Visión artificial para el reconocimiento automático, en tiempo real, de líneas urbanas de autobuses

Maina, Hernán Javier
Director: Sánchez, Jorge A.

Trabajo especial de la carrera
Licenciatura en Ciencias de la Computación

Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba
Argentina, 2019

“A mi madre Silvana, por sus consejos,
por su compañía en todos los momentos difíciles,
y por dejarme gran parte de lo que hoy soy.”

Agradecimientos

A mi familia, en especial a mi padre y a mi novia, por su confianza y apoyo incondicional a lo largo de todo el trayecto.

A mi director Jorge, quien con su tiempo y experiencia me brindó la posibilidad de culminar esta última etapa.

Y a todas las personas que de una u otra forma contribuyeron en la realización de este trabajo.

Resumen

En el presente trabajo, se aborda el problema de la detección y el reconocimiento de números de líneas de autobuses del transporte público de pasajeros de la ciudad de Córdoba, empleando imágenes obtenidas mediante dispositivos móviles estándar. El objetivo del mismo es la exploración de técnicas de visión por computadoras y análisis de imágenes, para la generación de herramientas que permitan asistir a personas con algún tipo de impedimento visual. A tal fin, se presenta y evalúa una arquitectura modular basada en detectores de objetos y reconocimiento óptico de caracteres, especialmente adaptada a las particularidades del problema.

Palabras claves: Inteligencia artificial, visión por computadoras, aprendizaje automático, redes neuronales convolucionales, detección de objetos, reconocimiento de texto en escenas, YOLO, EAST, OCR, Tesseract, autobuses, impedimento visual.

Abstract

In the present work, the problem of the detection and the recognition of bus line numbers of public transport in the city of Córdoba is addressed, using images obtained by standard mobile devices. The goal of this project is the exploration of computer vision techniques and the analysis of images, for the generation of tools that allows people with some type of visual impairments to be assisted. To achieve this, a modular architecture based on object detectors and optical character recognizers is presented and evaluated, especially adapted to the particularities of the problem.

Keywords: Artificial intelligence, computer vision, machine learning, convolutional neural networks, object detection, scenes text recognition, YOLO, EAST, OCR, Tesseract, buses, visual impairment.

Índice

Parte I
Introducción

En la actualidad, millones de personas viven en el mundo con dificultades para comprender el medio ambiente que las rodea, debido a algún tipo de impedimento visual. Por impedimento visual, también conocido como ceguera o pérdida de visión, se conoce a una diversidad funcional de tipo sensorial que refiere a la perdida parcial o total de la visión, a tal grado, que los problemas causados, no son solucionables mediante los métodos habituales como el uso de anteojos u otros dispositivos visuales.

Según datos recientes de la Organización Mundial de la Salud, se estima, que alrededor de 1.3 billones de personas en el mundo viven con algún tipo de discapacidad visual, entre las cuales: 188.5 millones de personas categorizan en un impedimento de tipo leve, 217 millones de moderado a severa y 36 millones de personas, poseen una perdida total de la misma.

Como en cualquier otra discapacidad, vivir con restricciones representa un constante desafío; en particular, la falta de agudeza visual, hace que las situaciones de la vida diaria se tornen mucho mas difíciles de realizar. Si bien pueden desarrollarse enfoques alternativos para lidiar con tales rutinas, una de las consecuencias inmediatas de esta pérdida, se ve reflejada en la sensación de inseguridad que se experimenta, al moverse o viajar de forma independiente, principalmente en ambientes exteriores y no conocidos.

A lo largo de los años, muchas técnicas y habilidades dirigidas al campo de la orientación y la movilidad, han sido creadas para intentar afrontar tales problemas. Desde el uso de bastones, perros guía, suelos texturados y entrenamientos de movilidad, hasta métodos más avanzados, como el uso de identificación de obstáculos por radiofrecuencia (RFID) [1], sistema de posicionamiento global (GPS) [2], diodos emisores de luz infrarroja [3] y sensores inalámbricos [4]; fueron algunas de las opciones destinadas a ayudar a estas personas, para que puedan manejarse de manera mas segura. No obstante, la gran capacidad de procesamiento que poseen los dispositivos actuales y las tecnologías como lectores, lupas y pantallas braille actualizables, permitieron un acceso a la información, que previo a esto, no hubiese sido posible.

En los últimos años, dado a los grandes avances en la disciplina de la visión por computadora, especialmente en lo que respecta a redes neuronales convolucionales y aprendizaje profundo, se han ido proponiendo muchos sistemas y aplicaciones con el mismo objetivo.

Decenas de herramientas basadas en visión artificial, fueron desarrolladas haciendo uso de las capacidades de las cámaras de los teléfonos móviles. Desde aplicaciones como *TapTapSee*[1], que permite describir una imagen capturada en pocos segundos; y los desarrollos realizados en los últimos años por empresas como Facebook y Twitter, las cuales, mediante tecnología de subtitulado de imágenes están logrando que sus contenidos sean cada vez mas inclusivos; son algunos de los ejemplos que intentan expandir y adaptar el mundo de las redes sociales y los contenidos visuales.

Por otra parte, sistemas basados en la eco localización y la utilización de sonido 3D [5], como experimentos extremos de tecnología como *vOICe*[2] [6], que intenta convertir el sentido visual en auditivo, asociando la altura con el tono, y el brillo con el volumen; fueron propuestos para ofrecer una mejor comprensión del entorno circundante, pero a costa de una curva de aprendizaje considerable.

El presente proyecto, fue desarrollado y basado en una intensa etapa de investigación. Ésta, tuvo como objetivo la interiorización y la comprensión, tanto de las principales problemáticas que enfrentan las personas con restricciones visuales, como de las herramientas y métodos que utilizan para solventarlas. En base a esto, se trato de discernir un problema tangible, cuya solución no implique el desplazamiento de las técnicas que habitualmente utilizan en su vida diaria, sino, sea una herramienta más que complemente y brinde información extra.

Muchos de los sistemas existentes, algunos de los cuales ya fueron nombrados, se basan en la generación artificial de instrucciones audibles, las cuales, mediante auriculares o altavoces paneados de izquierda a derecha, alertan o anticipan sobre posibles objetos o localizaciones, dando una devolución continua e intermitente del estado general de la situación. Éstos, y a pesar de que algunos tengan muy buena efectividad, olvidan algo fundamental:

> *"cuando un sentido es oprimido, los demás sentidos se intensifican para suplir esa pérdida";*

[1] `https://taptapseeapp.com`
[2] `https://www.seeingwithsound.com`

en este caso, el oído, herramienta primordial de toda persona con problemas de visión, estaría siendo obstruido por una sobrecarga de información, dificultando la comprensión del entorno cercano e impidiendo sortear potenciales situaciones de riesgo.

Entrevistas con varias personas y asociaciones afines, fueron la clave para comprender y sentar el enfoque del siguiente proyecto. *Una persona no es un objeto*, y si un buen rendimiento conlleva a una sobrecarga de dispositivos y sensores, se estaría reemplazando un paquete de problemas por otro, sustituyendo la humanidad de tal persona por un objeto ejecutor de instrucciones.

En base a esto, el trabajo, intentará explotar las posibilidades que hoy nos entregan las redes convolucionales profundas y la era del aprendizaje automático en materia de detección de objetos y reconocimiento de texto, mediante su aplicación a una solución coloquial, por medio de un algoritmo que se encargue de automatizar la tarea de *reconocimiento de líneas de autobuses*, en pos de facilitar el acceso al transporte urbano de la ciudad de Córdoba y potencialmente ayudar a personas que posean tales dificultades.

Parte II
Marco Teórico

En esta parte inicial del trabajo, se darán un conjunto de definiciones teóricas, necesarias para comprender los fundamentos subyacentes de los módulos que integrarán el algoritmo propuesto. Partiendo de conceptos principales, referidos al campo de la inteligencia artificial y relacionados, se detallarán diferentes técnicas de aprendizaje automático, que sentarán las bases para poder abordar y solucionar las distintas aristas que toca este proyecto. Por último, y mediante un desarrollo progresivo, se introducirán los distintos métodos y modelos, que de forma constructiva, culminarán por explicar como desarrollar un modelo de detección de objetos y porque es tan complicado realizar un reconocimiento de texto en entornos naturales.

1. Inteligencia Artificial y otros conceptos

La *Inteligencia Artificial* (IA), nació en los años 50 de la mano de algunos pioneros del naciente campo de las ciencias de la computación. Éstos, comenzaron a preguntarse como podían lograr que las maquinas "pensaran" por si mismas. Básicamente, una concisa definición de IA sería: *el campo de la ciencias de la computación que intenta automatizar tareas intelectuales, que aparentemente requerirían, de una inteligencia a nivel humano para poder ser realizadas.*

Alrededor de 1840 y 1850, Charles Babbage, invento el *«Motor Analítico»*. Éste, fue el primer computador mecánico de propósito general que asistía a matemáticos a automatizar el computo de ciertas operaciones en el campo del análisis matemático; sin embargo, no se tenían las pretensiones de originar algo nuevo, sino, de ayudar en lo que ya se sabía hacer. Fue luego, cuando en 1950, Alan Tuning, introdujo el famoso *«Test de Turing»*, llegando a la conclusión de que los computadores de propósito general, podrían ser capaces de "aprender" y "ser originales". Esto, sumado a preguntas como:

> *¿podría una computadora ir mas allá de lo que se le ha programado realizar? ¿podría ejecutar tareas sin supervisión humana por si misma? ¿podría aprender patrones o reglas automáticamente en base a datos ingresados?*

culminaron en el nacimiento de un nuevo paradigma de programación llamado *«Aprendizaje Automático»* (en *inglés*, Machine Learning).

1.1. ¿ A que llamamos Aprendizaje Automático?

El *aprendizaje automático,* es el campo de la inteligencia artificial que tiene como objetivo otorgarle a las máquinas la capacidad de aprender de la experiencia previa y de tomar sus propias decisiones. A diferencia de los programas informáticos tradicionales, éstos no se limitan a una capacidad previamente definida, sino, que aumentan sus conocimientos por si mismos.

Un algoritmo de aprendizaje automático es *"entrenado"* en lugar de ser *"programado"*. Tal entrenamiento, consiste en "mostrarle" al algoritmo muchos ejemplos para una tarea específica con el fin de que encuentre estructuras y patrones estadísticos, y eventualmente comprenda reglas que sirvan para automatizar dicha tarea.

Ya en el siglo XXI, aproximadamente a fines de 2010, con el desarrollo de las capacidades de computación, el aumento a la accesibilidad de los datos y utilizando los conocimientos del aprendizaje automático, nace una nueva subárea de estudio asociada, llamada *«Aprendizaje Profundo»* o (en *inglés*, Deep Learning).

1.2. ¿ Que se entiende por Aprendizaje Profundo?

En concepto, el *aprendizaje profundo* es muy similar al aprendizaje automático pero usa algoritmos diferentes. Mientras que este último trabaja con algoritmos de regresión o con árboles de decisión, el aprendizaje profundo usa algoritmos de alto nivel que trabajan con abstracciones de datos en forma matricial o tensorial, que admiten transformaciones no lineales e iterativas como las *redes neuronales artificiales,* Sección (2.3), que funcionan de forma muy parecida a las conexiones neuronales biológicas de nuestro cerebro.

1.3. Tipos de aprendizaje

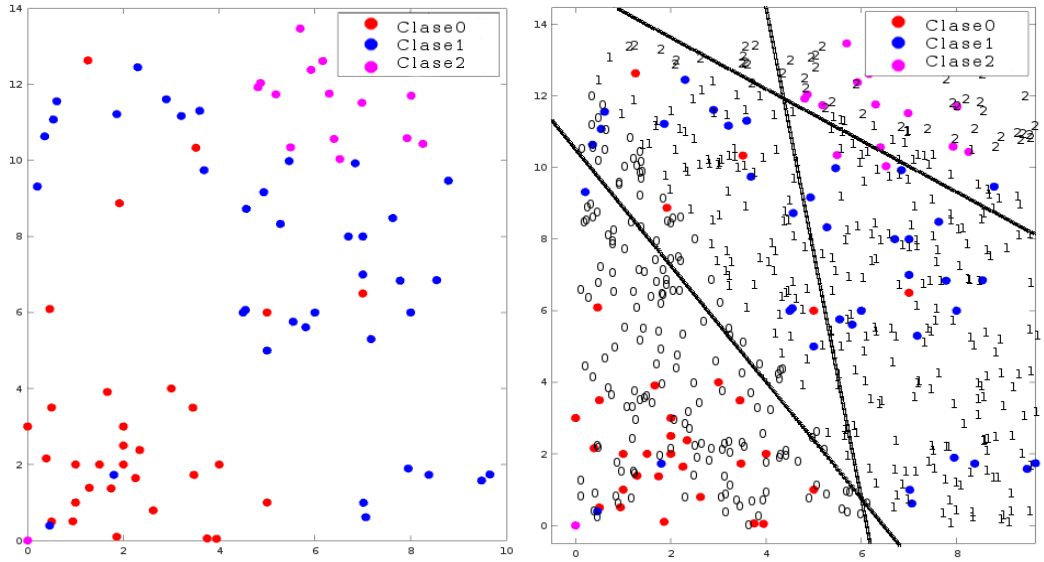
Dependiendo de las necesidades del problema, podemos encontrar distintos tipos de algoritmos de aprendizaje, agrupados en alguna de las siguientes dos categorías principales: *«Aprendizaje Supervisado»* y *«Aprendizaje No Supervisado».*

1.3.1. Aprendizaje supervisado

Los algoritmos de *aprendizaje supervisado,* generan un modelo predictivo basado en datos de entrada y salida. La palabra "supervisado", viene de tener un conjunto de datos previamente etiquetados y clasificados llamados *«conjunto de entrenamiento»* o «*conjunto de entrada*», a partir de los cuales se realizá el ajuste del modelo. De esta manera, el algoritmo va "aprendiendo" a clasificar las muestras de entrada, mediante la comparación entre su predicción y la etiqueta real de las mismas, realizando las compensaciones respectivas al modelo de acuerdo al error en la estimación del resultado. Un ejemplo de este tipo de aprendizaje puede ser visto en la Figura (1.1).

Algunos métodos y algoritmos de esta categoría son los siguientes:

1. K vecinos más próximos (K-nearest neighbors).
2. Redes neuronales artificiales (Artificial neural networks).
3. Máquinas de vectores de soporte (Support vector machines).
4. Clasificador Bayesiano ingenuo (Naïve Bayes classifier).
5. Árboles de decisión (Decision trees).
6. Regresión logística (Logistic regression).



(a) Conjunto de entrenamiento previamente etiquetado.

(b) Predicción de modelo sobre 500 puntos aleatorios.

Figura 1.1: Ejemplo de regresión logística múltiple.

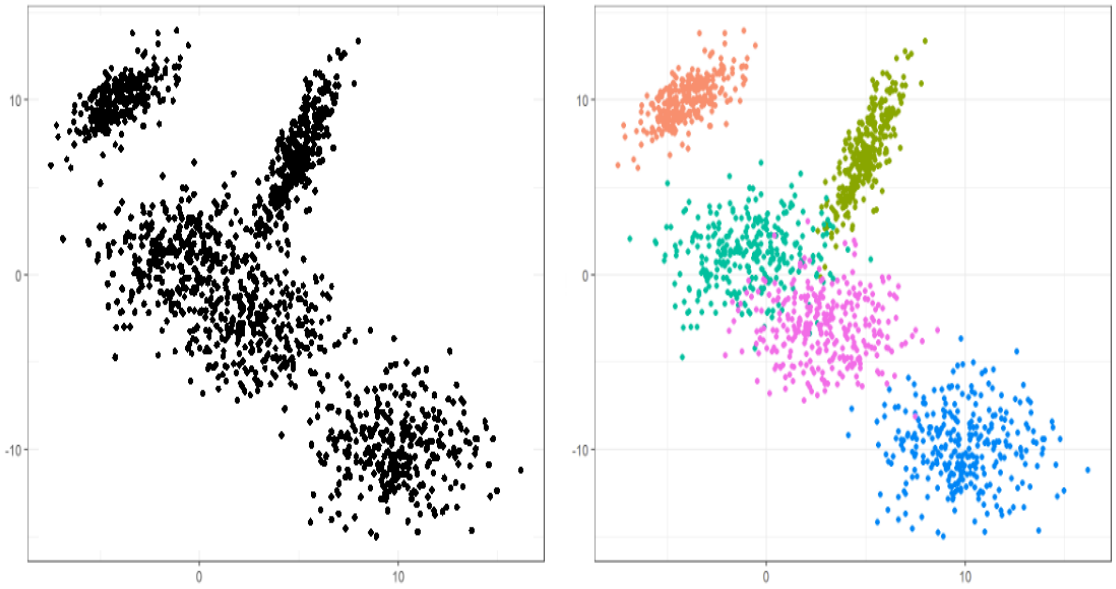
1.3.2. Aprendizaje no supervisado

Los algoritmos *no supervisados,* trabajan de manera muy similar a los de aprendizaje supervisado, pero a diferencia de ellos, ajustan el modelo predictivo tomando en cuenta solo los datos de entrada sin reparar en los de salida. Es decir, no es necesario que los datos de entrada estén clasificados ni etiquetados para entrenar el modelo.

Su estrategia básicamente radica en particionar y agrupar al conjunto de datos de entrada, basándose en similitudes de sus características (clusterización). Un ejemplo que ilustra tal procedimiento, puede ser visualizado en la Figura (1.2).

Algunos de los principales algoritmos no supervisados se listan a continuación:

1. K-medias (K-means).
2. Mezcla de Gaussianas (Gaussian mixtures).
3. Agrupamiento jerárquico (Hierarchical clustering).
4. Mapas auto-organizados (Self-organizing maps).



(a) Conjunto de datos de entrenamiento. (b) Datos etiquetados en diferentes grupos.

Figura 1.2: Ejemplo de clusterización mediante K-means.

1.4. ¿Que es la Visión Artificial?

La visión por computadoras o *«visión artificial»,* refiere al campo interdisciplinario que estudia y busca desarrollar técnicas para ayudar a que las computadoras o maquinas, estén "visualmente habilitadas". Es decir, es la ciencia que trata como lograr que las computadoras obtengan, partiendo de una imagen o video digital, un alto nivel de entendimiento, y de esta manera pueda imitar el sistema de visión humana y automatizar las tareas que este puede realizar.

2. Aproximación al reconocimiento de objetos en imágenes

2.1. Espacio de color

La elección de un «*espacio de color*» determinado, es una tarea muy importante en el campo de la visión artificial. Aplicaciones para la compresión de imágenes, la detección de objetos o seguimiento de los mismos, invierten gran parte del tiempo de desarrollo a tal selección, sin embargo, es bastante difícil definir un espacio de color universal, ya que puede modelarse de muchas maneras.

> *Un **espacio de color**, es un modelo matemático abstracto que describe la forma en la que los colores pueden representarse como tuplas de números, normalmente como tres o cuatro valores o componentes de color, con el cual se intenta describir la percepción humana que se conoce como color.*

La comunidad de visión por computadora, a lo largo de los años, ha realizado numerosas investigaciones para tratar de determinar que espacios son los más adecuados para cada aplicación. Es tanto el campo aplicativo, que propuestas para mejorar la segmentación de objetos y la segmentación biológica de imágenes [7], como análisis comparativos para determinar los espacios más óptimos en la tarea de detección de piel [8], reconocimiento de semáforos y clasificaciones textiles, fueron objetivo de tales estudios. En robótica por ejemplo, investigaciones relacionadas a aplicaciones para el reconocimiento de objetos de colores, en el contexto del fútbol robótico [9], terminaron concluyendo que el uso del espacio de color *rSc2,* podía logran una tasa de reconocimiento más alta en comparación con otros como *YUV* o *HSL*, y de esta manera aumentar la importancia de ciertas regiones dando prioridad de búsqueda para que el robot revise primero tales áreas.

2.1.1. Percepciones y propiedades

Sin entrar en detalles en los modelos matemáticos que rigen cada espacio de color, a continuación, se explicará cual es la intuición detrás de su utilización. Mediante la presentación de algunos de los espacios mas utilizados en la visión artificial, esta idea, será desarrollada a través de la comparación de dos escenarios de iluminación variable; uno, en condiciones exteriores de luz solar brillante, y otro, en condiciones interiores de iluminación normal, sobre una imagen que presenta una permutación aleatoria del famoso *«Cubo Mágico»*, Figura (2.1). Ésto, permitirá tener una primera aproximación de las diferentes percepciones producidas por las distintas transformaciones de espacios de colores, y en base a esto, tomar decisiones de implementación a futuro, que ayuden a seleccionar el conjunto de mejores espacios, que faciliten la segmentación de los números de línea de los autobuses.

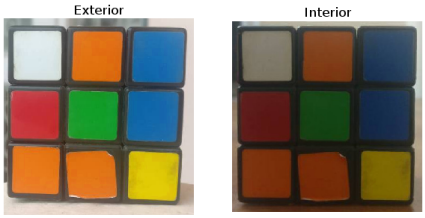


Figura 2.1: Imágenes de misma permutación tomada bajo diferentes iluminaciones (fuente: [10]).

2.1.1.1. Espacio de color RGB

El espacio de color RGB tiene la propiedad de ser un espacio de color aditivo, donde los colores se obtienen mediante una combinación lineal de valores rojo (del *inglés*, Red), verde (del *inglés*, Green) y azul (del *inglés*, Blue), y además, los tres canales están correlacionados por la cantidad de luz que golpea en su superficie.

Además de las claras diferencias generales, observables en los valores de las dos imágenes y sus descomposiciones en la Figura (2.2), particularmente, en el canal azul (B), se puede ver que las piezas azules y blancas se perciben similares en condiciones de iluminación interior, pero presentan una clara diferencia bajo iluminación exterior. Esta falta de uniformidad, hace que la segmentación

basada en el color sea muy difícil de aplicar mediante la utilización de este espacio, principalmente, debido a que representa los colores codificando información adicional relacionada a la luminosidad.

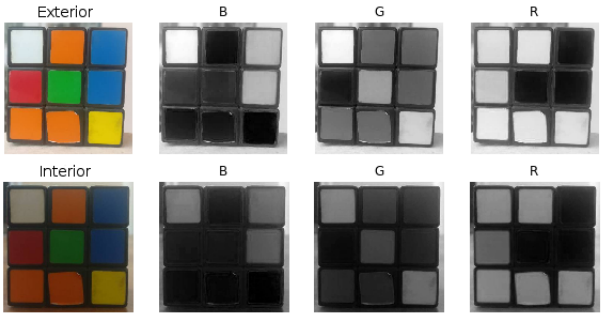


Figura 2.2: Visualización de los canales Azul (B), Verde (G), Rojo (R), del espacio de color RGB.

2.1.1.2. Espacio de color LAB

El espacio de color Lab tiene tres componentes: (L) luminosidad o intensidad de luz, (A) componente de color que varía de verde a magenta y (B) componente de color que va del azul al amarillo.

Lab, es bastante diferente del espacio de color RGB, en el cual como anteriormente fue descripto, la información del color es separada en tres canales que también codifican información de brillo. En el espacio de color Lab, solo codifica el brillo el canal L, independizándose de los otros dos, encargados de codificar el color. Todo ello, hace de este, un espacio perceptualmente uniforme, capaz de proporcionar una aproximación muy certera de como el ojo humano percibe al color, y ser independiente del dispositivo de captura.

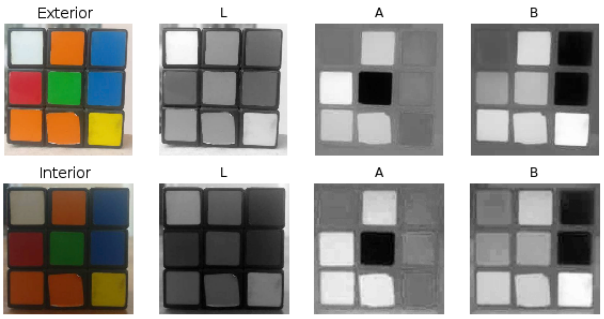


Figura 2.3: Luminosidad (L), y componentes de color (A) y (B), del espacio de color LAB.

En la Figura (2.3), puede observarse claramente, que el cambio de iluminación afecta principalmente al componente L, mientras que los componentes A y B, que contienen la información de color, no sufren cambios masivos.

Algo llamativo y digno de destacar, es que los valores extremos del componente A, representados por los colores Verde, Naranja y Rojo, se mantienen también invariantes en la componente B. De manera similar, los colores Azul y Amarillo, que son extremos en la componente B, tampoco presentan cambios en la componete A.

2.1.1.3. Espacio de color YCrCb

El espacio de color YCrCb se deriva del espacio RGB, y tiene las siguientes tres componentes:

1. (Y) componente de Luminancia o Luma, obtenido de RGB después de una corrección gamma.

2. (Cr) qué tan lejos está el componente Rojo en relación a la Luma, ie. (Cr = R - Y).
3. (Cb) a qué distancia está el componente Azul en relación a la Luma, ie. (Cb = B - Y).

La principal propiedad que caracteriza a este espacio de color, es que separa los componentes de *luminancia* (información relacionada a la intensidad) y *crominancia* (información relacionada al color) en diferentes canales, al contrario de lo que sucedía con RGB el cual presentaba una mezcla de ambas.

A continuación, en la Figura (2.4), se muestra como antes, las dos imágenes en el espacio de color YCrCb separadas en sus canales. Aquí, se pueden hacer observaciones similares a las realizadas en LAB en lo que respecta a componentes de intensidad y color en relación a los cambios de iluminación. Mientras que en la imagen con iluminación exterior, las diferencias de percepción entre los colores Rojo y Naranja son incluso menores que las presentadas por el espacio LAB, el color Blanco sufre cambios en todos sus canales.

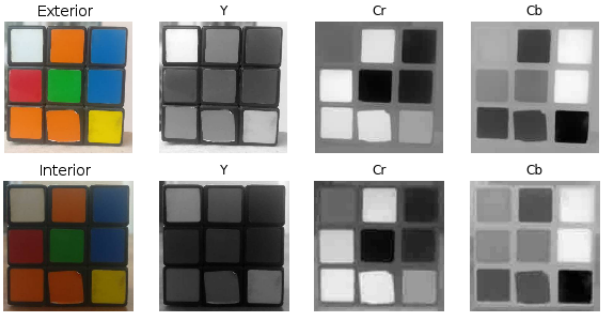


Figura 2.4: Luminancia (Y), y componentes de crominancia (Cr) y (Cb), del espacio de color YCrCb.

2.1.1.4. Espacio de color HSV

La característica principal del espacio de color HSV es que usa solo un canal para describir el color, el H. Esto lo hace muy útil e intuitivo a la hora de especificar de que color se esta hablando. Por otra parte, y a diferencia de LAB, este espacio es dependiente del dispositivo de captura.

HSV posee las siguientes tres componentes: (H) Hue (longitud de onda dominante), (S) saturación (pureza o matices del color) y (V) intensidad.

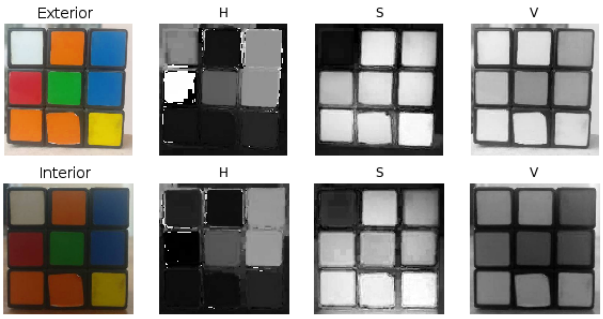


Figura 2.5: Componentes Hue (H), Saturación (S) e Intensidad (V), del espacio de colo HSV.

En base a la Figura (2.5), podemos realizar las siguientes observaciones:

- La componente H es muy similar en ambas imágenes, lo que indica que incluso bajo cambios de iluminación, la información de color permanece intacta, al igual que se observa en la componente S.
- V, al captura la intensidad de luz que cae sobre él, modifica sus valores acorde a los cambios de iluminación.

- Por último, se puede observar una dramática diferencia de los valores de las piezas Rojas entre la imagen exterior e interior. Esto, se debe a que Hue se representa como un círculo, por lo que se dispone de 360^o que se dividen en los 3 colores RGB, dando un total de 120^o grados por color. Para el caso del color Rojo, este, forma parte del ángulo inicial, por lo que puede tomar valores entre $[300^o, 360^o]$ (colores brillantes) o bien entre $[0^o, 60^o]$ (colores oscuros), de allí la diferencia observada.

2.2. Clasificación de imágenes

El reconocimiento o *clasificación de imágenes,* es uno de los problemas centrales en el área de la visión artificial ya que muchas de las técnicas utilizadas en este campo pueden reducirse a esta tarea.

Por clasificación de imágenes, se entiende a la acción de asignar, a una determinada imagen de entrada, una etiqueta (o clase) de entre un conjunto fijo de categorías.

Aunque la tarea de reconocer un concepto visual es relativamente trivial para un humano, hay que tener en cuenta, que para una computadora, una imagen se representa como una gran matriz tridimensional de números $[w, h, c]$, donde el primer y segundo parámetro (w y h) representan el ancho y alto respectivos de la imagen en pixeles, y el tercer parámetro c, el número de canales, Figura (2.6). Por lo tanto, si tenemos $c = 3$, representando a los canales Rojo, Verde y Azul, la imagen RGB resultante constaría de $w \times h \times 3$ valores, donde cada uno de ellos es un entero en el rango $[0, 255]$ variando desde el Negro hacia el Blanco respectivamente.

Sumada a la dificultad descripta anteriormente, debe considerarse que factores como la escala, condiciones de iluminación, deformaciones y el ocultamiento parcial de objetos, hacen de la tarea de clasificación, un procedimiento mucho mas desafiante.

Básicamente, un algoritmo de clasificación consta de tres etapas fundamentales: creación de conjunto de datos de entrada, entrenamiento de modelo y evaluación.

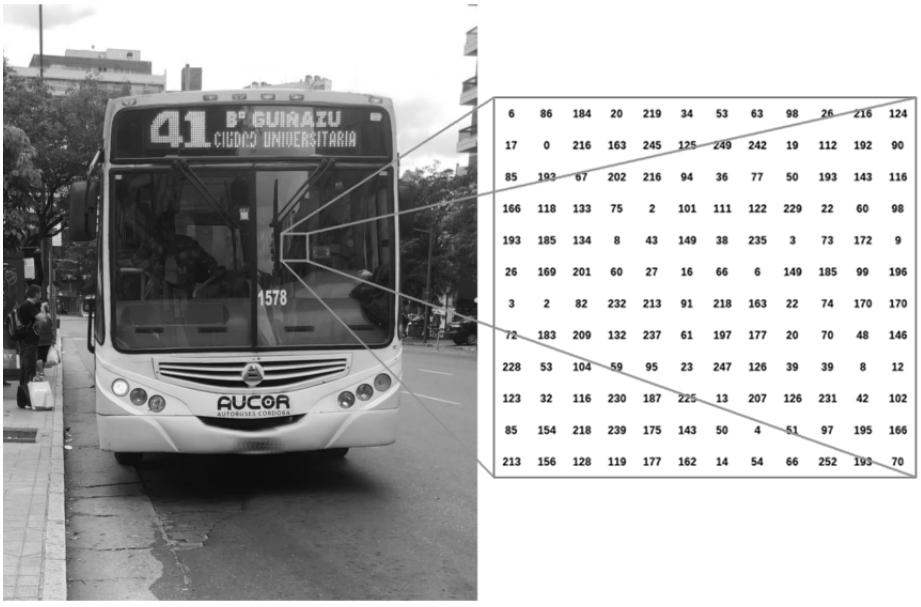


Figura 2.6: Región de imagen digital, en escala de grises, expresada como matriz 2D (valores de pixeles simulados).

2.2.1. Creación de conjunto de entrada

Como primera instancia, se debe crear un *«conjunto de datos», el cual* consiste de m tuplas de *(imagen, etiqueta)*, de las cuales nuestro algoritmo tomará como referencia para "aprender" a clasificar.

Cada imagen será representada como un vector $x^{(i)} \epsilon R^D$ de dimensión $D = w \times h \times 3$, y estará asociado a una etiqueta $y^{(i)} \epsilon \{1, ..., k\}$, donde k indica el número de categorías e $i = 1, ..., m$, la i-ésima imagen del conjunto de entrada.

2.2.2. Entrenamiento y evaluación

El objetivo principal de un clasificador es aproximar (detallado posteriormente), una función no conocida $f : R^D \rightarrow \{1, ..., k\}$ la cual mapea pixeles de una imagen a una categoría, mediante una nueva función $h : R^D \rightarrow R^k$, a la que llamaremos *«función hipótesis»,* la cual tomará una imagen $x^{(i)}$ como entrada y retornará un vector $[c_1, c_2, ..., c_k]$ de dimensión k, con un valor de certeza para cada una de las categorías, donde un alto valor c_j, es indicativo de que la imagen $x^{(i)}$ posee mayor probabilidad de pertenencia a la clase j. A esta etapa, en donde se construye y ajusta la nueva función de predicción, se la conoce como *entrenamiento de modelo.*

Una de las funciones de *hipótesis* mas simples y conocidas, es la siguiente función lineal:

$$h_{W,b}(x^{(i)}) = Wx^{(i)} + b \tag{2.1}$$

En la Ecuación (2.1), $x^{(i)}$ representa el vector de pixeles de dimensión $(D \times 1)$, mientras que W y b son parámetros de nuestra función h de dimensiones $(k \times D)$ y $(k \times 1)$ respectivamente. La Figura (2.7), ilustra la interpretación gráfica de como se vería un clasificador lineal de $k = 3$ clases.

Tanto los valores de W, frecuentemente llamados **pesos**, como los del vector b, llamado **bias** o sesgo, son elegidos de manera tal que nuestra función h tenga la capacidad de predecir la clase correcta en la mayor cantidad de tuplas $(x^{(i)}, y^{(i)})$ de nuestro *«conjunto de entrenamiento»*. Se denomina *conjunto de entrenamiento,* al subconjunto de las tuplas del conjunto de entrada (generalmente un 70 %) que es utilizado para entrenar el modelo. El 30 % restante, es llamado *«conjunto de evaluación»* y se utiliza para medir la precisión de la función de hipótesis, ya que al ser un conjunto nunca antes visto por nuestro modelo, arrojará un resultado menos sesgado y mas representativo.

Uno de los métodos mas comunes para elegir los parámetros W y b, es mediante la minimización de una función llamada *«función de costo* o *pérdida»*.

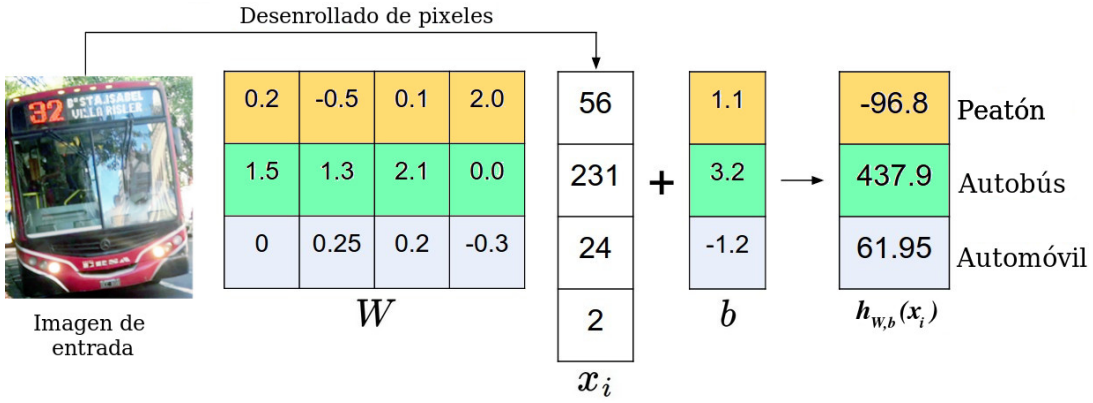


Figura 2.7: Ejemplo de predicción de puntajes para las clases 'peatón', 'autobús' y 'automóvil', mediante un *"clasificador lineal"*. Se asume imagen formada por solo 4 pixeles (referencia: [11]).

Función de costo

$$J(W, b) = \frac{1}{2m} \sum_{i=1}^{m} (h_{W,b}(x^{(i)}) - y^{(i)})^2 \tag{2.2}$$

Esta función (2.2), mide la precisión de nuestra hipótesis, mediante el calculo del ***error cuadrático medio*** entre: las predicciones realizadas por h y los valores de verdad $y^{(i)}$, de cada imagen $x^{(i)}$ en nuestro conjunto de entrenamiento. Por lo tanto, al minimizar J estaremos reduciendo el error de estimación de h y lograremos que $h_{W,b}(x^{(i)}) \simeq f(x^{(i)})$ (se aproxime) para cada $i = 1, ..., m$.

Para lograr tal minimización, existe una técnica denominada ***descenso de gradiente***. Ésta, mediante la actualizando iterativa de los parámetros W y b, permitirá lograr la convergencia de la función de costos J a un mínimo valor (idealmente, su mínimo global).

Descenso de gradiente

Antes de continuar y por comodidad para las siguientes explicaciones, realizaremos una simplificación y representaremos los dos parámetros W y b en un único parámetro W; por lo tanto $W = [w_1, ..., w_n]$.

Sea $\nabla J(W)$ el gradiente de J en W, donde $\nabla J(W) = (\frac{\partial J(W)}{\partial w_1}, ..., \frac{\partial J(W)}{\partial w_n})$ es el vector de las derivadas parciales, el algoritmo comenzará a iterar con algún valor fijo de W. En cada iteración, se determinará $\nabla J(W)$ en relación al punto actual y nos indicará la dirección de mayor descenso hacia la cual debemos movernos para hacer que el valor de la función $J(W)$ disminuya. La magnitud de cada paso que se realice, será determinado por α según:

$$w_j = w_j - \alpha \frac{\partial J(W)}{\partial w_j} \tag{2.3}$$

donde $j = 1, .., n$ representa el índice de cada parámetro W, y α *es el «coeficiente de aprendizaje»*. Cabe aclarar que la actualización de los parámetros se debe realizar en simultáneo.

Intuitivamente, en cada paso, dado a que siempre $\alpha > 0$, el valor de cada parámetro w_j disminuirá y eventualmente la función de costos J convergerá a su valor mínimo en T iteraciones. A continuación, se detalla el pseudocódigo del algoritmo completo para el descenso de gradiente, Algoritmo (1), y su interpretación gráfica por medio de la Figura (2.8).

