

**UNIVERSIDAD RAFAEL LANDÍVAR**  
**FACULTAD DE INGENIERÍA**  
**LICENCIATURA EN INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES**

**"SISTEMA DE VISIÓN ARTIFICIAL PARA EL RECONOCIMIENTO DE MATRÍCULAS DE  
AUTOMÓVILES EMPLEANDO LA TARJETA JETSON NANO"**

**TESIS DE GRADO**

**DIEGO FERNANDO BRAN ARRIOLA**

**CARNET 10872-17**

**GUATEMALA DE LA ASUNCIÓN, SEPTIEMBRE DE 2022**  
**CAMPUS CENTRAL "SAN FRANCISCO DE BORJA, S. J." DE LA CIUDAD DE GUATEMALA**

**UNIVERSIDAD RAFAEL LANDÍVAR**  
FACULTAD DE INGENIERÍA  
LICENCIATURA EN INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

**"SISTEMA DE VISIÓN ARTIFICIAL PARA EL RECONOCIMIENTO DE MATRÍCULAS DE  
AUTOMÓVILES EMPLEANDO LA TARJETA JETSON NANO"**

TESIS DE GRADO

TRABAJO PRESENTADO AL CONSEJO DE LA FACULTAD DE  
INGENIERÍA

POR  
**DIEGO FERNANDO BRAN ARRIOLA**

PREVIO A CONFERÍRSELE  
EL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES EN EL GRADO ACADÉMICO DE  
LICENCIADO

GUATEMALA DE LA ASUNCIÓN, SEPTIEMBRE DE 2022  
CAMPUS CENTRAL "SAN FRANCISCO DE BORJA, S. J." DE LA CIUDAD DE GUATEMALA

## **AUTORIDADES DE LA UNIVERSIDAD RAFAEL LANDÍVAR**

RECTOR: P. MIQUEL CORTÉS BOFILL, S. J.

VICERRECTORA ACADÉMICA: DRA. MARTHA ROMELIA PÉREZ CONTRERAS DE CHEN

VICERRECTOR DE INVESTIGACIÓN Y PROYECCIÓN: ING. JOSÉ JUVENTINO GÁLVEZ RUANO

VICERRECTOR DE INTEGRACIÓN UNIVERSITARIA: P. JOSE ANTONIO RUBIO AGUILAR, S. J.

VICERRECTORA ADMINISTRATIVA: MGTR. SILVANA GUISELA ZIMERI VELÁSQUEZ DE CELADA

SECRETARIO GENERAL: DR. LARRY AMILCAR ANDRADE - ABULARACH

## **AUTORIDADES DE LA FACULTAD DE INGENIERÍA**

DECANO: MGTR. JORGE ANTONIO GUILLEN GALVÁN

VICEDECANA: MGTR. ANABELLA CLARA BEATRIZ LOPEZ LOBOS

SECRETARIA: MGTR. CLAUDIA LORENA MÁRQUEZ FERNÁNDEZ

DIRECTOR DE CARRERA: ING. MANUEL ALEJANDRO RÍOS RIVAS

## **NOMBRE DEL ASESOR DE TRABAJO DE GRADUACIÓN**

MGTR. MANUEL ALEJANDRO RÍOS RIVAS

## **TERNA QUE PRACTICÓ LA EVALUACIÓN**

MGTR. WILLIAM LEONARDO MARTÍNEZ CONTRERAS

LIC. GERMAN ALBERTO CHEW LORENZANA

LIC. JUAN CARLOS ROMERO SALAZAR

Guatemala, 12 de julio de 2022

Mgtr. Claudia Lorena Márquez Fernández  
Secretaria  
Facultad de Ingeniería  
Universidad Rafael Landívar  
Presente

Estimada Magister:

Por este medio me es grato saludarle y deseándole éxito en sus labores diarias. El motivo es informarle que he revisado el informe final del Trabajo de Graduación titulado: **“SISTEMA DE VISIÓN ARTIFICIAL PARA EL RECONOCIMIENTO DE MATRÍCULAS DE AUTOMOVILES EMPLEANDO LA TARJETA JETSON NANO”**, del estudiante **DIEGO FERNANDO BRAN ARRIOLA**, de la carrera Ingeniería en Electrónica y Telecomunicaciones, quien se identifica con número de carné 10872-17.

De tal cuenta doy como aprobado dicho informe, de acuerdo a los requerimientos de la Facultad de Ingeniería de la Universidad Rafael Landívar.

Agradeciendo su atención y deseándole éxitos en sus actividades profesionales me suscribo.

Atentamente,



Mgtr. Manuel Alejandro Ríos Rivas  
Asesor



Universidad  
Rafael Landívar  
Tradición Jesuita en Guatemala

FACULTAD DE INGENIERÍA  
No. 02256-2022

### Orden de Impresión

De acuerdo a la aprobación de la Evaluación del Trabajo de Graduación en la variante Tesis de Grado del estudiante DIEGO FERNANDO BRAN ARRIOLA, Carnet 10872-17 en la carrera LICENCIATURA EN INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES, del Campus Central, que consta en el Acta No. 02235-2022 de fecha 2 de septiembre de 2022, se autoriza la impresión digital del trabajo titulado:

**"SISTEMA DE VISIÓN ARTIFICIAL PARA EL RECONOCIMIENTO DE MATRÍCULAS DE AUTOMÓVILES EMPLEANDO LA TARJETA JETSON NANO"**

Previo a conferírsele el título de INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES en el grado académico de LICENCIADO.

Dado en la ciudad de Guatemala de la Asunción, a los 28 días del mes de septiembre del año 2022.

MGTR. CLAUDIA LORENA MÁRQUEZ FERNÁNDEZ, SECRETARÍA  
INGENIERÍA

Universidad Rafael Landívar



## **DEDICATORIA**

<b>A Dios</b>	Por darme la vida, bendecirme constantemente y permitirme realizar este proyecto
<b>A mi Tía</b>	Paola Arriola por el valioso apoyo incondicional, las herramientas, consejos brindados, así como el constante animo transmitido.
<b>A mis Abuelos</b>	Manuel Rolando Arriola y María Irma Duran por el apoyo, acompañamiento y cariño desde pequeño y hasta la fecha
<b>A mis Padres</b>	Rony Bran y Consuelo Arriola por el apoyo brindado en el cumplimiento de mis metas y en mi proceso de estudio.
<b>A mis Familiares y Amigos</b>	A todos mis amigos, primos, hermanos y tíos por el apoyo, los ánimos constantes, consejos y experiencias compartidas en este proceso de crecimiento.
<b>A mi Asesor</b>	Manuel Ríos por el constante seguimiento, las críticas constructivas, las lecciones y experiencia compartida en el desarrollo del proyecto.
<b>A mis Docentes</b>	Por compartir sus conocimientos y experiencias en mi formación profesional. Especialmente a el Lic. Jorge Humberto Mahuad por su dedicación en los cursos, el apoyo extra en múltiples materias y situaciones presentadas en mis años de estudio, a los ingenieros Salvador Tuna, Ester Pineda, Héctor Eliú y German Chew. Por la dedicación al impartir sus cursos y laboratorios.
<b>A mi Compañera de Equipo</b>	Sara Elizabeth Castro Arriaga por su participación, esfuerzo y dedicación en la elaboración de este proyecto.

## ÍNDICE GENERAL

<i>RESUMEN EJECUTIVO</i> .....	1
<i>I. INTRODUCCIÓN</i> .....	2
<i>1. Marco Teórico</i> .....	3
1.1 Machine Learning .....	3
1.2 Deep Learning .....	7
1.3 Perceptrón de Rosenblatt.....	8
1.4 Funciones de Activación.....	9
1.5 Arquitecturas de Red .....	12
1.6 Descenso de Gradiente ( <i>Gradient Descent</i> ).....	14
1.7 Descenso del Gradiente en mini lotes (Mini-Batch Gradient Descent)....	17
1.8 Retropropagación .....	18
1.9 Sets de Datos .....	25
1.10 Set de entrenamiento, Set de prueba y Generalización.....	26
1.11 Ajuste de hiperparámetros y fuga de información en el conjunto de pruebas .....	28
1.12 Conjunto de validación para evitar el sobreajuste .....	30
1.13 Inferencia y entrenamiento .....	31
1.14 Redes neuronales Convolucionales CNN.....	31
Relleno (Padding) .....	39
Zancadas (Strides).....	40
Uso del sesgo.....	41
Capa ReLU.....	42
Single Shot Detector .....	44
1.15 Aprendizaje por Transferencia .....	49

1.16	Computación en la Nube (Cloud Computing) .....	51
	Nube .....	51
1.17	Modelos de servicio en la nube .....	54
	Infraestructura como servicio (IaaS – Infrastructure as a Service): .....	55
	Plataforma como servicio (PaaS-Platform as a Service):.....	55
	Software como servicio (SaaS- Software as a Service): .....	55
	Base de Datos como Servicio (DBaaS- Database as a Service): .....	56
	Función como Servicio (FaaS-Función como Servicio):.....	56
	Almacenamiento como Servicio (STaaS-Storage as a Service): .....	57
1.18	Internet de las Cosas (IoT-Internet Of Things) .....	57
	IoT, Machine-to-machine y SCADA.....	58
	Modelo de referencia de IoT .....	59
	1. Dispositivos físicos y controladores:.....	61
	2. Conectividad:.....	61
	3. Edge Computing: .....	61
	4. Acumulación de Datos: .....	61
	5. Abstracción de Datos:.....	62
	6. Capa de aplicación: .....	62
	7. Colaboración y Procesos:.....	62
1.19	Contenedores .....	62
	Beneficios de los contenedores.....	63
1.20	ONNX (Open Neuronal Network Exchange) .....	64
II.	PLANTEAMIENTO DEL PROBLEMA .....	65
2.1	Objetivo General.....	66
2.2	Objetivos Específicos .....	66
2.3	Alcances.....	66
2.4	Límites.....	67
2.5	Aportes.....	68



<i>III.</i>	<i>MÉTODO</i> .....	69
3.1	Descripción del Proyecto.....	69
3.2	Instrumentos .....	69
3.3	Procedimiento .....	78
	Configuración inicial del módulo Jetson .....	79
	Re-entrenamiento de la red neuronal convolucional Mobilenet V1 .....	80
	Desarrollo del Código de borde .....	82
	Archivo car2.py:.....	82
	Archivo my-detection3.py:.....	84
	Configuración de recursos y políticas en AWS .....	87
3.4	Análisis y Diseño.....	90
	Computación de Borde .....	91
	Computación de Nube .....	92
<i>IV.</i>	<i>PRESENTACIÓN Y ANÁLISIS DE RESULTADOS</i> .....	94
	Rendimiento del modelo re-entrenado.....	94
	Rendimiento de la detección en Rekognition .....	97
	Consumo de Recursos .....	99
<i>V.</i>	<i>CONCLUSIONES</i> .....	101
<i>VI.</i>	<i>RECOMENDACIONES</i> .....	103
<i>VII.</i>	<i>REFERENCIAS</i> .....	104
<i>VIII.</i>	<i>ANEXOS</i> .....	106
	Anexo 1: Enlace al repositorio de GitHub donde se encuentra el paso a paso del desarrollo del proyecto. ....	106
	<a href="https://github.com/DiegoBran16/JetsonNano-Cloud-Edge-Car-speed-plate-monitoring">https://github.com/DiegoBran16/JetsonNano-Cloud-Edge-Car-speed-plate-monitoring</a> .....	106
	Anexo 2: Enlace al repositorio de GitHub dusty-nv/jetson-inference:.....	106
	Anexo 3: Enlace de descarga para la red de base Mobilenet V1: .....	106

Anexo 4: Enlace para descargar sets de datos de Google en formato Open Images:.....	106
Anexo 5: Imagens de la detección de automoviles por el modelo re-entrenado .....	106
Anexo 6: Imágenes en S3 .....	107
Anexo 7: Arquitectura de red neuronal re-entrenada .....	107
Anexo 8: Sensor Sony IMX477 .....	114
Anexo 9: Lente Arducam 2.8-12mm varifocal montura C .....	114
Anexo 10: Costos del sistema Actual.....	115
Anexo 11: Costos del sistema Mejorado .....	116

## ÍNDICE DE ILUSTRACIONES

Ilustración 1: Relación entre Inteligencia artificial, Machine Learning, aprendizaje profundo y redes neuronales profundas .....	7
Ilustración 2 Comparación de la neurona biológica y la arquitectura del perceptrón .....	9
Ilustración 3 Funciones de activación .....	10
Ilustración 4 Comparación grafica de las arquitecturas de red.....	14
Ilustración 5 Método descendiente de gradiente.....	15
Ilustración 6 Red multinivel con una neurona por capa .....	18
Ilustración 7 Evolución de las curvas del error de entrenamiento y error de prueba durante el proceso de aprendizaje.....	27
Ilustración 8 Búsqueda de cuadrícula para 2 hiperparámetros. ....	29
Ilustración 9 Proceso de entrenamiento utilizando el conjunto de datos de entrenamiento, validación y prueba.....	31
Ilustración 10 Arquitectura de la red LeNet-5.....	32
Ilustración 11 Operación Convolución, entrada de 32x32x3 y filtro de 5x5x3 .....	34
Ilustración 12 Operación Convolución, entrada de 7x7x1 y filtro de 3x3x1.....	38
Ilustración 13 Ejemplo de relleno en un mapa de características .....	39
Ilustración 14 Ejemplificación del agrupamiento .....	43
Ilustración 15 Arquitectura SSD .....	45
Ilustración 16 Las cajas delimitadoras predominadas se agregan a cada celda de la cuadrícula en cada mapa de características .....	45
Ilustración 17 Mapa de Características con 4 cajas delimitadoras predeterminadas .....	46
Ilustración 18 Relación de Aspecto.....	47
Ilustración 19 SSD, entrenamiento e inferencia.....	48
Ilustración 20 Representación del campo receptivo de los mapas de características .....	49

Ilustración 21 Aprendizaje por transferencia, red convolucional inicial y nueva red re-entrenada para nuevas tareas de clasificación.....	50
Ilustración 22 BUSINESS DRIVERS OF DIGITAL TRANSFORMATION .....	56
Ilustración 23 Análisis de las búsquedas en Google para el termino IoT, patentes y publicaciones técnicas de IoT .....	58
Ilustración 24Capas del ecosistema IoT.....	60
Ilustración 25 IoT World Forum Reference Model.....	60
Ilustración 26 Disposición de entradas y salidas Jetson Nano .....	71
Ilustración 27Fases del desarrollo del sistema .....	79
Ilustración 28 Diagrama de la clase Auto .....	82
Ilustración 29 Diagrama de la clase Tracker .....	83
Ilustración 30 Diagrama de la función tracking .....	83
Ilustración 31 Diagrama de la función tracking .....	85
Ilustración 32 Diagrama de flujo de la función MQTT-S3getCar-Recognition-DynamoDB, .....	90
Ilustración 33 Arquitectura del Sistema .....	90
Ilustración 34 Implementación del Sistema de borde .....	92
Ilustración 35 Participación en el Desarrollo .....	93
Ilustración 36 El cuadrado verde representa la caja delimitadora de verdad básica mientras que el rectángulo rojo representa la caja delimitadora predictora.....	95
Ilustración 37 Comparación de funciones de perdida .....	97
Ilustración 38 Precisión en la detección de Caracteres de AWS Rekognition .....	98
Ilustración 39Precisión en la detección de Matrículas de AWS Rekognition .....	98
Ilustración 40 Tabla de autos detectados en la base de datos.....	99
Ilustración 41 Consumo de recursos del código en la Jetson Nano.....	100

## ÍNDICE DE TABLAS

Tabla 1 Especificaciones Jetson Nano .....	70
Tabla 2 Algunos componentes claves del JetPack 4.6.1 .....	73
Tabla 3 Resultados del rendimiento del modelo con 30 épocas de entrenamiento	96
Tabla 4 Resultados del rendimiento del modelo con 60 épocas de entrenamiento .....	96
Tabla 5 Características del sensor Sony IMX477 .....	114
Tabla 6 Características del lente zoom 2.8-12mm de ArduCam .....	115
Tabla 7 CAPEX del sistema actual .....	115
Tabla 8 OPEX del primer mes del sistema actual .....	116
Tabla 9 Costos del primer año con el sistema actual .....	116
Tabla 10 CAPEX sistema mejorado .....	117
Tabla 11 OPEX Primer mes .....	117
Tabla 12: Proyección de Costos para el primer año con el sistema mejorado ....	117

## RESUMEN EJECUTIVO

El objetivo del proyecto fue el desarrollo de un sistema de monitoreo de automóviles utilizando la plataforma Jetson, utilizando el modelo Jetson Nano Developer Kit como el dispositivo IoT en el borde para realizar las inferencias de los automóviles y la Nube publica Amazon Web Services para la detección y almacenamiento de la matrícula del automóvil detectado.

Para el desarrollo se utilizó como herramientas: El repositorio Jetson-inference, el cual provee un contenedor de Docker configurado con las dependencias necesarias para la manipulación de la tarjeta Jetson Nano y para ejecutar los scripts de entrenamiento y evaluación de la red neuronal. Los servicios de AWS para la detección de texto en las matrículas, almacenamiento de las imágenes de los automóviles, registro del módulo de borde en el servicio de IoT almacenamiento de los datos del automóvil en una tabla de una base de datos administrada y una función *serverless* para la ejecución de procesos en la nube.

Por último, se evaluó el rendimiento de la red neuronal para las inferencias de los automóviles, el rendimiento de la detección de texto en la nube y se observó el flujo de datos en el sistema y posibles áreas de mejora.

**Descriptores:** Jetson Nano, Amazon Web Services, IoT, Dispositivo de borde, Deep Learning, Detección de Objetos, Computación en la Nube.

## I. INTRODUCCIÓN

Los módulos Jetson de NVIDIA son pequeños dispositivos de bajo consumo y alto rendimiento gracias a sus unidades de procesamiento de gráficos (GPU). Estos módulos son optimizados para proveer un alto desempeño en la pila de software NVIDIA Cuda-X. De manera que permiten correr múltiples redes neuronales en paralelo para realizar tareas de IA en el borde. Además, gracias a sus distintas entradas y salidas, las tarjetas Jetson permiten integraciones con diversos sensores, cámaras y módulos de transmisión para capturar, analizar y procesar datos desde el origen.

Las nubes públicas como Amazon Web Services, Google Cloud, Azure, entre otras, poseen servicios para integrar los dispositivos IoT. Esto permite que los módulos Jetson interactúen con los múltiples servicios de la nube, entre ellos servicios de almacenamiento de objetos, reconocimiento de texto y bases de datos.

Integrar los dispositivos de borde con las arquitecturas de nube permite desplegar sistemas completos con presupuestos más reducidos que al utilizar infraestructuras tradicionales. Al poseer un modelo de pago por uso se eliminan las inversiones de bienes de capital, ya que no es necesario realizar compras de servidores, equipos de red, equipos de almacenamiento, construcción de cuartos de servidores, aire acondicionado, entre otros.

A lo largo de este documento se detalla el proceso de configuración de un módulo Jetson, el desarrollo del código de borde para la inferencia de automóviles y la captura de imágenes, configuración de recursos en la nube y la integración entre el módulo de borde y los recursos en la nube.

## **1. Marco Teórico**

## **2. Machine Learning**

Jesús Bobadilla (2020) describe el termino Machine Learning como la ciencia de programar computadoras de forma que estas puedan aprender de los datos. En vez de utilizar un enfoque tradicional en el cual se programa paso a paso las soluciones de cada una de las necesidades planteadas, el área de Machine Learning se centra en el desarrollo de algoritmos capaces de abstraer patrones de diferentes tipos de datos.

Una de las características de estos algoritmos es que suelen ser hasta cierto punto genéricos, por ejemplo, una solución que clasifique imágenes de distintos tipos de automóviles no diferirá sustancialmente de una solución que clasifique tipos de prendas, ambas se basan en algoritmos de Machine Learning que clasifican datos etiquetados.

En este momento se puede pensar que automatizar el desarrollo de modelos es una tarea fácil y automatizable pero la realidad es que no es una tarea tan simple. Para el desarrollo del modelo el ingeniero de datos debe realizar múltiples tareas, como por ejemplo identificar la fuente de datos, limpieza de datos, la eliminación de información fuertemente correlacionada, búsqueda de información sesgada, normalizaciones de datos, identificación de posibles soluciones para el desarrollo del algoritmo, ajustes finos de los hiperparámetros del método seleccionado, análisis de resultados e identificación de comportamientos imprecisos entre otros.

Con objetivo de seleccionar el mejor modelo para los datos presentados el ingeniero de datos deberá clasificar el algoritmo de alguno de los siguientes tipos:



## A) Aprendizaje supervisado

El aprendizaje supervisado se aplica cuando cada muestra se asocia a una etiqueta. Bobadilla (2020) expone el siguiente ejemplo:

- Un conjunto de imágenes en la que cada una posee algún tipo de etiqueta (pict0001.bmp, "perro"), (pict0002.bmp, "pájaro"), (pict0003.bmp, "gallo").

A partir de este conjunto de datos se pueden aplicar diversos algoritmos de clasificación para entrenar un modelo y predecir la etiqueta que le correspondería a una nueva imagen, la cual no existe dentro del conjunto de datos del entrenamiento. Éste sería un problema de clasificación. Sin embargo, dentro del aprendizaje supervisado también es posible tener un conjunto de valores numéricos como los precios de casas en un sector determinado. Con estos datos se puede determinar el precio de otras casas similares en el sector utilizando un modelo de regresión.

En pocas palabras se puede resumir que los algoritmos de clasificación son aquellos que predicen la clase o categoría de los datos, mientras que la regresión es el proceso de predecir distintos valores continuos.

Los algoritmos de Machine Learning crean un modelo a partir de datos, puede ser un modelo simple como la solución lineal que mejor se ajuste a las muestras de origen, valores objetivos o puede ser más complejo como la búsqueda de factores ocultos que representen la información más importante contenida.

La importancia del aprendizaje supervisado en Machine Learning está aumentando muy rápidamente, esto se debe a:

1. La cantidad masiva de datos brindadas por los dispositivos de IoT (internet of things)
2. La enorme cantidad de aplicaciones, mercado digital, e interacciones causadas por las redes sociales.
3. Los nuevos algoritmos destinados a resolver diversos tipos de aprendizaje supervisado hacen posible obtener resultados comerciales significativos, como reconocimiento facial, sistemas de recomendación entre otros.
4. Las crecientes capacidades de procesamiento, supercomputadoras paralelas y unidades de procesamiento gráfico (GPU)
5. La "democratización" del Machine Learning ha puesto a disposición de todos trabajar con recursos tecnológicos como Tensorflow o granjas de GPUs, APIs, entornos de programación IDEs y tecnologías proporcionadas por Scikit, Keras y Jupyter

## **B) Aprendizaje no supervisado**

El aprendizaje no supervisado, a diferencia del supervisado, utiliza información sin etiquetas. Una de las aplicaciones más conocidas son los modelos de clústering o agrupamiento, esta técnica tiene como objetivo agrupar muestras y suele ser de utilidad para obtener diferentes tipos de clientes para un producto o servicio, agrupar productos en algún comercio, identificar comportamientos, entre otros.

Partiendo de la referencia de una base de datos en la que se encuentran los datos de múltiples clientes que consumen los servicios de telefonía de una operadora móvil, se puede aplicar un modelo de clústering para segmentar los distintos consumidores del servicio. Con esta información la empresa telefónica podría realizar campañas publicitarias específicas según el clúster que desea abordar.

Otra de las aplicaciones del aprendizaje no supervisado es la reducción de dimensionalidad, esta suele ser utilizada como una etapa de pre-procesamiento en algún otro tipo de labores de Machine Learning, por ejemplo, clasificación o regresión. En muchos casos los datos se presentan dispersos o no proporcionan mucha información. Bobadilla (2020) utiliza de ejemplo los sistemas de recomendación. Los usuarios compran, hacen clic, consumen o votan en una porción pequeña de los productos disponibles. Cuando se coloca esta información en forma de matriz (usuarios x ítems) la matriz posee una gran proporción sin información. Al comprimir los datos, de igual forma como sucede las imágenes o ficheros de texto, los datos comprimidos posarían casi toda la información, pero de forma más “condensada”. Trabajar con esta información suele ser mucho más eficiente y precisa. La reducción de dimensionalidad convierte los datos multidimensionales, que son altamente dispersos, en información más concentrada y densa

### **C) Aprendizaje Semi-supervisado**

El aprendizaje semi-supervisado opera con conjuntos de datos en los que únicamente una porción se encuentra etiquetada y el resto no. Normalmente la porción etiquetada es mucho más pequeña que la no etiquetada. La mayoría de estos algoritmos son una mezcla de técnicas de aprendizaje supervisado y no supervisado.

### **D) Aprendizaje por Refuerzo**

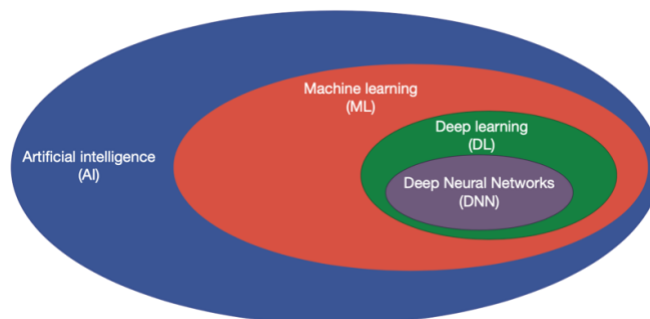
Según Bobadilla (2020) es una de las áreas más innovadoras y con gran futuro, ya que se inspira en mecanismos naturales. El algoritmo de aprendizaje recibe información de un entorno real o simulado. Cuando el sistema realiza una acción es compensado o penalizado, tal y como sucede con los seres vivos.

A estos algoritmos de aprendizaje se les denomina agentes y pueden aprender siguiendo los principios de la evolución natural. Los agentes aprenden “políticas” que maximizan o minimizan las penalizaciones.

## 1.1 Deep Learning

Magnus Ekman (2021) define Deep Learning como una clase de algoritmos de Machine Learning que utiliza múltiples capas de unidades computacionales en la que cada capa aprende su propia representación de los datos de entrada, estas capas representativas son combinadas por capas posteriores de forma jerárquica.

