UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

RECONOCIMIENTO DE MONTOS MANUSCRITOS EN CHEQUES A TRAVÉS DE MODELOS DE DETECCIÓN DE OBJETOS BASADOS EN REDES CONVOLUCIONALES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

DAVID ALBERTO SAJI SANTANDER

PROFESOR GUÍA:
JOSÉ SAAVEDRA RONDO

MIEMBROS DE LA COMISIÓN:
ALFREDO SCHNELL DRESEL
ALEXANDRE BERGEL
ADOLFO CARRASCO ACOSTA

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN;INGENIERO CIVIL ELÉCTRICO
POR: DAVID ALBERTO SAJI SANTANDER
FECHA: 2019
PROF. GUÍA: JOSÉ SAAVEDRA RONDO

RECONOCIMIENTO DE MONTOS MANUSCRITOS EN CHEQUES A TRAVÉS DE MODELOS DE DETECCIÓN DE OBJETOS BASADOS EN REDES CONVOLUCIONALES

El problema de reconocimiento de texto manuscrito en imágenes es muy importante en cuanto a aplicaciones en el área de visión por computadora. Un problema más acotado, pero no menos importante, es el de reconocimiento de secuencias de dígitos manuscritos de largo variable en imágenes.

El presente trabajo consiste en investigar un nuevo enfoque para resolver el problema de reconocimiento de secuencias de dígitos manuscritos en imágenes. Se plantea dicho problema como uno de detección de objetos en que los dígitos corresponden a los objetos a detectar. Los algoritmos utilizados corresponden al estado del arte en detección de objetos siendo estos: RetinaNet (2017), YOLOv3 (2018) y FCOS (2019).

La hipótesis de este trabajo es que un enfoque basado en detección de objetos logra tasas de reconocimiento superiores a los enfoques actuales basados en modelamiento secuencial, en particular, aquellos basados en la CTC.

Los resultados obtenidos confirman la hipótesis de que un enfoque basado en detección de objeto es capaz de superar al estado del arte cuyo mejor resultado se basa en una arquitectura de tipo CNN-LSTM-CTC la cual alcanza tasas de reconocimiento de 89.75 % y 91.14 % en los datasets ORAND-CAR-A y ORAND-CAR-B, respectivamente. Con el nuevo enfoque propuesto basado en detección de objetos, en particular YOLO, se alcanzan tasas de reconocimiento superiores al 96 %, específicamente **96.78 %** y **96.45 %** en ORAND-CAR-A y ORAND-CAR-B, respectivamente. Estos excelentes resultados marcan sin duda un nuevo precedente en el problema de reconocimiento de secuencias de dígitos manuscritos en imágenes.

Por último, en cuanto a las aplicaciones del modelo entrenado, se logra una mejora significativa con respecto al modelo previo utilizado en ORAND S.A. (empresa interesada en este trabajo y que facilitó los datasets). Mientras el modelo antiguo, que hacía uso de técnicas tradicionales de procesamiento de imágenes, alcanzaba un desempeño cercano al 80 % en los datasets ORAND-CAR-A y ORAND-CAR-B, y con un tiempo de procesamiento de 1FPS, el actual modelo alcanza tasas superiores al 96 % y tiempos cercanos a los 100FPS.

Palabras clave: *digit string recognition, convolutional neural network, recurrent neural network, object detection, you only look once, connectionist temporal clasification.*

A mis padres

Tabla de Contenido

Introducción

En los último años, la inteligencia artificial, y en particular, el *deep learning*, ha progresado considerablemente, y problemas que hace algunos años eran todo un desafío, como por ejemplo, el problema de clasificación de *ImageNet* [22], donde los primeros modelos (2011) llegaban a tasas de error que bordeaban el 30 %, en la actualidad se cuenta con modelos que superan al mismo ser humano con tasas de error menores al 5 %. Muchos de los problemas que resuelve el *deep learning* provienen de una necesidad del mundo real. Solo por mencionar algunos casos de éxito del *deep learning* aplicados en la industria: coches autónomos, asistentes personales inteligentes, modernización de la cadena de suministro, detección de fraudes e incluso en la industria alimentaria.

Desde hace tiempo ciertos sectores de la industria han visto la necesidad de automatizar sus procesos, en particular, el respectivo al análisis de documentos. Este es el caso del sector bancario el cual durante años buscó la forma de agilizar el proceso de reconocimiento automático de cheques, en particular, el monto de estos. Más aún, con la tendencia actual a la digitalización y el canje electrónico la importancia del problema es mucho mayor.

