conv7"
  top: "conv7"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv8"
  type: "Convolution"
  bottom: "conv7"
  top: "conv8"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu8"
  type: "ReLU"
  bottom: "conv8"
  top: "conv8"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv9"
  type: "Convolution"
  bottom: "conv8"
  top: "conv9"
  convolution_param {
    num_output: 256
    kernel_size: 1
    pad: 0
    stride: 1
  }
}
layer {
  name: "relu9"
  type: "ReLU"
  bottom: "conv9"
  top: "conv9"
  relu_param{
    negative_slope: 0.1
  }
```

```
}
layer{
  name: "conv10"
  type: "Convolution"
  bottom: "conv9"
  top: "conv10"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu10"
  type: "ReLU"
  bottom: "conv10"
  top: "conv10"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv11"
  type: "Convolution"
  bottom: "conv10"
  top: "conv11"
  convolution_param {
    num_output: 256
    kernel_size: 1
    pad: 0
    stride: 1
  }
}
layer {
  name: "relu11"
  type: "ReLU"
  bottom: "conv11"
  top: "conv11"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv12"
  type: "Convolution"
  bottom: "conv11"
  top: "conv12"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu12"
  type: "ReLU"
```

```
  bottom: "conv12"
  top: "conv12"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv13"
  type: "Convolution"
  bottom: "conv12"
  top: "conv13"
  convolution_param {
    num_output: 256
    kernel_size: 1
    pad: 0
    stride: 1
  }
}
layer {
  name: "relu13"
  type: "ReLU"
  bottom: "conv13"
  top: "conv13"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv14"
  type: "Convolution"
  bottom: "conv13"
  top: "conv14"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu14"
  type: "ReLU"
  bottom: "conv14"
  top: "conv14"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv15"
  type: "Convolution"
  bottom: "conv14"
  top: "conv15"
  convolution_param {
    num_output: 512
    kernel_size: 1
    pad: 0
    stride: 1
```

```
  }
}
layer {
  name: "relu15"
  type: "ReLU"
  bottom: "conv15"
  top: "conv15"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv16"
  type: "Convolution"
  bottom: "conv15"
  top: "conv16"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu16"
  type: "ReLU"
  bottom: "conv16"
  top: "conv16"
  relu_param{
    negative_slope: 0.1
  }
}
layer {
  name: "pool16"
  type: "Pooling"
  bottom: "conv16"
  top: "pool16"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer{
  name: "conv17"
  type: "Convolution"
  bottom: "pool16"
  top: "conv17"
  convolution_param {
    num_output: 512
    kernel_size: 1
    pad: 0
    stride: 1
  }
}
layer {
  name: "relu17"
  type: "ReLU"
```

```
  bottom: "conv17"
  top: "conv17"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv18"
  type: "Convolution"
  bottom: "conv17"
  top: "conv18"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu18"
  type: "ReLU"
  bottom: "conv18"
  top: "conv18"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv19"
  type: "Convolution"
  bottom: "conv18"
  top: "conv19"
  convolution_param {
    num_output: 512
    kernel_size: 1
    pad: 0
    stride: 1
  }
}
layer {
  name: "relu19"
  type: "ReLU"
  bottom: "conv19"
  top: "conv19"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv20"
  type: "Convolution"
  bottom: "conv19"
  top: "conv20"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
```

```
  }
}
layer {
  name: "relu20"
  type: "ReLU"
  bottom: "conv20"
  top: "conv20"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv21"
  type: "Convolution"
  bottom: "conv20"
  top: "conv21"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu21"
  type: "ReLU"
  bottom: "conv21"
  top: "conv21"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv22"
  type: "Convolution"
  bottom: "conv21"
  top: "conv22"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 2
  }
}
layer {
  name: "relu22"
  type: "ReLU"
  bottom: "conv22"
  top: "conv22"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "conv23"
  type: "Convolution"
  bottom: "conv22"
  top: "conv23"
```

```
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu23"
  type: "ReLU"
  bottom: "conv23"
  top: "conv23"
  relu_param{
    negative_slope: 0.1
  }
}


layer{
  name: "conv24"
  type: "Convolution"
  bottom: "conv23"
  top: "conv24"
  convolution_param {
    num_output: 1024
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
layer {
  name: "relu24"
  type: "ReLU"
  bottom: "conv24"
  top: "conv24"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
  name: "fc25"
  type: "InnerProduct"
  bottom: "conv24"
  top: "fc25"
  inner_product_param {
    num_output: 4096
  }
}
layer {
  name: "relu25"
  type: "ReLU"
  bottom: "fc25"
  top: "fc25"
  relu_param{
    negative_slope: 0.1
  }
}
layer{
```

```
  name: "fc26"
  type: "InnerProduct"
  bottom: "fc25"
  top: "result"
  inner_product_param {
    num_output: 1470
  }
}
```