Una de las partes fundamentales de Deep Learning son las redes neuronales profundas. Para poner en contexto las redes neuronales profundas y el aprendizaje profundo se debe mencionar la relación que poseen estos conceptos. No solo entre ellos sino con definiciones como Inteligencia artificial y Machine Learning. Inteligencia artificial es la ciencia que engloba los conceptos mencionados, de forma que las técnicas de Machine Learning son las que permiten que los sistemas aprendan patrones de parámetros. Las redes neuronales profundas son a su vez un subcampo de las técnicas de Machine Learning, en la que se utilizan redes neuronales profundas para el aprendizaje de los algoritmos. A continuación, se muestra un diagrama que ilustra las relaciones mencionadas



*Ilustración 1: Relación entre Inteligencia artificial, Machine Learning, aprendizaje profundo y redes neuronales profundas*

*Fuente: Ekman (2021)*

## 1.2 Perceptrón de Rosenblatt

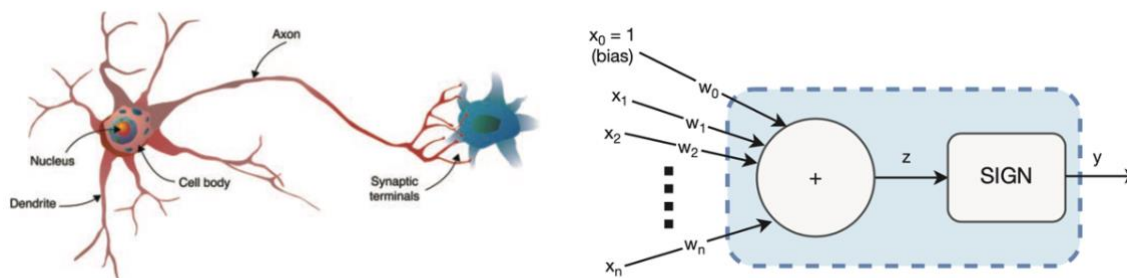
El perceptrón de Rosenblatt basa su arquitectura en el modelo de una neurona biológica. Una neurona biológica posee un cuerpo celular, múltiples dendritas y un axón. Las conexiones entre las neuronas se denomina sinapsis, este proceso consiste en que las dendritas reciben el estímulo de otras neuronas. Si el estímulo es suficiente la neurona activa un estímulo de salida el cual se propaga mediante el axón y se transmite a alguna otra neurona.

Los estímulos pueden ser excitatorios o inhibitorias, las excitatorias provocan una salida por parte de la neurona y en las inhibitorias se previene o evita provocar una salida. El perceptrón consiste en una unidad computacional con un número de entradas (las entradas simulan las dendritas de una neurona biológica) las cuales se asocian a un peso de entrada y una sola salida (la salida simula el axón de la neurona biológica). Estos pesos de entrada definen el nivel de relevancia de un parámetro con respecto a la salida.

Las entradas suelen denominarse  $x_0, x_1, \dots, x_n$  siendo  $x_0$  el parámetro bias o sesgo y  $x_1, \dots, x_n$  como las posibles entradas. Para la salida se acostumbra a utilizar la variable  $y$ . Como mencionaba anteriormente, cada entrada se asocia a un peso de entrada el cual se denomina peso sináptico.

Cada entrada se multiplica por el peso correspondiente antes de ser presentada a la unidad computacional, éste se basa en el cuerpo celular de la neurona biológica. La unidad computacional realiza la suma de los resultados obtenidos de la multiplicación de cada una de las entradas por sus pesos correspondientes. Posterior mente la suma de estos valores se pasan por la función de activación la cual se puede representar de la siguiente forma  $y = f(z)$  donde  $z$  representa la sumatoria de los resultados obtenidos de la multiplicación de las entradas y los pesos. Ekman (2021) menciona que la función de activación para un perceptrón simple es la función signo.

A continuación, se puede observar la neurona biológica y la arquitectura del perceptrón.



*Ilustración 2 Comparación de la neurona biológica y la arquitectura del perceptrón*

*Fuente: Ekman (2021)*

Ekman (2021) resume que el perceptrón es un tipo de neurona artificial. El cual suma las entradas para determinar un valor  $z$ , siendo ésta la entrada de una función de activación. El perceptrón utiliza la función signo como función de activación, sin embargo, otras neuronas artificiales utilizan funciones diferentes.

### **1.3 Funciones de Activación**

Aggarwal (2018) menciona que la selección de la función de activación es una parte crítica del diseño de las redes neuronales. Como se menciona en la sección 1.3 el perceptrón de Rossenblatt utiliza la función signo, esto se debe a que es necesario predecir la etiqueta de clase binaria. Sin embargo, es posible tener otros tipos de funciones de activación.

Según Aggarwal (2018) la importancia de las funciones de activación no lineales toma relevancia cuando se pasa del perceptrón de una sola capa a las arquitecturas de varias capas. Se pueden usar diferentes tipos de funciones no lineales como la función signo, sigmoide o tangente hiperbólica. Se utiliza la notación  $\Phi$  para denotar la función de activación, por lo tanto, se puede re-escribir la ecuación de la salida de una neurona de la siguiente manera:

$$\hat{y} = \Phi(\bar{W} \cdot \bar{X})$$

A continuación, se ilustran algunas funciones de activación:

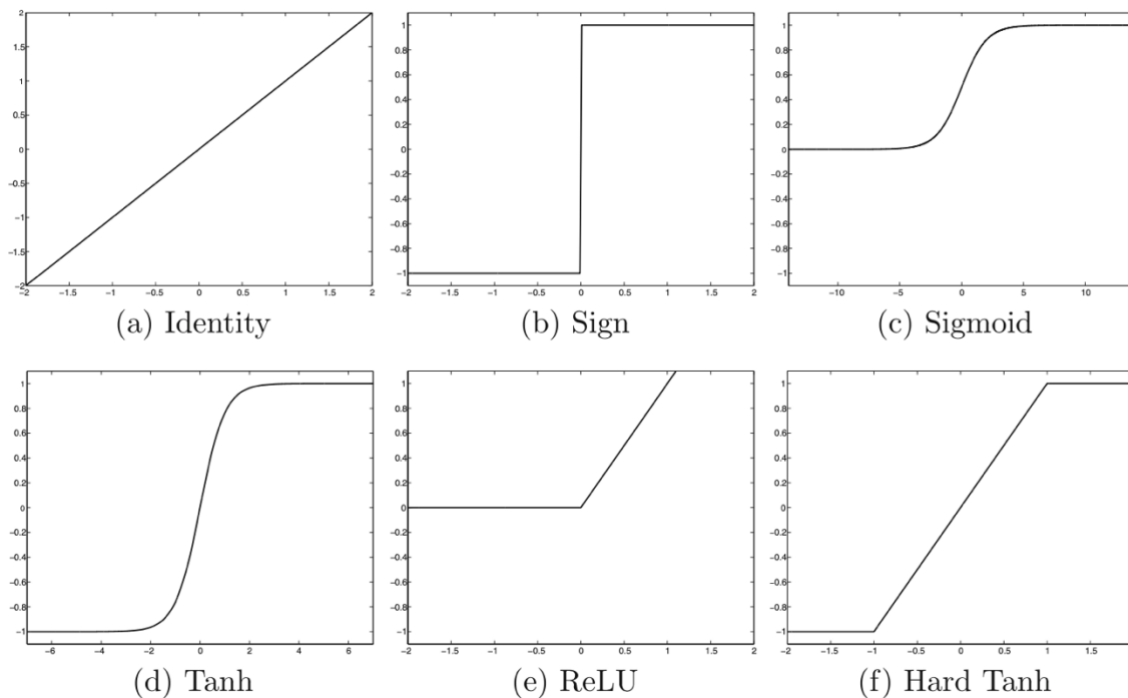


Ilustración 3 Funciones de activación

Fuente Aggarwal (2018)

La función de activación más básica es la función de identidad o activación lineal

$$\Phi(v) = v \quad (\text{Función de activación identidad})$$

Esta función es comúnmente utilizada en la salida del nodo, cuando el valor objetivo es un valor real, aunque también se utiliza para salidas discretas cuando es necesario configurar una función de pérdida sustituta suavizada.

Aggarwal (2018) menciona que las funciones de activación clásicas que se utilizaban en el desarrollo temprano de las redes neuronales eran la función signo, sigmoide y la tangente hiperbólica.

$$\Phi(v) = \text{sign}(v) \quad (\text{Función signo})$$

$$\Phi(v) = \frac{1}{1 + e^{-v}} \quad (\text{Función sigmoide})$$

$$\Phi(v) = \frac{e^{2v} - 1}{e^{2v} + 1} \quad (\text{Función tangente hiperbólica})$$

Si bien, la función signo se utiliza para mapear salidas binarias en el momento de la predicción, su no diferenciabilidad representa un obstáculo para crear la función de pérdida en el momento del entrenamiento.

La función de activación sigmoidea genera un valor entre cero y uno, lo cual es útil para realizar cálculos que deben interpretarse como probabilidades, por lo que también es útil para crear resultados probabilísticos y construir funciones de pérdida derivadas de modelos de máxima verosimilitud.

La función tangente hiperbólica tiene una forma muy parecida a la función sigmoidea excepto que se re-escala horizontalmente y se traslada/re-escala verticalmente a  $[-1, 1]$ . La función tangente hiperbólica y sigmoidea se relacionan de la siguiente manera:

$$\tanh(v) = 2 \cdot \text{sigmoid}(2v) - 1$$

La función tangente hiperbólica es preferible a la sigmoidea cuando se desea que los resultados de los cálculos sean tanto positivos como negativos. Además, posee



una media centrada y un gradiente más grande (debido al estiramiento) con respecto a la función sigmoide lo que resulta en un entrenamiento más sencillo. Tanto la función sigmoide como la función tangente hiperbólica han sido herramientas históricas elegidas para incorporar la no-linealidad en las redes neuronales. Sin embargo, en los últimos años se han vuelto más populares varias funciones de activación lineal por partes:

$$\Phi(v) = \max\{v, 0\} = \begin{cases} 0 & \text{para } v < 0 \\ v & \text{para } v > 0 \end{cases} \text{ (Rectified Linear Unit [ReLU])}$$

$$\Phi(v) = \max\{\min[v, 1], -1\} = \begin{cases} -1 & \text{para } v < -1 \\ v & \text{para } -1 \leq v \leq 1 \\ 1 & \text{para } v > 1 \end{cases} \text{ (Tangente Hiperbolica dura)}$$

Las funciones de activación ReLU y tangente hiperbólica dura han reemplazado en gran medida las funciones de activación sigmoide y tangente hiperbólica en las redes neuronales modernas debido a la facilidad para entrenar redes neuronales multicapa con estas funciones de activación.

#### 1.4 Arquitecturas de Red

Ekman (2021) define las arquitecturas de red como múltiples neuronas conectadas entre ellas para formar una red más compleja. Las redes neuronales conocidas como redes neuronales artificiales (ANN) o redes neuronales simuladas (SNN), son un subconjunto de Machine Learning y juegan un papel clave en los algoritmos de Deep Learning.

Estas arquitecturas se componen por capas de nodos, las cuales contienen una capa de entrada, una o más capas ocultas y una capa de salida. Cada nodo o neurona se conecta con otro, tiene un peso y un umbral asociado. Si la salida de cualquier nodo individual se activa por encima del umbral especificado, el nodo se activa y envía datos a la siguiente capa de la red. En el caso contrario el nodo permanece inactivo y no enviará ningún dato a la siguiente capa.

Las arquitecturas clave en las aplicaciones de las redes neuronales son la siguientes

- Redes neuronales totalmente conectadas de propagación hacia adelante:

Se componen de una capa de entrada, una o más capas ocultas y una capa de salida. Estas redes se componen de neuronas sigmoides en lugar de perceptrones, debido a que los problemas del mundo real no suelen ser lineales. Los datos generalmente se alimentan en estos modelos para entrenarlos y son la base para la visión por computadora, procesamiento de lenguaje natural y otras redes neuronales.

(IBM Cloud Education, 2020)

- Redes neuronales convolucionales

Según Ekman (2021) la característica principal de este tipo de red neuronal es que las neuronas individuales no poseen pesos únicos, sino que utilizan los mismos pesos que otras neuronas, a esto se le conoce como *weight sharing*.

Si comparamos la arquitectura de las redes neuronales convolucionales con las redes neuronales completamente conectadas de propagación hacia adelante notaremos similitud entre ambas, pero con la diferencia de que las redes neuronales convolucionales poseen considerablemente menos conexiones.

En lugar de ser completamente conectada es una red parcialmente conectada. Suelen ser utilizadas en aplicaciones de reconocimiento de imágenes, de patrones o visión artificial debido a que aprovechan los principios del álgebra lineal, particularmente la multiplicación de matrices para identificar patrones dentro de una imagen.

- Redes neuronales Recurrentes

A diferencia de las redes neuronales de propagación hacia adelante las Redes neuronales recurrentes tienen conexiones hacia atrás, esto quiere decir que poseen ciclos de retroalimentación. Estos algoritmos son aprovechados al utilizar datos de series temporales para hacer predicciones de resultados futuros, como predicciones en el mercado de acciones o previsiones de ventas. (IBM Cloud Education, 2020)

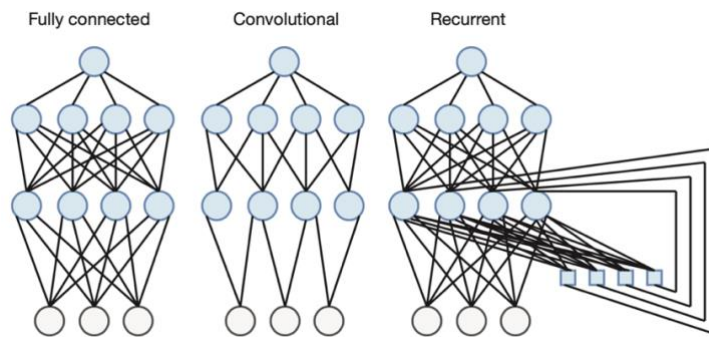


Ilustración 4 Comparación grafica de las arquitecturas de red

Fuente: Ekman (2021)

### 1.5 Descenso de Gradiente (*Gradient Descent*)

Según Ekman (2021) una forma de abordar el tema de identificar los pesos que, dadas las entradas de la muestra de entrenamiento, obtengamos el resultado esperado de la salida de la red. Podemos decir que es básicamente lo mismo que resolver la siguiente ecuación:

$$y - \hat{y} = 0$$

En la cual  $y$  representa la salida deseada y  $\hat{y}$  la salida producida por la red. En la práctica no se suele tener una única muestra de entrenamiento sino un set completo de datos. Para determinar el error, podemos utilizar el error cuadrático medio de el set de datos.

$$\frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2 \quad (\text{función error cuadrático medio})$$

Se observa la notación dentro del paréntesis en la función del error cuadrático medio notaremos que es la misma ecuación que describe la similitud entre el resultado esperado y el obtenido de la red. Sin embargo, en la mayoría de los casos el error cuadrático medio suele ser mayor a cero por lo que intentar hallar los pesos que produzcan un error cuadrático medio igual a 0 puede representar un problema. Por ello se suele abordar esta situación como un problema de optimización en el cual se busca determinar los pesos que minimizan el valor de la función de error.

Según Ekman (2021) en la mayoría de los problemas de Deep Learning, no es factible encontrar una solución única y cerrada para la minimización de la función de error. Por lo tanto, se suele utilizar un método numérico conocido como descenso de gradiente. Éste es un método iterativo que inicia con una suposición inicial de la solución, la cual es refinada glandularmente. A continuación, se ilustra el método de descenso del gradiente.

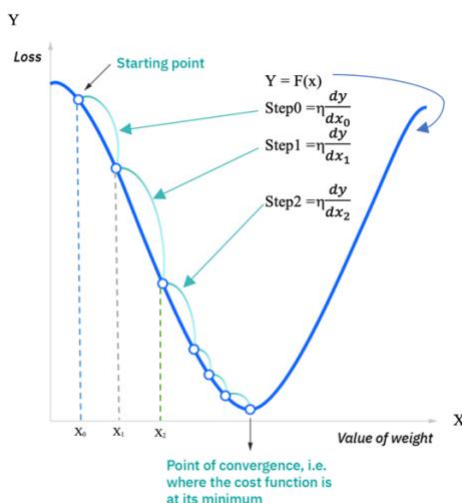


Ilustración 5 Método descendiente de gradiente

Fuente: IBM Cloud Education (2020)

En la figura se observa que la suposición inicial está representada en  $x_0$ , este valor posteriormente se puede evaluar en la función  $f(x)$  y obtener su valor en el plano vertical, además de obtener su derivada. De no estar en el valor mínimo del plano

vertical se incrementa o reduce el valor de  $x_0$  en cierta proporción. La decisión de incrementar o reducir dicho valor se toma a partir del signo de la derivada. Si la derivada es negativa indica que el desplazamiento se realiza sobre la porción de la curva que posee pendiente negativa, por lo tanto, se debe incrementar el valor de  $x_0$  para descender en la pendiente y ubicar el mínimo. En el caso contrario, el punto se encontrará en la porción de la curva cuya pendiente es positiva por lo tanto se debe reducir el valor de  $x_0$  para acercarse al mínimo.

Además de indicar en qué dirección ajustar el valor de  $x_0$ , la derivada provee un indicador de cuál es el valor actual de  $x$  y que tan alejado se encuentra el valor que minimiza  $y$ . El descendiente del gradiente aprovecha esta propiedad de la derivada para determinar en qué proporción debe ajustar el valor de  $x$ . A continuación se observa la ecuación de ajuste del descendiente del gradiente

$$x_{n+1} = x_n - \eta f'(x_n)$$

Donde eta ( $\eta$ ) representa el factor de aprendizaje. El tamaño del paso depende del factor de aprendizaje y la derivada, de manera que este disminuye si la derivada se reduce. En la ilustración 4 se puede observar como el tamaño del paso se reduce conforme la derivada se acerca a 0. Ya que el algoritmo converge en el mínimo, el hecho de que el tamaño del paso se acerque a 0 implica que la derivada se aproxima a 0.

Si el factor de aprendizaje es muy grande el descenso del gradiente puede pasar la solución y no llegar a la convergencia, en cambio si el factor de aprendizaje es muy pequeño los pasos podrían tardar tiempos muy prolongado e incluso no se garantiza que el algoritmo encuentre el mínimo local, sin embargo, en la práctica, se ha demostrado que funciona bien para redes neuronales.

## **1.6 Descenso del Gradiente en mini lotes (Mini-Batch Gradient Descent)**

El descenso de gradiente por lotes es una técnica que combina los beneficios del descenso del gradiente por un único lote y el descenso del gradiente estocástico.

En el descenso de gradiente por lote se calcula el promedio de todas las muestras de entrenamiento y luego se utiliza para actualizar los parámetros, lo que significa que solo se calcula el descendiente del gradiente una vez por época (una época es la cantidad de veces que el conjunto completo de datos se introduce en la red neuronal). Sin embargo, si el set de datos es demasiado grande esta aproximación no es la mejor debido a que se debería de calcular el gradiente de todas las muestras en un solo paso. Para atacar esta problemática se utiliza el descenso del gradiente estocástico, en esta aproximación se considera un ejemplo de entrenamiento a la vez. Primero se toma un ejemplo de entrenamiento, luego se ingresa a la red y se calcula el gradiente y por último se utiliza el gradiente para actualizar los pesos.

Ekman (2021) menciona que recorrer todo el conjunto de datos brinda estimaciones más precisas del gradiente, sin embargo, requiere de muchos más cálculos antes de actualizar los pesos. Una forma eficaz de calcular el gradiente es utilizar un pequeño conjunto de ejemplos de entrenamientos conocido como mini-lote. Esto permite actualizaciones de pesos más frecuentes que el descenso del gradiente en un único lote y además permite obtener una estimación más precisa que cuando se utiliza un único ejemplo. Las implementaciones de hardware modernas, en particular las unidades de procesamiento de gráficos han demostrado poseer buen rendimiento al calcular un mini lote completo en paralelo.

## 1.7 Retropropagación

El algoritmo de retropropagación consiste en un pase hacia adelante en el que se presentan los ejemplos del entrenamiento de la red. Posteriormente se realiza un pase hacia atrás en el cual se ajustan los pesos del descendiente de gradiente.

En el pase hacia adelante se presenta una muestra de entrenamiento a la red, para comparar la salida obtenida con la salida deseada.

En el pase hacia atrás se calculan las derivadas parciales con respecto a los pesos, dichas derivadas se utilizan para ajustar los pesos para acercar el valor de salida obtenido de la red con el valor deseado.

El propósito de la retropropagación es calcular el gradiente de una red neuronal de multinivel. A continuación, se describe el proceso para aplicar el descendiente del gradiente en una red multinivel con una única neurona por capa y con 2 entradas en la primera neurona.

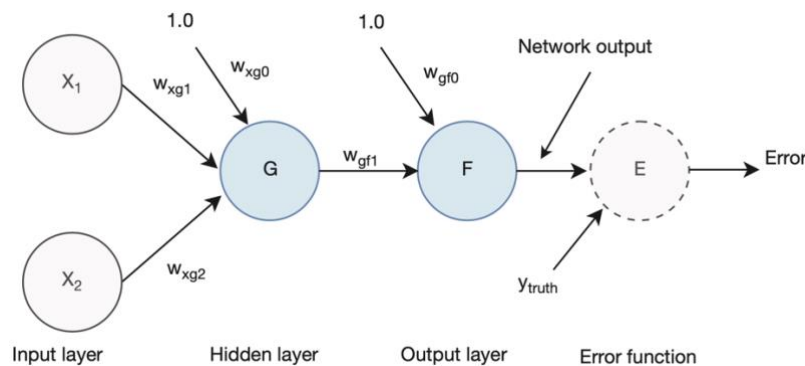


Ilustración 6 Red multinivel con una neurona por capa

Fuente: Ekman (2021)

Como se observa en la imagen las 2 neuronas fueron nombradas G y F, G posee 3 pesos ajustables y F posee 2 pesos ajustables. En total tenemos una red con 5 pesos ajustables. Se desea que el algoritmo determine de forma automática los valores de estos pesos para cumplir con la funcionalidad deseada de la red. En la figura se observa la nomenclatura convencional de pesos donde la primera letra del subíndice representa la capa de origen y la segunda letra la capa de destino, es decir, la capa de donde forma parte el peso, por ejemplo, en el caso  $W_{gf1}$ , la capa de origen es G y la capa de destino es F, por lo tanto, este peso corresponde a la segunda neurona (F).

A diferencia del perceptrón de Rosenblatt las funciones de activación utilizadas son, la función tanh para la neurona G y la función logística sigmoidea para F. Esto se debe a que uno de los requisitos para aplicar el descendiente de gradiente en redes de multinivel es que la función a la cual se aplique debe ser diferenciable. Utilizar estas 2 funciones permiten que un pequeño cambio en alguno de los pesos se traduzca en un pequeño cambio en la salida. En cambio, en el perceptrón se utiliza la función signo y cualquier pequeño cambio que se realice en alguno de los pesos no producirá ningún cambio, hasta que este sea lo suficientemente significativo para producir que el perceptrón volteé su salida.

La función de la red neuronal implementa la siguiente función:

$$\hat{y} = S\left(W_{gf0} + W_{gf1} \tanh(W_{xg0} + W_{xg1}x_1 + W_{xg2}x_2)\right)$$



El algoritmo descendiente de gradiente se utiliza para minimizar una función. Para ello se debe definir la función de error o función de perdida. En este caso, por simplicidad se utilizará como función de perdida la función del error cuadrático medio, sin embargo, no es óptima en combinación con una función sigmoide como función de actuación. La ecuación a continuación combina la fórmula del error cuadrático medio con la función de la red neuronal, dicha combinación es la función que se debe optimizar.

$$Error = \left( \left( y - S \left( W_{gf0} + W_{gf1} \tanh(W_{xg0} + W_{xg1}x_1 + W_{xg2}x_2) \right) \right) \right)^2$$

Como se menciona anteriormente se minimiza esta función calculando el descendiente del gradiente con respecto a los pesos **W**, luego se multiplican por el factor de aprendizaje y se resta este resultado con la suposición inicial de los valores de los pesos. Es importante recordar que el propósito es ajustar los pesos W, esto permite que las variables de entrada X sean consideradas como constantes.

Ekman (2021) menciona que una forma no optima de resolver este problema es calcular el gradiente numéricamente. Presentar un ejemplo de entrada a la red, calcular y almacenar su salida. Posteriormente agregar  $\Delta W$  a uno de los pesos y calcular la nueva salida, en este momento se puede calcular  $\Delta y$ . Una aproximación de la derivada parcial es ahora  $\frac{\Delta y}{\Delta W}$ . Este proceso debe repetirse para cada uno de los pesos, al finalizar se habrá calculado el gradiente. Desafortunadamente, realizar este proceso es altamente costoso computacionalmente ya que debe de realizarse el proceso n+1 veces donde n es la cantidad de pesos en la red y el +1 representa la ejecución inicial sin modificaciones en los pesos.

En este momento es donde la retropropagación ayuda a resolver el problema de forma más optimizada y por lo tanto con menor costo computacional. El punto de partida es descomponer la ecuación a optimizar en expresiones más pequeñas.

Se iniciará con una función que calcule la entrada de la función de activación de la neurona G.

$$z_g(W_{xg0}, W_{xg1}, W_{xg2}) = W_{xg0} + W_{xg1}X_1 + W_{xg2}X_2$$

Luego la función de activación de la neurona G

$$g(z_g) = \tanh(z_g)$$

Seguidamente la función que representa las entradas a la función de activación de la neurona F

$$z_f(W_{gf0}, W_{gf1}, g) = W_{gf0} + W_{gf1}g$$

Posteriormente la función de activación de la neurona F

$$f(z_f) = S(z_f)$$

Finalmente, la función de error:

$$e(f) = \frac{(y - f)^2}{2}$$

El “2” del denominador en la función de error sirve para simplificar la solución en pasos posteriores. El hecho de haber separado la función de error en múltiples expresiones permite reescribirla de la siguiente manera:

$$\text{error}(W_{gf0}, W_{gf1}, W_{xg0}, W_{xg1}, W_{xg2}) = e \circ f \circ z_f \circ g \circ z_g$$

Al representar la función de error como una composición de múltiples funciones, permite utilizar la regla de la cadena para calcular las derivadas parciales del error con respecto a las variables de entrada  $W_{gf0}, W_{gf1}, W_{xg0}, W_{xg1}$  y  $W_{xg2}$ . A continuación, se presentan las derivadas parciales resultantes:

$$\frac{\partial e}{\partial W_{gf0}} = \frac{\partial e}{\partial f} \cdot \frac{\partial f}{\partial z_f} \cdot \frac{\partial z_f}{\partial W_{gf0}}$$

$$\frac{\partial e}{\partial W_{gf1}} = \frac{\partial e}{\partial f} \cdot \frac{\partial f}{\partial z_f} \cdot \frac{\partial z_f}{\partial W_{gf1}}$$

$$\frac{\partial e}{\partial W_{xg0}} = \frac{\partial e}{\partial f} \cdot \frac{\partial f}{\partial z_f} \cdot \frac{\partial z_f}{\partial g} \cdot \frac{\partial g}{\partial z_g} \cdot \frac{\partial z_g}{\partial W_{xg0}}$$

$$\frac{\partial e}{\partial W_{xg1}} = \frac{\partial e}{\partial f} \cdot \frac{\partial f}{\partial z_f} \cdot \frac{\partial z_f}{\partial g} \cdot \frac{\partial g}{\partial z_g} \cdot \frac{\partial z_g}{\partial W_{xg1}}$$

$$\frac{\partial e}{\partial W_{xg2}} = \frac{\partial e}{\partial f} \cdot \frac{\partial f}{\partial z_f} \cdot \frac{\partial z_f}{\partial g} \cdot \frac{\partial g}{\partial z_g} \cdot \frac{\partial z_g}{\partial W_{xg2}}$$

Como se puede observar las derivadas parciales presentan similitudes entre ellas. Los primeros dos factores son exactamente iguales para las cinco fórmulas, y tres de ellas comparten dos factores más. He aquí porque la retropropagación es óptima. En vez de recalcular el gradiente una y otra vez, se calcula las subexpresiones una sola vez y se reúsan para cada derivada parcial al momento de ser necesario. A continuación, se presenta el cálculo de la primera derivada parcial  $\frac{\partial e}{\partial W_{gf0}}$ :

$$\frac{\partial e}{\partial f} = \frac{\partial \frac{(y-f)^2}{2}}{\partial f} = \frac{2(y-f)}{2} \cdot (-1) = -(y-f)$$

$$\frac{\partial f}{\partial z_f} = \frac{\partial (s(z_f))}{\partial f} = s'(z_f)$$

$$\frac{\partial z_f}{\partial W_{gf0}} = \frac{\partial(W_{gf0} + W_{gf1}g)}{\partial W_{gf0}} = 1$$

Por lo tanto:

$$\frac{\partial e}{\partial W_{gf0}} = \frac{\partial e}{\partial f} \cdot \frac{\partial f}{\partial z_f} \cdot \frac{\partial z_f}{\partial W_{gf0}} = -(y - f) \cdot S'(z_f)$$

$Y$  proviene del ejemplo de entrenamiento,  $f$  y  $z_f$  son calculados al realizar el pase hacia adelante en la red.

