

TESIS DE MAESTRÍA EN INGENIERÍA

DETECCIÓN E IDENTIFICACIÓN DE CARACTERÍSTICAS DE VEHÍCULOS UTILIZANDO ALGORITMOS DE MACHINE LEARNING.

Ing. Enrique Nicanor Mariotti
Maestrando

Ing. José Reloso
Director

Dr. Felix Rojo Lapalma
Co-director

Miembros del Jurado

Dr. Eugenio Urdapilleta (Instituto Balseiro - Universidad Nacional de Cuyo)
Dr. Ariel Hernán Curiale (Instituto Balseiro - Universidad Nacional de Cuyo)
Dr. Enzo Ferrante (Universidad Nacional del Litoral)

Octubre de 2020

INVAP SE

Instituto Balseiro
Universidad Nacional de Cuyo
Comisión Nacional de Energía Atómica
Argentina

> *Yb*Tb*RNtY
Nb*R*b*B*NTtYPMr*DX*NM
H*MRtP9*F*R>Y
It*RNt*Y*F*XMA
F*F*H*YH*MMRtX
H*WYRNtQRQdA

Índice de abreviaturas

En orden de relevancia:

ALPR Automatic Licence Plate Recognition.

ALNR Automatic Licence Number Recognition.

ML Machine Learning.

DL Deep Learning.

OOP Object Oriented Paradigm.

GPU Graphical Processing Unit.

SVM Support Vector Machine.

CNN Convolutional Neural Network.

RCNN Recurrent Convolutional Neural Network.

YOLO You Only Look Once.

CRAFT Character Region Awareness For Text Detection.

LSTM Long Short Term Memory.

STN Spatial Transformer Network.

CTC Connectionist Temporal Classification.

SGD Stochastic Gradient Descent.

ROI Region Of Interest.

IOU Intersection Over Union.

NMS Non Maximal Supression.

XML EXtensible Markup Language.

LBP Local Binary Patterns.

GT Ground Truth.

TP True Positive.

TN True Negative.

FP False Positive.

FN False Negative.

Índice de contenidos

Índice de abreviaturas	iii
Índice de contenidos	v
Índice de figuras	vii
Índice de tablas	ix
Resumen	xi
Abstract	xiii
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	4
1.3. Trabajos relacionados	5
1.4. Estructura de tesis	6
2. Detección	7
2.1. YOLO	7
2.1.1. Introducción	7
2.1.2. Red convolucional	9
2.1.3. Salida a una escala dada	10
2.1.4. Anchors y bounding boxes	11
2.1.5. Detección multiescala	13
2.1.6. Función de costo	14
2.1.7. Inferencia y filtrado	16
2.2. Implementación	17
3. Reconocimiento	21
3.1. Reconocimiento	21
3.1.1. Red convolucional	22
3.1.2. Transformación espacial	23

3.1.3. Memorias de largo-corto plazo	25
3.1.4. Función de costo	27
3.2. Segmentación a nivel palabra	30
3.3. Implementación	32
4. Resultados y discusión	35
4.1. Dataset	35
4.2. Métricas	38
4.2.1. Predicciones	38
4.2.2. Métricas	40
4.2.3. Métrica de Levenshtein	41
4.3. Open ALPR	41
4.4. Resultados y discusión	44
5. Conclusiones y perspectivas	51
5.1. Conclusiones	51
5.2. Trabajos a futuro	53
5.2.1. Performance	53
5.2.2. Latencia	53
5.2.3. Mecanismos de atención	53
A. Esquema	55
B. Hardware	57
B.1. Hardware principal	57
B.2. Jetson Nano	57
Bibliografía	59

Índice de figuras

1.1.	Segmentación de matricula vehicular.	2
1.2.	La detección y reconocimiento de texto impreso o manuscrito en escenarios naturales es un problema estrechamente relacionado con los sistemas de ALPR	2
1.3.	Bloques funcionales de un sistema ALPR típico.	3
2.1.	Intersección sobre la Unión [28].	8
2.2.	Interpretación de la salida de la red [28]. Aquí, el número de predicciones por celda es 3.	11
2.3.	Predicción de la celda (c_x, c_y) en función de las dimensiones predefinidas de los <i>anchors</i> y de una de las predicciones de la red [1].	12
2.4.	Salida multiescala de YOLOv3 [28].	13
2.5.	Imagen ilustrativa de un numero arbitrario de predicciones sin utilidad entregadas por un detector multiclasa [28].	17
2.6.	Imagen ilustrativa de la aplicación del filtrado NMS [28].	17
2.7.	Ejemplo de imagen usada en el entrenamiento de la red [2].	18
2.8.	Ejemplos contenidos en el <i>dataset</i> de ImageNet [3].	19
3.1.	Resultado de una transformación espacial STN [4] sobre una imagen del <i>dataset</i> de reconocimiento de señales de trafico alemanas [5].	23
3.2.	Transformación espacial sobre algunos ejemplos del conjunto MNIST [6].	24
3.3.	Bloques funcionales dentro de una capa de STN [6].	24
3.4.	Desglose del funcionamiento de una RNN [7].	25
3.5.	Bloques funcionales dentro de una compuerta LSTM . Modificada a partir de [8].	26
3.6.	Bloques funcionales dentro de la LSTM de acuerdo a la Ec. 3.2 y la Ec. 3.3 [9].	27
3.7.	Imagen generica de texto impreso de origen sintetico [10].	28
3.8.	Una alineación posible de la secuencia $\mathbf{Y} = [h, e, l, l, o]$ [11].	29
3.9.	Funcionamiento de la red entrenada con una función de costo CTC [11].	30
3.10.	Ejemplo de segmentación a nivel palabra de CRAFT	31

3.11. Ejemplo de segmentación a nivel palabra de CRAFT	32
3.12. Ejemplo de imágenes contenidas en <i>dataset</i> de Synthetic Word Data-set [10].	33
4.1. Imagen original sin modificar.	36
4.2. Imagen modificada con Albumentations [12].	37
4.3. Imagen modificada con Albumentations [12].	37
4.4. Imagen modificada con Albumentations [12].	37
4.5. Ejemplo de clasificación en categorías no triviales [13].	38
4.6. Ejemplo de una situación que da origen a un TP , un FP y un FN . . .	39
4.7. Intuición detrás de la métrica de <i>Levenshtein</i> . Las operaciones aceptadas son la inserción , eliminación o la sustitución de caracteres individuales.	41
4.8. Ejemplo de resultados de la API de OpenALPR	43
4.9. Algunas imágenes de <i>dataset</i> lluvioso pueden ser particularmente difíciles de leer.	45
4.10. Reconocimiento correcto con perspectiva izquierda.	46
4.11. Reconocimiento incorrecto con perspectiva derecha.	46
4.12. Detección y reconocimiento correcto en una patente en formato de la República Argentina.	47
4.13. Detección y reconocimiento correcto de un letrero en un escenario natural acotado. La predicción no ordena las sentencias en el caso mas general donde no se tiene una matrícula.	47
4.14. Escena con oclusión casi total del vehículo. Formato no presente en el conjunto de entrenamiento.	48
4.15. Escena con oclusión casi total del vehículo y oclusión parcial de la matrícula. Formato no presente en el conjunto de entrenamiento.	48
4.16. Reconocimiento correcto de una sentencia individual.	49
4.17. Reconocimiento correcto de una sentencia individual. Las sub-sentencias no son leídas.	49
4.18. Reconocimiento correcto de una sentencia individual. Sentencias no relevantes como esta son normalmente filtradas antes de entrar en la red de reconocimiento.	49
A.1. Diagrama de bloques del modelo desarrollado.	55
B.1. NVIDIA Jetson Nano [14].	58
B.2. Especificaciones del hardware de Jetson Nano [14].	58

Índice de tablas

2.1. CNN detrás del procesamiento de YOLO [1].	9
3.1. Arquitectura empleada.	22
4.1. Comparación de resultados. (*) Promedio de la similitud normalizada de <i>Levenshtein</i>	44

Resumen

En esta tesis de maestría en ciencias de la ingeniería se propone una implementación modular para resolver el problema del reconocimiento automático de patentes vehiculares (**ALPR**) basado en un esquema *detección-detección-reconocimiento*, usando íntegramente *deep learning*. Se pretende emplear esta implementación para aplicaciones prácticas que involucran control vehicular visual de cualquier tipo.

A lo largo de este trabajo se empleó un enfoque modular, dividiendo el objetivo principal en etapas progresivas. Se desarrollaron módulos individuales, que implementan una arquitectura de clases basada en el paradigma orientado a objetos (**OOP**), lo que los hace genéricos y fácilmente modificables. A su vez, los módulos u objetos individuales pueden ser usados con otros propósitos, como ser la lectura de texto impreso o señales de tránsito en imágenes de tráfico urbano.

Durante la realización de esta tesis se priorizó el uso de *transfer learning* en redes profundas convolucionales (**CNN**), con el objetivo de adaptarse a una disponibilidad de recursos limitada y para disminuir considerablemente los tiempos asociados al entrenamiento.

La etapa primaria de segmentación fue basada en un esquema de detección de código libre denominado usualmente **YOLOv3**, el cual fue re-entrenado con imágenes naturales de vehículos perteneciente a un *dataset* de acceso público. La etapa de reconocimiento de texto se lleva a cabo mediante una red recurrente-convolucional (**RCNN**) que realiza el reconocimiento de la sentencia. Finalmente, el nexo entre estas dos redes está dado por un esquema de uso comercial denominado **CRAFT**.

En este escrito se presentan todos los detalles de la implementación de dicho esquema, y por sobre el final del mismo, se evalúa cuantitativamente la *performance* del sistema usando diferentes métricas contra **OpenALPR**, un software de uso extendido en el área de reconocimiento de patentes vehiculares.

Palabras clave: RECONOCIMIENTO AUTOMÁTICO DE PATENTES VEHICULARES, APRENDIZAJE AUTOMÁTICO, APRENDIZAJE PROFUNDO, PROGRAMACIÓN ORIENTADA A OBJETOS

Abstract

In this master of science in engineering thesis, a modular implementation is proposed to solve the problem of automatic licence plate recognition (**ALPR**), based on a *detection-detection-recognition* scheme, done entirely using *deep learning*. The scheme here presented is intended to be used for practical applications involving visual vehicle control of any kind.

A modular approach was used throughout this work, dividing the main objective into progressive stages. Individual modules were developed, which implement a class architecture based on the object-oriented paradigm (**OOP**) that makes them generic and easily modifiable. In turn, the modules or individual objects can be used for other purposes, such as reading printed text or traffic signs in images of urban traffic.

During the completion of this thesis, the use of *transfer learning* on deep convolutional networks (**CNN**) was prioritized, in order to adapt to a limited availability of resources and to reduce considerably the time associated with training.

The primary segmentation stage was based on an open source detection scheme usually named **YOLOv3**, which was re-trained with natural images of vehicles belonging to a public access dataset. The text recognition stage is carried out by means of a recurrent-convolutional network (**RCNN**) that performs the recognition of the sentence. Finally, the link between these two networks is given by a commercial detection scheme called **CRAFT**.

In this writing, all the details of the scheme implementation are presented, and near the end of it, the performance of the system is quantitatively evaluated using different metrics against **OpenALPR**, a software widely used in the field of automatic licence plate recognition.

Keywords: AUTOMATIC LICENSE PLATE RECOGNITION, MACHINE LEARNING, DEEP LEARNING, OBJECT ORIENTED PROGRAMMING

Capítulo 1

Introducción

En este capítulo se presenta la problemática a abordar, los objetivos del trabajo y se realiza una breve reseña bibliográfica al respecto. Sobre el final del capítulo se presentan a grandes rasgos las herramientas individuales empleadas para la conformación de la solución propuesta y se expone la estructura general de la tesis.

1.1. Motivación

*Automatic Number Plate Recognition (**ALNR**) o *Automatic License Plate Recognition* (**ALPR**) es la denominación en inglés del reconocimiento automático de patentes, que se refiere a la tecnología de procesamiento de imágenes la cual permite la extracción de información de la patente vehicular a partir de una imagen o una secuencia de imágenes. La Fig. A.1 muestra un ejemplo de aplicación.*

Desde su primera aparición en 1976, cuando se desarrolló por la Policía Científica del Reino Unido, este tipo de soluciones ha ganado progresivamente más popularidad a la par de las mejoras en tecnología de captura de imágenes digitales, y sobre todo, con el aumento en la velocidad de procesamiento de las computadoras.

La tecnología **ALPR** ha sido durante las últimas décadas un tema frecuente de investigación debido a las muchas aplicaciones prácticas que conlleva, entre ellas, el control vehicular por parte de las fuerzas de la ley, el sondeo de información sobre lotes vehiculares vastos y el control de acceso a lugares privados. No de menor importancia es que se trata de un problema relativamente acotado, pero que permite la transferencia tecnológica a problemas de detección y reconocimiento más complejos, como puede ser

el procesamiento del lenguaje natural a partir de sus caracteres ópticos, por ejemplo, usando imágenes digitales de texto manuscrito o impreso. La Fig. 1.2 ilustra este tipo de aplicación.



Figura 1.1: Segmentación de matrícula vehicular.



Figura 1.2: La detección y reconocimiento de texto impreso o manuscrito en escenarios naturales es un problema estrechamente relacionado con los sistemas de **ALPR**.

Existen muchos trabajos e investigaciones donde se aborda esta problemática desde distintos puntos de vistas, aplicando diferentes metodologías y herramientas para tratar de resolver este problema específico. Típicamente un sistema **ALPR** consiste de tres etapas: (1) detección de la región que contiene la patente, (2) segmentación de los caracteres o secuencia de caracteres, y finalmente, (3) reconocimiento de los caracteres. Se sigue del esquema de la Fig. 1.3 que las etapas de detección iniciales necesitan de la mayor exactitud posible, ya que las etapas subsiguientes posiblemente fallen debido a un error en una de las etapas previas.

La gran mayoría de las herramientas disponibles tienen que ser adaptadas (o entrenadas) para reconocer las matrículas de diferentes países, provincias o estados. Estas placas suelen contener diferentes colores, estar escritas en diferentes idiomas y usar diferentes fuentes, ademas, muchas veces el tamaño de los dígitos puede no ser uniforme. Algunas placas pueden tener un solo color de fondo y otras tienen imágenes de fondo.

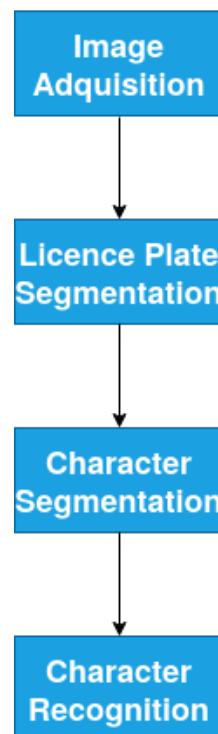


Figura 1.3: Bloques funcionales de un sistema **ALPR** típico.

A conocimiento del autor, la mayoría de los trabajos disponibles en la bibliografía se centran en etapas individuales de la Fig. 1.3 y no siempre abordan la resolución integral del problema de **ALPR**.

En general, la *performance* de estas herramientas suele ser fuertemente dependiente de factores que agregan complejidad al problema, como la calidad de las imágenes adquiridas, la velocidad del vehículo al momento de la captura, el ángulo de la cámara,

el trasfondo de la escena y las condiciones de iluminación, así como los problemas indirectos asociados a los anteriores.

A pesar de su estudio extendido y de la variedad de metodologías empleadas para su abordaje, muchas soluciones no son lo suficientemente robustas para abordar la variedad de escenarios presentes en las aplicaciones del mundo real, y el problema permanece aun como una tarea desafiante.

1.2. Objetivos

Según lo expuesto en la Sección 1.1 y con el objetivo de integrar abordajes al problema que puedan aportar robustez a un sistema **ALPR**, se explora en el curso de este trabajo una solución integral al problema usando *Deep Learning* (**DL**).