Algorithm 1 Descenso de gradiente

$t = 0$
while $t < T$ **do**
    $w_1^{(t+1)} = w_1^t - \alpha(\partial J(W)/\partial w_1^t)$
    ...
    $w_n^{(t+1)} = w_n^t - \alpha(\partial J(W)/\partial w_n^t)$
    $t = t + 1$
end while

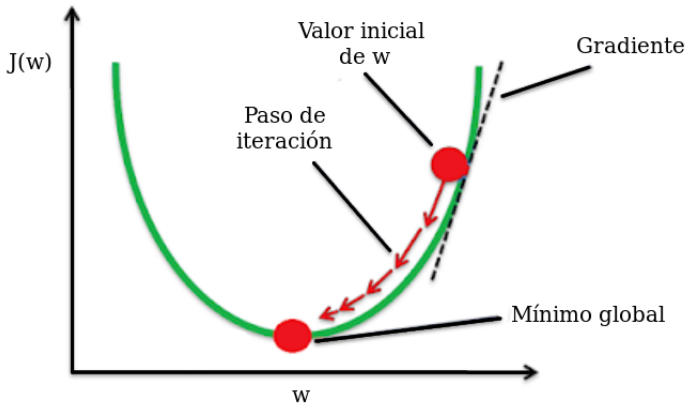


Figura 2.8: Intuición detrás del *"descenso de gradiente"*, mediante el trazado de una función de costos J, para un solo parámetro w.

Una vez culminado el entrenamiento, se utiliza la función de hipótesis h con los valores de los parámetros W aprendidos, para hacer predicciones sobre el conjunto de evaluación y determinar la eficiencia lograda.

Hasta la popularización de las redes neuronales artificiales (desarrolladas en las siguiente sección), los sistemas clasificadores basados en este tipo de estrategia, eran los que mejores resultados ofrecían a problemas de reconocimiento e identificación.

2.3. Redes Neuronales Artificiales

Una red neuronal artificial (ANN), es un modelo computacional que tiene como finalidad imitar el funcionamiento de una red de neuronas biológicas. Éste, surge de la necesidad de aprender problemas cuyas representaciones matemáticas son hipótesis no lineales complejas, las cuales, de ser solucionadas mediante mecanismos similares de aprendizajes lineales (como el descripto en la sección anterior), acabaría entregando muchas variables que aprender, un costo computacional muy elevado y un resultado mas ineficiente.

2.3.1. Representación del modelo

La neurona (también llamada perceptrón), es la mínima unidad de una red neuronal. Básicamente es una unidad lógica que toma *«features»* $x_1, ..., x_n$ como entrada, y computa como resultado el valor de la función de hipótesis h.

> *Se denominan* ***features****, a cualquier tipo de medida elegida para representar las características de los objetos que forman parte del conjunto de entrada de un modelo de aprendizaje determinado.*

Al igual que en cualquier otro modelo, se tiene un conjunto de entradas y un conjunto de valores objetivo, y se trata de obtener predicciones que se encuentren lo más cerca posible de tales valores objetivos.

Las entradas $x = (x_1, ..., x_n)$ de cada unidad, se combinan mediante una combinación lineal ponderada por los valores de W, y el resultado es modificado por una función no lineal, antes de ser emitido.

$$z(x) = \sum_{i=1}^{n}(w_i x_i) + w_0 x_0 \tag{2.4}$$

$$h(x, W) = g(z) \tag{2.5}$$

Las Ecuaciones (2.4) y (2.5) describen el funcionamiento de una neurona donde, $W = (w_0, ..., w_n)$ es el vector de *pesos,* cuyos valores deben ser aprendidos, x_0 es el *sesgo* o *bias* (generalmente igual a 1) *y* la función ***g*** un modificador llamado *«función de activación»* que hace no lineal a nuestra *hipótesis* h. En la Figura (2.9), se ilustran los principales componentes de una neurona, tanto a nivel biológico como matemático.

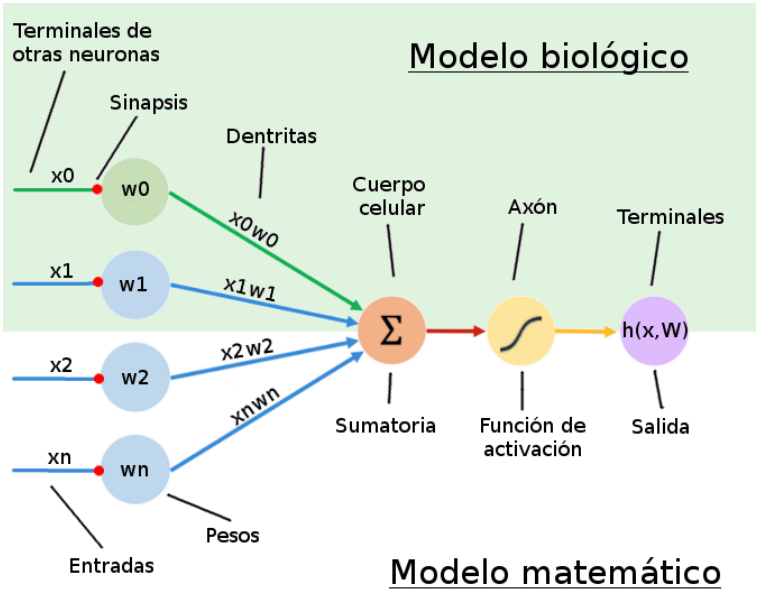


Figura 2.9: Representación de neurona: Principales componentes y comparación entre *"Modelo Biológico"* y *"Modelo Matemático"*.

Funciones de activación

Si observamos detenidamente la Ecuación (2.4), el valor de z podría ser cualquier cosa entre $[-\infty, +\infty]$, la neurona realmente no conoce los limites de los valores que maneja. Para decidir si debería dispararse o no (es decir, activarse o no), es que se agregan estas *funciones de activación* que permiten mapear los valores a un rango mas acotado, a partir de los cuales, mediante un valor umbral determinado, se tome la decisión de si la neurona debería ser considerada o no, por el resto de las unidades que integran la red.

Una de las funciones de activación mas comúnmente utilizada es la función **sigmoide**, Función (2.6) y Figura (2.10a). Ésta, toma valores reales y los mapea al rango [0,1] y es muy utilizada en procesos de clasificación ya que tiende a llevar las activaciones a ambos lados de la curva (por ejemplo, para valores de $-2 \leq x \geq 2$), haciendo distinciones claras sobre la predicción.

$$sigmoide(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

Otras funciones muy utilizadas en el proceso de activación son: **tanh** (tangente hiperbólica), y **ReLU** (unidad rectificadora lineal).

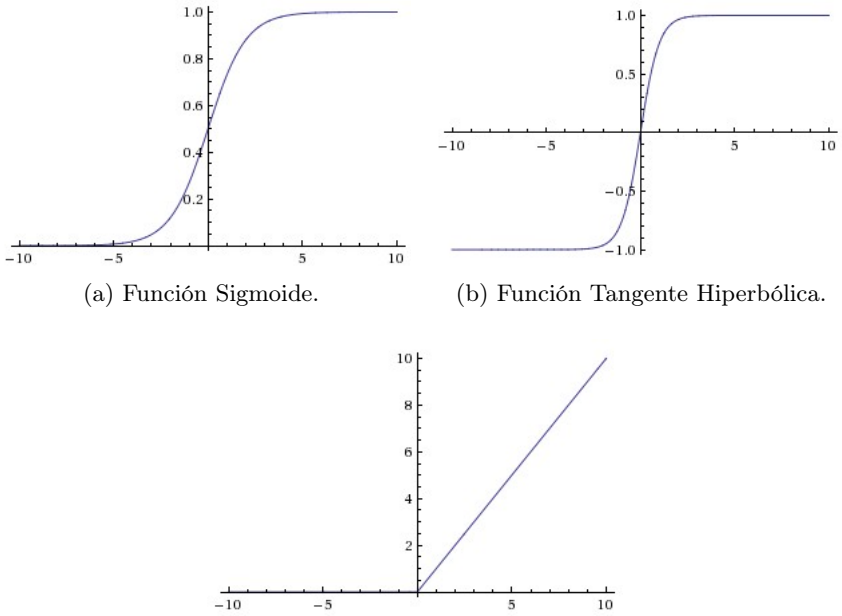
$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.7}$$

$$ReLU(x) = max(0, x) \tag{2.8}$$

El modificador **tanh,** Función(2.7) y Figura (2.10b), mapea valores reales al rango [-1,1], y a diferencia de la función **sigmoide**, se centra en 0. De echo, si prestamos atención a la Ecuación (2.9), puede apreciarse como **tanh** es una versión escalada de esta última.

$$tanh = 2\, sigmoide(2x) - 1 \tag{2.9}$$

Con respecto a la función **ReLU**, Función(2.8) y Figura (2.10c), ésta se encarga de mapear todo real negativo a 0 y deja invariantes los valores superiores o iguales a este.



(a) Función Sigmoide.

(b) Función Tangente Hiperbólica.

(c) Función ReLU.

Figura 2.10: Funciones de activaciones más utilizadas.

2.3.2. Arquitectura

Las neuronas de una red, pueden organizarse en capas y conectarse entre si con otras de la capa adyacente, nunca entre unidades de su mismo nivel. Este arreglo es conocido con el nombre de *«completamente conectadas»* (del *inglés*, *full connected*), por el hecho de que cada unidad se conecta con todas y cada una de las unidades alojadas en la capa adyacente.

Entre la *capa de entrada* (capa de features) y la *capa de salida* (resultado de función hipótesis), se pueden tener una o mas capas intermedias llamadas *capas ocultas*. Éstas, pueden alojar cualquier número de neuronas y serán las responsables de determinar la *arquitectura* de la red neuronal tal y como puede verse ilustrada en la Figura (2.11).

Sea L el número total de capas de la red, tal que $l = 1, .., L$; cada unidad alojada en la capa $l > 1$, procesará los valor de entrada recibidos por las unidades de la capa neuronal inferior $l - 1$ y pasarán su salida a la capa superior $l + 1$, a menos que la neurona pertenezca a la *capa de salida* L, cuyo resultado será el valor de la predicción esperada.

Es decir, que la función de hipótesis de una red neuronal, no es mas que una composición de funciones de activaciones no lineales $g_j \circ g_{j-1} \circ ... \circ g_0$, que dan como resultado la función $h(x, w)$, también no lineal.

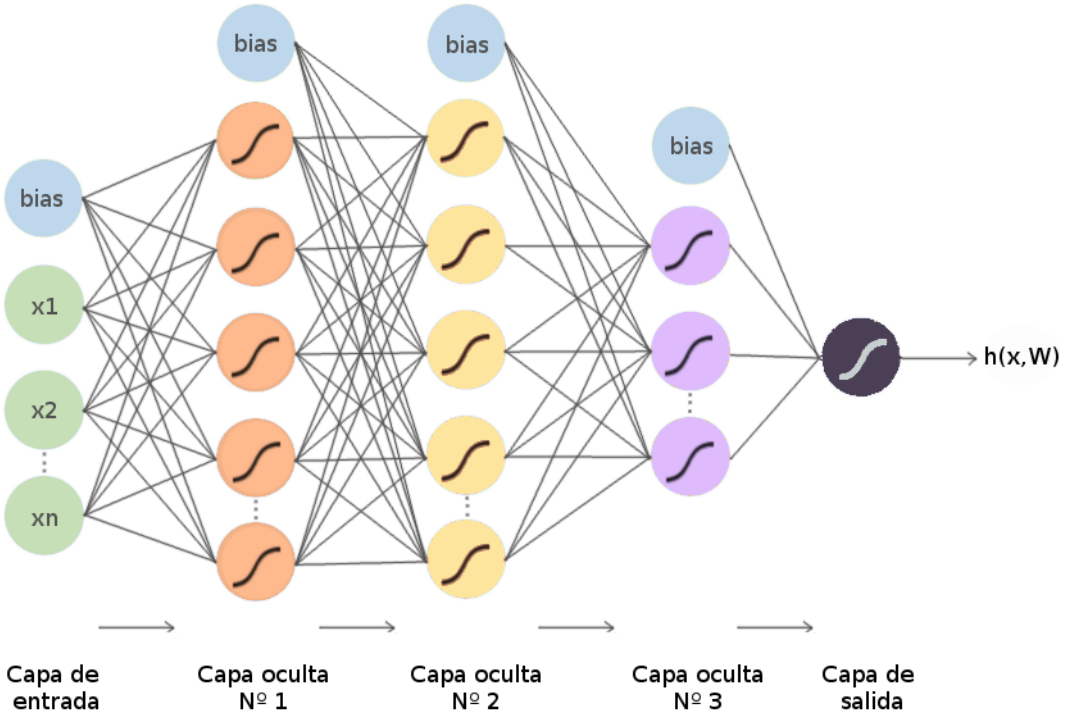


Figura 2.11: Red neuronal artificial con tres capas ocultas o intermedias.

2.3.3. Entrenamiento

Antes que nada, recordemos que estamos en un entorno de aprendizaje supervisado, Sección (1.3.1), es decir, tenemos nuestro conjunto de entrada de datos y sus respectivas etiquetas o clases que nos indican el valor esperado.

Aquí, entrenar la red neuronal, también es aprender los valores de nuestro vector o matriz de pesos W. El entrenamiento se puede ver como un proceso iterativo de "ida y vuelta" por las capas de la red de neuronas. La "ida", llamada *«forward propagation»* o propagación hacia adelante de la información, y la "vuelta" *«backpropagation» o* propagación hacia atrás de la información.

Forward propagation:

La propagación hacia adelante es la primera fase y ocurre cuando la red es expuesta al conjunto de datos de entrenamiento, los cuales atraviesan toda la red neuronal y se calculan sus "clases" o predicciones. Es decir, los datos de entrada pasan a través de la red, de tal manera, que todas las neuronas aplican su transformación a la información que reciben de las neuronas de la capa anterior, y la envían a las neuronas de la capa siguiente, hasta alcanzar la predicción de etiquetas para cada ejemplo de entrada, en la capa final.

Una vez realizada esta clasificación, se determina la validez de la misma, mediante una función *de error o costo* (de manera similar a lo acontecido en los clasificadores lineales), que calcula que tan buena o mala es, comparando cada predicción con las clases verdaderas a las que pertenecen cada uno de los ejemplos de entrenamiento introducidos a la red.

Backpropagation:

Una vez que se ha calculado el error a partir de la capa de salida, éste se propaga hacia atrás capa por capa, a todas las neuronas de la red. Cada una recibe una "porción" de error en relación a su aporte generado en la salida original.

Idealmente se busca que nuestra función de costo sea lo más cercana posible a cero, la próxima vez que se use la red para realizar una predicción. Por lo tanto, para lograr minimizar esta función se utiliza el método de *descenso de gradiente* (descripto en la Sección (2.2.2)) para ir ajustando gradualmente los valores de los pesos de cada interconexión neuronal, hasta alcanzar la convergencia a un mínimo u obtener buenas predicciones.

A continuación se da un resume del algoritmo de aprendizaje, mediante los siguientes pasos:

1. Iniciar los valores de nuestro parámetro W (generalmente aleatorios cercanos a 0).
2. Realizar *forward propagation* sobre un subconjunto de datos de entrada para obtener predicciones.
3. Calcular el error obtenido mediante una función de costo.
4. Realizar *backpropagation* para propagar el error a todos y cada uno de los parámetros que conforman la red.
5. Actualizar *pesos* mediante *descenso de gradiente.*
6. Repetir pasos del 1 al 5 hasta obtener un buen modelo.

2.4. Redes Neuronales Convolucionales

Una *«Red Neuronal Convolucional»* (de sus siglas en *inglés,* **CNN** o **ConvNet**), es un caso concreto de redes neuronales artificiales de aprendizaje profundo que ya se utilizaban a fines de los 90. En los últimos años, con el avance tecnológico y el incremento en el poder de cómputo, se han vuelto enormemente populares al lograr resultados muy buenos en el campo del reconocimiento de imágenes, impactando profundamente en el área de visión por computadoras.

Las ConvNets, son muy similares a las redes neuronales convencionales, y están formadas por neuronas que tienen parámetros en forma de pesos y sesgos, los cuales también deben ser aprendidos. A diferencial de estas últimas, las ConvNets, hacen la suposición explícita de que la entrada a su red son imágenes, permitiendo codificar ciertas propiedades en su arquitectura y reconocer elementos específicos de ellas, generando representaciones mas adecuadas.

Intuitivamente, una ConvNet, funciona similar a como nosotros los humanos reconocemos las cosas. Por ejemplo, si vemos una cara, la reconocemos porque tiene orejas, ojos, nariz, cabello, etc. Luego, para decidir si algo es una cara, lo hacemos verificando estas características que marcamos, y aunque a veces una cara puede "no tener orejas" porque éstas están cubiertas por el cabello, también la clasificaríamos con cierta probabilidad como cara, porque vemos los ojos, la nariz y la boca.

La principal ventaja de las CNNs, en comparación con sus predecesores, es que son modelos muy potentes y computacionalmente eficientes para realizar extracciones automáticas de características, permitiendo precisiones sobre humanas.

2.4.1. Operación de convolución

La operación de convolución '$\star$', es la operación fundamental de una ConvNet, por medio de la cual, se hace posible la extracción de las características de una imagen.

Por simplicidad para la explicación, y de aquí en adelante, definiremos como I, a una imagen en escala de grises representada por una matriz de dos dimensiones ($alto \times ancho$). Sea W, una matriz de dimensión ($f \times f$) (casi siempre 3×3 o 5×5), llamada *filtro* o *kernel,* diremos que "convolucionar" la imagen I con W ($I \star W$), consiste en superponer el filtro W sobre una región de la imagen e ir deslizándolo, calculando para cada posición, la suma sobre el producto (elemento a elemento) de la matriz W y la matriz de pixeles comprendidos en esa región de la imagen.

Matemáticamente la operación de convolución sigue la Ecuación (2.10).

$$(I \star W)_{(i,j)} = \sum_{k=0}^{f-1} \sum_{l=0}^{f-1} W_{(k,l)} I_{(i+k,j+l)} \tag{2.10}$$

donde $(I \star W)_{(i,j)}$ es el (i,j)-ésimo valor de la matriz $(I \star W)$, cuya dimensión D responde a la siguiente fórmula.

$$D = (\lfloor \frac{alto + 2p - f}{s} \rfloor + 1) \times (\lfloor \frac{ancho + 2p - f}{s} \rfloor + 1) \tag{2.11}$$

En la ecuación anterior (2.11), s es llamado *«stride»*, p *«zero-padding»* y f indica la dimensión del filtro W.

Por *stride*, se entiende a cómo se desliza el filtro sobre la imagen y puede verse como una forma de disminuir la resolución, ya que, a pasos más altos se obtienen dimensiones de salida más pequeñas. Por ejemplo, si el paso es 2, el filtro se moverá 2 pixeles a la vez.

Por otro lado, *zero-padding*, implica insertar p valores 0's alrededor de los bordes de la imagen. Por lo que si tenemos $p = 2$, se deberían agregar 2 capas de pixeles, de valor 0, alrededor de los bordes de la imagen de entrada. Su principal función es proporcionar mas cuadros de cobertura durante el proceso de convolución, a los pixeles ubicados en los bordes de la imagen, y de esta forma, obtener mas información de ellos. La Figura (2.12), ilustra con un ejemplo, tal operación de convolución.

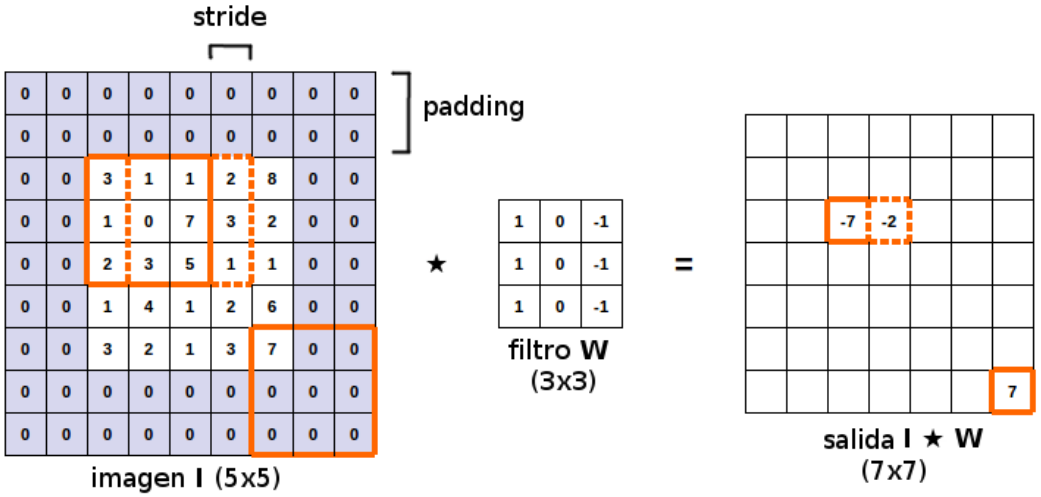


Figura 2.12: Operación de convolución.

Cabe aclarar que si la operación de convolución es realizada con mas de un filtro, la matriz resultante tendrá una profundidad igual a la cantidad de filtros aplicados. En caso de que la convolución se realice sobre una matriz de 3 dimensiones, por ejemplo ($alto \times ancho \times c_i$), como una imagen a color, los filtro W involucrados, estrictamente deben poseer la misma profundidad que el numero de canales de la imagen, es decir, ($f \times f \times c_f$) donde $c_f = c_i$, Figura (2.13).

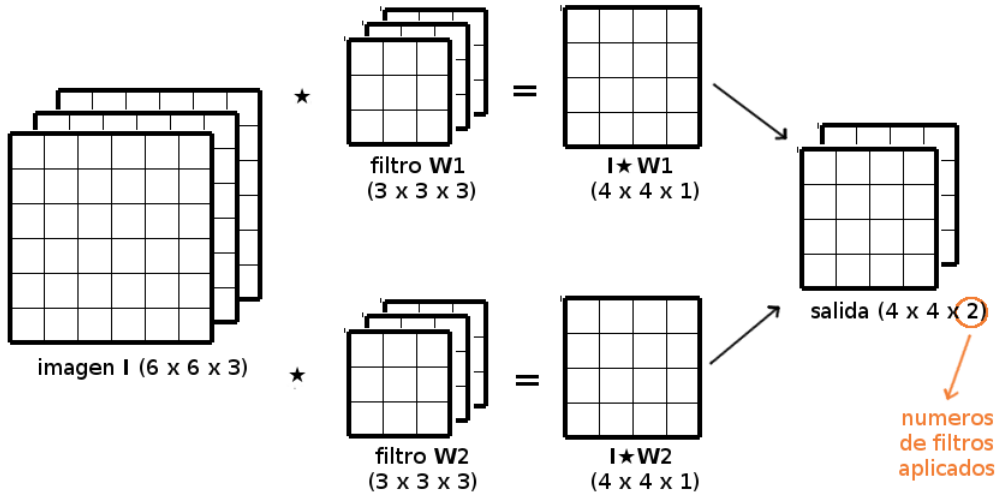


Figura 2.13: Ejemplo de convolución entre una imagen RGB y dos filtros diferentes.

2.4.2. Arquitectura

Dado el hecho de que las entradas a una ConvNet consisten solo de imágenes, las capas de su red, tienen neuronas dispuestas en 3 dimensiones: ancho, alto y profundidad; formando *«volúmenes de activación»*.

A diferencia de como se lo hacía en las redes neuronales convencionales, donde cada neurona de una capa oculta, se conectaba con cada una de las de la capa superior adyacente, en las CNNs, las neuronas de una capa oculta, se conectan solamente con una pequeña región de pixeles, o en su defecto, con una pequeña región del *volumen de activación* que generó la capa anterior, en caso de que la neurona no pertenezca a la primer capa oculta de la red. Esta región, a la cual cada neurona se conecta, se la conoce con el nombre de *«campo receptivo* o *campo receptivo* local», y su dimensión se corresponde con la dimensión del filtro aplicado en la capa, Figura (2.14).

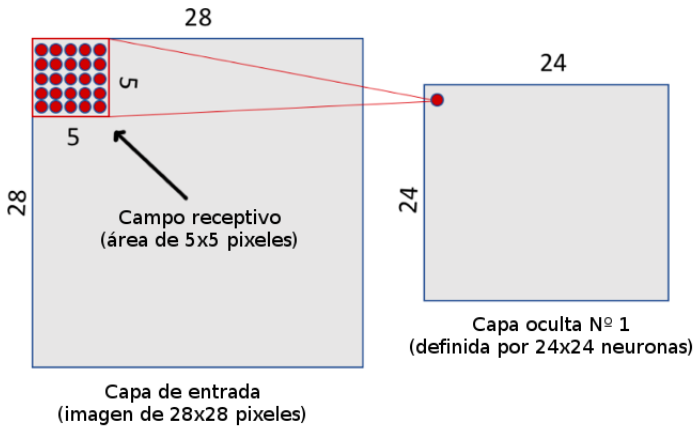


Figura 2.14: Campo receptivo.

A grandes rasgos, un ConvNet puede verse como una simple secuencia de capas, en donde cada una transforma un volumen de activación en otro, y nivel a nivel se continúan procesando y obteniendo representaciones cada vez mas complejas y abstractas, hasta llegar a un vector de valores finales que plasma las principales características de la imagen de entrada. Posteriormente, este vector es utilizado como entrada en algún otro modelo de aprendizaje, (habitualmente una red neuronal convencional), con la finalidad de clasificar, detectar patrones y demás objetivos en

los datos de entrada originales.

A este primer bloque, encargado de generar el vector de características de cada imagen de entrada, se lo conoce como *«extractor de características»* o (en *inglés*, features extraction) y es muy utilizado en los pipelines de diversos algoritmos de detección de objetos.

2.4.3. Bloques principales de construcción

Tres tipos de capas o bloques, son las utilizadas en la construcción de una red convolucional, las cuales se apilarán para formar una arquitectura determinada: «Capa convolucional» (CONV), «Capa de agrupación» (POOL) y «Capa totalmente conectada» (FC).

En particular, las capas CONV y FC realizan transformaciones en función de los volúmenes de activaciones de entrada y de los parámetros W y b de las neuronas (los pesos y los sesgos respectivamente), los cuales, deberán ser entrenados mediante el algoritmo de *descenso de gradiente*. Por otro lado, las capas POOL, siempre ejecutan una función fija.

Una arquitectura básica que ejemplifique una ConvNet, se ilustra en la Figura (2.15).

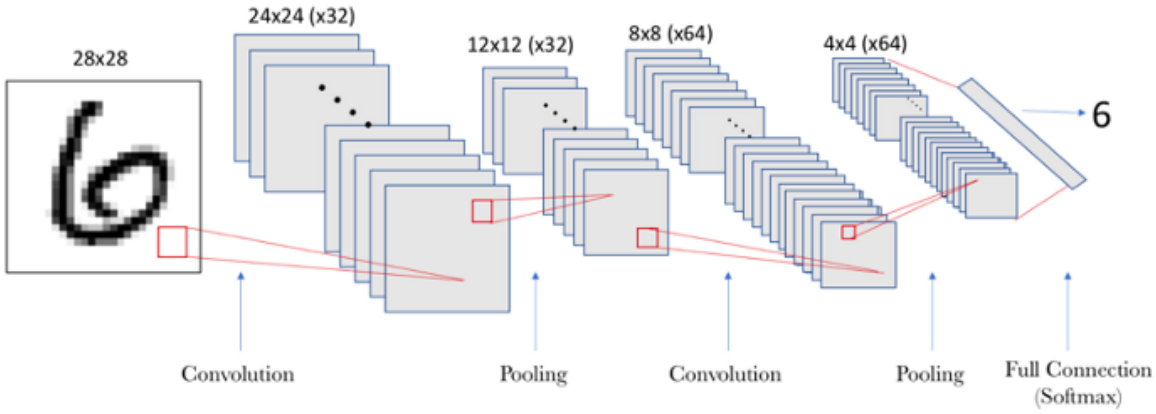


Figura 2.15: Ejemplo de arquitectura ConvNet (fuente: [12]).

A continuación, se presentará una explicación mas detallada de cada una de ellas.

Capa convolucional (CONV)

La capa de convolución, es el componente básico de una CNN y realiza la mayor parte del trabajo computacional pesado.

Cada una de las capas de tipo CONV, se encargan de recibir un volumen de activación y convolucionarlo con uno o mas filtros, con la finalidad de extraer características particulares y distintivas de la imagen (bordes verticales, lineas, manchas, etc). Como se describió antes, estos filtros son pequeñas matrices de valores que deberán ser aprendidos.

Al resultado de cada convolución, se le adiciona un *sesgo* (b) particular, y aplica una una operación de transformación no lineal (ReLU), generando por cada filtro aplicado, una matriz de dos dimensiones llamada *«mapa de activación»*. Luego, estos *mapas de activaci*ones, son apilados dando forma al nuevo *volumen de activación,* que pasará a la siguiente capa de la red para ser procesado, Figura (2.16).

Formalmente, el valor de la neurona (i, j)-ésima de una determinada capa convolucional, es:

$$relu(b + (\sum_{k=0}^{f-1}\sum_{l=0}^{f-1} W_{(k,l)}V_{(i+k,j+l)})) \tag{2.12}$$

donde W es la matriz de pesos de dimensión $(f \times f)$, V es el volumen de activación de entrada, b el sesgo adicionado y $relu$ la función de activación (por ejemplo, $max(0, x)$).

Dado a que cada neurona de una misma capa comparte iguales pesos y sesgos, y sumado al hecho de que su valor depende unicamente de una pequeña región de valores en la entrada; hacen

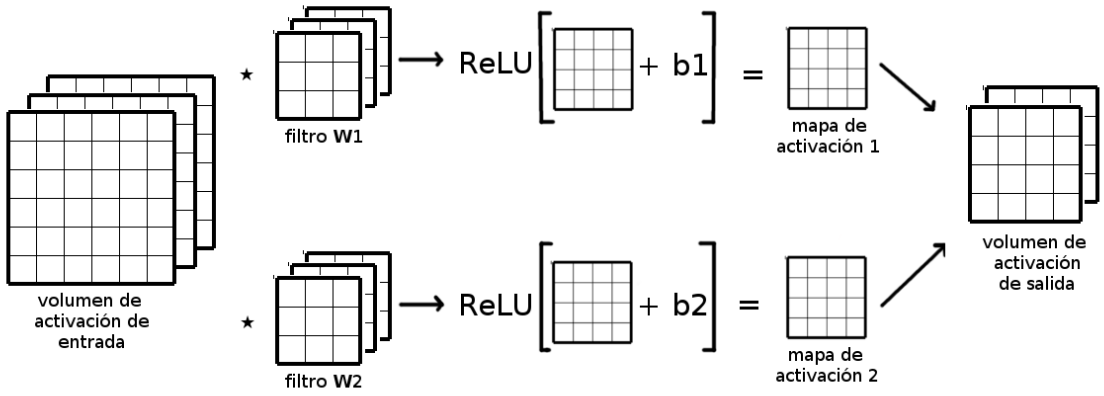


Figura 2.16: Ejemplo de capa tipo CONV, en una red neuronal convolucional.

que la cantidad de parámetros que la red deba aprender sea increíblemente menor, en comparación a los necesarios para realizar el mismo aprendizaje, si se utilizara una red neuronal convencional.

Capa de agrupación (POOL)

En toda arquitectura ConvNet es común insertar, entre capas CONV sucesivas, una capa de tipo POOL. El propósito principal de estas capas es reducir progresivamente el tamaño espacial de la representación, y por consiguiente, la cantidad de parámetros y cálculos realizados por la red.

La capa de agrupación, funciona independientemente en cada segmento de profundidad de la entrada, y la redimensiona espacialmente utilizando la operación MAX. La forma más común de aplicación de una capa de agrupación es: con filtros de dimensiones 2×2 y con un *stride* de 2, como se muestra en la Figura (2.17).

En ocasiones, las capas POOL, suelen emplear operadores como AVG o norma $L2$, aunque en la práctica, MAX otorgando mejores resultados.

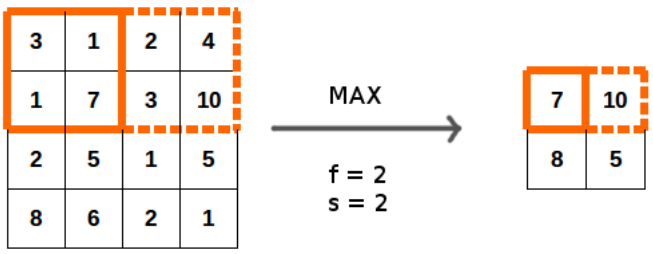


Figura 2.17: Ejemplo de capa tipo POOL, en una red neuronal convolucional.

Capa totalmente conectada (FC)

Al final de una red convolucional, la salida de la última capa POOL actúa como entrada a la capa *totalmente conectada*.

Por lo general, la capa FC es un red neuronal convencional que contiene alguna función de activación al final, que genera una probabilidad (número que oscila entre 0 y 1), permitiendo realizar una clasificación acorde a lo que el modelo intente predecir.

2.4.4. Algunas de las ConvNets mas conocidas

LeNet-5

LeNet-5 [13], fue la red convolucional pionera desarrollada por LeCun et al. in 1998. Esta red de 7-niveles, fue aplicada por varios bancos en la tarea de reconocer los números escritos a mano que figuraban en los cheques, mediante sus digitalizaciones a imágenes en escala de grises de 32×32 pixeles.

Debido a que la capacidad de procesar imágenes de mayor resolución requerían de más capas convolucionales, esta técnica estaba limitada por la disponibilidad de recursos informáticos de aquel entonces.

AlexNet

AlexNet [14], fue el primer trabajo que popularizó las redes convolucionales en visión artificial, desarrollado por Alex Krizhevsky, Ilya Sutskever y Geoff Hinton, la AlexNet se presentó al desafío ImageNet ILSVRC en 2012 y superó significativamente al segundo finalista.

Una particularidad de esta, fue que su arquitectura presentaba capas CONV apiladas una encima de la otra, a diferencia de lo habitual hasta ese entonces, de tener siempre una sola capa CONV inmediatamente seguida por una capa POOL.

GoogLeNet

La GoogLeNet [15], fue la red convolucional ganadora de ILSVRC 2014, desarrollada por Szegedy et al. de Google.

Su principal contribución fue el desarrollo de *«bloques Inception»*, lo cual redujo drásticamente el número de parámetros en la red (4M, en comparación con AlexNet con 60M).

Un *Módulo Inception*, es un tipo de capa ConvNet, la cual en lugar de ya tener asignados los tamaños de filtros para la convolución o para el proceso de agrupación, utiliza muchas combinaciones de estos y concatena todas las salidas, dejando de este modo, que la red aprenda cualquier parámetro que quiera usar, sea cual fuese las combinaciones utilizadas.

Otra característica interesante, es que esta red solo utiliza capas AVG-POOL, en lugar de capas FC en la parte superior de la ConvNet, eliminando una gran cantidad de parámetros.

VGGNet

VGGNet [16], obtuvo el segundo lugar en ILSVRC 2014. Desarrollada por Karen Simonyan y Andrew Zisserman, su principal contribución fue demostrar que la profundidad de la red es un componente crítico para un buen rendimiento.

Su mejor red final contiene 16 capas CONV / FC, y de manera atractiva, presenta una arquitectura extremadamente homogénea que solo realiza convoluciones 3×3 y agrupación 2×2 desde el principio hasta el final. Una desventaja de VGGNet es que es bastante costosa de evaluar y utiliza mucha más memoria y parámetros (140M). La mayoría de estos parámetros se encuentran en la primera capa FC, y desde entonces se descubrió que estas capas pueden eliminarse sin rebajar el rendimiento, lo que redujo significativamente la cantidad de parámetros necesarios.

MobileNet

MobileNet [17], es una de las mas novedosas y actuales redes. Desarrollada por Howard et al. en 2017, su característica principal es que tiene una arquitectura muy ligera, lo que la hace muy eficiente para aplicaciones de visión móvil.

Utiliza convoluciones separables en profundidad, lo que básicamente significa que realiza una sola convolución en cada canal de color, en lugar de combinar los tres y aplanarlos, proporcionando un efecto de filtrado de los canales de entrada.

Los autores de MobileNet explican este proceso de la siguiente manera:

> *"... la convolución profunda aplica un filtro único a cada canal de entrada. La convolución puntual aplica una convolución* 1×1 *para combinar las salidas de convolución en profundidad. Una convolución estándar filtra y combina entradas en un nuevo conjunto*

de salidas en un solo paso. La convolución separable en profundidad divide esto en dos capas, una capa separada para filtrar y una capa separada para combinar. Esta factorización tiene el efecto de reducir drásticamente el cálculo y el tamaño del modelo... ”

2.5. Localización de objetos

Para definir el concepto de localización de objetos, se necesita obligatoriamente, remitirse a la tarea de clasificación. En ésta, el algoritmo de clasificación toma un imagen como entrada y retorna una probabilidad sobre una clase (o etiqueta), indicando posiblemente a que corresponde tal imagen.

En la localización de objetos, el algoritmo no solo tiene que etiquetar la imagen, sino también, es el responsable de poner un cuadro delimitador alrededor de la posición del objeto en la imagen. De allí, a que a esta tarea se la suela llamar *clasificación con localización*, donde el término *localización* se refiere a averiguar en qué parte de la imagen está el objeto que se ha detectado, Figura (2.18).

Lo que habitualmente realiza un robusto clasificador de imágenes es, usando el poder de las ConvNet como extractoras de características, generar un vector de características para la imagen de entrada y alimentar un clasificador lineal, el cual se encarga de clasificar el objeto (por ejemplo, en 4 valores de probabilidades sobre las clases ‘peatón’, ‘autobús’, ‘automóvil’ y ‘ninguno’); donde la clase ‘ninguno’ se refiere a imágenes que no contienen objetos que pertenezcan a alguna de las restantes clases de interés.

Teniendo esto en cuenta, lo que realiza el algoritmo de localización, para poder localizar estos objetos en la imagen es, adaptar la red para que retorne, además de las 4 probabilidades de clase, algunos valores mas de salida que ayuden a delimitar el recuadro que rodeará la posición del objeto. En particular, cuatro valores (b_x, b_y, b_h, b_w) son agregados, donde (b_x,b_y) identifican las coordenadas centrales y (b_h,b_w) la altura y el ancho relativos del recuadro delimitador del objeto.

(a) AUTOBUS (b) AUTOBUS

Figura 2.18: (a) Clasificación vs. (b) Clasificación + Localización.

2.5.1. Entrenamiento

En este caso, para entrenar un algoritmo de localización, el conjunto de entrenamiento no solo deberá contener las etiquetas respectivas a la imagen a predecir, sino que también, deberá incluir cuatro parámetros adicionales que marquen cual es la posición de ese objeto en la imagen.