Al calcular la derivada del error con respecto a  $W_{gf1}$  se observa que el resultado es muy similar a excepción por un tercer factor, el cual proviene de realizar la derivada parcial  $\frac{\partial z_f}{\partial W_{gf1}}$

$$\frac{\partial z_f}{\partial W_{gf1}} = \frac{\partial(W_{gf0} + W_{gf1}g)}{\partial W_{gf1}} = g$$

Combinando este resultado con los primeros 2 factores se obtiene lo siguiente:

$$\frac{\partial e}{\partial W_{gf1}} = \frac{\partial e}{\partial f} \cdot \frac{\partial f}{\partial z_f} \cdot \frac{\partial z_f}{\partial W_{gf1}} = -(y - f) \cdot S'(z_f) \cdot g$$

Es prácticamente el mismo resultado de la derivada parcial con respecto a  $W_{gf0}$  pero multiplicado por  $g$ , el cual es la salida de la neurona G, a este punto dicho valor se habrá calculado en el pase hacia adelante. A continuación, se muestra el cálculo de las 5 derivadas parciales:

$$\frac{\partial e}{\partial W_{gf0}} = -(y - f) \cdot S'(z_f)$$

$$\frac{\partial e}{\partial W_{gf1}} = -(y - f) \cdot S'(z_f) \cdot g$$

$$\frac{\partial e}{\partial W_{xg0}} = -(y - f) \cdot S'(z_f) \cdot W_{gf1} \cdot \tanh'(z_g)$$

$$\frac{\partial e}{\partial W_{xg1}} = -(y - f) \cdot S'(z_f) \cdot W_{gf1} \cdot \tanh'(z_g) \cdot x_1$$

$$\frac{\partial e}{\partial W_{xg2}} = -(y - f) \cdot S'(z_f) \cdot W_{gf1} \cdot \tanh'(z_g) \cdot x_2$$

Se puede notar un patrón en las ecuaciones. Se inicia con la derivada de la función de error, la cual se multiplica por la derivada de la función de activación para la salida de la neurona, a esto se le puede denominar como: error de la neurona de salida. La derivada parcial con respecto a un peso de entrada para esa neurona se obtiene multiplicando el error de la neurona por el valor de entrada a ese peso. En el caso del peso de sesgo el valor de entrada es 1, por ende, la derivada parcial es simplemente el error de la neurona, para cualquier otro peso se debe multiplicar el error de la neurona por la salida de la neurona precedente, el cual es la entrada del peso.

Posteriormente se toma el error para la neurona de salida, se multiplica por el peso que se conecta a la neurona anterior y se multiplica el resultado por la derivada de la función de activación de la neurona anterior. A esto se le denomina el error de la neurona anterior (neurona G del ejemplo). Estos cálculos propagan el error hacia atrás desde la salida de la red hacia el comienzo de la red, de ahí el nombre algoritmo de retropropagación.

## 1.8 Sets de Datos

Según Ekman (2021) obtener los ejemplos de entrenamiento de la red puede ser un gran reto. Sin embargo, la popularidad que recientemente ha ganado las técnicas de Deep Learning, ha llevado a la construcción de grandes bases de datos en línea de imágenes, videos, texto en lenguaje natural entre otros, lo cual ha hecho posible obtener grandes conjuntos de entrenamiento. Al utilizar técnicas de aprendizaje supervisado no basta con conocer la entrada de la red, sino que también se debe conocer la salida esperada de la misma. El proceso de asociar cada entrada con la salida esperada se denomina etiquetado, dicho procedimiento suele ser manual.

Una definición bastante acertada de un set de datos puede ser la siguiente: Un set de datos consiste en una colección de ejemplos de entrenamiento etiquetados, que puede utilizarse para entrenar modelos de Machine Learning.

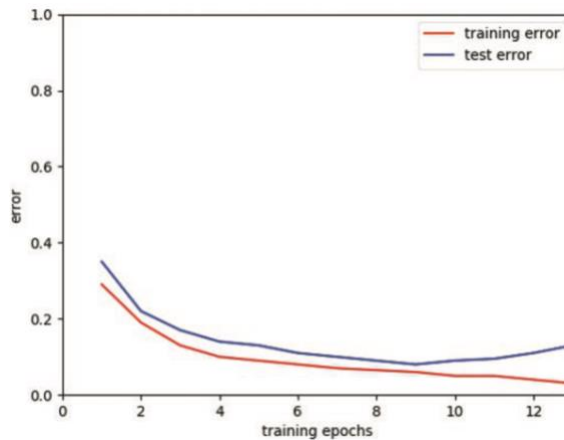
Debido a que los modelos aprenden de los datos de entrada son susceptibles al problema de “basura-entra/basura-sale”. Por lo que es muy importante asegurarse que los sets de datos a utilizar sean de alta calidad. Uno de los problemas comunes que se puede encontrar en un set de datos es el sesgo humano. Un ejemplo de esto es el set de datos “CalebFaces Attributes” (Liu et al., 2015) el cual se deriva del conjunto de datos de CelebFaces (Sun, Wang y Tang, 2013). El conjunto de datos consiste en un gran número de imágenes de rostros de celebridades, sin embargo, este set de datos contiene en una proporción bastante grande imágenes de individuos blancos de aspecto joven como representación de la sociedad. Entrenar un modelo con este conjunto de datos sesgado puede tener como efecto un pobre desempeño en individuos con tonos de piel más oscuros. Ekman (2021) recalca que, aunque las intenciones sean buenas se deben considerar las consecuencias involuntarias, ya que un set de datos influenciado por el racismo estructural en la sociedad puede resultar en un modelo que discrimine a las minorías.

## 1.9 Set de entrenamiento, Set de prueba y Generalización

Una de las razones principales de los modelos de Machine Learning es la generalización, esto se debe a que los modelos no solo deben ser capaces de producir la predicción correcta de los datos utilizados en el entrenamiento, sino que deben ser capaces de proveer la predicción acertada para los datos no antes vistos. Usualmente se divide el set de datos en un conjunto de entrenamiento y un conjunto de prueba. El set de datos de entrenamiento es el que se utiliza durante el entrenamiento del modelo y el set de datos de prueba es el que se utiliza para evaluar el desempeño del modelo sobre datos desconocidos.

En algunas ocasiones sucede que el modelo presenta un buen rendimiento en el conjunto de datos de entrenamiento, pero un desempeño pobre en el conjunto de datos de prueba, esto es una indicación de que el modelo no ha sido capaz de aprender la solución general para resolver problemas similares mas no idénticos. A este fenómeno se le conoce como *overfitting*, para evitar este fenómeno se debe monitorear el error de entrenamiento y el error de prueba durante el entrenamiento para establecer si el modelo está aprendiendo a generalizar.

Ekman (2021) menciona que de forma general el error de entrenamiento va a reflejar una tendencia descendiente hasta que finalmente se nivela. Sin embargo, el error de prueba refleja una curva en U donde muestra un comportamiento descendiente al inicio, sin embargo, en algún punto inicia a aumentar nuevamente. Si este aumenta mientras el error de entrenamiento aún está descendiendo, es una señal que el modelo se está sobre ajustando a los valores de entrenamiento. A continuación, se ilustra la forma en la que el error de entrenamiento y el error de prueba evolucionan durante el proceso de entrenamiento.



*Ilustración 7 Evolución de las curvas del error de entrenamiento y error de prueba durante el proceso de aprendizaje*

*Fuente: Ekman (2021)*

El sobreajuste no es la única razón para un pobre desempeño, también puede deberse a que el conjunto de datos de entrenamiento no es representativo en el conjunto de datos prueba o de los datos de producción. Una forma efectiva de evitar el sobreajuste es incrementar el tamaño del set de entrenamiento, sin embargo, existen otras técnicas, conocidas como *técnicas de regulación*, uno de los métodos más comunes es el método *early stopping*, el cual consiste en monitorear el error del set de prueba durante el entrenamiento y detenerse en el momento en el que este comience a aumentar.

En ocasiones el error fluctúa durante el entrenamiento y no se mueve estrictamente en una dirección u otra, por lo que no es fácil determinar en qué momento del entrenamiento detenerse. Una aproximación para determinar cuándo detenerse es guardar los pesos del modelo en intervalos fijo durante el entrenamiento, en otras palabras, es crear *checkpoints* del modelo durante el entrenamiento. Al finalizar el proceso se identifica el punto con menor error de prueba y se recarga el modelo correspondiente

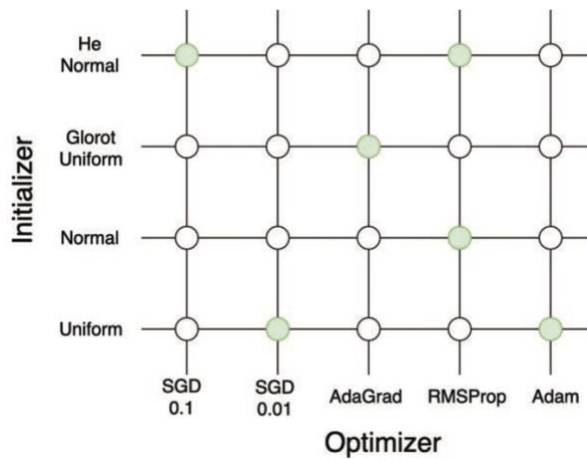


### 1.10 Ajuste de hiperparámetros y fuga de información en el conjunto de pruebas

Según Ekman (2021) es importante no filtrar información del conjunto de prueba en el proceso de entrenamiento, ya que hacerlo puede provocar que el modelo memorice el conjunto de prueba y resulte en una evaluación demasiado optimista del buen rendimiento del modelo en comparación con el rendimiento en producción. Al entrenar un modelo, es necesario ajustar varios parámetros que no son ajustados por el algoritmo de aprendizaje.

Estos parámetros se denominan *hiperparámetros*, algunos de ellos son tasa de aprendizaje, topología de la red y tipo de función de activación. El ajuste de dichos parámetros puede ser un proceso manual o automatizado. Si los hiperparámetros cambian con base en el desempeño del modelo en el conjunto de datos de prueba, significa que dicho conjunto corre el riesgo de influir en el proceso de entrenamiento, por lo que se induce la fuga de información del conjunto de prueba al proceso de entrenamiento. Para evitar esta fuga se utiliza un conjunto de datos de validación, el cual es utilizado para evaluar los ajustes de los hiperparámetros antes de realizar la validación final del conjunto de datos de prueba.

Una técnica popular para el ajuste de los hiperparámetros es la búsqueda de cuadrícula (grid search). A continuación, se muestra la búsqueda de cuadrícula para el caso de dos Hiperparámetros (optimizador e inicializador).



*Ilustración 8 Búsqueda de cuadrícula para 2 hiperparámetros.*

*Fuente: Ekman (2021)*

Simplemente se crea una cuadrícula en la que cada eje representa un hyperparametro, en el caso de dos hyperparemtros se convierte en una cuadrícula en 2D, pero se puede exteder a una cantidad de n dimensiones, aunque podemos visualizar como máximo tres dimensiones. Cada intersección en la cuadrícula representa una combinación distinta de los valores de los hyperparametros. Los círculos de la imagen anterior representan todas las posibles combinaciones de los valores que estos pueden tomar.

Podría realizarse una prueba para cada punto en la cuadrícula para determinar la mejor combinación sin embargo es altamente costoso computacionalmente debido a que el número de combinaciones crece rápidamente con respecto al número de hiperparámetros a evaluar. Una alternativa es seleccionar de forma aleatoria un subset de todas las combinaciones posibles (esto es representado por los puntos verdes de la imagen). Sin embargo, también se puede utilizar una aproximación híbrida, para ello e inicia con una búsqueda en la cuadrícula para identificar una o un par de combinaciones prometedoras, con el objetivo de crear una cuadrícula más detallada alrededor de estas combinaciones y hacer una búsqueda de cuadrícula exhaustiva en esta parte ampliada del espacio de búsqueda.

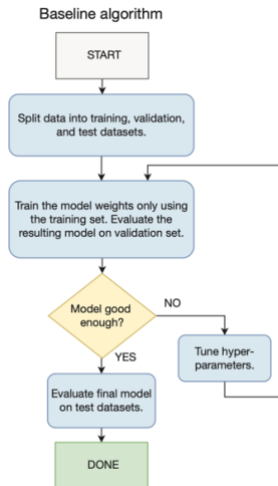
### 1.11 Conjunto de validación para evitar el sobreajuste

Según Ekman (2021) el proceso de ajuste de hiperparámetros introduce el riesgo de sobreajustar los parámetros. Aunque no se utiliza el conjunto de datos de prueba para entrenar los pesos del modelo, se usa para decidir qué conjunto de hiperparámetros es el mejor. Esto induce el riesgo de haber seleccionado un conjunto de hiperparámetros que son particularmente buenos para el conjunto de datos de prueba pero que no son tan buenos para el caso general. Esto es algo sutil en el sentido de que existe el riesgo de sobreajuste incluso si no se posee un circuito de retroalimentación en el que el resultado de un conjunto de hiperparámetros guíen el experimento de un siguiente conjunto. Dicho riesgo existe incluso si se deciden todas las combinaciones por adelantado y solo se utiliza el conjunto de datos de prueba para seleccionar el modelo con mejor rendimiento.

Este problema puede resolverse dividiendo el conjunto de datos en un conjunto de datos de entrenamiento, un conjunto de datos de validación y un conjunto de datos de pruebas. Se entrenan los pesos del modelo utilizando el conjunto de datos de entrenamiento y se ajustan los hiperparámetros usando el conjunto de datos de validación, una vez se ha llegado a el modelo final se utiliza el conjunto de datos de prueba para determinar el rendimiento del modelo en datos que aún no ha visto.

El desafío es decidir en cuanto dividir el conjunto de datos original en conjunto de entrenamiento, validación y prueba. Idealmente, se determina caso por caso y depende de la variación en la distribución de datos. En ausencia de dicha información, una división común entre el conjunto de entrenamiento y el conjunto de prueba cuando no hay necesidad de un conjunto de validación es 70/30 (70% en el conjunto de entrenamiento y 30% en el de prueba) o 80/20. En caso de necesitar el conjunto de validación una división típica es 60/20/20.

Para conjuntos de baja varianza se puede utilizar una fracción más pequeña para validación en el caso contrario, es necesaria una proporción más grande. A continuación, se muestra un diagrama con el proceso de entrenamiento utilizando el conjunto de datos de entrenamiento, validación y prueba.



*Ilustración 9 Proceso de entrenamiento utilizando el conjunto de datos de entrenamiento, validación y prueba.*

*Fuente Ekman (2021)*

## 1.12 Inferencia y entrenamiento

El proceso de utilizar la red si actualizar sus pesos se conoce como *inferencia* debido a que se utiliza la red para inferir un resultado. El entrenamiento refiere a determinar los pesos de la red y se realiza antes de implementarla en producción, una vez el modelo se implementa en producción se utiliza únicamente para realizar inferencias.

## 1.13 Redes neuronales Convolucionales CNN

Según Aggarwal (2018) las redes neuronales convoluciones son comúnmente utilizadas en visión artificial para la clasificación de imágenes y detección de objetos. El desarrollo de las redes neuronales convolucionales parte de la comprensión de Hubel y Wiesel del funcionamiento de la corteza visual del gato.

La corteza visual posee regiones pequeñas de células que son sensibles a regiones específicas del campo visual, por lo que, si se estimulan áreas específicas del campo visual, se activaran esas células en la corteza. Sin embargo, las células excitadas también dependerán de la forma y orientación de los objetos en el campo visual. Por ejemplo, los bordes verticales provocan la excitación de algunas células

neuronales, mientras que los bordes horizontales provocan la excitación de otras células neuronales.

Las células se conectan mediante una arquitectura en capas, y este descubrimiento llevó a la conjetura de que los mamíferos usan estas diferentes capas para construir porciones de imágenes en diferentes niveles de abstracción. Desde el punto de vista del aprendizaje automático, este principio es similar al de la extracción de características jerárquicas.

La primera arquitectura basada en dicho principio fue *neocognition*, sin embargo, hubo varias diferencias entre este modelo y las redes neuronales convolucionales modernas, la más significativa es que no se utilizó el concepto de pesos compartidos o “weight Sharing”. En base a esta arquitectura se desarrolló una de las primeras redes totalmente convolucionales LeNet-5. La cual fue utilizada por bancos para identificar números escritos a mano en los cheques.

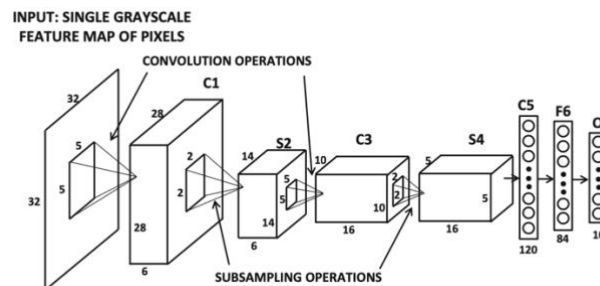


Ilustración 10 Arquitectura de la red LeNet-5

Fuente: Aggarwal (2018)

Según Aggarwal (2018) En las redes neuronales convolucionales, los estados en cada capa se organizan de acuerdo con una estructura de cuadrícula espacial. Estas relaciones espaciales son heredadas de una capa a la siguiente debido a que cada valor de característica se basa en una pequeña región espacial local en la capa anterior. Es importante mantener esta relación espacial entre las celdas de la cuadrícula, ya que la operación de convolución y la transformación a la siguiente capa es dependiente de esas relaciones.

Cada capa en una red convolucional es una estructura cuadriculada tridimensional que posee alto, ancho y profundidad. La profundidad de una capa en una red neuronal convolucional no debe confundirse con la profundidad de la red en sí. El término “profundidad”, al utilizarse en el contexto de una sola capa, hace referencia a el número de canales en la capa, como puede ser el número de canales primarios de la imagen de entrada (RGB, BGR) o el número de mapas de características en las capas ocultas. El término “profundidad” puede utilizarse tanto para referirse al número de mapas de características en cada capa como referirse al número de capas de la red neuronal.

Aggarwal (2018) menciona que la entrada de una red convolucional se organiza en una cuadrícula bidimensional y que los valores de los puntos en la cuadrícula se denominan píxeles, por lo tanto, cada píxel corresponde a una ubicación espacial dentro de la imagen.

Para codificar el color preciso del píxel, es necesario una matriz multidimensional de valores en cada ubicación de la cuadrícula. En el esquema de color RGB, se posee la intensidad de los tres colores primarios rojo, verde y azul respectivamente. Por lo tanto, si las dimensiones espaciales de una imagen son 32x32 píxeles y la profundidad es 3, entonces el número de píxeles en la imagen es 32x32x3. Este tamaño en particular es bastante común. Un ejemplo de esta organización se muestra a continuación.

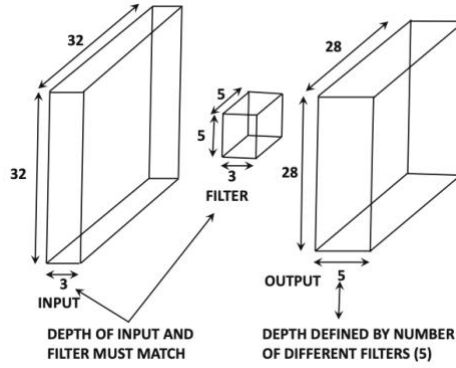


Ilustración 11 Operación Convolución, entrada de 32x32x3 y filtro de 5x5x3

Fuente: Aggarwal (2018)

Es natural representar la capa de entrada como una estructura tridimensional porque dos dimensiones están dedicadas a las relaciones espaciales y la tercera dimensión está dedicada a las propiedades independientes a lo largo de estos canales. Por ejemplo, las intensidades de los colores primarios son las propiedades independientes en la primera capa. En las capas ocultas, estas propiedades independientes corresponden a varios tipos de formas extraídas de las regiones locales de la imagen.

Aggarwal (2018) presenta el siguiente ejemplo, suponga que la entrada en la  $q$ -ésima capa tiene un tamaño  $L_q \times B_q \times d_q$ .  $L_q$  refiere a la altura,  $B_q$  al ancho, y  $d_q$  a la profundidad. En la mayoría de las aplicaciones centradas en imágenes, los valores  $L_q$  y  $B_q$  son los mismos. Sin embargo, en el ejemplo se trabajan con notaciones diferentes para  $L_q$  y  $B_q$  para mantener la generalidad.

En la capa de entrada, estos valores se deciden por la naturaleza de los datos de entrada y su preprocesamiento. En el caso de la organización de 32x32x3 de la imagen, los valores son  $L_1 = 32, B_1 = 32, y d_1 = 3$ . Las capas posteriores poseen exactamente la misma organización tridimensional, excepto que cada una de las cuadrículas de valores bidimensionales  $d_q$  para una entrada en particular ya no puede considerarse como una cuadrícula de píxeles sin procesar. Además, el valor

$d_q$  es mucho mayor a tres para las capas ocultas, esto se debe a el número de propiedades independientes de una región local que pueden llegar a ser relevantes para la clasificación.

En las redes neuronales convolucionales, los parámetros se organizan en agrupaciones de unidades estructurales tridimensionales conocidas como *filtros* o *kernels*. El filtro es usualmente cuadrado en termino de sus dimensiones espaciales, dichas dimensiones suelen ser mucho más pequeñas que las de la capa a la cual se aplica el filtro, en cuanto a la profundidad esta siempre es la misma que la de la capa a la que se aplica. Se puede decir que las dimensiones del filtro en la  $q$ -esima capa son  $F_q \times F_q \times d_q$ . Si observamos la ilustración 10 observaremos que el filtro aplicado a la organización 32x32x3 de la entrada posee las siguientes dimensiones  $F_1 = 5, F_1 = 5, d_q = 3$ . Es común que el valor de  $F_q$  sea un entero pequeño e impar.

La operación de convolución ubica el filtro en todas las posiciones posibles dentro de la imagen o capa oculta, con el objetivo de superponer completamente el filtro con la imagen y realizar un producto escalar entre los parámetros en el filtro y la cuadrícula coincidente en el volumen de entrada. El producto punto se realiza tratando las entradas en la región tridimensional relevante del volumen de entrada y el filtro como vectores de tamaño  $F_q \times F_q \times d_q$ , de manera que los elementos en ambos vectores se ordenen en función de sus posiciones correspondientes en la cuadrícula.



El número de alineaciones posibles entre el filtro y la imagen define la altura y el ancho espacial de la siguiente capa oculta. Las posiciones espaciales relativas de las características en la siguiente capa se definen en función de las posiciones relativas de las esquinas superiores izquierdas de las cuadrículas espaciales correspondientes en la capa anterior. Al realizar las convoluciones en la  $q$ -ésima capa, se puede alinear el filtro en posiciones  $L_{q+1} = (L_q - F_q + 1)$  a lo largo y  $B_{q+1} = (B_q - F_q + 1)$  a lo ancho de la imagen. De esto se obtiene un resultado total de  $L_{q+1} \times B_{q+1}$  posibles productos escalares, definiendo así el tamaño de la siguiente capa. Utilizando el ejemplo de la ilustración 10 se obtiene el siguiente resultado:

$$L_2 = 32 - 5 + 1 = 28$$

$$B_2 = 32 - 5 + 1 = 28$$

Por lo tanto, la siguiente capa pose un tamaño de 28x28 como se muestra en la ilustración 10. Sin embargo, esta capa posee una profundidad de  $d_2 = 5$ . Esto se debe a que utilizaron cinco filtros diferentes con su propio conjunto de parámetros independientes. Cada uno de estos cinco conjuntos de características obtenidas a partir de la salida de un solo filtro se denominan *mapas de características*.

El incremento de mapas de características resulta en un mayor número de filtros, el cual se puede representar por  $F_q^2 \cdot d_q \cdot d_{q+1}$  para la  $q$ -ésima capa. El número de filtros utilizados en cada capa controla la capacidad del modelo, porque controla directamente el número de parámetros. El aumentar la cantidad de filtros en una capa en particular, aumenta la profundidad en la siguiente capa debido al incremento de los mapas de características. Es posible que diferentes capas tengan cantidades muy distintas de mapas de características, dependiendo de la cantidad de filtros utilizados para la operación de convolución en la capa anterior.

El objetivo es que cada filtro intente identificar un tipo particular de patrón espacial en una pequeña región rectangular de la imagen, por lo que se requieren una gran cantidad de filtros para capturar la amplia variedad de formas posibles que se combinan para crear la imagen final. Generalmente las últimas capas tienen a tener una huella espacial más pequeña, pero mayor profundidad en términos de la cantidad de mapas de características.

Formalmente la operación de la convolución se describe de la siguiente manera el p-esimo filtro en la q-esima capa posee parámetros detonados por un tensor tridimensional  $\mathbf{W}^{(p,q)} = [\mathbf{w}_{ijk}^{(p,q)}]$ . Los índices  $i, j, k$  indican las posiciones a lo largo de la altura, el ancho y la profundidad del filtro. El mapa de características de la q-esima capa se representa por un tensor tridimensional  $\mathbf{H}^{(q)} = [\mathbf{h}_{ijk}^{(q)}]$ . Cuando el valor de q es 1, se considera un caso especial en el que  $\mathbf{H}^{(1)}$  simplemente representa la capa de entrada. Por lo tanto, la operación convolucional desde la q-esima capa hasta la (q+1)-esima capa se define de la siguiente manera.

$$h_{ijk}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} h_{i+r-1, j+s-1, k}^{(q)}$$

$$\forall_i \in \{1 \dots, L_q - F_q + 1\}$$

$$\forall_j \in \{1 \dots, B - F_q + 1\}$$

$$\forall_p \in \{1 \dots, d_{q+1}\}$$

La expresión puede parecer compleja, sin embargo, la operación es bastante simple, como mencionamos anteriormente la operación consta de realizar el producto escalar sobre todo el volumen del filtro el cual se repite en todas las posiciones espaciales válidas (i, j) y los filtros (p). A continuación, se muestra un ejemplo de la convolución entre una entrada 7x7x1 y un filtro de 3x3x1.