Los algoritmos de **DL** sobresalen en un escenario donde muchas aplicaciones de procesamiento de imágenes han obtenido un incremente significativo de *performance*, principalmente debido a la disponibilidad de una gran cantidad de datos anotados y recursos físicos que permiten el procesamiento en paralelo.

Este tipo de soluciones, que proveen una representación multi-nivel de los datos, prevalecen en aplicaciones complejas donde es necesario trabajar con datos no estructurados (categoría en la que recaen las imágenes “naturales”) y es necesario la extracción de patrones complejos de información. En estos casos, la interpretabilidad del resultado no es un factor preponderante y juega un rol secundario.

A pesar del notorio progreso de estas estrategias, todavía existe una notable demanda de bases de datos con anotaciones matriculares para aplicaciones de **ALPR**. Ya que el tamaño del *dataset* de entrenamiento es determinante en la robustez del resultado, la escasez de datos se presenta como una limitación seria. Con el objetivo de adaptarse a una disponibilidad de datos y recursos limitada, y para disminuir considerablemente los tiempos asociados al entrenamiento, proponemos como uno de los objetivos de este trabajo priorizar el uso de *transfer learning* en redes profundas.

Otra de las condiciones que imponemos en principio sobre la solución propuesta, es que consista de una secuencia de etapas modulares, que dividan el objetivo principal en etapas progresivas. Para ellos se desarrollaron módulos individuales, que implementan una arquitectura de clases basada en el paradigma orientado a objetos (**OOP**), lo que los hace genéricos y fácilmente modificables.

Tomaremos como caso de estudio el reconocimiento de la patente única del Mercosur, con especial atención de poder extender el uso de este sistema a otros formatos a futuro. A su vez, los módulos u objetos individuales pueden ser usados con otros propósitos, como ser la lectura de texto impreso o señales de tránsito en imágenes de tráfico urbano. Incluirímos algunos ejemplos de estas aplicaciones en el Capítulo 4.

Finalmente, el objetivo final de este trabajo es poder comparar nuestro modelo con

los resultados del software de usos comercial mas extendido en el mercado, llamado OpenALPR [15].

1.3. Trabajos relacionados

Se presenta a continuación una breve reseña de los trabajos que utilizan **DL** en el contexto de la resolución de un problema de **ALPR**, a la vez que una breve explicación de las etapas individuales de cada esquema presentado.

Estudios de otras técnicas relevantes de procesamiento de imágenes que no hacen uso de *Machine Learning (ML)* en ninguna de sus formas, aplicadas a **ALPR** pueden verse en [16, 17].

Cabe notar que la mayoría de las publicaciones científicas abordaran muchas veces únicamente etapas individuales del problema (e.g. la detección de la matrícula) o muestran los resultados obtenidos sobre *datasets* que no representan con fidelidad escenarios reales, haciendo que la evaluación integral de los diferentes esquemas sea relativamente difícil.

- **Segmentación de patente:**

Muchos autores han empleado redes convolucionales (**CNN**: *Convolutional Neural Networks*) para detección de objetos en la etapa de detección de la patente vehicular. *Montazzolli et al.* [18] usaron una única red **CNN** para la detección simultánea de la vista frontal del vehículo y su patente. *Hsu et al.* [19] adaptaron la arquitectura de la red anteriormente nombrada exclusivamente para la detección de patentes, y obtuvieron resultados con mejoras sustanciales. *Rafique et al.* [20] aplicaron *Support Vector Machine (SVM)* y **CNN** basadas en región para la detección de la matrícula, obteniendo buenos resultados para aplicaciones en tiempo real.

- **Segmentación de caracteres:**

Muchos sistemas de **ALPR** basados en **DL** usualmente realizan la tarea de segmentación y reconocimiento de caracteres simultáneamente.

Li et al. [21] entrenaron una **CNN** en base a caracteres recortados de imágenes genéricas de texto impreso, y la emplearon para realizar una detección a nivel de carácter único dentro de la patente, obteniendo resultados prometedores. *Montazzolli et al.* [18] propusieron una **CNN** para segmentar y reconocer caracteres dentro de una región de interés (**ROI**) que contuviera una imagen de la patente. *Laroca et al.* [22] replicaron estos resultados, pero usando redes independientes para la segmentación y el reconocimiento, obteniendo una mejor *performance* para aplicaciones en tiempo real. *Bulan et al.* [23] obtuvieron una alta exactitud

realizando la segmentación y el reconocimiento de caracteres de manera conjunta empleando un modelo oculto de Markov.

- **Reconocimiento de caracteres:**

Menotti et al. [24] propusieron el uso de **CNN** aleatorias para la extracción de *features* para el reconocimiento de caracteres, obteniendo mejor rendimiento que aprendiendo los pesos de los filtros mediante *back-propagation*.

Li et al. [21] propusieron realizar el reconocimiento de caracteres tratando el problema como un problema de clasificación de elementos dentro de una secuencia. De este modo, se empleo una red neuronal recurrente (**RNN**: *Recurrent Neural Networks*) con una función de costo *Connectionist Temporal Classification (CTC)* para clasificar los caracteres dentro de la secuencia de texto, sin necesidad de efectuar la segmentación a nivel individual. Mas tarde, los autores propusieron que la localización de la patente podría ser realizada en simultaneo con el reconocimiento de los caracteres con una única corrida hacia adelante de la red. En este modelo, muchos de los *features* convolucionales se podrían compartir de manera de reducir considerablemente el tamaño del modelo.

1.4. Estructura de tesis

El segundo capitulo de este trabajo está dedicado al esquema de detección empleado para localizar y segmentar la patente a partir de una imagen o serie de imágenes de entrada. Para esto, se uso un algoritmo de detección de objetos de redes neuronales convolucionales, basado en una implementación de **YOLO** (del ingles, *You Only Look Once*) [1, 25, 26], donde la salida tiene codificada la información sobre la localización y clasificación de los objetos dentro de la imagen.

En el tercer capitulo se presentará el modelo completo para el reconocimiento de sentencias, el cual emplea un pre-procesamiento convolucional sobre la imagen, para luego alimentar con esta información la entrada de capas recurrentes bi-direccionales, con compuertas de memoria, que realizan el reconocimiento de los elementos de la sentencia. También proponemos la utilización de un esquema de detección de palabras externo, para sobrellevar el problema de la falta de datos de entrenamiento.

El cuarto capitulo está dedicado a los resultados de algunos casos de validación simples y a evaluar la *performance* general del método de manera cuantitativa. La conclusión y las futuras perspectivas para este trabajo conforman la quinta parte del informe.

Finalmente, sobre el primer apéndice de este trabajo se muestra un diagrama esquemático del funcionamiento del modelo completo. Seguidamente, se pueden encontrar algunos detalles del *hardware* sobre el cual se realizó la implementación del sistema.

Capítulo 2

Detección

En este capítulo se presenta el esquema de detección empleado para localizar y segmentar la patente a partir de la imagen de entrada. Se realiza un breve repaso sobre las características mas importantes del esquema empleado. Sobre el final del capítulo se proveen detalles de la implementación realizada.

2.1. YOLO

2.1.1. Introducción

You Only Look Once (YOLO) [1, 25, 26] es un algoritmo de detección de objetos que usa redes neuronales convolucionales como herramienta principal. A pesar de no ser el esquema de detección de objetos mas preciso, es una buena opción de compromiso entre exactitud y velocidad de detección.

En contraste con los algoritmos de clasificación, un algoritmo de detección no solo debe predecir la clase a la que el objeto pertenece sino también la localización del mismo dentro de la imagen, siendo capaz de detectar múltiples objetos dentro de una imagen. Cabe destacar que la arquitectura de **YOLO** no entra dentro de la categoría de segmentación semántica, donde se asocia una etiqueta o categoría a cada píxel presente en una imagen. Al contrario, las predicciones de la red son directamente las regiones de interés que localizan el objeto dentro de la imagen.

YOLO aplica solo una única red neuronal a la imagen. A diferencia de los detectores convolucionales basados en región, **YOLO** usa la imagen completa de entrada como *input* para hacer predicciones, por lo que implícitamente codifica información contextual

de clases así como de su aparición en diferentes escenarios.

El algoritmo “divide” la imagen en una cuadrícula uniforme de $S \times S$. Si el centro de un objeto cae dentro de una de las celdas de esta cuadrícula, entonces esa celda es la encargada de detectar este objeto. Esta celda predice B locaciones del objeto en forma de tuplas (x, y, w, h) , un nivel de confiabilidad de que efectivamente haya un objeto asociado a la predicción y probabilidades condicionales de que ese objeto pertenezca a una clase dada $\text{Pr}(\text{Class}_i|\text{Object})$.

La puntuación de confiabilidad del objeto refleja que tan buena infiere el modelo que es su propia predicción. Formalmente se define a esta confiabilidad como:

$$\text{Pr}(\text{Object}) \times \text{IOU} \quad (2.1)$$

En la Ec. 2.1, Pr es igual a 1 si existe un objeto dentro de la celda e igual a 0 en el caso contrario. Aquí, **IOU** es la *Intersection Over Union* entre la predicción y la anotación del objeto.

Una explicación gráfica del cálculo de la **IOU** puede verse en la Fig. 2.1. Se sigue que la definición de **IOU** es análoga al índice de **Jaccard**, el cual mide el grado de similitud entre dos conjuntos y se define como la cardinalidad de la intersección de ambos conjuntos dividida por la cardinalidad de su unión.

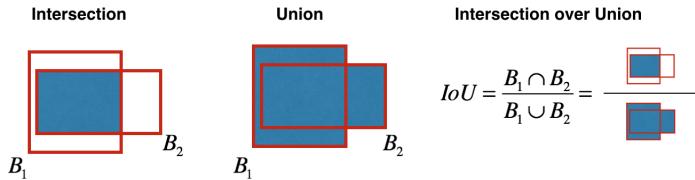


Figura 2.1: Intersección sobre la Unión [28].

En la versión oficial más actual de la red, denominada **YOLOv3** [1], se usan *priors* de forma para sobrellevar el problema de que haya más de un objeto dentro de una celda del entramado $S \times S$. Es decir se necesitan tantos *priors* (también llamados *anchors*) como predicciones posibles B de la celda.

Además, en esta última versión las predicciones se realizan a diferentes escalas simultáneamente para poder incluir mayor información semántica relevante presente en etapas anteriores en la red convolucional encargada de extraer los *features*, y a su vez, facilitar la detección de objetos de diferentes dimensiones dentro de la imagen.

Al momento de evaluar el funcionamiento de la red, el modelo designa a cada predicción con un confianza específica de la clase, la cual se calcula según la Ec. 2.2, multiplicando la confiabilidad de que haya un objeto dentro de la celda con su respectiva probabilidad condicional de clase. Esta puntuación empleada por los autores [25] evalúa la probabilidad que una clase aparezca dentro de la predicción y a su vez que tan precisa es la localización de la predicción.

$$Pr(Class_i|Object) \times Pr(Object) \times IoU_{pred}^{truth} = Pr(Class_i) \times IoU_{pred}^{truth} \quad (2.2)$$

Finalmente terminamos esta sección introductoria mencionando que el modelo usa un algoritmo de supresión y filtrado que ayuda a eliminar predicciones que no son relevantes en la etapa de inferencia.

A lo largo de este capítulo explicaremos con más detalles los puntos hasta aquí mencionados.

2.1.2. Red convolucional

YOLO solo hace uso de capas convolucionales en su estructura. En [1] los autores presentan una arquitectura de red para extracción de *features* llamada **Darknet-53**. Como su nombre indica, contiene 53 capas convolucionales, cada una seguida de una capa de *Batch Normalization* y una función de activación del tipo **Leaky ReLU**. Ninguna capa de *pooling* está presente en la arquitectura de la red, en vez de eso, una capa convolucional con *stride* de 2 es usada para disminuir la dimensión del mapa de *features*. Esto ayuda a prevenir la pérdida de *features* en las primeras capas, normalmente atribuida al *pooling* [25].

La estructura de la red puede verse en la Tabla 2.1.

Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x	32	1×1	
Convolutional	64	3×3	
	Residual		128×128
	Convolutional	$3 \times 3 / 2$	64×64
	Convolutional	1×1	
2x	128	3×3	
	Residual		64×64
	Convolutional	$3 \times 3 / 2$	32×32
	Convolutional	1×1	
8x	256	3×3	
	Residual		32×32
	Convolutional	$3 \times 3 / 2$	16×16
	Convolutional	1×1	
8x	512	3×3	
	Residual		16×16
	Convolutional	$3 \times 3 / 2$	8×8
	Convolutional	1×1	
4x	1024	3×3	
	Residual		8×8
	Avgpool		Global
	Connected		1000
	Softmax		

Tabla 2.1: CNN detrás del procesamiento de **YOLO** [1].

Aquí, para permitir el procesamiento paralelo de las imágenes en batches dentro de la GPU, las dimensiones de las imágenes de entrada se fijaron en $(416, 416, 3)$.

Al igual que en la arquitectura típica de **ResNet** [27] en la Tabla 2.1 hay conexiones “salteadas” lo que ayuda a mitigar problemas asociados al gradiente disminuyendo en las capas mas cercanas a la entrada.

La arquitectura de la red esta pensada para funcionar como un extractor de *features*. Por este motivo, en primera instancia las ultimas capas de la red funcionan como un clasificador. Toda la red esta entrenada como un clasificador sobre el dataset **ImageNet 1000** [3] y los datos del entrenamiento pueden encontrarse en [1, 25, 26].

En segunda instancia, el bloque de clasificación (es decir, la capa de *Global Average Pooling*, la ultima capa densa y su activación), son removidas y reemplazadas por un bloque de detección. Véase la Sección 2.1.5 para mas detalles de como el bloque de detección incorpora información de diferentes escalas.

2.1.3. Salida a una escala dada

La salida de **YOLO** es un tensor tridimensional. Para interpretar esta salida, se puede pensar que la red “divide” la imagen de entrada en una cuadrícula de $S \times S$ celdas.

Cada celda de la cuadrícula predice un numero fijo de *bounding boxes*, designado como B . El numero de predicciones a una escala dada depende del numero de *anchors* empleado a esa escala. Véase la Sección 2.1.4 al respecto.

A su vez, cada *bounding box* esta representada por los elementos: $(b_x, b_y, b_h, b_w, p_o, p_i, \dots, p_c)$. Aquí, b_x, b_y, b_h, b_w versan sobre la ubicación del centroide, como *offset* de la celda que realiza la predicción, y del alto y ancho de la detección, normalizados segúin el alto y ancho de la celda, o a veces, segúin el alto y ancho de la imagen.

Por otro lado, p_o es la probabilidad que otorga la red a que haya un objeto dentro de la *bounding box* predicha (a veces llamada *objectness*).

Finalmente, p_i son las probabilidades condicionales de clases, es decir la probabilidad de que el objeto detectado corresponda a cierta clase. En el caso de una sola clase a detectar (patente), existe un único valor p_1 , aunque en el caso mas general existen tantos valores p_1, \dots, p_c como las C clases detectables.

Finalmente, la red tiene una salida de dimension: $S, S, B \times (5 + C)$. La Fig. 2.2 muestra una representación de la salida de la red.

