BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

“Supervisión de políticas y procesos en áreas operativas usando redes neuronales.”

Tesis presentada para obtener el grado de:

LICENCIATURA EN INGENIERÍA

EN CIENCIAS DE LA COMPUTACIÓN

Presenta:

Gustavo Bañuelos Ochoa

Director de Tesis:

Dr. David Eduardo Pinto Avendaño

Asesor de Tesis:

Dr. David Eduardo Pinto Avendaño

Febrero 2023

A mis padres y hermanas,
todo lo que soy o alguna vez seré
se lo debo a su infinito amor y apoyo,
gracias por tanto en esta vida.

AGRADECIMIENTOS

Antes de comenzar esta memoria, quiero agradecer a aquellos que han hecho posible la realización de este trabajo.

En primer lugar, me gustaría agradecer a mi asesor, Dr. David Eduardo Pinto Avendaño, por compartir conmigo la propuesta de tesis, sus ideas y conocimientos. Gracias por su apoyo no solo como docente, también como amigo y ser humano; a todo el cuerpo académico del laboratorio *Language and Knowledge Engineering* (*LKE*), profesores que en múltiples ocasiones creyeron en mí y contribuyeron enormemente a mi adquisición de conocimiento.

A mi familia y amigos gracias por todo su apoyo día con día, antes y durante la realización de este proyecto, sin el cual no habría sido posible finalizar y por estar a mi lado en los momentos importantes de la vida.

Por último, agradecer a los grupos de investigación y entidades que crean bibliotecas, conjuntos de datos, comparten material de aprendizaje y financian proyectos que permiten a otros progresar a mayor velocidad en alcanzar sus objetivos.

RESUMEN

Las redes neuronales convolucionales profundas y arquitecturas derivadas de este paradigma, han demostrado su capacidad para la solución de problemas de detección, clasificación, localización y segmentación de imágenes, abstrayendo millones de características y aprendiendo de grandes bases de datos. Hoy en día existen distintas formas de modelar y simular su comportamiento. Este proyecto se ha dado a la tarea de concebir una herramienta de software que englobe y permita usar esta tecnología a través de una interfaz sencilla, creando y entrenando prototipos con relativa facilidad, permitiendo usar los resultados generados de forma complementaria a otros sistemas. Así mismo se prueba su eficacia usada en la búsqueda y reconocimiento de mobiliario en áreas operativas de trabajo; generando una base de datos propia recolectada para este caso de uso.

Tabla de Contenidos

Tabla de Contenidos 9

1. Introducción 12

1.1 Antecedentes 12

1.2 Planteamiento del problema 13

1.3 Objetivos 15

1.3.1 Objetivo general 15

1.3.2 Objetivo específico 15

1.4 Justificación de la demanda 16

2. Marco Teórico 17

2.1 La IA y las Redes neuronales artificiales 17

2.2 Machine Learning y Deep Learning 18

2.3 El modelo biológico 19

2.3.1 El Aprendizaje 20

2.4 Neuronas artificiales 20

2.4.1 El Perceptrón 22

2.5 Estructuras neuronales artificiales 24

2.6 Redes neuronales multicapa 25

2.7 Aprender y recordar lo aprendido 26

2.7.1 Error y descenso por el gradiente 27

2.7.2 Backpropagation 28

2.8 Redes neuronales convolucionales 29

2.9 Transferencia de aprendizaje 32

2.10 Estado del Arte 33

3. Desarrollo del Framework 35

3.1 Metodología 35

3.2 Entornos de trabajo para redes neuronales 36

3.2.1 Configuración del entorno 36

3.3 Arquitectura propuesta y programación de fases 37

3.3.1 Fase de preparación de datos 38

3.3.2 Fase de selección y configuración del modelo 43
3.3.3 Fase de entrenamiento 49
3.3.4 Fase de resultados 52
4. Aplicación caso de uso: Estado de mobiliario 57
4.1 Base de datos de aprendizaje y test 57
4.2 Creación de entradas 59
4.3 Configuración del modelo 60
4.3.1 SSD: Single Shot MultiBox Detector 61
4.3.2 Ajuste de parámetros para SSD 63
4.4 Resultados del entrenamiento 67
5. Conclusiones 71
6. Bibliografía 72

1. Introducción

1.1 Antecedentes

El impacto de la tecnología digital en la era actual es innegable y aunque cuestionable, está presente en casi todas facetas de la actividad social y profesional del ser humano, las tecnologías de la información y comunicación (TIC), han transformado de manera significativa la forma en que trabajamos, nos relacionamos, entretenemos, así como creamos y compartimos conocimiento. El término relacionado a este fenómeno se conoce apropiadamente como era de la información, un momento de la historia, donde alcanzado cierto grado de avance tecnológico, creamos, almacenamos y compartimos información de forma masiva, incluso generamos más datos de los que podemos procesar, esto debido a que con cada nuevo invento para la comunicación e información se genera un incremento en el volumen de información disponible en un entorno. Prueba de esto son las empresas más rentables a nivel mundial, tales como Apple, Microsoft corp, Amazon, Facebook Inc, Alibaba Grpdr, entre otros [1]. Las cuales dedican gran parte de su tiempo recolectando información de una manera u otra, así como ampliando el consumo de sus productos en mercados nacionales e internacionales. Dichos productos al mismo tiempo son potenciados por los beneficios surgidos de su capacidad de usar los datos que recolectan, convirtiéndolos en información útil de forma rápida y precisa.

En los últimos años este objetivo se ha visto altamente impulsado gracias a la inteligencia artificial; un área de prolífico interés es la del aprendizaje profundo o Deep learning, enfocado en otorgar a las máquinas la capacidad de aprender comportamientos inteligentes, recuperando esta funcionalidad automáticamente analizando datos. Esto ha logrado avances considerables en diversos campos y nos han permitido extender las competencias que damos a nuestras máquinas. Existe una variedad de técnicas y modelos de aprendizaje profundo para la solución de problemas, las redes neuronales son una de las más populares y extendidas, en gran parte debido a su propiedad de adaptación a diferentes problemas.

Gran parte de los servicios y dispositivos que consumimos, hacen uso de estas técnicas, desde nuestro proveedor de correo electrónico que clasifica el correo como spam, a los servicios de entretenimiento y ventas que constantemente nos aconsejan productos que sean de nuestro interés basados en nuestros gustos. No obstante, el diseño y correcto despliegue de estos modelos, no es tarea sencilla, lo cual genera una brecha en el uso de estas técnicas que suelen aislarse en el campo de la investigación, limitando en parte su uso para otros desarrolladores no expertos en la materia. En consecuencia, a este fenómeno pequeñas empresas que no pueden competir o son ajenas a las posibilidades que pueden representar para ellos, quedan fuera la competencia que ya ha comenzado.

Este enfoque a la industria retail será de gran valor en el estudio del capítulo 1 para comprender su aplicación desde una perspectiva técnica.

Según [2] *"En el campo de la visión por computadora comúnmente cuando trabajamos con imágenes, es necesario no solo la clasificación de diferentes imágenes, también es preciso estimar el concepto y ubicación de los objetos contenidos en cada imagen, esto se conoce como detección de objetos, y compone uno de los problemas fundamentales en el estudio de la visión artificial"*. Usualmente consiste de subtareas tales como detectar rostros, peatones, vehículos, etc. En los últimos años este problema ha encontrado un renovado impulso en el uso de técnicas de aprendizaje profundo, más específicamente en el uso de redes neuronales convolucionales (CNN), cuyos resultados han dominado notablemente el estado del arte e incluso superado la capacidad del ser humano.

Una razón por la cual las CNN resultan muy buenas en problemas que involucran imágenes o video, es porque la imagen es procesada conectada a pequeñas secciones en lugar de tener una cantidad ingente de neuronas por pixel fuertemente conectadas, estas secciones crean filtros llamados kernels, que se desplazan a lo largo de la imagen con el fin de calcular las operaciones respectivas de suma y producto. Las CNN surgen de los estudios relacionados al funcionamiento de la corteza visual en los mamíferos. Existe una variedad de arquitecturas que hoy obtienen los mejores resultados, respecto a velocidad y precisión, sin embargo, las CNN se encuentran presentes como base conceptual.

En este trabajo se investigan los principios de redes neuronales, se propone el contexto de un caso de uso y se da solución al mismo. Además, se propone una estructura de software (framework) que incluye un conjunto de herramientas necesarias para el correcto desarrollo en cada una de las fases que componen esta labor. Esto pensando en la creación de herramientas que faciliten su uso.

1.2 Planteamiento del problema

Con mayor frecuencia diversos sectores de la población ponen su atención en la IA (académicos, gobiernos, industria privada), experimentando y obteniendo resultados usando los modelos ya existentes o centrando esfuerzos en resolver nuevos problemas. El Dr. Kai-Fu Lee sostiene en [3] que:

El trabajo de estos investigadores y desarrolladores requiere buenas habilidades y un profundo conocimiento, tal como la habilidad de ajustar complejos algoritmos matemáticos para manipular masivas cantidades de datos, esto a menudo requiere un nivel Ph.D. en esas áreas, sin embargo, los avances son incrementalmente mejorados y optimizados, lo cual influencia dramáticamente el salto hacia el deep learning.

Esto deja en evidencia una visión general del problema, es decir el grado de dificultad en cuanto a la labor que supone hacer uso de dichas tecnologías, y la creciente demanda por mejorar continuamente las herramientas disponibles o incluso contribuyendo con nuevas.

La función de estas es la del instrumento, que diseña y simula modelos neuronales. A esta causa cada año diferentes comunidades hacen aportaciones a diversos proyectos ya sea con la creación de bibliotecas, expandiendo el número de entornos para su uso, o aportando a la mejora continua del estado del arte.

El campo de la visión por computadora no es la excepción, ahora diversas organizaciones hacen posible tener acceso a múltiples herramientas para solventar problemas de clasificación, localización, descripción de escenas, detección y segmentación de objetos. Usualmente los resultados obtenidos son empleados como parte de otras aplicaciones.

Un ejemplo serían los sistemas de conducción automática (self driving car), compuestos por múltiples sensores y cámaras, que observan el entorno alrededor del auto, detectando y estimando el tipo, cantidad y distancia de cada objeto, tal como peatones, autos, bicicletas, semáforos, caminos, animales, etc. Esta forma de percibir el ambiente a su alrededor permite estimar al automóvil que acción tomar respecto a su estado actual. Tradicionalmente un conductor debe concentrar sus sentidos para observar y decidir qué hacer en todo momento, sin embargo, cabe la posibilidad de que surjan eventos que aparten su atención incluso solo por un momento, lo cual es un factor de alto riesgo tanto para él como para terceros, a diferencia, estos sistemas pueden auxiliar al operador de forma ininterrumpida.

Estas herramientas siguen un ritmo de trabajo reconocible, ya que particularmente el flujo de actividades seguido es comúnmente el mismo. Para comenzar al igual que otros modelos de ML, la precisión del modelo dependerá en gran parte a la calidad y cantidad de sus datos, tareas como recolectar imágenes, depurar la base de conocimiento de datos que puedan perjudicar el entrenamiento, y describir etiquetas correspondientes al objeto en cada imagen, absorberán gran parte de tiempo en el desarrollo. Una vez los datos han pasado por este proceso es necesario ubicar una arquitectura neuronal adecuada a nuestros objetivos, así como un lenguaje que la codifique y pueda usar los resultados generados, en esta parte se especifican hiper parámetros que usualmente deben ajustarse con el fin de mejorar los resultados obtenidos. Posteriormente se inicia la fase de aprendizaje o entrenamiento, para esto es necesario tomar un porcentaje considerable de la base de datos, dejando el porcentaje restante para luego evaluar su exactitud. Los datos alimentados se dividirán en lotes más pequeños, cada lote es mostrado al sistema con su correspondiente etiqueta, en el momento que el número total de lotes se completa, habrá finalizado una época, transcurrido un número definido de épocas de iteración, el error generado debe decrementar su valor. Esto sería nada sin la capacidad de poder demostrar que el aprendizaje rindió frutos, por último, para comprobar su eficiencia el modelo debe someterse a una evaluación en sus predicciones, si la calidad en la detección es como lo esperado, solo queda liberar el modelo para su despliegue en otros entornos; por otro lado, si el resultado no es satisfactorio, es trabajo del desarrollador reajustar atributos y repetir el entrenamiento.

Hasta aquí definimos el planteamiento del problema principal, referente a las habilidades implicadas al detectar objetos y el necesario acercamiento de mejores herramientas que faciliten esta labor.

1.3 Objetivos

1.3.1 Objetivo general

Desarrollar una arquitectura de software basada en Inteligencia Artificial, que englobe las fases necesarias para entrenar modelos de redes neuronales (de forma relativamente simple), una vez entrenado el modelo será capaz de detectar y discriminar instancias de clases discretas proporcionadas por el usuario, las cuales deben delimitarse por recuadros de manera automática, a partir de imágenes o video. Como objetivo adicional se hará uso de dicha arquitectura para dar solución a un caso de estudio, en el cual se desea reconocer múltiples instancias de mobiliario en un establecimiento, así como su estado actual (libre u ocupado) con respecto a los comensales.

1.3.2 Objetivo específico

- Diseño de una interfaz de software que abarque cada una de las fases y módulos necesarios para el entrenamiento de redes neuronales.

- Definir una organización interna de rutas específicas, que ayuden al usuario a ubicar sus proyectos y herramientas del software.

- Adaptar una infraestructura de software para la caracterización o anotado manual de muestras, para su posterior uso en el proceso de entrenamiento de modelos para la clasificación.

- Determinar el tipo de bibliotecas existentes y unificar una estructura que permita acceder y esquematizar modelos neuronales, en forma comprensible incluso para usuarios no expertos.

- Permitir el despliegue de resultados de cada modelo entrenado de forma visual usando gráficas.

- Evaluar la utilidad del modelo al finalizar el entrenamiento de un proyecto.

- Acreditar la eficiencia del sistema para el caso de estudio de clasificar mesas ocupadas o libres en un comercio.

1.4 Justificación de la demanda

A diferencia de las pasadas décadas, en las cuales amplios esfuerzos de investigación en IA concretaron gran parte del conocimiento teórico necesario, ahora la humanidad cuenta con tres piezas claves para el desarrollo de estos sistemas: poder de cómputo, volúmenes masivos de datos y personal capacitado para su desarrollo. Siendo estos últimos no necesariamente científicos de élite en IA. Lo cual ha reanimado las expectativas de cómo cambiarán la calidad de vida de los seres humanos. Su impacto ha sido comparado a la revolución industrial suscitada a lo largo del siglo XIX y XX, momento en que la civilización humana absorbió tecnologías que condujeron a una gran conmoción económica, cuando la mano de obra de cientos de miles de granjeros, se convirtió en mano de obra para fábricas. Hoy en día esta disrupción apunta a una nueva revolución tecnológica.

Por otra parte, diversos estudios [4] han revelado que hasta el 89% de las personas han dejado de hacer negocios con una compañía debido a una carente experiencia de servicio al cliente. Un motivo de gran peso, se debe al recurrente incumplimiento de políticas, definidas para el correcto despliegue de operaciones en áreas de trabajo, siendo de mayor frecuencia las áreas donde intervienen empleados, incluso existiendo un supervisor. Podemos decir entonces que, en un entorno de este tipo, regularmente se exige un estimado de calidad en cuanto al servicio brindado al cliente.

No obstante, un sistema capaz de identificar y tomar acciones, respecto a un conjunto amplio de lineamientos, supone un reto de gran complejidad, ya que es necesario diseñar, implementar y conectar múltiples módulos. Eligiendo como ejemplo, la propuesta para una empresa y sus lineamientos, la labor de este módulo consiste en el monitoreo de un espacio de trabajo a través de cámaras, el agente utilizado debe ubicar y discrepar entre el estado de mesas en un lugar de reunión de clientes o comensales, esto permite saber si las mesas están ocupadas, o por el contrario están vacías, una vez identificado, puede tomarse como parámetro para diferenciar si dichas mesas necesitan ser aseadas si el caso lo amerita, una notificación en respuesta es enviada a un supervisor a través de un monitor o aplicación móvil. Una vez informado, el supervisor será capaz de dictaminar funciones correctivas.

Por último, la idea de compartir código y funcionalidad para que más personas puedan aprender y hacer sus propias aportaciones, son principios bajo los cuales nació el internet, en la actualidad no es la excepción, diferentes comunidades, incluso empresas grandes, con mayor frecuencia liberan código que antes se consideraba como privativo, bajo la protección de diferentes licencias, que posibilitan legalmente su distribución, uso y manipulación libre por otros usuarios, incluso al mismo tiempo ofrecen seguridad a los creadores. Esta podría describirse como la principal motivación de este proyecto, no solo es aproximar pequeños sectores del mercado a sus beneficios, sino a todos aquellos individuos o comunidades motivados por la pasión de crear nuevos productos, para mejorar la calidad de vida del ser humano, o sólo de comprender aquello que lo rodea analizando el mundo que habitamos.

El proyecto es propuesto en colaboración con el laboratorio *Language and Knowledge Engineering* (LKE) de la BUAP.

2. Marco Teórico

2.1 La IA y las Redes neuronales artificiales

Se considera a la Inteligencia artificial como un campo de estudio notablemente amplio y una ciencia relativamente nueva, que nace a mitad del siglo XX, dados los avances generados tras la segunda guerra mundial, lapso en que surge la primera computadora electrónica, avances que en las siguientes décadas contarán con un asombroso auge. Se caracteriza ya que en este campo es posible hallar una profunda exploración y unificación entre diversas áreas de estudio complementarias, tales como la lógica o la probabilidad, y ahonda sobre abstracciones del pensamiento complejo del ser humano; tales como el razonamiento, el aprendizaje, la percepción y la acción. Concentrando sus esfuerzos en la construcción de entidades inteligentes.

Según Stuart J. Russell, “Definimos la IA como el estudio de los agentes que reciben percepciones del entorno e imitan acciones inteligentes”, de esto se extrae como recurso principal la idea del “agente” o “agente inteligente”, el cual cambia según la tarea a realizar, por ejemplo: conducir un auto, reconocer voces, analizar patrones o incluso ganar en juegos de considerable complejidad, suscitando una gran cantidad de sistemas que concuerdan con esta descripción.[5] Es evidente que existen muchas formas en que una máquina puede simular un agente, dada esta amplitud, la IA agrupa sub ramas de estudio enfocadas en investigar con profundidad la formación de dichos agentes, creando soluciones a distintos problemas en particular, como la robótica la cual estudia la capacidad de una máquina para moverse y responder al entorno, o el procesamiento del lenguaje natural que estudia las interacciones entre las computadoras y el lenguaje humano.

El agente en términos prácticos, es cualquier cosa capaz de percibir el estado de su ambiente usando sensores, y reaccionar en respuesta al estado actual de ese medio, a través de algún tipo de actuador. Un agente de asistente inteligente, por ejemplo, percibe información de entrada a través de voz o texto usando un micrófono o un teclado, y actúa sobre el medio con mensajes de respuesta, usando una bocina o una pantalla.

La percepción desde este punto de vista, nos dice que el agente posee la capacidad de recibir entradas en cualquier momento, cuya respuesta estaría dada por una secuencia de percepciones hasta ese instante, un historial por así decirlo, donde cada resultado depende de la estructura de dicha secuencia. Si el sistema es capaz de describir de manera específica cada respuesta para cada secuencia o combinación de percepciones posibles, será capaz de englobar cada parte de su comportamiento, es decir la función del agente. Los llamados sistemas expertos serían un claro ejemplo, sin embargo, dada la complejidad del problema, el definir el tamaño de posibilidades existentes que un equipo de programación podría incluir en su software, sería una labor casi imposible y bastante impráctica. La naturaleza de este fenómeno la podemos apreciar en dos sistemas

diferentes, en 1996 la computadora DeepBlue creada por IBM, derrotó al campeón mundial de ajedrez Garry Kasparov, esta computadora confiaba en gran medida en su hardware personalizado para generar y evaluar posiciones para cada uno de los movimientos, así también requirió un equipo de jugadores profesionales de ajedrez, para adaptar técnicas heurísticas al software, el resultado fue una pieza maestra de ingeniería, que estaba basada en conjuntos muy largos de reglas preestablecidas. En ese momento se suscitó un gran revuelo sobre cómo estas tecnologías impactarían en la vida de los seres humanos, sin embargo, lejos de una revolución tecnológica, no pasó de ser una estrategia publicitaria, funcionaba, pero no estaba ni cerca de cubrir las expectativas tan altas que había generado. El segundo sistema surge en China, desarrollada por DeepMind, subsidiaria de Google, en mayo de 2017, AlphaGo vence al mejor jugador de Go en el mundo, Ke Jie joven chino, en el que es considerado por muchos como el juego mental más difícil del mundo, aquí es necesario hacer hincapié en la importancia de este suceso, el tablero está conformado por una alineación de nueve filas por nueve columnas, colocando fichas negras y blancas, cada jugador alterna turnos intentando encerrar en un círculo las fichas del adversario, sin embargo el número de posibles posiciones en un tablero de Go excede al número de átomos que existen en el universo. En comparación AlphaGo presenta un nivel de complejidad sin precedentes y uno de los sistemas más inteligentes en el mundo.

Ante esto se entiende que describir la función del agente para un sistema como lo es AlphaGo sería en comparación una tarea utópica. Para encontrar una solución a este tipo de problemas, en la IA surge una rama de estudio de gran valor, el Machine Learning (aprendizaje automático o de máquina), el cual concentra sus esfuerzos en la creación de sistemas cuyo aprendizaje se logre de forma automática a partir de datos, es decir sin haber sido programados explícitamente para hacer una función específica. Dentro de este conjunto de teorías, en los últimos años una de las técnicas que más ha destacado son las redes neuronales artificiales, esto debido a su intrínseca habilidad de generar estructuras jerárquicas, que se apilan en capas de creciente complejidad y abstracción; lo que ha suscitado un resurgimiento del interés y un avance significativo en distintas áreas de investigación. Su principal auge se debe a que estos programas requieren acceso a inmensas cantidades de datos y poder de procesamiento, ninguno de los cuales estaba fácilmente disponible para los programadores hasta la era de la información, la computación en paralelo y el cómputo en la nube.