Por lo tanto, si $x^{(i)} \epsilon R^D$ es el vector que representa a la i-ésima imagen del conjunto de entrenamiento (de dimensión D), b_x, b_y, b_h y b_w son los nuevos parámetros descriptos anteriormente, y $c_1, c_2, ..., c_k$ los valores de probabilidad para cada una de las k clases; el vector objetivo $y^{(i)}$ de la imagen $x^{(i)}$, será:

$$y^{(i)} = [p_c, b_x, b_y, b_h, b_w, c_1, c_2, ..., c_{k-1}]$$

donde p_c indicará la probabilidad de ser un objeto o no (es decir, $p_c = 0$ sii $y^{(i)}$ etiqueta la clase ‘ninguno’, $p_c = 1$ en caso contrario).

Tanto los parámetros b_x, b_y, b_h, b_w como los valores de $c_1, ..., c_{k-1}$ serán importantes solo si $p_c = 1$, de otra forma no tendrían ningún sentido porque no habría ningún objeto para localizar ni clasificar. En ese caso, el vector objetivo resultaría en:

$$y^{(i)} = [p_c = 0, b_x = ?, b_y = ?, b_h = ?, b_h = ?, c_1 = ?, ..., c_{k-1} = ?]$$

donde el símbolo '?' representa cualquier valor, ya que estos parámetros no son de interés y no serán tenidos en cuenta a la hora del entrenamiento.

En su defecto, supongamos que $x^{(i)}$corresponde a la clase 2 (por ejemplo, autobús); un posible vector objetivo se vería de la siguiente manera:

$$y^{(i)} = [p_c = 1, b_x = 0.5, b_y = 0.5, b_h = 0.4, b_w = 0.3, c_1 = 0, c_2 = 1, ..., c_{k-1} = 0]$$

indicando que se esta 100 % seguros de que es un objeto ($p_c = 1$), que se encuentra un autobús con 100 % de certeza ($c_2 = 1$), esta ubicado en el centro de la imagen ($b_x = 0.5, b_y = 0.5$) y su alto y ancho son el 40 % y 30 % respecto de las dimensiones originales de tal imagen.

Función de costo

Una de las opciones mas simples al momento de definir la *función de cost*o para este modelo, es la función de ***error cuadrático medio***.

Supongamos (similar a como se lo hizo en la clasificación lineal, Sección (2.2)), que $\widehat{y}$ es la predicción de salida de la red (función hipótesis), y m el número de ejemplos del conjunto de entrenamiento.

$$J_{\widehat{y},y} = \frac{1}{2m} \sum_{i=1}^{m} \begin{cases} \sum_{j=1}^{k-1} (\widehat{y_j}(x^{(i)}) - y_j^{(i)})^2 & si \;\; y_1^{(i)} = 1 \\ (\widehat{y_1}(x^{(i)}) - y_1^{(i)})^2 & si \;\; y_1^{(i)} = 0 \end{cases} \tag{2.13}$$

La ecuación (2.13), describe la función de *costo* utilizada para entrenar un modelo de localización de objetos. Cuando $y_1^{(i)} = 0$, es decir, la primer coordenada del vector objetivo del i-ésimo ejemplo de entrenamiento es igual a cero, solo éste parámetro ($p_c^{(i)}$) es tenido en cuenta para el computo del error; explicando de esta manera porque no se preocupó por los valores representados por el símbolo '?' al definir el vector para una imagen perteneciente a la clase 'ninguno'.

Finalmente, el modelo será ajustado por la minimización de $J_{\widehat{y},y}$ mediante la técnica de *descenso de gradiente* (2.2.2).

2.6. Detección de objetos

Con los avances recientes en los modelos de visión por computadoras basados en aprendizaje profundo, las aplicaciones de detección de objetos son más fáciles de desarrollar que nunca. Enfoques actuales, que centran sus arquitecturas en *pipeline* completos de extremo a extremo, han logrado mejorar aún mas el rendimiento, permitiendo que algunos de estos, incluso, puedan utilizarse en tiempo real.

Para comenzar, la principal diferencia entre un modelo de *localización de objetos* y uno de *detección de objetos*, es sutil. La localización de objetos, simplemente tiene como meta, ubicar el objeto principal en una imagen (objeto más visible o grande); mientras que la detección de objetos, intenta encontrar todas las instancias de aquellos objetos y sus límites, Figura (2.19). Sin embargo, existe cierta superposición entre este modelo y un clasificador de imágenes. Si bien, un detector necesita de un clasificador, si se desea clasificar una imagen en una determinada categoría, podría suceder que el objeto, o las características que se requieran para realizar tal "categorización", sean demasiado pequeñas con respecto a la imagen completa. En ese caso, la utilización del método de detección, en lugar de un clasificador de imágenes, podría lograr un mejor rendimiento, incluso, aunque no se estuviese interesado en recuperar las ubicaciones exactas del mismo.

2.6.1. Construcción del modelo

La razón principal por la que no puede modelarse el problema de detección de objetos con una ConvNet estándar seguida de una capa FC es, que debido al número de ocurrencias no fijas de los objetos de interés, hace que la longitud de la capa de salida de la red, sea variable.

Figura 2.19: Múltiples detecciones de objeto 'AUTOBUS', mediante la ejecución de modelo YOLO v2.

Un aproximación bastante primitiva para su construcción, podría ser, tomando diferentes regiones de una imagen, y por medio de una ConvNet, clasificar la presencia o no, de objetos dentro de la misma.

Siguiendo esta idea, y para hacer la explicación mas amena, supongamos que se esta interesado en detectar vehículos para analizar, mediante el conteo de los mismo, la carga vial de una determinada avenida principal.

Lo primero que se debe realizar, como en todo algoritmo de aprendizaje supervisado, Sección (1.3.1), es generar un conjunto de entrenamiento. En este caso, se etiquetan "recortes" de automóviles (es decir, se toman las imágenes con los límites lo mas cercanos posibles al objeto de interés, sin nada de escena circundante), e imágenes sin ellos (fondos que representarán a la categoría 'ninguno').

Luego, utilizando el poder de las ConvNet, se entrena un clasificador para que su predicción de salida sea '1', cuando una imagen clasifique positivamente como 'automóvil', y '0' en caso contrario. Una vez entrenada la red, es necesario utilizar algún método capaz de seleccionar *«regiones de interés»*, las cuales serán insertadas en el clasificador, y en caso de suceder una predicción positiva, obtener la localización del objeto en la imagen, en base a las coordenadas del recorte procesado.

2.6.2. Selección de regiones de interés

Formalmente, una *región de interés* o ROI (del *inglés*, Region of Interest), es una región de la imagen de entrada, que potencialmente podría contener un objeto a ser detectado.

Muchos son los enfoques que son utilizados para seleccionar tales regiones, desde algoritmos de búsqueda selectiva, hasta métodos de búsqueda por fuerza bruta. Detección por Ventana Deslizante (del *inglés*, *Sliding Window Detection*) y Propuesta de Regiones (del *inglés*, *Region Proposals*), son algunos de los algoritmos referentes de tales enfoques.

Sliding Window Detection

A grandes rasgos, lo que hace el algoritmo de Ventana Deslizable es, en base a una ConvNet entrenada como se menciono anteriormente, tomar pequeñas regiones rectangulares de pixeles e ir deslizando esta suerte de "ventana", de arriba hacia abajo y de izquierda a derecha, a lo largo de toda la imagen, con la esperanza de que en alguno de los pasos, el recorte insertado en la ConvNet, clasifique como 'automóvil'; para luego de cada ciclo, repetir el mismo proceso mediante diferentes dimensiones de ventana y número de pixeles para el desplazamiento, Figura (2.20a).

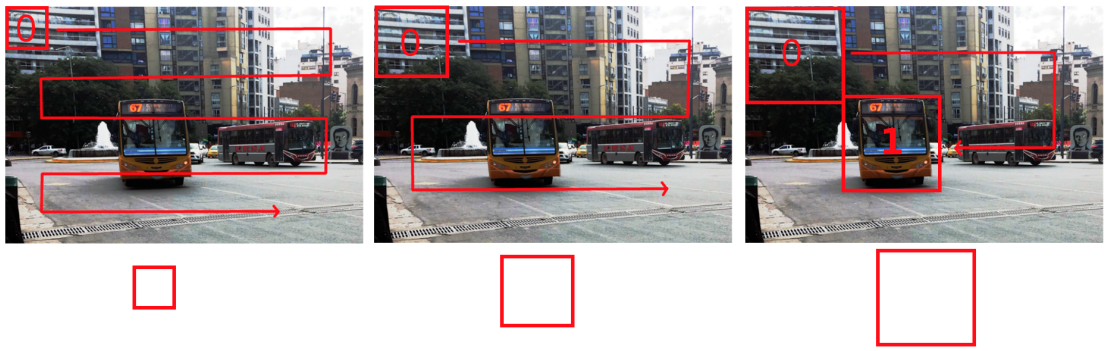
A pesar de no entregar malos resultados, éste, tiene la gran desventaja de ser un método demasiado lento; y si bien, mediante la disminución de la dimensión de la "ventana" y el número de pixeles para los desplazamientos, se puede mejorar la precisión y granularidad, el costo computacional asociado es tan alto que hace imposible su aplicación en tiempo real.

Mas tarde, como solución a estos problemas de velocidad y eficiencia, nuevos esfuerzos adaptaron el algoritmo para que pueda ser computado "convolucionalmente". Es decir, se modificaron y convirtieron las últimas capas FC de la ConvNet de clasificación, a capas CONV. Aunque esta modificación dio resultado, originó la pérdida de precisión en los recuadros delimitadores de los objetos detectados. Con el paso del tiempo, modelos mas complejos como YOLO, SSD y otros, surgieron para contrarrestar esta debilidad y serán explicados en las siguientes secciones.

Region Proposals

Con la actualización del algoritmo de Ventana Deslizante, ahora, se podía ejecutar de manera convolucional haciendo que los tiempos de detecciones se aceleren; a pesar de ello, se siguió manteniendo una gran desventaja que no se mencionó anteriormente. Para analizar y determinar la existencia o no, de algún objeto de interés en la imagen, éste, debía realizar fuego cruzado entre muchas de las regiones donde claramente no existía ningún objeto. Es decir, muchos de los rectángulos procesados no tenían nada interesante que clasificar. Así es, que de la mano de Ross Girshick et al., surge el algoritmo de Propuesta de Regiones, mas formalmente conocido como R-CNN [20] por sus siglas en inglés Regions-based with Convolutional Neural Networks, el cual será explicado en detalle en la Sección (4.1.2.1).

A diferencia de la detección por Ventana Deslizante, la cual rozaba la fuerza bruta al probar todas las combinaciones hasta dar con el objetivo, R-CNN selecciona unas pocas regiones de interés, basándose en la segmentación de colores en la imagen de entrada, reduciendo significativamente la cantidad de posibles candidatos a analizar, Figura (2.20b).



(a) Detección mediante Ventana Deslizante.

(b) Búsqueda selectiva basada en Propuesta de Regiones (segmentación de imagen simulada).

Figura 2.20: Enfoques para selección de regiones de interés.

2.6.3. Evaluación

Como en todo modelo de aprendizaje automático, en la detección de objetos también es necesaria una medida de evaluación que permita analizar que tan bien funciona el algoritmo. El índice de *«Intersección sobre Unión»* o IoU (*del inglés*, Intersection over Union), es la función utilizada para ello.

Sean $Box_{pred} = [bx_{pred}, by_{pred}, bh_{pred}, bw_{pred}]$ y $Box_{obj} = [bx_{obj}, by_{obj}, bh_{obj}, bw_{obj}]$ los parámetros de los recuadros delimitadores, predichos y objetivos, de un determinado objeto detectado;

donde (bx, by) indica la posición del vértice superior izquierdo del recuadro y (bh, bw) el alto y ancho respectivos (similar a como se estableció en la localización de objetos). Lo que hace la función IoU es medir que tanta similitud existe entre dos recuadros delimitadores (o rectángulos).

$$IoU(Box_{pred}, Box_{obj}) = \frac{A_{pred} \cap A_{obj}}{A_{pred} \cup A_{obj}} = \frac{A_{pred} \cap A_{obj}}{A_{pred} + A_{obj} - (A_{pred} \cap A_{obj})} \tag{2.14}$$

En la Ecuación (2.14), $A_{pred} \cap A_{obj}$ es la intersección entre: el área A_{pred} del rectángulo definido por Box_{pred}, y el área A_{obj} del rectángulo definido por Box_{obj}, cuyo valor es el resultado de la Ecuación (2.15).

$$A_{pred} \cap A_{obj} = max(0,\, p_{xmax} - p_{xmin}) \cdot max(0,\, p_{ymax} - p_{ymin}) \tag{2.15}$$

donde $(p_{xmin},\, p_{ymin})$ y $(p_{xmax},\, p_{ymax})$ representan las coordenadas de los vértices superior izquierdo e inferior derecho, de la intersección de los rectángulos delimitadores, donde:

$$p_{xmin} = max(bx_{pred},\, bx_{obj})$$

$$p_{ymin} = max(by_{pred},\, by_{obj})$$

$$p_{xmax} = min(bx_{pred} + bw_{pred},\, bx_{obj} + bw_{obj})$$

$$p_{ymax} = min(by_{pred} + bh_{pred},\, by_{obj} + bh_{obj})$$

Por convención, se puede juzgar el resultado de una predicción como buena si $IoU \geq 0{,}5$. En algunos casos más estrictos, este valor se puede elevar a 0,6 o incluso 0,7.

En caso de que ambas cajas delimitadoras, predichas y objetivo, se superpongan perfectamente, IoU sería igual a uno, dado que su intersección y su unión tendrían valores de áreas exactamente iguales. La Figura (2.21), muestra la interpretación gráfica de la definición de IoU.

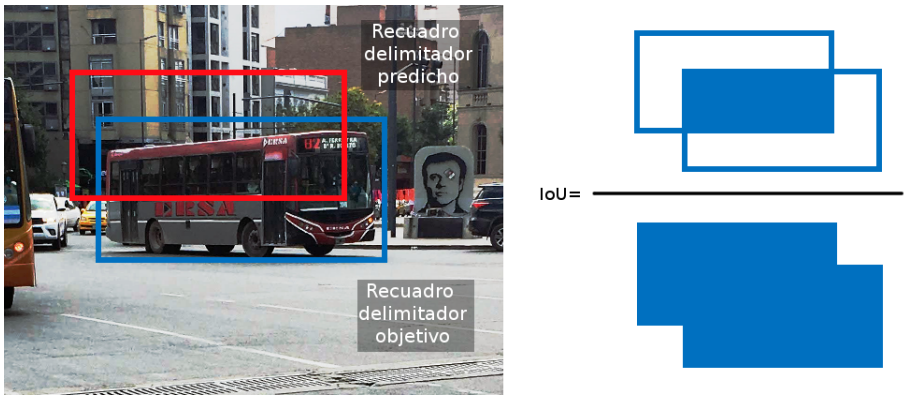


Figura 2.21: Visualización de la definición de IoU.

2.6.4. Non-Maximum Suppression

La superposición de recuadros delimitadores es un problema muy común en los algoritmos de detección de objetos. Estos, pueden arrojar múltiples detecciones para los mismos objetos y en lugar de detectar un objeto una sola vez, podrían incluso, detectarlo cientos de veces, generando muchas cajas delimitadoras superpuestas por cada uno.

Non-Maximum Suppression (NMS), es el método para asegurarse que cada objeto sea detectado solo una vez, agrupando las detecciones por cercanía espacial, mediante el índice IoU, manteniendo solo las propuestas más confiables de cada grupo.

Para hacer mas simple la explicación, se supondrá un detector entrenado para solo una clase. Siguiendo el ejemplo usado a lo largo de toda la sección, supondremos que solo detectará 'automóviles'.

Por lo tanto, sean $B = \{box_1, box_2, ..., box_N\}$ y $P = \{pc_1, pc_2, ..., pc_N\}$ los conjuntos de recuadros delimitadores e indices de certeza p_c, para N detecciones, en donde un valor pc_1 indica la probabilidad de que el objeto 'automóvil' sea localizado en la región de la imagen de entrada delimitada por los parámetros de box_1.

Lo que hace el método de *NMS* es una suerte de "limpieza de detecciones", lo cual podría resumirse en los siguientes pasos:

1. Descarte de detecciones con un valor de $pc < threshold_{conf}$.
   Generalmente $threshold_{conf} = 0{,}6$
2. Búsqueda de detección con mayor valor de pc.
3. Eliminación de detección remanentes con $IoU \geq threshold_{IoU}$ en relación a detección de paso (2). Generalmente $threshold_{IoU} = 0,5$.
4. Repetición de pasos 1 a 3, hasta agotar detecciones.

A continuación se ilustra un ejemplo de aplicación en la Figura (2.22) y se describe el pseudocódigo del algoritmo completo de NMS (2).

(a) Detecciones redundantes para un mismo objeto.

(b) Detección con mayor porcentaje de certeza.

Figura 2.22: Ejemplo de Non-Max Suppression en el contexto de detección de autobuses.

En caso de que un detector sea entrenado para más de una clase, supongamos $k = 3$ (c1, c2 y c3), debe realizarse un paso previo a la aplicación de NMS.

Sea:

$$\{[pc_1, box_1, c1_{1,}, c2_1, c3_1], [pc_2, box_2, c1_{2,}, c2_2, c3_2], ..., [pc_N, box_N, c1_{N,}, c2_N, c3_N]\}$$

el conjunto de N predicciones de salida de la red. Entonces para cada clase k, es generado un nuevo conjunto P, donde cada uno de los valores resultarán entre el producto del respectivo N-ésimo valor de pc y la probabilidad ck_N. Es decir:

$$P_k = \{(pc_1 \cdot ck_1), (pc_2 \cdot ck_2), ..., (pc_N \cdot ck_N)\}, \ k = 1, .., 3.$$

Luego se repite el algoritmo para cada par B,P_k.

Algorithm 2 Non-Maximum Suppression.

Input: $B = \{box_1, box_2, ..., box_N\}$, $P = \{pc_1, pc_2, ..., pc_N\}$

procedure NMS(B, P)
  $D \leftarrow \{\}$
  for pc_i in P **do**
    if $pc_i < threshold_{conf}$ **then**
      $B \leftarrow B - \{box_i\}$
    end if
  end for
  while $B \neq \emptyset$ **do**
    $max \leftarrow$ argmax(P)
    $D \leftarrow D \cup \{box_{max}\}$
    $B \leftarrow B - \{box_{max}\}$
    for box_i in B **do**
      if $IoU(box_{max}, box_i) \geq threshold_{IoU}$ **then**
        $B \leftarrow B - \{box_i\}$
        $P \leftarrow P - \{pc_i\}$
      end if
    end for
  end while
  return D, P
end procedure

3. Aproximación al reconocimiento de texto en escena

El texto, como una herramienta para la comunicación y la colaboración, ha desempeñado un papel muy importante a lo largo de la evolución de la humanidad. Dada su rica y precisa semántica incorporada, puede utilizarse para representar y describir fácilmente el mundo que nos rodea.

Actualmente, en la sociedad moderna, la información que ellos "contienen" y pueden "transmitir", juega un papel muy importante ya que es ampliamente utilizada en aplicaciones para traducciones instantáneas, navegación de robots y otras automatizaciones.

En este contexto y con los avances crecientes en materia de procesamiento y aprendizaje automático, la lectura automática de textos en entornos naturales, también conocida como *«reconocimiento de texto en escena»*, se ha convertido en uno de los temas de investigación más populares e importantes en disciplinas como la visión artificial y relacionadas.

A diferencia del reconocimiento que es realizado sobre un documento digitalizado, conocido generalmente por OCR (*del inglés*, Optical Character Recognition), reconocer texto en una imagen de un entorno natural, es mucho más desafiante debido a las muchas variaciones posibles en los fondos, las texturas, las fuentes y las condiciones de iluminación que pueden estar presentes en dichos entornos.

Aunque en la actualidad, existen sistemas de extremo a extremo capaces de realizar esta tarea en un solo paso, básicamente, un problema de reconocimiento de texto puede ser dividido en dos problemas diferentes cuyas soluciones contribuyen al resultado final del sistema completo, Figura (3.1). Un primer desafío, o etapa de *detección de texto*, cuyo objetivo es localizar aquellas regiones que contengan palabras individuales o líneas de texto; y un segundo desafío, de *reconocimiento de texto* en sí mismo, el cual se encarga de decodificar, los caracteres y/o palabras de las regiones detectadas por la etapa anterior, a texto plano.

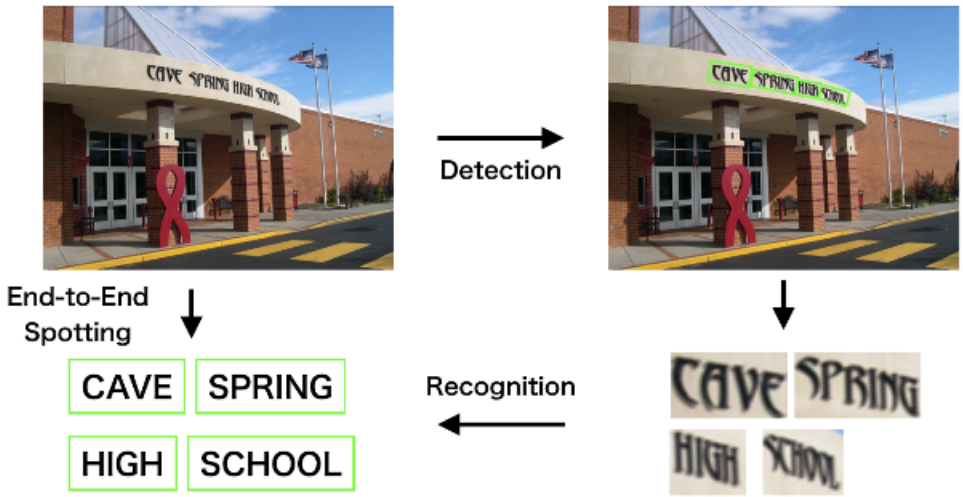


Figura 3.1: Diagrama de esquema de detección y reconocimiento de texto en escenas naturales (fuente: [21]).

3.1. Detección de texto

Como anteriormente se mencionó, el principal objetivo de la detección o localización de texto, es identificar potenciales regiones de una imagen de entrada, que puedan contener caracteres alfanuméricos en su interior. Típicamente, la tarea de detección, y al igual que como se hace en un modelo de detección de objetos, corresponde a generar recuadros delimitadores (o rectángulos) para cada palabra, o secuencia de estas, que aparezcan en la imagen.

Antes de la llegada del aprendizaje profundo, la mayoría de los métodos de detecciones estaban basados en CCA (del *inglés*, Connected Component Analysis[3]), el cual permitía obtener candidatos

[3] `https://en.wikipedia.org/wiki/Connected-component_labeling`

que luego eran filtrados mediante clasificadores automáticos entrenados sobre vectores de características generadas a mano, o métodos de clasificación basada en la técnica de *ventana deslizante,* donde la imagen era binarizada y posteriormente se agrupaban las regiones positivas resultantes mediante métodos como CRF (del *inglés*, Conditional Random Field[4]).

En la actualidad, se ha focalizado el esfuerzo en disminuir la longitud de los antiguos procesos de múltiples pasos, con el objetivo principal de reducir la propagación de errores y simplificar los procesos de entrenamiento.

Los métodos mas recientes siguen un esquema de *«pipeline simplificado»,* el cual consta solo de dos etapas: una primera, que consiste en el entrenamiento de una ConvNet, y una posterior, que procesa los resultados. Éstos, tomaron como inspiración las bases de las técnicas de detección general de objetos, en las cuales, mediante la modificación de las etapas de búsqueda de regiones de interés y los métodos de regresiones (utilizados para la predicción de los recuadros delimitadores), lograron que directamente se localicen instancias de texto.

3.2. Reconocimiento de texto

Los métodos de reconocimiento de texto son la última fase del sistema. En estos, la entrada es un recorte de una imagen que contiene una instancia de texto determinada.

Durante décadas, y al igual que en la tarea de detección, muchos fueron los esfuerzos, mediante la utilización de diversos enfoques y métodos, para dar solución a este problema. Desde algoritmos basados en la segmentación de caracteres y la implementación de técnicas como label embedding (la cual permite directamente la correspondencia entre palabras e imágenes), y métodos que proponían dividir el problema de reconocimiento en varias subetapas: tales como preprocesamiento de imagen, segmentación de caracteres/palabras, reconocimiento de caracteres/palabras y corrección; hasta aquellos que pretenden otorgar una solución mas integral (como los sistemas de extremo a extremo de la actualidad), han sido utilizados.

De todos ellos, el proceso de segmentación de caracteres/palabras, es considerada una de las tareas más difíciles y desafiantes de un sistema de reconocimiento en escenas naturales. Ésto, debido a la complejidad de los fondos y de las irregularidades que se presentan en relación a la disposición del texto, y a las cuales el modelo debe enfrentarse.

Por tal motivo y con la llegada de la era del aprendizaje profundo, fue que se comenzaron a desarrollar nuevas formas de suplantar tal proceso, el cual dada su complejidad, era el causante de disminuir en gran medida el rendimiento general del sistema. Algunos de los principales métodos adoptados, estuvieron basados en técnicas de *«Connectionist Temporal Classification»* (CTC) y «Attention Mechanisms».

[4] `https://en.wikipedia.org/wiki/Conditional_random_field`

Parte III
Metodología

En este parte del trabajo, se presentará la arquitectura del algoritmo propuesto para hacer frente a la tarea de *"reconocimiento automático de líneas urbanas de autobuses"*. De mayor a menor nivel, se describirán los detalles de implementación de cada uno de los módulos que integran el sistema completo, desde los criterios de selección utilizados en la búsqueda de cada modelo de aprendizaje, hasta las diferentes etapas de procesamiento y ajuste de sus parámetros, con la finalidad de obtener la mayor precisión y velocidad de ejecución posible.

Persiguiendo premisas de accesibilidad, familiaridad y debido a las altas prestaciones, con que en promedio, cuentan los teléfonos móviles modernos y sus cámaras integradas, el desarrollo del presente trabajo, inicialmente fue pensado y diseñado para una futura adaptación a una aplicación móvil. Sin embargo, y dada la multitud de plataformas portátiles de desarrollo de alto desempeño, que actualmente existen en el mercado, como *NVIDIA Jetson Nano*[5], *Coral Development Board*[6] y hasta algunos modelos mas recientes de la conocida Raspberry Pi[7], se deja la puerta abierta a cualquiera de estos otros dispositivos para ser soporte de una posible implementación.

Tomando como escenario de partida una parada de autobuses, el usuario iniciará indicándole al sistema el número de línea del autobús esperado. En paralelo a la aproximación y/o detención del autobús a la parada, se irán procesando las imágenes, hasta el momento capturadas, generando una lista de 5 potenciales líneas candidatas. En caso afirmativo, en el que el número de línea buscada forme parte de este "ranking", se dará aviso de la llegada del autobús deseado. Si bien esta fuera de nuestro alcance, tal aviso, podría ser implementado fácilmente mediante una devolución vibratoria, liberando al oído y evitando la sobrecarga auditiva que dificulte la comprensión del entorno cercano.

Básicamente, el sistema estará constituido por un algoritmo modular, donde cada uno de los bloques principales contendrán etapas de pre y post procesamiento (o módulos intermedios), las cuales jugarán un papel fundamental en el ajuste del buen desempeño del sistema completo de extremo a extremo. A grandes rasgos, costará de tres módulos principales: un *«módulo detector de objetos»*, uno de *«detección de textos»* y un último módulo para el *«reconocimiento de caracteres»*.

Como primera instancia, el módulo detector de objetos, será el encargado de recibir las imágenes de entrada (frames), y buscar regiones que potencialmente pudiesen contener autobuses en su interior. Posteriormente, cada una de estas regiones serán recibidas por una etapa intermedia de procesamiento, la cual, mediante una serie de transformaciones de espacios de colores y aplicaciones de filtros, generará nuevas representaciones, con el fin de exponer regiones de texto mediante cambios de contrastes, umbrales, saturaciones y ecualizaciones. Tales representaciones serán las entradas de la etapa de detección de texto.

El módulo de detección, como su nombre lo indica, tendrá la tarea de detectar y localizar regiones de caracteres alfanuméricos en las imágenes, con la esperanza de que en alguna de ellas se encuentre contenido el número de línea buscado. Finalmente, cada una de estas detecciones, en conjunto con las representaciones obtenidas como resultado de su paso por un pequeño algoritmo de segmentación basado en umbralización, serán decodificadas a texto plano mediante el módulo de reconocimiento, para luego, posterior al filtrando de resultados no numéricos, contabilizar y generar la lista de posibles números de líneas candidatas.

En las siguientes secciones, se describirán las técnicas, modelos y algoritmos empleados en el desarrollo de cada uno de los módulos y submódulos, en conjunto con la sucesión de decisiones que culminaron con la construcción final del *pipeline* del sistema.

[5] `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`
[6] `https://coral.ai/products/dev-board/`
[7] `https://www.raspberrypi.org/products/raspberry-pi-4-model-b/`

4. Modelos y herramientas utilizadas

4.1. Reconocimiento de autobuses

4.1.1. Contexto

Dada la naturaleza del problema tratado y en base a los resultados recolectados en una etapa previa de investigación, la necesidad de una respuesta rápida en las detecciones de autobuses, fue tomada con igual o mayor prioridad, que la precisión de las mismas al momento de seleccionar el modelo de detección. Más adelante, en la Sección(6), se hará una comparación detallada sobre este tema.

Se determinó que un autobús, desde su aproximación a la parada, hasta su salida para retornar el recorrido, tarda en promedio entre 5 y 20 segundos; en relación a la capacidad de avistaje de una persona sin problemas de visión, de aproximadamente 60 metros de distancia.

A continuación, y para dar un contexto, se detallarán tres de los más populares modelos que se encuentran, a la fecha, en el *estado del arte* en materia de detección de objetos, y que son referentes en relación a los enfoques que cada uno persigue. Sus principios y fundamentos, sirvieron para encausar y sentar las bases argumentativas de la selección, en busca de satisfacer las necesidades antes mencionadas.

4.1.2. Detectores en el Estado de Arte

4.1.2.1. Faster R-CNN

Faster R-CNN (del *ingés*, Faster Region-based Convolutional Neural Network), es el modelo de detección de objetos basado en aprendizaje profundo que sirvió de base e inspiración para muchos de los modelos de segmentación y detección que vinieron después.

Para comprender su funcionamiento y arquitectura, primero es necesario comprender sus cimientos mediante sus predecesores R-CNN y Fast R-CNN.

R-CNN

R-CNN [20], es el que realmente comenzó todo. Mediante un algoritmo de búsqueda selectiva, selecciona aproximadamente 2000 ROI's de la imagen de entrada original.

Cada una de estas ROI es pasada a una ConvNet para generar un vector de características que alimenta: 1) un clasificador lineal, encarado de realizar la clasificación del recorte, y 2) un modelo de regresión lineal, para realizar las predicciones de las coordenadas del recuadro delimitador del objeto. Cabe aclarar que se necesita entrenar k clasificadores para cada una de las k clases de interés.

El resumen de este modelo se ilustra en la Figura (4.1), en la cual se propone regiones, se extraen sus características, y en base a estas, finalmente se clasifican.

Fast R-CNN

Fast R-CNN [23], fue el descendiente directo de R-CNN. Manteniendo la idea y forma de trabajo de su predecesor, se focalizó en mejorar la velocidad de las detecciones.

Para lograrlo, primero procedió a ejecutar la ConvNet sobre la imagen de entrada, generando un mapa de características, sobre el cual se realiza la selección de las aproximadamente 2000 ROI's. Esto, claramente visible en la Figura (4.2), es lo que principalmente aceleró el proceso de detección, ya que ahora, en lugar de realizar 2000 corridas individuales de la ConvNet, solamente se realiza una.

Posterior a ello, las ROI's atraviesan un capa POOL, la cual se encarga de redimensionar la altura y el ancho de cada una, a un tamaño fijo, para que puedan ser ingresadas a una sucesión de capas FC's.

Al final de estas últimas capas, se genera un vector de características por cada región, que ahora alimenta: 1) a un único clasificador, y como antes, 2) a un modelo de regresión lineal, que se encarga de determinar las coordenadas del recuadro delimitador del objeto.

Ya Conociendo las bases de su funcionamiento, la idea principal de **Faster R-CNN** [24] fue, reemplazar el algoritmo de búsqueda selectiva lenta de sus antecesores, por una red neuronal rápida llamada *Red de Propuestas de Región* (RPN).

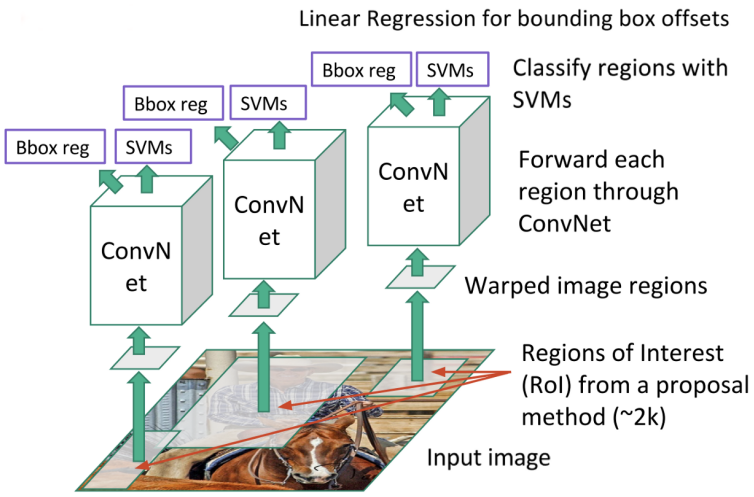


Figura 4.1: Resumen de etapas de modelo R-CNN (fuente: [22]).

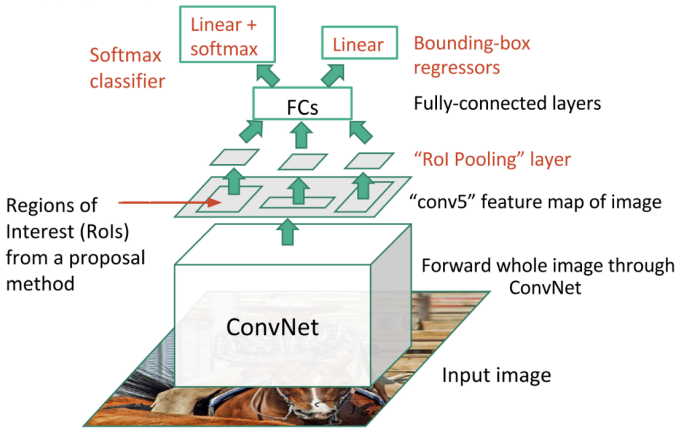


Figura 4.2: Resumen de etapas de modelo Fast R-CNN (fuente: [22]).

Al igual que antes, la imagen de entrada alimenta una ConvNet, pero ahora, el mapa de características obtenido se inserta en la RPN.

Sin entrar en detalles, RPN "observa", cada celda del mapa de características, y genera múltiples regiones de interés basadas en k diferentes rectángulos predefinidos de cierta altura y ancho.

Cada una de estas regiones propuesta, están formadas de un valor de certeza y de 4 coordenadas que representan el cuadro delimitador de la región. Aquellas regiones con un valor de certeza mayor a un umbral determinado, ingresan directamente a una sucesión de capas POLL y FC's, al igual que lo hacían en su antecesor Fast R-CNN, para generar un vector de características que alimenta finalmente un clasificador lineal y un modelo de regresión lineal, encargado de determinar los limites del los objetos detectados, Figura (4.3).

Con el paso del tiempo, la idea introducida por Faster R-CNN, de mejorar la velocidad compartiendo cálculos a través de una sola ejecución de la ConvNet, sería lo que motivo el surgimiento de su sucesor **R-FCN** [25] (del *inglés*, Region-based Fully Convolutional Net), el cual fue un paso más allá e implemento esta idea, pero por medio de una red convolucional de extremo a extremo.

4.1.2.2. YOLO (You Only Look Once)

YOLO [18], es un detector de objetos que utiliza características aprendidas por una ConvNet profunda para detectar un objeto. Este, fue el primer esfuerzo para obtener un detector en tiempo real y desde que se introdujo, se ha sometido a varias actualizaciones, siendo YOLO v3 [26], la última conocida a la fecha.

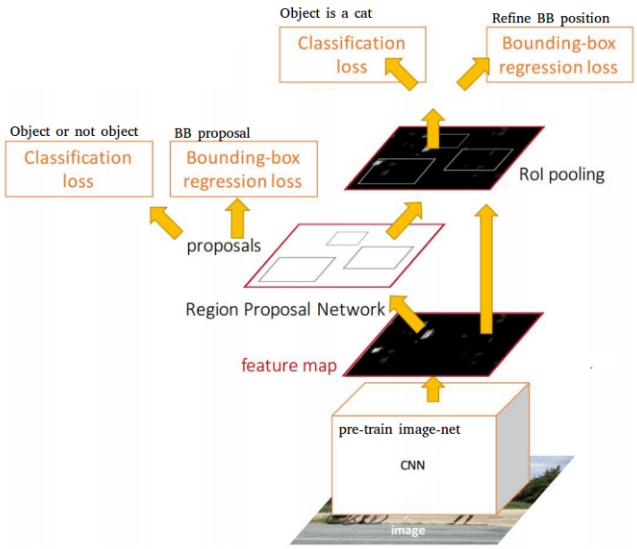


Figura 4.3: Resumen de etapas de modelo Faster R-CNN (fuente: [22]).

A diferencia de otros métodos de similares propósitos que operan en dos etapas: generación de regiones de interés y clasificación de las mismas, YOLO, usa solo una ConvNet para hacer todas las predicciones, tanto de las clases como de los recuadros delimitadores de los objetos detectados, en solo un paso *forward* de la red; de allí su nombre que significa *"Sólo miras una vez"*. Formalmente, se transfieren las características aprendidas por las capas convolucionales, a un clasificador para que realice tales predicciones.

Dependiendo de las diferentes sutilezas de cada versión, el algoritmo toma una imagen de entrada y la hace pasar a través de una ConvNet, retornando como salida, una grilla de $S \times S$ celdas llamada *«mapa de características»* que codifica las predicciones y cuyo tamaño es $\frac{1}{stride}$ veces el tamaño de la imagen de entrada original, donde el valor de S dependerá de tal factor de reducción (*stride*), el cual generalmente es igual a 32.