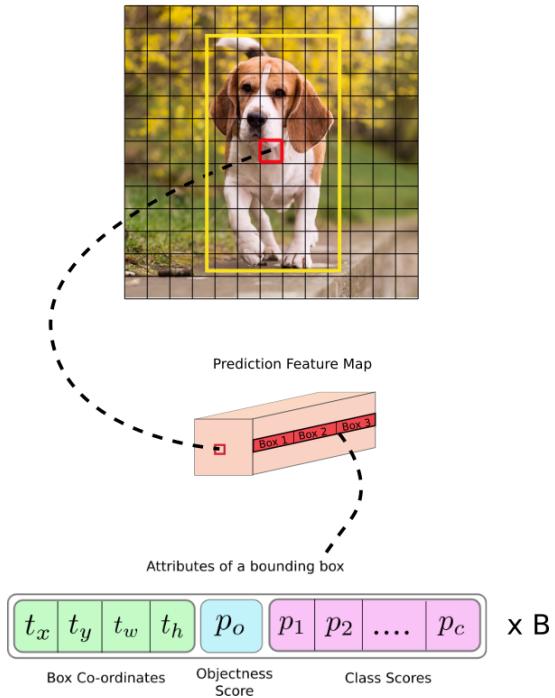


Figura 2.2: Interpretación de la salida de la red [28]. Aquí, el número de predicciones por celda es 3.

2.1.4. Anchors y bounding boxes

En la mayoría de los problemas prácticos que involucran imágenes naturales, los objetos tienen dimensiones que siguen patrones muy definidos para las diferentes clases. En la práctica, predecir el ancho y alto de estas regiones partiendo de valores totalmente arbitrarios puede llevar a gradientes inestables durante las etapas tempranas de entrenamiento. Además, desperdiciar información disponible *a priori* sobre la forma de los objetos conlleva un mayor consumo de recursos y tiempo de entrenamiento. En vez de esto, muchos detectores modernos no-semánticos basados en región predicen valores en espacios logarítmicos que luego deben ser transformados, o simplemente predicen corrimientos (*offsets*) sobre dimensiones predefinidas.

En lo que concierne particularmente a **YOLO**, queremos además que la red sea capaz de diferenciar cuando hay dos (o más) objetos en la misma celda. La solución utilizada por la red es emplear *priors* de forma, llamados *anchors*, y asignar una predicción a cada *anchor*. Este número es ajustable, y varía según la versión de la implementación elegida [1, 25, 26].

Es importante que en el proceso de codificación de las anotaciones (o *ground truth*, **GT**) se asigne un objeto a uno de los *anchors* según algún criterio particular. En la implementación de **YOLOv3** se asigna el **GT** al *anchor* con el cual la **IOU** es mayor.

Los *anchors* pueden setearse manualmente o calcularse mediante algún algoritmo de *clustering*, como *k-means* sobre los valores de **GT** del *dataset* empleado. El ancho y

alto de la región se predice como *offsets* sobre los *anchors* (centroides del algoritmo de *clustering*).

Formalmente:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \tag{2.3}$$

En la Ec. 2.3, b_x, b_y, b_h, b_w son los centros x, y de coordenadas y el alto y ancho de nuestra predicción, respectivamente.

t_x, t_y, t_h, t_w representan una predicción de la salida de la red. A estos 4 valores se suma la confiabilidad de *objectness* $\sigma(t_o)$ y las probabilidades de clases, que no son de importancia aquí.

c_x, c_y son los indices de la celda respecto al origen de la imagen. p_h, p_w son las dimensiones de los *anchors*. Véase la Fig. 2.3 como referencia.

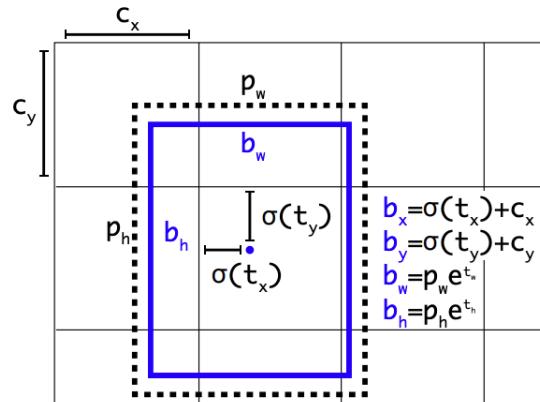


Figura 2.3: Predicción de la celda (c_x, c_y) en función de las dimensiones predefinidas de los *anchors* y de una de las predicciones de la red [1].

En un esquema **YOLO** las celdas de la cuadrícula son todas iguales. Las predicciones del centroide t_x, t_y están dadas relativas a la esquina superior-izquierda de la celda que realiza la predicción del objeto y, a su vez, su valor está normalizado por las dimensiones de la celda.

Además, aquí $\sigma(t_i)$ es la función sigmoidal. $\sigma(t_i)$ tiene el objetivo de mantener el valor del centroide en el intervalo $[0, 1]$, es decir, dentro de la celda que lo origina, y así evitar que el mismo sea mal interpretado como proveniente de una celda adyacente. Del mismo modo, la exponencial o transformación en el espacio logaritmo se hace para garantizar que los valores de alto y ancho sean siempre positivos.

2.1.5. Detección multiescala

En particular, **YOLOv3** está diseñado como un detector multiescala, para ello se toman *features* de los últimos 3 bloques residuales, y se usan en la detección subsecuente. Las escalas resultantes son 13×13 , 26×26 y 52×52 , como se muestra la Fig. 2.4.

La salida de *features* de la red de la Sección 2.1.3 alimenta el cabezal de detección de la escala 13×13 , el que consta de 2 bloques: (1) 1×1 *kernel* y 512 *features* y (2) 3×3 *kernel* y 1024 *features*. Estos 2 bloques se intercalan 3 veces. El bloque final consta de 1×1 *kernel* y la cantidad de *features* que llevan a la salida a su dimensión final.

Para construir el cabezal de detección de la escala 26×26 se toma un mapa de *features* anterior de la red convolucional y se concatena con los *features* de 2 capas anteriores de la salida del detector 13×13 , los cuales pasan por un *upsampling* previamente. Esta salida pasa nuevamente por el cabezal de detección previamente detallado para generar la segunda salida de la red.

El procedimiento se repite una vez más finalmente para generar la escala más fina 52×52 . Esta última, se beneficia de todos cálculos previos, así como de la información de un mapa de *features* más “profundo” de la red convolucional. Según el autor [1], este procedimiento permite obtener mayor información semántica relevante presente en etapas anteriores en la red.

NOTA: Las precisiones de esta implementación escapan el alcance del artículo original [1] y solo pueden encontrarse el código fuente [29].

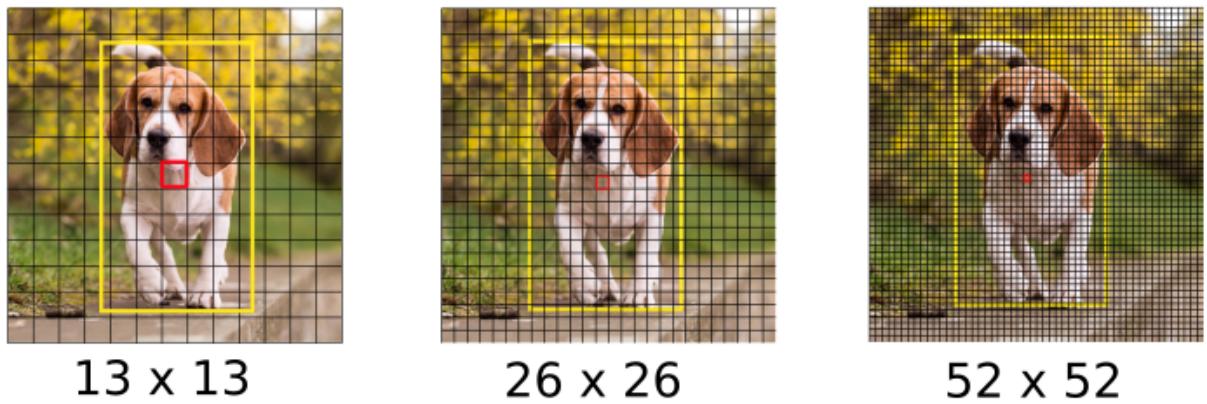


Figura 2.4: Salida multiescala de **YOLOv3** [28].

En lo que respecta a la asignación de *anchors*, la estrategia empleada por **YOLOv3** es simplemente determinar arbitrariamente 9 *clusters* diferentes mediante *k-means* y luego dividir estos valores de manera equitativa entre las escalas [1].

2.1.6. Función de costo

YOLO usa la la suma de errores cuadráticos como función de costo.

$$\begin{aligned}
 & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
 & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
 & + \lambda_{\text{obj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
 & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
 & + \lambda_{\text{classes}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned} \tag{2.4}$$

La función de costo de la Ec. 2.4 se compone de 3 términos:

1. Costo de localización.
2. Costo de confiabilidad.
3. Costo de clasificación.

IMPORTANTE: Como vimos en la Sección 2.1.4, en el caso mas general **YOLO** realiza múltiples predicciones por celda. Para calcular la función de costo queremos que solo una de esas predicciones sea comparada con la ubicación real de uno de los objetos del (**GT**).

Durante el entrenamiento no existe una manera completamente cierta de determinar que predictor (*bounding box*) es responsable del objeto detectado, por lo tanto la estrategia adoptada por **YOLOv3** [1] es asignar el **GT** al *anchor* con mayor **IOU**.

Antes de proseguir con los detalles de los términos que componen la Ec. 2.4, definimos algunas funciones útiles:

- $\mathbb{1}_{ij}^{\text{obj}}$: Es igual a 1 si hay un objeto presente en la celda i y en el *anchor* j . Es importante notar que:

1. La presencia del objeto esta determinada por el **GT**, la cual se encuentra codificada según la entramoto de celdas empleado.
2. La “responsabilidad” de la predicción esta determinada por el *anchor* al que pertenece la anotación del **GT**.

- $\mathbb{1}_{ij}^{\text{noobj}}$: Tiene el significado contrario a $\mathbb{1}_{ij}^{\text{obj}}$, es decir igual a 0 si hay un objeto presente en la celda i y en el *anchor* j . En el caso contrario es igual a 1.

Costo de localización

El costo de localización se compone de la Ec. 2.5 y de la Ec. 2.6.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \quad (2.5)$$

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \quad (2.6)$$

La Ec. 2.5 suma los errores cuadráticos de la localización del centroide (x, y) sobre las **B** predicciones por celda y sobre todos las celdas de la cuadrícula \mathbf{S}^2 .

La Ec. 2.6 suma los errores cuadráticos de ancho y alto (w, h) sobre las **B** predicciones por celda y sobre toda la cuadrícula \mathbf{S}^2 . Aquí, se ha introducido \sqrt{w} y \sqrt{h} para sopesar el hecho de que para un mismo valor absoluto de desviación sobre la **GT**, este error tiene menor importancia en *bounding boxes* grandes que en pequeñas.

En ambos casos, λ_{coord} es un parámetro ajustable para asignar un peso específico al error de localización dentro de la función de costo.

Costo de confiabilidad

El costo de confiabilidad de la Ec. 2.7 es el error cuadrático asociado con el puntaje de confiabilidad sobre cada predictor sumado sobre las **B** predicciones por celda y sobre toda la cuadrícula \mathbf{S}^2 .

$$\lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \quad (2.7)$$

Aquí hemos respetado la notación de la publicación original. La confiabilidad del predictor, C_i , es igual al valor de *objectness*. Esta es comparada con la **IOU** de la predicción sobre la **GT**, aquí designada \hat{C}_i .

Ya que la mayoría de las predicciones no contienen objetos, la relación entre λ_{obj} y λ_{noobj} sirve para disminuir el peso específico del error del ultimo término de la Ec. 2.7 dentro de la función de costo.

Costo de clasificación

El costo de clasificación de la Ec. 2.8 es simplemente el error cuadrático de la probabilidad condicional de clases, sumado sobre las **C** clases, las **B** predicciones por celda y toda la cuadrícula \mathbf{S}^2 .

$$\lambda_{\text{classes}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (2.8)$$

En la Ec. 2.8, al emplear $\mathbb{1}_{ij}^{\text{obj}}$ no penalizamos errores de clasificación cuando no hay ningún objeto presente en la celda o la detección es fallida.

Ademas, el costo de clasificación aplica a todas las predicciones determinadas como responsables de la detección de un objeto según el criterio de **YOLO**.

2.1.7. Inferencia y filtrado

De acuerdo a lo anteriormente expuesto, esta claro que la salida de la red durante la inferencia puede realizar múltiples detecciones para un mismo objeto presente en la imagen. Además, detecciones erróneas de objetos que no se encuentran en la imagen son igualmente posibles.

Por ejemplo, en la implementación multiescala de **YOLOv3**, tenemos $(13 \times 13) \times 3 + (26 \times 26) \times 3 + (52 \times 52) \times 3 = 10647$ predicciones en la red. La Fig. 2.5 muestra un numero arbitrario de predicciones que en su mayoría carecen de utilidad.

Para sobrellevar este punto, **YOLO** emplea filtrados sucesivos para eliminar estos posibles errores.

En primera instancia las predicciones son filtradas según su puntuación de *objectness*, dado un valor de umbral determinado. Seguidamente, se aplica un algoritmo de *Non-Maximum Suppression* (**NMS**) para atacar el problema de la detección múltiple de objetos.

El algoritmo se puede resumir como:

1. Ordenar las predicciones según su puntuación de *objectness*.
2. Empezando por la predicción mayor puntuada. Computar la **IOU** con todas las demás predicciones según el esquema de la Fig. 2.1. Eliminar aquellas que presentan una intersección mayor a un umbral predeterminado.
3. Iterar sobre la siguiente predicción en la escala de puntuación, hasta que no haya mas predicciones con puntuación menor.

Un ejemplo de la aplicación del algoritmo **NMS** sobre algunas detecciones arbitrarias puede verse en la Fig. 2.6.

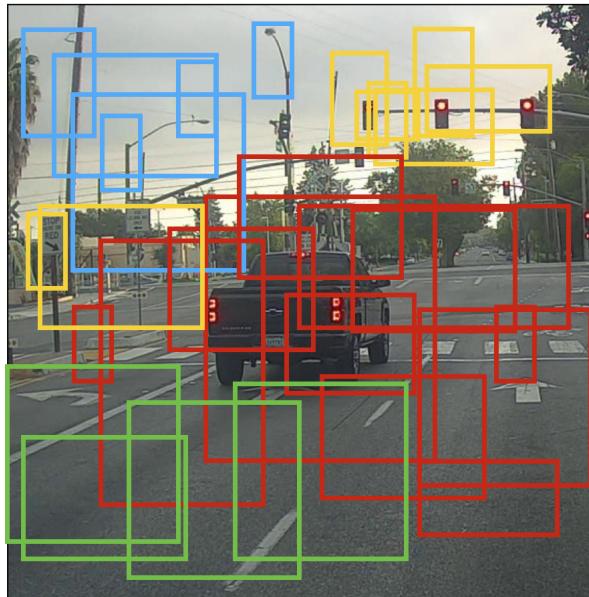


Figura 2.5: Imagen ilustrativa de un número arbitrario de predicciones sin utilidad entregadas por un detector multiclas [28].

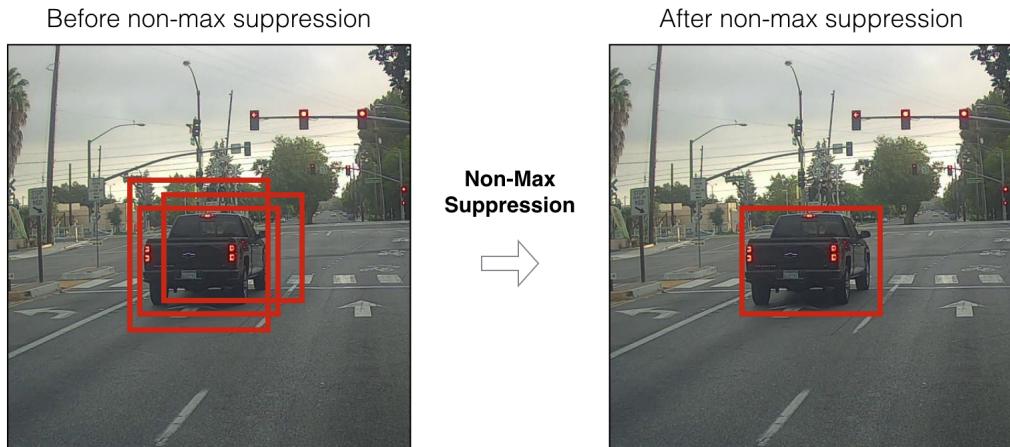


Figura 2.6: Imagen ilustrativa de la aplicación del filtrado NMS [28].