2.2 Machine Learning y Deep Learning

En el machine learning (ML) el agente inteligente a diferencia de otros enfoques, no surge de una cuidadosa y bien detallada programación de comportamiento, sino del desafío que es lograr que el agente “aprenda” dicho comportamiento. El aprendizaje en este contexto debe comprenderse como la aproximación a un valor generado por un modelo matemático. Para alcanzar dicho objetivo el ML engloba un conjunto de técnicas y teorías matemáticas, las cuales nos ayudan a convertir volúmenes de datos en información útil que pueden ser usadas en otros procesos.

En la última década un subconjunto dentro del campo del ML a recibido un cuantioso impulso, conocido como Deep learning o “aprendizaje profundo”, este concepto genera resultados o aprende basado en el ejemplo, en lugar de darle al ordenador una lista enorme de reglas para solventar un problema, le damos un modelo que pueda evaluar ejemplos previamente recolectados por nosotros, entre más veces evalúa qué tan lejos se encuentra su predicción del resultado que le estamos mostrando, aproxima sus distancias disminuyendo así el error, pasado un tiempo los modelos serán capaces de solucionar problemas de una forma sorprendentemente precisa. La etiqueta de profundidad se relaciona directamente a una de las técnicas más comunes, las redes neuronales artificiales, las cuales toman como base la estructura del funcionamiento del cerebro humano en su modelo más sencillo, la neurona, estos enfoques recrean simulaciones ya sean en software o hardware, que al ser estimuladas como unidades de procesamiento poseen una capacidad limitada, sin embargo una gran cantidad de ellas interconectadas pueden ser dispuestas a trabajar en paralelo logrando desarrollar fenómenos de extraordinario procesamiento. A continuación, se describe su estructura para una mejor comprensión.

2.3 El modelo biológico

El cerebro humano como elemento principal de nuestra capacidad de razonamiento y la máquina más compleja que hasta ahora conocemos, compone el núcleo del sistema nervioso humano y regula los procesos de nuestro cuerpo. Así mismo está compuesto por un tipo de células biológicas llamadas neuronas, (ver Figura 2.1) las cuales contienen todos los componentes usuales de las células, y poseen cualidades propias que posibilita la comunicación entre ellas. Cada uno de nuestros sentidos se conecta al sistema nervioso, los sentidos constituyen nuestra forma de percibir e interactuar con el entorno, es decir el mundo que nos rodea, a través de ellos se transfieren dichas percepciones a las neuronas, como consecuencia pueden ser estimuladas y producir una respuesta. Este proceso no es posible ser realizado por una sola neurona, para esto existe una red masiva interconectadas en paralelo [6]. La información es relegada en dirección a las capas de mayor profundidad, entonces, algunas de esas neuronas pueden ser activadas como respuesta a tal información, retransmitiendo a aquellas neuronas a las que estén conectadas, este proceso continúa eventualmente derivando en una respuesta o acción. Existe un estimado de que en el cerebro y sistema nervioso el ser humano posee cerca de cien mil millones de neuronas, esta forma de red asegura la división de trabajo, siendo que cada neurona desempeña un cierto tipo de rol, es decir que responde a ciertos estímulos.

Cada neurona posee una canal de entrada por el cual recibe una señal eléctrica de otras neuronas (dendritas), procesa esta información en su núcleo (soma) y finalmente transmite la salida a través de una conexión hacia otras neuronas (axón). Este punto de conexión hacia otras neuronas se le conoce como sinapsis. Esta simplificación en su funcionamiento corresponde a una forma clave para comprender el modelo artificial.

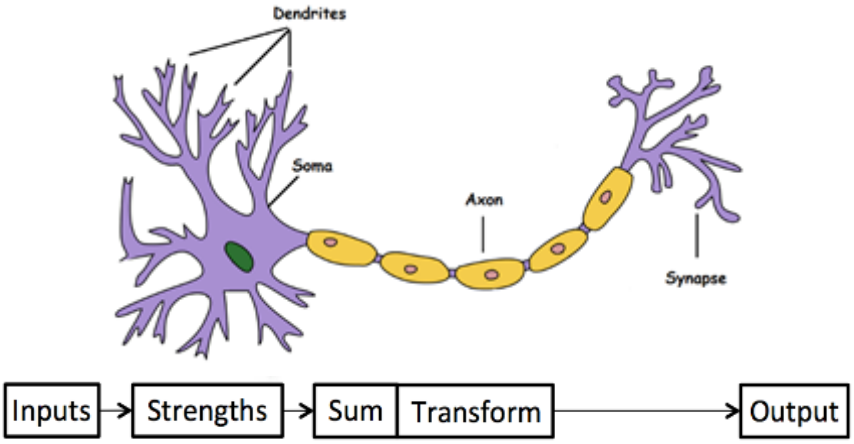


Figura 2.1 Estructura de una neurona biológica

2.3.1 El Aprendizaje

Pocos serían los avances generados por el ser humano sin su capacidad de aprender nuevas habilidades, comprender nuevos conceptos y hacer nuevas relaciones entre los conocimientos ya adquiridos. Durante el desarrollo de un ser vivo, el cerebro se modela, es decir que la configuración del cerebro no es fija, por el contrario, se modifica según la creación o incluso destrucción de nuevas y antiguas conexiones. También en el proceso de sinapsis no es necesario poseer una intensidad que deba permanecer intacta con el tiempo, esta puede cambiar dada la información que llega a ella. A este fenómeno se le denomina plasticidad cerebral, y constituye en gran medida el aprendizaje.

Estos principios de acción, principalmente la variación en la intensidad sináptica, son en los que se basan los sistemas neuronales artificiales para aprender.

2.4 Neuronas artificiales

Hoy poco más de medio siglo después a los avances posguerra, hemos desarrollado computadoras miles de veces más rápidas, un aplastante ejemplo sería la supercomputadora “Summit” creada por el Laboratorio Nacional Oak Ridge, situado en Tennessee, Estados Unidos y desarrollada por IBM [7][8]. Summit puede hacer 200 cuatrillones de cálculos en un segundo, para entender mejor esto supongamos que si una persona es capaz de hacer un cálculo por segundo, le tomaría 6.300 millones de años calcular lo que esta computadora logra en un instante. Es así que las computadoras se han transformado en un pilar fundamental de nuestra civilización y vida diaria, estas máquinas construidas por el hombre tienen una capacidad increíble, sin embargo, pese al gran desarrollo de la computación y la electrónica, existen tareas que para ellas representan grandes retos, en especial tareas cuya respuesta no tenga una solución tradicional, es decir, basada en lógica secuencial, tal como reconocer un rostro u objeto. Sin embargo,

sistemas de incluso menor complejidad que el cerebro humano son capaces de lograrlo sin mayor problema.

Para comprender esto es necesario recordar que cuando hablamos del núcleo o cerebro de una computadora es necesario recordar que hacemos alusión a una máquina basada en una arquitectura de “von Neumann”, donde un procesador ejecuta en secuencia programas almacenados en memoria, por otro lado, el cerebro humano se constituye por millones de neuronas, estás millones de unidades se apilan organizadas en capas, que dará como resultado un sistema funcional mayor. Para un sistema neuronal artificial se intenta imitar esta organización.

En 1943 Warren S. McCulloch y Walter Pitts (R. Prieto, sf) proponen el que es considerado como el primer modelo neuronal moderno (ver Figura 2.2), un pilar fundamental en el campo de las redes neuronales. Está concebido como un dispositivo lógico, un modelo cuyo funcionamiento recae exclusivamente en el área de la lógica. Surge como un intento por explicar el comportamiento del sistema nervioso, un homólogo de su conducta expresado de forma matemática.

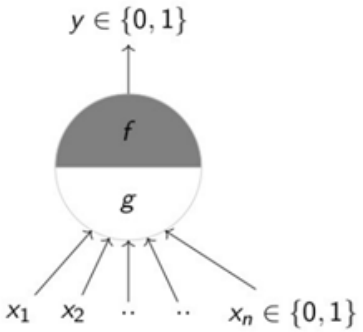


Figura 2.2. Modelo simplificado de Warren S. McCulloch y Walter Pitts

Dividido en dos partes, la primera parte llamada g, deja entrar diferentes valores a la función, expresado en términos de ‘x’ y la parte ‘f’ que toma una decisión basada en el valor agregado. Las entradas y salidas son binarias, tal que solo pueden tomar el valor de 0 o 1. Las entradas pueden ser excitadoras o inhibidoras, estas etiquetas corresponden a la capacidad que tiene la entrada para afectar el resultado. Las inhibidoras son aquellas que tienen el máximo efecto sobre la salida, prescindiendo a veces del valor de las otras entradas. Las excitadoras por sí mismas no pueden encender la neurona, pero es posible que lo hagan al combinarlas (Khapra, 2018). Formalmente se expresa:

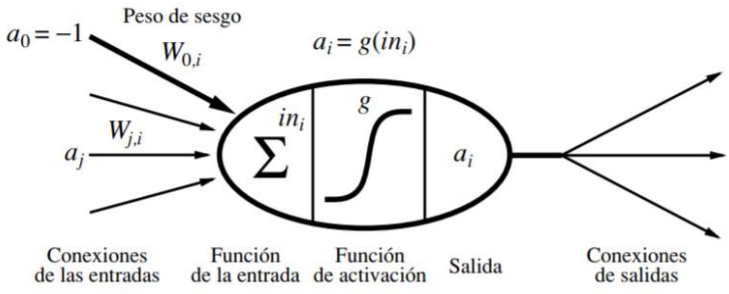
$$g(x1, x2, x3, \ldots, xn) = g(x) = \sum_{i=1}^{n} xi$$

$$y = f\big(g(x)\big) = 1 \; if \; g(x) \geq \theta$$
$$y = f\big(g(x)\big) = 0 \; if \; g(x) < \theta$$

Donde theta θ es el umbral de activación, por ejemplo: si la neurona se activa cuando g(x) es mayor o igual a 7, entonces theta es igual a 7 y g(x) como la sumatoria de las entradas xi. Una sola neurona de este tipo es capaz de representar funciones booleanas.

2.4.1 El Perceptrón

En 1958 el psicólogo americano Frank Rosenblatt propone el modelo actual del perceptrón, el cual evolucionó a futuro analizado por Minsky and Papert, quienes en 1969 presentan lo que ahora conocemos como modelo clásico. Pese a que el trabajo de McCulloch-Pitts se considera la base del perceptrón, este es un modelo computacional mucho más general, ya que las entradas no están limitadas a valores booleanos, e introduce una manera de medir la importancia que tiene la entrada con respecto a las demás, así como un mecanismo matemático para aprender esos valores, dando un siguiente paso en lo que antes se consideraban neuronas inhibidoras y excitadoras, llamadas pesos (ver Figura 2.3). A continuación, se describen con detalle sus elementos:



$$a_i = g(\sum_j W_{j,i} a_j)$$

Figura 2.3. Modelo de una neurona artificial

- **Entradas y salidas**

Compuesto por un vector a = (a_1,...,a_n) que representan los valores para alimentar el modelo. Como se mencionó antes las entradas pueden ser valores no sólo binarios, esto permite al modelo adaptarse mejor a diferentes problemas.

- **Pesos sinápticos**

Vector de pesos $W = (W_1,...,W_n)$ equivalente a las conexiones sinápticas, donde W_{ji} representa la intensidad de interacción presináptica *j* y la neurona postsináptica *i*. Aquí definimos el umbral o peso de sesgo como $W_{0,1}$ como entrada constante $a0 = -1$, este umbral permite la activación (cercano a +1) cuando la suma de los pesos de las entradas $\sum(Wj, i\ aj)$ esté por encima del valor del sesgo, e inactiva (cercano a 0) cuando sean erróneas las entradas.

- **Regla de propagación**

Potencial postsináptico $ni(t) = \sum(Wj, i\ aj)$ de la neurona *i*, se comprende como la suma de los productos en función de sus entradas y pesos.

- **Función de activación**

Genera el estado de activación de la neurona *i*, esto en función de su estado anterior $ai(t-1)$y del postsináptico actual. Es decir:

$$ai = g(ni) = g(\sum(Wj, i\ aj))$$

La función de activación permite no solo trabajar con problemas de regresión lineal, es decir no solo con líneas rectas, distorsionando estas para tomar formas más complejas, en otro caso la red neuronal colapsara completamente en una sencilla función lineal, agregar una NO-linealidad en la ecuación, permite al modelo adaptarse mejor a los datos mostrados, aproximando con mayor precisión el ai y discretizando el valor de la predicción.

La Figura 2.3 muestra como ejemplo el uso de la función sigmoide, esto genera un valor en un rango entre [0, +1]. Si tuviéramos una sola neurona de salida (para unidades sigmoides), el porcentaje por encima de la 0.5 corresponde a una clase y por debajo a 0.5 como otra clase. También posee la propiedad de ser diferenciable, lo cual permitirá que aprenda a través del algoritmo de aprendizaje de los pesos. Las redes neuronales poseen un conjunto de funciones usadas comúnmente, mostradas en la Tabla 2.1

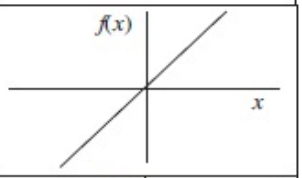
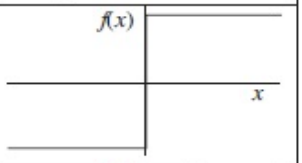
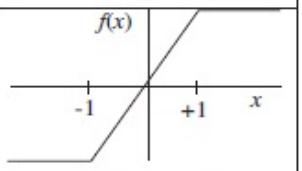
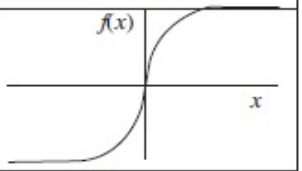
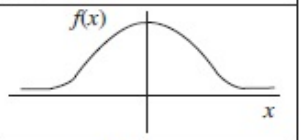
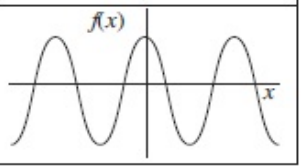
	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty,+\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1,+1\}$ $\{0,+1\}$	
Lineal a tramos	$y = \begin{cases} -1, si\ x < -l \\ x, si\ +l \le x \le -l \\ +1, si\ x > +l \end{cases}$	$[-1,+1]$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0,+1]$ $[-1,+1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0,+1]$	
Sinusidal	$y = A\ sen(\omega x + \varphi)$	$[-1,+1]$	

Tabla 2.1 Funciones de activación comunes en las RNA

2.5 Estructuras neuronales artificiales

Al hablar de estructura de redes neuronales se clasifican principalmente en:

- Redes feedforward: Estas redes (con alimentación-hacia-adelante) representan sus salidas únicamente en función de sus entradas actuales, es decir que no posee algún otro estado que afecte su comportamiento más que el de sus propios pesos.

- Redes feedback: En estas redes también llamadas recurrentes se permite que sus salidas alimenten las propias entradas. En este concepto entra la idea de memoria a corto plazo, ya que la red debe almacenar estados previos.

Las redes recurrentes por lo general presentan cierta inestabilidad y una mayor complejidad ya que conforman un tipo de sistema dinámico que debe alcanzar un estado estable. (ver Figura 2.4-b)

Las redes con alimentación-hacia-adelante comúnmente se ordenan en capas, cada capa contiene un número variable de neuronas (ver Figura 2.4-a), es necesario observar que las neuronas de la capa de entrada funcionan solo dejando pasar el valor de la entrada, a su vez la capa de salida funciona expresa resultados, por lo tanto, aquellas neuronas encapsuladas dentro de la capa oculta se comportan como se mencionó anteriormente. Una red neuronal es útil para resolver problemas de regresión, así como de clasificación por lo cual comúnmente el número de neuronas de salida corresponde al número *n* de clases que deseamos clasificar.

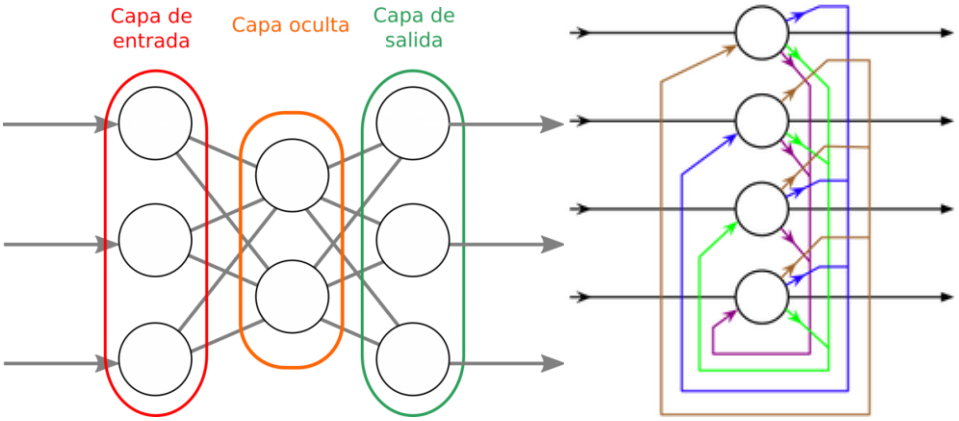


Figura 2.4 a) Red FeedForward. b) Red feedback

Para la organización interna de las neuronas de una red es necesario elegir una topología, es decir, determinar una serie de parámetros para nuestro modelo que constituyen su arquitectura, tales como número de capas ocultas, número de neuronas por capa, así como el tipo y nivel de conexión entre neuronas. Si la red está compuesta por dos o más capas recibe el nombre de multicapa.

2.6 Redes neuronales multicapa

El perceptrón multicapa (ver Figura 2.5), corresponde a la idea de la agrupación de unidades neuronales en múltiples capas, esto significa aumentar el grado de complejidad y de abstracción que el sistema puede realizar. Cada capa oculta contiene un conjunto de neuronas, de las cuales, cada una está fuertemente conectada a cada neurona de la capa anterior, la última capa es llamada de salida, la cual efectúa la representación de las probabilidades de las clases.

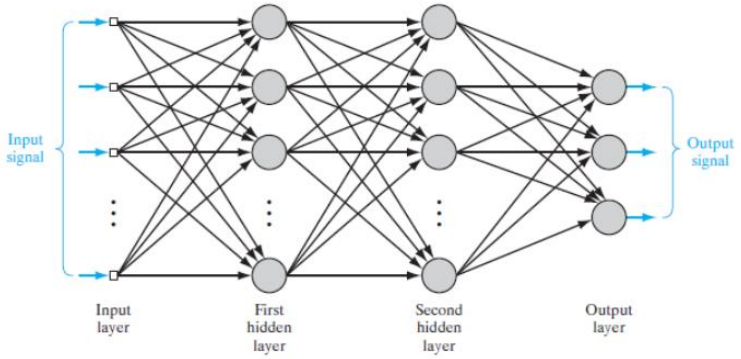
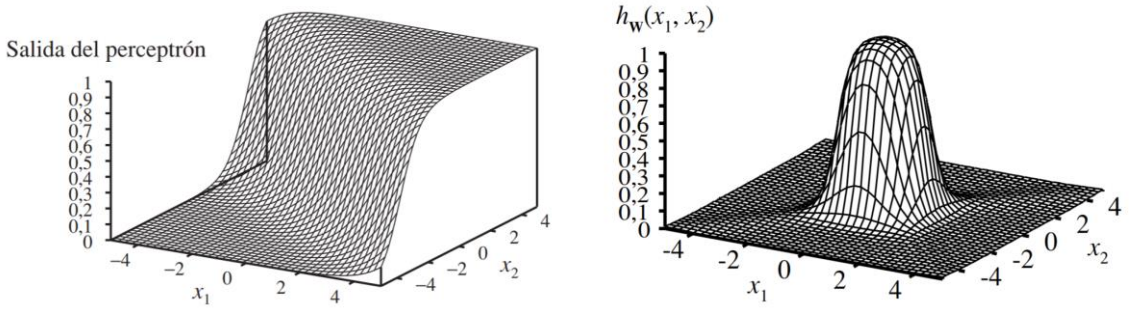


Figura 2.5. Perceptrón multicapa

Cada unidad por capa representa una función suave en el espacio de sus entradas (ver Figura 2.6-a), es así que la salida del perceptrón multicapa podría verse entonces como una combinación de varias gráficas de este tipo en su propio espacio(ver Figura 2.6-b), más unidades neuronales significa la creación de más montículos, más formas y tamaños.



a) b)

Figura 2.6 a) Gráfico de la salida del perceptrón de dos unidades de entrada con una función de activación sigmoide. b) Resultado de la combinación de diferentes neuronas creando un montículo

Es así que tenemos un espacio vectorial que crece en complejidad, cuya forma se adapta cambiando su forma mientras intenta acercarse a los valores que le hemos dado. Como un mapa que se extiende, con múltiples elevaciones y hundimientos en el terreno.

2.7 Aprender y recordar lo aprendido

Así como la intensidad sináptica puede ajustarse a lo largo del tiempo para aprender mejor un concepto o nueva habilidad en nuestro cerebro, las redes neuronales ajustan los valores internos de sus pesos, con el fin de aprender mejores relaciones de los datos que recibe. El proceso de evaluar y optimizar estos valores, es decir, aumentar el desempeño de una red, debe pasar por una fase de entrenamiento o aprendizaje.