Los primeros intentos para resolver este problema, antes que surgieran las técnicas modernas de *machine learning*, se basaban en técnicas tradicionales de procesamiento de imágenes, siendo las más utilizadas las basadas en segmentación. Dichas técnicas durante años fueron el estado del arte hasta que en 1998 LeCun et al. [9] propuso un modelo basado en las llamadas redes neuronales convolucionales (CNN), las cuales son descritas más adelante, y lo aplicó al reconocimiento de dígitos siendo tal su éxito que fue utilizado por varios bancos para el reconocimiento de cheques. Sin embargo, pese a este logro, se siguieron utilizando los métodos basados en segmentación puesto que el modelo de LeCun solo funcionaba para dígitos, además que era costoso computacionalmente para la época.

Así, durante mucho tiempo los métodos del estado del arte se basaron en técnicas como segmentar las imágenes en partes, las cuales eran clasificadas y finalmente se combinaban los resultados con algoritmos de búsqueda de caminos (*path-search*) para obtener el resultado final. Estos algoritmos tiene la desventaja que requieren de mucha heurística y conocimiento de la información del contexto a fin de generar segmentos adecuados, siendo métodos muy sensibles incluso a pequeñas variaciones de los casos considerados como los estilos de escritura, que tan pegados están los caracteres y el ruido de fondo. Esto hace que estos métodos sean poco usados en la práctica.

Con el aumento del poder de cómputo a lo largo de los años y la aparición de mejores modelos basados en redes convolucionales el problema de reconocimiento de dígitos aislados

quedó prácticamente resuelto con tasas de error inferiores al 1 %. Sin embargo, no ocurre lo mismo con el problema de reconocimiento de números con una cantidad arbitraria de dígitos, el cual, hasta hoy en día, sigue siendo un tópico abierto en el área de análisis de documentos. Las principales dificultades en este problema que contribuyen a desempeños no satisfactorios de los métodos disponibles en la literatura se relacionan usualmente con la alta variabilidad de estilos de escritura, el largo arbitrario de la secuencia de dígitos, la presencia de dígitos cuyos trazos son discontinuos, se traslapan o se tocan con otros dígitos, además de la naturaleza del ruido presente en las imágenes obtenidas del mundo real (diseño del documento o trazos ruidosos). En estos casos, la solución estándar de segmentar la imagen en componentes conexas donde cada una representa un dígito se vuelve infactible.

En los últimos años, con el desarrollo de nuevas técnicas basadas en *deep learning*, se están buscando nuevas soluciones a este problema libres de segmentación. La ventaja de este enfoque basado en *deep learning* es que el costo computacional es alto solo en el entrenamiento, pero una vez entrenado el modelo es muy bajo al evaluar, mientras que con los métodos de segmentación no hay entrenamiento y todo el costo computacional se produce al evaluar.

Actualmente se han presentado avances con redes convolucionales (CNN) reportándose tasas de reconocimiento sobre el 90 %. Sin embargo, hasta ahora, los distintos autores han hecho suposiciones restrictivas, como por ejemplo, solo reconociendo números de largo fijo o que hayan a lo más 2 dígitos superpuestos, etc. Por ejemplo, uno de estos trabajos es el presentado por Choi et al. [21], que entrenó una red modular compuesta de 100 subredes separadas para reconocer números de 2 dígitos, donde cada subred es responsable de reconocer una de las 100 clases. Así, la primera subred se especializa en reconocer el string *00*, la segunda el string *01*, y así sucesivamente hasta la red 100 responsable de reconocer el string *99* (ver figura 1). Con esta red se reporta un 95.3 % de aciertos sobre el dataset NIST.

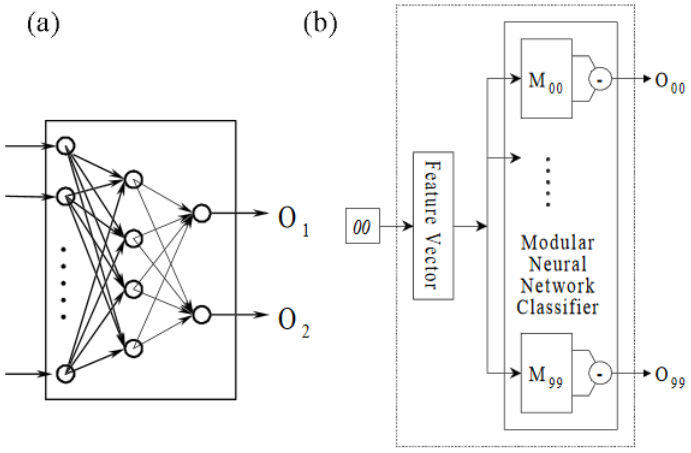


Figura 1: Clasificador propuesto por Choi. (a) una subred (b) red completa.