2.2. Implementación

En la implementación de este modelo **Darknet-53** actúa como extractor de *features*, por lo que las capas se mantuvieron “congeladas” durante el entrenamiento del detector. La Fig. 2.8 muestra un ejemplo de las imágenes contenidas en *dataset* de *ImageNet 1000*.

Se empleo una salida multiescala del detector de valores $S \times S = 13 \times 13$, 26×26 y 52×52 respectivamente. El numero de predicciones por celda es $B = 3$.

Para penalizar las no-detecciones correctas sobre la función de costo de la Sección 2.1.6 se tomaron valores de $\lambda_{obj} = 5$ y $\lambda_{noobj} = \lambda_{classes} = \lambda_{coord} = 1$ [29]. Al tratar-

se de una única clase a ser detectada, el termino de la Sección 2.1.6 se simplifica significativamente.

El detector se entrena por un total de 100 *epochs* con un *learning rate* de 1×10^{-4} . El optimizador empleado es el *Stochastic Gradient Descent (SGD)* con un valor de *decay* de 5×10^{-4} y momento de 9×10^{-1} [1]. El tamaño del *batch* de entrenamiento se fijo en 16 imágenes. Se ha empleado aumentación de datos no distorsiva durante el entrenamiento.

Para el entrenamiento de esta red se priorizo el usos de imágenes naturales, y no sintéticas, ya que *a priori* se puede suponer que la red se adapta a extraer los *features* característicos que corresponden a los alrededores de la matrícula.

Los *datasets* de este tipo son escasos, debido principalmente a la dificultad de adquirir las imágenes y el tiempo que demanda realizar las anotaciones pertinentes sobre cada imagen individual. En algunos países existen implicaciones legales a raíz usar este tipo de información vehicular.

Dada las similitudes de formato de la patente unica del Mercosur con la patente reglamentaria de la Unión Europea, se decidió usar una base de datos de acceso publico, la cual consiste de poco mas de 500 imágenes, anotadas en formato **XML** [2]. La Fig. 2.7 muestra a manera de ilustración un ejemplo de una de las imágenes contenidas en el *dataset*, las cuales pertenecen principalmente a escenas de trafico urbano, capturadas a bordo de un vehículo transitando.



Figura 2.7: Ejemplo de imagen usada en el entramiento de la red [2].

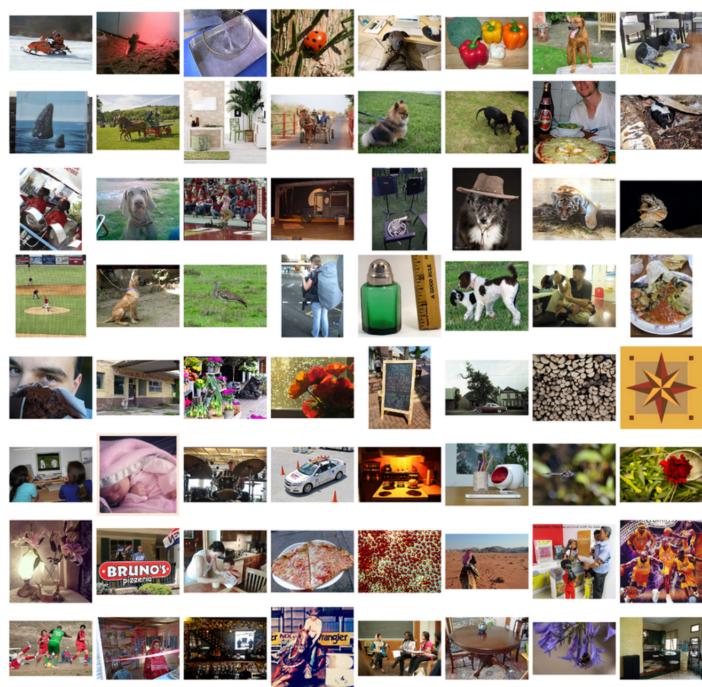


Figura 2.8: Ejemplos contenidos en el *dataset* de ImageNet [3].

Capítulo 3

Reconocimiento

En este capítulo se presenta el esquema de reconocimiento empleado para transformar imágenes de texto impreso en una secuencia de caracteres alfanuméricos. Se realiza un breve repaso sobre las características más importantes del esquema empleado. Sobre el final del capítulo se proveen detalles de la implementación realizada. Además, se aborda en líneas generales el uso de un esquema auxiliar de detección de texto.

3.1. Reconocimiento

En este capítulo trataremos el problema del reconocimiento de caracteres como un problema clasificación de elementos dentro de una secuencia. Para ello emplearemos una red neuronal convolucional recurrente (**RCNN**) con memoria de largo-corto plazo, entrenada con una función de costo del tipo *Connectionist Temporal Classification* (**CTC**) para clasificar los caracteres dentro de la secuencia de texto [21].

Para evitar tener que efectuar la segmentación a nivel individual, y sus posibles errores asociados, las imágenes serán previamente “pre-procesadas” por una red CNN que servirá para extraer los *features* correspondientes a los caracteres. En esta red han incorporado transformaciones espaciales aprendidas a través de capas de transformación espacial **STN** (*Spatial Transformer Network*), las cuales permiten enfocar la atención en los *features* deseados en contexto de perspectivas complejas o de inclusiones no deseadas.

Seguidamente los *features* son procesados por la red neuronal recurrente (**RNN**). Esta técnica permite incorporar información contextual en el reconocimiento de los

caracteres individuales. La salida de esta red finalmente es decodificada por la **CTC**, lo que facilita el uso de anotaciones sin “alineación” durante el entrenamiento de la red.

No de menor importancia es que esta solución permite la aplicación a problemas de reconocimiento más complejos, como puede ser el procesamiento del lenguaje natural a partir de sus caracteres ópticos, por ejemplo, imágenes digitales de texto impreso que pueden aparecer en contextos urbanos normales.

3.1.1. Red convolucional

La primera etapa de la red de reconocimiento de caracteres consta de una **CNN** con el objetivo de “pre-procesar” las imágenes de entrada con una serie de filtros aprendidos que extraigan los *features* de los caracteres.

Posteriormente esta red alimenta una capa **STN**, explicada en la Sección 3.1.2, para finalmente procesar la información en la **RNN** de la Sección 3.1.3.

La Tabla 3.1 muestra la arquitectura de la primera etapa de la red.

Bloque	Layer	Output Size	Features	Kernel Size	Padding	Activation
1	Input	200x31	-	-	-	-
	Conv2D	200x31x64	64	3x3	same	relu
	Conv2D	200x31x128	128	3x3	same	relu
	Conv2D	200x31x256	256	3x3	same	relu
	BatchNormalization	200x31x256	-	-	-	-
	MaxPool2D	100x15x256	-	2x2	-	-
2	Conv2D	100x15x256	256	3x3	same	relu
	Conv2D	100x15x512	512	3x3	same	relu
	BatchNormalization	100x15x512	-	-	-	-
	MaxPool2D	50x7x512	-	2x2	-	-
3	Conv2D	50x7x512	512	3x3	same	relu
	Conv2D	50x7x512	512	3x3	same	relu
	BatchNormalization	50x7x512	-	-	-	-

Tabla 3.1: Arquitectura empleada.

Las dimensiones de la capa de entrada se han adecuado a las dimensiones de las imágenes del *dataset* usado. Véase la Sección 3.3 para más detalles.

Las capas de *Max Pooling* se agregan para reducir el costo computacional del modelo y reducir el *over-fitting* del modelo [30].

Durante el entrenamiento las activaciones de las capas internas de la red van adaptándose a medida que se corrigen los pesos de los *kernels* convolucionales, por lo que las capas inmediatamente siguientes tienen el problema de recibir una entrada

variable en cada paso del entrenamiento. En otras palabras, para las capas internas el objetivo de la optimización va cambiando a medida que cambian los pesos. Por ese motivo se han agregado capas de *Batch Normalization* intermedias entre los bloques convolucionales para normalizar las activaciones de las capas previas. Esto permite que el aprendizaje de las capas internas se haga sobre una distribución más “estable” de *inputs*, y por lo tanto, acelera el proceso de aprendizaje [30].

3.1.2. Transformación espacial

Dentro de la arquitectura de la red de reconocimiento se ha agregado una capa **STN**, la cual se trata de un modulo diferenciable con el objetivo de atacar la falta de invarianza espacial de la red. En [6] se pueden encontrar los detalles de la formulación, así como resultados que muestran que es posible mejorar la *perfomance* de un clasificador mediante su aplicación como módulos individuales dentro de la red.

En líneas generales, las redes convolucionales no son robustas a las variaciones de la entrada, en particular, cuando ejemplos pertenecientes a la misma clase tienen mucha variación entre si.

Estas variaciones pueden incluir un número muy grande de transformaciones, como por ejemplo el cambio en iluminación o de fondo de un objeto. En el caso especial de transformaciones espaciales, la convolución discreta no es invariante a cambios de escala o de rotación. A su vez, la invarianza de translación, la cual si esta presente en el caso más simple, se pierde al agregar operaciones de *pooling* o por la adición de capas totalmente conexas (densas) dentro de la red.

La intuición detrás de las capas de **STN** es que pueden producir una transformación espacial de la imagen entrada (o de un mapa de *features* dentro de la red) de manera de enfocar el objeto de interés. Una de las principales ventajas de este método es que las capas son diferenciables, lo que permite que la transformación espacial sea aprendida. A su vez, no es necesaria la modificación de otros hiperparametros de la red para su implementación. La Fig. 3.1, y Fig. 3.2 ejemplifica la acción de la red sobre una imagen de entrada.

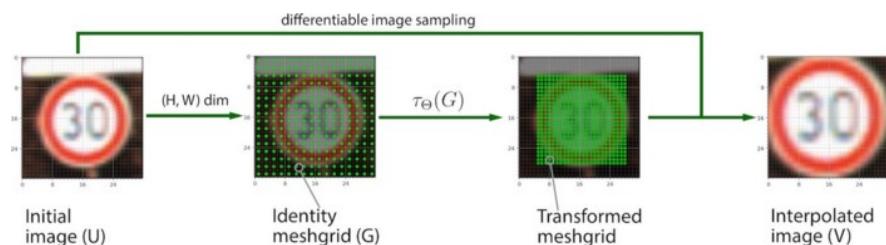


Figura 3.1: Resultado de una transformación espacial **STN** [4] sobre una imagen del *dataset* de reconocimiento de señales de tráfico alemanas [5].

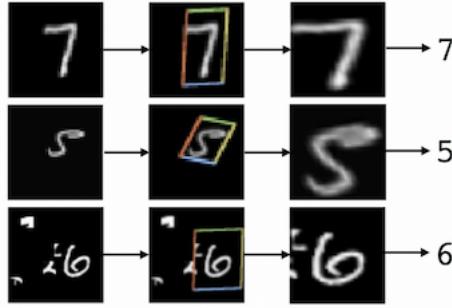


Figura 3.2: Transformación espacial sobre algunos ejemplos del conjunto MNIST [6].

El funcionamiento de la red esta descripto por la Fig. 3.3. La misma consta de una *localisation network* encargada de aprender en sus capas ocultas la transformación espacial a aplicar. Después, la transformación es usada para crear una grilla de donde la entrada sera *sampleada*. Finalmente, tomando la grillas y el mapa de entrada, se produce un mapa de salida transformado a partir de los valores de la entrada en los puntos de la grilla.

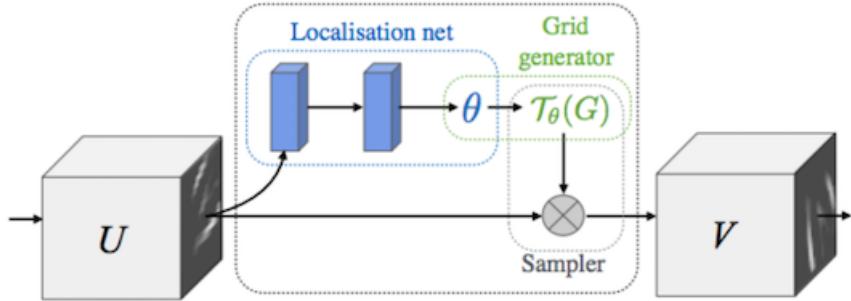


Figura 3.3: Bloques funcionales dentro de una capa de STN [6].

La Ec. 3.1 describe la transformación. Aquí, Φ_x y Φ_y son los parámetros de un kernel genérico $k()$. U_{nm}^c es el valor del mapa de entrada en las coordenadas (n, m) en el canal c . V_i^c es la salida para el pixel i en la locación (x_i^t, y_i^t) en el canal c . Notar que la transformación se realiza de igual manera en todos los canales.

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y) \quad \forall i \in [1 \dots H'W'] \quad \forall c \in [1 \dots C] \quad (3.1)$$

Vease [6] para la deducion de los gradientes de la transformación. La implementacion empleada en la red de reconocimiento esta basada en el trabajo de [31].

3.1.3. Memorias de largo-corto plazo

Red neuronal recurrente

Las redes neuronales recurrentes tienen un estado interno que puede representar información contextual, lo que las hace particularmente buenas para el procesamiento de secuencias, ya que pueden mantener información de entradas pasadas por un tiempo no fijado *a priori*, si no que depende de pesos entrenables.

Es decir, es posible transformar una secuencia de entrada en una secuencia de salida, teniendo en cuenta información del contexto de cada elemento en una manera flexible.

La Fig. 3.4 muestra el funcionamiento esquemático de una **RNN**.

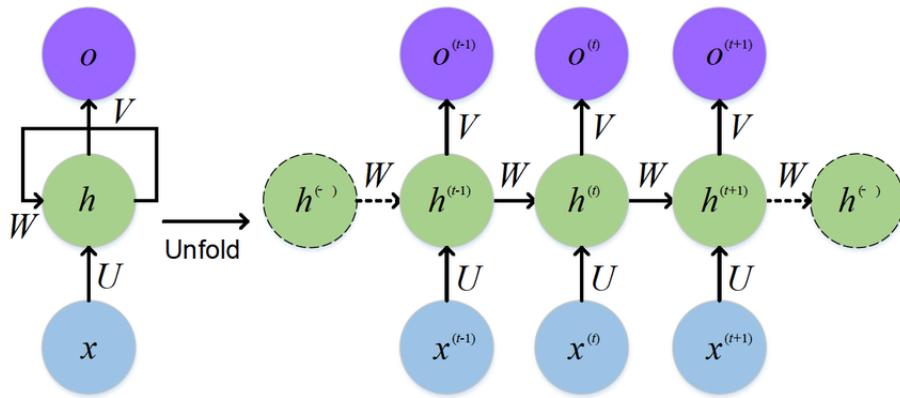


Figura 3.4: Desglose del funcionamiento de una **RNN** [7].

Como se puede apreciar en la Fig. 3.4, la red trabaja en “ciclos”, donde la información sobre las activaciones de un paso temporal previo influencian las predicciones del paso de tiempo actual. El flujo de información también puede ir en ambas direcciones, de “tiempo” creciente y decreciente, lo que da origen a una capa **RNN** bi-direccional.

Se sigue que la información contextual que se mantiene en el tiempo puede ser de muy largo plazo, dependiendo de la cantidad de pasos de tiempo.

Existen dos problemas principales con el entrenamiento de las **RNN** [32] que dificultan su implementación más simple:

1. Dependencia a largo plazo de la información que puede no ser de utilidad.
2. Gradientes que se desvanecen o explotan en la retro-propagación de errores.