El sistema inicia sus valores de pesos y sesgos en estado nulo o aleatorio. Al generar una predicción notaremos que dicho valor, como es de esperar resulta aleatorio, que, al

comparar con el valor verdadero o esperado de nuestros datos, se generará un error o costo, este error simboliza la distancia entre lo que predice el modelo y lo que esperamos que prediga, es decir permite medir qué tan cerca o lejos estamos de valores que sean de utilidad. Al cambiar el valor de los parámetros cambia el error, para hallar los valores correctos y disminuir el error en dicha relación, partimos de la salida generada. Lo que deseamos es extender ese error en dirección contraria a la salida, en otras palabras, hacia las primeras capas. Esto debido a que el error de las capas anteriores depende directamente de las capas posteriores. Esto se logra calculando la derivada parcial del costo con respecto a la derivada parcial de los pesos y sesgos de cada neurona por capa, el vector resultante que recolecta contiene las pendientes para cada una de las dimensiones del modelo. Sin embargo, calcular estos parámetros puede convertirse en una tarea bastante compleja, ya que mientras la red neuronal aumenta su complejidad y nivel de abstracción (aumentando número de capas y neuronas por capa), también será más difícil determinar cómo pequeños ajustes en los parámetros de las primeras capas afectarán el resultado de las capas posteriores, o visto de otra forma existe una sucesión o encadenamiento en el comportamiento de sus conexiones. Para esto es necesario aplicar un principio de cálculo muy útil para hallar el resultado de funciones compuestas, la regla de la cadena.

2.7.1 Error y descenso por el gradiente

La función de salida determinada por una red multicapa se observa como la combinación de las salidas no lineales de las neuronas en las capas anteriores, tal como el mapa de un valle, se muestra un espacio vectorial (ver Figura 2.7), formado por pesos y sesgos, un terreno cuyas elevaciones representan valores para los cuales el error aumenta, mientras que al descender al abismo el error disminuye.

Una práctica común como se dijo antes es declarar nuestros parámetros que estamos buscando como vectores aleatorios, entonces al comenzar el entrenamiento de una red neuronal se toma un lugar arbitrario en el terreno, desde el cual comenzaremos el descenso, para saber qué dirección seguir, desde el punto de partida, se calculan las derivadas parciales de cada parámetro con respecto al costo, el conjunto de estas derivadas conforma un vector llamado vector gradiente. Que en análisis vectorial nos indica el ritmo de variación de un vector y la dirección de dicho vector [9]. Una mayor inclinación indica la dirección en la cual moverse dando pequeños pasos para el descenso, a cada paso se repite el proceso, esta técnica se conoce como Descenso del Gradiente. La meta en este enfoque es igualar la pendiente $f'(x) = 0$, y hallar algo conocido como mínimos globales, o los puntos de mayor profundidad, no obstante así como en la superficie de un terreno con múltiples crestas, es posible llegar a puntos donde no sea posible seguir descendiendo, y al mismo tiempo resulte en una derivada $f'(x) = 0$, sin ser necesariamente el mínimo global de la función, esta característica puede suceder incluso múltiples veces en un modelo, a estos puntos se conocen como mínimos locales.

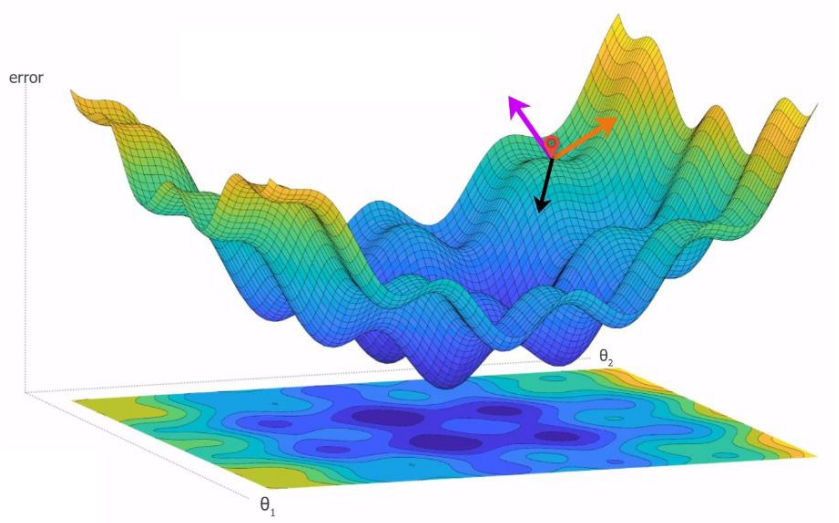


Figura 2.7 Representación tridimensional de la función error usando dos parámetros, en ML pueden existir modelos en función de miles de parámetros

2.7.2 Backpropagation

El cálculo de una masiva cantidad de derivadas parciales implica un método capaz de hacerlo independientemente de la profundidad de la red, incluyendo sistemas con miles o millones de neuronas. En 1974 Paul Werbos[10] describe en su tesis por primera vez el proceso de entrenamiento de una red neuronal.

El backpropagation o retro propagación-hacia-atrás es un algoritmo creado para calcular el vector gradiente, a través de ciclos de aprendizaje. Este proceso se puede dividir en:

1. Calcular el error de la última capa
2. Retro propagar el error a la capa anterior
3. Calcular las derivadas de la capa usando el error

Cuando se realiza un ciclo de aprendizaje es necesario tomar solo una fracción de los datos (lotes), este patrón se aplica a la entrada para estimular el modelo, el cual se propaga hacia adelante por la red y genera una salida, dicha señal debe ser comparada con la salida deseada, al contrastar ambos valores, estamos obteniendo el error. Usualmente para calcular esta comparación se usa el error cuadrático medio.

En la retro propagación de errores, al partir de este valor es posible saber en qué forma las decisiones y ponderaciones neuronales aportan más que otras al error, una vez calculado, retrocede de manera recursiva capa tras capa moviendo el error hacia atrás, influyendo solamente en aquellas neuronas de la capa previa que contribuyen al error. Al llegar a la primera capa habremos obtenido cuál es el error para cada neurona y para cada uno de sus parámetros, así como su contribución parcial al error total.

Para el cálculo de funciones compuestas es necesario usar la regla de la cadena, la cual dice que, para encontrar la derivada de una composición de funciones, es necesario multiplicar cada una de sus derivadas intermedias.

2.8 Redes neuronales convolucionales

La tarea de detectar objetos comprende la capacidad de un sistema para otorgar información referente al entendimiento semántico en imágenes o video. El desarrollo de algoritmos neuronales relacionados a sistemas de aprendizaje ha permeado fuertemente en las técnicas para detección de objetos.

Aquí partimos de la tarea de tomar una imagen y posteriormente dar como resultado la clase que describe mejor la imagen. Para los humanos esta es una tarea relativamente sencilla la cual aprendemos casi desde el nacimiento, esta capacidad nos permite identificar de manera inmediata nuestro entorno incluso de forma subconsciente, también una característica importante es que recordamos cosas que hemos visto antes. Entonces, podríamos decir, que estas capacidades de hacer un veloz reconocimiento del entorno, generalizar a partir de conocimientos previos y la adaptación en los pequeños o grandes cambios entre los entornos (invariancia espacial); es justo la capacidad que buscamos compartir con nuestras máquinas.

Como se ha explicado, las redes neuronales artificiales son fuertemente inspiradas en su equivalente biológico, lo que se intenta es simular el comportamiento orgánico del sistema nervioso, a partir de la abstracción matemática de estudios neurocientíficos. Una variante del perceptrón multicapa, conocidas como redes neuronales convolucionales tienen su origen en este paradigma. En 1981 los neurocientíficos David Hubel and Torsten Wiesel, ganaron el premio nobel, debido a sus contribuciones explicando la forma en que funciona la visión en los mamíferos, como perciben el mundo alrededor de ellos usando una arquitectura jerárquica reuniendo neuronas por capas, cada uno de estos grupos representan diferentes características que son aprendidas. Su hipótesis plantea que dentro de la corteza visual las respuestas funcionales complejas son generadas por “celdas complejas” construidas por “celdas simples”. Por ejemplo, es posible que las celdas simples sean activadas al detectar líneas, mientras que las celdas complejas son estimuladas al detectar formas o patrones con cierto grado de varianza espacial. Las celdas están constituidas por campos receptivos, encargados de responder a la adición de entradas de otras celdas locales. Si bien la mayoría de nosotros somos capaces de reconocer un rostro específico en diferentes situaciones, es porque aprendemos constantemente a abstraer tales condiciones sin que se altere la percepción que tenemos del objeto, tal como su: escala, iluminación, orientación, contraste, rotación.

Las CNN [11] pueden dividirse principalmente en dos partes (ver Figura 2.8), la parte de aprendizaje de características y la parte de clasificación. La primera parte está compuesta por bloques de operaciones, cada bloque corresponde a una secuencia básica, que se repite una y otra vez. La segunda parte corresponde a la distribución de la salida como un pequeño vector dimensional, en esta parte se enlaza una capa fuertemente conectada, lo

que se busca es comprimir un conjunto de valores probabilísticos, que corresponden a una clase específica, al coger el valor máximo, se obtiene el porcentaje de acierto al predecir la clase del objeto mostrado.

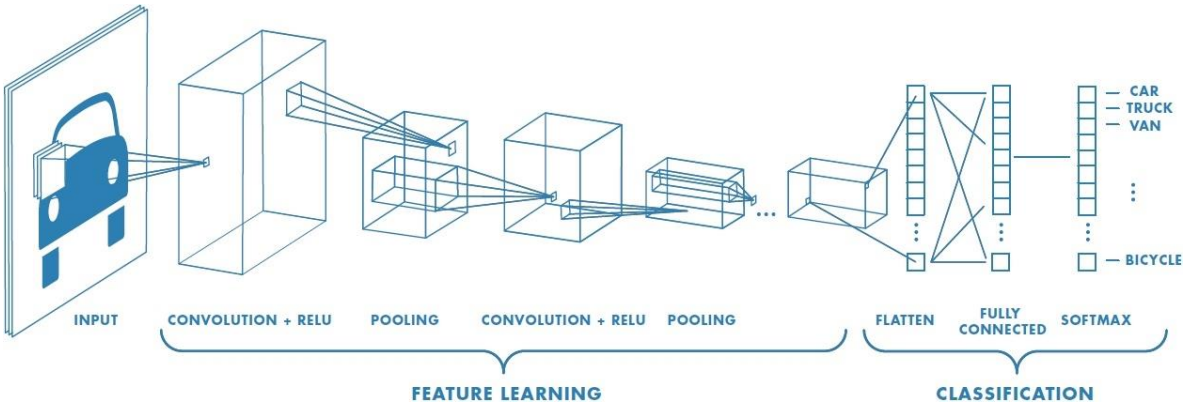


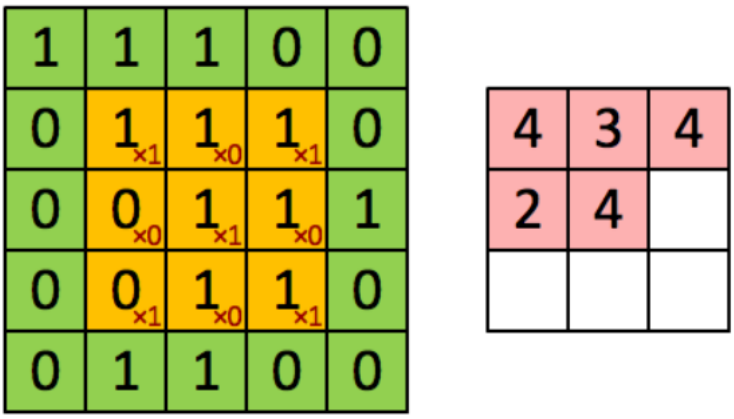
Figura 2.8 Arquitectura general de una CNN

Para comenzar es necesario preparar un conjunto de imágenes, una imagen a color se compone de tres canales según el estándar RGB, cada canal contiene el valor para un tono de azul, verde y rojo, de 0 a 255 en un píxel de la imagen, si miramos a la imagen descompuesta en esta forma, tendremos un vector tridimensional, donde su matriz de largo por ancho representa sus primeras dos dimensiones, y su profundidad RGB la tercera. Si la imagen está en escala de grises, la profundidad será igual a uno. Este vector funciona como entrada al primer bloque de convolución, cada bloque está formado por tres operaciones básicas: convolución, función de activación y pooling. Toda imagen debe estar acompañada de su correspondiente etiqueta.

Similar a los campos receptivos en la corteza visual, la imagen es analizada por segmentos, escaneando la figura usando pequeños recuadros o ventanas (ver Figura 2.9), desplazándose un pixel a la vez, tal ventana contiene la matriz de pesos, aquí se hace evidente una de sus principales diferencias al perceptrón multicapa, donde cada neurona se encuentra fuertemente conectada a cada neurona de la siguiente capa, así mismo el conectar en ese modo generaría muy elevados tiempos de cómputo.

La convolución es una operación matemática usada en casi todo campo de la ingeniería, consiste en un proceso ordenado de entrelazado de fuentes de información, en este caso los valores de las características contenidas en la imagen y la ventana o matriz de pesos. Para combinarlas es necesario sumar los términos del producto punto entre la matriz de la imagen y la matriz de pesos, recorriendo la imagen de izquierda a derecha y de arriba a abajo un valor a la vez (dicho valor se denomina stride), hasta completarla. El proceso se realiza para todos los canales de color. Cada vez que la matriz de pesos se desplaza, repite el proceso, la nueva matriz o mapa de activación indica las regiones de activación, es decir, regiones que serán activadas cuando características específicas al kernel sean detectadas en la entrada, y su vez reduce la complejidad computacional de la imagen. Durante el proceso de entrenamiento los valores en la matriz de pesos cambian, lo que indica que la

red está aprendiendo a identificar qué regiones son importantes para extraer características de los datos.



a) b)

Figura 2.9 Operación convolución, a) matriz de imagen y ventana o kernel de convolución, b) Mapa de activación o matriz resultante.

Antes de continuar al siguiente bloque, se aplica otra operación conocida como pooling, la cual reduce las dimensiones espaciales (ancho x alto) del volumen de entrada para la siguiente capa convolucional. Este proceso no afecta a la dimensión de profundidad del volumen. La operación se realiza tomando el valor máximo de los valores observables en la ventana (max pooling) (ver Figura 2.10), o tomando el promedio de los valores.

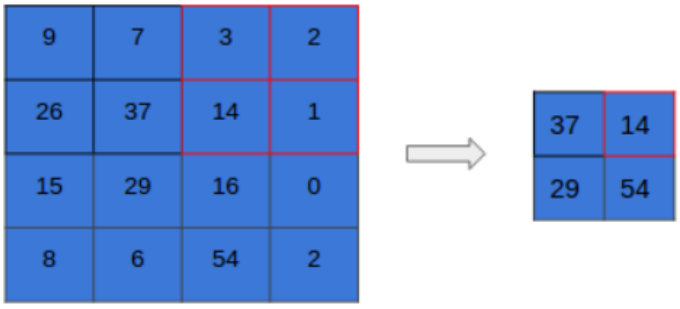


Figura 2.10 Max Pooling

Una vez terminado es necesario normalizar la imagen, lo cual evita que las matemáticas se rompan al convertir todos los números negativos a 0, para esto es necesario la aplicación de nuestra función de activación, el uso de la función RELU (unidad de línea rectificada) se encuentra entre las más extendidas para este tipo de problemas, debido a que solo permite la activación cuando los valores del píxel se encuentran por encima del umbral 0, es así que una pila de imágenes se convierte en una pila de imágenes sin valores negativos. El orden de estas operaciones se repite en cada capa con imágenes más pequeñas [12].

Una vez terminada la fase de extracción de características, es necesario convertir el conjunto de valores en datos que podamos clasificar, lo que se pretende entonces es crear un vector de probabilidades a partir de la salida. Para esto suele usarse la función Softmax para calcular el error, el resultado de esta función será un vector de valores discretos para cada clase que se intenta predecir. Al obtener el argumento máximo en el vector se obtiene la clase del objeto. El proceso de aprendizaje se lleva a cabo usando el algoritmo de backpropagation.

2.9 Transferencia de aprendizaje

La transferencia de aprendizaje o *transfer learning* es un método en *ML* donde un modelo desarrollado para algún tipo de tarea específicas, es reutilizado como punto de partida para un modelo que realizará una tarea secundaria. Se relaciona comúnmente a problemas de aprendizaje multitareas, cabe mencionar que su uso no está limitado exclusivamente a campos de estudio relacionados con el aprendizaje profundo. Este se considera un enfoque popular ya que es posible dar uso a modelos previamente entrenados, los cuales comúnmente son usados en tareas de procesamiento del lenguaje natural y visión por computadora, logrando así economizar los vastos recursos necesarios para entrenar estos modelos, tal como tiempos de cómputo y obtención de datos.

Según [13] *"En el aprendizaje por transferencia, primero entrenamos una red neuronal como base, en un conjunto de datos y tarea, y luego reutilizamos las funciones aprendidas, o las transferimos, a una segunda red de destino para recibir capacitación en un conjunto de datos y una tarea objetivo. Este proceso tenderá a funcionar si las características son generales, lo que significa que son adecuadas para las tareas base y de destino, en lugar de ser específicas para la tarea base*."

Esta descripción en la forma del aprendizaje por transferencia se denomina transferencia inductiva, es aquí donde es posible aumentar el alcance de distintos modelos para nuestro beneficio al utilizar un ajuste del mismo en una tarea diferente pero relacionada.

Para usar la transferencia de aprendizaje en nuestros modelos propios, existen dos enfoques que debemos seguir:

1. Enfoque de desarrollo del modelo
2. Enfoque de modelos pre-entrenados

Enfoque de desarrollo del modelo

- **Seleccionar fuente de tareas:** El desarrollador debe seleccionar un problema de predicción relacionado con abundantes datos, donde existen algunas relaciones entre los datos de entrada, los datos de salida, y/o conceptos aprendidos durante la asignación de datos de entrada a salida.

- **Desarrollar un modelo de fuente:** El modelo debe responder a un hábil desarrollo para la primera tarea

- **Rehusar modelo:** El ajuste del modelo en la tarea de origen se puede utilizar como punto de partida para un modelo en la segunda tarea de interés.

- **Modelo de sintonía:** Opcionalmente el modelo puede necesitar ser adaptado o refinado en los datos de pares de entrada-salida disponibles para la tarea de interés.

Enfoque de modelos pre-entrenados

- **Seleccionar modelo fuente:** Un modelo pre-entrenado es seleccionado de un conjunto de modelos disponibles. Muchas instituciones de investigación publican modelos entrenados y retados por robustos datasets, que pueden incluirse en el conjunto de modelos candidatos entre los que elegir.

- **Reutilizar modelo:** El modelo pre-entrenado se puede usar como punto de partida para un modelo en la segunda tarea de interés. Esto puede implicar el uso de todo o parte del modelo, dependiendo de la técnica de modelado utilizada.

- **Modelo de sintonía:** Opcionalmente, el modelo puede necesitar ser adaptado o refinado en los datos de pares de entrada-salida disponibles para la tarea de interés.

2.10 Estado del Arte

En la Universidad ESPE se realizó un proyecto de investigación para el reconocimiento de imágenes en frames de video utilizando redes neuronales, para esto se utilizó un clasificador de imágenes utilizando CNN (Convolutional neural network) para entrenar un clasificador cuyas imágenes a ser reconocidas son una secuencia de cuadros de video obtenido a través de una cámara de video vigilancia. El aporte del proyecto es la clasificación de imágenes (personas, vehículos) desde los frames de video que generalmente tienen baja calidad en definición, iluminación, distancia de objetos, etc., finalmente se obtuvo el clasificador con redes neuronales que permite reconocer y clasificar personas y autos en videos. [14]

En Centro Universitario UAEM Valle de México se realizó un trabajo para la detección de peatones y vehículos sobre FPGA donde se hizo la implementación del paradigma Deep Learning con una CNN para clasificar imágenes de señales de tránsito vehicular, con el propósito de medir el tiempo de entrenamiento y su desempeño en la clasificación. Los resultados obtenidos en la investigación son: La programación, entrenamiento y prueba de una CNN. Se realizaron una serie de experimentos, encontrando un error en la clasificación de 3.25% y un tiempo de entrenamiento de 0.33 horas, para los mejores casos de los ensayos realizados.[15]

En la Universidad de Sevilla, en el departamento de ingeniería de sistemas y automática se desarrolló un algoritmo en el lenguaje de programación R utilizando redes neuronales que, tras ser entrenado con una extensa base de datos, fue capaz de reconocer caracteres escritos a mano, específicamente se usaron números del 0 al 9. El objetivo principal fue el de conseguir, al menos, un porcentaje superior al 95%, con un conjunto de 10000 imágenes de comprobación, para esto fue parte fundamental el uso de las redes neuronales convolucionales.[16]

En la Universidad de François Rabelais, se desarrolló una aplicación para el reconocimiento de objetos usando el reconocedor de marcadores para poder identificar los elementos que se enfoquen mediante la cámara de una tableta y esta se encarga de dar una descripción acerca del objeto. Ese estudio abre la posibilidad de que la posibilidad de mejorar el aplicativo que contenga un mejor procesamiento de imágenes y métodos de aprendizaje automático. También que sea usado para reconocer en tiempo real. [17]

En la universidad de Firenze, Italia, desarrolló un proyecto con el objetivo de realizar la clasificación de objetos en tiempo real y el reconocimiento de ilustraciones usando un dispositivo portátil, para mejorar la experiencia del usuario durante una visita al museo proporcionando información contextual y realizando perfiles de usuario. Se propuso el uso de una red CNN compacta que realiza la clasificación de objetos y la localización de ilustraciones y, usando las mismas características de CNN, con el propósito de un reconocimiento robusto de las ilustraciones. [18]

Un consorcio de investigadores europeos coordinado por el Centro de Visión por Computador (CVC) de la Universidad Autónoma de Barcelona (UAB) ha desarrollado 'Hermes', un sistema integrado por cámaras de vídeo y software que es capaz de reconocer y prever el comportamiento humano, así como de describirlo en lenguaje natural. 'Hermes' (siglas de Human Expressive Representations of Motion and their Evaluation in Sequences) se basa en el análisis del comportamiento humano a partir de secuencias de vídeo capturadas con tres niveles de enfoque: el del individuo, como un objeto relativamente alejado; el de su cuerpo, con una proximidad media que permite analizar sus posturas; y el del rostro, que permite estudiar con detalle sus expresiones faciales. La información obtenida, procesada por algoritmos de visión por computador y de inteligencia artificial, permite al sistema aprender y reconocer patrones de movimiento. [19]

En 2016 investigadores del reino unido publicaron un estudio en el cual utilizaban CNN’s, para la detección de retinopatía diabética, la cual aparece como daño en los vasos sanguíneos del tejido ubicado en la parte posterior del ojo. Como resultado, en el entrenamiento final de la red, se logró un 95% de especificidad, 75% de precisión y 30% de sensibilidad. [20]

3. Desarrollo del Framework

En el capítulo 2 se estudió el conocimiento básico teórico de las redes neuronales. Este capítulo describe los requerimientos y propuesta de software, un sistema o framework end-to-end (extremo-a-extremo), que facilite a los desarrolladores completar el flujo de trabajo que representa entrenar estos modelos, de forma rápida, abierta y sin necesariamente un profundo conocimiento de programación.