Dado que las estrategias estudiadas hasta ahora se basan en hacer distintos supuesto respecto a los dígitos, el problema de reconocer números manuscritos sin hacer suposiciones restrictivas sigue estando abierto. Por esta razón, el año 2014 se anunció el concurso HDSRC 2014 [2] (*Competition on Handwritten Digit String Recognition in Challenging*) el cual buscaba identificar, evaluar y comparar los avances recientes en el reconocimiento de cadenas de

dígitos manuscritos de largo variable. Para ello se liberaron 2 datasets llamados ORAND-CAR-A y ORAND-CAR-B cuyas imágenes se obtuvieron del campo CAR (*Courtesy Amount Recognition*) de cheques reales. El campo CAR es el nombre técnico que designa al monto o valor numérico del cheque. En dicha competencia, el método ganador fue el propuesto por Wu et al. [2] con un 80.7 % y 70.1 % en ORAND-CAR-A y ORAND-CAR-B, respectivamente.

En la actualidad, el estado del arte en el problema de reconocimiento CAR corresponde a Zhan et al. [27] que alcanzó una tasa de reconocimiento (*accuracy*) de 89.8 % y 91.1 % en ORAND-CAR-A y ORAND-CAR-B, respectivamente. Zhan propone una usar un novedoso método de *deep learning* llamado CNN-LSTM-CTC el cual es descrito más adelante.

Al igual que el problema HDSR (*Handwritten Digit String Recognition*), el de detección de objetos ha avanzado considerablemente en los últimos años también gracias al *deep learning*, y en la actualidad se cuenta con excelentes modelos de detección como, por ejemplo, Faster R-CNN [17], YOLO [14] y RetinaNet [11].

De este modo, cabe preguntarse si será posible abordar el problema HDSR como uno de detección de objetos, en donde los objetos a detectar corresponden a los dígitos. Es así como nace este trabajo como un intento de responder a la pregunta de cómo funcionaría un detector de objeto moderno aplicado al reconocimiento de montos y cómo se compararía con los métodos del estado del arte de reconocimiento de secuencias numéricas.

El objetivo de este trabajo es investigar el impacto de los métodos actuales de detección de objetos en el problema de reconocimiento CAR. Para ello son aplicados los algoritmos de detección de objetos basados en *deep learning* de una etapa, siendo estos los que presentan mejor *trade-off* entre desempeño y rapidez. En particular, se utilizan los siguientes métodos: YOLO [14], RetinaNet [11] y FCOS [23].

La contribuciones en este trabajo son las siguientes:

1. Se propone un nuevo enfoque basado en detección de objetos con redes convolucionales para resolver para el problema de reconocimiento de secuencias de dígitos manuscritos en imágenes el cual supera considerablemente al estado del arte tanto en la academia como en la industria. La tabla 1 muestra una comparación del desempeño alcanzado por los distintos métodos medido por el *accuracy* (porcentaje de imágenes reconocidas correctamente).
2. Se compara el desempeño de los distintos métodos tanto propuestos como del estado del arte aportando valiosa información a la comunidad que estudia el problema de reconocimiento de secuencias de dígitos en imágenes.
3. Los modelos entrenados permiten la puesta en producción de un nuevo sistema para el reconocimiento CAR en cheques reales por parte de la empresa interesada en esta investigación (ORAND S.A.). Esto gracias a sus excelentes resultados muy superiores al sistema anterior basado en técnicas tradicionales de procesamiento de imágenes.

Tabla 1: Comparación en datasets ORAND-CAR-A y ORAND-CAR-A.

Métodos	CAR-A	CAR-B
Estado del Arte en la academia (Zhan [27])	89.75 %	91.14 %
Estado del Arte en la industria (ORAND)	80.0 %	
Propuesta (YOLO)	**96.78**	**96.45 %**

Este trabajo se ha organizado en 5 capítulos:

- El Capítulo 1 corresponde al marco teórico.
- El Capítulo 2 presenta la hipótesis y los objetivos de este trabajo.
- El Capítulo 3 profundiza en el desarrollo describiendo las arquitecturas diseñadas, las herramientas programadas, los datasets empleados y el procesamiento de los datos.
- En el Capítulo 4 se presentan los resultados experimentales y se muestran ejemplos de predicciones correctas y fallidas.
- El Capítulo 5 consiste en un análisis y discusión de los resultados obtenidos.
- Por último, se presentan las conclusiones de este trabajo.

Capítulo 1

Marco Teórico

1.1. Procesamiento de cheques

El procesamiento automático de un cheque bancario se puede dividir principalmente en dos tipos de problemas: CAR (*Courtesy Amount Recognition*) y LAR (*Legal Amount Recognition*). CAR es el proceso de reconocimiento del valor numérico del monto del cheque, mientras que LAR es el reconocimiento del valor escrito en palabras. La figura 1.1 muestra las distintas partes de un cheque bancario.