Compuertas de memoria

En la práctica una red **RNN** estándar solo puede aprender por una cantidad de pasos de tiempo limitada, ya que el problema del gradiente que se desvanece o explota dificulta el aprendizaje de la misma. Aquí entra en juego el modelo **LSTM** (*Long-Short Term Memory*), el cual es afectado por este problema en una magnitud mucho menor, ya que permite olvidar información no relevante.

Una **LSTM** consiste de una serie de bloques conectados recurrentemente, conocidos como bloques de memoria. Cada uno de estos bloques diferenciables contiene tres unidades multiplicativas que realizan operaciones análogas a la “lectura”, “escritura” y “borrado” de la información dentro de la red [33, 34].

La Fig. 3.5 muestra un desglose de los bloques funcionales que componen una compuerta diferenciable **LSTM**.

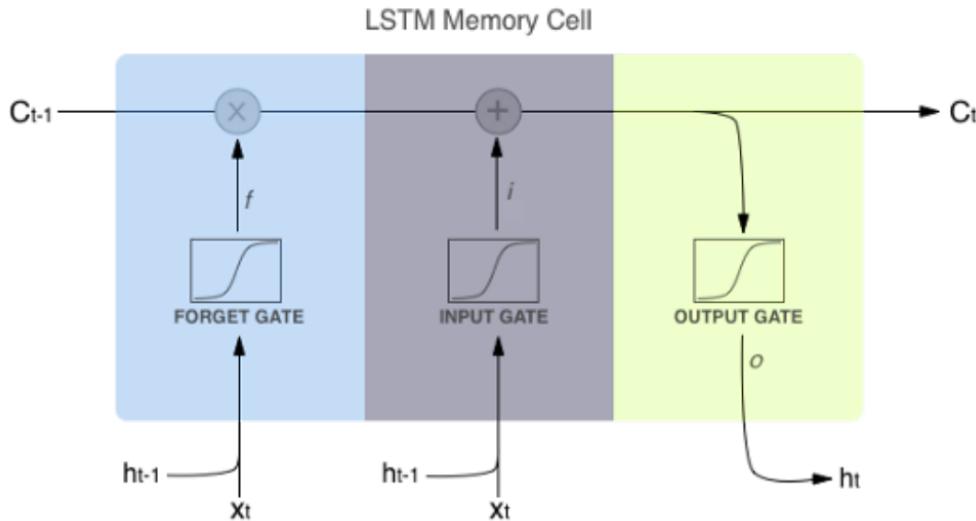


Figura 3.5: Bloques funcionales dentro de una compuerta **LSTM**. Modificada a partir de [8].

Los bloques de la Fig. 3.5, aunque complejos a simple vista, tienen ecuaciones que son relativamente simples.

$$\begin{aligned} i_t &= \sigma(w_i [h_{t-1}, x_t] + b_i) \\ f_t &= \sigma(w_f [h_{t-1}, x_t] + b_f) \\ o_t &= \sigma(w_o [h_{t-1}, x_t] + b_o) \end{aligned} \quad (3.2)$$

En la Ec. 3.2, σ es la operación sigmoidal, y $[,]$ es la operación de concatenación, a veces reemplazada por una suma común, elemento a elemento, dependiendo de la implementación.

La **LSTM** se compone en esencia de 3 compuertas, i_t , f_t , o_t , llamadas **Input gate**, **Forget gate** y **Output gate**, respectivamente. Son llamadas compuertas ya que la operación sigmoidal restringe los valores de los vectores al rango $[0, 1]$, regulando en mayor o menor la información que se pasa al nuevo estado. Aquí:

1. i_t define cuanta información del nuevo estado calculado pasa a la siguiente etapa.
2. f_t define cuanta información anterior es relevante para las siguientes etapas.
3. o_t provee la activación para la salida del bloque **LSTM** en el tiempo t .

A partir de estas 3 compuertas ahora podemos conformar la nueva salida t del bloque LSTM según la Ec. 3.3.

$$\begin{aligned}\tilde{c}_t &= \tanh(w_c [h_{t-1}, x_t] + b_c) \\ c_t &= f_t * c_{t-1} + i_t * \tilde{c}_t \\ h_t &= o_t * \tanh(c_t)\end{aligned}\quad (3.3)$$

Primero calculamos un “candidato” \tilde{c}_t del estado en tiempo t a partir de la nueva entrada x_t junto con la información de los estados “ocultos” h_{t-1} de las etapas previas.

Seguidamente es posible calcular la memoria interna de la unidad c_t , a partir del resultado de la multiplicación elemento a elemento (designada aquí como $*$) del estado previo c_{t-1} y la compuerta de olvido f_t y la información que tenemos que considerar del paso de tiempo actual $c_t * i_t$.

Finalmente este resultado pasa por la activación de la capa y es filtrado por la compuerta de salida o_t , que determina que porción del estado actual debe transmitirse a la siguiente capa en el estado $t + 1$.

El estado del bloque en t es directamente h_t , que puede pasarse, por ejemplo, por una función *softmax* para obtener la predicción de la red en ese paso de tiempo.

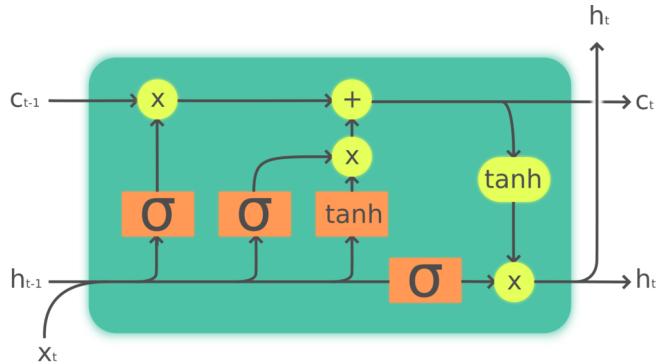


Figura 3.6: Bloques funcionales dentro de la LSTM de acuerdo a la Ec. 3.2 y la Ec. 3.3 [9].

Todo este procedimiento da como resultado el bloque funcional de la Fig. 3.6, que tiene la particularidad de ser diferenciable, y por ende sujeto al entrenamiento.

3.1.4. Función de costo

Como se mencionó en la Sección 3.1, la función de costo empleada para el procesamiento de las secuencias de texto fue la *Connectionist Temporal Classification (CTC)* [35–37].

Para explicar su funcionamiento, tómese por ejemplo una imagen conteniendo una palabra en texto impreso o manuscrito, como ser la Fig. 3.7. En el caso más común, las anotaciones contendrán la palabra transcrita en caracteres alfanuméricicos, pero no

habrá información sobre como se “alinea” cada carácter con cada pixel o región de pixeles dentro de la imagen. Aún más, queremos que el reconocimiento sea capaz de detectar distintas fuentes o caligrafías, que pueden tener mucha variación entre sí, por lo que cualquier regla fija que podamos idear sobre que región ocupa cada carácter es susceptible a ser rota.

Este problema es común en el procesamiento de sentencias de distinto tipo, por ejemplo, el reconocimiento del idioma hablado, o las anotaciones de acciones en capturas de vídeo.



Figura 3.7: Imagen generica de texto impreso de origen sintetico [10].

Alineación

Ahora bien, para una secuencia de entrada $\mathbf{X} = [x_1, x_2, \dots, x_t]$ se corresponde según nuestro mapeo deseado una secuencia de salida $\mathbf{Y} = [y_1, y_2, \dots, y_u]$.

De lo explicado anteriormente se sigue que:

1. Tanto \mathbf{X} como \mathbf{Y} pueden variar en longitud.
2. La relación entre longitudes de \mathbf{X} y \mathbf{Y} puede variar.
3. No conocemos la correspondencia entre elementos entre \mathbf{X} y \mathbf{Y} .

Una de las maneras de manejar estos inconvenientes entre las entradas y las anotaciones es mediante el uso la **CTC**, ya que para una entrada \mathbf{X} se obtiene una distribución de probabilidad sobre todos las posibles \mathbf{Y} , es decir, $p(\mathbf{Y}|\mathbf{X})$.

Se sigue que esta distribución puede ser usada para evaluar la probabilidad de una salida dada. Además, esta función debe ser diferenciable, para poder permitir el cálculo del gradiente y el entrenamiento de la red.

Además, en caso de la inferencia de la opción más probable (o una de ellas), nos interesa poder resolver la Ec. 3.4.

$$\mathbf{Y}^* = \operatorname{argmax}_{\mathbf{Y}} p(\mathbf{Y}|\mathbf{X}) \quad (3.4)$$

Para poder calcular $p(\mathbf{Y}|\mathbf{X})$, la **CTC** debe sumar sobre todos las posibles alineaciones que resultan en una salida valida \mathbf{Y} .

Consideraremos la secuencia $\mathbf{Y} = [h, e, l, l, o]$. Para alinear una entrada arbitraria \mathbf{X} con la salida \mathbf{Y} , se introduce el carácter nulo, denominado con la letra ϵ . Este carácter nulo es eliminado de la salida subsecuentemente.

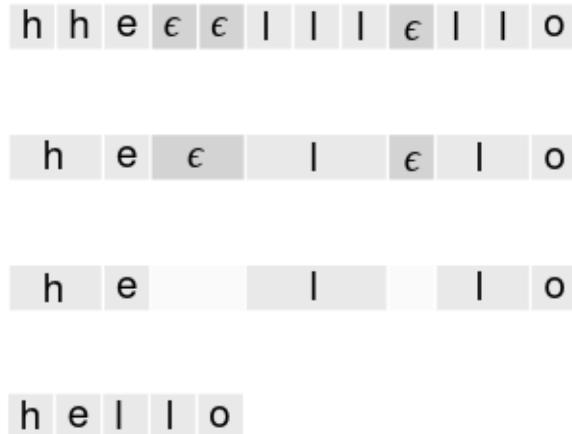


Figura 3.8: Una alineación posible de la secuencia $\mathbf{Y} = [h, e, l, l, o]$ [11].

Las alineaciones permitidas por la **CTC** tienen la misma longitud que la entrada \mathbf{X} . Aquí, permitimos que caracteres repetidos “colapsen” en un solo dígito o letra, y todos los caracteres nulos ϵ son removidos. La Fig. 3.8 muestra este proceso para una entrada arbitraria \mathbf{X} .

Estas alineaciones permitidas tienen la propiedad de que el número de alineaciones \mathbf{X} para una misma \mathbf{Y} es grande, pero a su vez no existe una alineación \mathbf{X} que se pueda corresponder con más de una anotación \mathbf{Y} . Además, a partir de la Fig. 3.8 se puede inferir que la longitud de \mathbf{Y} no puede ser mayor que la de \mathbf{X} .

Funcionamiento

Para emplear la función **CTC** es necesario que el modelo entregue una distribución de salida sobre todas las clases (diccionarios de caracteres permitidos en este caso), partiendo de una sección de tamaño fijo de la entrada.

Los modelos entrenados con **CTC** típicamente usan **RNN** para estimar $p(\alpha_t|\mathbf{X})$, es decir, la probabilidad del carácter α perteneciente al diccionario, en el paso de tiempo t . En la Fig. 3.9 se puede visualizar el funcionamiento de la red para el diccionario $\{h, e, l, o, \epsilon\}$.

Aquí, el vector de probabilidades sobre las clases esta representado como una columna de la matriz. Hay tantas columnas como pasos de tiempo de la **RNN**. La tonalidad de cada elemento representa la probabilidad del mismo.

Finalmente todas las posibles combinaciones de izquierda a derecha dan como resultado las alineaciones permitidas. Formalmente:

$$p(\mathbf{Y}|\mathbf{X}) = \sum_{A \in \mathcal{A}_{X,Y}} \prod_{t=1}^T p(\alpha_t|\mathbf{X}) \quad (3.5)$$



Figura 3.9: Funcionamiento de la red entrenada con una función de costo **CTC** [11].

En la Ec. 3.5, marginamos sobre el conjunto $A \in \mathcal{A}_{X,Y}$ de alineaciones validas que nos entregan la sentencia de salida correcta \mathbf{Y} . Aquí, se ha hecho la suposición, no del todo valida, de que $p(\alpha_t|\mathbf{X})$ son independientes. Véase [38] para una extensión de esta formulación sin emplear esta suposición.

Esta función puede ser muy costosa de computar, ya que el número de alineaciones posibles puede ser gigantesco para cada sentencia de salida. En la práctica se emplea programación dinámica para acelerar el proceso de computo de la Ec. 3.5 [35].

El cálculo se suele hacer en un espacio logarítmico para evitar problemas de estabilidad numérica [35], ya que la productoria de la Ec. 3.5 puede transformarse en una sumatoria, más fácil de manejar computacionalmente.

La implementación de la **CTC** es compleja y excede el alcance de este trabajo, por lo que se ha optado por usar la función integrada dentro de **Tensorflow** [39]. A su vez, la librería **cuDNN** de **NVIDIA** provee implementaciones que funcionan en *GPU*.

3.2. Segmentación a nivel palabra

Siguiendo el lineamiento del Capítulo 2, la salida de la red de segmentación es una imagen conteniendo la **ROI** de interés con la matrícula vehicular dentro de ella, junto con información del contexto del objeto.

Según trabajos anteriores sobre mecanismos de atención visual similares a los implementados aquí [40], podemos suponer que la arquitectura de la red de reconocimiento es capaz de extraer los *features* que identifican los caracteres dentro de la secuencia

de texto. Sin embargo, la disponibilidad de datos de entrenamiento con imágenes de matrículas segmentadas, y sobre todo, correctamente anotadas, es extremadamente escasa. A esto se le suma que el proceso de recolección y anotación de imágenes se juzgo en primera instancia como prohibitivo en tiempo.

Para sobrellevar este impedimento se ha optado por entrenar la red de reconocimiento sobre un *dataset* de texto impreso de origen sintético. Véase la Sección 3.3 para detalles al respecto de la implementación del modelo.

Esta estrategia tiene en cambio el inconveniente de necesitar de una etapa previa de segmentación del texto dentro de la imagen. Para sobrellevar este impedimento se ha empleado otra red de segmentación denominada **CRAFT** (*Character Region Awareness For Text Detection*) [41].

CRAFT funciona particularmente bien para segmentar texto a nivel palabra con orientaciones arbitrarias. Para ello, la red intenta reconocer caracteres individuales y explorar la afinidad entre ellos en una sentencia de varias palabras.

Para sobrellevar la falta de imágenes segmentadas a nivel carácter, **CRAFT** esta entrenada con datos de origen sintético, así como con escenas naturales que contienen texto, que han sido pre-procesadas para estimar las **GT** correspondientes a cada carácter individual.

Como se dijo anteriormente **CRAFT** sobresale en la detección de texto curvado o deformado que esta en orientaciones arbitrarias, haciéndolo particularmente bueno para este tipo de aplicación. Los detalles de su funcionamiento pueden verse en [41]. Las Figs. 3.10 y 3.11 muestran ejemplos de su aplicación. La arquitectura de la red no ha sido modificada, ni re-entrenada y su uso se limita a vincular las salidas de la redes presentadas en el Capítulo 2 y el Capítulo 3.



Figura 3.10: Ejemplo de segmentación a nivel palabra de **CRAFT**.



Figura 3.11: Ejemplo de segmentación a nivel palabra de **CRAFT**.

3.3. Implementación

A la salida del **Bloque 3** de la red convolucional de la Sección 3.1.1 se ha incluido una capa **STN** como mecanismo de atención espacial. Posteriormente, una capa densa disminuye la dimensión del modelo previo al uso de 2 capas **LSTM** bi-direccionales.

La expresión regular de las sentencias aceptadas es `^ [a-z 0-9]*$`, es decir, el método es indiferente al uso de mayúsculas o minúsculas. De la Sección 3.1.4 se sigue que la salida de la red es una capa densa, cuyo numero de unidades se condice con el numero de caracteres del diccionario de salida. Se emplea una función de activación del tipo *softmax*.