Codificación de salida: Estructura de mapa de características

El primer paso, necesario para comprender a YOLO, es conocer como codifica sus salidas. En él, la imagen de entada es "dividida" dentro de una grilla de $S \times S$ celdas en donde para cada objeto presente, sólo la celda que aloje el centro de dicho objeto, será la responsable de predecirlo.

Cada una de las celdas del mapa de características, tendrá una profundidad D dada por la siguiente ecuación:

$$D = \begin{cases} B * 5 + K & para\ YOLO \\ B * (5 + K) & para\ v2\ y\ v3 \end{cases} \tag{4.1}$$

En ambos casos, B representa la cantidad de *vectores de predicción* que cada celda puede determinar y está relacionado con la cantidad de *Anchor boxes* que se decidan usar en el modelo (Concepto explicado mas adelante). De ahora en adelante, y para evitar confusiones, se llamará *Box* a cada uno de tales vectores de predicción.

Cada Box se encuentra conformado por 5 predicciones $(p_c, b_x, b_y, b_h, b_w)$. En primer lugar, (b_x, b_y) representan las coordenadas del centro del recuadro delimitador del objeto detectado, relativos a la localización de la celda de la grilla responsable; (b_h, b_w) identifican los valores del alto y ancho del recuadro delimitador, normalizados a $[0, 1]$ y relativos a la dimensión de la imagen de entrada; y p_c representa el valor de certeza (o probabilidad de existencia) de que el recuadro delimitador predicho, contenga o no, un objeto determinado.

Por otro lado, las K probabilidades de las clases predichas se establecen, por cada celda, en caso de YOLO, y por cada Box, para YOLO v2 [27] y YOLO v3; haciendo en estos últimos dos, que a las 5 predicciones anteriores de cada Box se le agregar K parámetros adicionales, representativos

de las probabilidades de cada una de las k clases, para las que el modelo fuese entrenado; de allí, las diferencia observadas en las fórmulas de profundidades (4.1).

La Figura (4.4) ilustra a modo de ejemplo, el mapa de característica obtenido por YOLO v2 al procesar una imagen de entrada de 416×416 pixeles, con un *stride* de red igual a 32.

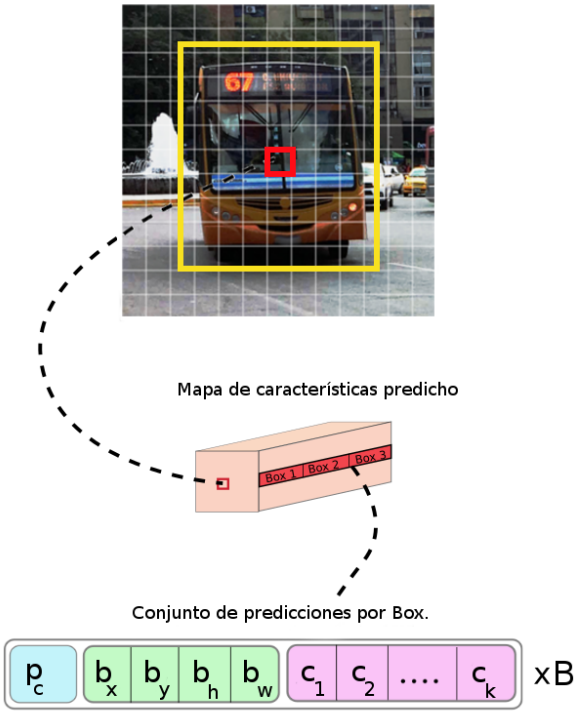


Figura 4.4: División de imagen de entrada en 13×13 celdas. La celda de color rojo, es responsable de la detección del autobús.

Anchor boxes y entrenamiento

Los *Anchor boxes* (cuadros o cajas de anclaje), son un conjunto de rectángulos predefinidos de cierta altura y ancho. Éstos, están definidos para capturar la escala y la relación de aspecto de las clases de objetos específicos que se desean detectar y generalmente se eligen en función de los tamaños de los objetos en el conjuntos de datos de entrenamiento, mediante técnicas de aprendizaje no supervisado de clusterización, Sección (1.3.2).

Las *cajas de anclaje,* surgen con como solución al problema de no poder detectar mas de un solo objeto por cada celda del *mapa de características.* La idea detrás de su funcionamiento, puede explicarse de una manera muy simple.

Supongamos que queremos entrenar un red de detección para las clases (c1 = 'bicicleta', c2 = 'persona', c3 = 'automóvil').

Como en todo modelo de aprendizaje supervisado, primero se debe etiquetar el conjunto de entrenamiento. Para el caso particular de YOLO, la creación de este conjunto se realiza como en cualquier otro modelo de detección, con la diferencia de que en lugar de crear un vector objetivo por cada imagen del conjunto, ahora se divide la imagen en $(S \times S)$ celdas (dependiendo de la precisión que se quiera obtener), y por cada una se crear un vector objetivo.

Supongamos que al querer etiquetar la i-ésima imagen del conjunto, $x^{(i)}$, ésta contiene una persona y un automóvil, y se da la coincidencia de que justo la (i,j)-ésima celda es el punto central de ambos objetos.

Por lo tanto, al generar el vector objetivo respectivo, tendríamos:

$$y_{(i,j)} = [p_c = 1, b_x, b_y, b_h, b_w, c_1 = 0, c_2 = 1, c_3 = 1]$$

Como se puede observar, es claro que no podemos representar a los dos objetos con el mismo

conjunto de parámetros para que definan su recuadro delimitador. Aunque se quisiera, las dimensiones de estas clases son muy diferentes y la precisión resultante del entrenamiento seria muy mala. Es aquí en donde entra en juego la idea de *cajas de anclaje*.

Sean B_1 y B_2 dos cajas de anclaje, definidas en base a los objetos de nuestro conjunto de entrenamiento. El nuevo vector objetivo tendría la siguiente estructura:

$$y_{(i,j)} = [(p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3), (p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3)]$$

donde la primera parte de los parámetros, estarán reservados a los objetos cuyas dimensiones posean valores de IoU mas altos en relación a B_1, que con B_2; y la segunda parte, para los objetos cuyas dimensiones otorguen índices de IoU mayores en relación a B_2, que a B_1.

Supongamos ahora (para simplificar la explicación) que B_1 tiene "forma" de rectángulo vertical, y B_2 "forma" de rectángulo horizontal. Es evidente, que la dimensión de la persona se asemejará más a la caja de anclaje B_1 que a la B_2, y viceversa para el caso del automóvil.

Por lo tanto, el nuevo vector objetivo debería ser codificado como sigue:

$$y_{(i,j)} = [(1, b_x, b_y, b_h, b_w, 0, 1, 0), (1, b_x, b_y, b_h, b_w, 0, 0, 1)]$$

donde la primera parte correspondería a la codificación de parámetros para el objeto 'persona', y la segunda para el objeto 'automóvil', Figura (4.5).

Para el caso de codificar un solo objeto, por ejemplo, una imagen con una 'bicicleta' (clase c1), cuyo centro se localiza en la celda (i,j)-ésima; el vector objetivo resultante se representaría de la siguiente manera:

$$y_{(i,j)} = [(0, ?, ?, ?, ?, ?, ?, ?), (1, b_x, b_y, b_h, b_w, 1, 0, 0)]$$

ya que es claro que las dimensiones de un objeto 'bicicleta' tendría mayo IoU con una caja horizontal (B_2), que con una vertical (B_1).

Notar que de esta idea se desprende la dimensión de profundidad B, mencionada anteriormente al describir la representación del *mapa de características*.

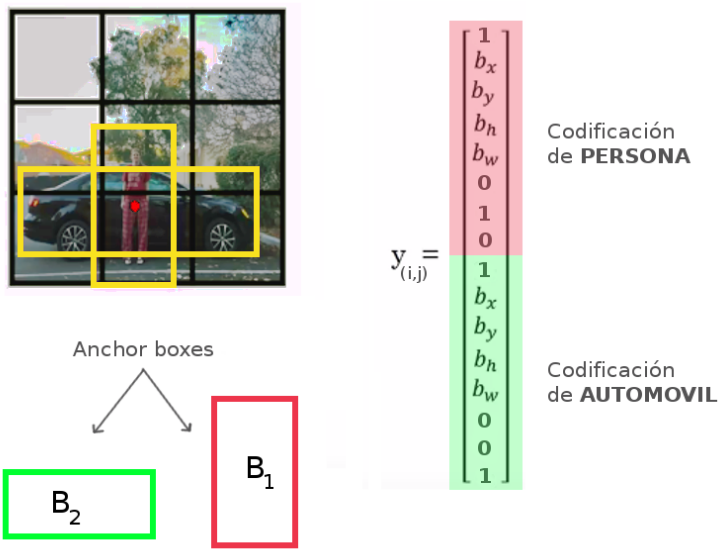


Figura 4.5: Ejemplo de vector objetivo, utilizando dos *Anchor boxes* para codificar dos objetos que comparten su centro en una misma celda del *Mapa de Características* (referencia: [28]).

Arquitectura

La arquitectura de la red YOLO está inspirada en el modelo *GoogLeNet,* Sección (2.4.4), para la clasificación de imágenes. Mediante algunas simplificaciones de ese clasificador, se obtuvo una

red de 24 capas CONV seguidas de 2 capas FC, Figura (4.6). Existe una versión rápida, llamada Fast YOLO, diseñada para incrementar la velocidad de las detecciones, utilizando una red neuronal con menos capas convolucionales (9 en lugar de 24). Una debilidad que presentaba esta primer versión de YOLO, era su baja precisión en comparación con los detectores de dos etapas de su época.

YOLO v2, tiene muchas ventajas en comparación con su anterior versión. Una de ellas, fue la implementación de *cajas de anclaje* para la determinación de los límites de los bboxes de cada objeto detectado. De esta forma supero a su antecesor, el cual basaba esta misma tarea mediante propuestas arbitrarias. Por otro lado, la arquitectura de la red también fue modificada, proponiendo un nuevo modelo, **Darknet-19**. Éste, esta formado por 19 capas CONV y 5 max POOL; en él, se removieron las últimas capas FC haciendo que la red sea totalmente convolucional (FCNN), otorgando de esta manera la capacidad de modificar la resolución de la imagen de entrada y por lo tanto, poder sincronizar precisión y velocidad.

La última mejora de YOLO, llamada YOLO v3, implemento el uso de una nueva red base para la extracción de características llamada **Darknet-53**, formada por 53 capas CONV, conexiones *«Skip»* y capas *«Upsampling»*.

Las capas *Skip* se encargan de realizar conexión entre capas alejadas en la ConvNet, tomando un volumen de salida de una y sumándolo (elemento a elemento) al volumen de entrada de otra capa mas alejada; mientras que una capa *Upsampling* se encarga de efectuar el proceso inverso a una capa POOL, es decir, toma una muestra y en lugar de reducir su dimensión, la aumenta, mediante agregado de ceros o interpolación de los valores.

Lo distintivo de esta nueva versión es que, mediante predicciones a 3 escalas diferentes, permite mejorar la detección de objetos pequeños; debilidad a la que estaban sujetas ambas versiones anteriores. Las dimensiones de los nuevos tres mapas de características de predicciones tienen un *stride* de 32, 16 y 8 respectivamente. Esto significa que con una imagen de entrada de 416×416 pixeles, hacemos detecciones a escalas de 13×13, 26×26 y 52×52.

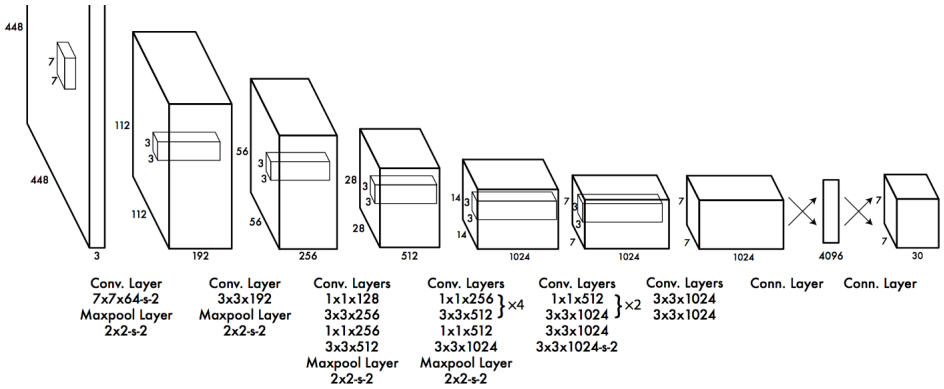


Figura 4.6: Arquitectura de red YOLO (fuente: Figura 3 de [18]).

4.1.2.3. SSD (Single Shot Multibox Detector)

SSD [19], fue el primer detector de una etapa en lograr una precisión razonablemente cercana a los detectores de dos etapas y al mismo tiempo, conservar la capacidad de trabajar en tiempo real.

Al igual que YOLO, SSD no se basa en métodos de propuesta de regiones, sino, en una ConvNet de principio a fin para predecir, tanto las clases como los cuadros delimitadores de cada detección.

La arquitectura de este modelo se compone de dos partes. La primera, consiste de una CNN base, la cual se utiliza para extraer características de las imágenes originales (específicamente de una VGGNet, Sección (2.4.4) sin sus capas FC; es decir, truncada antes de la etapa de clasificación).

La segunda parte de esta red, consta de una sucesión de capas convolucionales que progresivamente van disminuyendo de tamaño y produciendo mapas de características de diferentes escalas, a distintas profundidades de la red, Figura (4.7). Cada una de estas capas "multiescala", reduce la altura y el ancho del mapa de características proporcionado por la capa anterior.

De esta manera, cuanto más cerca del final del *pipeline* se encuentren, más pequeño será su mapa de características de salida, permitiendo una mejor detección de objetos pequeños. Contrariamente, mientras más cerca se encuentren de la red base, mayor será el campo receptivo de cada elemento en el mapa de características y más adecuado para detectar objetos de mayor tamaño.

Finalmente, por cada celda de cada uno de los mapas de características, se usa un pequeño kernel 3×3 para simultáneamente predecir las probabilidades de clases y los límites de la detección, en base a un conjunto fijo de recuadros (parecidos a los Anchor boxes utilizados en Faster R-CNN).

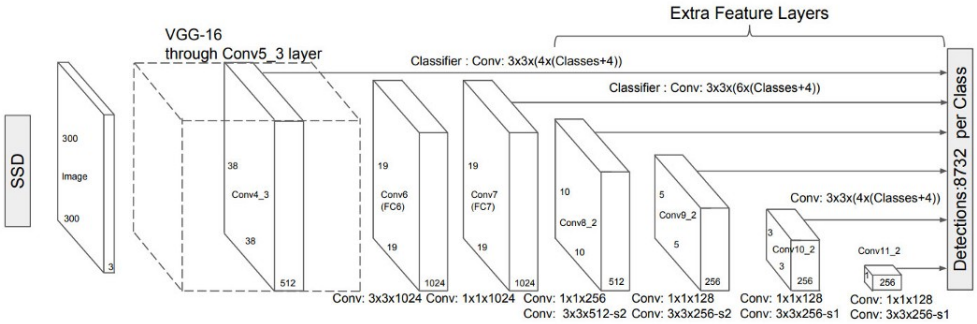


Figura 4.7: Arquitectura de la red SSD (fuente: Figura 2 de [19]).

4.1.3. Implementación

Para el reconocimiento de autobuses, el siguiente trabajo implementó y testeó, Sección (6), diferentes variaciones de los siguientes tres modelos: YOLO v2, YOLO v3 Tiny y SSD-MobileNet 300. A priori, y en relación a los fundamentos y enfoques descriptos anteriormente, fueron considerados por presentar mejores adaptaciones para la satisfacción del problema. Es decir, la velocidad y las capacidades de detección a tiempo real primaron por sobre la precisión, haciendo que Fast R-CNN, mucho mas preciso que sus competidores, quedara fuera de tal selección por su enfoque de dos etapas y consecuente relativa lentitud en sus detecciones.

YOLO v2, fue realizado en lenguaje Python mediante un wrapper de una recompilación del framework de redes neuronales Darknet[8]. Esta recompilación, se hizo necesaria para que sus métodos permitieran la entrada de imágenes en formato *numpy.ndarray* en lugar de solamente path absolutos de los nombre de las imágenes. Darknet es la implementación oficial escrita en lenguaje C con CUDA, creada por las mismas personas detrás del algoritmo.

La codificación del paso *forward* de la red, fue realizado en base al conjunto de pesos preentrenado por los mismos creadores de este framework, sobre COCO dataset[9].

YOLO v3 Tiny, es la versión liviana de YOLO v3. Básicamente, esta red puede satisfacer los requisitos en tiempo real basados en recursos de hardware limitados. Similar a su hermano mayor, también realiza predicciones a múltiples escalas pero solo sobre 2 dimensiones de mapas de características (ej. 13×13 y 26×26, si se parte de un tamaño de imagen de entrada de 416×416 pixeles).

Al igual que con YOLO v2, éste, fue implementado en lenguaje Python y Darknet, utilizando también un conjunto de pesos preentrenados por los creadores del framework sobre COCO dataset.

Por último, se implemento SSD-MobileNet 300; una red que combina la arquitectura MobileNet, Sección(2.4.4), y el modelo de detección SSD, Sección (4.1.2.3), para lograr un método rápido y eficiente basado en el aprendizaje profundo para la detección de objetos.

La construcción de este detector fue también realizada en lenguaje Python, mediante la utilización del módulo de redes neuronales profundas (dnn) de la librería OpenCV 4.0[10]. El modelo utilizado fue una versión Caffe basada en la implementación original en TensorFlow[11] de *Howard et al.* el cual fue entrenado por *chuanqi305*[12], primero sobre el conjunto de datos COCO dataset y luego ajustado sobre PASCAL VOC[13].

[8] `https://github.com/pjreddie/darknet`
[9] `http://cocodataset.org/`
[10] `https://opencv.org/opencv-4-0/`
[11] `https://github.com/Zehaos/MobileNet`
[12] `https://github.com/chuanqi305/MobileNet-SSD`
[13] `http://host.robots.ox.ac.uk/pascal/VOC/`

Cabe destacar que los valores de los pesos preentrenados, utilizados para cada uno de los modelos, ya consideraban la categoría 'bus', dentro de tales entrenamientos.

4.2. Detección de números de líneas

La detección de texto en entornos restringidos y controlados, generalmente puede lograrse mediante el uso de enfoques basados en heurísticas que aprovechan el hecho de que el texto, normalmente se agrupa y organiza en párrafos, y los caracteres, aparecen en línea recta. Sin embargo, y como ya se lo postuló en la Sección (3), la detección de texto en escenas naturales es diferente y mucho más desafiante.

De manera similar a los sucedido con los modelos para la localización de autobuses, se encuentran diversas propuestas formando parte del *estado del arte* para llevar a cabo esta tarea. Trabajos como TextBoxes++ (A Single-Shot Oriented Scene Text Detector) [29], SegLink (Detecting Oriented Text in Natural Images by Linking Segments) [30], EAST (Efficient and Accurate Scene Text detector) [31] y DMPNet (Deep Matching Prior Network) [32], son algunos de los modelos y métodos de detección más empleados y que mejores resultados otorgan.

Según literaturas del campo concerniente, en los últimos años, EAST viene siendo el modelo a seguir y una elección natural si se busca velocidad y precisión en las detecciones.

Dado que su modelo de aprendizaje profundo es de extremo a extremo, mediante su aplicación, es posible evitar subalgoritmos computacionalmente costosos y eludir la agregación de candidatos y el particionamiento de palabras, que otros detectores de texto suelen aplicar. Sumado a esto, la facilidad proporcionada por las últimas versiones de la librería OpenCV 3.4 y OpenCV 4, las cuales proveen un modelo serializado preentrenado en formato *protobuf* (el cual contiene tanto la definición de la arquitectura como los pesos de este modelo), hacen de EAST el candidato ideal para afrontar la tarea de localización de números de líneas de autobuses.

4.2.1. EAST (Efficient and Accurate Scene Text detector)

EAST, es una variante de los métodos de predicción basados en el uso de *Anchor boxes* predefinidos, que sigue los fundamentos subyascentes de un modelo detector de objetos.

A diferencia de la red SSD estándar, en donde varios mapas de características de diferentes tamaños son utilizados para realizar las detecciones en base a recuadros predefinidos; en EAST, todos los mapas de características son integrados mediante progresivas capas *Upsampling,* o estructura U-Net [33] para ser más específicos.

Según los autores, el *pipeline* de EAST es capaz de predecir palabras y líneas de texto en orientaciones arbitrarias en imágenes de 720p, y además, puede ejecutarse a 13 FPS.

Arquitectura y estructura de predicciones

A grandes rasgos, el modelo podría ser descompuesto en tres partes: modulo extractor de características, fase de fusión de mapas de características y capa de salida.

Las dos primeras no ameritan mucha explicación. En relación al módulo de salida, éste, se encarga de construir y retornar el mapa de características final cuyo tamaño es de $\frac{1}{4}$ veces el tamaño de la imagen de entrada original, siendo requisito que las dimensiones de la imagen de entrada sean múltiplos de 32.

Sea $W \times H$ la dimensión original de la imagen de entrada, el mapa de características retornado será un volumen de características de tamaño $(\frac{W}{4} \times \frac{H}{4} \times c)$, donde c indica la cantidad de canales, mediante los cuales se entregan las predicciones realizadas.

Específicamente, el primero de los canales, representa el mapa de probabilidades (o puntajes de certeza), que indican la existencia o no de texto en la región; los restantes, son los encargados de entregar los parámetros que describen la geometría de los recuadros delimitadores del texto localizado, en dos formatos diferentes, *RBOX* y *QUAD*.

Para *RBOX*, la geometría está representada por 5 canales. Los primeros 4 indican los valores de las 4 distancias, relativas a la dimensión del mapa de características, tomadas en función del punto central del recuadro que limita la zona de texto detectada y los límites superior, derecho, inferior e izquierdo del mismo. El último canal, entrega el ángulo de rotación ϑ (en radianes), del recuadro delimitador con respecto al eje horizontal de la imagen.

Por el otro lado, QUAD, representa la geometría mediante 8 canales, los cuales denotan los cuatro pares de coordenadas (x, y) que indican la posición de los vértices del cuadrilátero que delimita la región del texto detectada, relativos a su punto central.

La estructura de la red EAST, puede ser vista en detalle en la Figura (4.8).

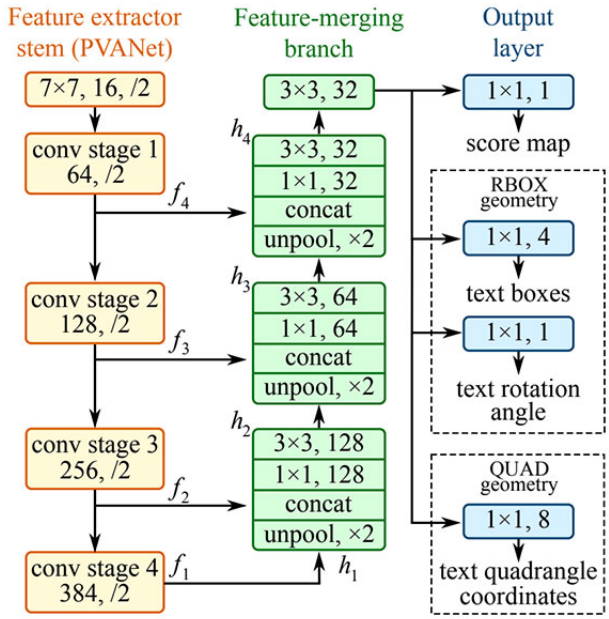


Figura 4.8: Estructura de Red Completamente Convolucional de detector de texto EAST (fuente: Figura 3 de [31]).

4.2.2. Implementación

La construcción de este detector fue realizado en lenguaje de programación Python, mediante la utilización principal del módulo (dnn) de redes neuronales profundas de la librería OpenCV 4.0 y la librería Numpy.

Numpy, fue fundamental para la vectorización general de las etapas, principalmente en la decodificación de las detecciones y geometrías RBOX, y en la implementación del algoritmo Non-Maximum Suppression, en pos de acelerar los cálculos computacionales.

Por otro lado, y como anteriormente mencionamos, OpenCV nos proveyó del modelo preentrenado, el cual en orden de efectuar la detección de texto, luego de la codificación del paso *forward* de la red, nos hizo posible obtener nuestro mapa de características de salida mediante las capas «*feature_fusion/Conv_7/Sigmoid*» y «*feature_fusion/concat_3*»; donde la primera es la capa de salida de activación sigmoidea, que provee la probabilidad de que una región contenga texto o no, y la segunda, el mapa de entidades que representan las "geometrías" de cada una de las detecciones realizadas.

4.3. Reconocimiento de números de líneas

Aunque los sistemas de reconocimiento de caracteres y/o palabras ya no son una novedad, éstos, siguen formando parte activa de múltiples investigaciones en la literatura de visión por computadora, especialmente cuando se aplican a imágenes de entornos no restringidos del mundo real. Si bien el aprendizaje profundo y las redes neuronales convolucionales permiten obtener excelentes precisiones de texto, mucho más allá de los métodos tradicionales de extracción de características y aprendizaje automático, todavía se esta muy lejos de ver sistemas "casi perfectos". Además, y debido a las múltiples áreas de aplicación que poseen, algunos de los mejores algoritmos empleados para esta tarea son de uso comercial y requieren licencias para ser utilizados en proyectos propios.

Tomando lo anterior como punto de partida, este proyecto optó por utilizar el motor de reconocimiento de caracteres Tesseract v4.1.

Tesseract[14], es un motor de OCR muy popular, el cual fue desarrollado originalmente por Hewlett Packard en la década de 1980 y luego fue de código abierto en 2005. Google adoptó el proyecto en 2006 y lo ha estado patrocinando desde entonces. Éste, y a diferencia de sus versiones anteriores, incluye un nuevo motor de reconocimiento basado en redes neuronales, que ofrece una precisión significativamente mayor, a cambio de un aumento significativo en la potencia de procesamiento requerida. El motor OCR subyacente utiliza una red de memoria a corto plazo (LSTM), la cual es una tipo de red neuronal recurrente (RNN).

> *En pocas palabras, una **Red Neuronal Recurrente** (RNN), es una clase de red neuronal artificial que integra bucles de realimentación, permitiendo que la información persista durante algunos pasos de entrenamiento, a través de conexiones desde las salidas de las capas, que "insertan" sus resultados en los datos de entrada.*

La implementación de este motor de reconocimiento fue realizado en lenguaje de programación Python mediante el uso de la librería Pytesseract, la cual hizo de enlace para que nuestro script pueda comunicarse con el binario Tesseract v4.1, y de esta manera poder realizar el reconocimiento de líneas de autobuses en las diferentes imágenes de entrada del modelo. En tal implementación, el uso de multithreading, como paradigma de programación, fue parte fundamental del proceso. Mediante una estructura de cola de prioridades, los potenciales ROI's contenedores de texto fueron alimentados en paralelo a los diferentes hilos otorgando un gran incremento en la velocidad final del sistema.

4.4. Módulos intermedios

Si bien Tesseract puede funcionar muy bien en condiciones controladas, éste podría realizar decodificaciones no muy certeras, si hay una cantidad significativa de ruido o si la imagen no se procesará y limpiará correctamente. En la mayoría de los casos, aproximaciones basadas en métodos de segmentación, son aplicadas para intentar "separar" el texto de primer plano de su ambiente. Estas segmentaciones deben tratar de tener la mayor resolución posible y los caracteres en la imagen de entrada deberán aparecer lo menos "pixelados" posible después de tal procedimiento, de lo contrario, Tesseract tendrá dificultades para reconocer correctamente el texto, incluso aunque se tratase de imágenes capturadas en condiciones ideales (digitalizaciones).

Las debilidades mencionadas anteriormente, también aplican al módulo de detección, ya que los ruidos y fondos variopintos que caracterizan a las imágenes de entrada, limitan su máxima eficiencia.

Tanto el módulo de *transformación de espacios de color,* situado inmediatamente después de la etapa de detección de autobuses; como el que se encarga de la aplicación de los *algoritmos de segmentación,* ubicado antes del la etapa de reconocimiento de texto; son aplicados con el fin de tratar de reducir tales debilidades y afinar el sistema final.

A continuación se describirán detalles de cada una de estas etapas, incluyendo sus bases de implementación.

4.4.1. Transformación de espacio de colores

Particularmente, y al igual que en muchas otras aplicaciones de visión por computadora, la selección de un buen espacio de color (o subconjunto de ellos), fue fundamental y destinado a tratar de abolir los efectos producidos por las diferentes condiciones de iluminación (a las cuales se vieron sometidas las capturas de los autobuses) y para ayudar en la importante tarea de destacar los números de sus líneas; todo ello en pos de facilitar futuras detecciones.

Aunque los detalles de los espacios utilizados serán desarrollados a lo largo de la Sección (5), este bloque fue el encargado de generar nuevas representaciones de imágenes en base a probar y analizar una batería de éstos, y seleccionar los que mejores resultados otorgaran, Secciones (5.4.1.1) y (5.4.2.2).

La codificación de esta etapa intermedia, al igual que el resto de los módulos, fue implementada en lenguaje de programación Python por medio de los métodos *cvtColor* y *filter2D*, incluidos en la librería OpenCV, los cuales permitieron realizar las transformaciones de los respectivos espacios de colores y la codificación de los diferentes filtros.

[14] `https://opensource.google/projects/tesseract`

4.4.2. Algoritmos de segmentación

El objetivo de una segmentación es simplificar y/o cambiar, la representación de una imagen, en otra más significativa y más fácil de analizar. En nuestro caso puntual, el método de segmentación, fue añadido culminado el pipeline del sistema para tratar de obtener mejores resultados.

El mayor desafío presentado estuvo relacionado a como, en su mayoría, estaban representados los números de líneas de los autobuses de la ciudad de Córdoba. Éstos, y a diferencia de muchos lugares en el país, en donde son pintados o estampados con vinilos, se forman mediante una agrupación de diodos led, que si bien se encuentran muy cerca unos de otros, los movimientos de capturas, las diferentes condiciones de iluminación, los reflejos producidos en el parabrisas y el brillo mismo que estos emiten, generan todo un reto al tratar de identificar a este grupo, como un objeto representativo de un carácter numérico.

En base a las debilidades descriptas para el modelo de reconocimiento, se encontró de manera empírica, que la aplicación del método de Valor Umbral Único (umbralización), podía en circunstancias, otorgar mayor legibilidad a los recortes de las regiones de texto arrojados por el módulo de detección.

> *Básicamente, la* ***umbralización****, es un método de segmentación que consiste en fijar límites, de forma tal, que los pixeles que se encuentre entre cada par de estos límites, formen un objeto.*

Un problema que acarreó tal umbralización, lo cual se pensó podría derivarse de los destellos emitidos por los diodos led que forman los números de líneas, fue que, en algunas circunstancias, los resultados obtenidos se perciban como si de "números pixelados" se tratasen; ésto introdujo una nueva dificultad.

Una aproximación, para tratar de eliminar esta percepción, fue la aplicación de un filtro Gaussiano. Éste, permitía difuminar los límites de los números y, en parte, abolir tal apariencia.

> *Un* ***Filtro Gaussiano****, también conocido como desenfoque Gaussiano, es uno de los filtros lineales más utilizados para la reducción de ruido y el suavizado de imágenes. Tales resultados, son logrados mediante la aplicación de una operación de convolución entre: la imagen y un kernel (o filtro), cuyos valores se corresponden a una distribución Gaussiana.*

4.4.2.1. Implementación

Para esta instancia, se implementó el algoritmo de Valor Umbral Único mediante los método *cvtColor y threshold* de la librería OpenCV. El primero, para convertir la imagen a escala de grises y el segundo, en conjunto con los modos *THRESH_BINARY* y *THRESH_BINARY_INV,* para obtener una umbralización normal y otra invertida. La fijación de los límites estuvieron entre los valores 150 y [180 | 255] , dependiendo de que transformación de imagen proviniera la detección.

Para la implementación del filtro Gaussiano, se utilizó el método *GaussianBlur*, también incluido en la librería OpenCV, mediante el uso de un kernel de dimensión 3×3 para que el desenfoque efectuado, no afectara la representación numérica original.

En la Figura (4.9) se muestra la sucesión de transformaciones que sufre una imagen, desde que entra al sistema, hasta que se retorna el resultado final, una vez atravesados todos los módulos y etapas que fueron descriptas en la Sección (4).

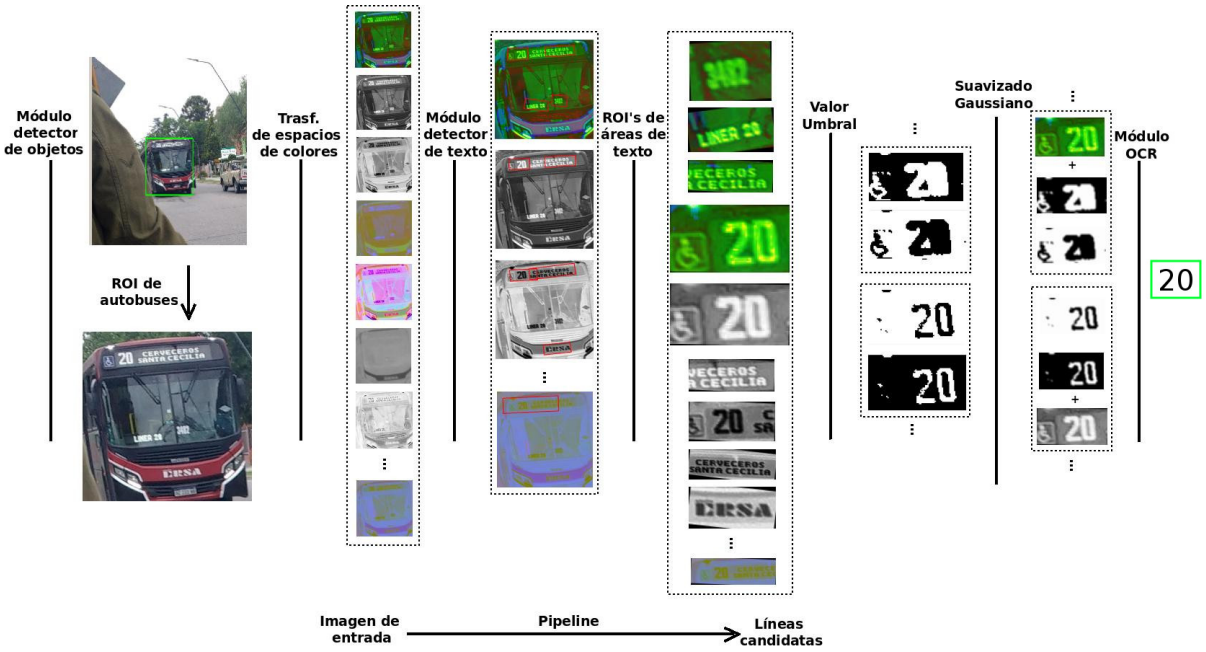


Figura 4.9: Secuencia de transformaciones sufridas por una imagen de un autobús, desde que entrada al sistema, hasta que se realiza la predicción del número de su línea.

5. Etapas de detección y reconocimiento de líneas: Ajustes

Una vez concluido el proceso de selección de los diferentes modelos y métodos, y habiendo determinado el diseño de la arquitectura del sistema, se procedieron a realizar diversas pruebas y análisis en pos de afinar y ajustar cada una de las partes, con la finalidad de obtener resultados acordes al objetivo de desarrollo del presente trabajo.

Antes de comenzar, la Figura (5.1), describe el diagrama de flujo del sistema completo puntualizando, tanto en los módulos principales (áreas de color verde), como en las distintas etapas intermedias (delimitadas por líneas punteadas), a fin de tener un panorama más amplio de lo que será analizado y puesto a prueba a lo largo de esta sección.

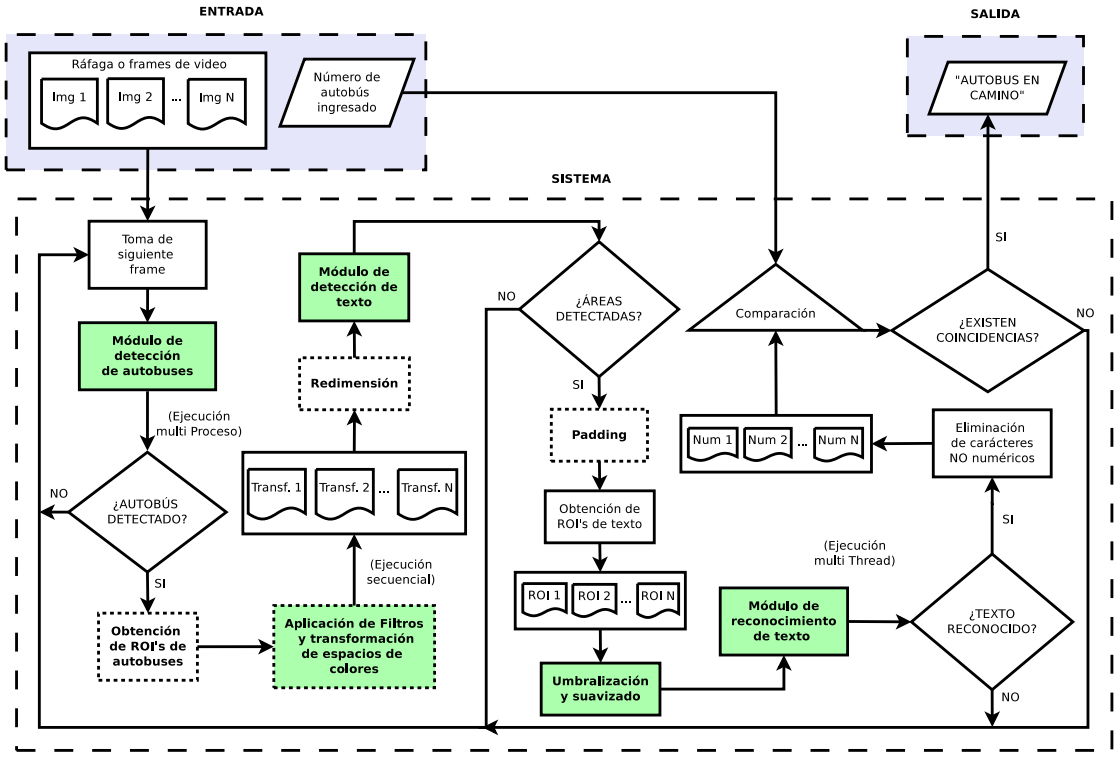


Figura 5.1: Diagrama de flujo del sistema de extremo a extremo.