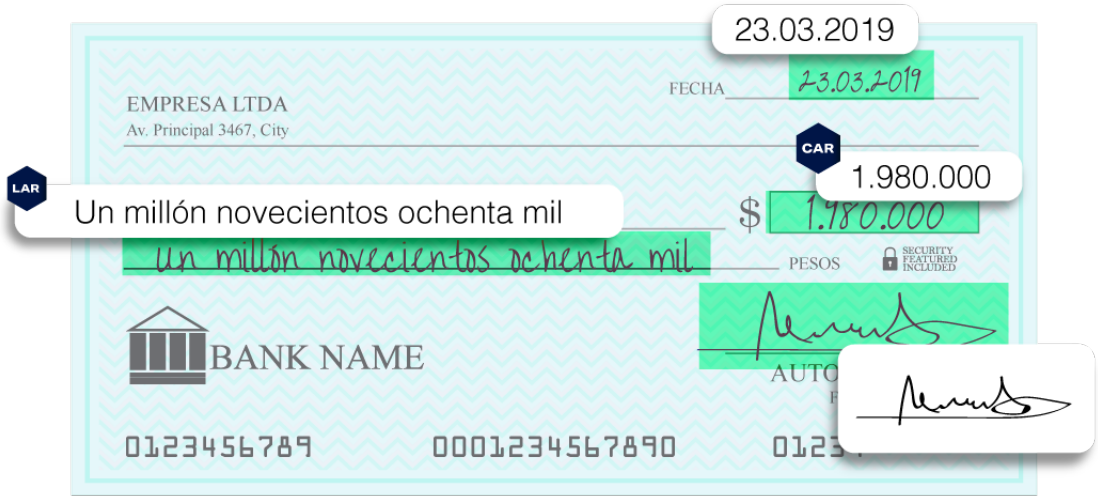


Figura 1.1: Procesamiento de cheques.

Se han propuesto distintos algoritmos para resolver el problema de reconocimiento CAR, pero los resultados no han sido del todo satisfactorios. Entre los métodos tradicionales basados en segmentación de las imágenes, existe un enfoque interesante que reduce la heurística necesaria para obtener los cortes óptimos el cual está basado en lo que se denomina sobre-segmentación (*over-segmentation*). En esta técnica se segmenta la imagen, tanto como sea necesario, en componentes que representan dígitos o partes de estos. Luego de reconocer cada componente o su combinación, el algoritmo calcula el resultado íntegro más probable (*path-*

search algorithms). Otro enfoque es el de Wu et al. [1] que transformaba la imagen en una secuencia de segmentos de imágenes post una binarización de la misma, para luego combinar dichos segmentos generando patrones de caracteres candidatos. Por último, un algoritmo de *beam search* se usaba para encontrar el camino óptimo sobre el *lattice* de candidatos. Este método ganó el primer lugar en la competición ICFHR2014 HDSR [2]. Por otra parte, Saabni [18] utilizó una ventana deslizante y una red neuronal en su modelo, y Gattal et al. [3] aplicó 3 métodos de segmentación dependientes del tipo de conexión entre dígitos.

El problema con estos métodos es que, si bien obtienen buenos desempeños en condiciones ideales, tiene un alto costo computacional ya que implican muchas operaciones de segmentación, además que, en la práctica, son muy sensibles al estilo de escritura, conexión de caracteres o el ruido de fondo (tanto del *layout* del cheque como de los trazos), haciendo de estos métodos poco robustos y confiables.

La tabla 1.1 resume el estado del arte en el problema de reconocimiento CAR en 3 datasets: ORAND-CAR-A (abreviado CAR-A), ORAND-CAR-B (abreviado CAR-B) y CVL. Los primeros 5 resultados fueron presentados en la competencia HDSRC 2014 [2], mientras que los 2 últimos fueron presentados con posterioridad.

Tabla 1.1: Estado del arte en el problema de reconocimiento CAR.

Métodos	CAR-A	CAR-B	CVL
Tebessa I [2]	0.3705	0.2662	0.5930
Tebessa II [2]	0.3972	0.2772	0.6123
Singapore [2]	0.5230	0.5930	0.5040
Pernambuco [2]	0.7830	0.7543	0.5860
BeiJing [2]	0.8073	0.7013	0.8529
CRNN [19]	0.8801	0.8979	0.2601
Saabni [18]	0.8580		-
Zhan [27]	0.8975	0.9114	0.2707

Como se puede notar de la tabla 1.1 el mejor resultado alcanzado hasta la fecha corresponde a Zhan et al. [27] con 89.8 % y 91.1 % en CAR-A y CAR-B respectivamente. Zhan propone usar un novedoso método de *deep learning* el cual consiste en la combinación de una red convolucional (CNN) residual con una red recurrente (LSTM bidireccional) y por último, una capa de salida CTC. Los resultados alcanzados por Zhan son sobresalientes en comparación a los modelos tradicionales anteriores basados en segmentación y/o clasificadores superficiales. La arquitectura CNN-LSTM-CTC es considerada hasta la fecha el estado del arte en el reconocimiento de texto manuscrito.