3.1 Metodología

El ciclo de vida del proyecto se ha estimado pensando primero en la adaptabilidad del mismo antes que a un estricto tiempo, exigiendo adaptarse a los cambios sobre la marcha debido a su naturaleza y motivación de unir múltiples tecnologías. Lo anterior mencionado hace a la metodología de programación extrema o eXtreme Programming (XP) la mejor opción, ya que esta metodología se adapta mejor a proyectos de corto plazo, y a los cambios de requisitos.

Antes de iniciar el proyecto es necesario definir las historias de usuario, como base para la implementación, y para estimar su cumplimiento basado en estas premisas. Estos conceptos expresan en términos simples la funcionalidad del sistema. Se describen a continuación:

- El usuario debe ser capaz de etiquetar los objetos que aparecen en un conjunto de imágenes, guardarlos y procesarlos para su uso.

- El usuario debe poder elegir un modelo adecuado a sus necesidades y entrenarlo con los datos previamente creados.

- Los resultados del entrenamiento deben poder visualizarse en forma gráfica.

- Al finalizar el entrenamiento el modelo debe poder probar su eficacia, para esto el usuario podrá seleccionar una imagen o video en la cual hacer inferencia.

Definidas las historias, se procede a la fase de codificación, primero es indispensable el diseño de una interfaz gráfica para la comunicación del usuario con el sistema, buscando hacer diseños sencillos y rápidos para que implementar su funcionalidad sea una tarea relativamente fácil. En cada iteración se mejora y modifica el código ya creado sin alterar su funcionalidad previa (refactorizar). La primera versión de este sistema antepone su funcionalidad a la experiencia de usuario.

3.2 Entornos de trabajo para redes neuronales

En el ecosistema de internet existen muchos entornos o herramientas de desarrollo que permiten programar modelos de redes neuronales, además de otros modelos matemáticos. Por ejemplo [21]:

- ❏ Tensorflow: Framework de código abierto creado por Google.
- ❏ CNKT: Soportado por Microsoft
- ❏ Torch: Framework creado en lua con soporte para algoritmos de machine learning
- ❏ PyTorch: Versión de torch en Python, soportado por Facebook, lanzado en 2017.
- ❏ Caffe: Permite la implementación de modelos en matlab para c y c++. Desarrollado por la universidad de Berkeley
- ❏ Caffe2: Sucesor de caffe soportado por Facebook

Para este trabajo se decidió el usar Tensorflow, debido a que el proyecto contiene una colección de múltiples modelos implementados por investigadores, estos modelos reciben mantenimiento de sus respectivos autores. Entre estos se encuentra su API para detección de objetos, este proyecto está construido por encima de Tensorflow, lo cual hace fácil construir, entrenar y desplegar modelos. Tensorflow cuenta con soporte para python, java, javaScript, Swift y GO, siendo su versión en python de las más estables, entonces haremos uso de este lenguaje. Para el siguiente trabajo ha sido seleccionada la API en su versión 1.12.1, se puede encontrar en [22].

3.2.1 Configuración del entorno

Tensorflow posee compatibilidad para IOS, Linux y Windows. Lo que significa obtener los binarios genéricos compatibles con el equipo, una versión de python y utilizarlo directamente. También si el usuario lo desea, puede descargar el código fuente, compilarlo y usarlo.

Debido a la versatilidad existen diferentes configuraciones para instalar bibliotecas de python para crear aplicaciones específicas, es usual crear entornos de trabajo, y moverse entre ellos con relativa facilidad, para esto se usó Anaconda [23], una plataforma de código abierto, para administrar entornos y dependencias de forma rápida en proyectos de python y R. Los detalles de la instalación de dependencias u otras herramientas para el uso de la API se pueden encontrar en [24].

Tensorflow posee dos modos de instalación, configurando el entorno para correr modelos usando los núcleos del procesador (estándar), o usando la instalación compatible con CUDA (GPU), si la tarjeta de video es compatible se habilita para acelerar ciertas tareas. A continuación, se describe la arquitectura de hardware del equipo utilizado:

- ❏ Laptop ASUS(ROG) modelo GL502V
- ❏ Sistema Ubuntu LTS 16.04
- ❏ Memoria RAM de 16 GB

- ❏ Procesador x64 Intel Core i7 CPU 2.81GHz
- ❏ Tarjeta de video NVIDIA GTX 1060
- ❏ Disco duro de estado sólido de 250GB

La lista de GPUs compatibles se encuentra en [25], fue usada esta instalación debido a que el equipo es compatible.

El prototipado de interfaces ha sido generado usando Qt Designer [26], ya que provee conjuntos de elementos clásicos UI para la creación de estilos de escritorio.

3.3 Arquitectura propuesta y programación de fases

Para englobar su correcta ejecución y basado en la descripción en las historias de usuario, este software se divide en 4 fases o etapas:

1. **Fase de preparación de datos:** Esta primera fase comprende el etiquetado, división de los datos en conjuntos de train/test, y conversión de imágenes, esto a un formato de entrada para la fase de entrenamiento.
2. **Fase de selección y configuración del modelo:** En esta fase el usuario podrá descargar un modelo previamente entrenado de una serie de ejemplos, y aumentar su aprendizaje para detectar objetos nuevos. Cada ejemplo se conforma por un archivo de configuración, el cual contiene especificaciones para la fase de entrenamiento, tales como radio de aprendizaje, tamaño del lote, número de épocas, entre otras, el ajuste de estos parámetros es una práctica usual al momento de buscar optimizar el aprendizaje.
3. **Fase de entrenamiento:** Una vez configurado correctamente, es necesario entrenar el modelo, será posible observar su progreso mediante consola, así como detenerlo en cualquier momento. Debido a que los tiempos de entrenamientos pueden resultar muy largos el sistema crea capturas de su comportamiento cada cierto número de iteraciones, así mismo reconoce automáticamente el último guardado para proseguir desde ese punto si elegimos detener el entrenamiento.
4. **Fase de resultados:** En esta fase podrán verse los resultados de su comportamiento de forma gráfica. También será posible exportar el último punto de guardado, para cargar una imagen o video, siendo el resultado de la detección delimitado por figuras rectangulares.

El directorio raíz del sistema se divide en:

- **data**: En esta carpeta se almacenan las imágenes para entrenar el modelo.
- **models**: Guarda los modelos que son descargados por internet.
- **projects**: Cuando se inicie un proyecto nuevo automáticamente se creará una carpeta con su nombre en este folder, así mismo cada carpeta de proyecto almacenará dos carpetas llamadas: labels y training, las cuales contendrán las etiquetas y archivos generados por el entrenamiento respectivamente.

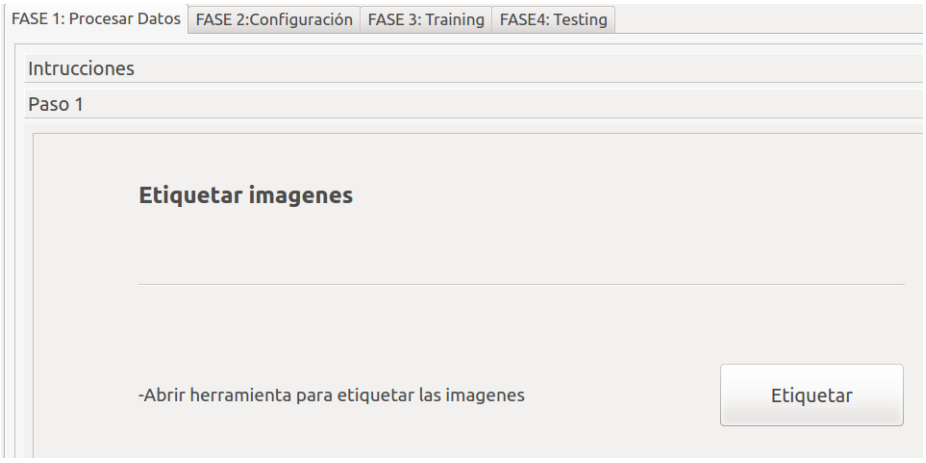
- **src**: Contiene el código correspondiente a la interfaz gráfica, así como scripts para el funcionamiento del programa.
- **tools**: Contiene herramientas adicionales para el framework:
    - TensorFlow Object detection API version 1.12.1
    - labelImg, herramienta de etiquetado.
- **runApp.py**: Programa principal de la aplicación.

Como se mencionó el sistema cuenta con 4 fases, en este apartado se explicará a detalle su funcionamiento y programación.

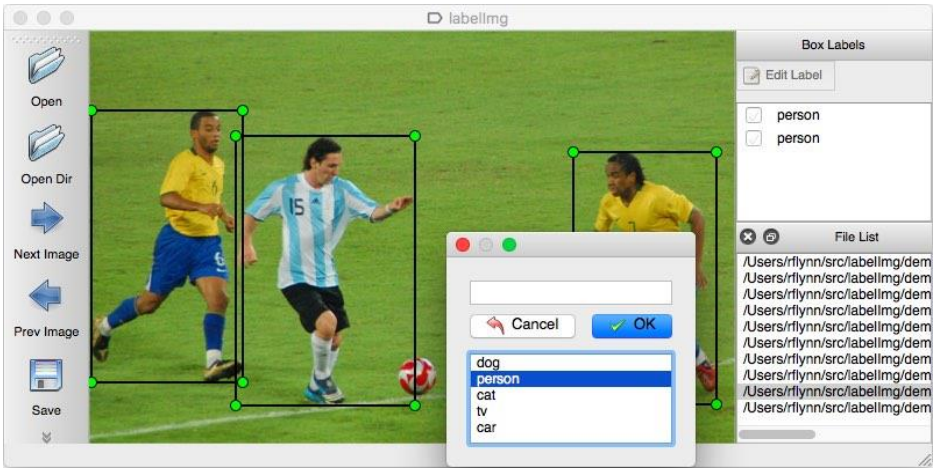
3.3.1 Fase de preparación de datos

En el estudio del machine learning para solucionar un problema es necesario tomar diferentes conjuntos de datos previamente etiquetados para crear conocimiento.
Para agregar funcionalidad a las interfaces, es necesario enlazar el nombre del widget a través de un evento, hacia un script. Esta fase se divide en 2 simples pasos:

1. Etiquetar imágenes: Para etiquetar cada imagen que hemos recolectado, haremos uso de una herramienta de código abierto llamada labelImg (ver Figura 3.1-b), esta nos permite clasificar incluso múltiples instancias de nuestros objetos en una sola imagen, almacenando los datos en formato xml. Para lanzar la aplicación presionar el botón 'Etiquetar'(ver Figura 3.1-a), en la Figura 3.1-c se puede observar la forma de enlazar el nombre del elemento UI a una script, a través del nombre del evento. Es necesario importar *subprocess,* su método *call()* lanza un proceso en consola usando una línea de comando, recibe dos parámetros, un arreglo de separado del comando y la ruta de ejecución.



a)



b)

```
QtCore.QObject.connect(self.ui.ButtonAbrir, QtCore.SIGNAL('clicked()'), self.openNew)
```

```
def openNew(self):
    """
    Abrir labelImg para etiquetar las imagenes
    :return:
    """
    try:
        limg = "framework/tools/labelImg/"
        mydir = os.path.dirname(os.getcwd())
        mydir = str(os.path.join(mydir, limg))

        subprocess.call(["python3", "labelImg.py"], cwd=mydir)

    except Exception as e:
        print(e)
        self.statusBar().showMessage(e)
```

c)

Figura 3.1 a) Fase 1 primer paso. b) Herramienta labelImg para etiquetar imágenes. c) Enlace funcionalidad boton-script

2. Dividir datos: Una práctica común es la segmentación del total de los datos en conjuntos, los cuales serán usados para la fase de entrenamiento y pruebas, en este paso se indica mediante un componente de barra deslizadora el porcentaje destinado a cada conjunto, si no cambiamos este valor, se establece de forma predeterminada 70% para entrenamiento y 30% para pruebas (ver Figura 3.2). Para este paso se requiere como parámetro la ruta donde han sido almacenadas previamente las etiquetas en formato xml, una vez cargada presionamos 'Dividir train/test'. Esto crea un nuevo objeto importado de *split.py* (ruta: /src) y llama a su método split.correr() para dividir los datos (ver Figura 3.3), primero busca y carga en una lista las rutas de todos los ficheros xml en buffer, y posteriormente importar ***from sklearn.model_selection import train_test_split***, esta función recibe tales rutas y retorna dos vectores correspondientes al conjunto de entrenamiento y de pruebas. Por último, se crea una copia de cada xml en su carpeta correspondiente. Si la

división es exitosa se crea (*ruta: /projects/{NOMBRE_DE_PROYECTO}/labels/sets*) en nuestro proyecto y se muestra un mensaje del estado actual en la barra de status.

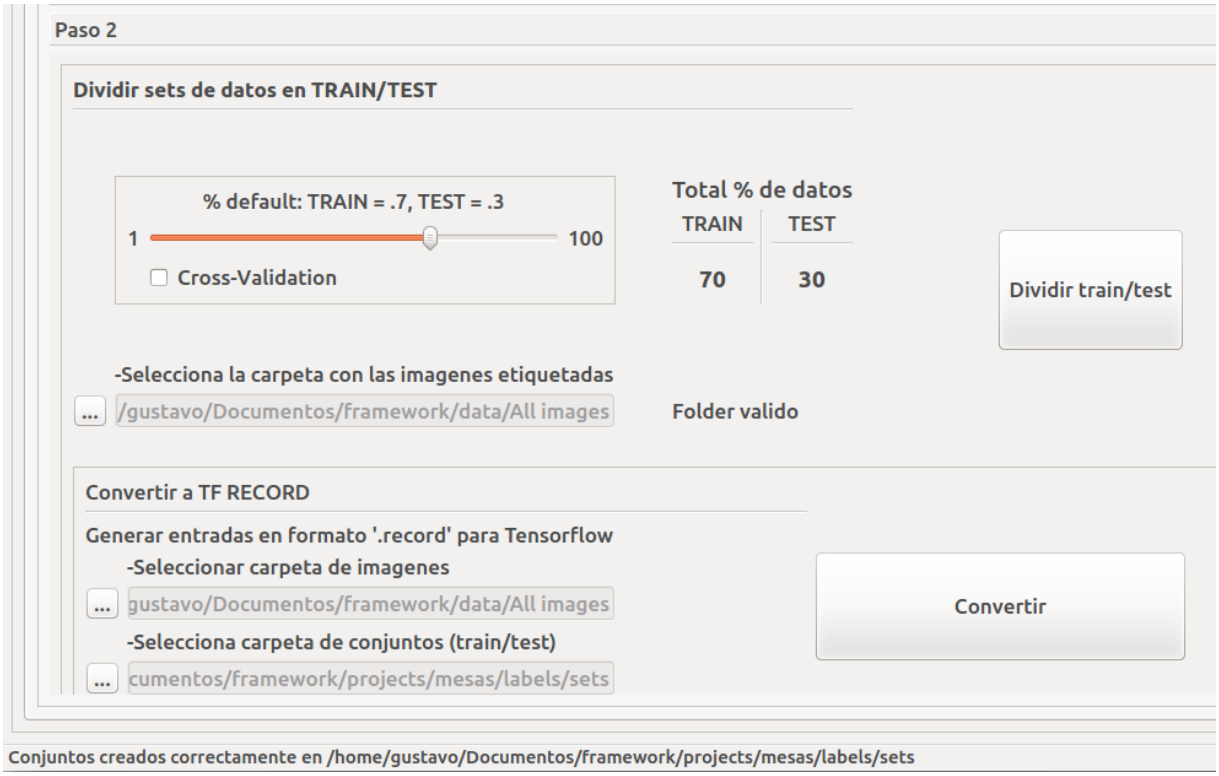


Figura 3.2 Fase 1 segundo paso, division y conversión de entradas

```python
class Split():
    def __init__(self, train, test, dir, dTrain, dTest):
        """
        Constructor
        :param train: valor en % para el conjunto de train
        :param test: valor en % para el conjunto de test
        :param dir: direccion que contiene las imagenes
        :param dTrain: direccion para el conjunto de train
        :param dTest: direccionn para el conjunto de test
        """
        self.train = train
        self.test = test
        self.dir = dir
        self.dTrain = dTrain
        self.dTest = dTest

    def correr(self, lblH):
        """
        Script para correr el proceso de dividir los datos
        :param lblH: array que contiene las etiquetas leidas desde la lista
        :return: None
        """
        try:
            print(self.train)
            print(self.test)
            print("path de imagenes: {}".format(self.dir))

            searcher = buscador.Buscador(path=self.dir, numero_clases=len(lblH), listaClases=lblH)
            searcher.encuentra_conjuntos_entrenamiento_test_desde_path()

            print("Imagenes cargadas!")

            x_dataXML = searcher.X_dataXML
            data_train, data_test = train_test_split(x_dataXML, test_size=self.test, random_state=101, shuffle=True)

            #Limpia la carpeta de train y test para no sobreescribir los datos
            self.limpiarbuffers(self.dTrain, self.dTest)

            #copiar archivo .xml a carpeta 'train'
            for y1 in data_train:
                if not os.path.exists(self.dTrain):
                    os.makedirs(self.dTrain)
                shutil.copy2(y1, self.dTrain)
            #copiar archivo .xml a carpeta 'test'
            for y2 in data_test:
                if not os.path.exists(self.dTest):
                    os.makedirs(self.dTest)
                shutil.copy2(y2, self.dTest)

            print("****** Datos divididos con EXITO ******** ")

        except Exception as ex:
            print(ex)
```

Figura 3.3 clase split para dividir los conjuntos de datos TRAIN/TEST

Una vez separados los conjuntos es necesario la conversión de entradas, este es un proceso en dos pasos intermedios (ver Figura 3.4-a), para que las imágenes de entrada sean de utilidad, primero es necesario la conversión de formato xml a csv y posteriormente crear dichas entradas en el formato aceptado para la creación de entradas que necesita la API de Tensorflow (ver Figura 3.4-b).

```python
def convertFiles(self):
    """
    Metodo llamado al oprimir el boton de convertir a formato .CSV y TFRecord
    :return:
    """
    try:
        if self.ui.lineE4.text() != '' and self.ui.lineE5.text() != '':

            if self._toCSV():
                if(self._generarTFRecord()):     #crear TENSORFLOW RECORD
                    print('TFRecord creados con exito')
                else:
                    print('algo salio mal al crear TFRecord')
            else:
                print('algo salio mal al crear CSV')
        else:
            print('No se puede inciar')

    except Exception as ex:
        print(ex)
```

a)

```python
def _toCSV(self):
    """
    convertir a csv
    :return:
    """
    try:
        for directory in self.sets:
            path = os.path.join(self.ui.lineE4.text(), directory)
            print(path)
            converter = xml_to_csv.xmltocsv(path)
            xml_df = converter.xml_to_csv(path=path)
            image_path = os.path.join(os.getcwd(),
                                      "projects/{}/labels/{}_labels.csv".format(self.nameProject, directory))
            folder = os.path.dirname(image_path)

            if not os.path.exists(folder):
                os.makedirs(folder)
            xml_df.to_csv(image_path, index=None)
            print('Conversión exitosa')

        self.statusBar().showMessage('Conversión exitosa, Guardado en: {}'.format(folder))
        return True
    except Exception as ex:
        print(ex)
        print('carpeta equivocada')
        return False

# ----  4
def _generarTFRecord(self):
    """
    Metodo llamado al oprimir boton 'Generar TFRecord
    :return:
    """
    try:
        argslist = []
        mydir = str(os.path.join(os.getcwd(), 'tools'))
        dirTF = str(os.path.dirname(self.ui.lineE4.text()))

        for set in self.sets:
            arg1 = str(os.environ['ENV1'])       #'/home/gustavo/anaconda3/envs/tfgpu/bin/python3'
            arg2 = 'generate_tfrecord.py'
            arg3 = '--csv_input={}/{}_labels.csv'.format(dirTF, set)
            arg4 = '--output_path={}/{}.record'.format(dirTF, set)
            arg5 = '--image_dir={}'.format(self.ui.lineE5.text())
            argslist = [arg1, arg2, arg3, arg4, arg5]
            subprocess.call(argslist, cwd=mydir)  # run

        self.statusBar().showMessage("TFRecord creados correctamente en: {}".format(
                                                    os.path.dirname(self.ui.lineE4.text())))
        return True
```

b)

Figura 3.4 a) Script convertir archivos b) Script conversión xml a csv, y script convertir csv a TFRecord.

Si ambos procesos son exitosos encontraremos dentro de la ruta labels de nuestro proyecto cuatro archivos nuevos:

- train_labels.csv
- test_labels.csv
- train.record
- test.record

Se tomarán como archivos de entrada soportados por la API train.record y test.record.

3.3.2 Fase de selección y configuración del modelo

La API de Tensorflow funciona usando protocolos de protobuf, esto permite definir bastante de su funcionalidad ajustando valores en archivos sencillos. Para entrenar correctamente un modelo es necesario dar formato a dos archivos en dos pasos:

- label_map.pbtxt: Contiene un mapeo con las etiquetas de los objetos a detectar.