Concretamente, se analizarán y probarán, los parámetros de todas las partes involucradas en la tarea de *detección y reconocimiento de texto*, EAST y OCR-Tesseract respectivamente, asumiendo que se tiene un modelo de *detección de autobúses* "teórico y preciso". Es decir, se dejará este último de lado y se alimentará manualmente al sistema con las coordenadas de los ROI's de autobuses, permitiendo de esta manera, aislar e independizar los resultados entregados de posibles errores provenientes de imprecisiones introducidas por el módulo de detección de objetos. En la Sección (6) se probarán diferentes variantes para este modelo.

Las pruebas y experimentos realizados, fueron orientados para encontrar las mejores combinaciones de valores de parámetros que mayor precisión y porcentaje de *«Detecciones Positivas»* le otorgarán al sistema.

> *"De ahora en adelante, nos referiremos como* ***Detección Positiva****, a toda aquella predicción correcta que es arrojada por una determinada combinación de parámetros. Es decir, una vez que la imagen es procesada por el sistema, su predicción es igual al número real ingresado del autobús analizado"*

Las mismas, se realizaron sobre una muestra poblacional de 100 imágenes de autobuses, las cuales fueron meticulosamente seleccionadas de un total de 983. Para tratar de asegurar heterogeneidad y representatividad, tal selección, destino su esfuerzo a reunir el conjunto de imágenes que mejor representara y/o rescatara: factores ambientales de iluminación, técnicos derivados de las fuentes de captura, y aquellos referidos a los niveles de estabilidad al momento de la toma (estática o en movimiento). Tal conjunto contó con dimensiones de 640×480, 960×1280, 1280×960, 2340×4160,

3120×4160, 1881×2508, 3020×4032, 3464×4618 y 3000×4000 pixeles, provenientes de las fuentes de teléfonos móviles: HUAWEI VNS-L23, Motorola MotoG3, Apple iPhone 7 Plus, Sony F5121, Xiaomi Mi A1, Apple iPhone 8 Plus, Samsung Galaxy A5, Apple iPhone 6s y Samsung Galaxy J3; incluyendo imágenes referentes a diversos horarios y condiciones meteorológicas, tales como días soleados, nublados, nocturnos y con leves lloviznas.

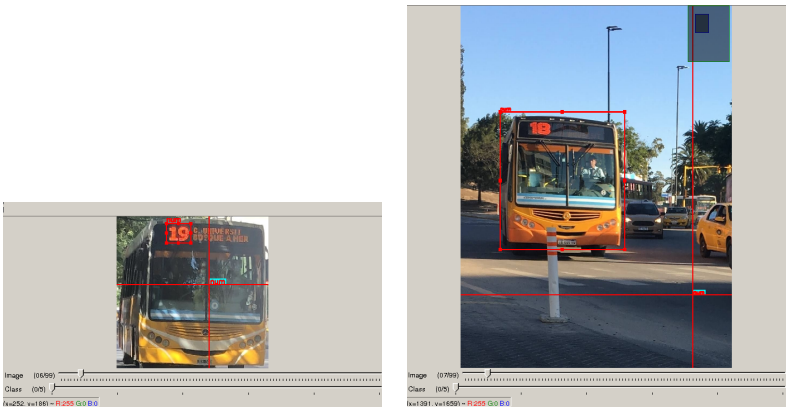
5.1. Anotación de coordenadas

A efectos de suplantar el modelo de detección encargado de localizar los autobuses, se realizaron las anotaciones manuales de los recuadros que delimitaban las regiones en donde se encontraba cada autobús, por cada una de las 100 imágenes de prueba.

En cada anotación, se almacenaron los cuatro valores que identificaban al vértice superior izquierdo e inferior derecho del ROI del autobús en cuestión; estos serían retornados, como si del mismísimo modulo de detección se tratase, para localizar el autobús, cada vez que se alimentara al sistema con alguna imagen de prueba. De manera similar, fue repetida la misma tarea, pero para la anotación de las coordenadas de las regiones ocupadas por los *números de líneas* de los autobuses. Como se verá en posteriores secciones, estas coordenadas permitieron estimar la precisión de las detecciones realizadas por el modelo EAST.

Tal procedimiento, fue implementado mediante la utilización de la herramienta *OpenLabeling*[15]; una aplicación open source codificada en Python, la que por medio de su interfaz gráfica, hizo de tal proceso, un trabajo menos tedioso.

Posteriormente, los archivos .xls generados se parsearon y almacenaron en DataFrames, mediante el uso de la librería Pandas. La Figura (5.2), muestra el proceso de anotación, tanto para ROI's de autobuses, como para los números de sus líneas.



(a) ROI de número de línea. (b) ROI de autobús.

Figura 5.2: Generación de anotaciones mediante *OpenLabeling*.

5.2. Lista de parámetros analizados

Los parámetros que se analizarán, se corresponden y forman parte, de los cuatro bloques señalados con lineas punteadas dibujados en el diagrama de flujo presentado anteriormente.

Para entrar en contexto, y a fin de explicar las nuevas etapas de «*Redimensión*» y «*Padding*», cuando un autobús es detectado, el bloque de *'Obtención de ROI's de autobuses'*, posterior al *módulo detector*, se ocupará de realizar importantes tareas en relación a los límites de los recuadros delimitadores de las detecciones que hayan sido realizadas. Por un lado, filtrará los autobuses detectados en función de la relación de aspecto de su cuadro delimitador; aunque parezca que se reducen las posibilidades con tal implementación, se gana al descartar aquellas detecciones de autobuses laterales, donde en la mayoría de los casos, no se encuentra el número de linea impreso. Por tal motivo, y posteriormente detallado en la Sección (5.6), se tomarán en cuenta solo relaciones de aspecto que se identifiquen con autobuses viniendo de frente o levemente lateralizados.

[15] `https://awesomeopensource.com/project/Cartucho/OpenLabeling`

Por otra parte, una vez superado el filtrado inicial, este bloque, tendrá la tarea de "mover" los límites de tal ROI, hasta que su ancho y alto respectivos obtengan una relación de aspecto cuadrada. Esta pequeña modificación, además de no interferir en los resultados, permite unificar dimensiones y fue pensada para facilitar los análisis realizados en la Sección (5.5), la cual tuvo como principal objetivo, encontrar patrones estadísticos que relacionen los diferentes tamaños originales de los recortes con el porcentaje de detecciones positivas que de ellos se derivaban.

El ROI final depurado, será sometido a un total de cincuenta y dos (52) transformaciones de imágenes, permitiendo analizar cuales de ellas efectuará mejor la tarea de exposición de regiones de texto. Pasada esta etapa, cada una será redimensionada a seis (6) nuevos tamaños por el módulo de *'Redimensión'*. Ésto, permitirá chequear que tamaños de entrada a la red EAST entregan los mayores porcentajes de detecciones.

Para finalizar, una vez localizadas las áreas de texto, podría suceder que los límites de tales detecciones se encuentren tan cerca de los caracteres que, por ejemplo, causen que Tesseract prediga ***'o'*** en lugar de ***'9'*** *o* ***'6'***. Para solucionar este inconveniente fue implementada, previo a la entrega de los ROI's de texto, la etapa de *'Padding'.* En ella, se probarán cuatro (4) diferentes porcentajes de "padeo", ampliando los márgenes de los límites de detección, con el fin de seleccionar el que en promedio, permita mejores resultados al momento de decodificar la imagen a texto plano (a número de linea, en nuestro caso).

A continuación se detallan los diferentes valores de parámetros a probar, para cada uno de los módulos mencionados.

Transformaciones y filtros de imágenes

Los siguientes, representan las siglas de 7 espacios de colores y 2 filtros de imágenes, el resto, se corresponde a los canales derivados de cada uno de los espacios. Cuando se escribe *s-hls,* se hace referencia a la trasformación de imagen que resulta de aplicar la conversión $rgb \rightarrow hls$, tomar el canal s, y copiarlo en los dos restantes. Dicho de otra manera, nuestra nueva representación sería de la forma sss_{hls}, es decir, con tres canales s provenientes del espacio de color *hls.*

Por otro lado, un sigo '-' inicial (por ejemplo '-'h-hls), indica la aplicación de la inversa a la transformación (es decir, mapea zonas brillantes a oscuras y viceversa).

Por último, *edges* es un filtro que se codificó para realizar detecciones de bordes horizontales y verticales, resultado de convolucionar la imagen con dos kernels 3×3.

- *rgb, -rgb, r-rgb, -r-rgb, g-rgb, -g-rgb, b-rgb, -b-rgb,*
- *hls, -hls, h-hls, -h-hls, l-hls, -l-hls, s-hls, -s-hls,*
- *hsv, -hsv, s-hsv, -s-hsv, v-hsv, -v-hsv,*
- *lab, -lab, l-lab, -l-lab, a-lab, -a-lab, b-lab, -b-lab,*
- *yuv, -yuv, y-yuv, -y-yuv, u-yuv, -u-yuv, v-yuv, -v-yuv,*
- *YCrCb, -YCrCb, Cr-YCrCb, -Cr-YCrCb, Cb-YCrCb, -Cb-YCrCb,*
- *luv, -luv, u-luv, -u-luv, v-luv, -v-luv,*
- *edges, -edges*

Redimensión:

De ahora en más, se referirá a este parámetro como *newsize* y hará referencia al tamaño que tendrá la imagen al momento de entrar al *módulo de detección de texto.*

Dado que EAST, necesita que las dimensiones de las imágenes de entrada a su red sean múltiplos de 32, se probarán los tamaños de 64×64, 96×96, 128×128, 160×160 y 320×320 pixeles. La utilización de dimensiones cuadradas, es consecuente con la decisión tomada con respecto al aspecto que deben tener los ROI's de los autobuses, para de esta forma evitar posibles problemas de detección, consecuencia de las deformaciones en la imagen, y por consiguiente del texto, que se podrían provocar.

Padding

La etapa de *padding*, en mención a su nombre, ampliará los márgenes entre el texto y los límites de cada detección realizada por el módulo EAST. Los porcentajes de pixeles a agregar, a cada una de las detecciones realizadas, serán de 0 %, 5 %, 10 % y 15 %. Este porcentaje, de ahora en adelante denominado *pad*, será aplicado tanto en ancho como en alto.

5.3. Representación de datos de prueba

Con el fin de obtener las mejores combinaciones de valores que otorguen el mayor rendimiento posible, inicialmente se hizo necesario buscar una manera de representar y estructurar, tanto los valores de los parámetros probados, como los resultados entregados por el sistema, consecuencia de tales configuraciones.

Para ello, se decidió confeccionar un dataset que contuviera los resultados de cada etapa, para cada una de las 100 imágenes de autobuses de la muestra poblacional descripta en la sección anterior. Cada uno, se generó mediante los resultados retornados por el sistema para cada posible combinación de parámetros, es decir, para todas las combinaciones existentes entre: las 52 transformaciones de imágenes, los 6 valores de *newsize* y los 4 valores de *pad.*

Específicamente, cada dataset se estructuró de la siguiente manera:

- $realBusNumber$: Número real de autobús.
- $Threshold_{conf}$: Umbral utilizado por Non-Max Suppression en EAST.
- $Threshold_{IoU}$: Umbral utilizado para el computo del índice IoU en EAST.
- Lista de túplas de la forma $[[result, transformId, originsize, newsize, pad, prediction], ...]$ donde:
  - *result:* Valor $\epsilon \{0, 1\}$, donde 1 indica $prediction = realBusNumber$, 0 c.c
  - *transformId:* Valor $\epsilon \{0, .., 51\}$ que representa el *id* de la transformación de imagen utilizada.
  - *originsize:* Valor correspondiente a la dimensión original del recorte del autobús detectado.
  - *newsize:* Valor correspondiente a la dimensión del recorte del autobús, luego de atravesar la etapa de *Redimensión.*
  - *pad:* Valor $\epsilon \{0, 5, 10, 15\}$, utilizado en la etapa de *Padding*, como se explicó en la sección anterior.
  - *prediction:* Detección realizada por *el módulo de reconocimiento de texto*, luego de ser removidos los caracteres no numérico y posterior conversión a la clase entero.

5.4. Aproximación en la búsqueda de los mejores parámetros

En esta sección, se enumerarán y describirán las pruebas y algoritmos empleados para tratar de dar con las mejores combinaciones de parámetros. Tales análisis, serán realizados partiendo de dos enfoques diferentes, donde en cada uno, se opta por una manera diferente de como comenzar a transitar el pipeline del sistema.

En el primer acercamiento, se partió de los ROI's de autobuses originales, de dimensiones cuadradas, retornados por el bloque de *"obtención de ROI's de autobuses",* como ya se detalló en la Sección (5.2). El segundo, y a diferencia del primero, en lugar de iniciar con los recortes completos, lo hará desde una subregión particular de los mismos, la cual, basada en estadísticas, es la que potencialmente debería contener el número de línea del autobús; esto, con el objetivo de centrar la atención sólo en aquellas partes importantes de la imagen y aumentar las posibilidades de predicción, Sección (5.4.2.1).

A lo largo de todo el desarrollo, se puntualizará en buscar el mínimo conjunto de mejores transformaciones de imágenes, y a partir de ellas, las tuplas de la forma $(newsize, pad)$ que arrojen mayores porcentajes de detecciones positivas. Además, y en base a los resultados obtenidos, los valores de los mejores parámetros encontrados, serán tomados como base en posteriores desarrollos de heurísticas y análisis, que ayuden a afinar aún más el sistema.

5.4.1. Primer enfoque

Este enfoque toma como punto de partida, para el análisis y la creación de los dataset, los recortes originales de aspecto cuadrado de cada autobús. Algunos de los ROI's utilizados pueden encontrarse en Anexo (V).

Las configuraciones de los módulos principales fueron las siguientes:

- Recortes de autobuses: Originales, de aspecto cuadrado.
- Redimensión
  - *newsize*: 64×64, 96×96, 128×128, 160×160 y 320×320 pixeles.
- Módulo detector de texto (EAST)
  - $Threshold_{conf}$: 0,001.
  - $Threshold_{IoU}$: 0,1.
- Padding
  - *pad*: 0 %, 5 %, 10 % y 15 %.
- Módulo de reconocimiento de texto (OCR-Tesseract)
  - *lenguaje*: Inglés.
  - *psm* (Modo de Segmentación de Página): Modo 7, el cual trata a la imagen como una simple línea de texto.
  - *oem* (Modo del Motor OCR): Modo 1, el cual utiliza una red de memoria a corto plazo (LSTM).
  - Filtrado de caracteres no numéricos.

En relación a los parámetros $Threshold_{conf}$ y $Threshold_{IoU}$, y como nota al margen de la sección, se amerita una pequeña explicación en donde se detalle el porque de la elección de sus valores.

Como anteriormente se mencionó, EAST, y al igual que en cualquier otro modelo de detección, filtrará todas aquellas predicciones que: o bien posean un porcentaje de certeza menor a $Threshold_{conf}$, o sean redundantes con un área de solapamiento mayor a $Threshold_{IoU}$. Teniendo en cuenta que su objetivo es localizar regiones ocupadas por los números de los autobuses, los cuales se encuentran representados por un conjunto de diodos led; los valores de estos parámetros debieron ser modificados fuera de los rangos habituales ($[0.5 , 0.8]$ para $conf$ y $[0.4 , 0.6]$ par IoU).

El problema que trajo aparejado el hecho de que las líneas estén representados a través de estos puntos luminosos, fue la no cohesión de pixeles, que conducían al algoritmo, a la difícil tarea de identificar a tal conjunto de puntos como un carácter numérico individual. En consecuencia, múltiples detecciones parciales que cortaban la línea en diferentes partes, eran retornadas.

A fin de tratar de unificar tales regiones, es que se decidió disminuir radicalmente el valor de $Threshold_{conf}$; de esta forma, y a costa de redundancias, se dejaba la posibilidad de que en alguna de las detecciones pudiese encontrarse la totalidad del número del coche. Como forma de compensar tales redundancias, es que se decidió disminuir el valor umbral de IoU.

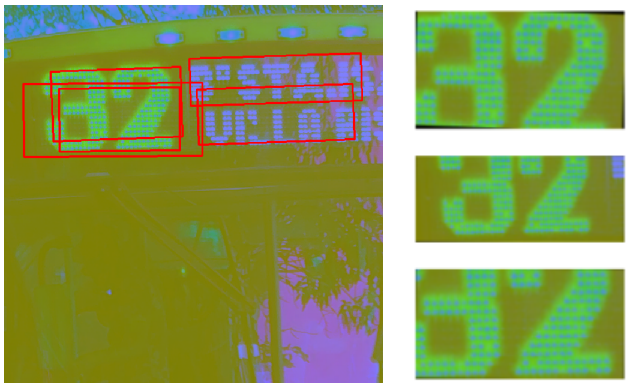
A continuación, la Figura (5.3), muestra los problemas que ocasionaba la utilización de valores estándares de $Threshold_{conf}$ y $Threshold_{conf}$, mediante una analogía en las detecciones realizadas por EAST entre: un autobús de la ciudad de Córdoba, y otro de la ciudad de Buenos Aires, el cual cuenta con un homogéneo vinilo como número de línea, que facilitaba enormemente la tarea.

Por otro lado, la Figura (5.3a), también ejemplifica lo explicado en Sección (4.2), mostrando la forma final que adopta un ROI, al producirse una detección con un determinado ángulo de inclinación.

5.4.1.1. Selección de mejores transformaciones y tuplas (newsize, pad)

Para tener una visión más global del desafío que representa seleccionar de entre 52 transformaciones, aquellas que mejores rendimientos proporcionen, se realizó, mediante el procesamiento de los dataset confeccionados, un heatmap o bitmap "ponderado", el cual brindó una primera impresión de la relación entre *Detecciones Positivas* y *Transformaciones de imágen*es.

(a) Autobús de ciudad de Buenos Aires - Detección sobre transformación **'rgb'**.



(b) Autobús de ciudad de Córdoba - Detección sobre transformación **'yuv'**.

Figura 5.3: Detección de EAST utilizando $Threshold_{conf} = 0{,}5$ y $Threshold_{IoU} = 0{,}6$.

La Figura (5.4), muestra el heatmap generado por la ejecución del sistema sobre los 100 recortes de autobuses, en relación a las 52 transformaciones de imágenes y de las posibles combinaciones de los parámetros *newsize* y *pad;* ordenado de manera decreciente en función de la cantidad de recortes, que cada transformación predijo correctamente (acertó su número de línea).

Sea $V_{(i,j)}$=n el valor de la (i,j)-ésima "celda" del heatmap, n suma una unidad por cada uno de los valores de *newsize* y *pad* que participaron de la *Detección Positiva,* que la transformación i-ésima realizó sobre el j-ésimo recorte.

Partiendo de la suposición de que la mejor de las transformaciones es aquella con el mayor número de aciertos realizados, la búsqueda del mínimo conjunto de mejores transformaciones, podría, en un principio, reducirse a la trivial tarea de seleccionar las primeras n de nuestro heatmap y en cada paso de selección, computar hasta encontrar el conjunto que mayor porcentaje de predicciones entregue sobre el total de imágenes testeadas. Si bien ésta, podría considerarse una aproximación válida y funcionar correctamente, al observar en detalle, se encontraría que muchos de los aciertos fueron efectuados por múltiples transformaciones en simultáneo y con intensidades similares, conllevando a una sobrecarga de cómputo innecesario, dedicado a la generación de tales transformaciones, el cual podría ser evitado ya que las mismas no aportan más de lo que lo hacen alguna de sus pares.

Para dar solución al problema, se codificó un Algoritmo (3), capaz de reordenar tales transformaciones en relación al nivel de robustez de detección y orden de complementariedad entre las mismas. Es decir, tomando como entrada un Heatmap ordenado (como el presentado anteriormente), y manteniendo la postura de que la mejor, es aquella que mayor número de detecciones realizó, el algoritmo reordena las restantes, de acuerdo a sus capacidades de adaptación a esta última, priorizando:

1. Porcentaje extra de detecciones que proporcione la nueva transformación.
2. Mayor número de intersecciones en lo que respecta a detecciones.

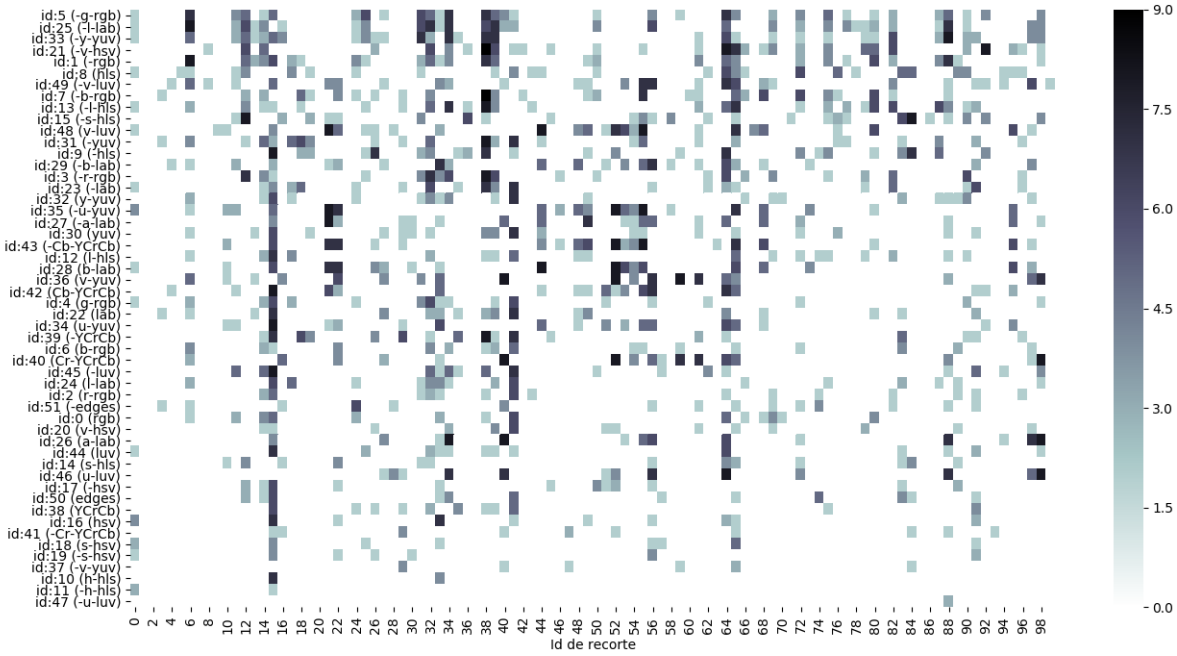


Figura 5.4: Heatmap de Detecciones positivas vs. Transformaciones de imágenes, realizado sobre datasets confeccionados partiendo del primer enfoque.

3. Mayor valor ponderado, dado por *newsize* y *pad*, en cada una de las posibles intersecciones.

Debajo, se muestran los resultados obtenidos luego de ejecutar el algoritmo, en donde se detalla cual es el porcentaje máximo que se lograría y cuales las posibilidades, si se decidiera utilizar un grupo mas reducido que el mínimo conjunto de mejores transformaciones encontrado.

- El ∼33 % podría llegar a ser detectado por → *[-g-rgb]*.

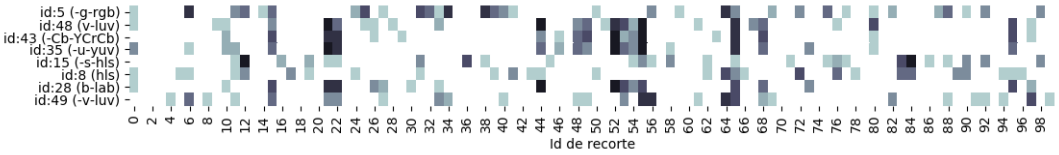
- El ∼53 % podría llegar a ser detectado por → *['-g-rgb', 'v-luv']*.

- El ∼55 % podría llegar a ser detectado por → *['-g-rgb', 'v-luv', '-Cb-YCrCb']*.

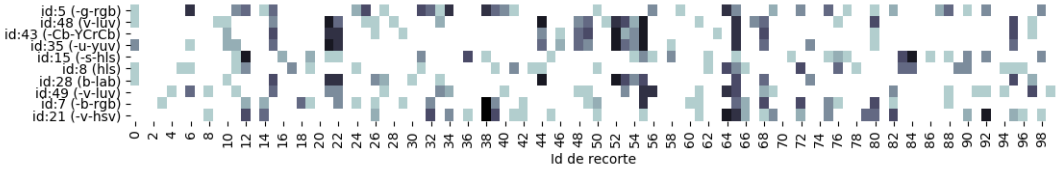
- El ∼65 % podría llegar a ser detectado por → *['-g-rgb', 'v-luv', '-Cb-YCrCb', '-u-yuv', '-s-hls']*.

- El ∼75 % podría llegar a ser detectado por → *['-g-rgb', 'v-luv', '-Cb-YCrCb', '-u-yuv', '-s-hls', 'hls']*.

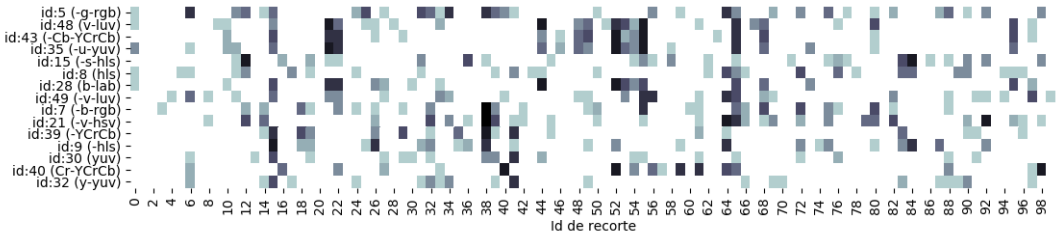
- El ∼80 % podría llegar a ser detectado por → *['-g-rgb', 'v-luv', '-Cb-YCrCb', '-u-yuv', '-s-hls', 'hls', 'b-lab', '-v-luv']*.



- El ∼85 % podría llegar a ser detectado por → *['-g-rgb', 'v-luv', '-Cb-YCrCb', '-u-yuv', '-s-hls', 'hls', 'b-lab', '-v-luv', '-b-rgb', '-v-hsv']*.



- El ∼90 % podría llegar a ser detectado por → *['-g-rgb', 'v-luv', '-Cb-YCrCb', '-u-yuv', '-s-hls', 'hls', 'b-lab', '-v-luv', '-b-rgb', '-v-hsv', '-YCrCb', '-hls', 'yuv', 'Cr-YCrCb', 'y-yuv']*.



Con una idea mas clara acerca de cual podría ser el grupo de mejores transformaciones que integren la configuración final del sistema, se abocó a la selección del resto de parámetros que mejor rendimiento proporcionaran. Si bien, simplemente se podría encontrar la tupla de parámetros (*newsize*, *pad*) que mayor porcentaje obtuvo en el anterior Top 15 de transformaciones, con un 90 % de aciertos; dado que el Algoritmo (3) fue aplicado sobre el total de datasets, es necesario asegurarse de que los resultados no estén sesgados y realizar pruebas más generalistas antes de confirmarlos.

Para ello, se decidió recomputar el mismo algoritmo de búsqueda, pero utilizando, como plataforma de partida, una modificación de la técnica «*K-fold cross-validation*» (Validación cruzada de K iteraciones).

> *La **validación cruzada** es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones. En la validación cruzada de K iteraciones, los datos de muestra se dividen en K subconjuntos. Uno de los subconjuntos*

Algorithm 3 Búsqueda de menor conjunto de mejores transformaciones

Input: $Heatmap = \{[V_0, .., V_{99}]_{t_0}, ..., [V_0, .., V_{99}]_{t_{51}}\}$

procedure GETBESTSET($Heatmap$,$percent$)
  $best \leftarrow [V_0, .., V_{99}]_{t_0}$
  $AccumTransf \leftarrow \{t_0\}$
  $TopTransf \leftarrow \{\}$
  $Weigths \leftarrow \{\}$
  $Idxs \leftarrow \{\}$
  for e_{t_i} in $Heatmap - \{best\}$ **do**
    $num_{EP}, sum_{EP} \leftarrow getExtraPositions(e_{t_i}, best)$
    $num_{SP}, sum_{SP} \leftarrow getSharedPositions(e_{t_i}, best)$
    $Weigths \leftarrow Weigths \cup \{[num_{EP}, sum_{EP}, num_{SP}, sum_{SP}]_{t_i}\}$
  end for
  $Weigths \leftarrow decrescentSorted(Weigths)$
  $Idxs \leftarrow getIndexes(Weigths)$
  $pivot \leftarrow toLogicalVector(best)$
  if $average(sum(pivot)) < percent$ **then**
    for t_i in $Idxs$ **do**
      $or \leftarrow logicalOr(pivot, toLogicalVector(Heatmap[t_i])$
      if $sum(or) > sum(pivot)$ **then**
        $AccumTransf \leftarrow accumTransf \cup \{t_i\}$
        $pivot \leftarrow or$
        if $average(sum(or)) \geq percent$ **then**
          $TopTransf \leftarrow AccumTransf$
          break
        end if
      end if
    end for
  else
    $TopTransf \leftarrow AccumTransf$
  end if
  return $TopTransf$
end procedure
procedure GETMINIMUMBESTSET($Heatmap$)
  $Set \leftarrow \{\}$
  for $percent$ in $\{100, ..., 0\}$ **do**
    $Set \leftarrow$ GETBESTSET($Heatmap, percent$)
    if $Set \neq \emptyset$ **then**
      break
    end if
  end for
  return $Set, percent$
end procedure

se utiliza como datos de prueba y el resto (K-1), como datos de entrenamiento. El proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Finalmente se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado.

En este caso, se realizó la nueva búsqueda de transformaciones a través de la implementación de 5 repeticiones de "10-fold cross-validation".

En cada validación cruzada, se tomaron $\frac{9}{10}$ grupos aleatorios de datasets y calcularon los mejores conjunto de $3, 5, 7, 10, 13\, y\, 15$ transformaciones. Una vez obtenidos, se utilizó cada uno de ellos para obtener las mejores 4 tuplas (*newsize*, *pad*) que mayores porcentajes de detecciones positivas otorgaran al otro $\frac{1}{10}$ conjunto de datasets. En cada una de las $K = 10$ iteraciones se acumularon, tanto las probabilidades medias de cada tupla, como las frecuencia de aparición de las transformación que integraba cada uno de los mejores conjuntos. Luego de 5 repeticiones de este procedimiento, los resultados se presentan en los Cuadros (5.1) y (5.2) (Referencias en Anexo (V)).

	Top3		**Top5**		**Top7**		**Top10**		**Top13**		**Top15**	
Rank	Id transf.	Frec.	Id transf.	Frec.	Id transf.	Frec.	Id transf.	Frec.	Id transf.	Frec.	Id transf.	Frec.
1	48	48	48	50	48	50	15	50	8	50	8	50
2	43	33	35	50	43	49	48	50	15	50	15	50
3	5	27	43	40	35	42	43	49	48	50	48	50
4	-	-	15	30	15	38	8	47	49	50	49	50
5	-	-	5	24	8	33	7	44	7	49	7	49
6	-	-	-	-	5	32	49	44	43	48	43	49
7	-	-	-	-	49	24	35	43	35	42	9	48
8	-	-	-	-	-	-	28	38	9	41	35	42
9	-	-	-	-	-	-	31	29	28	37	40	38
10	-	-	-	-	-	-	5	28	31	34	28	37
11	-	-	-	-	-	-	-	-	40	30	5	26
12	-	-	-	-	-	-	-	-	5	25	31	26
13	-	-	-	-	-	-	-	-	21	23	32	25
14	-	-	-	-	-	-	-	-	-	-	21	22
15	-	-	-	-	-	-	-	-	-	-	30	19

Cuadro 5.1: Mejores seis conjuntos de transformaciones. Primer enfoque.

5.4.1.2. Testeo y generación de nuevas representaciones

En busca de nuevas heurísticas que mejoren el rendimiento de las detecciones, y en un intento por reducir la cantidad de transformaciones necesarias para obtener altos porcentajes de detecciones positivas, se testearon seis nuevos espacios de colores. Éstos, generados a partir de cada una de las posibles permutaciones entre los canales pertenecientes al conjunto **Top 3**, desarrollado en la sección anterior.

Nuevamente, se confeccionaron y procesaron 100 nuevos datasets, uno por cada recorte de autobús de la muestra, y se calcularon las probabilidades de las cuatro mejores tuplas (newsize, pad), y los porcentajes de detecciones positivas realizadas por estos nuevo seis espacios de colores. En este caso, cada dataset, se conformó por los resultados arrojados por el sistema, para cada combinación entre: las 6 nuevas transformaciones de imágenes y todos los valores de parámetros correspondientes a las variables *newsize* y *pad*.

El Cuadro (5.3), muestra los resultados luego de ejecutar tal heurística.

5.4.1.3. Precisión de detecciones: EAST

En esta parte del trabajo, se determinó la precisión del módulo de detección de texto (EAST), en relación al reconocimiento de las áreas donde se sitúan los números de líneas de los autobuses. La misma, fue llevada a cabo mediante el cálculo de índices IoU entre: los recuadros delimitadores de las detecciones producidas y los ROI's de los números anotados manualmente en cada uno de los 100 recortes de autobuses de nuestra muestra, como se detallo en la Sección (5.1).

Todas las pruebas, fueron realizadas utilizando las dos mejores tuplas (*newsize, pad*) de cada uno de los seis conjuntos de mejores transformaciones que resultaron de los análisis anteriores. El objetivo principal fue verificar la existencia de una relación lineal entre: la *'buena precisión'* y el porcentaje de *'detecciones positivas'*.

Los resultados obtenidos en el Cuadro (5.4), permitirán conocer en mayor profundidad, que nivel de responsabilidad tiene EAST, en el los resultados finales del reconocimiento numérico llevado a cabo por el motor OCR-Tesseract.

5.4.2. Segundo enfoque

En esta segunda aproximación, y al igual que en el primer enfoque, se trato de buscar el mejor conjunto de parámetros que ajuste al sistema.

	Top3				**Top5**			
Rank	Newsize	Pad	Frec.	Prob.	Newsize	Pad	Frec.	Prob.
1	96	0	23	20 %	96	0	21	23,33 %
2	128	10	22	28,18 %	128	15	18	29,44 %
3	128	5	18	22,78 %	128	0	17	28,82 %
4	128	0	18	18,89 %	128	10	17	28,82 %
	Top7				**Top10**			
Rank	Newsize	Pad	Frec.	Prob.	Newsize	Pad	Frec.	Prob.
1	128	0	23	30,43 %	128	0	25	42,8 %
2	96	0	22	29,55 %	160	0	20	43,0 %
3	128	10	19	32,11 %	160	5	18	41,11 %
4	128	5	18	31,11 %	128	5	18	38,33 %
	Top13				**Top15**			
Rank	Newsize	Pad	Frec.	Prob.	Newsize	Pad	Frec.	Prob.
1	128	0	27	42,96 %	128	0	23	61,9 %
2	96	5	19	42,63 %	160	5	21	45,78 %
3	128	5	18	43,33 %	96	5	18	40,56 %
4	160	5	17	45,88 %	128	5	17	44,12 %

Cuadro 5.2: Mejores combinaciones de tuplas (newsize, pad), para cada conjunto de mejores transformaciones. Primer enfoque.

A diferencia de este último, el análisis y la creación de los datasets, fue realizado focalizando la atención sobre un área específica de los recortes originales de aspecto cuadrado. Algunos de los ROI's utilizados pueden encontrarse en Anexo (V).

Sus límites, fueron seleccionados en base al análisis estadístico de las localizaciones promedio de los números de línea, sobre el total de recortes de la muestra, haciendo uso de las anotaciones manuales realizadas.

Las configuraciones de los módulos principales fueron las siguientes:

- Recortes de autobúses: Heurística de atención detallada en Sección (5.4.2.1).
- Redimensión
  - *newsize*: 64×64, 96×96, 128×128, 160×160 y 320×320 pixeles.
- Módulo detector de texto (EAST)
  - $Threshold_{conf}$: 0,001.
  - $Threshold_{IoU}$: 0,1.
- Padding
  - *pad*: 0 %, 5 %, 10 % y 15 %.
- Módulo de reconocimiento de texto (OCR-Tesseract)
  - *lenguaje*: Inglés.
  - *psm* (Modo de Segmentación de Página): Modo 7, el cual trata a la imagen como una simple línea de texto.
  - *oem* (Modo del Motor OCR): Modo 1, el cual utiliza una red de memoria a corto plazo (LSTM).
  - Filtrado de caracteres no numéricos.

5.4.2.1. Heurística de atención

Con el fin de obtener mejores rendimientos en las detecciones positivas, se planteó reducir el área de los recortes analizados, focalizando la búsqueda sobre una región de menor dimensión que

Rank	**'v-luv' , '-Cb-YCrCb' , '-g-rgb'** = 17,0 %	**'v-luv' , '-g-rgb' , '-Cb-YCrCb'** = 25,0 %
1	P(newsize_160 ⋂ pad_10) = 9,0 %	P(newsize_160 ⋂ pad_0) = 7,0 %
2	P(newsize_320 ⋂ pad_10) = 6,0 %	P(newsize_160 ⋂ pad_5) = 7,0 %
3	P(newsize_160 ⋂ pad_5) = 5,0 %	P(newsize_320 ⋂ pad_15) = 6,0 %
4	P(newsize_160 ⋂ pad_15) = 5,0 %	P(newsize_128 ⋂ pad_0) = 5,0 %
Rank	**'-Cb-YCrCb' , 'v-luv', '-g-rgb'** = 29,0 %	**'-Cb-YCrCb' , '-g-rgb', 'v-luv'** = 27,0 %
1	P(newsize_320 ⋂ pad_5) = 9,0 %	P(newsize_160 ⋂ pad_5) = 8,0 %
2	P(newsize_320 ⋂ pad_10) = 8,0 %	P(newsize_160 ⋂ pad_0) = 7,0 %
3	P(newsize_320 ⋂ pad_15) = 8,0 %	P(newsize_160 ⋂ pad_10) = 7,0 %
4	P(newsize_160 ⋂ pad_15) = 6,0 %	P(newsize_128 ⋂ pad_0) = 6,0 %
Rank	**'-g-rgb' , 'v-luv' , '-Cb-YCrCb'** = 23,0 %	**'-g-rgb' , '-Cb-YCrCb', 'v-luv'** = 29,0 %
1	P(newsize_320 ⋂ pad_10) = 8,0 %	P(newsize_320 ⋂ pad_10) = 10,0 %
2	P(newsize_320 ⋂ pad_15) = 6,0 %	P(newsize_320 ⋂ pad_5) = 9,0 %
3	P(newsize_160 ⋂ pad_5) = 5,0 %	P(newsize_160 ⋂ pad_10) = 8,0 %
4	P(newsize_320 ⋂ pad_0) = 5,0 %	P(newsize_320 ⋂ pad_0) = 8,0 %

Cuadro 5.3: Porcentaje de detecciones positivas arrojados por cada nueva representación generada. Primer enfoque.