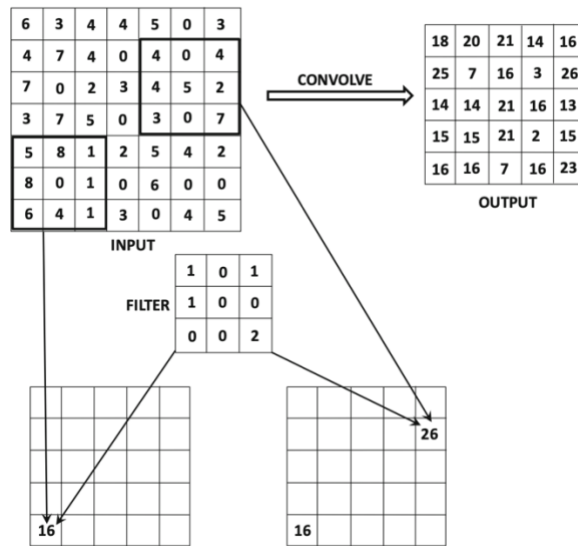


Ilustración 12 Operación Convolución, entrada de 7x7x1 y filtro de 3x3x1

Aggarwal (2018)

Según el estudio de Hubel y Wiesel (1959) en el que estímulos en pequeñas zonas del campo visual activan neuronas particulares. En el caso de las redes neuronales, el campo visual es definido por el filtro el cual es aplicado en todas ubicaciones de la imagen para detectar la presencia de una forma determinada en cada ubicación espacial. Además, los filtros de las primeras capas tienden a detectar formas primitivas mientras que los filtros de las últimas capas crean composiciones más complejas a partir de las formas anteriormente detectadas.

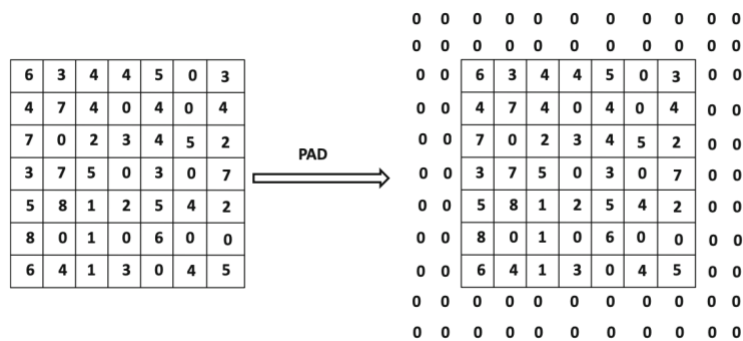
Además, los filtros de las capas anteriores tienden a detectar formas más primitivas, mientras que los filtros de las capas posteriores crean composiciones más complejas de estas formas primitivas. Esto no es particularmente sorprendente porque la mayoría de las redes neuronales profundas son buenas en la ingeniería de características jerárquicas.

Una de las propiedades de la convolución es que muestra equivarianza en la traslación, esto quiere decir que, si se alteran los valores de entrada de los píxeles en una unidad en cualquier dirección y luego se aplica la convolución, los valores de las características correspondientes cambiarán con los valores de entrada. Esto

se debe a los parámetros compartidos del filtro en toda la convolución. La razón por la cual se comparten dichos parámetros en toda la convolución es que la presencia de una forma particular en cualquier parte de la imagen debe procesarse de la misma manera, independientemente de la ubicación espacial.

### Relleno (Padding)

Aggarwal (2018) menciona que la operación convolución reduce el tamaño de la  $(q+1)$  esima capa en comparación con la  $q$ -esima capa. Esta reducción no es deseada debido a que tiende a perder información contenida en los bordes de la imagen o de los mapas de características. Para resolver este problema se utilizan rellenos, esto consiste en agregar  $(F_q - 1)/2$  píxeles alrededor de los bordes de un mapa de características para mantener la huella espacial, cada uno de estos valores de relleno se coloca como “0”. Como resultado la altura y ancho espacial del volumen de entrada aumenta en  $(F_q - 1)$ , lo cual es exactamente la proporción en que se reduce después de realizar la convolución.



*Ilustración 13 Ejemplo de relleno en un mapa de características*

*Fuente: Aggarwall (2018)*

Lo que el relleno hace es permitir la operación de convolución con una parte del filtro “sobresalido” de los bordes de la capa y luego realizar el producto escalar solo sobre la parte de la capa donde se definen los valores. Este tipo de rellenado se le denomina “medio relleno” porque usualmente la mitad del filtro sobresale de todos

los lados de la entrada espacial en el caso de que el filtro se coloque en su posición espacial externa a lo largo de los bordes.

Cuando no se utilizan rellenos la contribución de los píxeles en el borde de la capa son sub-representados en comparación con los píxeles centrales en la siguiente capa oculta. Por lo tanto, el relleno se debe realizar en todas las capas y no solo en la primera donde las ubicaciones espaciales corresponden a los valores de entrada. Aggarwall (2018) propone el siguiente ejemplo: considere una situación en la que la capa tiene un tamaño de  $32 \times 32 \times 3$  y el filtro tiene un tamaño de  $5 \times 5 \times 3$ . Por lo tanto  $(5 - 1)/2 = 2$ , se deben agregar 2 ceros en todos los lados de la imagen. Obteniendo un aumento en la huella espacial a  $36 \times 36$  la cual se reduce nuevamente a  $32 \times 32$  al aplicar la convolución. La Ilustración 13 muestra un ejemplo similar, a una entrada de  $7 \times 7$  se le aplicará un filtro de  $5 \times 5$  por cual la entrada de  $7 \times 7$  crece a  $11 \times 11$  después de aplicar el relleno.

### Zancadas (Strides)

Aggarwall (2018) menciona que existen otras formas en la que la convolución puede reducir la huella espacial de una imagen o mapa de características. El método de rellenos realiza la convolución en cada posible posición espacial del mapa de características. Sin embargo, no es necesario realizar la convolución en cada posición espacial de la capa. Se puede reducir el nivel de granularidad de la convolución utilizando el enfoque de zancadas o pasos. La descripción anterior corresponde al caso de que se utilice una zancada de 1. Si se utiliza una zancada de  $S_q$  en la q-esima capa, la convolución es realizada en las ubicaciones  $1, S_q + 1, 2S_q + 1$  y así sucesivamente a lo largo de ambas dimensiones espaciales de la capa.

El tamaño espacial de la salida al realizar esta convolución posee un alto de  $(L_q - F_q)/S_q + 1$  y un ancho de  $(B_q - F_q)/S_q + 1$ . Como resultado el uso de zancadas resulta en una reducción de cada dimensión espacial de la capa en un

factor aproximado de  $S_q$  y el área por  $S_q^2$ , aunque el factor real puede variar debido a efectos de borde. Comúnmente se utiliza una zancada de 1, aunque ocasionalmente se utiliza una zancada de 2.

Las Zancadas más grandes pueden ser de utilidad en entornos con limitaciones de memoria o para reducir el sobre ajuste si la resolución espacial es innecesariamente alta. Las Zancadas tienen el efecto de aumentar rápidamente el campo receptivo de cada característica en la capa oculta, a la vez que reducen la huella espacial de toda la capa. Un campo receptivo más grande es de utilidad para capturar una característica compleja en una región espacial más grande de la imagen.

### Uso del sesgo

Aggarwall (2018) establece que al igual que todas las redes neuronales es posible agregar sesgos en las operaciones de avance. Cada filtro está asociado a su propio sesgo. Por lo tanto, el p-esimo filtro en la q-esima capa tiene un sesgo  $b^{(p,q)}$ . Al momento de realizar cualquier convolución en el p-esimo filtro en la q-esima capa, el valor de  $b^{(p,q)}$  suma al producto escalar. El uso del sesgo simplemente aumenta el número de parámetros en cada filtro en 1 y por lo tanto no representa una sobrecarga significativa.

El sesgo puede ser tratado como un peso de una conexión cuya entrada siempre se establece como +1. Esta entrada especial se utiliza en todas las convoluciones, independientemente de la ubicación espacial de la convolución. Por lo tanto, el número de entidades de entrada en la q-esima capa es  $1 + L_q \times B_q \times d_q$ . Esta técnica estándar de la ingeniería de características es utilizada para manejar el sesgo en todas las formas de aprendizaje automático

## Capa ReLU

Aggarwall (2018) menciona que la operación de convolución se intercala con las operaciones de agrupación y ReLU. La activación ReLU no es muy distinta a como se aplica en una red neuronal tradicional. Para cada uno de los valores  $L_q \times B_q \times d_q$  de una capa se aplica la función ReLU con el objetivo de crear los valores de umbral  $L_q \times B_q \times d_q$ . Estos valores son enviados a la siguiente capa, por lo tanto, aplicar la función ReLU no cambia las dimensiones de la capa, esto se debe a que es un mapeo uno-a-uno de los valores de activación. En las redes neuronales tradicionales, la función de activación se combina con una transformación lineal y con una matriz de pesos para crear la siguiente capa de activaciones. De manera similar una capa ReLU generalmente sigue una operación convolución.

## Agrupaciones (Pooling)

Según Aggarwall (2018) la operación de agrupamiento opera en regiones pequeñas de dimensiones  $P_q \times P_q$  en cada capa y produce una nueva capa con la misma profundidad. Cuando se retorna el valor máximo para cada región cuadrada de  $P_q \times P_q$  en cada uno de los mapas de activación se denomina *max-pooling*. Si se utiliza una zancada de 1, se produce una nueva capa de dimensiones  $(L_q - P_q + 1) \times (B_q - P_q + 1) \times d_q$ . Sin embargo en las operaciones de agrupación se suelen utilizar valores mayores a 1 para las zancadas, en estos casos la longitud de la nueva capa sería  $(L_q - P_q)/S_q + 1$  y el ancho sería  $(B_q - P_q)/S_q + 1$ . Por lo que se concluye que el agrupamiento reduce drásticamente las dimensiones de cada mapa de activación.

Aggarwall (2018) aclara que, a diferencia de las operaciones de convolución, el agrupamiento es realizado a nivel de cada mapa de activación. La convolución utiliza simultáneamente todos los mapas de características  $d_q$  en combinación con un filtro para producir otro mapa de características. El agrupamiento no altera el número de mapas de características, por lo tanto, no altera la profundidad de la

capa. A continuación, se ilustran ejemplos de agrupamiento utilizando zancadas con valor  $S_q = 1$  y  $S_q = 2$ . En el ejemplo se realizan operaciones de agrupamiento en regiones de  $3 \times 3$ , sin embargo, el tamaño típico de la región de agrupamiento es  $2 \times 2$ . En el ejemplo no se realiza el traslape entre las diferentes regiones a agrupar para  $S_q = 2$ , sin embargo, es recomendable tener cierta superposición entre las unidades espaciales en las que se realiza el agrupamiento porque reduce la probabilidad del sobreajuste.

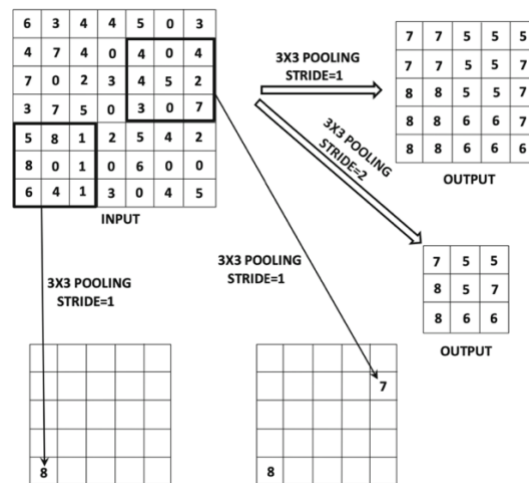


Ilustración 14 Ejemplificación del agrupamiento

Fuente: Aggarwall (2018)

El ejemplo anterior es un caso de *max-pooling* en un mapa de activación de  $7 \times 7$  con zancadas  $S_q = 1$  y  $S_q = 2$ . La zancada  $S_q = 1$  crea un mapa de activación de  $5 \times 5$  que se repiten constantemente debido a la maximización de zonas superpuestas. Una zancada de  $S_q = 2$  crea un mapa de activación con menos superposición.

Según Aggarwall (2018) existen otras formas de agrupamiento, como *average-pooling*, pero que son raramente utilizadas. En general las capas de *max-pooling* son más populares, estas capas suelen ser intercalarse con las capas convolucionales/ReLU, aunque suele suceder con mucha menos frecuencia en arquitecturas profundas debido a que la agrupación reduce drásticamente el tamaño



espacial del mapa de características y solo se requieren unas pocas operaciones de agrupación para reducir el mapa espacial a un tamaño pequeño y constante.

Es habitual utilizar agrupamiento con filtros  $2 \times 2$  y zancada  $S_q = 2$  cuando se desea reducir la huella espacial de los mapas de activación. La agrupación resulta en invariancia a la traslación ya que mover ligeramente la imagen no cambia el mapa de activación de forma significativa. A esta propiedad se le denomina *invariancia a la traslación*. La idea es que las imágenes a menudo tienen ubicaciones relativas muy diferentes de las formas distintivas dentro de ellas, y la invariancia de traslación ayuda a clasificar dichas imágenes de forma similar.

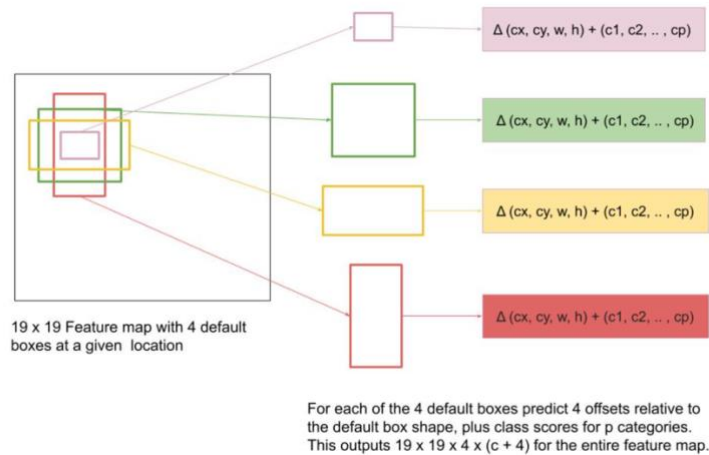
Otro propósito importante de la agrupación es que aumenta el tamaño del campo receptivo mientras reduce la huella espacial de la capa por las zancadas mayores a 1. Se necesitan campos receptivos de mayor tamaño para capturar regiones más grandes de la imagen.

### Single Shot Detector

La idea principal detrás de las arquitecturas SSD es tener una red neuronal convolucional que tome por entrada una imagen y produzca detecciones en diferentes escalas, formas y ubicaciones. Para producir detecciones a diferentes escalas las redes SSD utilizan mapas de características de diferentes capas de redes VGG16 o ResNet. Debido a que cada elemento del mapa de características corresponde a diferentes ubicaciones en la imagen original, los mapas de características al inicio de la red permiten detecciones de objetos pequeños mientras que los del final permiten la detección de objetos más grandes.



predice simultáneamente 4 coordenadas del cuadro delimitador en relación con el cuadro predeterminado  $\Delta(cx, cy, w, h)$ . Esto quiere decir que desplaza el cuadro predeterminado para obtener un mejor ajuste con el cuadro real, además para cada caja predeterminada el filtro genera puntajes de confianza para todas las categorías de objetos ( $c_1, c_2, \dots, c_p$ ). Esto se procesa en simultaneo por lo que de allí viene el nombre “Single Shot”.



*Ilustración 17 Mapa de Características con 4 cajas delimitadoras predeterminadas*

*Fuente: Aidouni (2019)*

La razón por la cual las redes neuronales SSD predicen compensaciones y puntajes de confianza en múltiples mapas de características de diferentes resoluciones es para poder detectar objetos en diferentes escalas. Los mapas de características de menor resolución se encuentran en las últimas capas de la red y permiten detectar objetos más grandes. Esto se debe a que las características semánticas de los objetos a pequeña escala se pierden en las capas superiores.

Aidouni (2019) presenta el ejemplo de la ilustración 15, el mapa de características de 8x8 detecta el gato de la imagen, el cual es el objeto pequeño mientras que mapa de características de menor resolución 4x4 detecta el perro de la imagen, siendo este el objeto grande. De esto se concluye que los mapas de características de las primeras capas de la red, que poseen mayor resolución permiten detectar objetos más pequeños mientras que las últimas capas de la red que poseen menor resolución permiten detectar objetos más grandes.

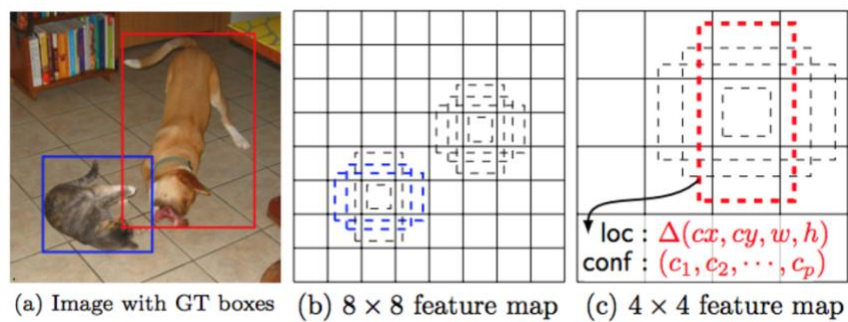
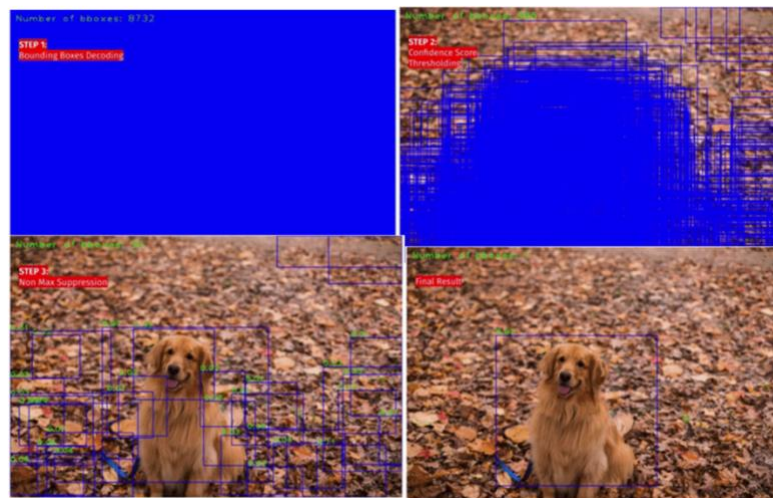


Ilustración 18 Relación de Aspecto

Fuente: Aidouni (2019)

La técnica SSD busca simplificar el proceso de aprendizaje durante el entrenamiento seleccionando los mejores cuadros delimitadores predeterminados que más se superponen con cada cuadro real o cuadro de verdad, en lugar de elegir el mejor. Para cada cuadro de verdad, SSD selecciona el mejor cuadro predeterminado que más se superpone con este, para encontrar dicho cuadro realiza la operación IoU (intersección sobre unión) también conocido como índice Jaccard y conserva los cuadros con  $\text{IoU} > 0.5$

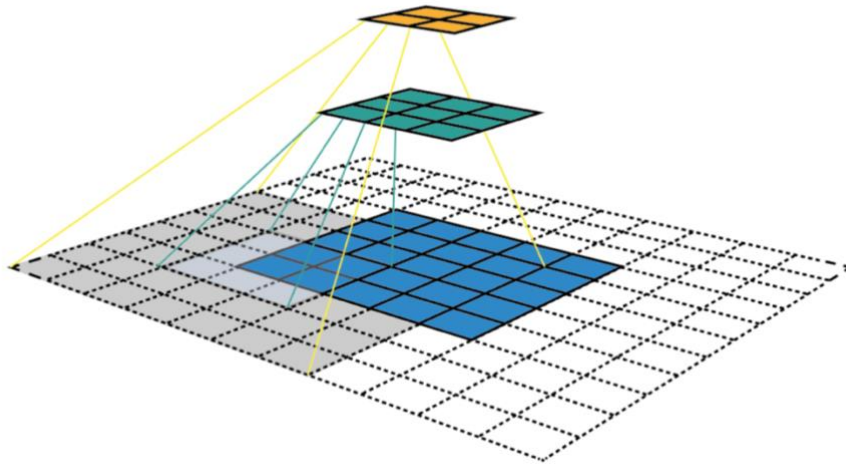
Al momento de las inferencias SSD descarta todas las cajas que presenten resultados de confianza menores a 0.01, luego procede realizar el proceso de supresión no máxima el cual consiste en que, si múltiples cajas detectaron el mismo objeto, solo se conserve el que posee mayor puntaje de confianza.



*Ilustración 19 SSD, entrenamiento e inferencia.*

*Fuente: Lee (2021)*

La característica principal de esta arquitectura de red es el campo receptivo, este se define como la región y espacio de entrada que la función de la red neuronal observa, es decir la región que está siendo afectada. Debido a la operación de convolución, las características de diferentes capas representan diferentes tamaños de regiones en la imagen de entrada. A medida que nos adentramos en las capas de la red, el tamaño representado por una función se hace más grande. A continuación, se presenta un ejemplo el cual posee una capa inicial de 5x5 y al aplicar la convolución se obtiene una capa de 3x3, siendo esta la capa intermedia. Al realizar la convolución en esta capa se obtiene una capa de 2x2 donde cada característica corresponde a una región de 7x7 de la imagen de entrada. Estos mapas refieren al conjunto de características creadas mediante la aplicación del mismo extractor de características en diferentes ubicaciones del mapa de entrada en una fijación de ventana deslizante. Las características del mismo mapa de características tienen el mismo campo receptivo y buscan el mismo patrón, pero en ubicaciones distintas



*Ilustración 20 Representación del campo receptivo de los mapas de características*

*Fuente: ArcGis Developer (s.f)*

## **1.14 Aprendizaje por Transferencia**

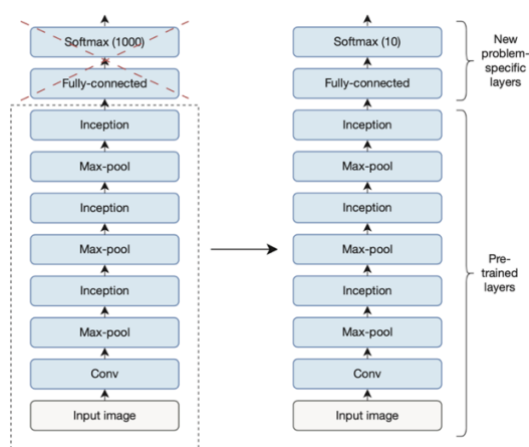
El aprendizaje por transferencia es una técnica que utiliza como base un modelo pre-entrenado para realizar un entrenamiento con un conjunto de datos distinto, pero para el mismo tipo de problema. Esta técnica es particularmente beneficiosa ya que puede reducir el tiempo necesario para realizar el entrenamiento sobre un conjunto de datos.

Ekman (2021) menciona que en muchos casos el problema que se desea atacar está relacionado en cierta forma con el propósito original para el cual la red fue entrenada. Por ejemplo, supongamos el caso en el que se poseen 10 perros y se desea distinguir particularmente a cada uno de ellos y algunos son de la misma raza. Este es un problema de clasificación, sin embargo, utilizar una red entrenada para clasificación de imágenes con 1,000 clases no es útil, ya que se necesita una red que clasifique cuál de los 10 perros es el que se encuentra en la imagen. Es aquí donde se utiliza el aprendizaje por transferencia. Para ello se utiliza un modelo, o partes de este, que fue entrenado para realizar una tarea en particular y luego

utilizarlo para realizar una tarea diferente, aunque relacionada con su propósito original.

La idea es que algunas de las habilidades aprendidas para el propósito original sean transferidas y aplicables en la nueva tarea. En el ejemplo de la clasificación de los perros, se podría utilizar una red convolucional previamente entrenada y sustituir las últimas capas de la red por las nuevas capas para el problema actual y finalizar con la capa de softmax de 10 salidas. Posteriormente se entrena el modelo sobre el conjunto de datos con los 10 perros que la red deberá clasificar.

A continuación, se muestra una representación gráfica del aprendizaje por transferencia. Del lado izquierdo se ilustra la red convolucional inicial pre-entrenada con las 1000 clases y a la derecha la nueva red basada en las capas pre-entrenada, pero con la sustitución de las últimas 2 capas por las capas entrenadas para la clasificación de las 10 clases de perros



*Ilustración 21 Aprendizaje por transferencia, red convolucional inicial y nueva red re-entrenada para nuevas tareas de clasificación*

*Fuente: Ekman (2021)*

Ekman (2021) menciona que cuando se inicia el entrenamiento, las capas pre-entrenadas del modelo pre-entrenado ya se han entrenado durante muchas épocas con un gran conjunto de datos, mientras que los pesos en las capas finales son completamente aleatorios. Al entrenar con un conjunto de datos nuevo, existe el

riesgo de que el algoritmo estropee los pesos cuidadosamente entrenados del modelo original. Por lo tanto, a menudo se congelan estos pesos y se entrenan únicamente las capas recién agregadas. Esto también ayuda a que el proceso sea más rápido debido a que la cantidad de parámetros ajustables es significativamente menor. Posteriormente al entrenamiento con las capas congeladas es recomendable ajustar el modelo descongelando estas capas y entrenarlo unas pocas épocas más con una tasa de aprendizaje menor.

### **1.15 Computación en la Nube (Cloud Computing)**

#### Nube

Según EI (2013) el término de “Nube” hace referencia a un ambiente de IT el cual es diseñado para aprovisionar de manera remota, escalable y medida recursos de TI. El termino se origina de la metáfora del internet, el cual en esencia es un acceso a un set de recursos de TI descentralizados. Antes de que la nube se convirtiera en una formalidad en la industria de la TI, el símbolo de la nube era comúnmente utilizada para representar el internet en gran parte de especificaciones y documentación convencional de arquitecturas web, este mismo símbolo ahora es utilizado para representar ambientes en la nube. Los recursos de TI de la nube son dedicados a suplir el procesamiento de back-end y el acceso por parte del usuario a estos recursos.

Existen varias definiciones de la computación en la nube. Kanr, Doshi y Fagbola (2019) lo definen como un servicio que dinámicamente escalable que provee servicios virtualizados basados en web. El Instituto Nacional de Estándares y Tecnología de los estados unidos lo describe como un modelo que permite el acceso a la red ubicuo, conveniente y bajo demanda a un grupo compartido de recursos informáticos configurables que se pueden aprovisionar y liberar rápidamente con un mínimo esfuerzo de gestión o integración con el proveedor de servicios.



También puede definirse como un sistema de cómputo distribuido conformado por un conjunto de servidores virtualizados dinámicamente provisionados y presentados como uno o más recursos informáticos unificados basados en acuerdos de nivel de servicio (SLA, *Service Level Agreement*).

De las definiciones anteriores se pueden observar que poseen características en común un conjunto compartido de recursos virtualizados, escalabilidad, servicios bajo demanda, entre otros. A continuación, se mencionan algunas características de la computación en la nube:

- Recursos de cómputo a gran escala

Los sistemas de cómputo en la nube son implementados en grandes escalas para reducir costos y aprovechar los beneficios de las economías de escala.

- Agrupaciones de recursos compartidos

El proveedor de recursos agrupa los recursos con el propósito de servir a múltiples clientes utilizando un modelo de *multitenancy*. Para esto utiliza múltiples recursos virtuales y físicos que son asignados y reasignado a demanda del consumidor.

La agrupación de recursos físicos y virtuales provee independencia de la ubicación de los recursos y evita que los consumidores sepan la ubicación exacta de los recursos, sin embargo, se permite que el usuario elija la ubicación de los recursos a consumir a un alto nivel como el país, estado o zona de disponibilidad.