- Archivo de configuración (*.config): Describe cada aspecto relacionado al entrenamiento y evaluación del modelo, existe un gran número de parámetros que se pueden configurar, la mejor configuración dependerá entonces de la aplicación. Se especifica siguiendo un esquema simple de 5 partes, descritas a continuación:

    1. *Model config*: Define que tipo de modelo se entrenará
    2. *Train config*: Ajusta parámetros para entrenar modelo
    3. *Eval config*: Define las métricas que serán reportadas
    4. *Train Input Config*: Define qué conjunto de datos se usará para el entrenamiento
    5. *Eval Input Config*: Define qué conjunto de datos se usará para testear el modelo

Para comenzar el usuario tiene a su disposición una serie de ejemplos, contenidos en la ruta, *~/tools/Tensorflow/object_detection/research/samples/configs*, del apartado herramientas del directorio raíz.

Una vez el archivo de configuración está completado, se pasará al entrenamiento del mismo.

- Paso 1 - label_map.pbtxt: La estructura de este archivo es relativamente sencilla, para cada objeto a detectar se escribe un bloque adicional de cuatro líneas, cambiando únicamente el número de id y el nombre para la etiqueta.

```
item{
  id: 1
  name: 'persona'
}
```

  Estas pueden ser editadas a mano o agregando el número de clases en la esquina superior izquierda y pulsando el botón OK (ver Figura 3.5-a), esto agregara automáticamente el número de bloques necesarios con su respectivo id (ver Figura 3.5-b), una vez editado el nombre para todos los bloques debe ser salvado usando el botón guardar.

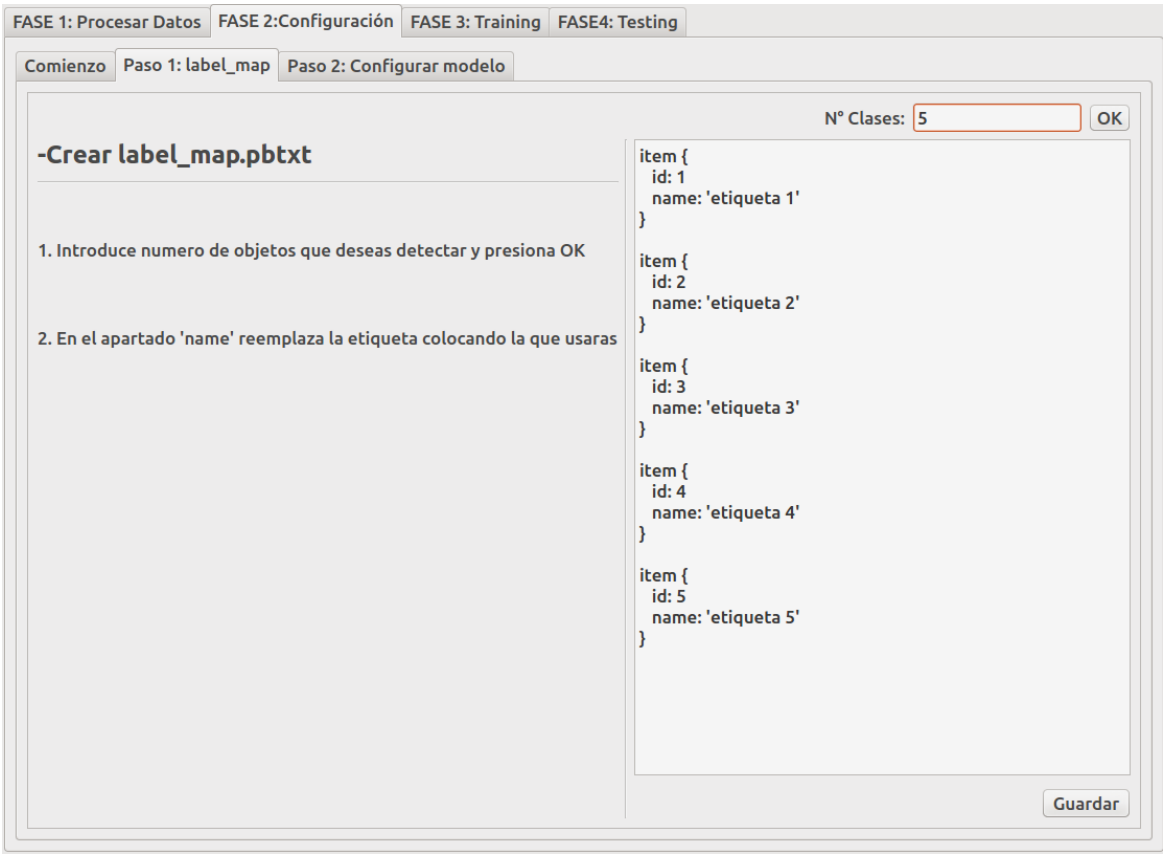


Figura 3.5-a Fase 2 primer paso, creación mapa de etiquetas.

```python
def load_lblmap(self):
    """
    Se crea:
        "item {
            id: 1
            name: 'etiqueta 1'
        }"
    :return:
    """
    try:
        if self.ui.lineE_lbls.text() != '':
            aux = self.ui.lineE_lbls.text()
            aux = int(aux)  # la convertimos a int, si es una letra cae en la excepccion
            self.ui.textE_lblmap.clear()
            for x in range(aux):
                self.ui.textE_lblmap.append("item {")
                self.ui.textE_lblmap.append("    id: {}".format(x + 1))
                self.ui.textE_lblmap.append("    name: 'etiqueta {}'".format(x + 1))
                self.ui.textE_lblmap.append("}")
                self.ui.textE_lblmap.append("")

            self.statusBar().showMessage('{} Clases seran cargadas'.format(aux))

        else:
            self.statusBar().showMessage("Inserte un numero")

    except Exception as ex:
        print(ex)
        self.statusBar().showMessage("Inserte un numero")
```

Figura 3.5-b Fase 2 primer paso, script cargar mapa de etiquetas.

- Paso 2 - Archivo de configuración: Para seleccionar algún tipo de arquitectura, el usuario tiene acceso a una colección de modelos pre entrenados basados en distintos datasets. Estos modelos son de gran utilidad debido a que es posible hacer inferencia en ellos para detectar objetos que ya conoce, o hacer que aprendan con base en imágenes recolectadas personalmente. En esta sección el usuario deberá seleccionar un modelo a partir de su archivo de configuración, descargarlo y posteriormente editar algunos cambios. En [27] se encontrará una mayor información sobre los diferentes modelos existentes y cuál se adecua mejor al proyecto del usuario. Cada modelo pre entrenado contiene:

    - Nombre del modelo, su nombre corresponde a un archivo.
    - Un link para descargar un tar.gz que contiene el modelo, para consultar con detalle ir a README.md en la carpeta models del directorio raíz.
    - Velocidad del modelo en ms, considerar que los tiempos dependen de la configuración de hardware, siendo tiempos relativos que ayudan a comparar el rendimiento entre diferentes arquitecturas en función de su velocidad y precisión, para consultar con detalle ir a README.md en la carpeta models del directorio raíz.
    - Salidas de tipo caja, permiten crear un seguimiento de predicción en la imagen.

Una vez seleccionado el archivo se cargará en el área de texto, y será posible obtenerlo al presionar el botón descargar (ver Figura 3.6), el porcentaje en la descarga se observa en la barra de progreso. Si se desea es posible cancelar la descarga, una vez cancelada el archivo se eliminará de la carpeta models. Si el modelo ya ha sido descargado previamente, se tomará como descarga completada.

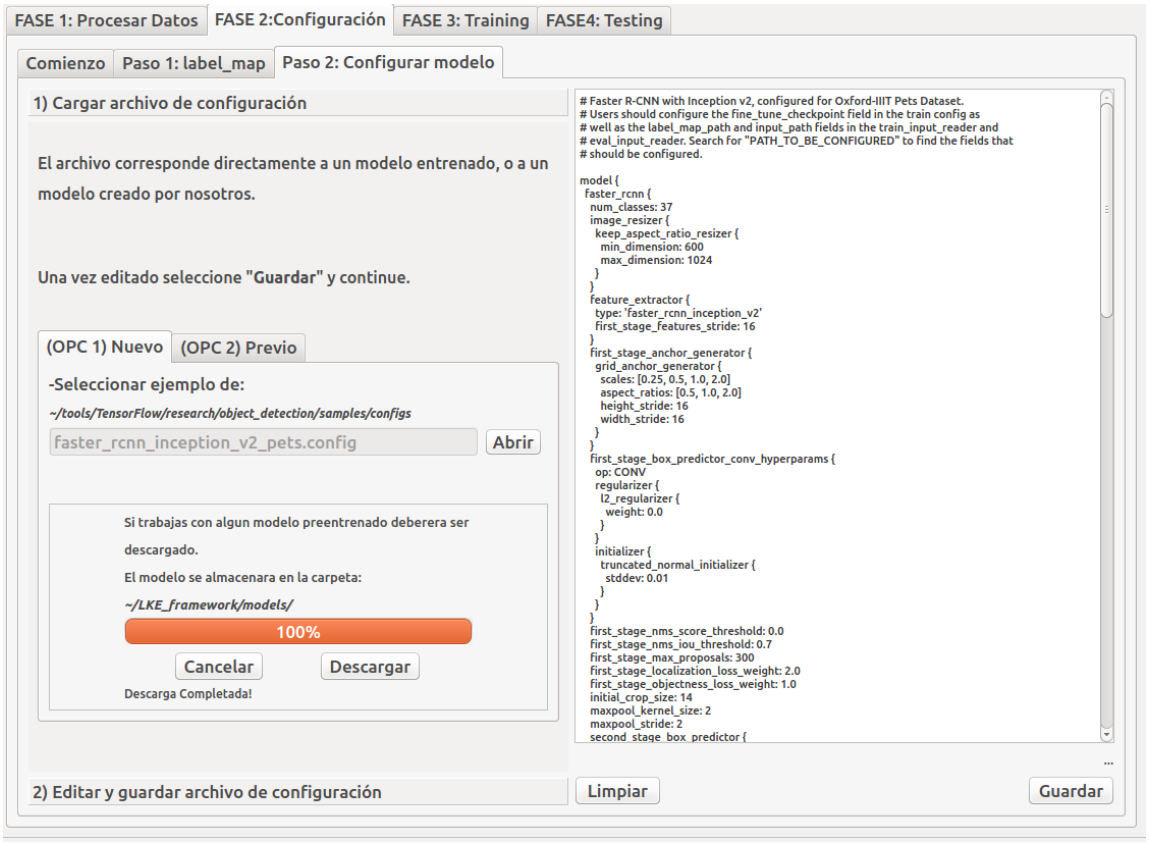


Figura 3.6 Fase 2 segundo paso, selección y descarga de modelo.

Para el proceso de descarga, primero es necesario verificar la url correspondiente (ver Figura 3.7), el conjunto de urls' disponibles se encuentra en el archivo *models4download.txt* (ruta: */src*), posteriormente se lanza el proceso de descarga (ver Figura 3.8), por default los modelos descargados estarán contenidos en la ruta *models* del directorio raíz.

Si el usuario desea continuar el entrenamiento de un proyecto previo, puede pasar al apartado (OPC 2) Previo, este solo deberá abrir el archivo y continuar a la fase de entrenamiento.

```
def downloadModel(self):
    """
    descargar modelo
    buscar en el archivo '/src/model4download.txt' la ruta especificada de des
    :return:
    """
    self.ui.downModel.setEnabled(0)

    m4d = os.path.join(os.getcwd(), "src/models4download.txt")
    flag = False
    url = None
    try:
        # abrir modelos para descarga
        with open(m4d, "r") as f:
            array = []
            for line in f:
                if not line.startswith('#'):
                    if not line.startswith('\n'):
                        line = line.strip()
                        l = line.split(',')
                        array.append(l)
            a = np.asarray(array)

            for idx, val in enumerate(a):
                if self.modelConfig in val:
                    url = array[idx + 2]
                    url = ''.join(url)
                    print('URL:' + url)
                    flag = True

        f.close()
        self.thread3.url = url  # pasamos url al hilo 3
        self.url = url          # pasamos url a ventana principal

    except Exception as ex:
        print(ex)
        flag = False

    if not flag: #
        self.ui.download.setText('No se puede iniciar la descarga')
        self.ui.downModel.setEnabled(1)
    else:
        try:
            # lazamos thread para descargar el modelo
            self.thread3.start()
        except Exception as ex:
            print(ex)
            print('Error descarga hilo 3')

def cancelDonwload(self):
    """
    cancelar descarga del modelo seleccionado en la carpeta ~/models
    :return:
    """
    if not self.thread3.isRunning():
        print("Hilo inactivo")
    else:
        try:
            print("Hilo activado y listo para detener")
            self.ui.download.setText('Descarga cancelada')
            self.ui.downModel.setEnabled(1)
            self.ui.progressBar.setValue(0)

            modelsDir = str(os.path.join(os.getcwd(), "models"))  # se guarda
            filename = os.path.join(modelsDir, os.path.basename(self.url))
            os.remove(filename)
            self.thread3.stop()

        except Exception as ex:
            print(ex)
            print('!error descargar modelo')
```

Figura 3.7 Métodos buscar url y cancelar descarga

```python
def run(self):
    """
    iniciar descarga del modelo, si ya exite en el dir modelos ya no se descarga,
    si no comenzara automaticamente
    :return:
    """
    try:
        time.sleep(2)
        print('In thread')
        modelsDir = str(os.path.join(os.getcwd(), "models"))  # se guarda en carpeta models
        filename = self.url.split('/')[-1]
        m = os.path.join(modelsDir, os.path.basename(self.url))

        if os.path.exists(m):
            self.emit(self.signal)

        else:
            with open(os.path.join(modelsDir, filename), 'wb') as f:
                self.emit(self.signal2)
                response = requests.get(self.url, stream=True)
                total = response.headers.get('content-length')
                if total is None:
                    f.write(response.content)
                else:
                    downloaded = 0
                    total = int(total)
                    for data in response.iter_content(chunk_size=max(int(total / 1000), 1024 * 1024)):
                        downloaded += len(data)
                        f.write(data)
                        done = int(50 * downloaded / total)
                        ok = done * 2
                        self.emit(self.signal3, ok)
                        sys.stdout.write('\r[{}{}]'.format('█' * done, '.' * (50 - done)))
                        sys.stdout.flush()

                self.emit(self.signal4)

            sys.stdout.write('\n')
            f.close()

        print('End thread 3')
    except Exception as ex:
        print(ex)
        print('!error descargar modelo')
        self.emit(self.signal5)
```

Figura 3.8 Método para iniciar descarga

El modelo descargado debe descomprimirse, contiene:

- Un frozen_inference_graph, o gráfico computacional suspendido, es decir un grafo computacional unido a sus pesos como constantes, este formato facilita su inferencia.
- Un checkpoint o punto de guardado, el cual comprende la estructura básica de guardado para un modelo en Tensorflow, se compone de tres archivos:

    - model.ckpt.data-00000-of-00001
    - model.ckpt.index
    - model.ckpt.meta: Protocolo de buffer que almacena el gráfico del modelo, es decir sus operaciones, variables, colecciones, etc.

Siendo .data y .index archivos que almacenan los valores de los pesos, bias, gradientes y variables guardadas.

Una vez finalizado este apartado es necesario editar algunos parámetros tal que se ajuste a las necesidades del usuario. En [28] se encuentra la documentación

correspondiente al ajuste de parámetros en el archivo de configuración, como ayuda para el usuario se añade una pestaña adicional para editar el archivo de configuración (ver Figura 3.9). Por último es necesario guardar el archivo de configuración, este dará la opción por defecto para guardarlo en la carpeta training de nuestro proyecto, como paso siguiente continuar a la fase de entrenamiento.

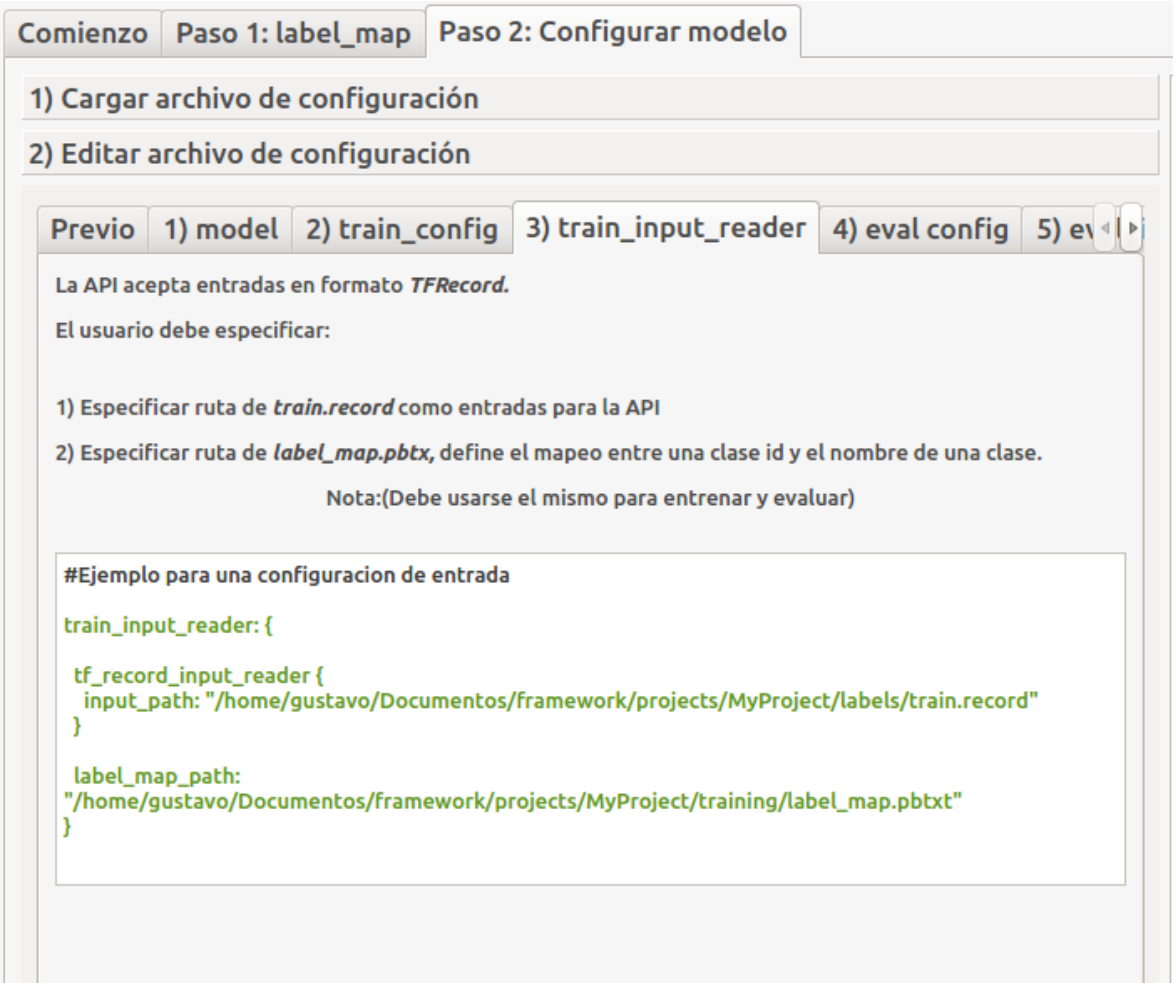


Figura 3.9 Fase 2 Apartado editar archivo de configuración.

3.3.3 Fase de entrenamiento

Esta fase comprende el entrenamiento del modelo configurado, como requisito debe cargarse el archivo de configuración (ver Figura 3.10), una vez cargado, presionar botón entrenar, en el área de texto será posible observar el comportamiento de la red durante el entrenamiento, es decir sus variaciones al disminuir el error y épocas. Durante el entrenamiento cada cierto número de épocas el sistema guarda un checkpoint, con los archivos antes mencionados, así como un archivo binario de eventos, este archivo será de utilidad para el despliegue de resultados en forma visual en la fase de resultados.