$\geq IoU$	**Top3**		**Top5**		**Top7**		**Top10**		**Top13**		**Top15**	
	(96,0)	**(128,10)**	**(96,0)**	**(128,15)**	**(128,0)**	**(96,0)**	**(128,0)**	**(160,0)**	**(128,0)**	**(96,5)**	**(128,0)**	**(160,5)**
.5	15 %	13 %	22 %	6 %	57 %	29 %	59 %	73 %	60 %	14 %	62 %	77 %
.55	11 %	8 %	16 %	4 %	50 %	23 %	52 %	70 %	52 %	11 %	56 %	73 %
.6	5 %	3 %	7 %	1 %	43 %	11 %	45 %	64 %	45 %	6 %	47 %	63 %
.65	3 %	1 %	4 %	-	39 %	7 %	41 %	55 %	42 %	4 %	43 %	58 %
.7	3 %	-	4 %	-	24 %	5 %	27 %	41 %	29 %	-	31 %	45 %
.75	1 %	-	1 %	-	9 %	2 %	11 %	27 %	14 %	-	15 %	32 %
.8	-	-	-	-	4 %	-	5 %	21 %	7 %	-	8 %	14 %
.85	-	-	-	-	2 %	-	2 %	12 %	2 %	-	2 %	5 %
.9	-	-	-	-	-	-	-	5 %	-	-	-	1 %
.95	-	-	-	-	-	-	-	-	-	-	-	1 %
1	-	-	-	-	-	-	-	-	-	-	-	-

Cuadro 5.4: Precisión de las detecciones realizadas por modelo EAST. Primer enfoque.

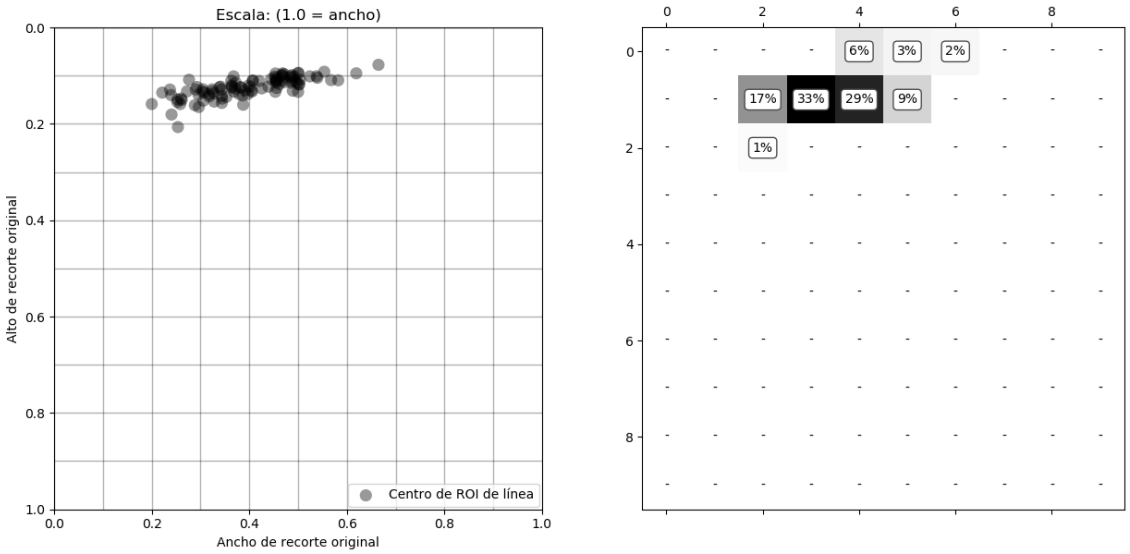
contuviese el número de línea deseado. Esta estrategia, en el mejor de los casos, permitiría tanto al algoritmo de detección de texto como al módulo de reconocimiento, centrar su atención sólo en ciertas regiones de interés, evitando que los modelos procesen lugares, que a priori, se sabe que no serán contenedores de números de línea.

Para ello, se elaboró un análisis en base a las anotaciones manuales de los rectángulos delimitadores de las líneas de los autobuses, y tomando en cuenta el mapa de distribuciones espaciales producido por los límites de cada uno, se intento identificar y calcular, la dimensión y la posición del cuadrante más representativo que alojara el mayor porcentaje de números en su región.

Inicialmente, y para entrar en contacto con nuestro espacio muestral, se confeccionó el mapa de distribuciones en relación a las coordenadas que identifican el punto central de cada recuadro delimitador. La Figura (5.5), muestra tal distribución y los porcentajes captados, ubicando a los cuadrantes [4,5,6,12,13,14,15 y 22] como los únicos afectados (si se consideran 100 en total, y son numeramos de izquierda a derecha y de arriba hacia abajo, comenzando desde 0).

Aunque esto podría sugerir un área de actuación bastante mas pequeña que la utilizada en el primer enfoque, no hay que perder de vista que: primero, el área seleccionada, y al igual que en el primer enfoque, será de dimensiones cuadradas por lo explicado en Sección (5.2); y segundo, estas distribuciones pertenecen al centro de cada ROI, por lo tanto, si se selecciona el área cuyo ancho toca los límites de los cuadrantes antes mencionados, los números ubicados hacia los extremos quedarían cortados a la mitad.

Dado lo anterior, se elaboró un segundo mapa de distribuciones, pero en esta instancia, en



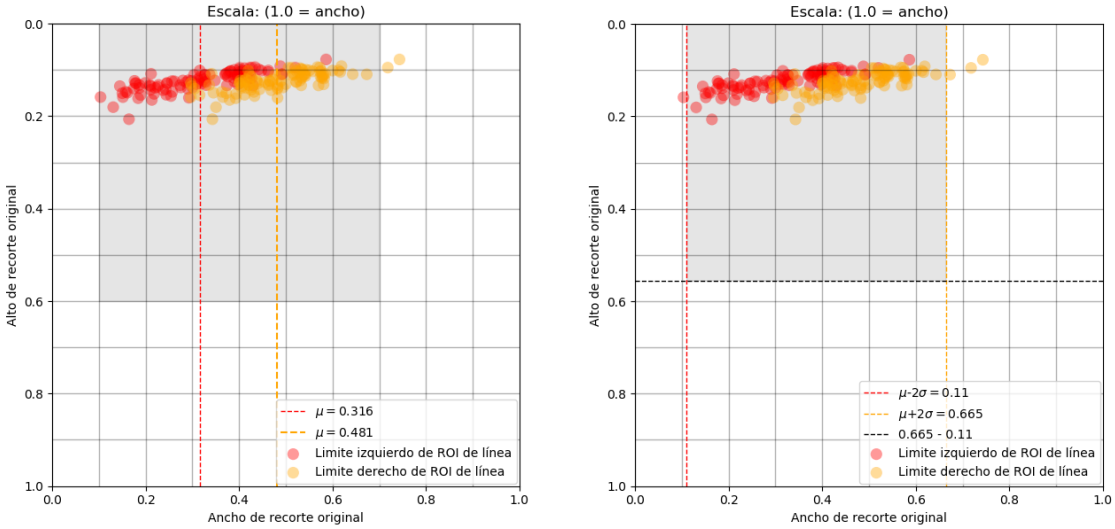
(a) Distribución de ubicaciones.

(b) Porcentajes de acierto por cuadrante.

Figura 5.5: Distribuciones de números de línea en relación al punto central de sus ROI's.

relación a las posiciones de los límites de inicio y fin de cada recuadro delimitador. Es decir, se representaron las coordenadas $(x_{mínimo}, \frac{height}{2})$ y $(x_{máximo}, \frac{height}{2})$ de cada ROI.

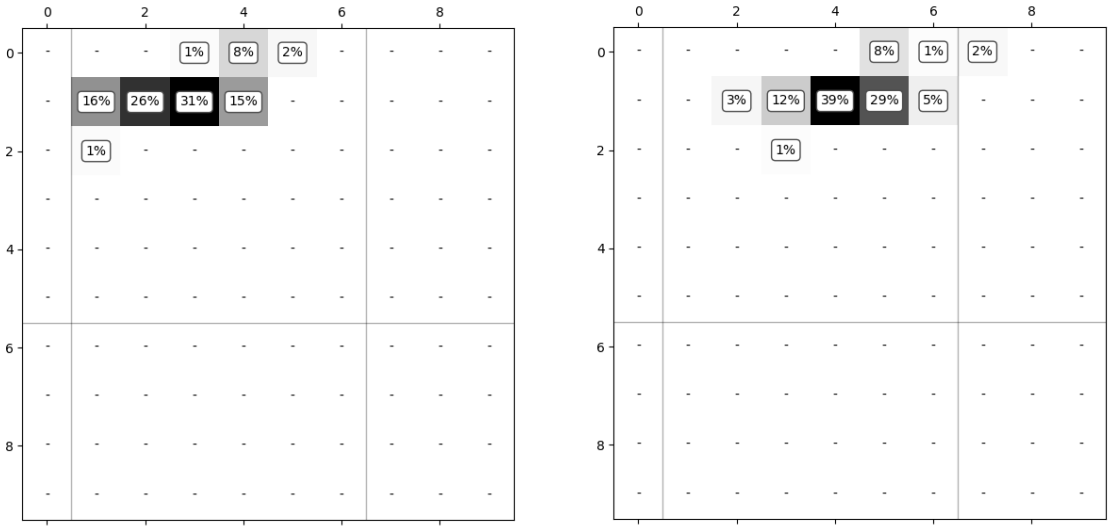
La Figura (5.6), en este caso, llevando la dimensionalidad al aspecto cuadrado, muestra que, de seleccionarse la región sombreada delimitada por $\mu - 2\sigma$ (calculado en base a $x_{mínimos}$) y $\mu + 2\sigma$ (calculado en base a $x_{máximos}$), la cual fácilmente podría representarse por los cuadrantes [1-6,11-16,21-26,31-36,41-46 y 51-56]; la mayoría de las captaciones, tanto de los límites izquierdos como de los derechos, serían percibidos en aproximadamente un porcentaje de 99 % y 97 % respectivamente, como se observa en la Figura (5.7). Esto significa, que casi la totalidad de los números de líneas, caen dentro de este área seleccionada.



(a) Cuadrantes más representativos.

(b) Región mas representativa en función de estimadores estadísticos.

Figura 5.6: Distribuciones de número de líneas en relación a limites Izq. y Der. de sus ROI's.



(a) Límites izquierdos de ROI's.

(b) Límites derechos de ROI's.

Figura 5.7: Cuadrantes y porcentajes de acierto.

Sean $w \times h$ la dimensión del recorte original, con $(w = h)$, el nuevo ROI, será generado por las coordenadas de los vértices superior izquierdo $[w * (0.316 - 2 * 0.103), 0]$ e inferior derecho $[w * (0.481 + 2 * 0.092), x_{max} - x_{min}]$. A continuación, en la Figura (5.8), se muestra la aplicación de la heurística sobre tres recortes originales de autobuses.

Figura 5.8: Aplicación de heurística de atención sobre tres recortes aleatorios de la muestra.

5.4.2.2. Selección de mejores transformaciones y tuplas (newsize, pad)

Al igual que en el primer enfoque, la Figura (5.9), muestra el heatmap confeccionado para esta segunda aproximación, que describe los resultados de la relación entre Detecciones Positivas y Transformaciones de imágenes.

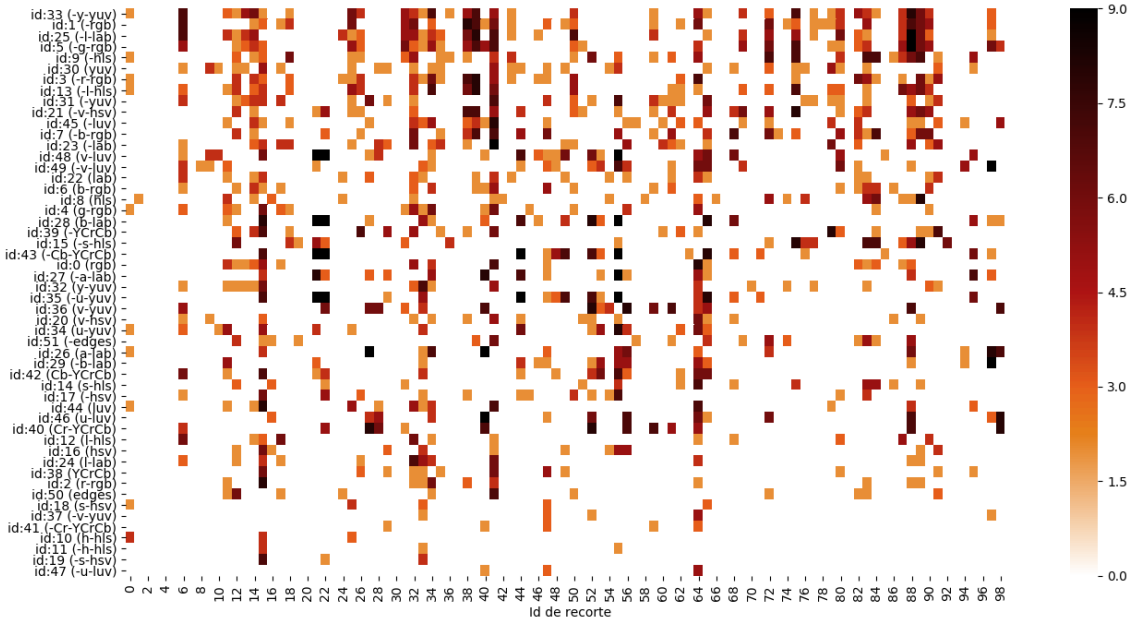
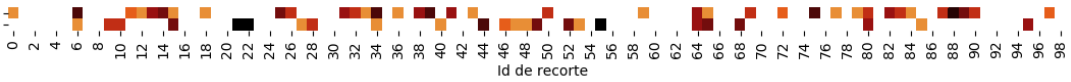


Figura 5.9: Heatmap de Detecciones positivas vs. Transformaciones de imágenes, realizado sobre datasets confeccionados partiendo del segundo enfoque.

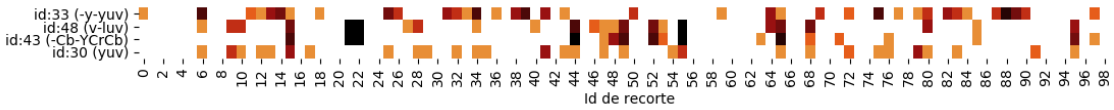
A continuación, se enumeran las salidas de ejecutar el Algoritmo (3), de "búsqueda del menor conjunto de mejores transformaciones de imágenes", sobre el total de datasets. Por otro lado, los Cuadros (5.5) y (5.6), detallan los resultados de su ejecución mediante la aplicación de 5 repeticiones de la técnica 10-fold cross-validation, en búsqueda de los mejores conjuntos de 3, 5, 7, 10, 13 y 15 transformaciones y sus mejores tuplas de parámetros (*newsize*, *pad*).

- El ∼37 % podría llegar a ser detectado por → *['-y-yuv']*.

- El ∼55 % podría llegar a ser detectado por → *['-y-yuv', 'v-luv']*.

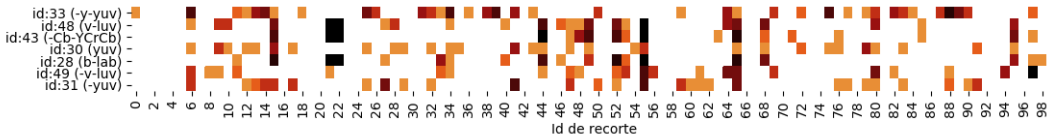


- El ∼60 % podría llegar a ser detectado por → *['-y-yuv', 'v-luv', '-Cb-YCrCb', 'yuv']*.

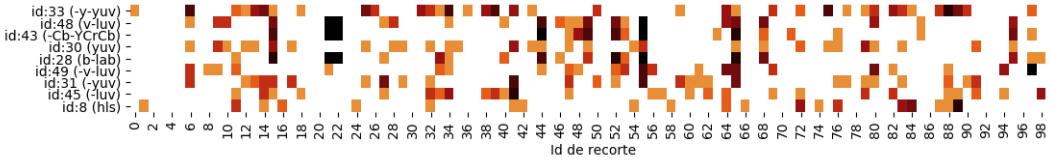


- El ∼65 % podría llegar a ser detectado por → *['-y-yuv', 'v-luv', '-Cb-YCrCb', 'yuv', 'b-lab', '-v-luv']*.

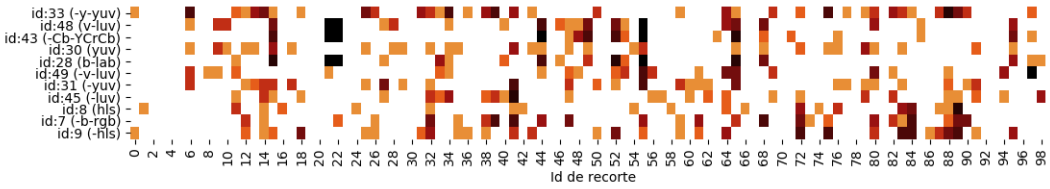
- El ∼70 % podría llegar a ser detectado por → *['-y-yuv', 'v-luv', '-Cb-YCrCb', 'yuv', 'b-lab', '-v-luv', '-yuv']*.



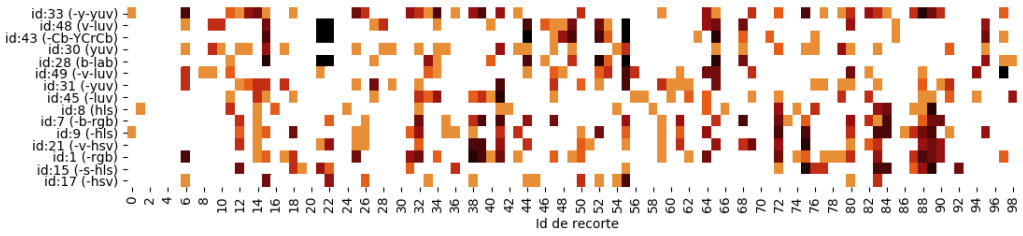
- El ∼75 % podría llegar a ser detectado por → *['-y-yuv', 'v-luv', '-Cb-YCrCb', 'yuv', 'b-lab', '-v-luv', '-yuv', '-luv', 'hls']*.



- El ∼80 % podría llegar a ser detectado por → *['-y-yuv', 'v-luv', '-Cb-YCrCb', 'yuv', 'b-lab', '-v-luv', '-yuv', '-luv', 'hls', '-b-rgb', '-hls']*.



- El ∼85 % podría llegar a ser detectado por → *['-y-yuv', 'v-luv', '-Cb-YCrCb', 'yuv', 'b-lab', '-v-luv', '-yuv', '-luv', 'hls', '-b-rgb', '-hls', '-v-hsv', '-rgb', '-s-hls', '-hsv']*.



	Top3		Top5		Top7		Top10		Top13		Top15	
Rank	Id transf.	Frec.	Id transf.	Frec.	Id transf.	Frec.	Id transf.	Frec.	Id transf.	Frec.	Id transf.	Frec.
1	48	50	33	50	30	50	30	50	30	50	8	50
2	33	48	48	50	33	50	43	50	33	50	30	50
3	43	34	43	49	43	50	48	50	43	50	43	50
4	-	-	30	47	48	50	33	49	48	50	48	50
5	-	-	28	39	28	44	49	49	49	50	49	50
6	-	-	-	-	49	40	28	45	7	45	33	49
7	-	-	-	-	31	26	45	43	45	45	7	45
8	-	-	-	-	-	-	31	36	8	44	28	45
9	-	-	-	-	-	-	8	32	28	44	45	45
10	-	-	-	-	-	-	7	29	31	38	15	37
11	-	-	-	-	-	-	-	-	9	30	31	35
12	-	-	-	-	-	-	-	-	21	27	1	34
13	-	-	-	-	-	-	-	-	1	21	17	28
14	-	-	-	-	-	-	-	-	-	-	9	27
15	-	-	-	-	-	-	-	-	-	-	21	27

Cuadro 5.5: Mejores seis conjuntos de transformaciones. Segundo enfoque.

	Top3				Top5			
Rank	Newsize	Pad	Frec.	Prob.	Newsize	Pad	Frec.	Prob.
1	64	0	24	22,5 %	64	0	24	24,58 %
2	96	10	19	22,63 %	64	5	21	24,29 %
3	64	5	18	21,67 %	96	5	18	28,89 %
4	64	10	16	18,12 %	96	0	18	28,33 %
	Top7				**Top10**			
Rank	Newsize	Pad	Frec.	Prob.	Newsize	Pad	Frec.	Prob.
1	96	15	25	35,2 %	96	5	24	38,33 %
2	96	5	22	34,55 %	96	15	23	40,87 %
3	64	0	20	29,5 %	96	0	20	38,5 %
4	96	10	18	32,78 %	96	10	10	41,58 %
	Top13				**Top15**			
Rank	Newsize	Pad	Frec.	Prob.	Newsize	Pad	Frec.	Prob.
1	96	10	22	41,36 %	96	5	24	45,83 %
2	96	15	21	49,52 %	96	10	18	46,67 %
3	96	0	20	42,5 %	96	0	18	46,11 %
4	64	0	18	41,67 %	96	15	17	47,06 %

Cuadro 5.6: Mejores combinaciones de tuplas (newsize, pad), para cada conjunto de mejores transformaciones. Segundo enfoque.

5.4.2.3. Testeo y generación de nuevas representaciones

Del mismo modo que en la Sección (5.4.1.2), y en un intento por reducir la cantidad de transformaciones necesarias para obtener altos porcentajes de detecciones positivas, se crearon seis nuevas representaciones de imágenes basadas en las posibles permutaciones de los canales *'v-luv', '-Cb-YCrCb' y '-g-rgb',* integrantes del **Top 3**, calculado en la sección anterior. El Cuadro (5.7), detalla las nuevas probabilidades entregadas.

Rank	**'v-luv' , '-y-yuv' , '-Cb-YCrCb'** = 36,0 %	**'v-luv' , '-Cb-YCrCb' , '-y-yuv'** = 30,0 %
1	P(newsize_160 ⋂ pad_15) = 8,0 %	P(newsize_320 ⋂ pad_15) = 10,0 %
2	P(newsize_96 ⋂ pad_15) = 7,0 %	P(newsize_64 ⋂ pad_10) = 8,0 %
3	P(newsize_64 ⋂ pad_15) = 6,0 %	P(newsize_128 ⋂ pad_5) = 7,0 %
4	P(newsize_96 ⋂ pad_0) = 6,0 %	P(newsize_128 ⋂ pad_15) = 7,0 %
Rank	**'-y-yuv' , 'v-luv' , '-Cb-YCrCb'** = 30,0 %	**'-y-yuv' , '-Cb-YCrCb' , 'v-luv'** = 31,0 %
1	P(newsize_128 ⋂ pad_15) = 13,0 %	P(newsize_96 ⋂ pad_0) = 9,0 %
2	P(newsize_320 ⋂ pad_15) = 10,0 %	P(newsize_128 ⋂ pad_5) = 9,0 %
3	P(newsize_128 ⋂ pad_5) = 9,0 %	P(newsize_96 ⋂ pad_5) = 8,0 %
4	P(newsize_160 ⋂ pad_10) = 9,0 %	P(newsize_128 ⋂ pad_10) = 8,0 %
Rank	**'-Cb-YCrCb' , 'v-luv' , '-y-yuv'** = 27,0 %	**'-Cb-YCrCb' , '-y-yuv' , 'v-luv'** = 36,0 %
1	P(newsize_320 ⋂ pad_15) = 9,0 %	P(newsize_128 ⋂ pad_10) = 10,0 %
2	P(newsize_96 ⋂ pad_15) = 7,0 %	P(newsize_96 ⋂ pad_15) = 8,0 %
3	P(newsize_160 ⋂ pad_15) = 7,0 %	P(newsize_64 ⋂ pad_0) = 7,0 %
4	P(newsize_64 ⋂ pad_0) = 6,0 %	P(newsize_96 ⋂ pad_10) = 7,0 %

Cuadro 5.7: Porcentaje de detecciones positivas arrojados por cada nueva representación generada. Segundo enfoque.

5.4.2.4. Precisión de detecciones: EAST

Al igual que como se lo hizo en el primer enfoque, en el Cuadro (5.8), se detalla el análisis de las precisiones en las detecciones realizadas por el módulo detector de texto EAST, en base a las anotaciones manuales de las regiones de los números de líneas, realizadas sobre los recortes de autobuses, luego de atravesar el algoritmo de atención detallado en la Sección (5.4.2.1).

$\geq IoU$	**Top3**		**Top5**		**Top7**		**Top10**		**Top13**		**Top15**	
	(64,0)	**(96,10)**	**(64,0)**	**(64,5)**	**(96,15)**	**(96,5)**	**(96,5)**	**(96,15)**	**(96,10)**	**(96,15)**	**(96,5)**	**(96,10)**
.5	36 %	31 %	38 %	30 %	40 %	58 %	52 %	72 %	66 %	54 %	77 %	69 %
.55	34 %	27 %	36 %	26 %	23 %	53 %	33 %	66 %	61 %	35 %	70 %	63 %
.6	29 %	21 %	30 %	15 %	10 %	50 %	19 %	63 %	54 %	19 %	66 %	55 %
.65	22 %	13 %	23 %	9 %	6 %	41 %	10 %	56 %	40 %	10 %	59 %	42 %
.7	14 %	5 %	18 %	6 %	2 %	35 %	5 %	42 %	16 %	5 %	49 %	19 %
.75	6 %	2 %	7 %	-	1 %	25 %	3 %	28 %	7 %	3 %	33 %	7 %
.8	4 %	2 %	4 %	-	-	15 %	1 %	17 %	3 %	2 %	19 %	3 %
.85	2 %	-	2 %	-	-	3 %	-	4 %	2 %	-	5 %	2 %
.9	-	-	-	-	-	-	-	-	-	-	-	-
.95	-	-	-	-	-	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	-	-	-

Cuadro 5.8: Precisión de las detecciones realizadas por modelo EAST. Segundo enfoque.

5.5. Dimensión original vs. Detecciones Positivas

En esta sección, se analizaron las dimensiones originales de cada uno de los recortes de los autobuses, tanto las del primer, como las del segundo enfoque; y se midieron las cantidades de detecciones [positivas | negativas] que arrojaron, en relación a todas las combinaciones de valores de parámetros que [predijeron | no predijeron] el número correcto del autobús. Ésto, con similares objetivos a los de las Secciones (5.4.1.2) y (5.4.1.1), de detectar patrones distinguibles y elaborar heurísticas para mejorar los porcentajes de detecciones positivas. Cabe aclarar que, cuando se hace referencia a "dimensiones originales", se esta indicando, las dimensiones de los recuadros delimitadores anotados manualmente de los autobuses, luego de modificar sus límites a aspecto cuadrado, para el primer enfoque, y de este, posterior a ser procesados por el algoritmo de atención en la Sección (5.4.2.1), para el segundo enfoque.

Tales análisis fueron realizados en base a los datasets confeccionados en cada una de las dos aproximaciones, y resumidos mediante el cálculo de los estimadores: media, mediana, moda y desviación estándar, que muestran el comportamiento de la dimensión original de los recortes en relación a las detecciones positivas y a las negativas.

Las Figura (5.10) y Figura(5.11), describen los resultados para el primer y el segundo enfoque respectivamente.

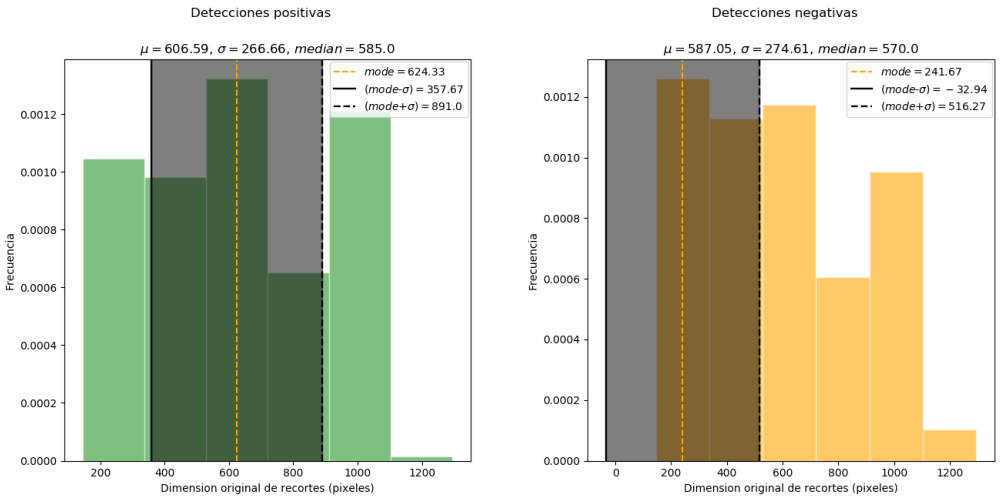


Figura 5.10: Primer enfoque: Relación entre dimensiones originales y detecciones realizadas.

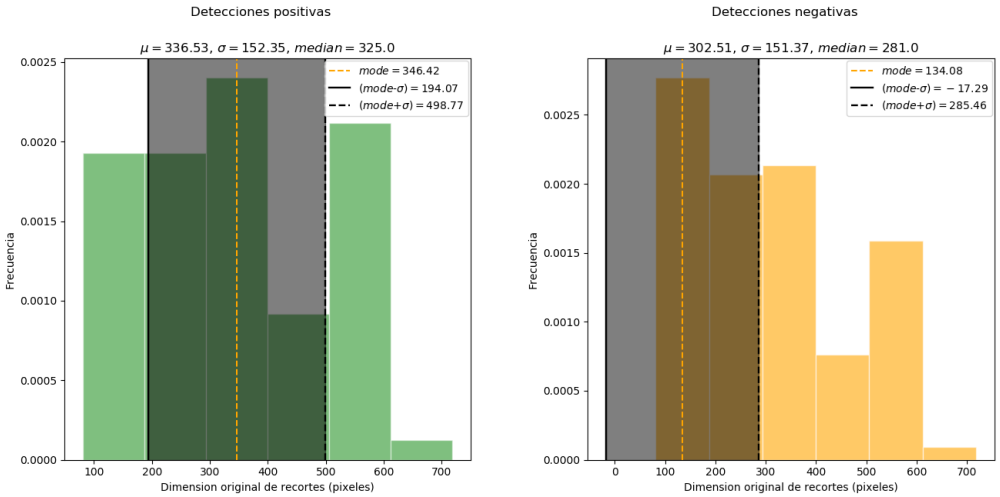


Figura 5.11: Segundo enfoque: Relación entre dimensiones originales y detecciones realizadas.

5.5.1. Comprobación de heurística

Para comprobar la veracidad y las potenciales mejoras, se procedieron a redimensionar los recortes por fuera del rango $mode \pm \frac{\sigma}{2}$, a un valor de $mode$ (con $mode$ y σ pertenecientes a cada uno de los estimadores arrojados por el gráfico de detecciones positivas); luego, se realizaron nuevamente las pruebas de validación cruzada de las Secciones (5.4.1.1) y (5.4.2.2), para verificar si hubo cambios favorables en tales resultados.

Tales pruebas se resumen en el Cuadro (5.9).

Primer enfoque					**Segundo enfoque**				
Newsize	**Pad**	**Frec.**	**Detecciones**	**Top**	**Newsize**	**Pad**	**Frec.**	**Detecciones**	**Top**
128	0	16	41,25 %	15	96	5	11	46,36 %	15
160	5	9	53,33 %	15	96	15	11	45,45 %	13

Cuadro 5.9: Mejores resultados obtenidos para cada enfoque, luego de aplicar la heurística de redimensión correspondiente.

Conclusión

En relación al primer enfoque, aunque la tupla (160, 5, Top 15) haya tenido muy buen rendimiento con un 53,33 %, cuenta con una frecuencia de detecciones dos veces menor que en la propuesta original.

Al igual que lo analizado en el primer enfoque, en el segundo, los porcentajes se mantienen semejantes, pero también se observa una disminución marcada en las frecuencias de detecciones en comparación con las originales.

En base a los resultados, se concluye que no hay motivos suficientes para modificar los mejores parámetros obtenidos antes de esta nueva propuesta, dado a que no propone mejoras significativas; por tal motivo, la aplicación de la misma es descartada y no se encuentra reflejada en los posteriores análisis realizados en las Secciones siguientes.

5.6. Relación de aspecto de autobuses: Análisis

Debido a que el presente trabajo fue orientado y pensado para que el reconocimiento sea realizado en tiempo de ejecución, el sistema, debe tener en cierta forma, una estrategia para poder anticipar y controlar cuales podrían llegar a ser las detecciones de autobuses que posibilitaran mayores chances de entregar una identificación correcta en el número de su línea.

Dado a que el 100 % de los mismos posee su número sobre el parabrisas; una de las posibilidades mas tangibles de poder tomar tal decisión, es mediante el reconocimiento de la orientación en la que esta posicionado el autobús, al momento de capturar la imagen.

Basándose en las dimensiones frontales de los coches, se tomó como criterio de selección, filtrar toda detección realizada por el *módulo de detección de autobuses*, que posea una relación de aspecto diferente a las que se identifiquen con autobuses viniendo de 'frente' o 'levemente lateralizados'.

En busca de tales relaciones, y tomando de referencia los 100 autobuses de nuestra muestra poblacional seleccionada, se evaluaron las relaciones de aspectos de sus recuadros delimitadores y se calcularon los estimadores: media, mediana, moda y desviación estándar para tres categorías diferentes, según la distancia a la que se encontraba el autobús al momento de la fotografía.

La Figura (5.12), muestra los resultados obtenidos de autobuses a corta y media distancia. Es decir, tanto para coches próximos a llegar a la parada (o en espera del arribo de pasajeros), como para aquellos que se encontraban de 10 a 15 mts. de distancia o tomados con un ángulo moderadamente lateral.

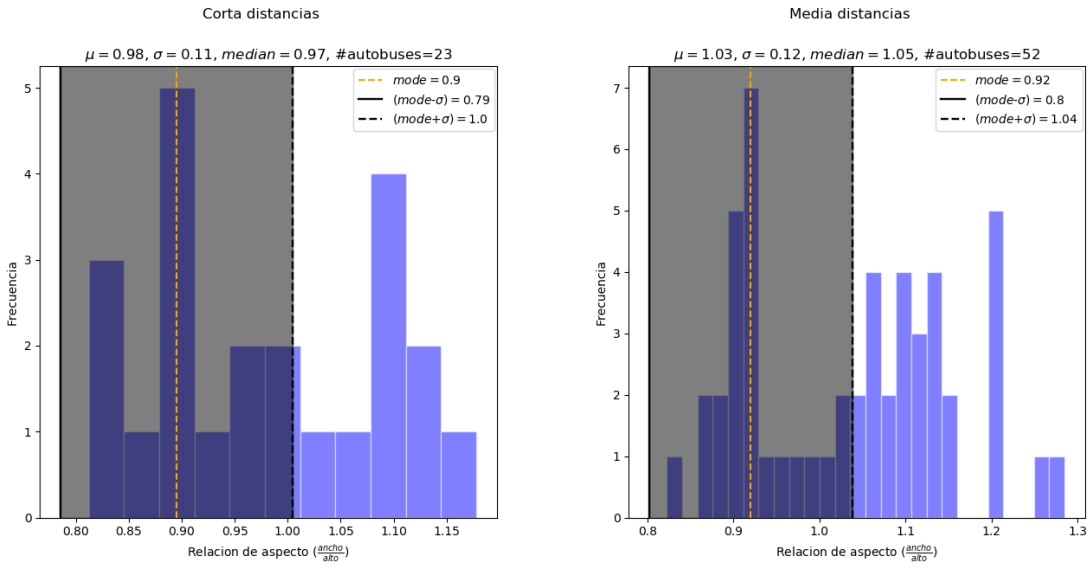


Figura 5.12: Frecuencia de relaciones de aspecto para autobuses a corta y media distancia.

En la Figura (5.13), en cambio, las imágenes fueron categorizadas como de larga distancia, dado que la captura fue realizada luego de una larga cola de personas apostadas a la espera del autobús. Como extra, se muestra un gráfico con los análisis de los aspectos de los recuadros delimitadores, que resultaron de todas las anotaciones manuales, lo cual conduce a un panorama general de las posibles variaciones.

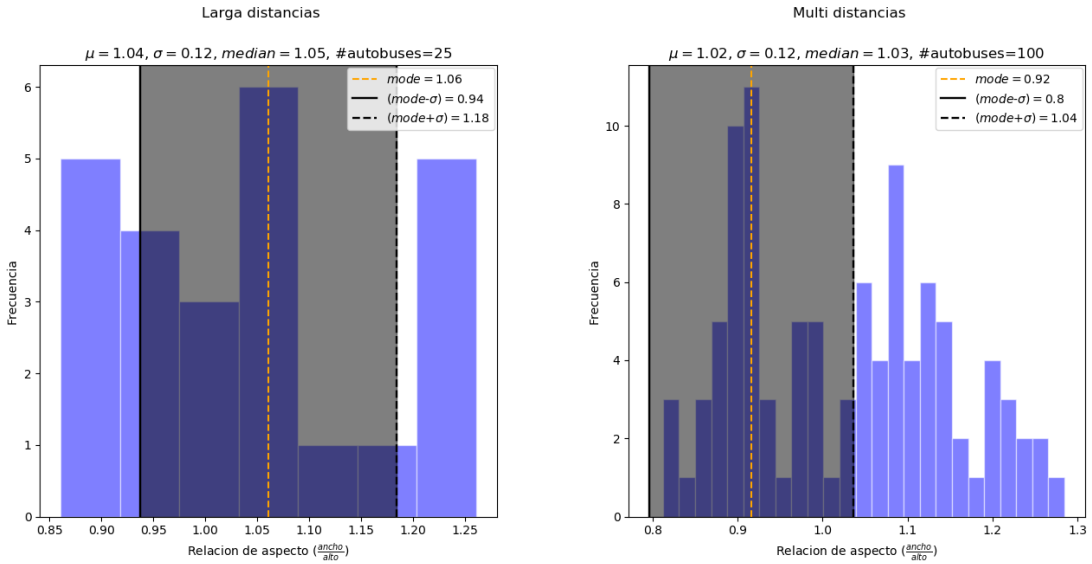


Figura 5.13: Frecuencia de relaciones de aspecto para autobuses a larga distancia.