- Autoservicio de recursos a demanda

El consumidor puede aprovisionar capacidades de cómputo de manera unilateral de manera automática sin requerir interacción humana con cada proveedor de servicios.

- Servicios Medidos

Los sistemas de nube controlan y optimizan el uso de recursos permitiendo medir la utilización. En uso de recursos puede ser monitoreado, controlado y reportado con transparencia para el cliente y el proveedor de servicios.

- Programación del aprovisionamiento de recursos

A través de la automatización del software, los recursos pueden ser aprovisionados de forma dinámica basado en los requerimientos de la demanda actual.

- Escalabilidad y Elasticidad

Las arquitecturas de nube permiten provisionar o liberar nodos o servidores de forma automática basado en la demanda sin necesidad de reconfigurar los nuevos componentes. Por lo que el consumidor percibirá que tiene acceso a recursos ilimitados en cualquier momento.

- Confiabilidad

El uso de múltiples sitios redundantes permite la alta disponibilidad y tolerancia a fallas asegurando la disponibilidad del servicio. Además, permite que los clientes adopten arquitecturas de nube para sus cargas de trabajo críticas y recuperación de desastres.

- Virtualización

La virtualización abstrae los recursos físicos de los equipos y los presenta como un conjunto de recursos virtuales. Esto permite correr múltiples cargas en paralelo en los dispositivos físicos.

- Acceso a Trávez de la red

Las capacidades de la nube están disponible a través de la red, estos pueden ser accedidos a través de diversos mecanismos estandarizados para promover el uso de dispositivos heterogéneos.

Algunos de los beneficios de la computación en la nube son los siguientes:

Agilidad empresarial: gracias a la capacidad de la nube de provisionar recursos de forma rápida, los tiempos de despliegue de nuevas aplicaciones y servicios se reducen en gran medida permitiendo a las empresas reducir su tiempo de respuesta antes los cambios del mercado

Reducción de costos de TI: la capacidad de la nube de rentar recursos de TI en un modelo de pago por uso reduce el CAPEX y lo transforma en OPEX ya que solo deberá de pagar por aquello que use.

Continuidad del servicio: a través del uso de las soluciones de continuidad de servicio de la nube, las organizaciones pueden reducir el impacto del tiempo fuera de servicio y recuperarse de caídas que afecten adversariamente las operaciones.

Administración simplificada de la infraestructura: al utilizar servicios en la nube, las tareas de administración se reducen a administrar los recursos que requieren acceso de los servicios de la nube.

## **1.16 Modelos de servicio en la nube**

Los modelos de servicio de la nube especifican los servicios y las capacidades que se brindan a los consumidores. EN el SP 800-145, NIST clasifica las ofertas de servicio de la nube en tres modelos primarios. Infraestructura como servicio, plataforma como servicio y software como servicio.

#### Infraestructura como servicio (IaaS – Infrastructure as a Service):

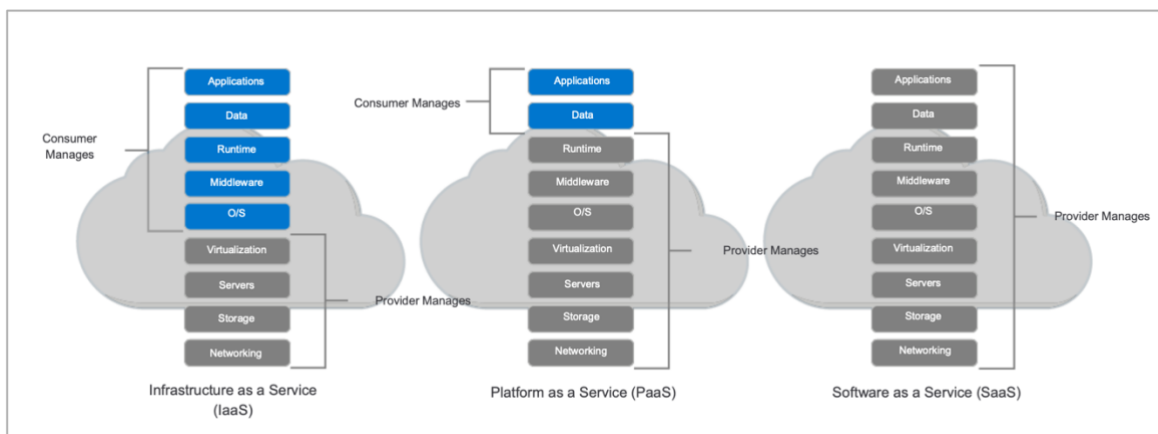
Provee al cliente la capacidad de obtener componentes de infraestructura como servidores, almacenamiento y red, además de permitir que el cliente despliegue y ejecute software, seleccionar su sistema operativo y las aplicaciones. Por lo general permiten acceder a las características de conexión en red, a los equipos virtuales o en software dedicado y al espacio de almacenamiento de datos. La infraestructura como servicio ofrece el mayor nivel de flexibilidad y control de la administración en torno a los recursos de TI y guarda el mayor parecido a los recursos TI tradicionales a los cuales los administradores están acostumbrados. Ejemplos de esto son Amazon EC2 y Google Compute Engine y Azure VNets

#### Plataforma como servicio (PaaS-Platform as a Service):

Permiten al cliente implementar aplicaciones desarrolladas o adquiridas por él, en la infraestructura del proveedor. El consumidor posee control sobre las aplicaciones desplegadas y de las configuraciones del ambiente en el cual fueron desplegadas. Este modelo elimina la necesidad de las compañías de administrar la infraestructura subyacente, como el sistema operativo y el hardware, y permiten centrarse en la implementación de aplicaciones. Ejemplos de esto son Google App Engine, AWS Beanstalk, Microsoft Azure SQL.

#### Software como servicio (SaaS- Software as a Service):

Al cliente utilizar las aplicaciones del proveedor, las cuales se ejecutan en una infraestructura de nube. La pila completa, incluida la aplicación se proporciona como un servicio. Se puede acceder a la aplicación desde varios dispositivos cliente, a través de una interfaz del cliente ligero como un navegador web. Un ejemplo de una aplicación SaaS es un correo electrónico basado en la web, como Gmail que permite enviar y recibir mensajes sin la necesidad de administrar la incorporación de características ni los servidores en los que se ejecuta el programa.



*Ilustración 22 BUSINESS DRIVERS OF DIGITAL TRANSFORMATION*

*Fuente: Dell Technologies (2022)*

A pesar de que los tres modelos generales se pueden definir una gran cantidad de subcategorías de estos modelos. Un ejemplo de esto son los siguientes.

#### Base de Datos como Servicio (DBaaS- Database as a Service):

Según la documentación de Nutanix (2022) el término “Base de Datos como Servicio” se refiere al software y/o los servicios que permiten a los usuarios crear, operar y ampliar las bases de datos sin necesidad de configurar el hardware físico, instalar software o cambiar el rendimiento. El proveedor de servicios se encarga de todas las tareas administrativas y de mantenimiento. Sin embargo, las tareas como asegurar los datos, encriptarlos en reposo y en traslado quedan del lado del usuario por lo que se considera una subcategoría de PaaS.

#### Función como Servicio (FaaS-Función como Servicio):

El término hace referencia a una subcategoría del modelo PaaS, ya que provee una plataforma que permite correr y manejar las funciones de las aplicaciones sin necesidad de construir y mantener la infraestructura relacionada con el desarrollo y lanzamiento de una aplicación. El desarrollo de una aplicación bajo este modelo se denomina *serverless* y suele utilizarse al crear aplicaciones de microservicios.

### Almacenamiento como Servicio (STaaS-Storage as a Service):

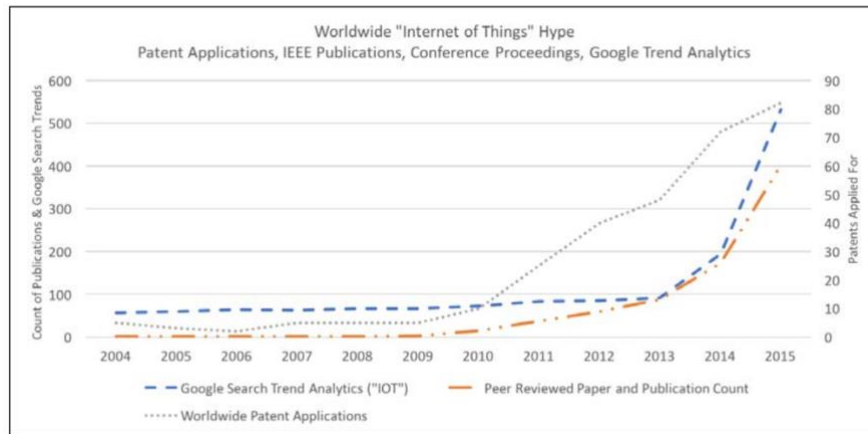
El modelo de almacenamiento como servicio permite al usuario colocar sus datos ya sea de bloque, objetos o archivos. En las soluciones de almacenamiento del proveedor. Estos servicios también proveen un *tiering* de almacenamientos, permitiendo que el cliente archive sus datos menos consultados en un almacenamiento más económico. Esta clase de servicios se cataloga como una subclase de IaaS.

### **1.17 Internet de las Cosas (IoT-Internet Of Things)**

Se puede entender por “Internet de las Cosas” como una red de dispositivos físicos, vehículos y otros ítems embebidos con componentes electrónicos, software, actuadores y conectividad a internet que permite a estos objetos recolectar e intercambiar datos. (Hakim, 2018)

Lea (2021) menciona que el término de IoT se le suele atribuir a Kevin Ashton en 1997 y su trabajo en Procter and Gamble utilizando etiquetas RFID para la gestión de cadenas de suministro. Lo cual lo llevó al MIT en 1999 donde él y un grupo de individuos con ideas similares iniciaron el consorcio de investigación Auto-ID. Desde entonces, IoT despegó de las etiquetas RFID a un ecosistema e industria que prevé tener 1 billón de dispositivos conectados a internet para el 2030.

Hasta 2012 el concepto que se tenía de dispositivos conectados a internet eran teléfonos inteligentes, tabletas, PC y computadoras portátiles. Estos dispositivos en esencia habían funcionado en todos los aspectos como una computadora. Desde el inicio del internet con ARPANET hasta el año 2000 estos eran los únicos dispositivos asociados al internet. El término IoT ha generado internet en las últimas décadas, fue a partir del 2010 donde se generó una tendencia exponencial en el número de patentes relacionadas con IoT y en 2013 donde se observó la tendencia exponencial en relación con la cantidad de búsquedas realizadas en Google y la cantidad de publicaciones revisadas por la IEEE.



*Ilustración 23 Análisis de las búsquedas en Google para el termino IoT, patentes y publicaciones técnicas de IoT*

*Lee (2020)*

### IoT, Machine-to-machine y SCADA

Según Lee (2020) antes de que IoT se popularizara y se convirtiera en la tecnología dominante, M2M era la tecnología utilizada y antes de M2M se utilizaba SCADA (supervisory control and data acquisition). A continuación, se describe cada una de estas.

- M2M:

Es un concepto general que incluye un dispositivo autónomo comunicándose directamente con otro dispositivo autónomo. Entendiendo por *autónomo* a la habilidad de un nodo de instanciar comunicación sin la intervención humana. La forma de comunicación se deja abierta a la aplicación. Es posible que un dispositivo M2M no utilice topologías o servicios inherentes para la comunicación, dejando fuera los dispositivos típicos de internet utilizados para los servicios de nube. Estos sistemas también pueden utilizar comunicación a través de canales no basados en IP como un puerto serial o protocolos personalizados.

- SACDA:

Estos sistemas de control industrial se han utilizado en automatización de fábricas, instalaciones, infraestructura fabricación desde la década de 1960. Por lo general involucra controladores Lógicos programables (PLC) que monitorean o controlan varios sensores y actuadores en la máquina. Con estos sistemas se produjo la industria 2.0. Suelen utilizar protocolos como ModBus, BACNET y Profibus

- IoT:

Los sistemas de IoT pueden incorporar algunos nodos M2M (como las mallas Bluetooth utilizando comunicaciones no basadas en IP). Pero agregan datos en un enrutador o puerta de enlace de borde. El hecho de que posean un método para conectarse a internet es lo que define a IoT.

Al mover datos a internet de fuentes como sensores, procesadores de borde y dispositivos inteligentes, el mundo de los servicios en la nube puede aplicarse a los dispositivos más simples. Antes de que las tecnologías de nube y las comunicaciones móviles se convirtieran en un medio rentable, estos dispositivos no tenían buenos medios para comunicar datos globalmente en segundos, almacenar información a perpetuidad y utilizar los datos para encontrar tendencias y patrones. A medida que avanzan las tecnologías de nube, los sistemas de comunicación inalámbrica se generalizan, los nuevos dispositivos de energía como los de iones de litio se vuelven más rentables y los modelos de aprendizaje automático evolucionan. Esto mejora enormemente la propuesta de valor de IoT.

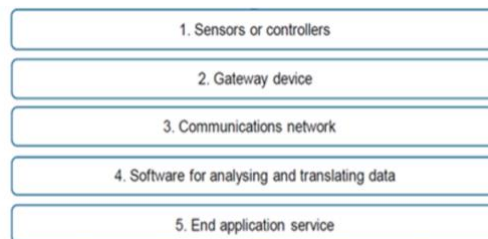
### Modelo de referencia de IoT

El termino IoT es utilizado para describir un amplio y diverso ecosistema, el cual incluye un diverso rango de tipos de conectividad y casos de uso. Por ello se dificulta describir el modelo de IoT como un todo, de manera que suelen utilizarse un modelo de capas para describir el ecosistema de IoT.



Este ecosistema consta de cinco capas esenciales para todos los casos de uso. Estas se presentan a continuación:

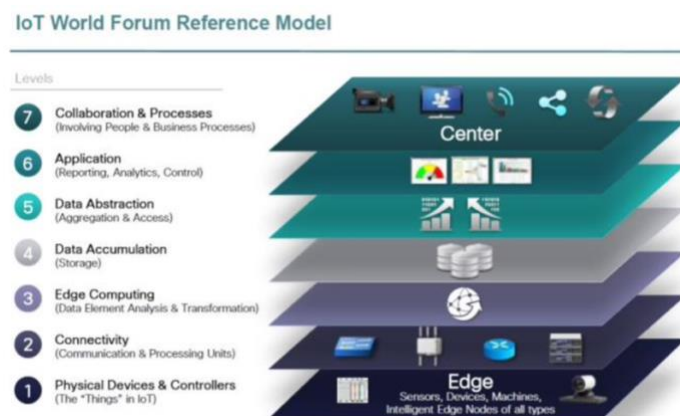
- Sensores o controladores embebidos o conectados a los dispositivos.
- Un gateway para agregar y transmitir datos en la red.
- Una red de comunicación para enviar datos.
- Un Software para analizar y transformar los datos.
- El servicio de aplicación final.



*Ilustración 24 Capas del ecosistema IoT*

*Fuente: Hakim (2018)*

Sin embargo, esta representación de 5 capas es una representación a alto nivel, al entrar en detalle de las capas de una arquitectura IoT multicapa del *World Forum Reference Model* se puede observar un modelo de referencia más robusto.



*Ilustración 25 IoT World Forum Reference Model*

*Fuente Hakim (2018)*

A continuación, se detallan cada una de las 7 capas del modelo:

1. Dispositivos físicos y controladores:

El modelo se refiere a esto como la capa de “cosas” del internet de las cosas. Desde una perspectiva de diseño, las “cosas” son los sensores y dispositivos que son directamente administrados por la arquitectura IoT. Es necesario implementar en esta capa un concepto importante de IoT, Edge Intelligence, para permitir una reacción de baja latencia a los eventos de campo y niveles más altos de autonomía y procesamiento distribuido.

3. Conectividad:

Esta capa se extiende desde el “medio” de un dispositivo Edge Node hasta el transporte a la nube. Se pueden utilizar muchas alternativas para las comunicaciones, esta capa incluye la asignación de datos de campo a las tecnologías, lógicas y físicas utilizadas, así como de la red de retorno a las instalaciones o la nube y la siguiente capa, Edge Computing.

4. Edge Computing:

Esta capa es requerida hasta cierto punto en cualquier sistema IoT, esta capa conecta los datos y los planos de control con las capas superiores de la nube o las capas de software empresarial. En esta capa se implementará la conversión de protocolo, el enrutamiento a funciones de software de capa superior e incluso la lógica de “ruta rápida” para tomar decisiones de baja latencia. En esta capa también se realiza el pre-procesado de datos con el objetivo de reducir las cargas de la nube.

5. Acumulación de Datos:

Debido a la velocidad, el volumen y la variedad de datos que pueden proporcionar los sistemas de IoT, es esencial proporcionar almacenamiento de los datos entrantes para su posterior procesamiento, normalización integración y preparación para las aplicaciones *upstream*.

## 6. Abstracción de Datos:

En la capa de abstracción de datos se le “da un sentido” a los datos, se recopila información “similar” de múltiples sensores, se acelera el tráfico o alarmas de alta prioridad y se organizan los datos entrantes del *Data Lake* en esquemas y flujos apropiados para el procesamiento *upstream*.

## 7. Capa de aplicación:

En esta capa se ejecuta la lógica de aplicación del plano de control y del plano de datos. Monitoreo, optimización de procesos, gestión de alarmas, análisis estadístico, lógica de control, logística, patrones de consumo entre otras aplicaciones de IoT.

## 8. Colaboración y Procesos:

En esta capa, el procesamiento de la aplicación se presenta a los usuarios y los datos procesados en capas inferiores se integran en las aplicaciones comerciales. Se trata de la interacción humana con todas las capas del sistema IoT y donde se entrega el valor económico. El desafío es aprovechar de manera efectiva el valor de IoT y las capas de infraestructura y servicios debajo y aprovechar esto para el crecimiento económico, la optimización comercial y/o el bien social

### **1.18 Contenedores**

Los contenedores son unidades ejecutables de software en los que el código de aplicación se empaqueta junto con sus bibliotecas y dependencias, de forma común para que pueda ejecutarse en cualquier lugar, ya sea un desktop, infraestructura tradicional de TI o en la nube.

Los contenedores utilizan virtualización del sistema operativo en la que las características del sistema operativo, en específico los espacios de nombres y las primitivas de *cgroups* en el caso del *kernel* Linux, se aprovechan al aislar procesos y controlar la cantidad de CPU, memoria y disco a los que dichos procesos tienen acceso.

Los contenedores son pequeños rápidos y portátiles, a diferencia de las máquinas virtuales, los contenedores no necesitan incluir un sistema operativo invitado en todos los casos, sino que simplemente hacen uso de las características y recursos del sistema operativo del host. (IBM Cloud Education)

### Beneficios de los contenedores

Las ventajas principales de los contenedores con comparación con una máquina virtual es que proporcionan un nivel de abstracción que los hace ligeros y portátiles.

- Ligeros:

Los contenedores comparten el *kernel* del sistema operativo de la máquina, lo que elimina la necesidad de una instancia completa del sistema operativo por aplicación y hace que los contenedores sean pequeños y fáciles de usar. Al poseer un tamaño menor al de una máquina virtual, significa que pueden adaptarse de forma rápida y soportar mejor las aplicaciones nativas de la nube que escalan horizontalmente. (IBM Cloud Education)

- Portátiles e independientes de la plataforma:

Los contenedores incorporan todas sus dependencias, lo que significa que el software puede escribirse una vez y luego ejecutarse sin la necesidad de volver a configurarlos a través de portátiles, la nube y entornos de TI on premise. (IBM Cloud Education)

- Soporta el desarrollo y la arquitectura modernos:

Gracias a la combinación de la portabilidad, consistencia de implementación entre plataformas y su tamaño pequeño, los contenedores son ideales para el desarrollo moderno y los estándares de aplicaciones como DevOps, *serverless* y microservicios, que se crean con implementaciones de código regular en pequeños incrementos. (IBM Cloud Education)

- Mejora la utilización:

Al igual que las máquinas virtuales, los contenedores permiten a los desarrolladores y operadores mejorar la utilización de la CPU y la memoria de las máquinas físicas. Uno de los principales beneficios de los contenedores es que, dado que también permiten arquitecturas de microservicio, los componentes de la aplicación se pueden implementar y escalar de forma granular. (IBM Cloud Education)

### **1.19 ONNX (Open Neuronal Network Exchange)**

ONNX es un ecosistema abierto que permite a los desarrolladores de Inteligencia Artificial elegir herramientas adecuadas a medida que evolucionan los proyectos. Proporciona un formato de código abierto para modelos de Inteligencia Artificial, tanto en aprendizaje automático como en aprendizaje profundo. Define un modelo gráfico extensible, así como definiciones de operadores integrados y tipos de datos estándar. (Open Neuronal Network Exchange)

ONNX es ampliamente compatible y se puede encontrar en muchos marcos de herramientas de hardware. Este formato habilita la interoperabilidad entre diferentes *frameworks* y simplifica el camino desde la investigación hasta la producción. Aumentando la velocidad de la innovación en la comunidad de IA. (Open Neuronal Network Exchange)

## II. PLANTEAMIENTO DEL PROBLEMA

La plataforma Jetson de Nvidia, se compone de pequeños módulos de producción y desarrollo de bajo consumo que ofrecen aceleración de alto rendimiento de la pila de software NVIDIA CUDA-X. Esto permite realizar tareas de IA en el borde. La nube pública Amazon Web Services permite integraciones ágiles con estos módulos por medio de su servicio IoT Core, al ser compatible con protocolos de mensajería liviana como MQTT, esto permite conectar dispositivos en localidades con ancho de banda limitados como en redes WAN.

IoT se ha convertido en una de las tecnologías más importantes de la última década, ya que al conectar objetos cotidianos es posible una comunicación fluida entre personas, procesos y cosas. Mediante la informática de bajo costo, el análisis de datos masivos y la nube. Los objetos, entre estos los módulos Jetson, pueden recopilar datos de forma masiva con mínima intervención humana.

Gracias al poder de procesamiento de los módulos Jetson, estos permiten procesar los datos desde su fuente de origen con el fin de reducir la latencia, el ancho de banda y la cantidad de procesos que se ejecutan en la nube. En un sistema de monitoreo de automóviles integrar un módulo Jetson permite realizar las inferencias para la detección del automóvil por medio de un algoritmo de detección de objetos utilizando una red neuronal profunda, procesamiento de la imagen para asignarle un ID en el seguimiento del objeto y abstracción del automóvil del resto de la imagen, todo desde la fuente de los datos. Seguidamente se envía a la nube la imagen del automóvil a través de la red en pequeños fragmentos de datos de forma paralela, permitiendo integrar el borde con los múltiples servicios en la nube.

Entre los servicios es posible utilizar un servicio de reconocimiento de texto en imágenes para abstraer el número de matrícula del automóvil en cuestión y una base de datos administrada que permita almacenar no solo el número de matrícula

sino también la hora en la que el automóvil fue registrado. Un ejemplo donde este sistema sería funcional es el monitoreo de automóviles que ingresan en una institución.

## **2.1 Objetivo General**

- Abstraer la matrícula de un automóvil y asociarla a un ID utilizando visión artificial por computador.

## **2.2 Objetivos Específicos**

- Utilizar una red convolucional profunda para la detección de un automóvil.
- Asociar un identificador único de seguimiento al automóvil detectado por la red neuronal.
- Utilizar Aprendizaje por transferencia para re-entrenar la red neuronal convolucional de base para la identificación de automóviles.
- Obtener el texto de la matrícula del automóvil detectado utilizando el servicio *Rekognition* de la nube pública Amazon Web Services.
- Almacenar en una base de datos administrada en la nube, el identificador único de seguimiento y el texto de la matrícula del automóvil.

## **2.3 Alcances**

### **a. Colaborativos**

- Re-entrenamiento de una red neuronal convolucional SSD para detectar automóviles.

- Se utilizan contenedores de Docker para brindar un microservicio que ejecute el modelo de inferencia, seguimiento del automóvil y la publicación de imágenes en la nube.
- El sistema requiere que la posición de la cámara sea en una ubicación elevada, donde logre captar alrededor de 10 metros de la calle.
- El sistema utiliza el módulo Jetson Nano Developer kit como elemento de procesamiento en el borde.

#### **b. Propios**

- El algoritmo para el reconocimiento de matrículas fue desarrollado para carreteras de una única vía en tramos rectos de la calle.
- El presente proyecto aborda la configuración de políticas y roles necesarios para la interacción entre los servicios DynamoDB, lambda y S3
- Las matrículas analizadas en este proyecto pertenecen a algún país de Asia Oriental.

## **2.4 Límites**

### **a. Colaborativos**

- En este proyecto no se aborda el procedimiento para construir una red neuronal convolucional.

Este proyecto aborda las modificaciones necesarias que se deben realizar a un contenedor de Docker ofrecido por NVIDIA.



- Este sistema no es aplicable para carriles de doble vía ni en tramos curvos de la calle.
- La cámara no puede ser colocada al nivel de la calle ni en ubicaciones donde la vista del carro sea lateral.
- El módulo Jetson Nano Developer Kit se cataloga como un módulo de desarrollo y no un módulo de producción

#### **b. Propios**

- Este proyecto no aborda las configuraciones necesarias para encriptar los datos en reposo en el bucket de S3.
- El proceso de detección del texto en la matricula se ajustó a la nomenclatura de las matrículas presentes en el video.
- En este proyecto no se detalla la configuración de las herramientas de utilidad para la depuración de errores de AWS como CloudWatch.
- El reconocimiento de texto en matrículas se limita a matrículas visibles y en buen estado

### **2.5 Aportes**

El presente diseño busca ilustrar la integración de un sistema de reconocimiento de imágenes de borde y su integración con los servicios en la Nube. Demostrando desde la configuración y el desarrollo del código del dispositivo de borde hasta la configuración de recursos y el desarrollo del código en la nube.

### III. MÉTODO

#### 3.1 Descripción del Proyecto

El presente desarrollo consta de utilizar la plataforma Nvidia Jetson, la cual es una microcomputadora de bajo consumo optimizada en GPU para realizar las inferencias de los automóviles en un video utilizando una red neuronal convolucional SSD. Dicha tarjeta realiza el preprocesado de datos en el borde al abstraer el automóvil del panorama general de la imagen, la cual es enviada a un bucket en la nube por medio de conectividad WiFi. Seguidamente se utiliza un servicio de detección de texto en imágenes instanciado a partir de una función serverless, al obtener el resultado de la detección se escribe el resultado en una base de datos administrada de característica llave valor.

#### 3.2 Instrumentos

##### a. Jetson Nano

El módulo Jetson Nano de NVIDIA es una pequeña pero poderosa microcomputadora de bajo consumo que permite ejecutar múltiples redes neuronales en paralelo para aplicaciones como la clasificación de imágenes y detención de objetos. Entre las principales características de la tarjeta se encuentran: un procesador ARM A57 de cuatro núcleos a 1.43 GHz, 4GB de 64-bit LPDDR4 a 25.6 GB/s, un procesador de gráficos Maxwell de 128 núcleos y un slot para una tarjeta microSD para el almacenamiento. A continuación, se muestra en detalle las especificaciones técnicas y la disposición de entradas y salidas en la tarjeta.