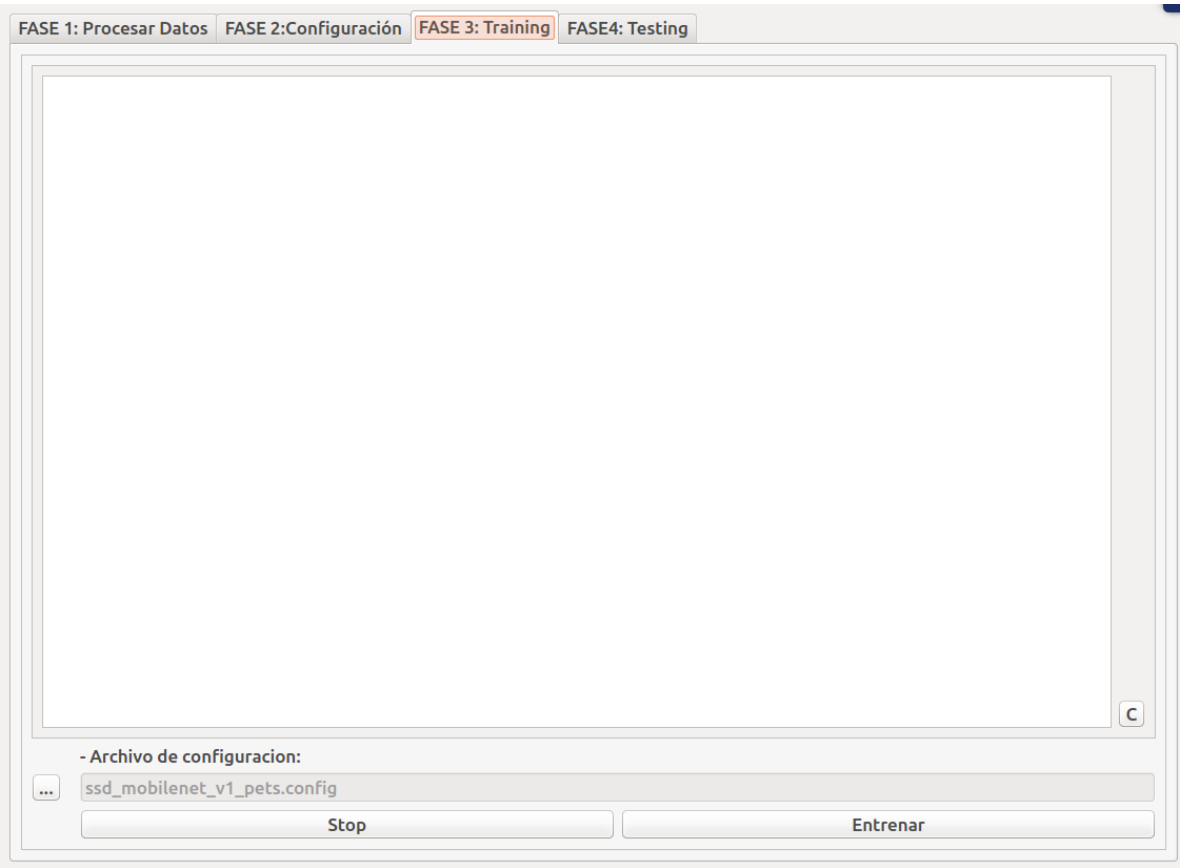


Figura 3.10 Interfaz para el entrenamiento

Para iniciar el entrenamiento usando la API y el archivo de configuración, es necesario ejecutar train.py el siguiente comando en terminal:

```
# From the tensorflow/models/research/ directory
python object_detection/train.py \
    --logtostderr \
    --pipeline_config_path=${PATH_TO_YOUR_PIPELINE_CONFIG} \
    --train_dir=${PATH_TO_TRAIN_DIR}
```

En la Figura 3.11 se muestra cómo ejecutar el script pasando como parámetros la ruta training para guardar los archivos generados durante el entrenamiento.

```
class hilo(QtCore.QThread):
    """
    Clase hilo que corre el proceso de entrenamiento
    Example usage:

    ./train \
        --logtostderr \
        --train_dir=path/to/train_dir \
        --pipeline_config_path=pipeline_config.pbtxt
    """
    def __init__(self, parent=None):
        QtCore.QThread.__init__(self, parent=app)
        self.signal  = QtCore.SIGNAL("mySignal1")
        self.signal2 = QtCore.SIGNAL("mySignal2")
        self.signalC = QtCore.SIGNAL("mySignal3")
        self.nameP = None
        self.fileCfg = None

    def run(self):
        time.sleep(2)
        print('In THREAD TRAINING')
        self._runtrain()
        print('END THREAD TRAINING')

    def _runtrain(self):
        mydirpath = str(os.environ['OBJECTDETECTIONPATH']) #path Object_detection
        d1 = str(os.environ['ENV1'])        #/envs/tfgpu/bin/python'

        arg1 = d1
        arg2 = 'train.py'
        arg3 = '--logtostderr'
        model = str(os.path.join(os.getcwd(), "projects/{}/training/".format(self.nameP)))
        arg4 = '--train_dir={}'.format(model)
        arg5 = '--pipeline_config_path={}{}'.format(model, self.fileCfg)

        command = arg1 + ' ' + mydirpath + arg2 + ' ' + arg3 + ' ' + arg4 + ' ' + arg5

        self.emit(self.signalC)

        try:
            p = subprocess.Popen(shlex.split(command), stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
            print('-- proceso creado --')
            stdout = []
            while True:
                line = p.stdout.readline()
                stdout.append(line)
                print(line)
                self.emit(self.signal, line)
                if line == '' and p.poll() != None:
                    break
        except Exception as ex:
            print(ex)
```

Figura 3.11 Script Iniciar proceso de entrenamiento

3.3.4 Fase de resultados

La última fase es mostrada en la Figura 3.12, se compone de tres tareas de gran utilidad para evaluar el aprendizaje:

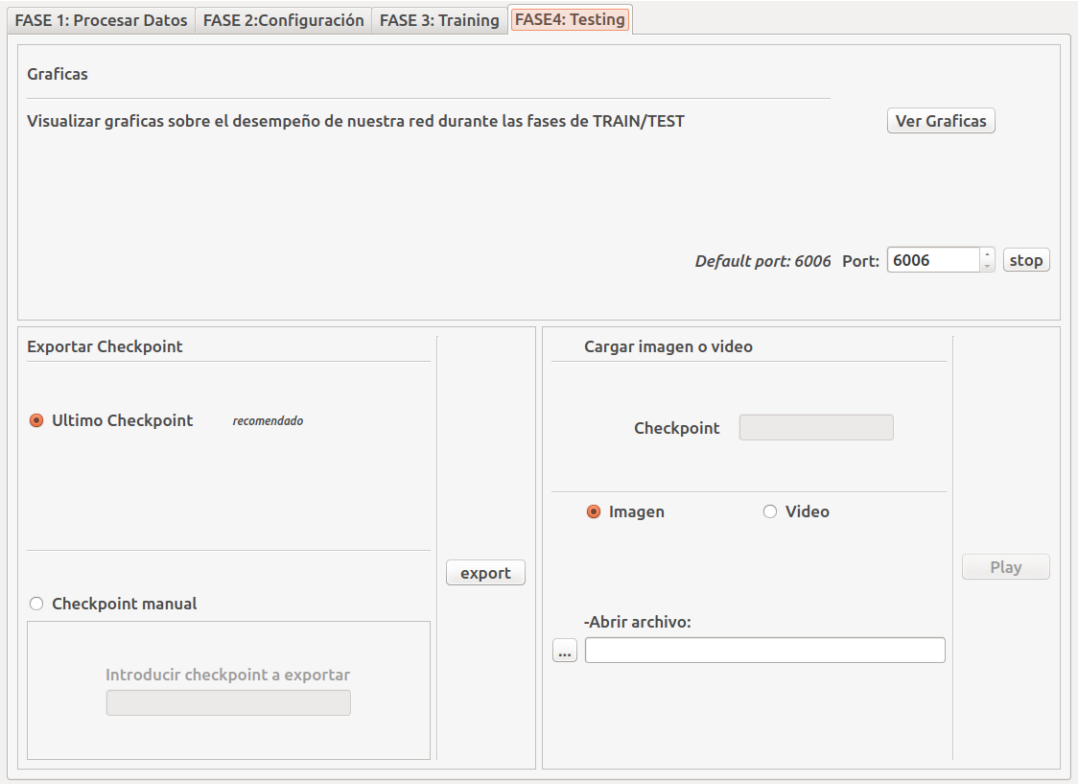


Figura 3.12 Fase 4 Graficar resultados, exportar modelos y probar modelos

Visualizar gráficas: En este paso se recogen los archivos binarios de eventos generados durante la fase de entrenamiento usando TensorBoard, esto permite abrir una pestaña nueva en el navegador conectada a localhost usando un puerto específico, si el puerto no es ajustado, se usará el puerto 6006 de forma predeterminada. Al oprimir 'Ver Gráficas' se lanza el proceso pasando como parámetros la ruta que contiene los archivos events creados durante el entrenamiento (ver Figura 3.13).

```python
def verGraficas(self):
    """
    Cargamos graficas usando TENSORBOPARD
    :return:
    """
    if self.thread2.isRunning():
        print("puerto {} aun activo".format(self.thread2.port))
    else:
        arg1 = str(os.environ['ENV1'])      #'/home/gustavo/anaconda3/envs/tfgpu/bin/python'
        arg2 = 'main.py'
        arg3 = '--logdir={}'.format(os.path.join(os.getcwd(), "projects/{}/training/".format(self.nameProject)))
        arg3 = str(arg3)

        self.thread2.port = str(self.ui.spinBport.value())
        arg4 = '--port={}'.format(self.thread2.port)

        #modificamos las variables dentro del hilo para que tome esos valores
        # al momento de correr tensorboard
        self.thread2.list = [arg1, arg2, arg3, arg4]
        self.thread2.dir = str(os.environ['TENSORBOARD'])   #'/home/gustavo/anaconda3/envs/tfgpu/lib/python3.5/site

        try:
            self.thread2.start()
        except Exception as ex:
            print(ex)
            print('Error al ver las graficas ')
```

```python
def run(self):
    """
    Run hilo para visualizar graficas usando tensorboard
    :return:
    """
    time.sleep(2)
    print('In thread')
    self._launchTensorBoard()
    print('End thread2')

def _launchTensorBoard(self):
    """
    lanza TB en el canvas para web
    :return:
    """
    try:

        subprocess.call(self.list, cwd=self.dir)

        def_URL_test = 'http://localhost:{}'.format(self.port)
        print('Abrir TensorBoard ---> ' + def_URL_test)

        webbrowser.open_new_tab(def_URL_test) #abre nueva pestaña en el navegador
        return

    except Exception as ex:
        print(ex)
```

Figura 3.13 Script lanzar TensorBoard usando el navegador

Exportar modelo: Para poder hacer una inferencia a partir de un modelo, es necesario exportar dicho modelo, al exportarlo se creará una carpeta nueva en el directorio training, este contiene el gráfico suspendido, similar al que descargamos en la fase 2. Es posible exportar el modelo detectando automáticamente el último punto de guardado o exportarlo especificando el número de entrenamiento que desee ser probado. La API permite esta función ejecutando el siguiente comando en la ruta /Tensorflow/research/object_detection:

```
#From tensorflow/research/
python object_detection/export_inference_graph.py \
    --input_type image_tensor \
    --pipeline_config_path ${PIPELINE_CONFIG_PATH} \
    --trained_checkpoint_prefix ${TRAIN_PATH} \
    --output_directory output_inference_graph.pb
```

La Figura 3.14-a muestra la definición de parámetros y la ejecución del comando.

```
try:
    if(self._checkModelConfig()):
        num_check = ''
        b = 0       #si b = 1 los datos son validos y se puede exportar

        #cambiar el checkpoint segun sea el radioButton
        if(self.ui.rb_lastCheck.isChecked()):

            num_check = self._findlastcheckp()  #buscar el ultimo
            if num_check.__eq__(-1):    # regressa -1 si da error al buscar
                b = 0

            else:
                print('checkpoint = {}'.format(num_check))
                b = 1

        if(self.ui.rb_manualCheck.isChecked()):

            if(str(self.ui.lineE_checkpoint.text()) == ''):
                self.statusBar().showMessage('escribe un numero valido')
                print('escribe un numero valido')
                b = 0
            else:
                num_check = self.ui.lineE_checkpoint.text()
                print('checkpoint = {}'.format(num_check))
                # metodo buscar si el numero de checkpoint es valido
                # si no es valido informamos, si lo es bandera = 1
                b = self._validarCheckpoint(num_check)


        arg1 = str(os.environ['ENV1'])      #'/home/gustavo/anaconda3/envs/tfgpu/bin/python3'

        arg2 = 'export_inference_graph.py'
        arg3 = '--input_type image_tensor'
        arg4 = '--pipeline_config_path {}/projects/{}/training/{}'.format(os.getcwd(),
                                                    self.nameProject, self.modelConfig)

        #si la bandera == 1 entonces corremos el comando
        if(b.__eq__(1)):
            self.exportfiles = '{}_graph_{}'.format(self.nameProject, num_check)

            arg5 = '--trained_checkpoint_prefix {}/projects/{}/training/model.ckpt-{}'.format(
                                            os.getcwd(), self.nameProject, num_check)

            arg6 = '--output_directory {}/projects/{}/training/{}'.format(
                                            os.getcwd(), self.nameProject, self.exportfiles)

            path = os.path.join(os.getcwd(), 'projects/{}/training/{}'.format(
                                                    self.nameProject, self.exportfiles))

            command = arg1 + ' ' + str(os.environ['OBJECTDETECTIONPATH']) + arg2 + ' ' + arg3 + ' ' + arg4 + ' ' + arg5 + ' ' + arg6

            self.statusBar().showMessage('Checkpoint valido')
            self._exportar(path, command)
        else:
            print('no se puede iniciar')
            self.statusBar().showMessage('Error: Intente un checkpoint valido')

except Exception as ex:
    print(ex)
    self.statusBar().showMessage('error al exportar')
```

Figura 3.14 a) Definición de parámetros para exportar

```
def exportar(self, path, command):
    """
    :param path:
    :param command:
    :return:
    """
    # run comando
    if (os.path.exists(path)):
        self.statusBar().showMessage('Ya existe la carpeta'
                                     ' "{}" en: {}'.format(self.exportfiles, os.path.dirname(path)))
        self.checkpointPath = path
        self.ui.lineE_checkpoint_main.setText(os.path.basename(self.checkpointPath))

    else:
        print('Creando carpeta {}'.format(self.exportfiles))
        subprocess.Popen(shlex.split(command))
        time.sleep(2)
        self.statusBar().showMessage('Carpeta '
                                     '"{}" exportada'.format(self.exportfiles))
        self.checkpointPath = path
        self.ui.lineE_checkpoint_main.setText(os.path.basename(self.checkpointPath))
```

Figura 3.14 b) Ejecución de comando exportar modelo

Para exportar correctamente un checkpoint de forma manual, verifique que los archivos correspondientes a dicho checkpoint exista. Si el checkpoint fue exportado exitosamente se hallará una nueva carpeta en el directorio training (ver Figura 3.15), llamada: {nombreProyecto_graph_NumCheckpointExportado}.

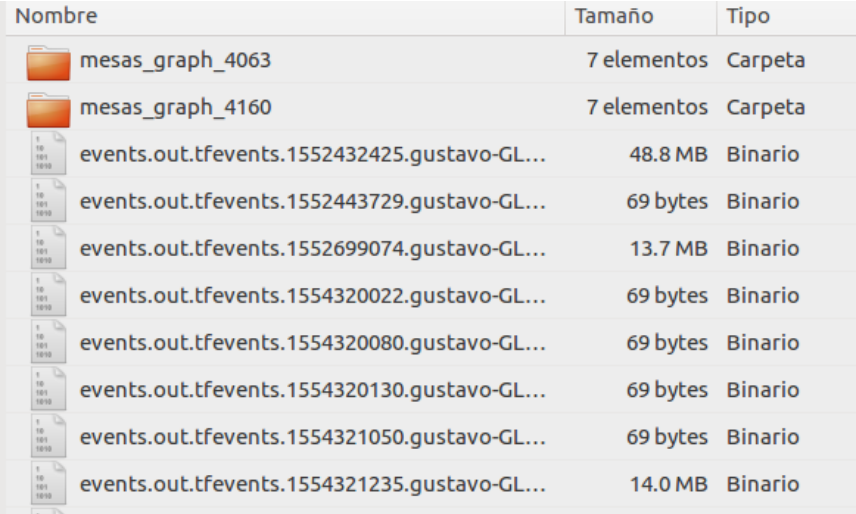


Figura 3.15 Carpeta que contiene el modelo exportado

Probar modelo con imágenes o video: En este último paso es posible verificar que tan bien o mal ha aprendido la red neuronal, una vez cargado un checkpoint válido y una imagen o video, el programa hará inferencia en dicho archivo al presionar el botón play. Si el modelo ha aprendido correctamente a detectar el objeto de entrenamiento dibujara un recuadro alrededor del objeto, lo mismo para múltiples instancias en una sola imagen o video. Dependiendo si el objeto cargado es una imagen o un video el sistema ejecutará un script

separado para cada proceso (ver Figura 3.16); como base de ambos scripts se ha tomado [29] en la documentación de la API, este permite usar modelos pre entrenados para detectar objetos en imágenes.

```
def play(self):
    """
    PLay video o imagen usando la red neuronal
    :return:
    """
    try:
        if (self._checkexport()): #checar valida este cargada

            self.numClases = self._checkclsslbl_map()  # obtiene el numero de clases


            if self.numClases.__eq__(-1):
                print('error al leer labelmap.pbtxt')

            else:

                if (self.ui.rb_imagen.isChecked() and os.path.exists(self.ui.lineEpathvideo.text())):

                    mydir = str(os.environ['OBJECTDETECTIONPATH'])
                    arg1 = str(os.environ['ENV1'])
                    arg2 = 'object_detection_test.py'
                    arg3 = '--image_dir={}'.format(self.ui.lineEpathvideo.text())
                    arg4 = '--graph_exported={}'.format(self.checkpointPath)
                    arg5 = '--label_map={}'.format(os.path.join(os.getcwd(),
                                                   "projects/{}/training/label_map.pbtxt".format(self.nameProject)))
                    arg6 = '--numClass={}'.format(self.numClases)

                    argslist = [arg1, arg2, arg3, arg4, arg5, arg6]
                    subprocess.call(argslist, cwd=mydir)  # run

                    self.statusBar().showMessage('Cargado con exito')
                else:
                    self.statusBar().showMessage('path {} no existe'.format(self.ui.lineEpathvideo.text()))

                if (self.ui.rb_video.isChecked() and os.path.exists(self.ui.lineEpathvideo.text())):
                    mydir = str(os.environ['OBJECTDETECTIONPATH'])
                    arg1 = str(os.environ['ENV1'])
                    arg2 = 'object_detection_test_video.py'
                    arg3 = '--image_dir={}'.format(self.ui.lineEpathvideo.text())
                    arg4 = '--graph_exported={}'.format(self.checkpointPath)
                    arg5 = '--label_map={}'.format(os.path.join(os.getcwd(),
                                                   "projects/{}/training/label_map.pbtxt".format(self.nameProject)))
                    arg6 = '--numClass={}'.format(self.numClases)

                    argslist = [arg1, arg2, arg3, arg4, arg5, arg6]
                    subprocess.call(argslist, cwd=mydir)  # run

                    self.statusBar().showMessage('Cargado con exito')
                else:
                    self.statusBar().showMessage('path {} no existe'.format(self.ui.lineEpathvideo.text()))

        else:
            self.statusBar().showMessage("error: exporte un checkpoint valido ")

    except Exception as ex:
        print(ex)
```

Figura 3.16 Script inferencia en modelo para imágenes/video

4. Aplicación caso de uso: Estado de mobiliario

4.1 Base de datos de aprendizaje y test

Como base de datos de aprendizaje y test se ha recopilado una serie de imágenes directamente de las cámaras de seguridad de tres sucursales diferentes en el estado de México, el acceso a tales fue proporcionado por el laboratorio LKE. En la Figura 4.1 se puede ver un ejemplo de las imágenes de este conjunto de datos.

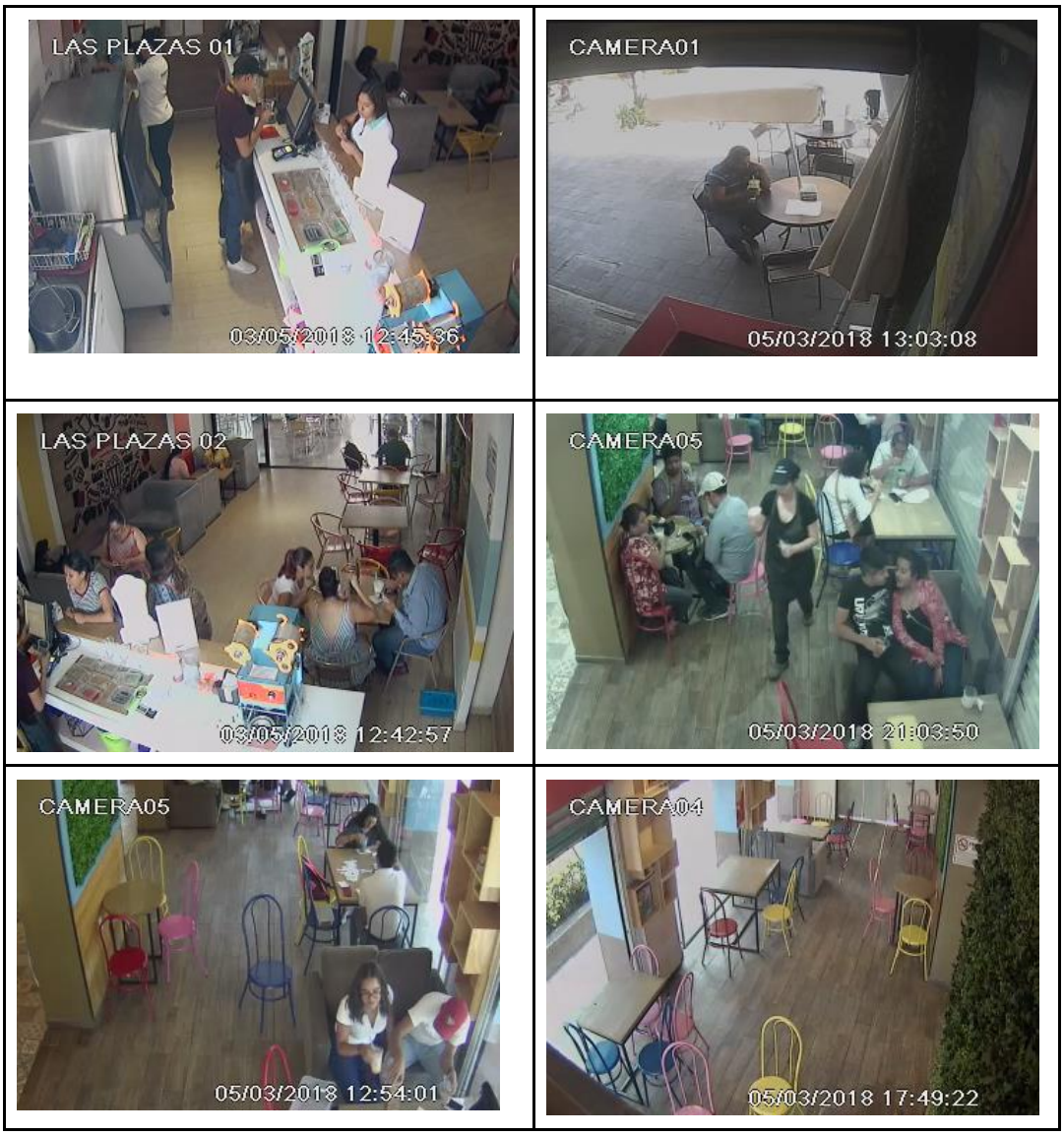


Figura 4.1 Muestra de imágenes en la base de datos

Las imágenes fueron recolectadas distintos días, a lo largo de diferentes horas del día, buscando obtener mejores variaciones lumínicas, la resolución de cada imagen es de 352 x 240 pixeles en color. La base de datos cuenta con cerca de 400 elementos.