1.2. Detección de objetos

El problema de detección de objetos en imágenes tiene una larga historia desde que se empezó a popularizar el uso masivo de la información multimedia. Este problema está asociado al área de visión por computadora y procesamiento de imágenes, y consiste en detectar instancias de objetos pertenecientes a una cierta clase (por ejemplo, caras, personas, autos,

etc.) contenidos en una imagen o en un vídeo (el cual se puede ver como una secuencia de imágenes). El objetivo de la detección de objetos es, por una parte, localizar todos los objetos de interés en una imagen y posteriormente clasificarlos. Nótese que este problema va un paso más allá en cuanto a complejidad respecto al de clasificación de imágenes ya que no solo se requiere clasificar los objetos de interés sino que también se deben localizar en la imagen, lo cual significa predecir su *bounding box*, o sea, el rectángulo mínimo que lo contiene tal como se muestra en la figura 1.2.

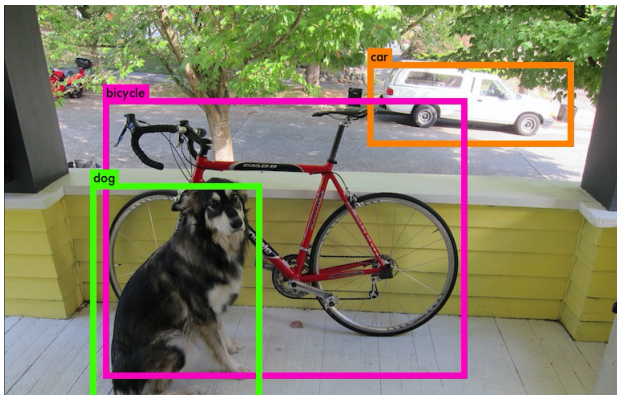


Figura 1.2: Detección de objetos en acción.

La relevancia del problema de detección de objetos es sin duda su potencial de aplicaciones, entre las cuales están, por mencionar algunas, la detección los rostros en una imagen, cámaras de vigilancia que detectan peatones, el seguimiento del balón en un partido de fútbol e incluso automóviles autónomos.

El concepto detrás de este problema es que cada objeto perteneciente a una misma clase tiene un conjunto particular de características que lo distingue del resto de objetos de las demás clases. Estas características permiten clasificar dicho objeto.

Las técnicas modernas de detección de objetos se pueden clasificar en 2 grupos principalmente: métodos basados en *machine learning* y los basados en *deep learning*. Bajo el enfoque de *machine learning* se requiere definir primero las características de los objetos a detectar, para lo cual se suelen usar descriptores locales tales como: Haar, HOG, SIFT, entre otros. Luego, dichas características pasan por un clasificador como por ejemplo SVM. Durante mucho tiempo este enfoque fue el dominante hasta que llegaron de los métodos basadas en *deep learning*.

Mientras los métodos basados en *maching learning* requieren de la llamada *ingeniería de características* la cual es una rama de la ciencia de los datos encargada del diseño de las características apropiadas a cada problema, los modelos basado en *deep learning* no requieren de este paso ya que estas son extraídas automáticamente por el mismo modelo. En el caso de los problemas con imágenes los método más extendidos son aquellos basados en las llamadas redes neuronales convolucionales o CNN por sus siglas en inglés. El impacto de estas redes es tal que permitió el uso industrial de los detectores de objetos por sus excelentes resultados. La explicación de tal éxito es debido a la complejidad de tales redes, algunas de las cuales llegan a tener cientos de capas, en comparación a los modelos tradicionales que tienen arquitecturas más superficiales por lo que su desempeño se estanca. De este modo, las redes convolucionales

profundas son capaces de aprender las características óptimas para el problema a resolver, combinando características de bajo nivel (baja información semántica) en las primeras capas y de alto nivel (alta información semántica) en las capas más profundas. Esto hace que estas redes sean entrenables de extremo a extremo, ya que solo necesitan de imágenes de entrada y sus respectivas anotaciones.

El concepto de CNN data de 1998 cuando LeCun et al. [9] aplicó una de estas redes conocida como LeNet-5 al reconocimiento de dígitos siendo tal su éxito que fue utilizada por varios bancos para el reconocimiento de cheques. Este es considerado el primer caso de éxito de las CNNs, sin embargo, lejos de los que se podría pensar, estos modelos cayeron en el olvido durante mucho tiempo (2 décadas aproximadamente) principalmente debido al costo computacional que significaba entrenar tales modelos, el cual era bastante para los computadores de la época. Fue con las nuevas implementaciones con GPUs (*graphics processing units*) que estos modeles comenzaron una revolución en el área de la visión por computadora. La primera de estas implementaciones data del 2006 en que K. Chellapilla et al. reporta que su implementación era 4 veces más rápida que su equivalente en CPU. En 2010, Dan Ciresan et al. at IDSIA demostró que incluso redes profundas con muchas capas pueden ser entrenadas rápidamente con una GPU. Su red superó con creces los métodos del estado del arte basados en *machine learning* en el problema de reconocimiento de dígitos de MNIST [1]. En 2011 utilizaron una CNN implementada en GPU y ganaron un concurso de reconocimiento de imágenes siendo la primera vez que un modelo alcanza desempeños superhumanos. Posteriormente, una CNN similar permitió a Alex Krizhevsky et al. ganar en 2012 la competencia de reconocimiento visual a gran escala ImageNet (ILSVR) [22].