<b>GPU</b>	128-core Maxwell
<b>CPU</b>	Quad-core ARM A57 @ 1.43 GHz
<b>MEMORIA</b>	4 GB 64-bit LPDDR4 25.6 GB/s
<b>ALMACENAMIENTO</b>	microSD
<b>CODIFICACIÓN DE VIDEO</b>	4K @ 30   4x 1080p @ 30   9x 720p @ 30 (H.264/H.265)
<b>DECODIFICACIÓN DE VIDEO</b>	4K @ 60   2x 4K @ 30   8x 1080p @ 30   18x 720p @ 30 (H.264/H.265)
<b>CAMARAS</b>	Hasta 4 cámara   12 carriles MIPI CSI-2   D-PHY 1.1 hasta 18 Gbps
<b>PCIE</b>	1x4 (PCIe Gen2)
<b>CONECTIVIDAD</b>	10/100/1000 Ethernet BASE-T
<b>DISPLAY</b>	2 multi-mode DP 1.2/eDP 1.4/HDMI 2.0 1 x2 DSI (1.5Gbps/lane)
<b>USB</b>	1x USB 3.0 + 3x USB 2.0
<b>POTENCIA</b>	5W   10W
<b>MECÁNICAS</b>	69.6 mm x 45 mm   260-pin SO-DIMM connector
<b>MECÁNICOS</b>	69 mm x 45 mm, 260-pin edge connector

*Tabla 1 Especificaciones Jetson Nano*

*Fuente: NVIDIA (2022)*

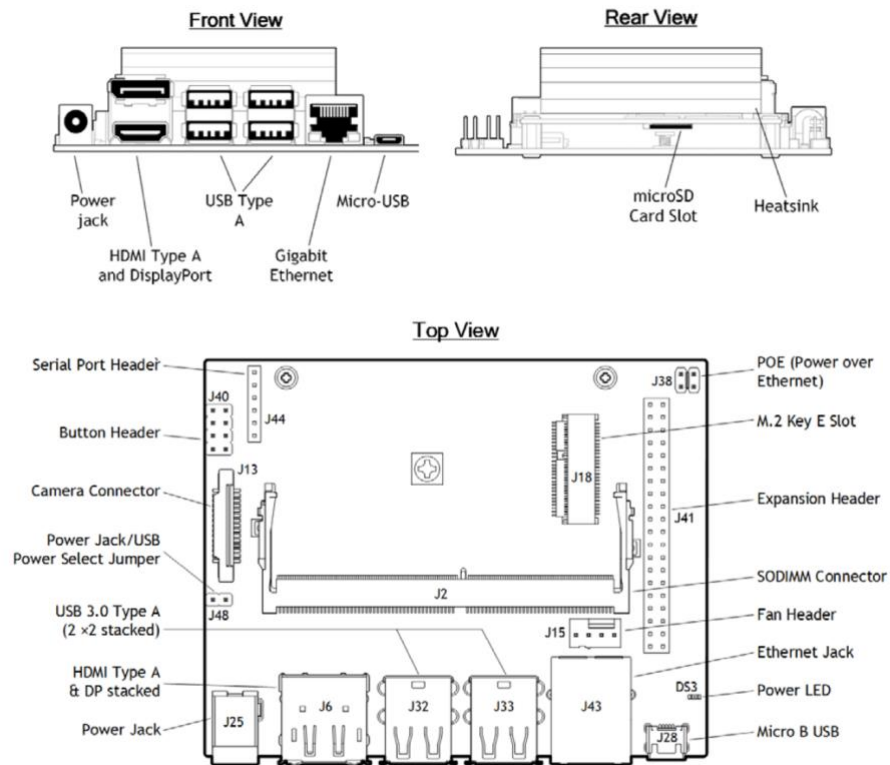


Ilustración 26 Disposición de entradas y salidas Jetson Nano

Fuente: Jetson Nano Developer Kit User Guide (2019)

Para utilizar la tarjeta fue necesario utilizar la última versión de JetPack disponible para el módulo. El JetPack SDK 4.6.1 de NVIDIA proporciona un entorno de desarrollo de IA en el borde, acelerado por hardware. Entre los componentes principales se encuentran los siguientes:

<b>OS</b>	Provee un bootloader, Linux kernel 4.9, firmwares necesarios, drivers de NVIDIA y un sistema de archivos basado en Ubuntu 18.04.
<b>TensorRT 8.2.1</b>	TensorRT es un runtime de inferencia para el aprendizaje profundo de alto rendimiento como clasificación de imágenes, segmentación y redes neuronales de detección de objetos. TensorRT se basa en CUDA y permite optimizar la inferencia para todos los marcos del aprendizaje profundo.
<b>cuDNN 8.2.1</b>	La librería CUDA Deep Neuronal Network provee frameworks de alto rendimiento para deep learning.
<b>CUDA 10.2</b>	El CUDA Toolkit proporciona un entorno de desarrollo completo para desarrolladores de C y C++ que crean aplicaciones aceleradas por GPU. Este kit incluye un compilador para GPU NVIDIA, bibliotecas matemáticas y herramientas para depurar y optimizar el rendimiento de sus aplicaciones.
<b>MULTIMEDIA API</b>  - <b>Scalable Video Coding (SVC) H.264 encoding</b>	<p>EL paquete Jetson Multimedia API provee APIs de bajo nivel para el desarrollo flexible de aplicaciones</p> <p>API de aplicaciones de cámara: libargus ofrece una API síncrona de fotogramas de bajo nivel para aplicaciones de cámara con control de parámetros de cámara por fotograma, compatibilidad con varias cámaras y salidas de</p>

<ul style="list-style-type: none"> <li>- <b>YUV444 8, 10-bit encoding and decoding</b></li> </ul>	<p>flujo EGL. Las cámaras CSI de salida RAW que necesitan ISP se pueden usar con el complemento libarvus o GStreamer utilizando la API del sensor del controlador de medios V4L2</p> <p>Sensor driver API: V4L2 permite la codificación, decodificación, conversión de formato y funcionalidad de escalado en videos.</p>
<p><b>VISION POR COMPUTADORA</b></p> <ul style="list-style-type: none"> <li>- <b>OPENCV 4.1.1</b></li> <li>- <b>VISIONWORKS 1.6</b></li> </ul>	<p>VPI (Vision Programing Interface) es una librería que provee algoritmos de procesamiento de imágenes y visión por computadora en GPU y CPU.</p> <p>OpenCV es la librería de código abierto líder para el procesamiento de imágenes, Machine Learning y visión por computadora.</p> <p>VisionWorks es un paquete de desarrollo de visión por computadora y procesamiento de imágenes.</p>

*Tabla 2 Algunos componentes claves del JetPack 4.6.1*

*Fuente: NVIDIA (2022)*

#### b. Contenedor Jeston Inference

El contenedor Jetson Inference es una imagen de Docker en Linux para arquitecturas ARM la cual posee las librerías necesarias para utilizar la tarjeta, entre ellas la librería jetson-utils y jetson-inference. La versión del contenedor depende de la versión del JetPack instalado en el módulo. En este caso se utilizó la versión 4.6.1 del JetPack y la versión del contenedor L4T R32.7.1. En este contendor se encuentran múltiples ejemplos de las inferencias de las que es capaz la tarjeta,

así como los scripts necesarios para descargar un set de datos y re-entrenar una red neuronal para la detección de objetos. Estos scripts se describen a continuación

- i. open\_images\_downloader.py: utilizado para realizar la descarga de uno o más datasets de Google.
- ii. train.py: Script utilizado para realizar el entrenamiento del modelo haciendo uso de uno o más datasets y una red neuronal de base.
- iii. onnx\_export.py: Script utilizado para convertir el checkpoint del entrenamiento con menor pérdida en la validación y lo convierte en un formato ONNX.
- iv. eval\_ssd.py: Script utilizado para realizar la evaluación del modelo utilizando la precisión media promedio para un valor de intersección sobre unión especificado.
- v. Run.sh: Script utilizado para validar el entorno de ejecución del contenedor y lo ejecuta.
- vi. Build.sh: Script utilizado para la construcción del contenedor.

Este contenedor se utilizó de base para generar uno nuevo pero que incorporará las librerías OpenCV, Matplotlib, sckit-learn, python3-tk y AWSIoTPythonSDK.

c. Boto3

Boto3 es un SDK de AWS para Python, el cual facilita la integración de aplicaciones, bibliotecas o scripts de Python con los servicios de AWS como S3, *Rekognition*, DynamoDB entre otros.

d. Amazon S3

Amazon Simple Storage Service (Amazon S3) es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento. S3 almacena datos como objetos dentro de *buckets* (un bucket es un contenedor de objetos). Para almacenar datos en este servicio se debe crear un *bucket* y especificar el nombre y región de AWS. Posteriormente se cargan los datos como objetos, cada objeto tiene una clave, la cual es el identificador único del objeto dentro del *bucket*. Los *buckets* y objetos que contienen son privados y solo pueden ser accedidos si se concede explícitamente permisos de acceso como políticas de *bucket*, AWS Identity and Access Managment (IAM), listas de control de acceso y puntos de acceso S3.

e. Lambda

Lambda es un servicio informático que permite ejecutar código sin aprovisionar ni administrar servidores. Este servicio ejecuta el código en una infraestructura informática de alta disponibilidad y realiza todas las tareas de administración de los recursos informáticos como el mantenimiento del servidor y sistema operativo, el aprovisionamiento de capacidad, escalado automático, monitoreo y funciones de registro. Al utilizar lambda solo se necesita escribir el código debido a que lambda administra la flota de computación equilibrada de memoria, CPU, red y otros recursos para ejecutar el código



f. Rekognition

Amazon *Rekognition* es un servicio de AWS que facilita el análisis de imagen y video, entre los servicios proporcionados se encuentra la identificación de objetos, personas, texto, escenas y actividades. Incorpora dos conjuntos de API, *Amazon Rekognition Image* para el análisis de imágenes y *Amazon Rekognition Video* para analizar videos. Entre los tipos de análisis se encuentra la detección de texto, permitiendo convertir el texto detectado en un texto legible por una máquina. Para detectar texto en un formato JPEG o PNG se utiliza la operación DetectText, posterior a la detección *Rekognition* crea una representación en palabras y líneas de texto detectadas, muestra la relación entre ellas e indica dónde está el texto en un fotograma.

g. VM en Azure

Para realizar el re-entrenamiento de la red neuronal MobilenetV1 se utilizó una máquina virtual de Azure. La máquina poseía una imagen Linux Ubuntu 20.04 personalizada por NVIDIA, denominada *NVIDIA Image for AI using GPUs*, esta imagen se ejecutó sobre una instancia Standard\_E8bs\_v5 que posee 8 vCPUs y 64 GiB de RAM. Además, se creó un volumen de 512GiB HDD standard para almacenar las imágenes del set de datos. El costo de esta configuración se redondea a \$0.68/hora.

h. Red neuronal pre-entrenada Mobilenet

La red pre-entrenada MobilenetV1 con el conjunto de datos de coco, se utilizó como red base para realizar el re-entrenamiento utilizando aprendizaje por transferencia con el conjunto de datos de automóviles.

i. Pytorch

PyTorch es un *framework* de Deep Learning de código abierto utilizado en aplicaciones como el reconocimiento de imágenes y el

procesamiento de lenguaje. Este escrito en Python y es relativamente fácil de aprender a utilizar. Se distingue por su excelente soporte de GPUs y el uso de diferenciación automática en modo inverso, permitiendo modificar gráficos de cálculo sobre la marcha. PyTorch combina las eficientes y flexibles librerías de *backend* aceleradas por GPU de Torch con una interfaz intuitiva de Python, que se enfoca en la creación rápida de prototipos, código legible y soporte y soporte para la gran variedad de posibles de modelos de aprendizaje profundo.

j. Docker

Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones. Esta plataforma permite separar las aplicaciones de la infraestructura para entregar software de forma rápida ya que permite empaquetar una aplicación en un entorno aislado llamado contenedor. Gracias a la seguridad y el aislamiento se pueden ejecutar muchos contenedores en paralelo en un host.

k. IoT Core

AWS IoT proporciona los servicios en la nube que conectan los dispositivos IoT a otros dispositivos y servicios en la nube AWS. AWS IoT permite seleccionar la tecnología más adecuada y apropiada para la solución en particular. *AWS IoT Core message broker* soporta clientes que utilicen protocolos MQTT para publicar y suscribir mensajes.

l. Tarjeta de red

La conectividad a internet de la Jetson Nano se realizó por medio de WiFi, para esto se utilizó una tarjeta de red Makeroincs 8265AC/8265NGW, la cual se conecta en la ranura M.2 de módulo Jetson.

m. DynamoDB

Amazon DynamoDB es un servicio de base de datos NoSQL de clave-valor administrado que ofrece un alto rendimiento y escalabilidad. DynamoDB permite crear tablas de base de datos capaces de almacenar y recuperar cualquier cantidad de datos, así como atender cualquier nivel de tráfico de solicitudes, permitiendo escalar la capacidad de rendimiento de las tablas para aumentarla o reducirla sin tiempos de inactividad.

n. AWS IAM

AWS Identity and Access Management es un servicio web que permite controlar el acceso a los recursos de AWS de forma segura, permite controlar quién está autenticado y autorizado para utilizar los recursos. Para esto se pueden definir políticas y roles para autorizar el acceso a los recursos.

### 3.3 Procedimiento

La implementación del proyecto puede resumirse en cuatro fases principales:

- Configuración inicial del módulo Jetson
- Re-entrenamiento de la red neuronal convolucional Mobilenet V1
- Desarrollo del código de borde
- Configuración de recursos y políticas de AWS.

A continuación, se muestra un diagrama que detalla las sub-tareas de cada fase.

Configuración inicial del módulo Jetson	Re-entrenamiento de la red neuronal convolucional Mobilenet V1	Desarrollo del Código de borde	Configuración de recursos y políticas en AWS
Montar el JetPack en la Tarjeta	Aprovisionamiento de una máquina virtual en Azure para el entrenamiento	Módulo de identificación y seguimiento del objeto	Registro del módulo Jetson en IoT Core
Conectar el ventilador e instalar las dependencias	Configuración de la VM e instalación de dependencias	Modulo de captura de la imagen del automóvil	Aprovisionamiento y configuración de políticas del Bucket
Configuración del contenedor de docker	Descarga del set de datos OpenImages	Modulo de envío de la imagen a la nube	Aprovisionamiento y configuración de políticas de la Base de Datos en
Conectar la tarjeta de red junto con las antenas	Re-entrenamiento de la red neuronal		Desarrollo de una función en lambda y asignación de políticas.
	Conversión a formato ONNX		

*Ilustración 27 Fases del desarrollo del sistema*

*Fuente: elaboración propia (2022)*

## **Configuración inicial del módulo Jetson**

El primer paso consistió en desmontar el módulo Jetson de la placa base, esto se debe a que debajo de este se encuentra la entrada M.2 (entrada J18 de la ilustración 26). En la entrada mencionada se colocó la tarjeta de red y se conectaron los cables de extensión al conector IPMEX, luego se apretaron las tuercas y arandelas en los conectores SMA y se enroscaron las antenas en dichos conectores, por último, se colocó el módulo Jetson en la placa nuevamente.

La siguiente tarea fue atornillar el ventilador al disipador de la tarjeta y conectarlo al puerto correspondiente (entrada J15 de la ilustración 26). Luego se montó el JetPack en la tarjeta microSD, en este paso se utilizó una computadora con entrada microSD, la imagen del JetPack version 4.6.1 y el software Balena Etcher. Se inicia el software y se selecciona la imagen del JetPack y el almacenamiento de destino. Al finalizar de montar la imagen en la tarjeta microSD se debe de retirar de la computadora e introducir en la ranura microSD de la tarjeta Jetson Nano.

Luego se conectó la tarjeta a corriente y se realizaron las configuraciones del sistema operativo basado en Ubuntu 18.04, como nombre de usuario, contraseña, zona horaria entre otros. Al finalizar esta configuración se procedió a clonar el repositorio Dusty-nv/jetson-inference de GitHub, el enlace se puede encontrar en el anexo 1. Posteriormente se construyó un nuevo contenedor utilizando como base

la configuración del contenedor L4T R32.7.1. del repositorio antes mencionado y agregando las dependencias adicionales necesarias: OpenCV, Matplotlib, Scikit-learn, python3-tk y AWSIoTPythonSDK en el Dockerfile.

Al finalizar de compilar el contenedor se creó un directorio /mydetection3 en el cual se almacena el código a ejecutar en el contenedor. Debido a que los contenedores son ambientes aislados fue necesario montar este directorio como un volumen. Al encontrarnos en este punto ejecutamos el contenedor, el cual cuenta con una interfaz gráfica para descargar los modelos de base como MobilenetSSD v1, VGG16 entre otras. En la siguiente ventana del menú se seleccionó la versión de PyTorch compatible con Python3. Por último, cerramos el contenedor y configuramos un espacio de swap de 4 GB.

### **Re-entrenamiento de la red neuronal convolucional Mobilenet V1**

Esta fase consiste en re-entrenar la red neuronal MobilenetSSD v1, para esto se utilizó una instancia en Azure. La máquina virtual se configuró con una imagen de Ubuntu 20.04 personalizada por NVIDIA, esta puede encontrarse con el nombre *NVIDIA Image for AI using GPUs*, además esta máquina se configuró con el tamaño Standard\_E8bs\_v5, el cual posee 8 vCPUs y 64 GiB de RAM, adicional se aprovisiono un disco de 512 GiB para almacenar el set de datos del entrenamiento.

Al finalizar el aprovisionamiento de la máquina virtual, se procedió a configurar el ambiente en esta máquina, debido a que la arquitectura de la VM era x86 y el contenedor se creó para arquitecturas ARM, fue necesario configurar manualmente el proyecto Jetson-inference. Para esto se inició con conectarse por medio de SSH a la máquina virtual e instalar una interfaz gráfica. Al completar la instalación se procedió a realizar la conexión a la VM por medio del protocolo RDP.

Posteriormente se creó una variable de entorno denominada *tesisenv*, la cual fue activada y se instaló la versión 3.6 de Python. Luego se procedió a clonar el repositorio dusty-nv/jetson-inference de github (el enlace se encuentra en el anexo

1) y a descargar la red de base MobilenetSSD V1 (el enlace se encuentra en el anexo 2).

A continuación, nos posicionamos en el directorio *jetson-inference/python/training/detection/ssd* e instalamos las librerías del archivo *requirements.txt*. Al finalizar la instalación de los requerimientos. Se montó el disco creado anteriormente para almacenar el set de datos en el directorio */home/tesis/jetson-inference/python/training/detection/ssd/data*. Seguidamente se ejecutó el script *open\_images\_downloader.py* del repositorio *jetson-inference* para realizar la descarga del set de datos en formato Open Images desde el enlace del anexo 3. A este script se le indicaron cuatro parámetros, el directorio de almacenamiento (*/home/tesis/jetson-inference/python/training/detection/ssd/data/Car*), el nombre de la clase a descargar (Car), y el número de imágenes a descargar (25,000).

Al finalizar la descarga de las imágenes se inició el re-entrenamiento de la red neuronal. Para esto se utilizó el script *train\_ssd.py* del repositorio *jetson-inference*. A este script se le realizaron algunas modificaciones. La primera fue importar la librería CSV para poder almacenar en documentos .CSV las pérdidas del entrenamiento y de la validación, la segunda modificación fue indicar como predeterminada la red de base MobilenetSSD V1 (*home/tesis/jetson-inference/python/training/detection/ssd/models/mobilenet-v1-ssd-mp-0\_675.pth*).

Al ejecutar este script se le indican algunos parámetros como: el directorio para almacenar los *checkpoints* del entrenamiento (*home/tesis/jetson-inference/python/training/detection/ssd/models/Car*), la cantidad de épocas (60), tamaño de los lotes (103 imágenes por lote) y el directorio del set de datos (*home/tesis/jetson-inference/python/training/detection/ssd/data/Car*). La

Arquitectura de la red re-entrenada se encuentra en el anexo 7

Posterior al entrenamiento se utilizó el script *onnx\_export.py* del repositorio *jetson-inference* para convertir el *checkpoint* del entrenamiento con menor pérdida en la

validación en una red neuronal y ejecutarla en código de Python. A este script se le indicó el directorio de los *checkpoints* (*home/tesis/jetson-inference/python/training/detection/ssd/models/Car*).

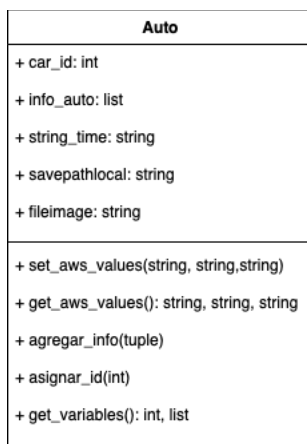
### **Desarrollo del Código de borde**

En esta fase se trabajaron dos archivos, *mydetection3.py* y *car2.py*. Estos 2 códigos interactúan entre en si para realizar la detección del automóvil, seguimiento, recorte del automóvil de la imagen y el envío de la imagen al bucket de AWS.

#### **Archivo car2.py:**

Este archivo posee la clase *Auto* y la clase *Tracker*.

A continuación, se muestra un diagrama de la clase *auto*.



*Ilustración 28 Diagrama de la clase Auto*

*Fuente: elaboración propia (2022)*

La clase *auto* posee variables para almacenar el ID del automóvil (*car\_id*), las coordenadas del recuadro que encierra el objeto detectado (*info\_auto*), el *timestamp* en el que se recorta la imagen (*string\_time*), la ruta de almacenamiento de la imagen (*savepathlocal*), el nombre de la imagen (*fileimage*) y el tiempo en el que se detectan los centroides. Además de las funciones necesarias para acceder y modificar estas

variables (set\_aws\_values, get\_aws\_values, agregar\_info, asignar\_id, get\_variables, set\_speed, get\_speed).

A continuación, se muestra un diagrama de la clase Tracker

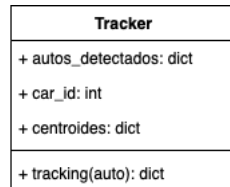


Ilustración 29 Diagrama de la clase Tracker

Fuente: elaboración propia (2022)

La clase Tracker posee un diccionario que almacena los autos detectados (autos\_detectados), el identificador del automóvil (car\_id) y un diccionario para almacenar los centroides de los automóviles (centroides). Esta clase posee la función *tracking*, esta es la función que realiza el seguimiento de los automóviles y se puede describir mediante el siguiente diagrama de flujo.

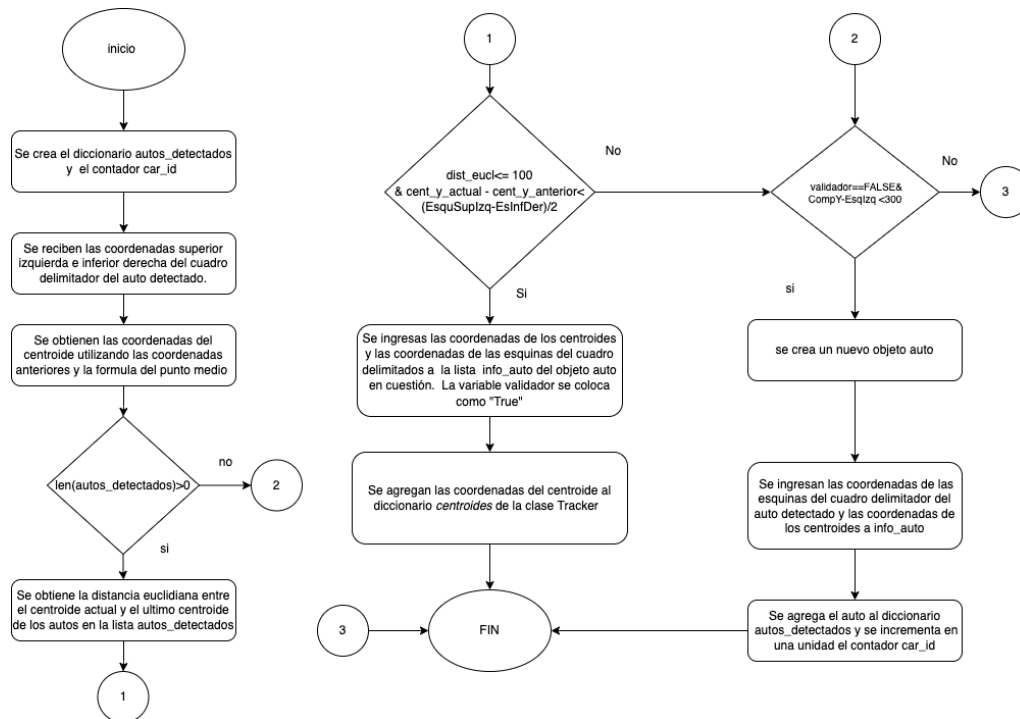


Ilustración 30 Diagrama de la función tracking

Fuente: elaboración propia (2022)



La función `tracking` recibe las coordenadas de la esquina superior izquierda e inferior derecha que proporciona la red de detección de objetos para obtener las coordenadas del centroide del automóvil, por medio de la ecuación del punto medio. Al recibir las coordenadas se realizan dos validaciones para definir si el auto es un auto no identificado o si es un auto identificado que se ha desplazado a otra región de la imagen entre el frame actual y el frame anterior.

Inicialmente se obtiene la distancia euclidiana entre el centroide del objeto detectado y el ultimo centroide registrado de los automóviles identificados en el diccionario (`autos_detectados`) y la distancia euclidiana entre las 2 coordenadas de las esquinas anteriormente mencionadas. Posteriormente evalúa si a la distancia entre centroides es menor a 100 pixeles y se la diferencia entre las coordenadas en `x` y de los centroides es menor a la distancia entre las coordenadas entre las esquinas del recuadro que encierra el objeto. Al cumplirse la condición se establece que el automóvil ya fue detectado y las coordenadas de sus centroides se asignan al diccionario de centroides de la clase *Tracker*. En el caso que los centroides de las coordenadas recibidas no cumplan la condición establecida y el valor en `x` y de la esquina inferior derecha sea menor a 300 se crea un nuevo objeto auto, al cual se le asignan estos centroides y se agrega al diccionario de autos detectados (`autos_detectados`) y el contador de identificadores asignados `car_id` de la clase *Tracker* se incrementa en una unidad.

#### Archivo `my-detection3.py`:

El archivo `my-detection3.py` es el archivo principal, éste utiliza las clases del archivo `car2.py`. A continuación, se observa un diagrama de flujo del archivo.