En base a los resultados obtenidos se concluyó que: sea $w \times h$ el ancho y alto respectivos de una determinada detección, un autobús detectado sería considerado buen candidato, si su aspecto de radio se encontrase en el rango de valores $(radio - delta) \leq \frac{w}{h} \leq (radio + delta)$, con $radio = 0.92$ y $delta = 0.24$; resultado de tomar los valores de los estimadores $mode$, correspondiente a los análisis de autobuses de media distancia, y al doble de su desviación estándar, para hacer menos restrictivo el filtrado.

5.7. Conclusiones

A lo largo de esta sección, se desarrollaron diferente experimentos para tratar de encontrar los valores de los parámetros que mejor ajustaran la etapa de detección y reconocimiento de números de líneas. Durante el proceso, se partió de dos diferentes enfoques con el objetivo de rescatar de cada uno, el mejor conjunto de transformaciones de imágenes y las mejores tuplas (*newsize, pad*) que generaran mayores porcentajes de detecciones positivas. En el Cuadro (5.10), se resume lo mejor de cada uno.

Primer enfoque				**Segundo enfoque**			
Newsize	**Pad**	**Detecciones**	**Transformaciones**	**Newsize**	**Pad**	**Detecciones**	**Trasformaciones**
160	5	45,78 %	Top15	96	5	45,83 %	Top15
128	0	61,9 %	Top15	96	15	49,52 %	Top13

Cuadro 5.10: Mejores combinaciones de parámetros encontrados para cada uno de los enfoques estudiados.

En base a estos resultados, y con la intención de aumentar tales porcentajes, se desencadenaron una sucesión de nuevas propuestas y análisis, para intentar conocer con mayor profundidad el comportamiento detrás de los mismos.

Las pruebas de precisión realizadas al módulo de EAST, mostraron que las mejores tuplas de ambos enfoques, permitieron alrededor del 70 % de 'matcheo' en el 45 %-49 % del total de detecciones realizadas, en relación a sus coordenadas verdaderas. Si bien los resultados no fueron excelentes, y adjudican al modelo parte de la responsabilidad en relación a las detecciones positivas erróneas arrojadas por el sistema, esta precisión, aunque no parezca demasiado buena, era de esperarse y consecuente con los porcentajes entregados por tales tuplas, Cuadro (5.2) y (5.6). A lo largo de la prueba, se pudo observar que gran parte de las detecciones realizadas por EAST, al margen de los 'padeos' aplicados, poseían recuadros delimitadores de un ancho excesivo, en donde, además de incluir al número del coche buscado, detectaba parte de los caracteres que formaban el nombre de su destino. Esta conjunción, podría ser el justificante del 30 % de no coincidencia, y parte de los posibles errores cometidos por el módulo de reconocimiento de texto. En la Figura (5.14), puede observarse lo comentado, con un ejemplo obtenido de las pruebas realizadas.

(a) ROI objetivo. (b) Detección de EAST sobre transformación **'-g-rgb'.** (c) Ampliación de detección.

Figura 5.14: Ejemplo de precisión entregada por EAST, sobre recorte aleatorio de autobús.

En un intento por seguir ajustando el sistema, se propusieron diferentes tipos de heurísticas que atacaban distintas partes del pipeline.

Por un lado, se probaron nuevas transformaciones de imágenes a partir de la combinación de los canales que integraban cada **Top 3**, Sección (5.4.1.2) y (5.4.2.3), con la esperanza de que "combinar los mejores, condujera a mejoras"; los resultados no fueron los esperados. Tanto los porcentajes de las mejores combinaciones resultantes, como los otorgados por sus mejores tuplas, no se acercaron, en el mejor de los casos, ni a la mitad de los valores basales obtenidos anteriormente, Cuadro (5.11).

Primer enfoque

Newsize	Pad	Detecciones	Combinación
320	5	P(Combinación / (newsize,pad)) = 9 %	'-Cb-YCrCb' , 'v-luv', '-g-rgb' = 29,0 %
320	10	P(Combinación / (newsize,pad)) = 10 %	'-g-rgb' , '-Cb-YCrCb', 'v-luv' = 29,0 %

Segundo enfoque

Newsize	Pad	Detecciones	Combinación
160	15	P(Combinación / (newsize,pad)) = 8 %	'v-luv' , '-y-yuv' , '-Cb-YCrCb' = 36,0 %
128	10	P(Combinación / (newsize,pad)) = 10 %	'-Cb-YCrCb' , '-y-yuv' , 'v-luv' = 36,0 %

Cuadro 5.11: Mejores porcentajes obtenidos, luego de probar nuevas transformaciones de imágenes.

Por otra parte, se elaboraron estadísticas con el fin de encontrar la existencia de algún tipo de relación entre: las dimensiones originales de los ROI's de los autobuses y las detecciones positivas que éstas podían lograr, Sección (5.5). Si bien se pudo verificar una diferencia de aproximadamente 400 pixeles para el primer enfoque, y 200 pixeles para el segundo, entre las dimensiones que mas resultados positivos producían de las que no; a la hora de implementar la propuesta, se encontró con resultados muy similares a los entregados por el Cuadro (5.10), haciendo de esta heurística, un agregado innecesario.

6. Etapa de detección de autobuses: Análisis y selección de modelo

Esta parte del proyecto, será la encargada de testear diferentes aspectos de los modelos de detección de objetos que fueron seleccionados y descriptos en la Sección (4.1.3), para determinar, según las prestaciones que demuestren, cual de ellos completará el sistema final.

Hasta aquí, y a lo largo de las secciones anteriores, se fueron proponiendo y evaluando distintos parámetros y algoritmos, todos, con la intención de promover el mayor potencial a la última parte de nuestro pipeline, es decir, a la tarea de reconocer y devolver el número detectado de un autobús. Ahora, y mediante chequeos de precisión y pruebas de velocidad, se suplirá el modelo "teórico y preciso" de detección de objetos (hasta el momento implementado mediante la alimentación de las coordenadas de los ROI's de los autobuses, recabadas a mano), y se pondrá a prueba la totalidad del sistema.

Los siguientes análisis, fueron ejecutados utilizando una Laptop Toshiba Satelite "Intel(R) Core(TM) i7-3632QM, CPU 2.20GHz; GPU AMD Thames XT-M2 1 Gb DDR3", y estuvieron orientados a tres modelos de detección de objetos en particular: «YOLO v2», «YOLO v3 Tiny» y «SSD-MobileNet 300». A su vez, cada una de las pruebas, se realizaron teniendo en cuenta diferentes dimensiones de entrada a la ConvNet de cada red, por lo cual, de ahora en más, se referirá a cada modelo con su particular configuración, como a un modelo distinguible del otro. A continuación se listan los nueve modelos testeados:

1. YOLO v2: Entrada $(608 \times 608) \rightarrow$ Mapa de características (19×19),
2. YOLO v3 Tiny: Entrada $(608 \times 608) \rightarrow$ Mapa de características (19×19) y (38×38),
3. YOLO v2: Entrada $(416 \times 416) \rightarrow$ Mapa de características (13×13),
4. YOLO v3 Tiny: Entrada $(416 \times 416) \rightarrow$ Mapa de características (13×13) y (26×26),
5. YOLO v2: Entrada $(352 \times 352) \rightarrow$ Mapa de características (11×11),
6. YOLO v3 Tiny: Entrada $(352 \times 352) \rightarrow$ Mapa de características (11×11) y (22×22),
7. YOLO v2: Entrada $(320 \times 320) \rightarrow$ Mapa de características (10×10),
8. YOLO v3 Tiny: Entrada $(320 \times 320) \rightarrow$ Mapa de características (10×10) y (20×20),
9. SSD-MobileNet 300: Entrada (300×300)

6.1. Precisión de detecciones

Se examinará, al igual que se lo hizo con el modelo de detección de texto EAST, en las Secciones (5.4.1.3) y (5.4.2.4), la precisión de las detecciones realizadas sobre los autobuses. Para ello, se calcularán los índice IoU entre: los recuadros delimitadores de las detecciones producidas y los ROI's anotados manualmente de cada uno de los 100 autobuses de nuestra muestra.

Los nueve modelos, fueron puestos a prueba en tres categorías diferentes: corta, media y larga distancia; en relación a donde se encontraba el autobús al momento de tomar la fotografía, similar al enfoque perseguido en la Sección (5.6).

Los parámetros configurados para los modelos, fueron:

- $Threshold_{conf}$: 0,5.
- $Threshold_{IoU}$: 0,45.
- Relación de aspecto de autobuses permitidos: Sin restricción.

A continuación, los Cuadros (6.1),(6.2) y (6.3), muestran los resultados arrojados para autobuses categorizados a corta, media y larga distancia respectivamente.

>= IoU	YOLO v2				YOLO v3 Tiny				SSD-MobileNet
	608	416	352	320	608	416	352	320	300
.5	100,0 %	95,65 %	91,3 %	95,65 %	73,91 %	82,61 %	86,96 %	78,26 %	78,26 %
.55	100,0 %	95,65 %	91,3 %	95,65 %	73,91 %	82,61 %	86,96 %	78,26 %	78,26 %
.6	100,0 %	95,65 %	91,3 %	95,65 %	73,91 %	82,61 %	86,96 %	78,26 %	78,26 %
.65	95,65 %	95,65 %	91,3 %	95,65 %	73,91 %	82,61 %	86,96 %	78,26 %	78,26 %
.7	95,65 %	95,65 %	91,3 %	95,65 %	73,91 %	82,61 %	86,96 %	69,57 %	78,26 %
.75	95,65 %	95,65 %	91,3 %	91,3 %	69,57 %	78,26 %	73,91 %	60,87 %	69,57 %
.8	91,3 %	91,3 %	91,3 %	86,96 %	69,57 %	65,22 %	60,87 %	47,83 %	69,57 %
.85	78,26 %	73,91 %	73,91 %	69,57 %	43,48 %	30,43 %	39,13 %	26,09 %	56,52 %
.9	52,17 %	39,13 %	30,43 %	26,09 %	26,09 %	13,04 %	4,35 %	8,7 %	43,48 %
.95	4,35 %	4,35 %	-	-	4,35 %	-	-	4,35 %	4,35 %
1	-	-	-	-	-	-	-	-	-
Detectado	100,0 %	95,65 %	91,3 %	95,65 %	73,91 %	82,61 %	86,96 %	78,26 %	78,26 %

Cuadro 6.1: Precisión a corta distancia. 23 imágenes analizadas.

>= IoU	YOLO v2				YOLO v3 Tiny				SSD-MobileNet
	608	416	352	320	608	416	352	320	300
.5	100,0 %	100,0 %	98,08 %	100,0 %	92,31 %	76,92 %	63,46 %	40,38 %	73,08 %
.55	100,0 %	100,0 %	98,08 %	98,08 %	92,31 %	76,92 %	63,46 %	40,38 %	73,08 %
.6	100,0 %	100,0 %	98,08 %	96,15 %	92,31 %	76,92 %	63,46 %	40,38 %	73,08 %
.65	98,08 %	100,0 %	98,08 %	96,15 %	92,31 %	76,92 %	63,46 %	40,38 %	73,08 %
.7	98,08 %	100,0 %	92,31 %	96,15 %	90,38 %	75,0 %	63,46 %	40,38 %	73,08 %
.75	98,08 %	92,31 %	86,54 %	90,38 %	86,54 %	65,38 %	55,77 %	38,46 %	69,23 %
.8	90,38 %	80,77 %	67,31 %	57,69 %	78,85 %	46,15 %	36,54 %	30,77 %	63,46 %
.85	73,08 %	55,77 %	36,54 %	34,62 %	48,08 %	23,08 %	21,15 %	13,46 %	46,15 %
.9	15,38 %	15,38 %	7,69 %	13,46 %	19,23 %	3,85 %	1,92 %	1,92 %	26,92 %
.95	-	-	-	1,92 %	1,92 %	-	-	-	1,92 %
1	-	-	-	-	-	-	-	-	-
Detectado	100,0 %	100,0 %	98,08 %	100,0 %	92,31 %	80,77 %	75,0 %	55,77 %	73,08 %

Cuadro 6.2: Precisión a media distancia. 53 imágenes analizadas.

>= IoU	YOLO v2				YOLO v3 Tiny				SSD-MobileNet
	608	416	352	320	608	416	352	320	300
.5	100,0 %	92,0 %	84,0 %	72,0 %	76,0 %	12,0 %	-	-	44,0 %
.55	100,0 %	92,0 %	84,0 %	72,0 %	76,0 %	12,0 %	-	-	44,0 %
.6	100,0 %	92,0 %	84,0 %	72,0 %	76,0 %	12,0 %	-	-	44,0 %
.65	100,0 %	92,0 %	84,0 %	60,0 %	76,0 %	12,0 %	-	-	40,0 %
.7	100,0 %	84,0 %	72,0 %	44,0 %	64,0 %	12,0 %	-	-	24,0 %
.75	96,0 %	68,0 %	52,0 %	40,0 %	64,0 %	4,0 %	-	-	16,0 %
.8	88,0 %	44,0 %	36,0 %	28,0 %	32,0 %	-	-	-	12,0 %
.85	24,0 %	20,0 %	16,0 %	8,0 %	32,0 %	-	-	-	12,0 %
.9	12,0 %	4,0 %	4,0 %	4,0 %	12,0 %	-	-	-	4,0 %
.95	-	-	4,0 %	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-
Detectado	100,0 %	92,0 %	84,0 %	72,0 %	92,0 %	40,0 %	36,0 %	24,0 %	44,0 %

Cuadro 6.3: Precisión a larga distancia. 25 imágenes analizadas.

6.2. Detección temprana vs. velocidad de detección

El objetivo de esta parte del trabajo, fue seleccionar el modelo de detección de objetos, que mayor velocidad y porcentajes de detecciones positivas entregara. Para ello fue necesario estudiar el trade-off entre:

- Modelos con capacidades de *"Detección temprana"*:
  - Detección de autobuses a mayores distancias.
  - Análisis mas lento y costoso, dada la mayor dimensión de entrada a la red.
  - Recuadros delimitadores mas precisos.
  - Número de autobús no siempre legible, dada la distancia y la pérdida de calidad del objeto.
- Modelos con capacidades de *"Detecciones Veloces"*:
  - Posibilidad de detección más instantánea y a menores distancias.
  - Análisis mas rápido, dada la menor dimensión de entrada a la red.
  - Recuadros delimitadores menos precisos.
  - Número de autobús con óptima legibilidad, en la mayoría de los casos.

A diferencia de las pruebas realizadas anteriormente, las cuales tomaban como base al conjunto de 100 imágenes de la muestra poblacional, en esta parte, y dadas las circunstancias, se chequearon las velocidades y los porcentajes de las detecciones positivas sobre 22 nuevos conjuntos, los cuales estuvieron a cargo de simular una situación mas acorde a la realidad.

Cada conjunto, estuvo formado por una serie de imágenes (frames), que corresponden a un mismo objetivo 'autobús' arribando a una parada; en el que se cercioró, que tanto el autobús como su número de línea, sean visibles y legibles al ojo humano, para reducir posibles falsos negativos. Estas ráfagas, o conjuntos de pruebas, fueron confeccionados mediante la segmentación de 22 videos capturados por siete dispositivos móviles diferentes y a diferentes calidades (integrantes del conjunto de dispositivos que se utilizaron en la construcción de la muestra poblacional, Sección (5)). Para tal fin fue utilizada la herramienta *Ffmpeg*[16]. *Ésta,* open source y basada en línea de comandos, permitió obtener cada una de las imágenes a una velocidad configurada a 4 FPS.

Ante la necesidad de distinguir el origen de la velocidad, de una determinada detección positiva, realizada por alguno de los modelos (ie. distinguir si fue efectuada a mayor velocidad, porque se detecto el autobús a mayor distancia, o porque el modelo analizó una mayor cantidad de frames en menor tiempo), se ejecutó el sistema completo, nuevamente teniendo en cuenta los mejores resultados de cada uno de los enfoques de la Sección (5), registrándose las siguientes variables en cada corrida:

- **F.T** : Cantidad *total de frames* de conjunto de prueba.
- **F.U** : Cantidad de *frames procesados* hasta detección positiva.
- **V** : *Velocidad de análisis* (ie. tiempo en segundos hasta realizar detección positiva).
- **P** : *Probabilidad de certeza* de detección positiva (ie. valor de veracidad de la línea candidata).
- **R** : *Ranking.* Puesto de la línea candidata, en relación a **P**.

A continuación, y antes de proseguir con los análisis, la Figura (6.4), expone visualmente las diferentes detección realizadas por cada uno de los 9 modelos, sobre una ráfaga de 10 imágenes, en el acercamiento de un autobús a una parada. Estos resultados, aunque no parezcan de mucha utilidad, permitirán conocer las máximas y mínimas distancias útiles de trabajo de cada uno, muy importantes a la hora de seleccionar el mejor.

Debajo, se detallan los resultados obtenidos de las pruebas realizadas para ambos enfoques, teniendo en cuenta los siguientes conjuntos de parámetros:

[16] `https://ffmpeg.org/`

	Distancia aproximada (en Metros)									
	Larga					Media			Corta	
	47m	40m	30m	25m	20m	15m	10m	7	5m	2m
										
YOLOv2 608		X	X	X	X	X	X	X	X	X
YOLOv2 416				X	X	X	X	X	X	X
YOLOv2 352					X	X	X	X	X	X
YOLOv2 320					X		X	X	X	X
YOLOv3 Tiny 608					X	X	X	X	X	X
YOLOv3 Tiny 416							X	X	X	X
YOLOv3 Tiny 352								X	X	X
YOLOv3 Tiny 320								X	X	X
SSD-MobileNet 300						X	X	X	X	X

Cuadro 6.4: Distancias vs. detecciones de modelos.

- Relación de aspecto de detecciones permitidas: 0.92 ± 0.24. Sección (5.6).
- Módulo detector de texto (EAST)
  - $Threshold_{conf}$: 0,001.
  - $Threshold_{IoU}$: 0,1.
- Módulo de reconocimiento de texto (OCR-Tesseract)
  - *lenguaje*: Inglés.
  - *psm*: Modo 7.
  - *oem*: Modo 1.
- Primer enfoque
  - Tuplas *(newsize, pad, transformaciones)*: (128, 0, Top 15) y (160, 5, Top 15). Sección (5.4.1.1).
  - Criterio de recorte de autobuses: Relación de aspecto cuadrada.
- Segundo enfoque
  - Tuplas *(newsize, pad, transformaciones)*: (96, 5, Top 15) y (96, 15, Top 13). Sección (5.4.2.2).
  - Criterio de recorte de autobuses: Heurística de Sección (5.4.2.1).

6.2.1. Pruebas: Primer enfoque

- YOLO v2: Cuadro (6.6).
- YOLO v3 Tiny: Cuadro (6.8).
- SSD-MobileNet 300: Cuadro (6.10a).

6.2.2. Pruebas: Segundo enfoque

- YOLO v2: Cuadro (6.7).
- YOLO v3 Tiny: Cuadro (6.9).
- SSD-MobileNet 300: Cuadro (6.10b).

6.3. Conclusiones

Para concluir la sección, comenzaremos analizando las tablas de precisiones, Cuadros (6.1), (6.2) y (6.3). En éstas, claramente puede observarse que, para el caso de los modelos derivados de YOLO v2 y YOLO v3 Tiny, la disminución de la dimensión de entrada a sus redes, es directamente proporcional con la cantidad de detecciones que estos pueden realizar; es decir, que la calidad de las imágenes de entrada juegan un rol muy importante a la hora de las detecciones que realizan. Si se compara el porcentaje de detecciones en relación a las distancias de ubicación de los autobuses, claramente se observa que las variaciones de YOLO v2 sacan ventaja del resto, con una precisión del 80 % y 75 % en alrededor del 90 % de las imágenes detectadas para la corta y media distancia respectivamente. Aunque tal precisión se ve disminuida a la hora de reconocer autobuses a larga distancia, sigue manteniendo el 87 % de detecciones realizadas.

Con respecto al modelo SSD-MobileNet, a primera vista podría parecer que no tuviese forma de competir con sus adversarios, pero, si se observa detenidamente, este pequeño modelo, con una dimensión de entrada a su red de tan solo 300×300 pixeles, obtiene tanto en corta como en media distancia, casi los mismos porcentajes de detecciones que las variantes de YOLO v3 Tiny; otorgando 78 % y 73 %, en comparación con 79 % y 75 % que en promedio, calculan las variantes de este último. Además, en cuanto a precisión se trata, SSD-MobileNet entrega a corta distancia, un 85 % en el 56 % de los autobuses detectados, superando en 10 puntos al mejor rendimiento otorgado por la versión 608, y superando a las demás de 16 a 25 puntos, en el mejor de los casos. Con respeto a la media distancia sucede algo similar; con una precisión de aproximadamente 80 % en el 64 % de los autobuses detectados, solo se ve superada en un 12 % por YOLO v3 Tiny 608, pero es un 45 % mejor que el resto de sus variaciones.

La categoría de larga distancia merece un punto aparte de análisis. Si bien YOLO v2, en su versión 608, otorga muy buenos resultados con una precisión del 80 % en el 88 % de los autobuses con un 100 % de detecciones, superando ampliamente al modelo SSD-MobileNet y a todas las versiones de su par YOLO v3 Tiny; no hay que perder de vista que se están comparando arquitecturas ConvNet de complejidades, y en consecuencia, velocidades muy diferentes, lo cual pude ser determinante al momento de tomar la decisión final.

Siguiendo este enfoque, en la Sección (6.2), se pudo analizar el comportamiento de cada uno de estos modelos sobre 22 nuevos conjuntos de imágenes. En cada uno de los análisis, se evaluaron los rendimientos en base a la cantidad de frames analizados hasta detectar el número correcto del autobús, el tiempo transcurrido hasta tal proceso, y cual fue la confiabilidad de la detección mediante un valor de certeza. Esta batería de pruebas, fueron realizadas configurando la etapa de detección y reconocimiento de líneas con los mejores valores de los parámetros de cada uno de los dos enfoques seleccionados en la Sección (5.7). Los mejores resultados arrojados por cada uno de estos modelos, en base a los porcentajes y las velocidades de las detecciones realizadas, fueron resumidos en el siguiente Cuadro (6.5).

Modelo	Primer Enfoque				Segundo Enfoque			
	(128,0,Top15)	V	(160,5,Top15)	V	(96,5,Top15)	V	(96,15,Top13)	V
YOLOv2 608	81 %	33,4s	63 %	30,2s	68 %	33,9s	72 %	34,6s
YOLOv2 416	68 %	25,3s	81 %	21,5s	72 %	20,2s	77 %	22,1s
YOLOv2 352	72 %	18,8s	63 %	19s	72 %	16,5s	72 %	14,7s
YOLOv2 320	72 %	15,3s	68 %	16,5s	72 %	13s	72 %	14,2s
YOLOv3 tiny 608	68 %	14,2s	68 %	13,9s	72 %	13,5s	59 %	11,9s
YOLOv3 tiny 416	63 %	12,3s	54 %	12,2s	54 %	11,5s	54 %	9,8s
YOLOv3 tiny 352	40 %	10,3s	36 %	11,8s	50 %	10,6s	36 %	9,9s
YOLOv3 tiny 320	50 %	8s	50 %	9,5s	54 %	8,5s	45 %	7s
SSD-MobileNet 300	72 %	5s	59 %	4,8s	63 %	4,8s	63 %	4,4s

Cuadro 6.5: Mejores resultados de ejecuciones del sistema para cada modelo de detección de objetos.

Antes de empezar a realizar observaciones y sacar conclusiones sobre los diferentes valores arrojados, retomaremos sobre las bases del problema, que tuvo como objetivo el desarrollo del presente trabajo.

Como primer punto, el algoritmo propuesto tuvo como finalidad determinar los números de líneas de los autobuses, mediante el análisis de una sucesión de imágenes provenientes de las capturas realizadas desde una parada aleatoria, durante la aproximación y temporal detención de un determinado autobús.

Acorde a lo analizado en la Sección (5.7), y con el objetivo de conseguir los mejores resultados para la *"detección y el reconocimiento de las líneas"* (etapa final del sistema), se observo que en el mejor de los casos, una combinación de los parámetros $newsize = 128$ y $pad = 0$, en conjunto con las mejores transformaciones que integraban el $Top\,15$ del primer enfoque, Cuadro (5.2), permitían un reconocimiento correcto del número del coche, con una probabilidad de aproximadamente 62 % sobre una imagen simple.

En esta última sección, y en pos de seleccionar el mejor modelo de detección de autobuses que integre la etapa inicial del pipeline, se ejecuto el sistema de extremo a extremo, probando nueve variantes de diferentes modelos, con los mejores parámetros de ambos enfoques; en cada corrida, a partir de un conjunto de x imágenes, y con una probabilidad p de realizar una detección positiva sobre cada una (en relación a los parámetros escogidos), se midió el rendimiento general mediante diferentes variables, descriptas anteriormente.

En base al resumen del Cuadro (6.5), podríamos decir, si solo nos basásemos en los porcentajes de detecciones entregados, que tanto la variación 608 como la 416 de YOLO v2, mediante el uso de las mejores tuplas del primer enfoque, serían los candidatos. El problema surge cuando vemos los tiempos, que en promedio, cada uno invirtió para deducir el número correcto del autobús. Si observamos, YOLO v2 608 con 33,4 segundos, tardó 10 segundos más que la variación 416; si a esto se suma, Sección (4.1), que un autobús, desde su aproximación a la parada hasta su salida para retornar el recorrido, tarda entre 5 y 20 segundos, para cuando el sistema entregue la respuesta habría grandes chances de haberlo perdido. Ésto, nos hace retrotraer a los análisis realizados sobre la precisión y la comparación realizada en el Cuadro (6.4), que evaluaba las capacidades de detección en función de las distancias. Recordando las diferentes propiedades que tienen los modelos con capacidades de "Detecciónes tempranas" y los de "Detecciones veloces", podríamos deducir que no nos sería útil detectar autobuses a largas distancias; primero, por la poca precisión que entregaron los modelos en tal categoría (salvo YOLOv2 608, pero quedo descartado por lo dicho recientemente), y segundo, por los efectos que trae aparejada una detección a gran distancia en relación a la poca visualización de los números de línea.

Finalmente y como consecuencia de tales planteamientos, con porcentajes de detecciones de **72 %**, los siguientes cuatro modelos fueron los considerados como potenciales candidatos finales:

1. YOLO v2 320, tupla: (96, 5, Top 15), Segundo enfoque.
2. YOLO v2 352, tupla: (96, 15, Top 13), Segundo enfoque.
3. YOLO v3 Tiny 608, tupla: (96, 5, Top 15), Segundo enfoque.
4. SSD-MobileNet 300, tupla: (128, 0, Top 15), Primer enfoque.

En relación a todo lo detallado, se consideró que el modelo a seleccionar debía poseer un justo equilibrio entre velocidad y precisión, a media y corta distancia, y por supuesto, un buen porcentaje de detecciones positivas. Basándose en tales premisas, y principalmente por los "bajísimos" tiempos de procesamiento que obtuvo a lo largo de cada una de las ejecuciones, se optó por **SSD-MobileNet 300** con el conjunto de parámetros *newsize*=**128**, *pad*=**0** y el ***Top 15*** de transformaciones, como los integrantes finales de nuestro sistema.

Éste, con tiempos que oscilan los **5 segundos**, fue 2.16 veces mas veloz que el segundo mejor tiempo obtenido (modelo 1). Además, y según lo detallado en Cuadro (6.10a), fue capaz de realizar el reconocimiento del número del coche antes del $\mathbf{7}^{mo}$ frame analizado, con una certeza de un **71,1 %**, retornando la mayoría de las veces el candidato ranqueado en $\mathbf{1}^{er}$ posición.

Configuración - (128, 0, Top15)

Info. de set			608				416				352				320			
Id	**Bús**	**F.T**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**
01	10	4	1	8,45	80	1	2	9,18	62	1	2	7,51	83	1	2	6,27	50	1
02	65	10	4	23,72	33	1	5	15,85	100	1	1	3,68	50	1	5	11,97	50	1
03	24	4	1	8,39	100	1	-	-	-	-	1	4,5	50	1	3	11,69	100	1
04	67	7	7	46,81	50	1	-	-	-	-	-	-	-	-	7	20,43	100	1
05	29	4	4	27,94	50	1	3	13,0	50	1	3	10,15	50	1	-	-	-	-
06	18	5	2	14,43	25	3	2	8,44	50	1	-	-	-	-	2	6,44	25	1
07	71	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
08	19	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09	12	6	-	-	-	-	-	-	-	-	-	-	-	-	6	19,98	33	1
10	41	18	12	85,28	100	1	16	70,49	50	1	17	59,01	100	1	12	28,42	100	1
11	18	15	5	34,17	100	1	5	19,49	67	1	5	12,22	33	1	7	15,61	100	1
12	45	12	6	46,23	50	1	9	45,93	50	1	5	17,79	100	1	5	15,01	100	1
13	61	7	1	8,27	86	1	1	5,43	71	1	1	4,28	60	1	1	4,13	75	1
14	66	19	3	20,46	50	1	6	17,96	67	1	8	16,85	100	1	9	16,15	75	1
15	36	19	1	8,03	80	1	1	4,72	100	1	2	7,58	57	1	50	3,71	50	1
16	35	22	1	12,25	100	1	7	22,15	75	1	6	13,55	67	1	6	11,92	67	1
17	35	14	2	18,28	60	1	1	6,23	33	1	1	3,99	60	1	2	7,68	71	1
18	10	39	15	88,73	67	1	19	57,45	71	1	21	41,99	100	1	21	36,25	100	1
19	29	10	6	42,95	100	1	-	-	-	-	-	-	-	-	-	-	-	-
20	23	14	-	-	-	-	-	-	-	-	8	21,95	50	1	-	-	-	-
21	12	12	3	25,54	100	1	10	32,24	100	1	11	24,45	100	1	-	-	-	-
22	32	16	10	81,9	43	1	10	50,56	50	1	14	51,87	43	1	10	29,51	50	1
Promedios			4,7	33,4	70,8	1,1	6,5	25,3	66,4	1,0	6,6	18,8	68,9	1,0	9,3	15,3	71,6	1,0
Detecciones +			18/22 → 81 %				15/22 → 68 %				16/22 → 72 %				16/22 → 72 %			

Configuración - (160, 5, Top15)

Info. de set			608				416				352				320			
Id	**Bús**	**F.T**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**
01	10	4	1	9,13	50	1	2	9,73	57	1	2	8,73	33	1	2	7,19	100	1
02	65	10	4	24,78	75	1	7	22,82	86	1	3	12,25	50	1	5	14,34	100	1
03	24	4	1	9,06	50	1	1	6,03	67	1	1	4,69	100	1	1	5,11	100	1
04	67	7	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05	29	4	4	-	-	-	4	18,16	33	1	-	-	-	-	-	-	-	-
06	18	5	3	22,98	40	1	3	14,6	40	1	2	7,8	20	1	2	6,95	50	1
07	71	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
08	19	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09	12	6	-	-	-	-	2	12,18	50	1	-	-	-	-	-	-	-	-
10	41	18	-	-	-	-	14	60,85	100	1	-	-	-	-	15	47,0	12	3
11	18	15	6	43,66	100	1	3	10,04	100	1	4	9,98	100	1	5	9,78	100	1
12	45	12	3	24,06	33	1	5	24,75	33	1	9	40,87	25	1	5	18,05	100	1
13	61	7	1	8,29	50	1	1	5,32	100	1	1	4,41	50	1	1	4,22	33	1
14	66	19	2	12,85	67	1	6	17,75	60	1	8	18,49	100	1	9	17,69	100	1
15	36	19	1	8,04	17	2	1	5,24	40	1	1	4,37	100	1	1	4,18	33	1
16	35	22	3	20,93	100	1	7	21,43	80	1	6	15,41	75	1	6	12,5	83	1
17	35	14	1	10,27	14	1	1	7,48	33	2	1	5,07	22	1	1	5,01	33	1
18	10	39	15	84,07	100	1	19	53,6	86	1	22	49,46	67	1	21	40,04	75	1
19	29	10	6	46,06	33	1	-	-	-	-	-	-	-	-	5	22,28	50	1
20	23	14	-	-	-	-	3	9,86	50	1	8	25,2	50	1	-	-	-	-
21	12	12	-	-	-	-	10	29,63	100	1	-	-	-	-	-	-	-	-
22	32	16	11	98,8	75	1	10	56,95	100	1	13	59,89	44	1	10	33,6	25	2
Promedios			4,3	30,2	57,4	1,1	5,5	21,5	67,5	1,1	5,8	19,0	59,7	1,0	5,9	16,5	66,3	1,2
Detecciones +			14/22 → 63 %				18/22 → 81 %				14/22 → 63 %				15/22 → 68 %			

Cuadro 6.6: Primer enfoque - YOLO v2.

Configuración - (96, 5, Top15)

Info. de set			608				416				352				320			
Id	**Bús**	**F.T**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**
01	10	4	1	8,41	67	1	2	8,7	100	1	2	6,69	50	1	2	6,18	100	1
02	65	10	4	23,59	100	1	7	22,26	50	1	6	18,47	50	1	5	12,21	100	1
03	24	4	-	-	-	-	-	-	-	-	2	7,33	50	1	1	3,77	100	1
04	67	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05	29	4	3	20,51	33	1	4	16,21	14	2	3	9,29	67	1	-	-	-	-
06	18	5	2	14,3	17	2	1	4,05	100	1	1	3,35	50	1	3	9,54	33	2
07	71	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
08	19	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09	12	6	6	41,33	100	1	3	13,12	20	1	-	-	-	-	4	12,32	50	1
10	41	18	12	83,5	33	1	-	-	-	-	18	53,17	33	1	-	-	-	-
11	18	15	6	43,23	100	1	7	23,89	50	1	8	20,75	50	1	5	9,21	100	1
12	45	12	5	39,69	33	1	7	28,16	100	1	5	15,73	33	2	5	13,89	100	1
13	61	7	1	7,91	25	2	1	4,61	33	1	1	3,69	50	1	3	8,05	29	1
14	66	19	3	19,53	50	1	6	17,27	60	1	8	16,63	100	1	9	15,57	75	1
15	36	19	1	7,27	33	1	1	4,47	67	1	1	3,9	50	1	1	3,39	25	2
16	35	22	3	19,84	50	1	7	19,75	100	1	6	12,96	71	1	6	11,49	67	1
17	35	14	1	8,42	25	2	2	9,42	33	2	2	7,35	75	1	1	3,25	67	1
18	10	39	12	86,0	67	1	19	51,93	60	1	22	46,96	100	1	22	37,82	100	1
19	29	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
20	23	14	-	-	-	-	9	35,04	14	2	8	21,32	40	1	8	15,53	20	2
21	12	12	-	-	-	-	7	20,22	33	1	-	-	-	-	9	17,69	100	1
22	32	16	11	84,76	33	1	10	44,55	33	1	5	16,06	100	1	10	28,04	67	1
Promedios			4,7	33,9	51,1	1,2	5,8	20,2	54,2	1,2	6,1	16,5	60,6	1,1	5,9	13,0	70,8	1,2
Detecciones +			$15/22 \rightarrow 68\,\%$				$16/22 \rightarrow 72\,\%$				$16/22 \rightarrow 72\,\%$				$16/22 \rightarrow 72\,\%$			

Configuración - (96, 15, Top13)

Info. de set			608				416				352				320			
Id	**Bús**	**F.T**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**
01	10	4	1	8,69	100	1	3	13,5	100	1	2	6,87	50	1	-	-	-	-
02	65	10	6	37,88	67	1	7	21,83	50	1	6	18,2	75	1	5	11,88	50	1
03	24	4	-	-	-	-	2	8,91	50	1	-	-	-	-	2	6,6	33	1
04	67	7	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-
05	29	4	3	19,95	50	1	4	15,15	50	1	4	13,33	20	1	-	-	-	-
06	18	5	1	7,19	33	1	1	4,44	100	1	1	3,17	100	1	2	5,14	50	1
07	71	6	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-
08	19	5	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-
09	12	6	6	41,22	33	2	5	21,94	33	1	4	12,8	50	1	4	11,34	33	2
10	41	18	18	121,47	100	1	18	62,48	50	1	-	-	-	-	15	33,62	50	1
11	18	15	6	39,71	100	1	7	23,2	100	1	5	12,06	100	1	6	11,43	50	1
12	45	12	2	14,05	100	1	9	34,66	25	1	9	27,54	100	1	6	16,14	33	1
13	61	7	1	7,62	50	1	1	4,23	67	1	1	3,64	50	1	4	10,66	67	1
14	66	19	6	39,37	50	1	6	17,9	100	1	8	16.84	75	1	9	15,27	100	1
15	36	19	1	7,32	67	1	1	4,27	100	1	1	3,9	100	1	1	3,33	50	1
16	35	22	3	18,87	25	1	7	20,33	100	1	6	13,23	50	1	6	11,18	50	1
17	35	14	1	7,72	100	1	1	4,79	60	1	1	4,02	75	1	1	3,16	80	1
18	10	39	15	83,17	50	1	19	51,63	71	1	22	45,16	100	1	22	36,71	100	1
19	29	10	3	20,54	100	1	-	-	-		-	-	-	-	-	-	-	-
20	23	14	-	-	-	-	-	-	-		8	20,06	100	1	8	13,6	100	1
21	12	12	-	-	-	-	7	19,78	25	2	10	20,88	100	1	12	24.31	100	1
22	32	16	11	78,83	67	1	11	46,75	100	1	5	15,22	100	1	9	22,27	33	1
Promedios			5,3	34,6	68,2	1,1	6,4	22,1	69,5	1,1	5,8	14,7	77,8	1,0	7,0	14,2	61,2	1,1
Detecciones +			$16/22 \rightarrow 72\,\%$				$17/22 \rightarrow 77\,\%$				$16/22 \rightarrow 72\,\%$				$16/22 \rightarrow 72\,\%$			

Cuadro 6.7: Segundo enfoque - YOLO v2.