Antes del surgimiento del *deep learning*, o mejor dicho, resurgimiento, la tarea de detección de objetos era abordada por los llamados métodos tradicionales, entre los cuales está la costosa estrategia conocida como ventana deslizante o *sliding window*, donde rectángulos o ventanas de diferentes tamaños recorren toda la imagen intentando encontrar objetos relevantes. Para este propósito se aplica un clasificador por cada una de estas ventanas, pero dada la cantidad de veces en que hay que aplicar dicho clasificador, este algoritmo resulta ser extremadamente costoso computacionalmente. Si bien este método tiene un buen desempeño, es debido a su lentitud que en la práctica resulta ser completamente infactible. Es este concepto de la ventana deslizante en que se basan las CNN, pero de un modo mucho más eficiente.

1.3. Redes Neuronales Convolucionales

En *deep learning* las redes neuronales convolucionales o más comúnmente CNNs por sus siglas en inglés (*Convolutional Neural Network*) son un tipo especial de red neuronal adaptada al procesamiento de imágenes. Las CNNs se aplican a reconocimiento y clasificación de imágenes, así como también al problema de detección de objetos, entre otros.

La idea de estas redes es aplicar varios filtros o *kernels* entrenables a una imagen de modo de extraer características de esta. Por lo general, una CNN se compondrá de capas convolucionales las cuales recibirán las características o *features* de la capa previa y aplicará

más filtros o convoluciones y cuyo resultado se lo pasará a la capa siguiente. La figura 1.3 muestra la arquitectura típica de una red convolucional.

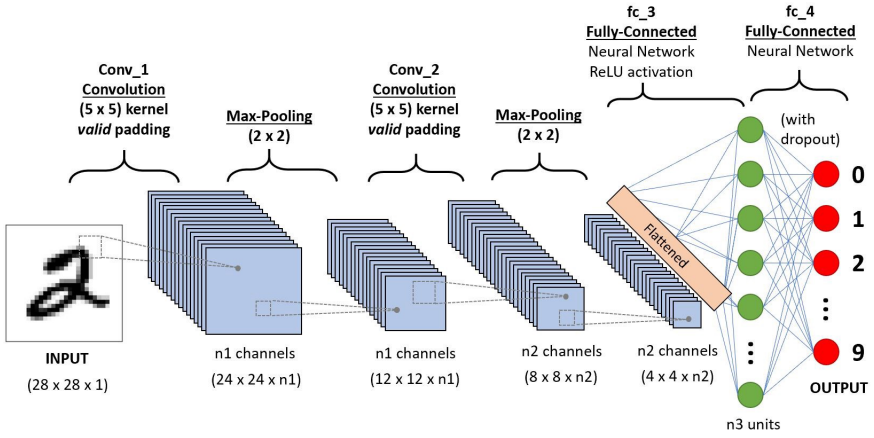


Figura 1.3: Arquitectura de una red convolucional típica.

Por el hecho que las CNNs son capaces de aprender características altamente discriminativas, es posible explotarlas en otros contextos además de clasificación. Estas redes se han aplicado exitosamente al problema de búsqueda por similitud, detección de objetos, segmentación semántica, etiquetado de imágenes, entre otros.

Un concepto importante asociado a las neuronas de una CNN es el de campo receptivo, el cual se define como la región local de la imagen de entrada que afecta a una cierta neurona. Por ejemplo, una neurona de la primera capa de una CNN que aplica una convolución de 3x3 tendrá un campo receptivo de tamaño 3x3 en la imagen de entrada, mientras que una en la capa siguiente verá una porción de tamaño 5x5 suponiendo un *stride* (separación entre la aplicación de los kernels) de 1.

Un componente comúnmente usado en las CNNs es el *pooling*, el cual es un tipo especial de capa que se suele aplicar entre 2 capas convolucionales sucesivas, y tiene como función reducir la dimensionalidad espacial de la imagen de entrada. De este modo la capa de *pooling* ayuda a reducir el número de parámetros, además que logra aumentar el campo receptivo de las neuronas ubicadas en capas más profundas ya que al reducir la imagen de entrada a, por ejemplo, la mitad, las neuronas de la capa siguiente tendrán un campo receptivo amplificado al doble, de modo que con unas pocas capas se logra tener un campo receptivo suficientemente grande para que las neuronas puedan aprender los patrones globales de las imágenes.