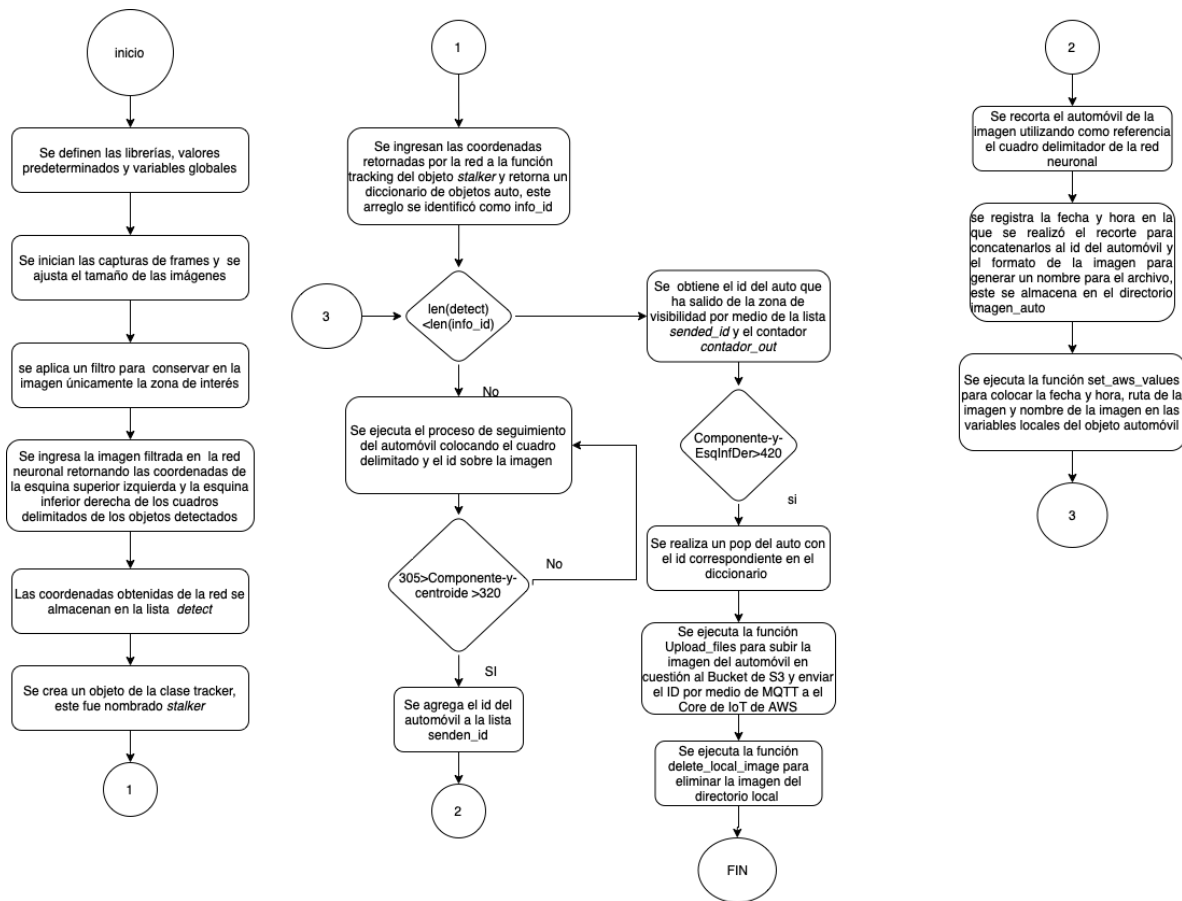


Ilustración 31 Diagrama de la función *tracking*

Fuente: elaboración propia (2022)

En las primeras líneas se definen las librerías necesarias, los valores predeterminados como la fuente de video y la red neuronal para las inferencias. Además, se definen las variables globales, en este caso se utilizó un arreglo para almacenar el id de las imágenes enviadas (*senden\_id*) y el contador de autos eliminados del diccionario de automóviles (*contador\_out*), este proceso de detalla más adelante. Posteriormente de detallan las funciones principales (*delete\_local\_image*, *upload\_files*), se instancia un objeto *stalker* que posee los atributos de la clase *Traker* del archivo *car.py* y se colocan las rutas de los certificados del dispositivo registrado en AWS, el *endpoint*, id del cliente, las credenciales de acceso del usuario cliente de AWS y el tópico de MQTT.

El proceso del análisis del video inicia con leer los frames, ajustarlos en un tamaño menor para facilitar el análisis y establecer una zona. Esta zona posee las coordenadas de la carretera, y se utiliza un filtro `bitwise_and` para eliminar de la imagen los pixeles ajenos a la carretera. Posteriormente la imagen filtrada se ingresa a la red neuronal para obtener las inferencias de los automóviles y se retornan las coordenadas de la esquina superior izquierda y la esquina inferior derecha del cuadro delimitador, estas coordenadas son almacenadas en la lista *detect*. Estas coordenadas se ingresan a la función *tracking* del objeto *stalker*. Esta función retorna un diccionario de objetos con los automóviles detectados *info\_id*

Posteriormente se ejecuta un proceso de validación entre la longitud de la lista *detect* y la longitud del diccionario *info\_id*. Si no existe diferencia entre estas quiere decir que la cantidad de objetos detectados por la red coincide con la cantidad de objetos retornados por la función de seguimiento el código continuo su curso. Ejecutando el proceso del seguimiento en el cual se coloca el id en la posición del recuadro delimitado de la última posición registrada de los objetos en el diccionario *info\_id*, que en este caso son los autos en la zona delimitada. Dentro de este proceso existe una condición en la que si el componente en y del centroide se encuentra dentro de la posición 305 y 320 se agrega el Id de este objeto a la lista *senden\_id*, se recorta la imagen utilizando las coordenadas del cuadro delimitador de la detección y se registra la fecha y hora en la que se realiza el recorte. También se concatenan el id del objeto junto con la fecha y hora del recorte y el formato de la imagen, para guardarlo en el directorio *imagen\_auto* por último se utiliza la función *set\_aws\_values* para colocar la fecha y hora, la ruta de la imagen y el nombre de la imagen en las variables locales del objeto.

En caso de obtener una diferencia entre las longitudes de la lista *detect* y la longitud del diccionario *info\_id*, indica que alguno de los objetos ha salido de la zona de visibilidad. Para ello utiliza el *contador\_out* y la lista *sended\_id*. En esta lista se almacenan los ids de los automóviles cuyas imágenes han sido recortadas. El *contador\_out* nos sirve para posicionarnos en la posición de la lista que contiene el

id del automóvil que ha salido de la zona establecida y por lo tanto debe salir del diccionario *info\_id*. Posteriormente se obtienen las coordenadas del cuadro delimitador de la última posición del arreglo *info\_auto* por medio de la función *get\_variables* y se accede a la última registrada para evaluar que el valor de *y2* sea mayor a 420 se, si la condición se cumple se retira el objeto del diccionario y se realiza una función *upload\_files* en paralelo la cual sube la imagen recortada a el *bucket* de AWS y la función *delete\_local\_image* para eliminar la imagen del automóvil del directorio local

La función *upload\_files* recibe la ruta del directorio de la imagen, el nombre del *bucket* y el nombre del archivo a enviar. Esta función utiliza el cliente del servicio AWS S3 por medio de la librería boto3. La función *delete\_local\_image* utiliza las librerías del sistema operativo para remover las imágenes con comandos de la terminal.

### **Configuración de recursos y políticas en AWS**

La fase de configuración de recursos inició con la creación de un objeto en el IoT Core de AWS. Para ello se creó un grupo de objetos/cosas denominado *JetsonEdgeCards-S1* se seleccionó la opción de autogenerar un nuevo certificado para los objetos y se creó una política la cual permite a el objeto que posea estos certificados realizar todas las acciones relacionadas a IoT. Esta política se vincula a los certificados autogenerados, estos fueron asociados al objeto *JetsonNano-S1-1*, el cual se miembro del grupo *JetsonEdgeCards-S1*. Posteriormente se deben descargar los tres certificados autogenerados, estos corresponden al certificado del dispositivo, la llave publica, la llave privada y el certificado raíz de la entidad certificadora de Amazon con llave de longitud de 2048 bits. También se procedió a anotar el *endpoint* de los dispositivos.

Los certificados y las llaves se almacenaron en la tarjeta Jetson y deben indicar las rutas dentro del código de borde para identificarse en AWS al momento de enviar datos por MQTT, sin embargo, también se indicó el *endpoint* al que se dirigieron los datos y el tópico al cual publicar (*\$aws/things/JetsonNano-S1-1/shadow/name/detectCar*).

Para almacenar las imágenes de los automóviles se creó un grupo de usuarios denominado *JetsonNano-car-traffic-accounts* a este grupo se le asoció una política que otorga permisos para colocar objetos en un *bucket* de S3. En este grupo se creó el usuario *JetsonNano-car-traffic-user-1*, y se descargaron sus llaves de acceso. Estas llaves se indican en el código de borde para autenticar a la tarjeta Jetson y permitir la subida de imágenes al *bucket*, el cual fue creado con el nombre *carimages-traffic-jetson-nano-4gb* y se asoció a una política que permite que el usuario *JetsonNano-car-traffic-user-1* enlistar y colocar objetos almacenados. En esta política también se autorizó a la función lambda obtener objetos del *bucket*.

El registro de automóviles monitoreados se creó una tabla en una base de datos clave-valor en DynamoDB la cual se denominó *Detected-Cars-DB*. El ID se utilizó como llave de partición y el *timestamp* como llave de ordenamiento. También se agregaron los campos matrícula y nombre de la imagen.

Para escribir los valores en la tabla se creó una función lambda denominada *MQTT-S3getCar-Recognition-DynamoDB*, esta función se ejecuta utilizando Python3.9 en una arquitectura x86\_64. Se realizó la configuración de un desencadenador asociado al servicio IoT Core, en el cual cada vez que se publique en el tema *\$aws/things/JetsonNano-S1-1/shadow/name/detectCar* se ejecute la función. Posteriormente se editaron las políticas asociadas al rol de ejecución de la función para leer y escribir en la tabla *Detected-Cars-DB*, enlistar y obtener objetos del *bucket* de S3 y se autorizó realizar cualquier acción en el servicio *Rekognition*.

El código en lambda lee el valor del ID recibido por MQTT, este ID es el mismo que identifica la imagen en el *bucket*. Por lo tanto, se ejecuta la función *call\_rekognition*, esta función recibe el nombre de la imagen y el nombre del bucket para utilizar el cliente de *rekognition* de la librería boto3 y realizar la detección de texto en la imagen en S3. La detección de texto se realiza con la función *detect\_text* del cliente de Rekognition, este devuelve un JSON con los textos detectados, para ubicar un texto de la matrícula dentro del JSON de respuesta. Se utilizó una condicional que almacena en una variable el texto de tipo *línea* con una longitud mayor a 6. Debido a que en ocasiones *Rekognition* coloca un espacio en lugar del guion entre la primera letra A y el resto de la numeración se realizó un ajuste. Este ajuste sustituye los espacios por guiones. Luego la función *call\_rekognition* retorna el texto de la posición previa al guion de la matrícula, hasta la posición 6 índices después del guion para capturar el texto completo de la matrícula. Esta salida se ingresa posteriormente a la función *normalize* esta función elimina las tildes que en ocasiones son colocadas por la detección de texto de *Rekognition*.

Por último, se ejecuta la función *Dynamo\_write*, esta toma como entrada el ID y la matrícula. La función utiliza el cliente de DynamoDB de la librería boto3 para escribir en la tabla de la base de datos el ID, el *timestamp*, la matrícula y el nombre de la imagen. El *timestamp* se obtiene ejecutando la función *get\_timestamp*, esta función toma de entrada el id y abstrae el *timestamp* que concatenamos al momento de generar el ID en el borde y lo retorna. El nombre de la imagen se obtiene concatenando al ID el formato .JPG. A continuación, se muestra un diagrama de flujo que la función *MQTT-S3getCar-Recognition-DynamoDB*.

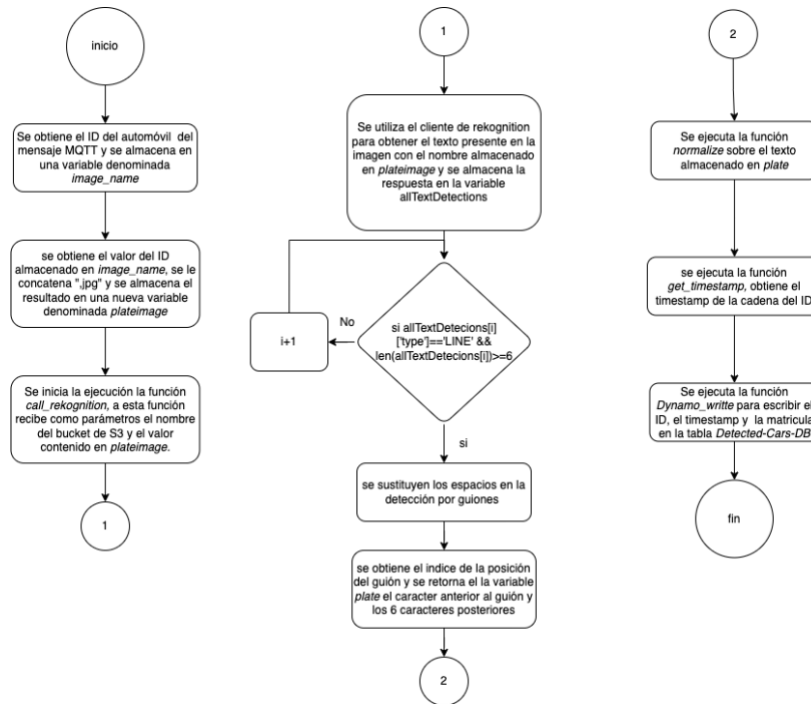


Ilustración 32 Diagrama de flujo de la función MQTT-S3getCar-Recognition-DynamoDB,

Fuente: elaboración propia (2022)

### 3.4 Análisis y Diseño

A continuación, se presenta un diagrama de la arquitectura utilizada para el desarrollo del sistema.

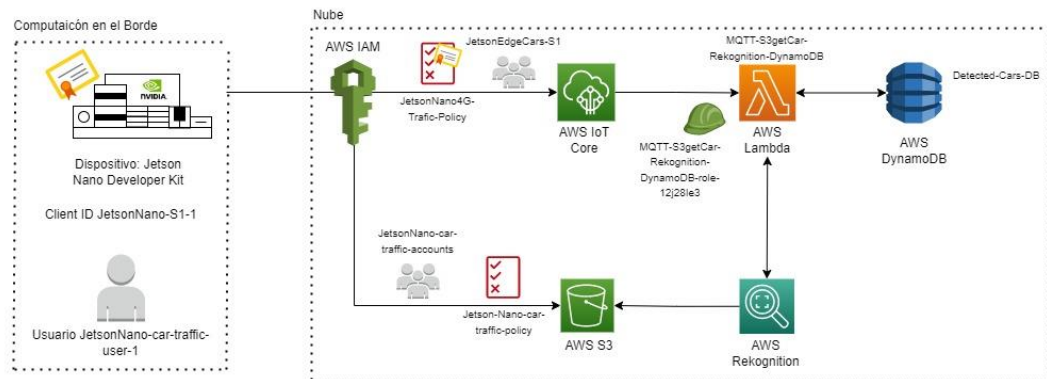


Ilustración 33 Arquitectura del Sistema

Fuente: elaboración propia (2022)

La arquitectura utilizada consta de dos módulos principales, computación en el borde y computación en la nube.

### **Computación de Borde**

El módulo de computación de borde se conforma en su mayor parte de la tarjeta Jetson Nano Developer Kit, como se mencionó antes esta tarjeta fue seleccionada por su poder computacional en GPU y su bajo consumo, además de que al utilizar un SO basado en Linux provee una integración sencilla con Python y los servicios de la Nube.

En la tarjeta se almacenan los certificados asociados a la *política JetsonNano4G-Traffic-Policy* que identifica al objeto en el *IoT Core* de AWS con el nombre *Jetson-NanoS1-1*. En el código de borde también se indican las rutas a los certificados y el ID del Cliente, dicho ID es el nombre del objeto (*Jetson-NanoS1-1*), también se deben colocar las llaves de acceso del usuario *JetsonNano-car-traffic-user-1*. Este usuario es el que identifica la tarjeta al momento de realizar acciones sobre el *bucket*. Por último, es importante recordar que esta tarjeta es la responsable de las inferencias por lo tanto también debe almacenar la red neuronal convolucional re-entrenada.

En la implementación el sistema de borde deberá de colocarse en un poste junto con una cámara para realizar las inferencias. Este se alimentará con un panel solar de 45w, seguido por un regulador a 9V para controlar el voltaje que recibe el UPS y sus baterías. El módulo del UPS junto con las baterías se conecta directamente a la tarjeta Jetson Nano Developer Kit. A continuación, se muestra el diagrama.





*Ilustración 34 Implementación del Sistema de borde*

*Fuente: elaboración propia (2022)*

### **Computación de Nube**

Como se puede observar en el diagrama, el primer contacto en la nube es el servicio *AWS Identity and Access Manager*, esto no significa que todo el tráfico sea filtrado por este servicio. Se colocó como primer punto de contacto debido a que todos los usuarios, políticas y roles se definen por medio de este.

En el diagrama se observa que el módulo de borde interactúa directamente con 2 servicios, el IoT Core de AWS y el *Bucket* de AWS S3. Al IoT Core se envían el ID del automóvil su velocidad por medio del protocolo de mensajería liviana MQTT, al momento que estos salen de la zona de interés, mientras que la imagen del automóvil es recorta cuando el automóvil se encuentra en una región específica de la zona de interés y es enviada en el instante que el auto sale de dicha zona.

La función lambda es nuestro punto pivote entre los servicios del IoT Core, el *bucket* de S3, *Rekognition* y *DynamoDB*. Esta función denominada *MQTT-S3getCar-Rekognition-DynamoDB* utiliza como desencadenador el servicio IoT Core, el cual cada vez que se publique un evento bajo el tópico *\$aws/things/JetsonNano-S1-1/shadow/name/detectCar*, ejecutará la función. El código de borde publica y envía el ID, la velocidad y la imagen del automóvil a los servicios antes mencionados. Al ejecutarse la función esta utiliza el cliente de *Rekognition* para obtener la matrícula de la imagen correspondiente al automóvil con el ID en cuestión que se encuentra almacenada en el *bucket*. Por último, la función utiliza el cliente de *DynamoDB* para escribir en la tabla de la base de datos clave-valor el registro del ID, *timestamp*, matrícula, nombre de la imagen del *bucket* y velocidad.

Este proyecto se realizó de forma colaborativa por lo que el procedimiento detallado del registro de la tarjeta en el *IoT Core*, el algoritmo de detección de velocidad, alimentación y transmisión no se abordan a detalle en este documento. A continuación, se resaltaron en el diagrama las secciones abordadas en este documento.

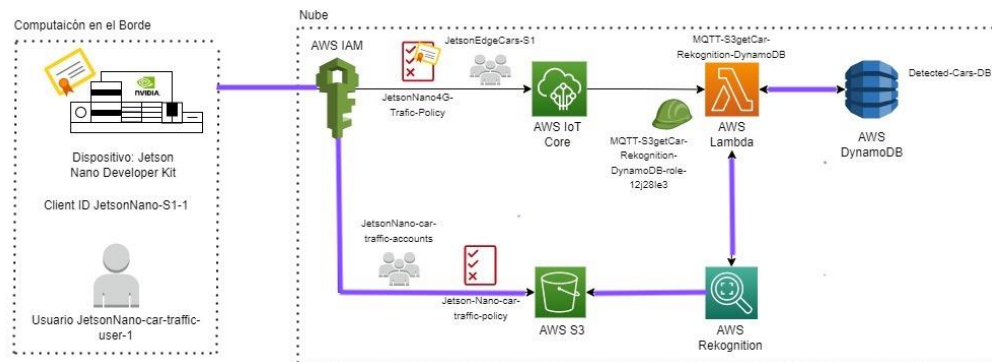


Ilustración 35 Participación en el Desarrollo

Fuente: elaboración propia (2022)

#### IV. PRESENTACIÓN Y ANÁLISIS DE RESULTADOS

##### Rendimiento del modelo re-entrenado

Durante el módulo de entrenamiento del modelo se realizaron 2 entrenamientos, en ambos se utilizó un set de datos de 25,000 imágenes de automóviles y lotes de 103. La diferencia fue la cantidad de épocas realizadas. En el primer modelo se realizaron 30 épocas mientras que en el segundo se realizaron 60. El parámetro utilizado para medir el rendimiento fue el promedio de la precisión media promedio de la clase “car” para diferentes límites inferiores de Intersección sobre unión. Este parámetro toma en consideración la precisión y el *recall*.

La precisión se calcula mediante la siguiente ecuación:

$$\text{Precisión} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos positivos}}$$

Los *Verdaderos positivos* son las detecciones correctas de un automóvil y los *falsos positivos* son detecciones de un automóvil en ausencia de este. Dicho parámetro indica la proporción de detecciones correctas en relación con la cantidad de detecciones.

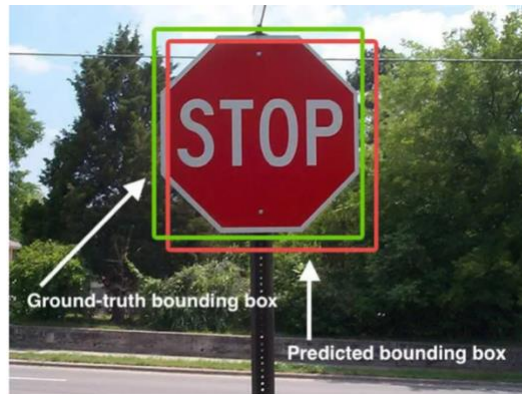
El *recall* se calcula mediante la siguiente formula:

$$\text{recall} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos negativos}}$$

Los Falsos negativos ocurren cuando el modelo debió detectar un automóvil, pero no lo detectó. Este parámetro indica la fracción de detecciones correctas sobre el total de detecciones objetivo.

Posteriormente el parámetro precisión media promedio traza la curva Precisión vs *recall* y calcula el área debajo de esta para determinar el rendimiento de la clase en cuestión. Posteriormente se promedia el resultado para todas las clases, sin embargo, en este caso únicamente se utilizó una clase.

Debido a que el modelo utilizado es un Single Shot Detector se utiliza el umbral de la Intersección sobre Unión en el cálculo de la precisión media promedio. Esta métrica utiliza las cajas delimitadoras de verdad básicas y las cajas delimitadoras predictoras. En la siguiente imagen se ilustran ambas cajas delimitadoras en una imagen de ejemplo:



*Ilustración 36 El cuadrado verde representa la caja delimitadora de verdad básica mientras que el rectángulo rojo representa la caja delimitadora predictora.*

*Fuente: Rosebrock (2016)*

El cálculo de la Intersección sobre unión utiliza la siguiente formula:

$$IoU = \frac{\text{Área de Traslape}}{\text{Área de Unión}}$$

En el numerador se calcula el área de traslape entre el cuadro delimitador de verdad básica y el cuadro delimitador predictor mientras que en el denominador se calcula el área de unión de ambos cuadros. Al utilizar este valor como umbral quiere decir que se consideran validas todas las predicciones que posean un IoU mayor a el valor de umbral seleccionado.

A continuación, se muestran las tablas de resultados de la precisión media promedio para distintos límites de intersección sobre unión para ambos modelos.

Umbral IoU	Precisión Media Promedio
0.5	69.16%
0.6	64.20%
0.7	57.00%
0.8	43.39%
0.9	11.75%
Promedio	46.95%

*Tabla 3 Resultados del rendimiento del modelo con 30 épocas de entrenamiento*

*Fuente: elaboración propia (2022)*

Umbral IoU	Precisión Media Promedio
0.5	73.17%
0.6	68.91%
0.7	62.81%
0.8	51.9%
0.9	24.88%
Promedio	56.33%

*Tabla 4 Resultados del rendimiento del modelo con 60 épocas de entrenamiento*

*Fuente: elaboración propia (2022)*

En el segundo modelo se observó mejoras en el desempeño teniendo un promedio de 56.33% siendo superior en un 9.38% con respecto al primer modelo. Sin embargo, el modelo aún puede ser mejorado a través del entrenamiento. Esto se evidencia en las gráficas de pérdida del entrenamiento y la validación.

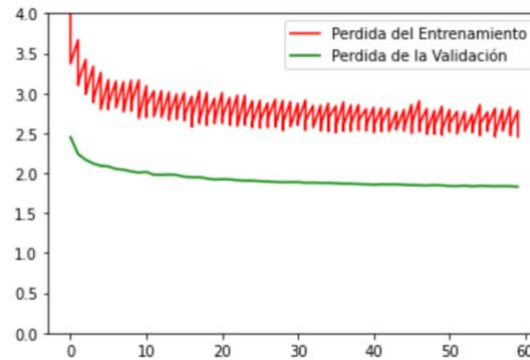


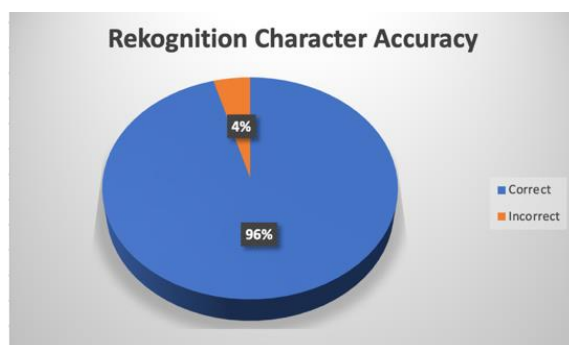
Ilustración 37 Comparación de funciones de perdida

Fuente: elaboración propia (2022)

### Rendimiento de la detección en Rekognition

Para evaluar el desempeño del modelo utilizado por *Rekognition* para la detección de texto en matrículas se utilizaron 113 imágenes de automóviles con matrículas de diversos países. Estas imágenes se ingresaron al servicio de *Rekognition* para la detección de texto se evaluó la precisión en la detección de caracteres y la precisión en detectar el texto completo de la matrícula.

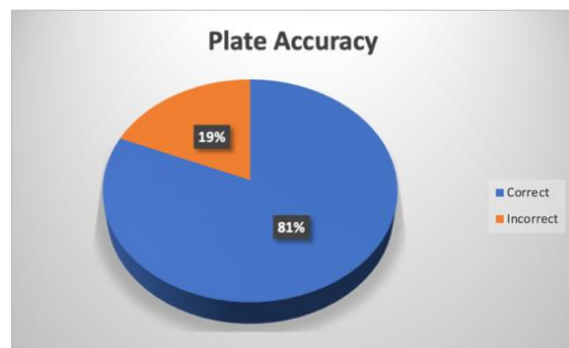
Al sumar la cantidad de caracteres presentes en las 113 matrículas de los automóviles se obtuvo la cantidad de 842 caracteres detectando correctamente 805 de estos. Obteniendo un 96% de detecciones acertadas y un 4% de caracteres fallidos. Durante el proceso se observó que la mayor cantidad de caracteres fallidos se obtenía en las imágenes con menor resolución. Por lo que es se determinó que analizar imágenes mayor resolución minimiza el problema y en caso de no poder incrementar la resolución se puede minimizar capturando la imagen lo más cerca posible de la cámara.



*Ilustración 38 Precisión en la detección de Caracteres de AWS Rekognition*

*Fuente: elaboración propia (2022)*

Para medir la precisión en la detección de la numeración completa de las matrículas se tomaron como detecciones correctas todas las que el 100% de sus caracteres fueron detectados sin errores. Dando por resultado la detección correcta de 92 matrículas de las 113 utilizadas. Obteniendo una precisión del 81% sin realizar algún proceso de corrección. Dependiendo del país donde se implemente el sistema pueden aplicarse funciones de corrección. Por ejemplo, en Guatemala la nomenclatura de las matrículas inicia con una letra *P* seguida de 3 números y 3 letras. Conociendo la nomenclatura se puede implementar que asegure que los 3 caracteres siguientes a la *P* sean números, de manera que si detecta una */* o una *.* esta sea sustituida por el número *1*. De la misma forma se puede implementar una condición para evaluar los últimos 3 caracteres de la matrícula para asegurar que sean letras, de forma que, si detecta el número *0*, sea sustituido por la letra *O*.