Configuración - (128, 0, Top15)

Info. de set			608				416				352				320			
Id	**Bús**	**F.T**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**
01	10	4	2	6,93	80	1	-	-	-	-	-	-	-	-	-	-	-	-
02	65	10	1	6,19	33	1	-	-	-	-	-	-	-	-	7	7,28	100	1
03	24	4	2	8,03	33	1	2	4,39	33	1	-	-	-	-	-	-	-	-
04	67	7	-	-	-	-	-	-	-	-	-	-	-	-	7	10,27	50	1
05	29	4	-	-	-	-	3	6,43	25	1	-	-	-	-	3	4,35	25	1
06	18	5	1	3,34	50	1	1	3,47	50	1	3	6,2	50	1	-	-	-	-
07	71	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
08	19	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09	12	6	-	-	-	-	6	13,33	33	1	-	-	-	-	-	-	-	-
10	41	18	15	38,23	50	1	17	26,0	67	1	13	20,64	33	2	-	-	-	-
11	18	15	5	9,57	50	1	9	13,62	50	1	9	7,32	100	1	11	7,38	50	1
12	45	12	5	17,97	75	1	8	20,21	50	1	-	-	-	-	3	3,4	100	1
13	61	7	1	3,92	75	1	1	3,08	75	1	1	3,06	17	2	1	3,82	86	1
14	66	19	9	17,17	100	1	10	14,15	50	1	14	14,88	50	1	14	11,42	62	1
15	36	19	1	3,61	67	1	14	15,61	100	1	16	11,97	60	1	16	10,14	33	2
16	35	22	3	6,82	100	1	14	15,57	50	1	-	-	-	-	-	-	-	-
17	35	14	1	4,02	50	1	1	4,1	33	1	2	4,96	67	1	1	4,14	57	1
18	10	39	20	36,67	100	1	20	18,34	17	2	21	14,64	67	1	22	13,21	75	1
19	29	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
20	23	14	8	21,68	100	1	-	-	-	-	-	-	-	-	-	-	-	-
21	12	12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
22	32	16	10	28,49	100	1	10	13,38	33	2	10	8,72	100	1	14	12,57	56	1
Promedios			5,6	14,2	70,9	1,0	8,3	12,3	47,6	1,1	9,9	10,3	60,4	1,2	9,0	8,0	63,1	1,1
Detecciones +			15/22 → 68 %				14/22 → 63 %				9/22 → 40 %				11/22 → 50 %			

Configuración - (160, 5, Top15)

Info. de set			608				416				352				320			
Id	**Bús**	**F.T**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**
01	10	4	2	6,86	75	1	-	-	-	-	-	-	-	-	-	-	-	-
02	65	10	1	8,23	100	1	-	-	-	-	-	-	-	-	7	9,54	62	1
03	24	4	1	4,97	100	1	3	9,3	100	1	-	-	-	-	3	5,44	100	1
04	67	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05	29	4	4	14,94	67	1	3	8,17	20	1	-	-	-	-	-	-	-	-
06	18	5	3	10,99	20	2	1	4,56	100	1	-	-	-	-	-	-	-	-
07	71	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
08	19	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09	12	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	41	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	18	15	6	14,05	100	1	8	11,93	100	1	9	7,93	100	1	11	8,12	100	1
12	45	12	2	9,44	100	1	3	6,87	33	1	5	11,71	25	1	7	15,9	20	2
13	61	7	1	4,34	50	1	1	3,46	60	1	1	3,76	100	1	1	5,53	50	1
14	66	19	9	18,38	100	1	10	15,81	50	1	14	17,56	71	1	14	12,53	88	1
15	36	19	1	4,42	50	1	13	13,62	100	1	16	12,84	75	1	16	11,32	60	1
16	35	22	3	7,9	50	1	14	16,79	60	1	-	-	-	-	-	-	-	-
17	35	14	1	5,0	33	1	1	5,2	50	1	2	6,24	57	1	1	5,51	29	1
18	10	39	20	41,27	50	1	20	19,72	100	1	21	15,56	57	1	22	13,52	50	1
19	29	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
20	23	14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
21	12	12	10	23,77	33	1	-	-	-	-	-	-	-	-	-	-	-	-
22	32	16	10	35,3	75	1	11	18,5	50	1	13	18,54	33	1	10	7,61	50	1
Promedios			4,9	13,9	66,9	1,1	7,3	11,2	68,6	1,0	10,1	11,8	64,6	1,0	9,20	9,5	60,9	1,1
Detecciones +			15/22 → 68 %				12/22 → 54 %				8/22 → 36 %				10/22 → 50 %			

Cuadro 6.8: Primer enfoque - YOLO v3 Tiny.

Configuración - (96, 5, Top15)

Info. de set			608				416				352				320			
Id	**Bús**	**F.T**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**
01	10	4	4	13,4	50	1	-	-	-	-	-	-	-	-	-	-	-	-
02	65	10	4	15,33	100	1	-	-	-	-	-	-	-	-	7	6,78	50	1
03	24	4	1	3,67	25	1	3	6,26	20	2	-	-	-	-	4	6,01	33	2
04	67	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05	29	4	3	8,6	40	1	-	-	-	-	3	5,88	20	1	-	-	-	-
06	18	5	1	2,94	100	1	1	2,8	100	1	3	5,18	33	1	3	3,96	80	1
07	71	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
08	19	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09	12	6	-	-	-	-	4	8,41	50	1	-	-	-	-	-	-	-	-
10	41	18	-	-	-	-	18	25,8	100	1	15	19,37	50	1	17	18,14	50	1
11	18	15	5	9,47	50	1	10	14,31	100	1	9	7,3	100	1	11	7,55	50	1
12	45	12	8	24,2	100	1	-	-	-	-	8	13,17	33	1	9	12,91	25	1
13	61	7	3	9,0	40	1	2	4,99	50	1	1	2,43	33	1	1	2,81	50	1
14	66	19	9	16,53	100	1	10	13,29	50	1	14	14,75	80	1	14	11,21	88	1
15	36	19	1	3,72	60	1	13	13,07	80	1	16	12,1	20	2	16	10,18	25	2
16	35	22	3	6,48	100	1	14	14,92	20	2	-	-	-	-	-	-	-	-
17	35	14	1	3,29	75	1	1	3,37	40	1	2	3,88	100	1	1	2,83	17	2
18	10	39	22	40,73	64	1	20	17,87	100	1	22	16,84	100	1	22	13,01	83	1
19	29	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
20	23	14	8	20.45	33	1	-	-	-	-	-	-	-	-	-	-	-	-
21	12	12	10	19,39	100	1	-	-	-	-	-	-	-	-	-	-	-	-
22	32	16	10	26,5	60	1	10	12,94	40	1	13	15,83	33	2	10	7,3	67	1
Promedios			5,8	13,5	68,6	1	8,8	11,5	62,5	1,2	9,6	10,6	54,7	1,2	9,6	8,5	51,5	1,2
Detecciones +			16/22 → 72 %				12/22 → 54 %				11/22 → 50 %				12/22 → 54 %			

Configuración (96, 15, Top13)

Info. de set			608				416				352				320			
Id	**Bús**	**F.T**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**
01	10	4	3	9,15	75	1	-	-	-	-	-	-	-	-	-	-	-	-
02	65	10	-	-	-	-	-	-	-	-	-	-	-	-	7	5,99s	100	1
03	24	4	1	3,57	50	1	-	-	-	-	-	-	-	-	3	3,58	50	1
04	67	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05	29	4	4	11,08	40	1	4	7,57	33	2	-	-	-	-	3	3,54	20	2
06	18	5	-	-	-	-	1	2,68	100	1	2	2,58	33	1	-	-	-	-
07	71	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
08	19	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09	12	6	5	15,98	50	1	4	8,16	100	1	-	-	-	-	-	-	-	-
10	41	18	-				-	-	-	-	-	-	-	-	-	-	-	-
11	18	15	5	9,42	50	1	10	13,38	33	1	13	13,32	50	1	13	10,41	67	1
12	45	12	-	-	-	-	5	6,81	25	2	-	-	-	-	-	-	-	-
13	61	7	1	3,27	33	1	1	2,34	60	1	1	2,17	100	1	1	2,55	100	1
14	66	19	9	16,44	80	1	10	12,52	100	1	14	13,39	80	1	14	10,75	88	1
15	36	19	1	3,55	33	2	13	13,15	43	1	16	11,57	33	1	16	9,92	83	1
16	35	22	3	6,41	100	1	14	14,41	100	1	-	-	-	-	-	-	-	-
17	35	14	1	3,18	100	1	1	3,16	75	1	2	3,55	100	1	1	2,7	33	1
18	10	39	22	39,47	78	1	22	21,06	86	1	22	16,0	100	1	22	13,0	100	1
19	29	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
20	23	14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
21	12	12	10	18,88	100	1	-	-	-	-	-	-	-	-	-	-	-	-
22	32	16	10	24,82	50	1	10	12,64	25	2	13	16,32	20	1	10	6,86	67	1
Promedios			5,8	11,9	64,5	1,1	7,9	9,8	65	1,2	10,4	9,9	64,5	1	9	7,0	70,8	1,1
Detecciones +			13/22 → 59 %				12/22 → 54 %				8/22 → 36 %				10/22 → 45 %			

Cuadro 6.9: Segundo enfoque - YOLO v3 Tiny.

Info. de set			(128, 0, Top15)				(160, 5, Top15)			
Id	**Bús**	**F.T**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**
01	10	4	2	3,3	75	1	2	3,67	57	1
02	65	10	1	2,3	100	1	3	6,57	33	1
03	24	4	1	2,57	100	1	1	3,41	50	1
04	67	7	-	-	-	-	-	-	-	-
05	29	4	-	-	-	-	-	-	-	-
06	18	5	1	1,71	100	1	2	4,07	25	1
07	71	6	-	-	-	-	-	-	-	-
08	19	5	-	-	-	-	-	-	-	-
09	12	6	-	-	-	-	-	-	-	-
10	41	18	14	11,55	40	1	-	-	-	-
11	18	15	8	6,52	50	1	6	4,55	100	1
12	45	12	5	4,96	100	1	-	-	-	-
13	61	7	2	3,03	75	1	2	2,69	50	1
14	66	19	11	3,41	88	1	11	3,57	100	1
15	36	19	1	2,14	50	1	1	2,8	50	1
16	35	22	10	4,9	45	2	-	-	-	-
17	35	14	1	2,28	33	1	1	2,83	25	1
18	10	39	19	4,83	50	1	19	5,41	71	1
19	29	10	-	-	-	-	-	-	-	-
20	23	14	6	6,23	100	1	4	3,93	100	1
21	12	12	11	6,21	100	1	9	3,17	100	1
22	32	16	10	13,91	14	2	10	15,3	33	1
Promedios			6,2	5,0	71,7	1,1	5,5	4,8	61,1	1,0
Detecciones +			$16/22 \rightarrow 72\,\%$				$13/22 \rightarrow 59\,\%$			

(a) Primer enfoque.

Info. de set			(96, 5, Top15)				(96, 15, Top13)			
Id	**#Bús**	**F.T**	**F.U**	**V**	**P**	**R**	**F.U**	**V**	**P**	**R**
01	10	4	2	2,58	100	1	4	5,9	33	1
02	65	10	3	3,94	100	1	3	3,8	67	1
03	24	4	4	7,3	100	1	4	6,16	50	1
04	67	7	-	-	-	-	-	-	-	-
05	29	4	4	4,69	50	1	-	-	-	-
06	18	5	-	-	-	-	2	3,01	17	2
07	71	6	-	-	-	-	-	-	-	-
08	19	5	-	-	-	-	-	-	-	-
09	12	6	-	-	-	-	-	-	-	-
10	41	18	13	7,52	33	1	11	3,99	50	1
11	18	15	7	4,02	50	1	7	3,97	100	1
12	45	12	5	4,08	25	2	5	3,72	80	1
13	61	7	2	2,04	67	1	2	2,11	20	1
14	66	19	11	3,34	75	1	11	3,23	86	1
15	36	19	1	1,87	75	1	1	1,94	100	1
16	35	22	-	-	-	-	-	-	-	-
17	35	14	1	1,97	25	2	1	1,83	100	1
18	10	39	22	8,82	78	1	19	4,13	100	1
19	29	10	-	-	-	-	-	-	-	-
20	23	14	5	4,49	100	1	3	1,83	50	1
21	12	12	-	-	-	-	-	-	-	-
22	32	16	10	10,71	33	2	14	15,59	60	1
Promedios			6,5	4,8	65,1	1,2	6,2	4,4	65,2	1,1
Detecciones +			$14/22 \rightarrow 63\,\%$				$14/22 \rightarrow 63\,\%$			

(b) Segundo enfoque.

Cuadro 6.10: SSD-MobileNet 300.

7. Experimentos

Para finalizar el trabajo, se decidió realizar una última prueba incorporando los mejores componentes al sistema. Se tomó como base al modelo SSD-MobileNet 300, seleccionado en la Sección anterior (6.3), y la combinación de parámetros $newsize$=128, pad=0 y el Top 15 de transformaciones (Sección (5.1)); en conjunto con las heurísticas del primer enfoque y el filtrado de aspectos de autobuses entre los valores $(0.92 - 0.24) \leq \frac{ancho}{alto} \leq (0.92 + 0.24)$, para intentar simular una situación mas cercana a la realidad, en donde se pueda conocer el verdadero rendimiento del sistema.

Para ello, y a diferencia de los test realizados en las Secciones (6.2.1) y (6.2.2), en donde el sistema era alimentado mediante un conjunto de imágenes cargadas desde disco; se procedió a adaptar el algoritmo, para que en este caso, pudiese realizar capturas en tiempo real desde una cámara web estándar. Ésta, configurada a una velocidad de lectura de 5 FPS y una resolución de 640×480 pixeles, fue posicionada frente un televisor, Figura (7.1), en el cual, intentando recrear una "parada de autobuses virtual", reproducía un video de la aproximación de la línea 66.

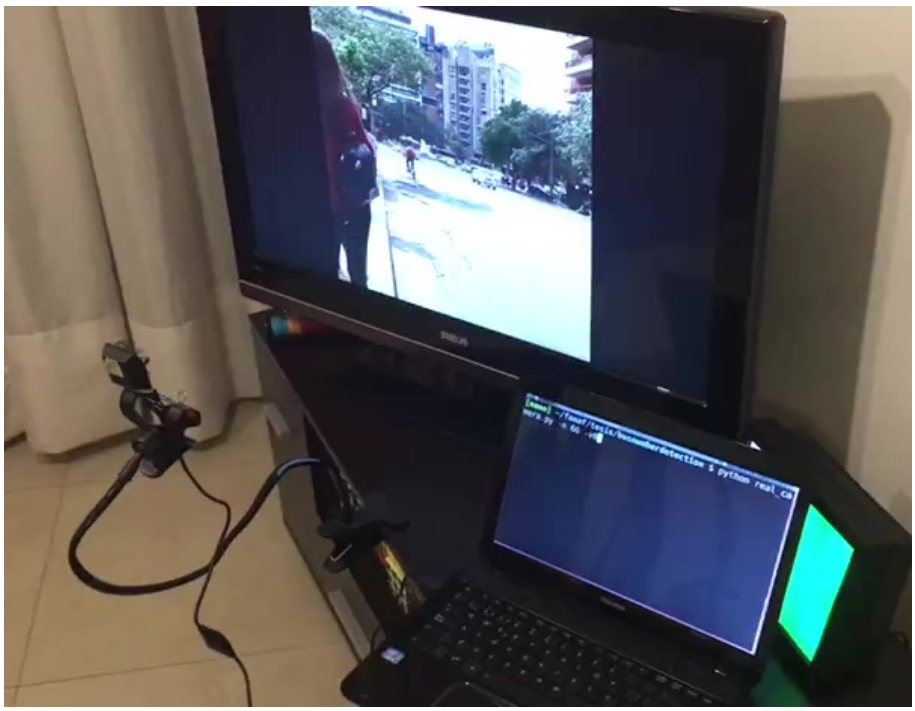
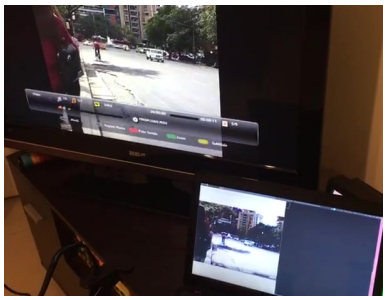


Figura 7.1: Recreación de parada virtual de autobuses.

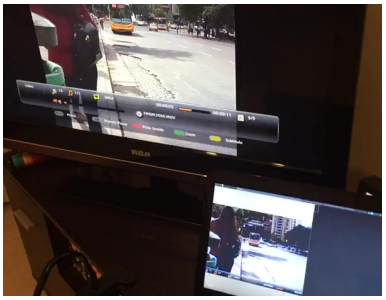
En este video, se comenzaba a capturar la llegada del autobús desde una distancia aproximada de 60 metros, para que luego de 17 segundos arribara a destino, al mismo tiempo que el sistema efectuaba el reconocimiento.

Cabe aclarar, que durante tal experimento, se probaron diferentes estrategias en relación al análisis de las detecciones que iban siendo realizadas; comenzando con un enfoque basal, en donde el procesamiento de reconocimiento de línea era disparado instantáneamente luego de cada detección de un autobús, a la codificación de un productor-consumidor de buffer acotado, para disparar tales análisis luego de k detecciones realizadas.

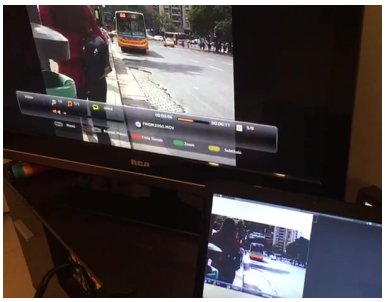
La Figura (7.2), muestra una secuencia de los instantes mas relevantes de tal experimento, en donde se puede observar, que luego de **3.18 segundos** de análisis, se da con el resultado correcto, con un porcentaje de certeza del **60 %**, utilizando la estrategia productor-consumidor de buffer $k = 3$.



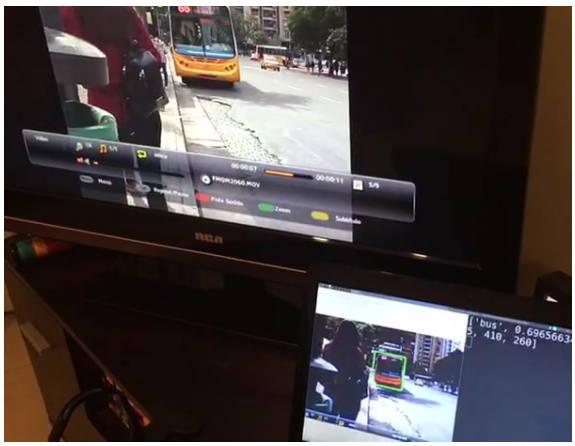
(a) Inicio de prueba.



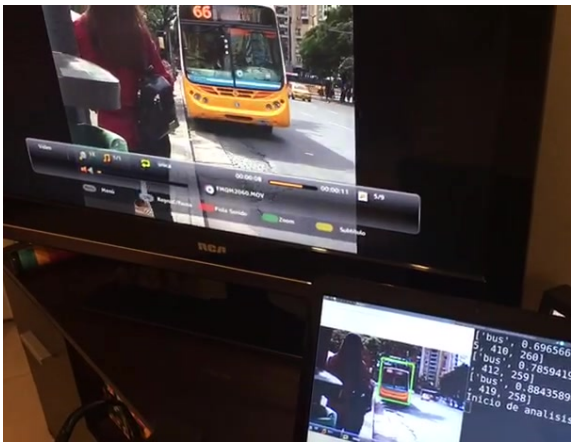
(b) ...aproximación.



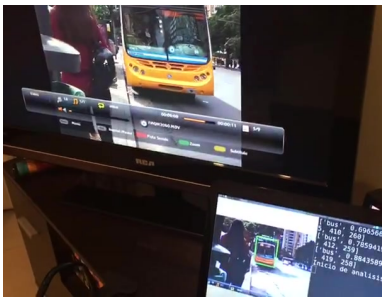
(c) ...aproximación.



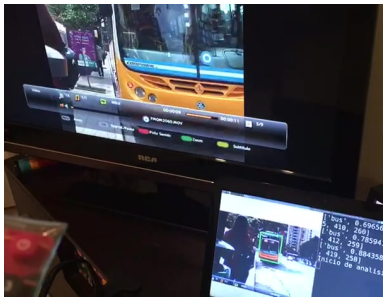
(d) 1^{er} detección.



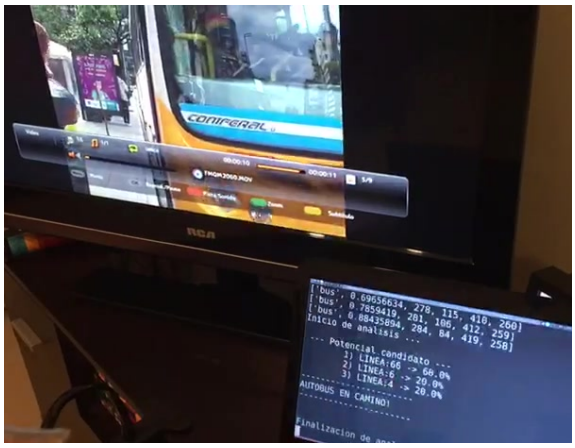
(e) 3^{er} detección e **Inició de análisis.**



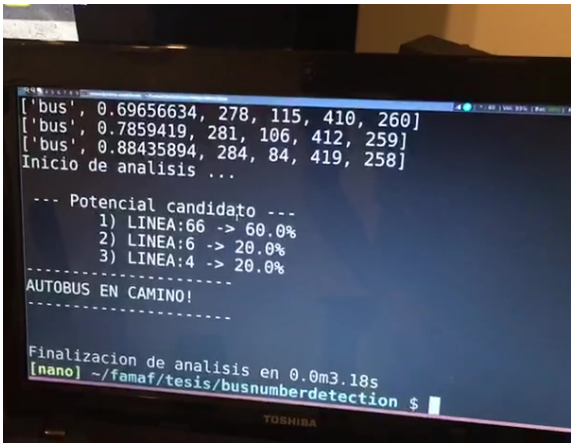
(f) ...análisis en proceso.



(g) ...análisis en proceso.



(h) **LINEA RECONOCIDA!!**



(i) Resultados.

Figura 7.2: Simulación de sistema en tiempo real.

Parte IV
Conclusiones finales

Para concluir, en una primera instancia se realizará un resumen del trabajo presentado en esta tesis. Luego, se describirán algunas de las limitaciones encontradas y posibles direcciones futuras.

Resumen

Con el fin de proponer una herramienta que potencialmente pudiese beneficiar a personas con algún tipo de impedimento visual, el desarrollo de esta tesis, tuvo desde los comienzos la intención de generar un algoritmo con la capacidad de automatizar la tarea de *«reconocer números de líneas urbanas»* en autobuses de la ciudad de Córdoba.

Comenzando con una base teórica de conceptos concernientes al campo de la inteligencia artificial, la visión por computadoras y las diferentes técnicas de aprendizaje profundo; se desarrollaron y describieron, de manera constructiva, los fundamentos y principios básicos que aproximaron a la tarea de reconocimiento de objetos en imágenes, y de texto en entornos naturales.

A partir de estos, en base a una etapa previa de investigación y con una idea clara de cuales eran las mejores opciones para llevar a cabo la propuesta, se realizó un primer planteamiento de la arquitectura del sistema, sobre la cual, y mediante el desarrollo de diferentes análisis y heurísticas, se seleccionarían los modelos y configuraciones de parámetros que mejor rendimiento otorgaran.

El algoritmo final estuvo constituido por dos etapas principales: una, de detección de autobuses, y otra, encargada de la detección y el reconocimiento de las líneas de los mismos. Para la primera, fueron evaluados un total de nueve variantes diferentes de tres modelos particulares de detección de objetos, de los cuales SSD-MobileNet 300 resulto cumplir con la mayoría de las necesidades requeridas. La segunda etapa, fue implementada mediante el modelo de detección EAST, y el motor de reconocimientos de caracteres OCR-Tesseract; los mismos fueron puestos a prueba sobre diversos escenarios, para seleccionar, de entre un abanico de opciones de transformaciones de imágenes y demás variables, las combinaciones de los parámetros que mayores porcentajes de reconocimientos positivos entregaran.

Con respecto a los resultados finales obtenidos, aunque podrían haber sido mejores, estuvieron por encima de lo esperado. Con una probabilidad de reconocimiento máximo del **62 %** en una imagen simple; sobre una ráfaga de imágenes, el sistema final, resulto capaz de realizar correctamente el reconocimiento de la línea de autobús en el **72 %** de los casos, a una velocidad media de **5 segundos**, reconociendo correctamente el número de la línea, en promedio, antes del $\mathbf{7}^{mo}$ frame analizado, y entregando una confianza en sus predicciones que oscila el **71 %.**

Limitaciones y direcciones futuras

Una de las mayores limitaciones encontradas en el sistema propuesto, estuvo relacionada a la dificultad experimentada para efectuar predicciones correctas en condiciones de poca iluminación o en horarios nocturnos. Si bien de noche, los números iluminados de las líneas sirven para ser visualizados de manera mas clara, los destellos que estos producen en los lentes de los dispositivos utilizados para capturar las imágenes, hacen de la detección y el reconocimiento, una muy difícil tarea. Por otra parte, aunque ningún sistema es 100 % efectivo, un porcentaje de reconocimiento del 72 %, significa que tendríamos grandes chances de tomarnos un autobús equivocado o directamente no tomarlo.

En virtud de mejorar el sistema, la aplicación de nuevas estrategias para lograr una reducción del conjunto de transformaciones de imágenes utilizadas, en base a nuevas propuestas de espacios de colores y métodos de segmentaciones menos generalistas; la adaptación e integración del algoritmo en arquitecturas GPU; y la incorporación de más rápidos y novedosos modelos al pipeline, tanto de detección de objetos como de texto; podrían dotar al sistema de un significativo aumento de velocidad y precisión de reconocimiento.

Parte V
Anexo

Referencia de índices de transformaciones de imágenes

id:**0**	→	'rgb',	id:**1**	→	'-rgb',
id:**2**	→	'r-rgb',	id:**3**	→	'-r-rgb',
id:**4**	→	'g-rgb',	id:**5**	→	'-g-rgb',
id:**6**	→	'b-rgb',	id:**7**	→	'-b-rgb',
id:**8**	→	'hls',	id:**9**	→	'-hls',
id:**10**	→	'h-hls',	id:**11**	→	'-h-hls',
id:**12**	→	'l-hls',	id:**13**	→	'-l-hls',
id:**14**	→	's-hls',	id:**15**	→	'-s-hls',
id:**16**	→	'hsv',	id:**17**	→	'-hsv',
id:**18**	→	's-hsv',	id:**19**	→	'-s-hsv',
id:**20**	→	'v-hsv',	id:**21**	→	'-v-hsv',
id:**22**	→	'lab',	id:**23**	→	'-lab',
id:**24**	→	'l-lab',	id:**25**	→	'-l-lab',
id:**26**	→	'a-lab',	id:**27**	→	'-a-lab',
id:**28**	→	'b-lab',	id:**29**	→	'-b-lab',
id:**30**	→	'yuv',	id:**31**	→	'-yuv',
id:**32**	→	'y-yuv',	id:**33**	→	'-y-yuv',
id:**34**	→	'u-yuv',	id:**35**	→	'-u-yuv',
id:**36**	→	'v-yuv',	id:**37**	→	'-v-yuv',
id:**38**	→	'YCrCb',	id:**39**	→	'-YCrCb',
id:**40**	→	'Cr-YCrCb',	id:**41**	→	'-Cr-YCrCb',
id:**42**	→	'Cb-YCrCb',	id:**43**	→	'-Cb-YCrCb',
id:**44**	→	'luv',	id:**45**	→	'-luv',
id:**46**	→	'u-luv',	id:**47**	→	'-u-luv',
id:**48**	→	'v-luv',	id:**49**	→	'-v-luv',
id:**50**	→	'edges',	id:**51**	→	'-edges'.

1^{er} enfoque: Muestra de 20 ROI's y sus mejores espacios de detección

(a) 04.png
('s-hsv', 0.11)
('hsv', 0.11)
('-h-hls', 0.11)
('-u-yuv', 0.11)

(b) **05.png**
Línea no reconocida

(c) **06.png**
Línea no reconocida

(d) 13.png
('-yuv', 0.25)
('lab', 0.25)
('-edges', 0.25)
('-b-rgb', 0.25)

(e) 14.png
('Cb-YCrCb', 0.33)
('-b-lab', 0.33)
('-v-luv', 0.33)

(a) 15.png
('hls', 1.0)

(b) 16.png
('-rgb', 0.14)
('-l-lab', 0.12)
('-g-rgb', 0.12)
('-y-yuv', 0.09)

(c) **17.png**
Línea no reconocida

(d) 19.png
('-v-hsv', 0.5)
('-v-luv', 0.5)

(e) 21.png
('v-luv', 1.0)

(a) 22.png
('-Cb-YCrCb', 0.22)
('-u-yuv', 0.22)
('s-hls', 0.11)
('b-lab', 0.11)

(b) 25.png
('-luv', 0.14)
('-g-rgb', 0.14)
('-b-lab', 0.1)
('-l-lab', 0.1)

(c) 30.png
('-s-hls', 0.17)
('-r-rgb', 0.15)
('-l-lab', 0.11)
('-v-hsv', 0.09)

(d) 31.png
('-rgb', 0.29)
('yuv', 0.14)
('lab', 0.14)
('-l-lab', 0.14)

(e) 32.png
('-yuv', 0.09)
('-v-hsv', 0.09)
('-rgb', 0.07)
('r-rgb', 0.07)

(a) 33.png
('u-yuv', 0.07)
('-hls', 0.06)
('-luv', 0.06)
('luv', 0.06)

(b) 34.png
('Cr-YCrCb', 0.25)
('v-yuv', 0.19)
('-y-yuv', 0.19)
('-s-hls', 0.12)

(c) 37.png
('-yuv', 0.22)
('l-lab', 0.17)
('-lab', 0.11)
('l-hls', 0.11)

(d) 41.png
('-YCrCb', 0.27)
('-yuv', 0.23)
('-lab', 0.18)
('-b-rgb', 0.14)

(e) 42.png
('-yuv', 0.23)
('-YCrCb', 0.23)
('-s-hls', 0.15)
('-hls', 0.15)

2^{do} enfoque: Muestra de 20 ROI's y sus mejores espacios de detección

(a) 04.png
('h-hls', 0.25)
('-l-hls', 0.08)
('luv', 0.08)
('-hls', 0.08)

(b) 05.png
('hls', 1.0)

(c) **06.png**
Línea no reconocida

(d) **13.png**
Línea no reconocida

(e) **14.png**
Línea no reconocida

(a) **15.png**
Línea no reconocida

(b) 16.png
('-rgb', 0.13)
('-l-lab', 0.11)
('-y-yuv', 0.1)
('-g-rgb', 0.08)

(c) **17.png**
Línea no reconocida

(d) 19.png
('-v-luv', 1.0)

(e) 21.png
('yuv', 0.38)
('v-luv', 0.38)
('v-hsv', 0.12)
('-v-luv', 0.12)

(a) 22.png
('v-luv', 0.6)
('yuv', 0.2)
('u-yuv', 0.2)

(b) 25.png
('-b-lab', 0.13)
('u-yuv', 0.13)
('-l-lab', 0.1)
('hls', 0.1)

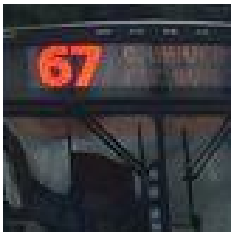
(c) 30.png
('edges', 0.14)
('-s-hls', 0.11)
('-l-hls', 0.08)
('-edges', 0.05)

(d) 31.png
('-l-lab', 0.23)
('-y-yuv', 0.23)
('-yuv', 0.15)
('-g-rgb', 0.15)

(e) 32.png
('-y-yuv', 0.11)
('-l-lab', 0.11)
('-YCrCb', 0.08)
('-r-rgb', 0.08)

(a) 33.png
('r-rgb', 0.08)
('b-lab', 0.07)
('-s-hsv', 0.06)
('luv', 0.05)

(b) 34.png
('s-hls', 0.25)
('u-luv', 0.25)
('hls', 0.25)
('hsv', 0.12)



(c) 37.png
('l-hls', 0.22)
('-yuv', 0.17)
('-lab', 0.17)
('g-rgb', 0.11)

(d) 41.png
('-hls', 0.24)
('-lab', 0.14)
('-s-hls', 0.14)
('-l-hls', 0.1)

(e) 42.png
('-s-hls', 0.5)
('-edges', 0.5)

Referencias

[1] Saleh Alghamdi and Ron [van Schyndel], et al. (2014). Accurate positioning using long range active RFID technology to assist visually impaired people. Elsevier, Journal of Network and Computer Applications **41**, 135-147. 6

[2] Ivanov R. (2012). Real-time GPS track simplification algorithm for outdoor navigation of visually impaired. Elsevier, Journal of Network and Computer Applications **35**(5), 1559-1567. 6

[3] Park S, Choi I-M, et al. (2014). A portable mid-range localization system using infrared LEDs for visually impaired people. Elsevier, Infrared Physics & Technology **67**, 583-589. 6

[4] Mpitziopoulos, Aristides & Konstantopoulos, et al. (2011). A pervasive assistive environment for visually impaired people using wireless sensor network infrastructure. J. Network and Computer Applications. **34**, 194-206, **10.1016/j.jnca.2010.07.017**. 6

[5] Betsworth L., Rajput N., Srivastava S., Jones M. (2013). Audvert: Using Spatial Audio to Gain a Sense of Place. In: Kotzé P., Marsden G., Lindgaard G., Wesson J., Winckler M. (eds) Human-Computer Interaction – INTERACT 2013. INTERACT 2013. Lecture Notes in Computer Science, vol 8120. Springer, Berlin, Heidelberg. 6

[6] Brown, D., et al., (2011). T.M.: Seeing with sound? exploring different characteristics of a visual-to-auditory sensory substitution device. Perception **40**(9), 1120-1135. 6

[7] V. Meas-Yedid, E. Glory, E. Morelon, et al. (2004). "Automatic color space selection for biological image segmentation". Proceedings of the 17th International Conference on Pattern Recognition **3**, 514-517, **10.1109/ICPR.2004.1334579**. 11

[8] Gupta Ankur, Chaudhary Ankit. (2014). Robust Skin Segmentation using Color Space Switching. Pattern Recognition and Image Analysis. **10.1134/S1054661815040033**. 11

[9] Da-Lei Song, Lei-Hua Ge, et al. (2010). "Illumination invariant color model selection based on genetic algorithm in robot soccer". The 2nd International Conference on Information Science and Engineering, Hangzhou, pp. 1245-1248, **10.1109/ICISE.2010.5689376**. 11

[10] Learn OpenCV. Color spaces in OpenCV. `https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/`. 11

[11] Stanford CS class. Spring 2018. CS231n Convolutional Neural Networks for Visual Recognition. `http://cs231n.github.io/`. 15

[12] Towards Data Science. Convolutional Neural Networks for Beginners. https://towardsdatascience.com/convolutional-neural-networks-for-beginners-practical-guide-with-python-and-keras-dc688ea90dca. 23

[13] Y. Lecun, L. Bottou, et al. (1998). "Gradient-based learning applied to document recognition". Proceedings of the IEEE, **86**(11), 2278-2324, **10.1109/5.726791**. 25

[14] Krizhevsky Alex, Sutskever Ilya, et al. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems **25**, **10.1145/3065386**. 25

[15] Szegedy, C., et al. (2014). W.L.: Going deeper with convolutions. CoRR **abs/1409.4842**. 25

[16] Simonyan Karen, Andrew Zisserman. (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". CoRR **abs/1409.1556**. 25

[17] Howard Andrew G. et al. (2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". ArXiv **abs/1704.04861**. 25

[18] J. Redmon, S. Divvala, et al. (2016). "You Only Look Once: Unified, Real-Time Object Detection". IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 779-788, **10.1109/CVPR.2016.91**. 37, 41

[19] Liu W. et al. (2016). SSD: Single Shot MultiBox Detector. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9905. Springer, Cham. 41, 42

[20] R. Girshick, J. Donahue, et al. (2014). "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 580-587, **10.1109/CVPR.2014.81**. 29, 36

[21] C. K. Ch'ng, C. S. Chan. (2017). "Total-Text: A Comprehensive Dataset for Scene Text Detection and Recognition". 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Kyoto, 2017, pp. 935-942, **10.1109/ICDAR.2017.157**. 33

[22] KDnuggets. Deep Learning for Object Detection: A Comprehensive Review. `https://www.kdnuggets.com/2017/10/deep-learning-object-detection-comprehensive-review.html`. 37, 38

[23] R. Girshick. (2015). "Fast R-CNN". IEEE International Conference on Computer Vision (ICCV), Santiago, 2015, pp. 1440-1448, **10.1109/ICCV.2015.169**. 36

[24] S. Ren, K. He, et al. (2017). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". IEEE Transactions on Pattern Analysis and Machine Intelligence, **39**(6), 1137-1149, **10.1109/TPAMI.2016.2577031**. 36

[25] Jifeng Dai, Yi Li, et al. (2016). "R-FCN: object detection via region-based fully convolutional networks". In Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16). Curran Associates Inc., Red Hook, NY, USA, 379–387. 37

[26] Redmon Joseph, Ali Farhadi. (2018). "YOLOv3: An Incremental Improvement". ArXiv **abs/1804.02767**. 37

[27] Redmon Joseph, Ali Farhadi. (2017). "YOLO9000: Better, Faster, Stronger". IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 6517-6525, **10.1109/CVPR.2017.690**. 38

[28] Andrew NG et al. Convolutionals Neural Networks, deeplearning.ai. Coursera. 40

[29] M. Liao, B. Shi, et al. (2018), "TextBoxes++: A Single-Shot Oriented Scene Text Detector". IEEE Transactions on Image Processing, **27**(8), 3676-3690, **10.1109/TIP.2018.2825107**. 43

[30] B. Shi, X. Bai, et al. (2017). "Detecting Oriented Text in Natural Images by Linking Segments". IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 3482-3490, **10.1109/CVPR.2017.371**. 43

[31] X. Zhou et al. (2017). "EAST: An Efficient and Accurate Scene Text Detector". IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 2642-2651, **10.1109/CVPR.2017.283**. 43, 44

[32] Y. Liu and L. Jin. (2017). "Deep Matching Prior Network: Toward Tighter Multi-oriented Text Detection". IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 3454-3461, 10.1109/CVPR.2017.368. 43

[33] Ronneberger O., Fischer P., Brox T. (2015) U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab N., Hornegger J., Wells W., Frangi A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science, vol 9351. Springer, Cham. 43

[34] Andrew NG. Machine learning, Universidad de Stanford. Coursera.

[35] PaperspaceBlog. How to implement a YOLO v3 object detector from scratch in PyTorch. `https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch`.

[36] DataCamp. A Beginner's Guide to Object Detection. `https://www.datacamp.com/community/tutorials/object-detection-guide`.