1.4. Arquitecturas estándar

A lo largo de los últimos años se han propuesto varias arquitecturas de redes convolucionales. Por citar algunos ejemplos de las más famosas: LeNet-5 [9], AlexNet [22], VGG [20], ResNet [7].

La figura 1.4 muestra una tabla comparativa de dichas redes. En esta se muestran sus

nombres, año de desarrollo, los creadores, premios en competiciones, *performance* en términos de tasa de error y finalmente el número de parámetros. Nótese que conforme pasan los años se han desarrollado mejores modelos lo cual se relaciona directamente con el incremento en la capacidad de cómputo. Si antes se entrenaba en CPU lo cual limitaba considerablemente la complejidad del modelo (medida en el número de parámetros a entrenar), en la actualidad es común entrenar en GPU, e incluso *clusters* de estas, lo cual permite entrenar grandes redes con millones de parámetros.

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

Figura 1.4: Tabla comparativa entre distintas arquitecturas de CNNs.

En lo que sigue se centra la atención en ResNet que, en la actualidad, es la arquitectura más usada para resolver problemas complejos de visión computacional.

1.5. ResNet

ResNet [7] es una arquitectura de red convolucional la cual ganó el primer lugar en la competencia de clasificación de imágenes ILSVR 2015 con un error top-5 de 3.57 %, también ganó la competencia COCO 2015 en los problemas de clasificación, detección y segmentación. Además, sustituyendo VGG-16 como red *backbone* (red principal encargada de extraer características de las imágenes) por una ResNet-101 (101 capas) en Faster R-CNN (un tipo de detector de objetos del cual se hablará más adelante) se obtiene una mejora de 28 %.

La importancia de las redes residuales es que resuelven en parte el problema del desvanecimiento del gradiente (*vanishing gradient*). El problema del desvanecimiento del gradiente se da cuando redes profundas comienzan a converger, así, a medida que las redes se hacen más profundas el desempeño se satura y degrada rápidamente. El problema reside en que en la etapa de *backpropagation* los pesos de la red se actualizan de forma proporcional a la derivada del error, de modo que, si esta se hace demasiado pequeña los pesos prácticamente

no cambiarán haciendo que el aprendizaje de la red se estanque. Hasta antes de que apareciera ResNet se habían propuesto varias soluciones a este problema, pero con resultados poco concluyentes. ResNet justamente se presenta como una de las formas más efectivas de resolver el problema del desvanecimiento del gradiente.

La novedosa idea de ResNet es simplemente, en lugar de aprender el mapeo $x \rightarrow y$ con una función $H(x)$ (formada por una concatenación de capas con funciones de activación no lineales), se intenta aprender $F(x) = H(x) - x$, de modo que para recuperar el mapeo original simplemente se suma el *input* a la salida de la red: $F(x) + x = H(x)$. La hipótesis es que, tomando como ejemplo la función identidad $H(x) = x$, resulta más fácil aprender $F(x) = 0$ mediante apilamiento de capas convolucionales no lineales, que el mapeo original. La función $F(x)$ se denomina función residual.

Luego, la propuesta de Resnet es utilizar conexiones *shortcuts* las cuales saltan una o más capas. Estas conexiones se agrupan en bloques residuales como se muestra en la figura 1.6.

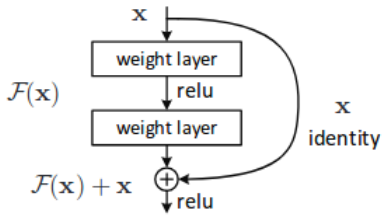


Figura 1.5: Conexión residual.

Hay 2 tipos de conexiones residuales:

- Conexiones identidad las cuales se utilizan cuando el *input* y el *output* tienen la misma dimensionalidad (número de canales).

$$y = F(x, \{W_\text{i}\}) + x$$

- Por el contrario, cuando las dimensiones son distintas, se aumenta o reduce la dimensionalidad de la entrada de modo de igualar a la de la salida, lo cual se logra aplicando una convolución de 1x1. A este tipo de conexiones residuales se le llama proyectadas.

$$y = F(x, \{W_\text{i}\}) + W_s x$$

La figura 1.6 muestra dos tipos de bloques residuales con 2 y 3 convoluciones, respectivamente. Estos bloques residuales se suelen agrupar formando super bloques o grupos los cuales se caracterizan porque el primer bloque residual es de tipo proyectado, a fin de aumentar el número de canales y a la vez que aplica una convolución con *stride* 2 lo cual reduce la dimesionalidad espacial a la mitad. Estos grupos se apilan creando una red convolucional profunda. La tabla 1.7 muestra ejemplos de redes ResNet.

La ventaja de las redes residuales es que alcanzan un menor error de entrenamiento y test que su contraparte plana (sin conexiones residuales) como se puede observar en la figura 1.8. Esto implica que se pueden hacer redes más profundas y por lo tanto obtener una mejor representación de características sin la preocupación de que el gradiente se degrade.