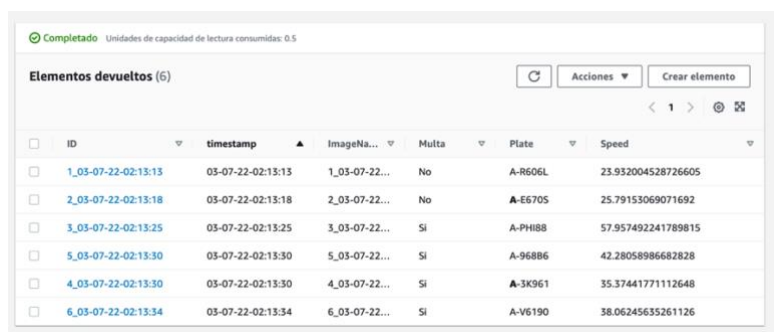
*Ilustración 39 Precisión en la detección de Matrículas de AWS Rekognition*

*Fuente: elaboración propia (2022)*

¿Por qué existe una diferencia de 15% entre la precisión de la detección de caracteres y la precisión de la detección de matrículas? Esto se debe a que una sola matrícula posee entre 6 y 10 caracteres, nuestra muestra de 113 matrículas equivale a una muestra de 842 caracteres. Un carácter detectado de forma incorrecta es más representativo en la muestra de matrículas que en la muestra de caracteres, esto se debe a que se necesita solo 1 carácter detectado de forma incorrecta para invalidar toda la matrícula.

### Análisis de un vídeo

Para poner a prueba el modelo y la configuración de la arquitectura en la nube se analizó un video en una ruta de una vía única en la cual se observa el paso de 6 automóviles. Al ejecutar el código se observó la detección y el seguimiento de los 6 automóviles en la zona de interés. Las imágenes se pueden observar en el anexo 5 y 6. En paralelo se observó el ingreso de las imágenes en el *bucket*, estas imágenes se muestran en el anexo 6. Por último, se observó el registro de los automóviles en la base de datos. De los cuales 6 de 6 matrículas se detectaron de forma correcta. A continuación, se muestra una imagen de los registros y campos en la base de datos.



ID	timestamp	ImageName	Multa	Plate	Speed
1_05-07-22-02:13:13	05-07-22-02:13:13	1_05-07-22...	No	A-R606L	23.932004528726605
2_05-07-22-02:13:18	05-07-22-02:13:18	2_05-07-22...	No	A-E670S	25.79153069071692
3_05-07-22-02:13:25	05-07-22-02:13:25	3_05-07-22...	Si	A-PH88	57.957492241789815
5_05-07-22-02:13:30	05-07-22-02:13:30	5_05-07-22...	Si	A-96886	42.28058986682828
4_05-07-22-02:13:30	05-07-22-02:13:30	4_05-07-22...	Si	A-3K961	35.37441771112648
6_05-07-22-02:13:34	05-07-22-02:13:34	6_05-07-22...	Si	A-V6190	38.06245635261126

*Ilustración 40 Tabla de autos detectados en la base de datos*

*Fuente: elaboración propia (2022)*

### Consumo de Recursos

Para determinar la cantidad de recursos computacionales para ejecutar el código en la tarjeta Jetson Nano se utilizó la herramienta de utilidad *tegrastats*. Este



comando retorna el consumo de CPU, memoria RAM, consumo energético entre otros.

```
RAM 2932/3964MB (lfb 108x4MB) SWAP 46/4096MB (cached 5MB) IRAM 0/252kB(lfb 252kB) CPU [52%@1479,35%@1479,24
%@1479,26%@1479] EMC_FREQ 23%@1600 GR3D_FREQ 50%@921 NVDEC 396 APE 25 PLL@23C CPU@26.5C iwlwifi@27C PMIC@50
C GPU@24.5C AO@34C thermal@25.25C POM_V5_IN 5006/4058 POM_V5_GPU 1869/876 POM_V5_CPU 1097/1131
RAM 2978/3964MB (lfb 107x4MB) SWAP 46/4096MB (cached 5MB) IRAM 0/252kB(lfb 252kB) CPU [46%@1479,41%@1479,34
%@1479,37%@1479] EMC_FREQ 22%@1600 GR3D_FREQ 0%@921 NVDEC 396 APE 25 PLL@23.5C CPU@26.5C iwlwifi@30C PMIC@5
0C GPU@24.5C AO@34C thermal@25.5C POM_V5_IN 4762/4074 POM_V5_GPU 1221/883 POM_V5_CPU 1628/1143
RAM 2976/3964MB (lfb 107x4MB) SWAP 50/4096MB (cached 7MB) IRAM 0/252kB(lfb 252kB) CPU [53%@1479,62%@1479,52
%@1479,53%@1479] EMC_FREQ 22%@1600 GR3D_FREQ 82%@921 NVDEC 396 APE 25 PLL@23C CPU@27.5C iwlwifi@30C PMIC@50
C GPU@24.5C AO@34.5C thermal@26C POM_V5_IN 5945/4116 POM_V5_GPU 1941/907 POM_V5_CPU 1698/1155
RAM 2932/3964MB (lfb 108x4MB) SWAP 50/4096MB (cached 7MB) IRAM 0/252kB(lfb 252kB) CPU [44%@1479,36%@1479,45
%@1479,33%@1479] EMC_FREQ 23%@1600 GR3D_FREQ 63%@921 NVDEC 396 APE 25 PLL@23C CPU@27C iwlwifi@28C PMIC@50C
GPU@24.5C AO@36C thermal@25.25C POM_V5_IN 5356/4142 POM_V5_GPU 2312/937 POM_V5_CPU 1014/1152
RAM 2933/3964MB (lfb 108x4MB) SWAP 50/4096MB (cached 7MB) IRAM 0/252kB(lfb 252kB) CPU [52%@1479,34%@1479,27
%@1479,25%@1479] EMC_FREQ 23%@1600 GR3D_FREQ 14%@921 NVDEC 396 APE 25 PLL@23C CPU@26.5C iwlwifi@27C PMIC@50
C GPU@24C AO@35C thermal@25.25C POM_V5_IN 4843/4157 POM_V5_GPU 1546/950 POM_V5_CPU 1300/1155
RAM 2978/3964MB (lfb 107x4MB) SWAP 50/4096MB (cached 7MB) IRAM 0/252kB(lfb 252kB) CPU [36%@1479,17%@1479,8%
@1479,6%@1479] EMC_FREQ 18%@1600 GR3D_FREQ 0%@921 APE 25 PLL@22.5C CPU@25C iwlwifi@30C PMIC@50C GPU@23C AO@32
.5C thermal@24C POM_V5_IN 2232/4117 POM_V5_GPU 123/933 POM_V5_CPU 495/1141
RAM 2934/3964MB (lfb 108x4MB) SWAP 49/4096MB (cached 6MB) IRAM 0/252kB(lfb 252kB) CPU [15%@1479,33%@1479,2%
@1479,1%@1479] EMC_FREQ 12%@1600 GR3D_FREQ 0%@921 APE 25 PLL@22C CPU@25C iwlwifi@28C PMIC@50C GPU@23C AO@32
.5C thermal@24C POM_V5_IN 2683/4088 POM_V5_GPU 123/917 POM_V5_CPU 700/1132
RAM 2924/3964MB (lfb 108x4MB) SWAP 48/4096MB (cached 5MB) IRAM 0/252kB(lfb 252kB) CPU [6%@1479,47%@1479,5%@
```

Ilustración 41 Consumo de recursos del código en la Jetson Nano

Fuente: elaboración propia (2020)

De las métricas obtenidas de *tegrastats* se determinó que al ejecutar el código se consumen 2.932 GB de RAM de 3.965 GB disponibles, 2.026GHz de 5.916GHz disponibles 6 W de potencia y posee picos de consumo del 82% en GPU

## V. CONCLUSIONES

- Se utilizó aprendizaje por transferencia para re-entrenar la red neuronal para las inferencias de automóviles y se determinó que el promedio de la precisión media promedio para diferentes valores de Intersección sobre unión del modelo re-entrenado fue del 56.33%,
- Aunque el promedio de precisiones medias promedio fue de 56.33%, un umbral IoU de 0.5 será lo mínimo necesario para hacer una predicción, lo cual eleva la precisión de nuestro modelo a 73.17
- Se desarrolló el código para la identificación y seguimiento del automóvil partiendo de las coordenadas de las esquinas superior izquierda e inferior derecha retornadas por la red neuronal
- Se abstraigo la numeración de la matricula utilizando *Rekognition* y se determinó que el rendimiento en la detección de caracteres de matrículas en imágenes de automóviles similares a los que se observarían en producción fue del 96% mientras que el rendimiento en detección de toda la secuencia de la matricula fue del 81%.
- Se observó que al aplicar funciones correctivas al resultado de la detección de *Rekognition* se podía minimizar la cantidad de detecciones incorrectas.
- La implementación de un sistema borde-nube permite descargar a la nube del procesamiento de datos masivos y relegar esa tarea a los dispositivos de borde.
- Por medio de las arquitecturas IoT, la computación de borde y la computación de nube se pueden implementar soluciones agiles con presupuestos menores a los de las infraestructuras tradicionales.

- Se almacenó el ID de seguimiento, *timestamp* y numeración de la matrícula en una tabla de DynamoDB

## VI. RECOMENDACIONES

- Colocar el sistema en producción utilizando un módulo Jetson de categoría productiva en sustitución del módulo Jetson Nano Developer Kit
- Encriptar los datos en reposo en el *bucket* de S3
- Configurar alertas y presupuestos en AWS Budgets
- Realizar el entrenamiento por transferencia con las capas de la red neuronal de base congeladas y únicamente descongelarlas en las últimas épocas, esto permite ahorrar tiempo en el entrenamiento y evitar desajustar los pesos de las capas de la red neuronal de base
- Colocar el sistema en producción configurar la opción *Intelligent Tiering* de AWS S3 para el archivo de imágenes de los vehículos en un almacenamiento de menor costo.
- Construir o re-entrenar la red neuronal utilizando Keras, PyTorch o Tensorflow en lugar de utilizar los scripts del entrenamiento del contenedor `dusty-nv/jetson-inference`
- Observar de mejor forma el proceso de aprendizaje imprimiendo la curva de aprendizaje en cada época
- La efectividad del modelo se puede optimizar mediante un set de datos más representativo, ajustes en el tamaño de los lotes y épocas de entrenamiento, ajustes en la función de optimización y tiempo para realizar y evaluar los ajustes.

## VII. REFERENCIAS

- Aggarwal, C. (2018) *Neuronal Networks and Deep Learning*. Springer
- Bobadilla, J. (2021) Machine Learning y Deep Learning: Usando Python, Scikit y Keras
- Cloud Education, IBM, (2020). ¿Qué son las redes neuronales? Obtenido de: <https://www.ibm.com/mx-es/cloud/learn/neural-networks>
- Cloud Education, IBM, 2020. What are Convolutional Neural Networks? Obtenido de: <https://www.ibm.com/cloud/learn/convolutional-neural-networks>
- Cloud Education, IBM (2020) *Gradient Descent*. Obtenido de: <https://www.ibm.com/cloud/learn/gradient-descent>
- Developers.arcgis.com. n.d. How single-shot detector (SSD) works? | ArcGIS Developer. Obtenido de: <https://developers.arcgis.com/python/guide/how-ssd-works/>
- Ekman, M. (2021) *Learning Deep Learning Theory and Practice of Neuronal Networks, computer vision, natural language processing and transformers using tensorflow*. Addison-Wesley
- Erl, T. Mahmood, Z. Puttini R. (2013) *Cloud Computing concepts, technology & architectures*. Prentice Hall
- Hakim, A. (2018) *Internet of Things (IoT) System Architecture and Technologies*. Obtenido de: [https://www.researchgate.net/publication/323525875 Internet of Things IoT System Architecture and Technologies White Paper](https://www.researchgate.net/publication/323525875_Internet_of_Things_IoT_System_Architecture_and_Technologies_White_Paper)

Lea, P. (2020) *IoT and Edge Computing for Architects*. Packt

Kant, k. Temitayo, R. Mahrishi, M. (2019) *Cloud Computing*. BPB

IBM. (s.f) *Introduction to Deep Learning and Neuronal Networks with Keras*.  
Coursera

Lee, S. (2021) *Implementing Single Shot Detector (SSD) in Keras*. Obtenido de:  
<https://towardsdatascience.com/implementing-ssd-in-keras-part-i-network-structure-da3323f11cff>

Open Neuronal Network Exchange (2022) *ONNX*. Obtenido de:  
<https://github.com/onnx/onnx>

Patrikar, S. (2019) *Batch, Mini Batch & Stochastic Gradient Descent*. Obtenido de:  
<https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a>

Rosebrock A. (2016) Intersection over Union (IoU) for object detection. Obtenido de:  
<https://pyimagesearch.com/2016/11/07/intersection-over-https://towardsdatascience.com/implementing-single-shot-detector-ssd-in-keras-part-ii-loss-functions-4f43c292ad2a> [union-iou-for-object-detection/](#)

Technologies, D. (2022) *Modern Datacenter Environment*.

Tsang, S. (2018) *Review: SSD — Single Shot Detector (Object Detection)*. Obtenido de:  
<https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11>

## VIII. ANEXOS

**Anexo 1:** Enlace al repositorio de GitHub donde se encuentra el paso a paso del desarrollo del proyecto.

<https://github.com/DiegoBran16/JetsonNano-Cloud-Edge-Car-speed-plate-monitoring>

**Anexo 2:** Enlace al repositorio de GitHub dusty-nv/jetson-inference:

<https://github.com/dusty-nv/jetson-inference>

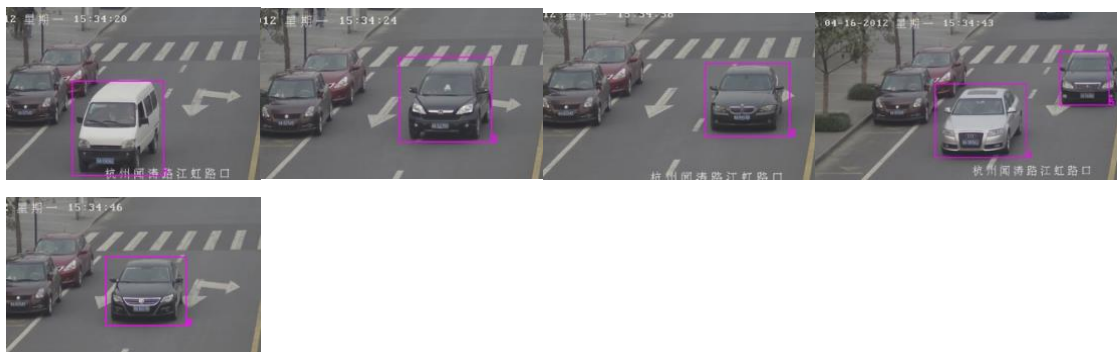
**Anexo 3:** Enlace de descarga para la red de base Mobilenet V1:

[https://nvidia.box.com/shared/static/djf5w54rjvpqocsiztzaandq1m3avr7c.pth -O models/mobilenet-v1-ssd-mp-0\\_675.pth](https://nvidia.box.com/shared/static/djf5w54rjvpqocsiztzaandq1m3avr7c.pth -O models/mobilenet-v1-ssd-mp-0_675.pth)

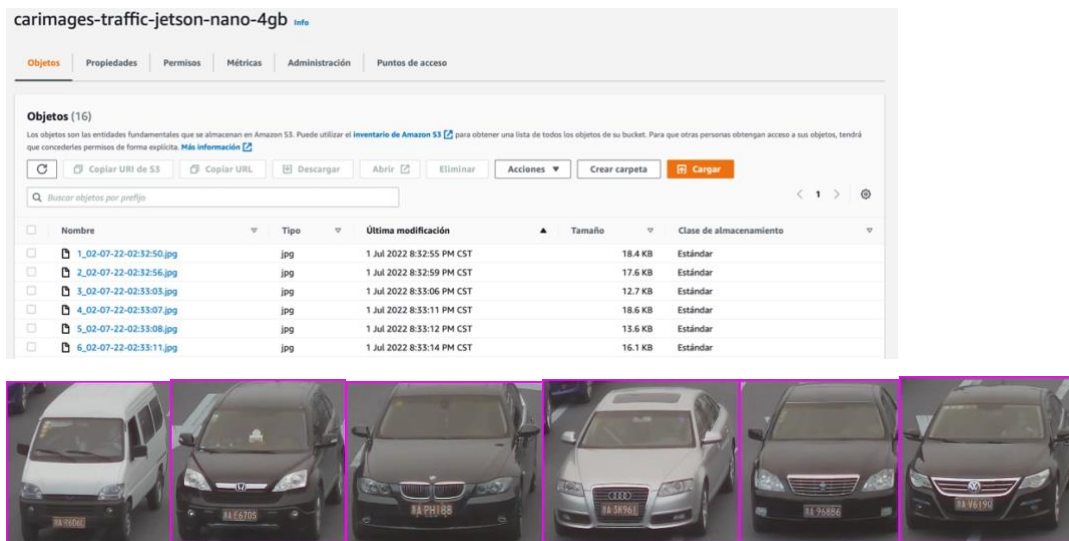
**Anexo 4:** Enlace para descargar sets de datos de Google en formato Open Images:

<https://storage.googleapis.com/openimages/web/visualizer/index.html?set=train&type=detection&c=%2Fm%2F0fp6w>

**Anexo 5:** Imagens de la detección de automoviles por el modelo re-entrenado



## Anexo 6: Imágenes en S3



## Anexo 7: Arquitectura de red neuronal re-entrenada

SSD(

(base\_net): Sequential(

(0): Sequential(

(0): Conv2d(3, 32, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1),  
bias=False)

(1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,  
track\_running\_stats=True)

(2): ReLU(inplace=True)

)

(1): Sequential(

(0): Conv2d(32, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1),  
groups=32, bias=False)

(1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,  
track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): Conv2d(32, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)



```

        (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
    )
    (2): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
groups=64, bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (5): ReLU(inplace=True)
    )
    (3): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=128, bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (5): ReLU(inplace=True)
    )
    (4): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
groups=128, bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

```

```

(2): ReLU(inplace=True)
(3): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
(4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(5): ReLU(inplace=True)
)
(5): Sequential(
  (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=256, bias=False)
  (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (5): ReLU(inplace=True)
)
(6): Sequential(
  (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
groups=256, bias=False)
  (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (5): ReLU(inplace=True)
)
(7): Sequential(
  (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=512, bias=False)

```

```

(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(2): ReLU(inplace=True)
(3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
(4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(5): ReLU(inplace=True)
)
(8): Sequential(
  (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=512, bias=False)
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (5): ReLU(inplace=True)
)
(9): Sequential(
  (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=512, bias=False)
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (5): ReLU(inplace=True)
)
(10): Sequential(

```

```

(0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=512, bias=False)
(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(2): ReLU(inplace=True)
(3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
(4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(5): ReLU(inplace=True)
)
(11): Sequential(
  (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=512, bias=False)
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (5): ReLU(inplace=True)
)
(12): Sequential(
  (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
groups=512, bias=False)
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (4): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (5): ReLU(inplace=True)
)

```

```

)
(13): Sequential(
  (0): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=1024, bias=False)
  (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (4): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (5): ReLU(inplace=True)
)
)
(extras): ModuleList(
  (0): Sequential(
    (0): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (1): ReLU()
    (2): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): ReLU()
  )
  (1): Sequential(
    (0): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1))
    (1): ReLU()
    (2): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): ReLU()
  )
  (2): Sequential(
    (0): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
    (1): ReLU()
    (2): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): ReLU()
  )

```

```

)
(3): Sequential(
  (0): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
  (1): ReLU()
  (2): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (3): ReLU()
)
)
(classification_headers): ModuleList(
  (0): Conv2d(512, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): Conv2d(1024, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (2): Conv2d(512, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): Conv2d(256, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): Conv2d(256, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): Conv2d(256, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
)
(regression_headers): ModuleList(
  (0): Conv2d(512, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): Conv2d(1024, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (2): Conv2d(512, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): Conv2d(256, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): Conv2d(256, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): Conv2d(256, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
)
(source_layer_add_ons): ModuleList()
)

```

## Anexo 8: Sensor Sony IMX477

Se recomienda utilizar el sensor Sony IMX477 en la implementación en producción, este cuenta con las siguientes características:

Resolución	4056(H) x 3040(V)
Megapíxeles	12.3
Diagonal óptica	7.8 mm
Conector	CSI
Tamaño del píxel	1.55um x 1.55um
Matriz de filtros de color	RGB
Interfaz de salida	MIPI de 4 líneas
Resolución ADC	12-bit, on-chip
Cuadros por segundo	<ul style="list-style-type: none"><li>- 15 a 240 fps dependiendo del modo de video</li><li>- Resolución completa @60 frames/s (normal), 4k2k @60 frames/s, 1080p @240 frames/s</li><li>- Resolución completa @60 frames/s (12 bit normal), Resolución completa @30 frames/s (DOL-HDR, 2 frame),</li><li>- (Si se utiliza una configuración MIPI de 2 líneas, los cuadros por segundo se dividen a la mitad)</li></ul>

*Tabla 5 Características del sensor Sony IMX477*

*Fuente: ArduCam(2022)*

## Anexo 9: Lente Arducam 2.8-12mm varifocal montura C

El lente recomendado para colocar el sistema en producción es un lente de zoom variable de 2.8-12mm con montura en C, este lente es compatible con el sensor Sony IMX477. Sus principales características se muestran a continuación:

Formato Óptico	1/2"
Longitud Focal	2.8-12 mm
Montaje	Montaje en C
Enfoque de Respaldo	17.53 mm
M.O. D.	0.3 m
Dimensiones	Φ44.5 x58.5 mm
Campo de visión	125°(H)-43°(H), 120°(H)-38°(H) on 1/2.3"
Tipo de enfoque	Manual
Apertura	1:1.6

*Tabla 6 Características del lente zoom 2.8-12mm de ArduCam*

*Fuente ArduCam (2022)*

## **Anexo 10: Costos del sistema Actual**

Estimado de CAPEX de implementación del sistema utilizando una tarjeta Jetson Nano

Componente	Precio	Envío	Horas	Total
Kit de Desarrollo Jetson Nano	\$ 220.00	Q 95.00	N/A	Q 1,817.60
Tarjeta de Red y Antenas	\$ 25.47	Q 88.10	N/A	Q 287.53
Ventilador	\$ 8.90	Q 82.40	N/A	Q 152.09
Máquina Virtual Azure ( \$ 200 gratis)	\$ 0.68	N/A	200	Q 136.00
Tarjeta MicroSD	\$ 700.00	N/A	N/A	Q 700.00
Cámara Raspberry Pi con Sensor Sony IMX477	\$ 50.00	Q 86.40	N/A	Q 477.90
Lente Varifocal 2.8mm - 12 mm	\$ 65.00	Q 85.15	N/A	Q 594.10
Panel Solar 45 W	Q 256.00	N/A	N/A	Q 256.00
Regulador a 9V	\$ 5.08	Q 81.36	N/A	Q 120.98
UPS Waveshare	\$ 26.00	Q 88.50	N/A	Q 291.30
Baterías Recargables 9900 mAh	\$ 34.00	Q 90.00	N/A	Q 355.20
<b>CAPEX TOTALES</b>				<b>Q 5,188.70</b>

*Tabla 7 CAPEX del sistema actual*

*Fuente: elaboración propia (2020)*



Estimación del OPEX para el primer mes utilizando de parámetro 15,000 autos detectados diarios de lunes a sábado, reteniendo únicamente 1 mes de datos.

Servicio	Precio \$	Precio Q	Total Q
S3 Standar por GB	\$ 2.069	Q 16.200	Q 104.98
Lambda por millón de solicitudes	N/A	N/A	N/A
DynamoDB al mes DetectedCars	\$ 1.09	Q 8.535	Q 8.535
DynamoDB al mes CarOwners	\$ 0.51	Q 3.99	Q 3.99
IoT Core	\$ 0.36	Q 2.819	Q 1.01
IAM	N/A	N/A	N/A
<b>OPEX TOTALES</b>			<b>Q 118.521</b>

Tabla 8 OPEX del primer mes del sistema actual

Fuente: elaboración propia (2022)

Proyección de costos mensuales para el primer año con el sistema actual

Mes	Costo Operativo
Mes 1	Q 118.521
Mes 2	Q 118.521
Mes 3	Q 118.521
Mes 4	Q 118.521
Mes 5	Q 118.521
Mes 6	Q 118.521
Mes 7	Q 118.521
Mes 8	Q 118.521
Mes 9	Q 118.521
Mes 10	Q 118.521
Mes 11	Q 118.521
Mes 12	Q 118.521
<b>Total Año 1</b>	<b>Q 1,422.246</b>

Tabla 9 Costos del primer año con el sistema actual

Fuente: elaboración propia (2022)

## Anexo 11: Costos del sistema Mejorado

Estimado de CAPEX de implementación del sistema utilizando una tarjeta Jetson Xavier de 8GB

Componente	Precio	Envío Q	Horas	Total Q
Kit de Desarrollo Jetson Xavier 8GB	\$ 893.99	Q 1,626.77	N/A	Q 8,626.71
Máquina Virtual Azure	\$ 0.68	N/A	500.00	Q 340.00
Cámara Raspberry Pi con Sensor Sony IMX477	\$ 50.00	Q 86.40	N/A	Q 477.90
Lente Varifocal 2.8mm - 12 mm	\$ 50.00	Q 85.15	N/A	Q 476.65
Panel Solar 45 W	Q 256.00	N/A	N/A	Q 256.00
Regulador a 9v	\$ 5.08	Q 81.36	N/A	Q 120.98
UPS Waveshare	\$ 26.00	Q 88.50	N/A	Q 291.30
Baterías Recargables 9900 mAh	\$ 34.00	Q 90.00	N/A	Q 355.20
<b>CATEX TOTALES</b>				<b>Q 10,944.75</b>

Tabla 10 CAPEX sistema mejorado

Fuente: elaboración propia (2022)

Estimación del OPEX para el primer mes utilizando de parámetro 15,000 autos detectados diarios de lunes a sábado, reteniendo únicamente 1 mes de datos.

Servicio	Precio \$	Precio Q	Total Q
S3 Standar por GB	\$ 2.069	Q 16.200	Q 104.98
Lambda por millón de solicitudes	N/A	N/A	N/A
DynamoDB al mes DetectedCars	\$ 1.09	Q 8.535	Q 8.535
DynamoDB al mes CarOwners	\$ 0.51	Q 3.99	Q 3.99
IoT Core	\$ 0.36	Q 2.819	Q 1.01
IAM	N/A	N/A	N/A
<b>OPEX TOTALES</b>			<b>Q 118.521</b>

Tabla 11 OPEX Primer mes

Fuente: elaboración propia (2022)

Proyección de costos mensuales para el primer año con el sistema mejorado

Mes	Costo Operativo
Mes 1	Q 118.521
Mes 2	Q 118.521
Mes 3	Q 118.521
Mes 4	Q 118.521
Mes 5	Q 118.521
Mes 6	Q 118.521
Mes 7	Q 118.521
Mes 8	Q 118.521
Mes 9	Q 118.521
Mes 10	Q 118.521
Mes 11	Q 118.521
Mes 12	Q 118.521
<b>Total Año 1</b>	<b>Q 1,422.246</b>

Tabla 12: Proyección de Costos para el primer año con el sistema mejorado

Fuente: elaboración propia (2022)