El modelo de reconocimiento se encuentra entrenado con un subconjunto de imágenes del *dataset* de **Synthetic Word Dataset** [10]. Para el entrenamiento de esta red no se emplearon imágenes naturales.

El subconjunto esta compuesto por $1,5 \times 10^5$ palabras del idioma inglés, donde además se incluyen algunos dígitos. Las imágenes se encuentran en escala de grises exclusivamente.

No se emplea aumentación de datos sobre el *dataset*, debido a la variedad de iluminaciones y orientaciones presentes en las imágenes. El tamaño de entrada de la red es 200×31 , lo permite que una gran cantidad de imágenes puedan ser procesadas sin modificar. Sobre las imágenes donde esto no fue posible, se ha aplicado un *padding* para no distorsionar demasiado los ejemplos de entrada.

La red se entrena por un total de 50 *epochs* con un *learning rate* de 1×10^{-3} . El optimizador empleado es el *Adam* con un valores de $\beta_1 = \beta_2 = 9 \times 10^{-1}$, $\epsilon = 1 \times 10^{-8}$ y un decay de 5×10^{-3} . El tamaño del batch de entrenamiento se fijo en 128.

La Fig. 3.12 muestra un ejemplo de las imágenes contenidas en el *dataset*.



Figura 3.12: Ejemplo de imagenes contenidas en *dataset* de **Synthetic Word Dataset** [10].

Capítulo 4

Resultados y discusión

*En base al modelo presentado en el Capítulo 2 y en el Capítulo 3, presentamos aquí los resultados del sistema realizando predicciones sobre un dataset recolectado a medida para testear su funcionamiento en condiciones lo mas reales posibles. En primera instancia se introducen brevemente las características de estas imágenes y se presentan las metricas a ser evaluadas. Después, a modo de comparación, presentamos los resultados obtenidos usando un sistema **ALPR** ampliamente usado comercialmente. Finalmente presentamos los resultados de nuestro sistema sobre el dataset e incluimos una breve discusión al respecto.*

4.1. Dataset

No se tiene conocimiento de una gran variedad de *datasets* de acceso público que contengan una cantidad significativa de imágenes de matriculares anotadas.

Open Images v4 [42], mantenida por Google, es una de las bases datos conocidas con la mayor cantidad de patentes entre sus imágenes, aunque las imágenes esta tomadas de diferentes locaciones alrededor del mundo y muchas contienen caracteres que no se corresponden con alfabeto latín. A su vez, no todas las imágenes de patentes se encuentran anotadas, y algunas anotaciones son incorrectas [42].

El **INTI** mantiene un *dataset* de patentes del Mercosur de Argentina, aunque el mismo no es de publico acceso [43].

Con el motivo de poder testear el modelo presentado en los Capítulos 2 y 3, se

ha ensamblado un conjunto de imágenes de patentes del Mercosur tomadas desde un teléfono móvil en resolución de 12 MP con dimensiones de 4608×2592 , desde diferentes orientaciones e iluminaciones. La mayoría de ellas se corresponden con vehículos estáticos, aunque se incluyen algunas imágenes en movimiento. Unas pocas imágenes extraídas de internet se suman al conjunto.

Ya que el modelo de la patente única del Mercosur coincide con el formato europeo de matrículas [44], se puede emplear este conjunto de imágenes para comparar el funcionamiento del modelo con utilidades comercialmente disponibles que funcionan con el formato europeo por default, por ejemplo, **OpenALPR**, como se vera en la Sección 4.3. Por este motivo, la patente Argentina antigua ha sido filtrada de las fotos, aunque algunos resultados pueden observarse en la Sección 4.4.

Al tratarse únicamente de 50 imágenes, se ha empleado aumentación de datos para generar una base de datos mas amplia en la que probar el modelo. Para ello, empleamos el *software* **Albumations** [12], una librería de código abierto rápida y flexible con varias operaciones de transformación a disposición para aplicar sobre imágenes. Ya que las capturas han sido tomadas desde diferentes perspectivas y orientaciones, las modificaciones hechas sobre las imágenes no son distorsivas, si no que solamente conllevan cambios en la iluminación o el agregado de ruido. Estas transformaciones particulares no se aplicaron sobre el conjunto de entrenamiento del Capítulo 2.

En las Figs 4.2, 4.3 y 4.4 muestran modificaciones realizadas sobre la imagen original de la Fig. 4.1.



Figura 4.1: Imagen original sin modificar.



Figura 4.2: Imagen modificada con Albumentations [12].



Figura 4.3: Imagen modificada con Albumentations [12].

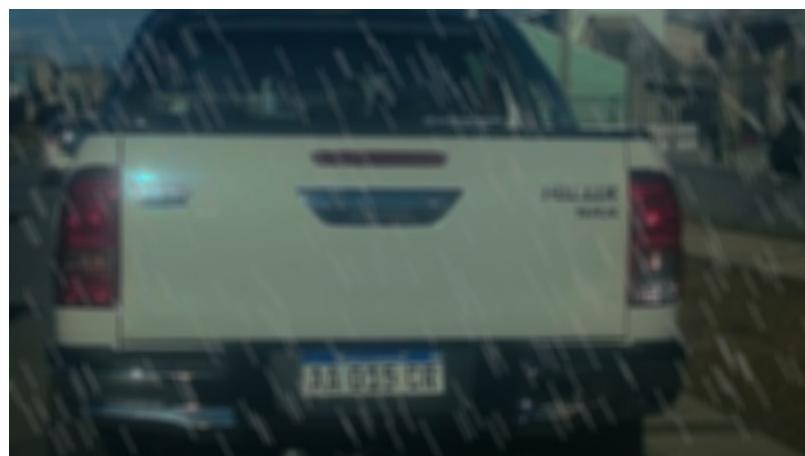


Figura 4.4: Imagen modificada con Albumentations [12].

4.2. Métricas

Ya que nuestro sistema resuelve un doble problema de detección y reconocimiento, debemos definir métricas acordes a esta situación para poder evaluar los resultados de manera coherente y poder compararlos con otros sistemas similares. Las métricas aquí presentadas están basadas en métricas de detección de objetos multi-clase, las cuales son significativamente mas complejas que la empleadas para problemas de clasificación.

A su vez, nuestras métricas son ligeramente distintas y se evalúan sobre una única clase. Además están adaptadas a las condiciones particulares de un sistema de **ALPR**.

4.2.1. Predicciones

Para nuestro problema específico las predicciones pueden clasificarse, dentro de las siguientes categorías:

- **TP (True Positives)**: Describe la situación donde una matricula dentro la imagen de entrada es detectada y reconocida correctamente.
- **TN (True Negatives)**: Describe la situación donde la falta de matriculas es reconocida como “ningún objeto”. Es un escenario trivial y no sera empleado ya que nuestro *dataset* de prueba solo contiene imágenes “positivas”, es decir, que contienen al menos una matricula.
- **FP (False Positives)**: En este escenario, existe una predicción de la matricula, pero es incorrecta. Esto se puede deber a una detección fallida (la región segmentada esta completamente o parcialmente errada), o bien existe un error de reconocimiento (al menos un carácter mal reconocido).
- **FN (False Negatives)**: En este escenario, no hay ninguna predicción de matricula para una imagen “positiva” de entrada.

La Fig. 4.5 ejemplifica los 3 tipos no triviales de predicciones en relación al **GT**.

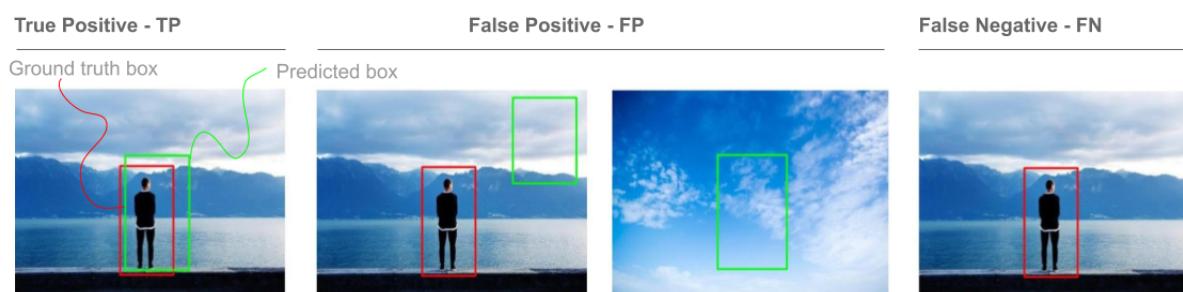


Figura 4.5: Ejemplo de clasificación en categorías no triviales [13].

Notar que la definición de **FP** es ligeramente distinta a lo usualmente usado en problemas de detección puros, ya que también consideramos un reconocimiento errado (parcial o total) como un falso positivo. Seremos consistentes con estas definiciones para realizar la comparación entre métodos. Además, no usamos la **IOU** como una parte fundamental de las métricas, ya que queremos medir el funcionamiento global del sistema, no solo de su etapa de segmentación.

NOTA: Una misma situación puede dar origen a diferentes categorías. Tómese por ejemplo la Fig. 4.6. Aquí la predicción correcta del auto de la derecha da origen a un **TP**, mientras que la predicción errada del auto de la izquierda da lugar a un **FP** (ya que hay un error en el numero 8, que es leído como un 6), pero también a un **FN** ya que no hay ninguna predicción correcta sobre esta matrícula. Otra situación atípica se presenta cuando varias predicciones reconocen la misma placa correctamente, una de ellas se considera **TP**, mientras que las cajas restantes se consideran **FP**.

Esta claro que la etapa de detección juega un rol predominante en la generación de predicciones **FP** y **FN**. La elección de un umbral de confiabilidad alto hace al sistema robusto contra errores del tipo **FP**, pero actúa en detrimento del numero de **FN**, el cual aumenta. Disminuir este umbral tiene el efecto contrario. Aquí fijaremos este umbral en $\lambda^{obj} = 0,75$. Esto se combina al mecanismo de **NMS** de **YOLOv3** lo que nos permite disminuir el numero de **FP**. El umbral de este algoritmo se fija en $\lambda^{nms} = 0,45$, es decir, el valor por default de la implementación original [29]. Es posible optimizar estos parámetros según el *dataset* sobre el que cual realizamos la evaluación de la métricas para obtener mejores resultados.



Predictions: [[['aa', '205', 'nx'], ['gdt', '465']]]

Figura 4.6: Ejemplo de una situación que da origen a un **TP**, un **FP** y un **FN**.

4.2.2. Métricas

Según las definiciones de la sección precedente definimos 3 métricas a evaluar sobre nuestro *dataset*. A saber:

1. **Accuracy:** Según la Ec. 4.1, la definimos como el porcentaje de ejemplos correctamente predichos del total de predicciones.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \quad (4.1)$$

2. **Precision:** Según la Ec. 4.2, la definimos como una medida de la probabilidad de las predicciones se adecuen a las anotaciones de la **GT**. A la precisión a veces también se la denomina valor de predicción positivo.

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{\text{true object detection}}{\text{all detected boxes}} \quad (4.2)$$

3. **Recall:** Según la Ec. 4.3, la definimos como una medida de la probabilidad de las anotaciones de la **GT** sean correctamente detectadas. También se la denomina sensibilidad.

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{\text{true object detection}}{\text{all ground truth boxes}} \quad (4.3)$$

Las diferencias entre las definiciones pueden resultar sutiles. Ejemplificando 3 situaciones típicas:

- *Recall* alto pero *Precision* baja; implica que se han detectado la mayoría de los objetos correctamente, pero la mayoría de las predicciones son incorrectas (muchos **FP**).
- *Recall* bajo pero *Precision* alta; implica que la mayoría de las predicciones son correctas, pero la mayoría de los objetos no han sido detectados (muchos **FN**).
- *Recall* alto y *Precision* alta; implica que la mayoría de las predicciones son correctas y además que se han detectado la mayoría de los objetos correctamente (sistema ideal).

Tener en cuenta que hemos evaluado el rendimiento global manteniendo los umbrales de confiabilidad de las 3 redes constantes, lo que da un *trade-off* entre las métricas de *Recall* y *Precision*, las cuales son complementarias y difíciles de optimizar al mismo tiempo variando los hiper-parámetros de cada red individual.

4.2.3. Métrica de Levenshtein

Ahora bien, supongamos que el **GT** de una matrícula es **AA000AA**, y un **FP** esta dado por **AA000AB**, mientras que otra predicción esta dada por **AA123AB**. Se sigue que no todas las predicciones erradas están erradas en igual magnitud. Para tener esto en consideración introducimos el concepto de métrica de *Levenshtein* con la que evaluaremos las categorías erradas.

La métrica de *Levenshtein*, es una distancia entre palabras que se define como el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación según el criterio *Levenshtein*, bien una **inserción**, **eliminación** o la **sustitución** de un carácter. Véase la Fig. 4.7 como ejemplo.

Se le considera una generalización de la distancia de *Hamming*, que se usa para cadenas de la misma longitud y que solo considera como operación la sustitución. Hay otras generalizaciones, como la distancia de *Damerau-Levenshtein*, que consideran el intercambio de dos caracteres adyacentes.

Aunque costosa computacionalmente, esta métrica es relativamente simple de implementar y efectiva para cadenas cortas. Ya que las matrículas no tienen un significado semántico asociado, no hay necesidad *a priori* de emplear métricas mas complejas como las usadas en el procesamiento del lenguaje natural.

En este trabajo usaremos esta métrica como una similitud normalizada para facilitar la comparación entre métodos.

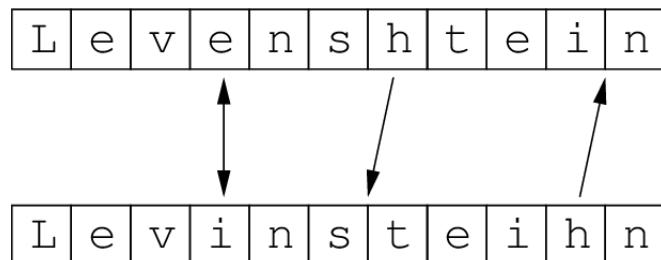


Figura 4.7: Intuición detrás de la métrica de *Levenshtein*. Las operaciones aceptadas son la **inserción**, **eliminación** o la **sustitución** de caracteres individuales.

4.3. Open ALPR

Con el objetivo de comparar el funcionamiento de nuestro modelo, evaluaremos las métricas contra los resultados obtenidos mediante **OpenALPR** [15]. **OpenALPR** es una biblioteca de reconocimiento automático de matrículas escrita en el lenguaje de programación C++. El software se distribuye en dos versiones, una de código abierto y otra comercial, aquí emplearemos la versión de código abierto que reconoce patentes en formato europeo, entre otras. Explicaremos, a continuación, sin entrar demasiado en detalle, su funcionamiento:

1. En primer lugar, la fase de detección utiliza el algoritmo de *Local Binary Patterns* (**LBP**) [45, 46], generalmente utilizado para la detección de rostros, para encontrar las posibles regiones de la placa. Cada una de estas regiones se envía a las posteriores fases del proceso para su procesamiento. La fase de detección es por lo general la fase de mayor coste computacional, y como veremos en la Sección 4.4 también la de menor calidad.
2. Seguidamente, la etapa de binarización crea múltiples imágenes binarias para cada una de las regiones de la placa [47, 48]. Además, se generan imágenes binarias con diferentes umbrales, para maximizar la oportunidad de encontrar todos los caracteres, ya que el resultado depende fuertemente del histograma de la imagen.
3. A continuación se realiza un análisis de caracteres intentando encontrar zonas de tamaño predeterminado según el tamaño original de los caracteres especificados en la patente, el cual ha sido cargado por *default* en la aplicación. **OpenALPR** busca *blobs* conectados en la región central en la imagen binarizada de la placa tentativa. A continuación, busca aquellos elementos que son aproximadamente del ancho y de la altura de un carácter de placa, y que además estén alineadas en línea recta entre si y sean de dimensiones similares. Si no se encuentra nada en la región, entonces la misma es descartada y no sigue con las siguientes etapas, en caso contrario la región pasa la siguiente etapa de análisis.
4. Sobre la posible región donde puede existir una placa de matrícula se trata de identificar los bordes de la misma con mayor precisión, ya que la estimación del **LBP** es en general un poco más grande o más pequeña que la placa real. Para ellos, primero se determina la ubicación de las líneas mediante la transformada de *Hough* [49, 50]. Luego se utiliza esta información, así como la altura de los caracteres para encontrar los bordes más probables. Se utiliza un número de pesos configurables según el formato de la patente para determinar qué borde tiene más sentido.
5. Dado los bordes de la placa, se efectúa una transformación espacial para obtener la imagen de la placa orientada correctamente, eliminando la distorsión ocasionada por la perspectiva de la captura.
6. Para segmentar los caracteres que componen la imagen de la placa, **OpenALPR** utiliza un histograma vertical para encontrar huecos entre los caracteres de la sentencia. Esta etapa también elimina manchas pequeñas desconectadas y descalifica regiones de caracteres que no son suficientemente grandes. Asimismo, se trata de eliminar regiones “borde” de modo que el contexto de la placa no sea mal interpretado como un carácter.