Para comenzar se colocan las imágenes dentro de la carpeta ‘data’ ubicada en la raíz del proyecto, e iniciamos un nuevo proyecto. Se da un nombre para el proyecto, de igual manera agregamos (de forma opcional) una breve descripción y presionamos OK para continuar (ver Figura 4.2). A Continuación, podremos observar la creación de una nueva carpeta llamada mobiliario en la ruta ‘projects’. Aquí se almacenan todos los datos necesarios para el entrenamiento.

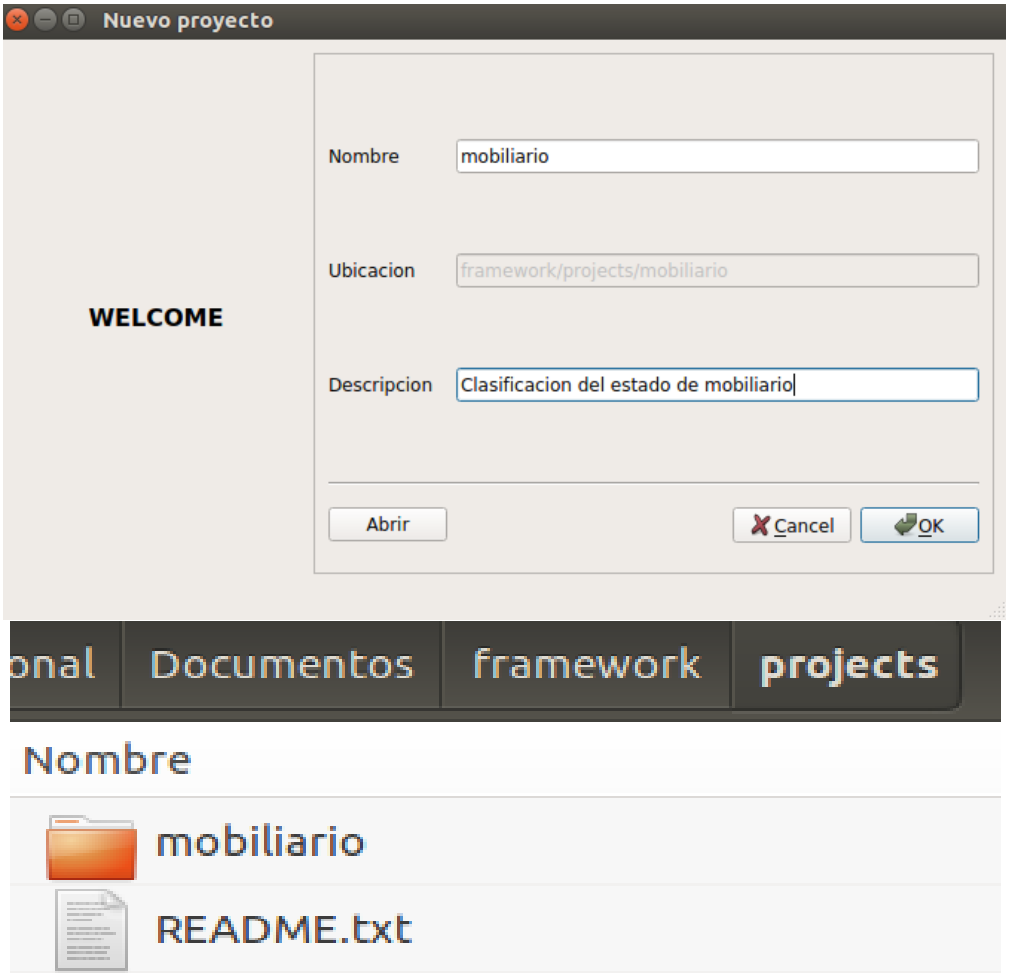


Figura 4.2 Nuevo proyecto llamado mobiliario y ruta del proyecto

Haciendo uso del apartado etiquetar abrimos la carpeta con las imágenes, y seleccionamos manualmente las partes de la imagen que contengan información útil para el entrenamiento (ver Figura 4.3) asignándoles su etiqueta correspondiente.

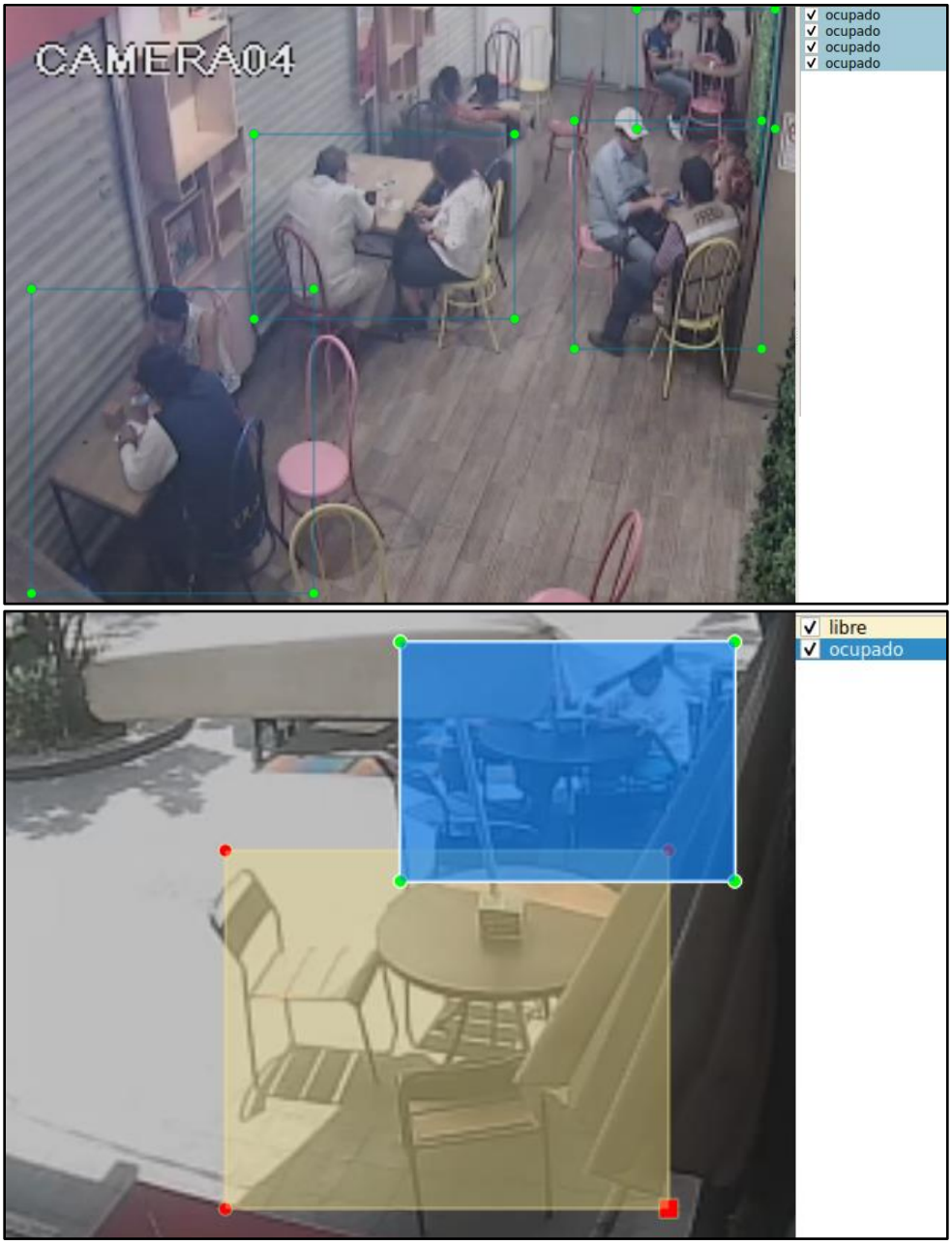


Figura 4.3 Muestra etiquetado de imágenes para el proyecto

4.2 Creación de entradas

Una vez finalizado el proceso de etiquetado es necesario crear los respectivos conjuntos de TRAIN/TEST, así como convertir las entradas xml al formato aceptado por la API. Para dividir los datos se ha tomado el porcentaje predeterminado (70% Train, 30% Test). Una vez divididos se oprimimos botón convertir, como resultado ahora tenemos dos carpetas nuevas en el directorio de nuestro proyecto (labels, training), labels ahora contiene una carpeta con los conjuntos en xml, y las entradas binarias en formato record. Debe aparecer de la siguiente forma:

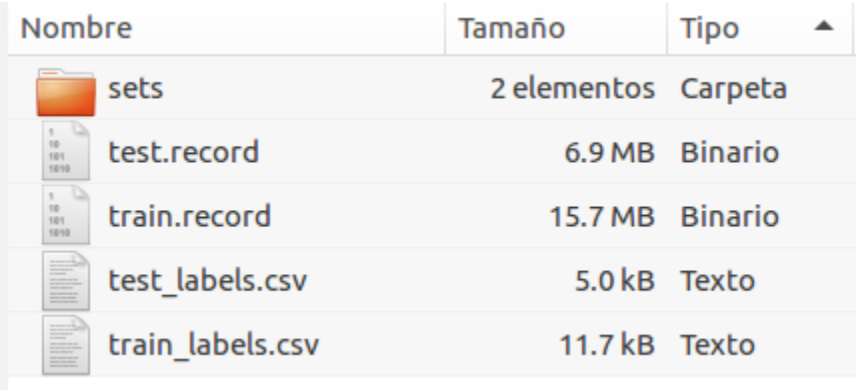


Figura 4.4 Resultado del proceso de creación de entradas

4.3 Configuración del modelo

El configurar correctamente el modelo requiere la creación de dos archivos adicionales:

- label_map.pbtxt
- Archivo de configuración

En el apartado para la creación del label_map introducimos el número de clases que deseamos clasificar, en este caso dos y presionamos OK, luego cambiamos el nombre con el de la etiqueta de la clase. El archivo debe lucir de la siguiente forma:

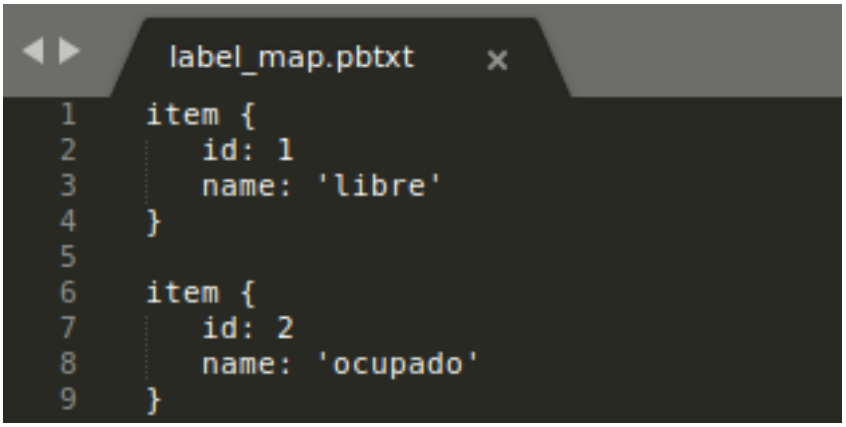


Figura 4.5 Archivo label_map.pbtxt

Ahora es necesario elegir un modelo proporcionado por la API de Tensorflow, estos modelos siguen diferentes arquitecturas, y actualmente son ampliamente utilizados debido a que han demostrado su capacidad de aprendizaje en problemas de reconocimiento de imágenes con una tasa aceptable de aciertos. A continuación, se describe el modelo que elegimos para este proyecto.

4.3.1 SSD: Single Shot MultiBox Detector

Esta arquitectura [30] fue presentada a finales de 2016, como un modelo para detectar imágenes usando una simple red neuronal profunda (ver Figura 4.6), la cual se enfoca en la discretización de las salidas usando cajas delimitadoras sobre diferentes aspectos y escalas dependiendo de la localización del mapeo de características. En tiempo de predicción la red genera puntajes para la presencia de cada categoría, ajustando la caja para coincidir mejor con la forma del objeto. En los siguientes puntos se describe de forma general de donde surge el nombre para esta arquitectura:

- **Single Shot:** Significa que las tareas de localización y clasificación de objetos son realizadas en un simple paso hacia adelante de la red.
- **MultiBox:** Cajas múltiples es el nombre de la técnica para delimitar recuadros que encierran la clasificación.
- **Detector:** La red es un detector de objetos que también clasifica los objetos detectados.

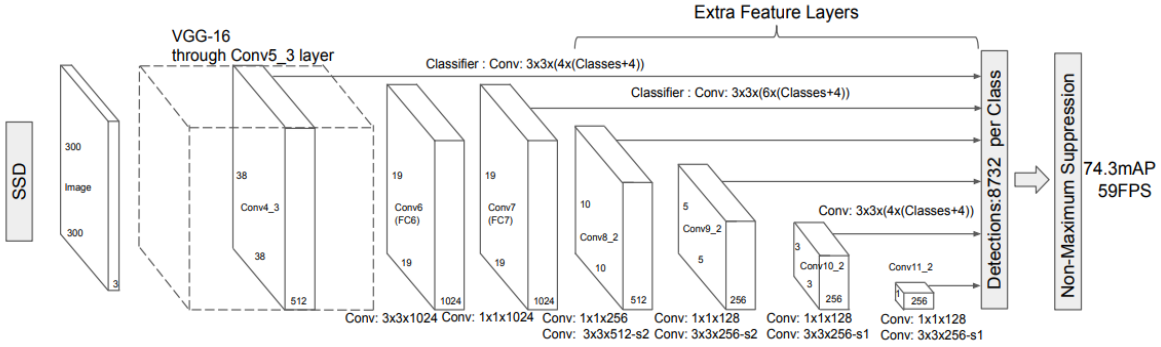


Figura 4.6 Arquitectura SSD

Las primeras capas de la red se basan en una arquitectura estándar utilizada para clasificación de imágenes de alta calidad (VGG-16), también es usada como base de la red debido a su popularidad donde la transferencia de aprendizaje (transfer learning) ayuda a mejorar los resultados, sin embargo se diferencia de la arquitectura original VGG (ver Figura 4.6) ya que descarta capas fuertemente conectadas, e introduce un conjunto de capas convolucionales auxiliares, que son añadidas al final de la base de la red. Cada capa añadida puede producir un ajustado conjunto de predicciones usando un conjunto de filtros convolucionales, esto habilita la extracción de características en múltiples escalas y progresivamente decrementa el tamaño de la entrada en las capas subsecuentes.

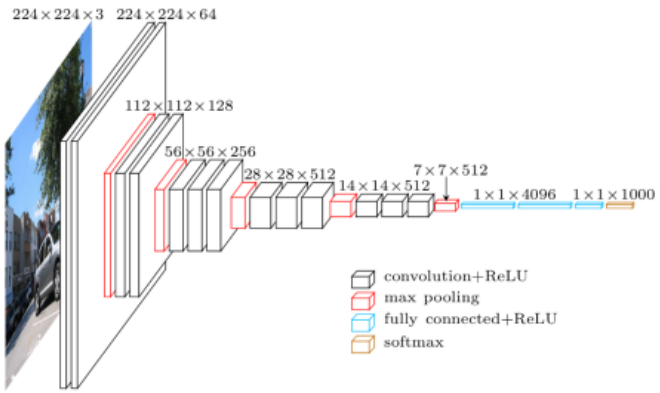


Figura 4.7 VGG architecture (input is 224x224x3)

La técnica de regresión para cajas delimitadoras está inspirada por el trabajo de Szegedy's, llamado MultiBox[31], un método lógico para la rápida propuesta de coordenadas de cuadros límites.

En MultiBox los investigadores crean cajas delimitadoras de tamaño fijo precalculado, las cuales coinciden cercanamente con la distribución de cuadros originales (*priors*), esto permite obtener su relación de intersección sobre la unión, llamado índice IoU. En la figura 4.8 se muestra la lógica de su funcionamiento, por ejemplo, si su índice IoU es mayor que 0.5, es posible inferir que no es lo suficientemente bueno, pero propone un fuerte punto de partida para el algoritmo de regresión, lo cual resulta mucho mejor que comenzar las predicciones con coordenadas aleatorias. Es así que comienza con los priors como predicciones e intenta retroceder más cerca de las cajas limitadoras reales. Al final MultiBox sólo retiene K predicciones principales que han minimizado la pérdida.

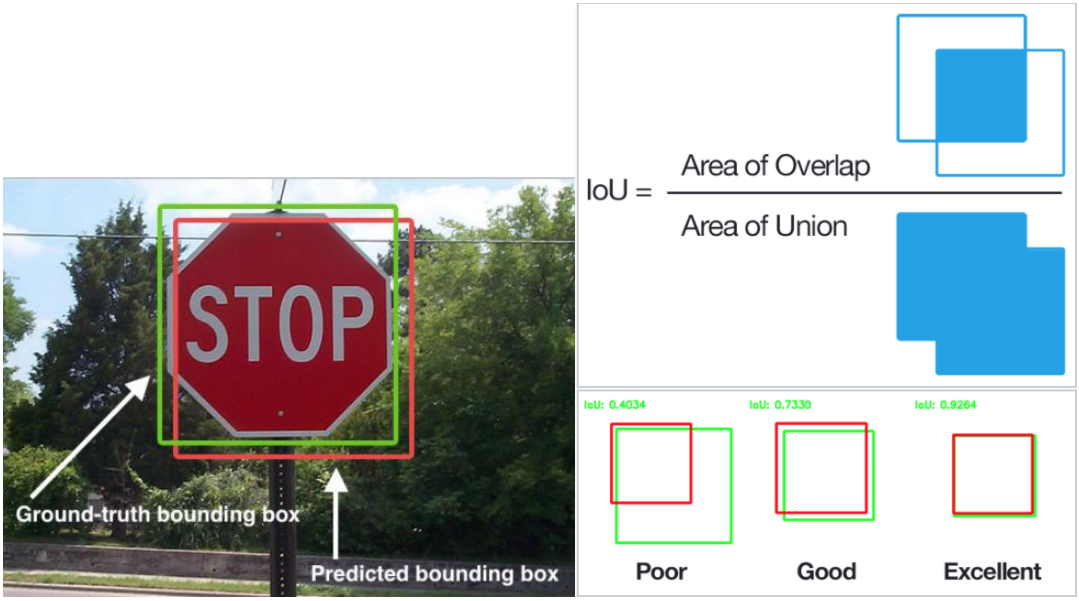


Figura 4.8 Diagrama IoU

Adicionalmente el modelo hace las siguientes observaciones:

- A mayor cantidad de cajas por default resulta en detecciones más precisas, así mismo usar MultiBox en múltiples capas genera una mejora en la detección debido a que el detector funciona con características en múltiples resoluciones.
- SSD produce un rendimiento bajo en pequeños objetos, tal que pueden no aparecer a través de todo el mapa de características, incrementando la resolución de la imagen es posible mitigar este problema, sin embargo, no lo soluciona completamente.

4.3.2 Ajuste de parámetros para SSD

Una vez seleccionado el archivo correspondiente al modelo SSD (ssd_mobilenet_v1_pets.config) , es necesario ajustar algunos parámetros, los cuales son:

- Número de clases
- fine_tune_checkpoint: Ruta del modelo descargado
- input_path (train_input_reader, eval_input_reader): Ruta de las entradas TFRecord
- input_label_map (train_input_reader, eval_input_reader): Ruta del archivo label_map.pbtxt

A continuación, se muestran los cambios realizados al archivo de configuración:

```
# SSD with Mobilenet v1, configured for Oxford-IIIT Pets Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.

model {
  ssd {
    num_classes: 2
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
      }
    }
    similarity_calculator {
```

```
  iou_similarity {
  }
}
anchor_generator {
  ssd_anchor_generator {
    num_layers: 6
    min_scale: 0.2
    max_scale: 0.95
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    aspect_ratios: 3.0
    aspect_ratios: 0.3333
  }
}
image_resizer {
  fixed_shape_resizer {
    height: 300
    width: 300
  }
}
box_predictor {
  convolutional_box_predictor {
    min_depth: 0
    max_depth: 0
    num_layers_before_predictor: 0
    use_dropout: false
    dropout_keep_probability: 0.8
    kernel_size: 1
    box_code_size: 4
    apply_sigmoid_to_scores: false
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.03
          mean: 0.0
        }
      }
      batch_norm {
        train: true,
        scale: true,
        center: true,
        decay: 0.9997,
        epsilon: 0.001,
      }
    }
  }
}
feature_extractor {
  type: 'ssd_mobilenet_v1'
```

```
      min_depth: 16
      depth_multiplier: 1.0
      conv_hyperparams {
        activation: RELU_6,
        regularizer {
          l2_regularizer {
            weight: 0.00004
          }
        }
        initializer {
          truncated_normal_initializer {
            stddev: 0.03
            mean: 0.0
          }
        }
        batch_norm {
          train: true,
          scale: true,
          center: true,
          decay: 0.9997,
          epsilon: 0.001,
        }
      }
    }
    loss {
      classification_loss {
        weighted_sigmoid {
        }
      }
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      hard_example_miner {
        num_hard_examples: 3000
        iou_threshold: 0.99
        loss_type: CLASSIFICATION
        max_negatives_per_positive: 3
        min_negatives_per_image: 0
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}
```

```
train_config: {
  batch_size: 24
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
        exponential_decay_learning_rate {
          initial_learning_rate: 0.004
          decay_steps: 800720
          decay_factor: 0.95
        }
      }
      momentum_optimizer_value: 0.9
      decay: 0.9
      epsilon: 1.0
    }
  }
fine_tune_checkpoint:
"/home/gustavo/Documentos/LKE/models/ssd_mobilenet_v1_coco_2017_11_17/model.ckpt"
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the pets dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
 input_path:
"/home/gustavo/Documentos/LKE/projects/mobiliario/labels/train.record"
  }
label_map_path: "/home/gustavo/Documentos/LKE/projects/mobiliario/training/label_map.pbtxt"
}

eval_config: {
  num_examples: 2000
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
  input_path:
"/home/gustavo/Documentos/LKE/projects/mobiliario/labels/test.record"
  }
label_map_path: "/home/gustavo/Documentos/LKE/projects/mobiliario/training/label_map.pbtxt"
  shuffle: false
```

```
  num_readers: 1
}
```

El archivo editado es guardado al interior de la carpeta training de nuestro proyecto, posteriormente damos a la fase de entrenamiento.