7. La fase de reconocimiento analiza cada carácter independientemente empleado una herramienta externa llamada **TesseractOCR** [51]. Para cada imagen de letra o numero se calcula todos los caracteres posibles y su nivel de confianza.
8. Dada una lista de todos los posibles caracteres y sus niveles de confianza, un procesamiento posterior determina las mejores combinaciones de letras y números en la posible placa.
9. El procesamiento posterior descalifica a todos los caracteres por debajo de un determinado umbral. Además, un analizador de sintaxis verifica que la patente este de acuerdo con el formato de la región geográfica a la que pertenece la placa. Por ejemplo, para la patente única del Mercosur, la expresión regular empleada sería: [A – Z] [A – Z] [0 – 9] [0 – 9] [0 – 9] [A – Z] [A – Z]. Finalmente, son entregados los N resultados mas probables, como lo ilustra la Fig. 4.8.

```
Image name: 9.jpeg
Image size: 680x383
Processing Time: 75.367523
```



```
Plate #1
      Plate  Confidence
*     IA018XI   90.109863
*     AA018XI   85.497215
```

Figura 4.8: Ejemplo de resultados de la API de **OpenALPR**.

4.4. Resultados y discusión

A continuación presentamos los resultados cuantitativos sobre nuestro *dataset* en la Tabla 4.1.

Dataset	Pipeline				OpenALPR			
	Accuracy	Precision	Recall	LS*	Accuracy	Precision	Recall	LS*
Normal	0.38	0.6	0.52	0.46	0.40	0.83	0.43	0.17
Soleado	0.55	0.78	0.65	0.34	0.44	0.79	0.50	0.21
Lluvioso	0.16	0.35	0.23	0.36	0.34	0.75	0.39	0.18
Ruidoso	0.42	0.48	0.76	0.49	0.61	0.90	0.65	0.16

Tabla 4.1: Comparación de resultados. (*) Promedio de la similitud normalizada de *Levenshtein*.

En líneas generales, las prestaciones globales del modelo son iguales o superiores sobre los *datasets* de iluminación normal o sobre-iluminación, mientras que **OpenALPR** se impone sobre los *datasets* en condiciones de lluvia o de capturas con ruido gaussiano.

El funcionamiento del modelo es particularmente bueno en condiciones de sobre-iluminación, donde todas las métricas superan los resultados de **OpenALPR**, a excepción de la precisión, que se encuentra en el mismo orden de magnitud.

Podemos observar que el promedio de la similitud normalizada de *Levenshtein* (**LS**) sobre las predicciones falsas es consistentemente mayor en nuestro *Pipeline* que en **OpenALPR**. Esto también aplica a los valores de *recall*, los cuales solo pierden contra **OpenALPR** en condiciones de lluvia.

Esto se debe a que la mayoría de los fallos en el modelo obedecen a errores en el reconocimiento en uno o mas caracteres de la sentencia. En otras palabras, predominan los **FP**. Se incurren en muy pocos errores de detección, lo que resulta en pocos **FN** y se ve reflejado en mejores valores de *recall* y similitud, en detrimento de la precisión.

Por el contrario, la fuente de errores primordial en **OpenALPR** es la falla de detección en el algoritmo de **LBP**, es decir, predominan los **FN**, lo que hace que la similitud entre sentencias sea menor. A su vez, este mecanismo de fallo, afecta negativamente el *recall* del sistema, y no añade peso sobre la medición de la precisión, la cual es consistentemente mayor que en nuestro modelo, exceptuando el caso del *dataset* sobre-iluminado.

Como puede apreciarse en la Fig. 4.9, el *dataset* lluvioso puede ser particularmente desafiante de leer. Las métricas humanas sobre este conjunto de imágenes no fueron medidas. En particular, el detector es propenso a fallar en imágenes donde la escala de la patente es pequeña en comparación con las dimensiones de la imagen, como es también el caso de la Fig. 4.9. Por otro lado, las Figs. 4.10 y 4.11 muestran que la perspectiva de la captura puede afectar la calidad del reconocimiento, dada una detección correcta.



Figura 4.9: Algunas imágenes de *dataset* lluvioso pueden ser particularmente difíciles de leer.

El tiempo de predicción ronda aproximadamente ~ 1 seg, siendo ligeramente mas rápida la ejecución de **OpenALPR**, aun a pesar del tamaño de las imágenes (sección 4.1). Estos tiempos se corresponden con predicciones sobre una **NVIDIA** Quadro M4000 8GB GDDR5. Véase el Apéndice ???. Mas trabajo es necesario para que el sistema pueda ejecutarse en tiempo real.

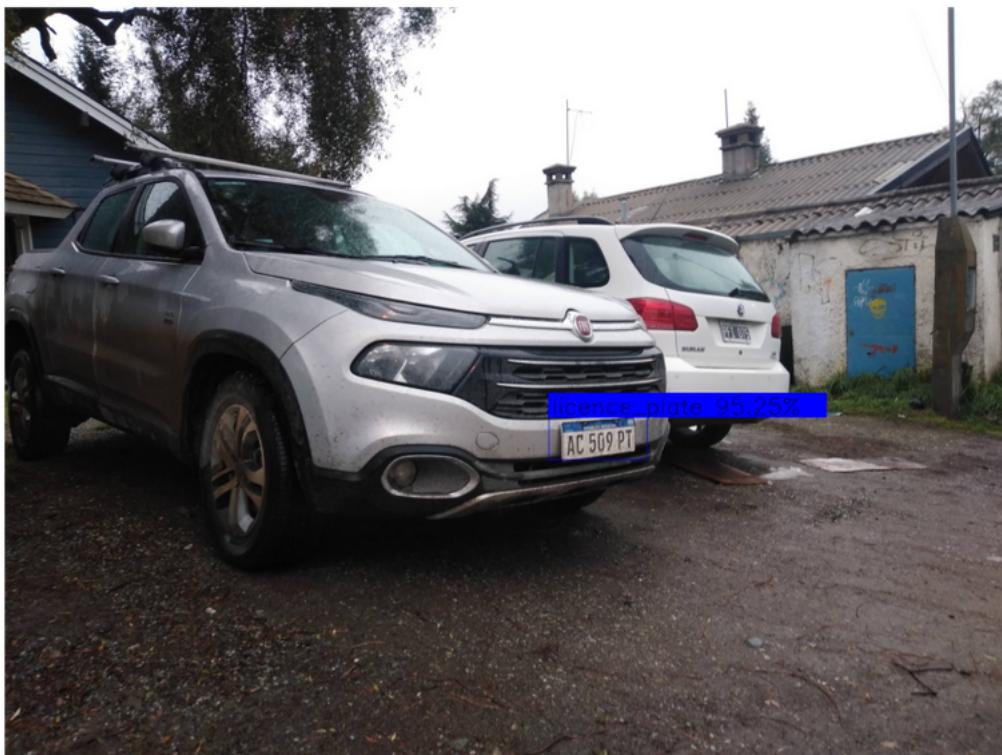
Si bien la precisión del método no ha sido evaluada en un *dataset* que contenga patentes con el formato antiguo de la República Argentina, el sistema es capaz de realizar detecciones de matriculas con este formato, como lo muestra cualitativamente la Fig. 4.12. La ventaja del entrenamiento con imágenes naturales hace suponer que el sistema es capaz de identificar y extraer características del “entorno” de la matricula, haciendo a la red resiliente en cierta medida a diferentes formatos. La detección demasiado amplia hacia los costados es posiblemente debida al efecto que tienen los *anchors* de **YOLOv3** sobre una matricula con otra relación de aspecto. Véase el Capítulo 2.

En la Fig. 4.13 se ha usado **CRAFT** en conjunto con la **RCNN** del Capítulo 3 para efectuar una inferencia sobre un cartel callejero en un contexto acotado. Los resultados son prometedores e incentivan a testear el sistema mas exhaustivamente en *datasets* que contengan imágenes de esta naturaleza.



Predictions: [['ac', '509', 'pt']]

Figura 4.10: Reconocimiento correcto con perspectiva izquierda.



Predictions: [['ac', '501', 'pt']]

Figura 4.11: Reconocimiento incorrecto con perspectiva derecha.



Predictions: [['mea', '959']]

Figura 4.12: Detección y reconocimiento correcto en una patente en formato de la República Argentina.



Predictions: [['castle', 'stirling', 'lodgings', 'argylls']]

Figura 4.13: Detección y reconocimiento correcto de un letrero en un escenario natural acotado. La predicción no ordena las sentencias en el caso mas general donde no se tiene una matricula.



Figura 4.14: Escena con oclusión casi total del vehículo. Formato no presente en el conjunto de entrenamiento.



Figura 4.15: Escena con oclusión casi total del vehículo y oclusión parcial de la matrícula. Formato no presente en el conjunto de entrenamiento.

Como se ha mencionado anteriormente, el sistema de detección entrenado sobre **YOLOv3** parece mostrar mucha robustez. Las Figs. 4.14 y 4.15 muestran cualitativamente la detección de 2 matrículas con formato diferente al conjunto de entrenamiento de la red. En el caso de la Fig. 4.14 la detección es correcta, incluso cuando no vemos el contexto del vehículo en la imagen. Por otro lado, en la Fig. 4.15 la persona en la escena ejerce una oclusión parcial de la matrícula en cuestión, además no verse completamente el vehículo en el contexto.

Finalmente, las Figs. 4.17, 4.16 y 4.18 muestran cualitativamente el reconocimiento de 3 sentencias arbitrarias extraídas de matrículas con diferentes formatos, bajo iluminaciones diferentes. Sentencias del estilo de la Fig. 4.18 fueron filtradas automáticamente para la elaboración de la Tabla 4.1, y así evitar contaminar la salida del reconocedor con información no relevante.



Figura 4.16: Reconocimiento correcto de una sentencia individual.



Figura 4.17: Reconocimiento correcto de una sentencia individual. Las sub-sentencias no son leídas.



Figura 4.18: Reconocimiento correcto de una sentencia individual. Sentencias no relevantes como esta son normalmente filtradas antes de entrar en la red de reconocimiento.

Capítulo 5

Conclusiones y perspectivas

En base a los resultados expuestos en el Capítulo 4, incluimos aquí una breve conclusión. Finalmente, dedicamos la última sección de este trabajo a posibles mejoras a implementarse sobre el sistema, tanto para aumentar la calidad de los resultados, como la velocidad de procesamiento.

5.1. Conclusiones

A lo largo de este trabajo se incursionó en un método para la resolución del problema de **ALPR**, prestando especial atención a poder extender la capacidad de generalización del sistema a otros escenarios. Se espera poder tener una implementación embebida funcionando, al menos parcialmente, en tiempo real en algún momento cercano.

Empezamos nuestro trabajo mediante la investigación del estado del arte de sistemas **ALPR** que emplean redes neuronales en alguna de sus etapas. En base a los trabajos más recientes, pero a su vez teniendo en cuenta la disponibilidad limitada de recursos y de datos de entrenamiento, se delineó una estrategia de 3 etapas, común en la mayoría de los sistemas clásicos de **ALPR**. Véase la Sección 5.2.3 por posibles mejoras sobre este esquema tripartito.

Sobre la base de esta estrategia, se propuso un modelo de segmentación basado en el detector multi-escala **YOLOv3** para efectuar segmentación no-semántica sobre las imágenes de entrada y poder determinar la región que contiene la patente del vehículo. El detector fue entrenado usando un *dataset* relativamente pequeño de patentes europeas segmentadas en imágenes naturales tomadas desde un vehículo en movimiento. Las anotaciones de las imágenes no contienen información sobre los caracteres de las

matriculas. La segmentación a nivel sentencia se realiza mediante el detector **CRAFT**.

Seguidamente, se planteó un sistema que resuelve el problema de reconocimiento como un problema equivalente de clasificación de elementos dentro de una secuencia dada, en este caso, una sentencia de texto impreso. La segmentación de los elementos de la secuencia se realiza de manera automática empleando una etapa convolucional que incluye una transformación espacial **STN**. A continuación, estos *features* alimentan una **RNN** del tipo **LSTM** entrenada con una función de costo del tipo **CTC**. El modelo de reconocimiento se entreno con un subconjunto de imágenes de palabras sintéticas.

Para poder testear el modelo propuesto en un escenario real, se ha ensamblado un conjunto de imágenes de patentes del Mercosur, desde diferentes orientaciones e iluminaciones. Para aumentar la variabilidad dentro del conjunto de testeo, empleamos varias operaciones de transformación no distorsiva sobre el mismo.

Se presentó una serie de métricas estándar con el objetivo de poder realizar una evaluación lo más objetiva posible del modelo implementando y compararlo con otros métodos. A esto se le añade además la métrica de *Levenshtein* como medio para poder evaluar los errores del sistema con mayor detalle e identificar falencias de reconocimiento. Se evaluaron estas métricas contra los resultados obtenidos mediante **OpenALPR**, un estándar comercial de **ALPR** de uso ampliamente difundido hace décadas.

Se evaluaron las métricas determinando que dependiendo de la iluminación de la escena, uno u otro método es susceptible de entregar mejores resultados. El modelo presentado en este trabajo ofrece en líneas generales buenos valores de *recall* y de similitud entre sentencias bajo distintas condiciones. Finalmente, se presentaron algunos resultados cualitativos en patentes antiguas de la República Argentina y carteles urbanos. Por último, se presentaron algunos resultados atípicos, por ejemplo, incluyendo occlusiones parciales.

Más trabajo debe tener lugar para arribar a una aplicación viable, mejorando la *performance* del método y disminuyendo los tiempos de latencia. Siempre bajo la misma metodología modular, cada bloque individual es reemplazable por otros objetos que acepten las mismas entradas, pero potencialmente procesen los datos de manera diferente.

El problema de **ALPR** es sólo una versión de complejidad reducida del problema de lectura de texto visual en condiciones reales, el cual es más apto para ser abordado como un problema de procesamiento de secuencias que por métodos más clásicos de *Computer Vision*.

5.2. Trabajos a futuro

5.2.1. Performance

Es altamente recomendable re-entrenar la red de detección con un *dataset* de patentes locales para mejorar su desempeño, especialmente sobre las matrículas que no tienen el formato común del Mercosur. La arquitectura de *Darknet-53* puede permanecer congelada durante este entrenamiento.

Para mejorar el funcionamiento de la red de reconocimiento es posible implementar una o más capas convolucionales a la entrada, y entrenar sobre imágenes de origen natural que contengan sentencias extraídas de matrículas. Las capas entrenadas con imágenes sintéticas podrían permanecer congeladas para disminuir el tiempo de entrenamiento.

El diccionario de la **RCNN** de reconocimiento podría componerse de *tokens* con combinaciones de letras, pero ya que las patentes no tienen una probabilidad condicional entre caracteres muy marcada (exceptuando tal vez los primeros 2 caracteres de la patente del Mercosur), la mejora podría no ser sustancial. Esta estrategia sería mejor dirigida si se quisiese resolver un problema de reconocimiento de palabras dentro un lenguaje humano en particular.

5.2.2. Latencia

Diversas estrategias son posibles para disminuir la latencia de ejecución del sistema. Por ejemplo, si las imágenes de entrada tuvieran una perspectiva y orientación fijas, no sería necesario emplear la detección multi-escala de **YOLOv3**. O bien se podrían reducir la cantidad de *anchors* empleados por la red, al tratarse de detección de una única clase, con una relación de aspecto fijada para esa escala y perspectiva.

La red de detección puede ser reemplazada por otra versión de **YOLOv3**, llamada **TinyYOLOv3**, la cual puede procesar las imágenes consumiendo menor tiempo en la inferencia. Además, si se busca una implementación en tiempo real de la red se podría implementar un esquema de *tracking* para evitar que la red tenga que realizar la inferencia en todos los *frames* de una secuencia de imágenes.

5.2.3. Mecanismos de atención

Tal vez la mejora de mayor importancia y relevancia que podría ser implementada sería el agregado de un mecanismo diferente de atención visual, que actué procesando los *features* extraídos por la **CNN** antes del ingreso a la **RNN**, ayudando a focalizar la atención en los elementos que son de interés.

Numerosos trabajos [40, 52–54] muestran que es posible construir un sistema completo de reconocimiento *end-to-end* que focalice el procesamiento directamente en las

partes importantes de la imagen. Se sigue de este razonamiento, que las 3 redes podrían ser reemplazadas por una única red que realice todas las funciones, con la desventaja de ser más compleja y costosa de entrenar. Para acotar la complejidad del problema, es posible continuar usando la red de segmentación que detecta la **ROI** con la patente antes de la **RCNN**.

Otros modelos más complejos, comúnmente referidos como *transformers* [55] trabajan con mecanismos de atención diferentes y en años recientes han demostrado poder superar los resultados obtenidos por redes **LSTM** en problemas de reconocimiento de sentencias, como el procesamiento del lenguaje natural y el análisis del habla humana.

En conocimiento del autor, no hay sistemas **ALPR** que utilicen los métodos previamente nombrados. El costo computacional de entrenar estas redes puede ser muy elevado, con las limitaciones que eso conlleva dependiendo de la aplicación particular y los recursos disponibles.

Apéndice A

Esquema

Sobre este apéndice se encuentra un esquema del funcionamiento del modelo desarrollado durante la tesis.

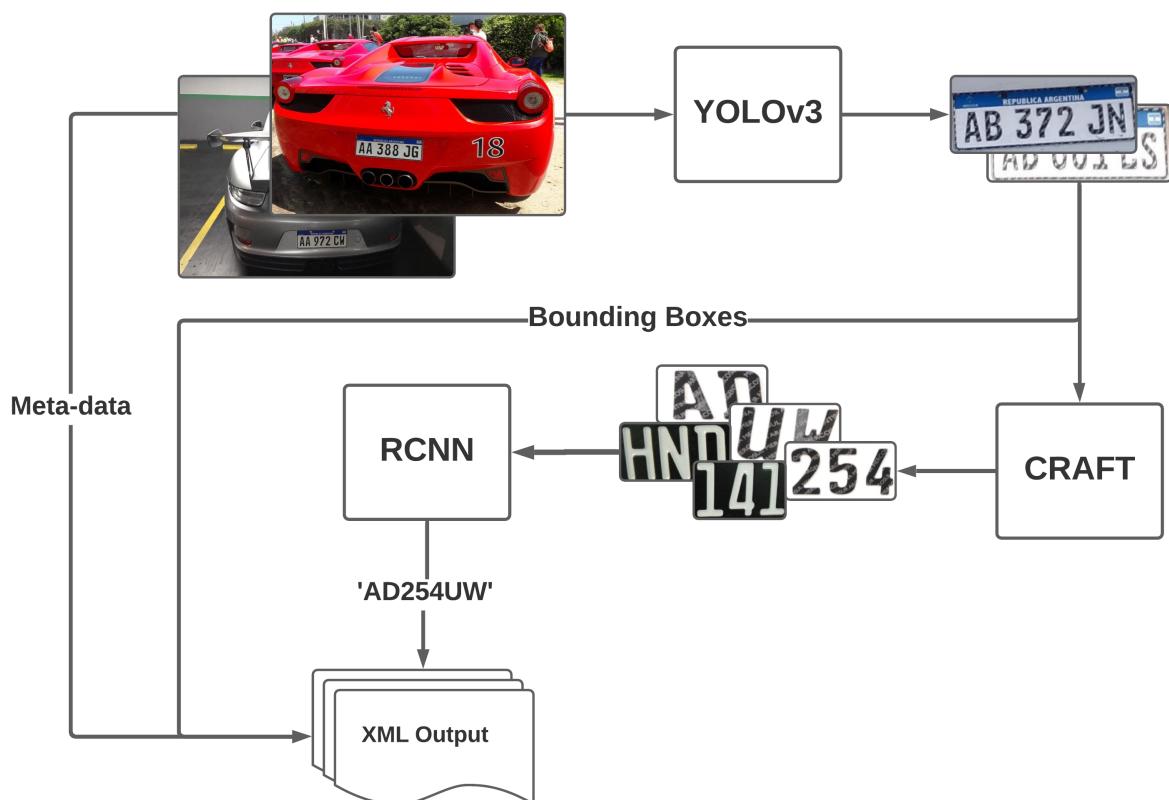


Figura A.1: Diagrama de bloques del modelo desarrollado.

Apéndice B

Hardware

Sobre este apéndice damos algunos detalles del hardware empleado para la implementación de los algoritmos del modelo.

B.1. Hardware principal

El *hardware* empleado para la mayoría de los cómputos de entrenamiento e inferencia que se han realizado en esta tesis es el siguiente:

- **CPU** 2 x Intel(R) Xeon(R) CPU E5-2620 v4 2.10GHz 8 cores por procesador, 16 subprocesos.
- **GPU** NVIDIA Tesla K-40m 12 GB GDDR5
- **Memoria** 32 GB DDR2

Con posterioridad se usó este mismo *hardware* con la única diferencia de que la GPU fue reemplazada por una **NVIDIA** Quadro M4000 8GB GDDR5.

B.2. Jetson Nano

La **NVIDIA Jetson Nano** es un sistema embebido **SoM** (*system-on-module*), de la familia de **NVIDIA Jetson**, el cual incluye una GPU Maxwell 128-núcleos, una CPU quad-core ARM A57 64-bit, memoria de 4GB LPDDR4 y soporte para comunicaciones MIPI CSI-2 y PCIe Gen2.

Pensada para despliegue en aplicaciones de *Computer Vision*, la plataforma provee 472 GFLOPS en precision FP16, con un consumo de energía entre 5 y 10 W. La Fig. B.2 muestra un resumen de las características principales de *hardware* de la plataforma.

Como parte del presente trabajo, se ha incursionado en la aplicación embebida del sistema en dicha plataforma, si bien diversas optimizaciones son aun necesarias para poder ejecutar el modelo enteramente, y mas aun, poder ejecutarlo en tiempo real.



Figura B.1: NVIDIA Jetson Nano [14].

GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 4x 1080p @ 30 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 2x 4K @ 30 8x 1080p @ 30 18x 720p @ 30 (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I ² C, I ² S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector

Figura B.2: Especificaciones del hardware de **Jetson Nano** [14].

Bibliografía

- [1] Redmon, J., Farhadi, A. Yolov3: An incremental improvement. *arXiv:1804.02767*, 2018.
- [2] Eu licence plate dataset. URL <https://github.com/RobertLucian/license-plate-dataset>.
- [3] Deng, J., Dong, W., Socher, R., Li, L., Li, K., Fei-Fei, L. Imagenet: A large-scale hierarchical image database. En: CVPR09. 2009.
- [4] Convnets series: Spatial transformer networks. URL <https://towardsdatascience.com/convnets-series-spatial-transformer-networks-cff47565ae8>
- [5] Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 2012. URL <http://www.sciencedirect.com/science/article/pii/S0893608012000457>.
- [6] Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K. Spatial transformer networks. *arXiv:1506.02025*, 2015.
- [7] W. Feng, Y. L. X. Z. Z. L., N. Guan. Audio visual speech recognition with multimodal recurrent neural networks. En: IEEE International Joint Conference on Neural Networks, págs. 681–688. 2017.
- [8] The vanishing gradient problem. URL <http://harinisuresh.com/2016/10/09/lstms/>.
- [9] Wikipedia: Long-short term memory. URL https://en.wikipedia.org/wiki/Long_short-term_memory.
- [10] Jaderberg, M., Simonyan, K., Vedaldi, A., Zisserman, A. Synthetic data and artificial neural networks for natural scene text recognition. En: Workshop on Deep Learning, NIPS. 2014.
- [11] Stanford university: Sequence modeling with ctc. URL <https://distill.pub/2017/ctc/>.

- [12] Buslaev, A., Iglovikov, V., Khvedchenya, E. Albumentations: Fast and flexible image augmentations, 2020. URL <https://www.mdpi.com/2078-2489/11/2/125>.
- [13] Evaluating object detection models: Guide to performance metrics. URL <https://manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html>.
- [14] Jetson nano developer kit. URL <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>.
- [15] Open alpr. URL <https://github.com/openalpr/openalpr>.
- [16] Du, S., Ibrahim, M., Shehata, M., Badawy, W. Automatic license plate recognition (alpr): A state-of-the-art review. In: IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 2, págs. 311–325. IEEE, 2013.
- [17] Gou, C., Wang, K., Yao, Y., Li, Z. Vehicle license plate recognition based on extremal regions and restricted boltzmann machines. In: IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 4, págs. 1096–1107. 2016.
- [18] Montazzolli, S., Jung, C. R. Real-time brazilian license plate detection and recognition using deep convolutional neural networks. In: 30th SIBGRAPI Conference on Graphics, Patterns and Images, págs. 55–62. 2017.
- [19] Hsu, G. S., Ambikapathi, A., Chung, S. L., Su, C. P. Robust license plate detection in the wild. In: 14th IEEE International Conference on Advanced Video and Signal Based Surveillance, págs. 1–6. IEEE, 2017.
- [20] Rafique, M. A., Pedrycz, W., Jeon, M. Vehicle license plate detection using region-based convolutional neural networks, 2017.
- [21] Li, H., Shen, C. Reading car license plates using deep convolutional neural networks and lstms. *arXiv:1601.05610*, 2016.
- [22] Laroca, R., Severo, E., Zanlorensi, L. A., Oliveira, L. S. A robust real-time automatic license plate recognition based on the yolo detector. *arXiv:1802.09567v6*, 2018.
- [23] Bulan, O., Kozitsky, V., Ramesh, P., Shreve, M. Segmentation and annotation-free license plate recognition with deep localization and failure identification. In: IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 9, págs. 2351–2363. IEEE, 2017.

- [24] Menotti, D., Chiachia, G., Falcão, A. X., Neto, V. J. O. Vehicle license plate recognition with random convolutional networks. En: 27th SIBGRAPI Conference on Graphics, Patterns and Images, págs. 298–303. 2014.
- [25] Redmon, J., Farhadi, A. You only look once: Unified, real-time object detection. En: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016.
- [26] Redmon, J., Farhadi, A. Yolo9000: Better, faster, stronger. *arXiv:1612.08242*, 2016.
- [27] He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.
- [28] Yolov3 theory explained. URL <https://pylessons.com/YOL0v3-introduction/>.
- [29] Darknet repository. URL <https://github.com/pjreddie/darknet>.
- [30] Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [31] Spatial transformer network for tensorflow. URL https://github.com/skaae/transformer_network.
- [32] Bengio, Y., Simard, P., Frasconi, P. Learning long-term dependencies with gradient descent is difficult. En: IEEE Transactions on Neural Networks, vol. 5, no. 2. 1994.
- [33] Gers, F., Schmidhuber, J., Cummins, F. Learning to forget: Continual prediction with lstm. En: Neural Computation, vol. 12, no. 10, págs. 2451–2471. 2000.
- [34] Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J. A novel connectionist system for unconstrained handwriting recognition. En: IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, págs. 855–868. 2009.
- [35] Graves, A., Fernandez, S., Gomez, F., Schmidhuber, J. Connectionist temporal classification : Labelling unsegmented sequence data with recurrent neural networks. En: Proceedings of the 23rd international conference on Machine Learning, págs. 369–376. 2006.
- [36] Liwicki, M., Graves, A., Bunke, H., Schmidhuber, J. A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks. En: Proceedings of the 9th Int. Conf. on Document Analysis and Recognition, págs. 367–371. 2007.

- [37] Graves, A. Supervised Sequence Labelling with Recurrent Neural Networks. Springer, 2012.
- [38] Graves, A. Sequence transduction with recurrent neural networks. *arXiv:1211.3711*, 2012.
- [39] Tensorflow repository. URL <https://github.com/tensorflow/tensorflow>.
- [40] Xu, K., Lei, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., *et al.* Show, attend and tell: Neural image caption generation with visual attention. En: Proceedings of the 32nd International Conference on Machine Learning, vol. 37, no. 32. 2015.
- [41] Baek, Y., Lee, B., Han, D., Yun, S., Lee, H. Character region awareness for text detection. *arXiv:1904.1941*, 2019.
- [42] Kuznetsova, A. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv:1811.00982*, 2018.
- [43] Moretti, I., Jorge, J., Amado, J. Software libre para reconocimiento automático de las nuevas patentes del mercosur.
- [44] Especificaciones tecnicas - patente unica mercosur. URL <http://www.casademoneda.gob.ar/wp-content/uploads/2019/08/ESPEC.-TECNICAS2.pdf>.
- [45] He, D., Wang, L. Texture unit, texture spectrum, and texture analysis. En: IEEE Transactions on Geoscience and Remote Sensing, vol. 28, págs. 509–512. 1990.
- [46] Wang, L., He, D. Texture classification using texture spectrum. En: Pattern Recognition, vol. 23, no. 8, págs. 905–910. 1990.
- [47] Wolf, C., Jolion, J. M. Extraction and recognition of artificial text in multimedia documents. En: Pattern Analysis and Applications, vol. 6, no. 4, págs. 309–326. 2003.
- [48] Sauvola, J., Seppanen, T., Haapakoski, S., Pietikainen, M. Texture classification using texture spectrum. En: 4th International Conference On Document Analysis and Recognition, págs. 147–152. 1997.
- [49] Hough, P. U.s. patent 3 069 654: Method and means for recognizing complex patterns, 1962.
- [50] Duda, R., Hart, P. Use of the hough transformation to detect lines and curves in pictures. En: ACM Communications, vol. 15, págs. 11–15. 1972.

- [51] Tesseract ocr. URL <https://github.com/tesseract-ocr/tesseract>.
- [52] Wojna, Z., Gorban, A., Lee, D. S., Murphy, K. Attention-based extraction of structured information from street view imagery. *arXiv:1704.03549*, 2017.
- [53] He, T., Huang, W., Qiao, Y., Yao, J. Text-attentional convolutional neural network for scene text detection. En: IEEE Transactions on Image Processing, vol. 25, no. 6, págs. 2529–2541. 2016.
- [54] Lee, C., Osindero, S. Recursive recurrent nets with attention modeling for ocr in the wild. págs. 2231–2239, 2016.
- [55] Polosukhin, I., Kaiser, L., Gomez, A. N., Jones, L. Attention is all you need. *arXiv:1706.03762*, 2017.