4.4 Resultados del entrenamiento

La fase de entrenamiento se realizó en primera a un total cerca de 7000 épocas, a continuación, se muestra el progreso generado y sus principales métricas mostradas:

Global Step	Loss	Loss / classification _loss	Loss / localization _loss	Loss / regularization _loss	Loss / total_loss
1	17.261168	-	-	-	-
255	6.7105513	5.072078	1.2929502	0.34552187	6.7105513
701	3.3755214	5.161645	1.0998743	0.34548315	6.6070004
1101	3.3835344	6.023662	1.1786941	0.34541684	7.547775
1501	3.3290782	5.517826	1.1496817	0.34532169	7.0128284
2001	2.989937	5.533779	0.99393135	0.34518263	6.872894
2501	3.1997135	4.9953012	0.8839436	0.3450516	6.2242966
3101	2.4859824	4.687141	0.77445304	0.3448715	5.806465
3601	2.2956934	5.245084	0.82603586	0.3447178	6.415837
4301	3.005285	5.0596185	0.77591056	0.34444788	6.1799765
4801	2.2538133	5.046226	0.79472697	0.3442898	6.185242
5201	2.3953006	4.6846232	0.82054806	0.34412572	5.849297

Tabla 4.1 Muestra progreso de métricas durante la fase de entrenamiento

Para apreciar mejor estos resultados en el entrenamiento, iniciamos TensorBoard desde la fase de Testing.

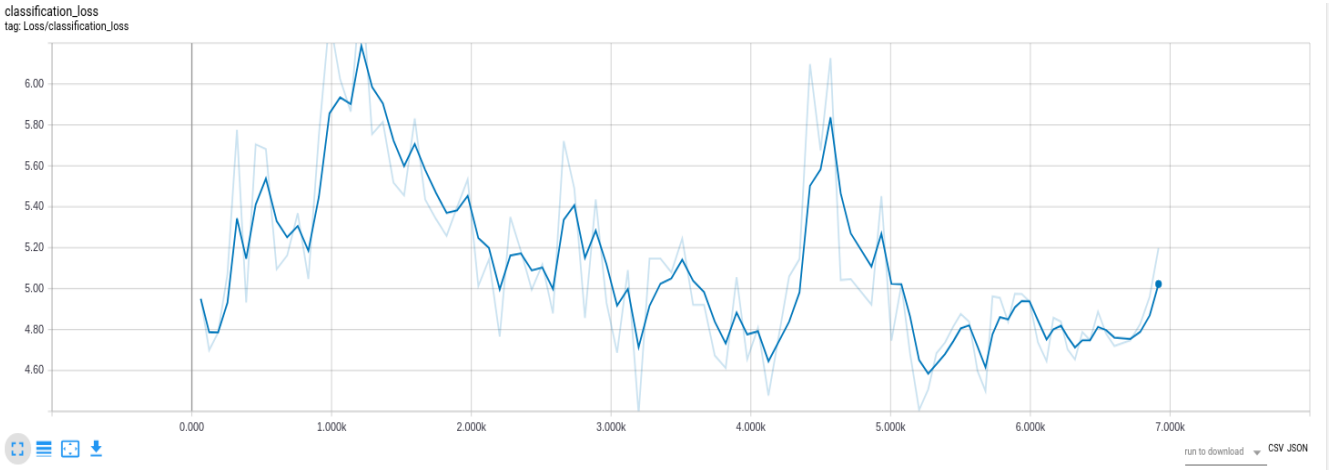


Tabla 4.2 Rendimiento classification loss

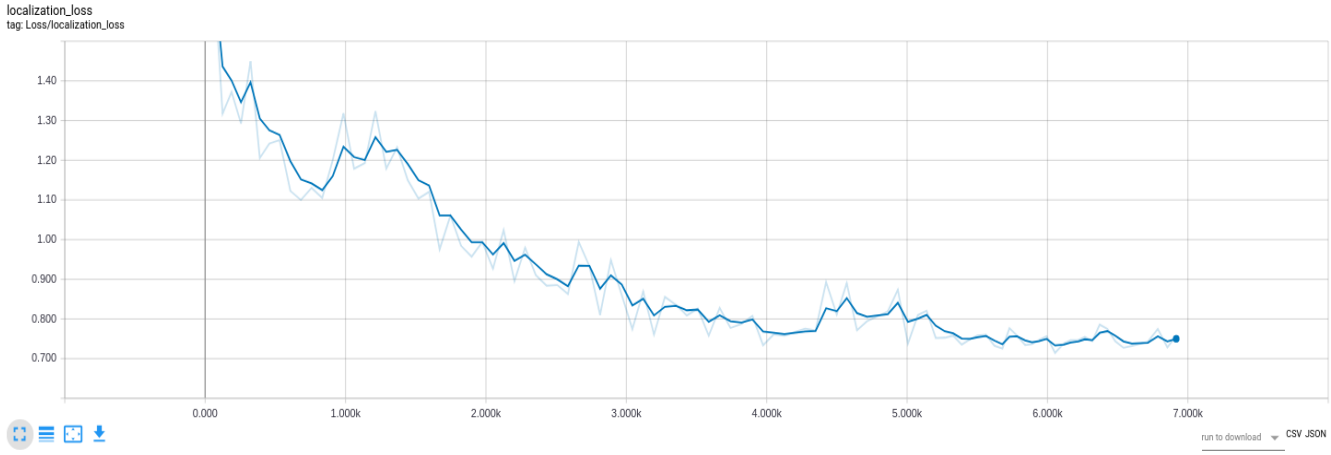


Tabla 4.3 Rendimiento localization loss

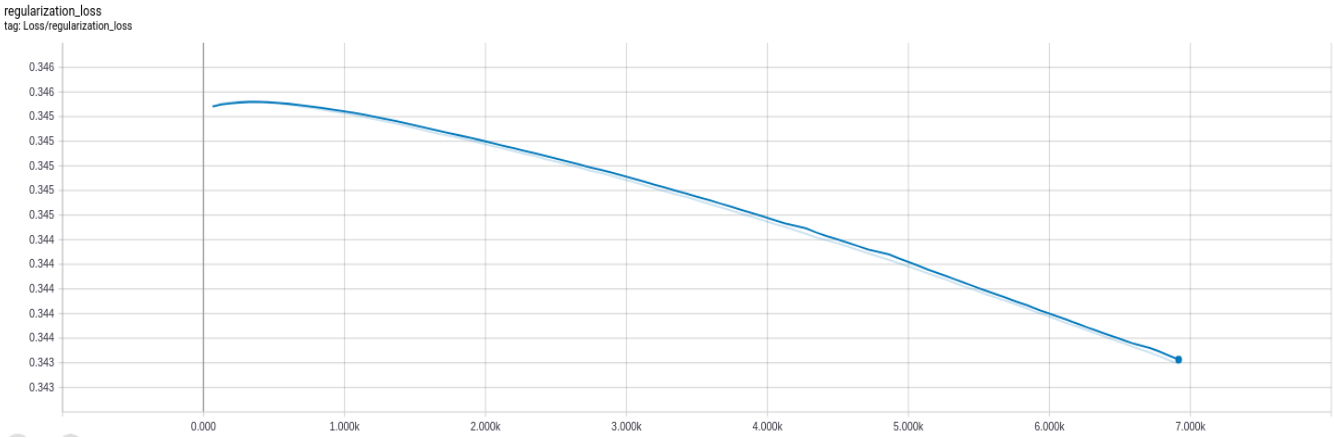


Tabla 4.4 Rendimiento regularization loss

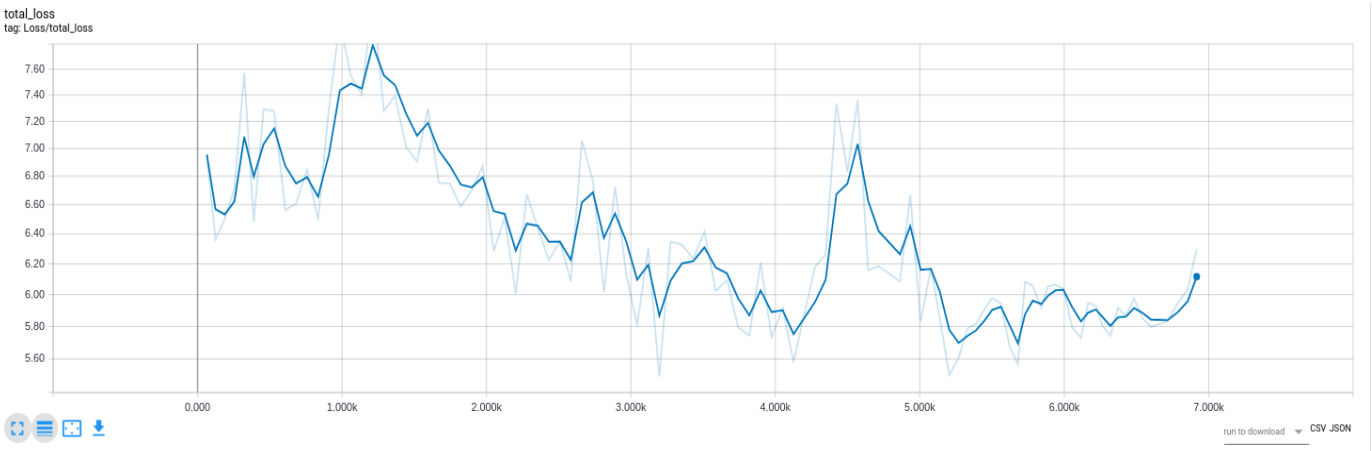


Tabla 4.5 Rendimiento total loss

Los resultados mostrados en la Figura 4.9 se pueden obtener directamente al iniciar TensorBoard en el apartado IMAGES, o es posible hacer inferencia en imágenes seleccionadas manualmente desde el apartado *Crear imagen o video* en la fase de Testing.

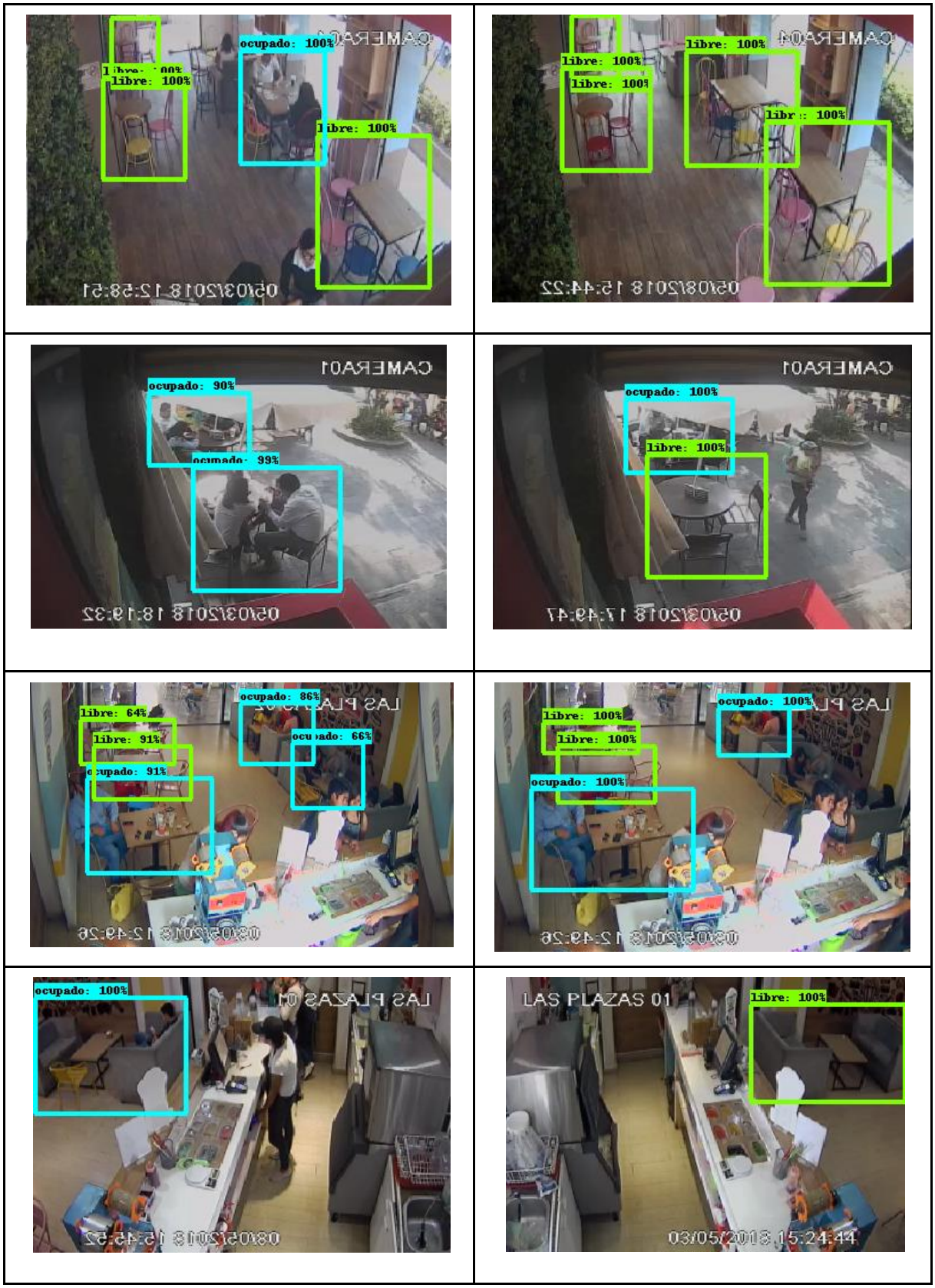


Figura 4.9 Muestra de resultados sobre la inferencia en imágenes

5. Conclusiones

El objetivo principal de esta tesis ha sido el desarrollo de una herramienta de software para el uso, desarrollo ágil y generalizado de estructuras de aprendizaje autónomo (redes neuronales artificiales). Se presentó una estructura específica para resolver problemas de visión por computadora para múltiples modelos cuyo rendimiento ha sido probado exhaustivamente por empresas e instituciones de alto calibre, los cuales han asentado una base sólida en el campo de la investigación. Durante el trayecto de desarrollo se observó el impacto que pueden generar tecnologías de esta clase, no solo para programadores, incluso extender su alcance para usuarios especializados en distintas áreas del conocimiento, que deseen aplicar sus juicios en distintos problemas de forma rápida y sencilla.

Una vez definido el sistema se presentó la solución a un problema en específico, que localizara y detectara instancias del estado de mobiliario, como se ha mostrado a través de las pruebas realizadas. Cabe mencionar que el conjunto de datos de estudio a pesar de obtener resultados aceptables en las inferencias sobre imágenes o video, en ocasiones las predicciones no están del todo balanceadas, esto debido al número de elementos con los que cuenta la base de datos. La base puede balancearse obteniendo un número mayor de datos, así mismo diferentes técnicas usadas pueden ayudar a expandir los datos ya recolectados, generando transformaciones aleatorias: traslación, rotación y zoom, esto podría ayudar a mejorar considerablemente la precisión de los resultados.

También cómo es posible apreciar en la tabla 4.1 el descenso en el error puede subir o bajar durante el entrenamiento (aunque siempre se espera disminuya), este comportamiento cambia dependiendo de la red y los lugares internos del gradiente por los cuales esté recorriendo la red neuronal. Por eso es necesario y labor del usuario crear prototipos y entrenar diferentes versiones de la red, ajustando y experimentando con los parámetros con la intención de generar y conservar los mejores resultados. Así mismo es importante mencionar que la selección de algún tipo de red en específico va de la mano a la aplicación que deseamos realizar, esto con el objetivo de mantener un equilibrio entre velocidad y precisión.

Para concluir, el presente trabajo se encuentra en [32] abierto a la comunidad científica y otros desarrolladores, desde aquí es posible acceder de forma gratuita haciendo uso del código fuente realizado sobre estas tecnologías, todas Open Source.

6. Bibliografía

[1] López, José Francisco. “Empresas Más Grandes Del Mundo 2019.” *Economipedia*, 21 Feb. 2019, economipedia.com/ranking/empresas-mas-grandes-del-mundo-2019.html.

[2] Zhao, et al. “Object Detection with Deep Learning: A Review.” *ArXiv.org*, 16 Apr. 2019, arxiv.org/abs/1807.05511.

[3] Lee, Kai-Fu, and Jan W. Haas. *AI-Superpowers China, Silicon Valley Und Die Neue Weltordnung*. Campus, 2019.

[4]RightNow Technologies. (2011). *Customer Experience Impact Report.* Harris interactive.

[5]Stuart J. Russell. *Inteligencia Artificial Un Enfoque Moderno* Segunda edición. Madrid: Pearson educación, 2004.

[6] Juan Bojórquez Mora, (2011). *Uso de redes neuronales artificiales para estimar la respuesta sísmica de sistemas estructurales* (Maestría). Universidad Nacional Autónoma de México.

[7] Redacción. “4 Misiones Clave Que Tendrá Summit, La Supercomputadora Más Poderosa Del Mundo Que Acaba De Entrar En Funcionamiento - BBC News Mundo.” *BBC News*, BBC, 11 June 2018, www.bbc.com/mundo/noticias-44444567.

[8] Esta Es LA COMPUTADORA MÁS RAPIDA DEL MUNDO En 2019.” *Jungla Moderna*, 20 Feb. 2019, junglamoderna.com/noticias-tecnologia/esta-es-la-computadora-mas-rapida-del-mundo-en-2018/.

[9] “Gradiente.” *Wikipedia*, Wikimedia Foundation, 14 Mar. 2019, es.wikipedia.org/wiki/Gradiente.

[10] “Paul Werbos.” *Wikipedia*, Wikimedia Foundation, 19 Apr. 2019, es.wikipedia.org/wiki/Paul_Werbos.

[11] “CS231n: Convolutional Neural Networks for Visual Recognition.” *Stanford University CS231n: Convolutional Neural Networks for Visual Recognition*, cs231n.stanford.edu/.

[12] Raval, Siraj. “Convolutional Neural Networks - The Math of Intelligence (Week 4).” *YouTube*, YouTube, 12 July 2017, www.youtube.com/watch?v=FTr3n7uBIuE.

[13] Yosinski, et al. “How Transferable Are Features in Deep Neural Networks?” *ArXiv.org*, 6 Nov. 2014, arxiv.org/abs/1411.1792.

[14] Silva Tapia, R. (2017). *Reconocimiento de imágenes en frames de vídeo utilizando redes neuronales (Licenciatura).* Universidad de las Fuerzas Armadas ESPE.

[15] Flores Albino, J. (2018). *Deep Learning para la Detección de Peatones y Vehículos (Maestría).* Universidad Autónoma del Estado de México.

[16] Durán Suárez. (2017). *Redes neuronales convolucionales en R : Reconocimiento de caracteres escritos a mano.* Universidad de Sevilla, Sevilla.

[17] Frédéric Rayar, A. R. (2015). Marker-free Object Recognition on Tabletops: An exploratory study. *10th ACM International Conference on Interactive Tabletops and Surfaces.* Funchal, Madeira, Portugal.

[18] Giovanni Taverriti, S. L. (2015). *Real-time Wearable Computer Vision System for Improved.* Università degli Studi di Firenze, Media Integration and Communication Center. Obtenido de https://www.micc.unifi.it/wp-content/uploads/2016/10/p703-taverriti.pdf

[19] Gonzalez, J. R. (2018). HERMES: A research project on human sequence evaluation. Obtenido de https://upcommons.upc.edu/handle/2117/2763

[20]Harry Pratta, F. C. (2016). Convolutional Neural Networks for Diabetic Retinopathy. *Elsevier B.V*, 6.

[21]“Comparison of AI Frameworks.” *Skymind*, skymind.ai/wiki/comparison-frameworks-dl4j-tensorflow-pytorch.

[22] Tensorflow. “Tensorflow/Models.” *GitHub*, 24 Apr. 2019, github.com/tensorflow/models/tree/master/research/object_detection.

[23] “Anaconda Python/R Distribution.” *Anaconda*, www.anaconda.com/distribution/.

[24] Tensorflow. “Tensorflow/Models.” *GitHub*, github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md.

[25] “CUDA GPUs.” *NVIDIA Developer*, 1 May 2019, developer.nvidia.com/cuda-gpus.

[26] *Qt Designer Manual*, doc-snapshots.qt.io/qt5-5.9/qtdesigner-manual.html.

[27] Huang, Jonathan, et al. “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors.” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, doi:10.1109/cvpr.2017.351.

[28] Tensorflow. “Tensorflow/Models.” *GitHub*, github.com/tensorflow/models/blob/master/research/object_detection/g3doc/configuring_jobs.md.

[29] Tensorflow. “Tensorflow/Models.” *GitHub*, github.com/tensorflow/models/blob/master/research/object_detection/object_detection_tutorial.ipynb

[30] Liu, Wei, et al. “SSD: Single Shot MultiBox Detector.” *Computer Vision – ECCV 2016 Lecture Notes in Computer Science*, 2016, pp. 21–37., doi:10.1007/978-3-319-46448-0_2.

[31] Scott, et al. “Scalable, High-Quality Object Detection.” *ArXiv.org*, 9 Dec. 2015, arxiv.org/abs/1412.1441.

[32] Gustavo Bañuelos Ochoa. “mrguz170/LKE.” *GitHub*, 31 Mar. 2019, github.com/mrguz170/LKE.