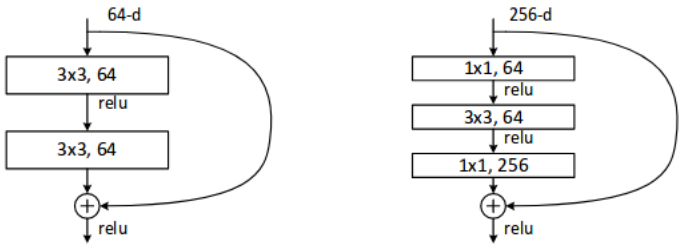


Figura 1.6: Bloques residuales.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 1.7: Arquitecturas ResNet típicas.

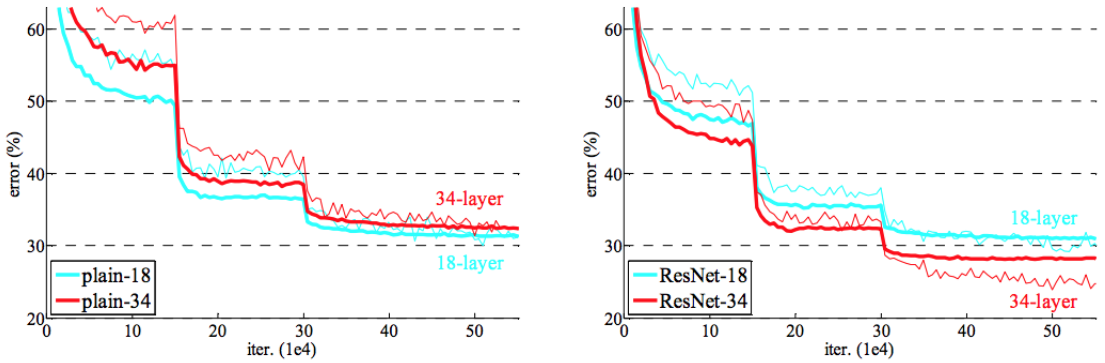


Figura 1.8: Comparación entre las curvas de error durante el entrenamiento (linea delgada) y en la validación (linea gruesa).

Nótese que las redes residuales son en realidad ensamblajes de redes relativamente pequeñas (si un bloque no es efectivo en extraer características de la entrada retorna la misma entrada y así sucesivamente con las capas siguientes). Luego, ResNet no resuelve el problema del desvanecimiento del gradiente porque lo preserva a lo largo de toda la red, sino que simplemente evita el problema construyendo ensambles de muchas redes pequeñas.

En este trabajo, se usará una red residual como red *backbone* y cuya función será la de

extraer características de las imágenes.

1.6. Modelos de detección de objetos

Una parte del área de la visión por computadora es la detección de objetos. La diferencia entre un problema de detección de objetos y uno de clasificación es que en el de detección lo que se busca es encontrar *bounding boxes* que encierren a los objetos de interés además de clasificarlos. En los últimos años han habido significativos avances en la detección de objetos. La figura 1.9 muestra como ha ido mejorando el desempeño de los modelos a lo largo de los últimos años, especialmente gracias al avance del *deep learning*.

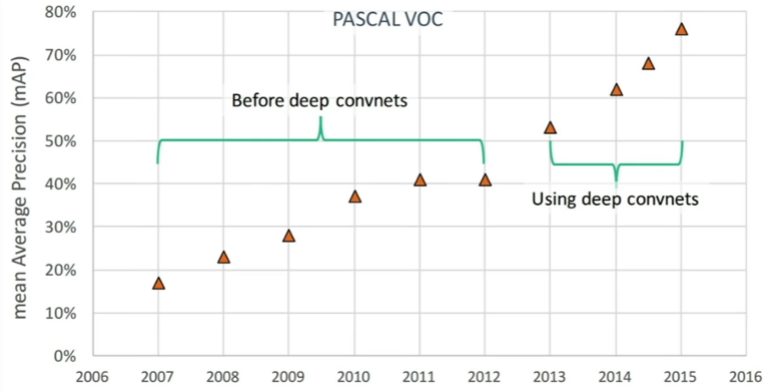


Figura 1.9: Evolución del desempeño de los modelos a lo largo de los años.

En lo que sigue se describirán brevemente los tipos de detectores de objetos.

1.7. Tipos de detectores de objetos

Los detectores de objetos se pueden clasificar en tres categorías: detectores de objetos clásicos, detectores de 2 etapas y detectores de 1 etapa.

Detectores de objetos clásicos: El enfoque de la ventana deslizante o *slinding window* en que un clasificador va recorriendo toda la imagen densamente, ha sido por mucho tiempo el método más estudiado. Sin embargo, dado que los objetos pueden estar en cualquier parte de la imagen y tener diferentes tamaños y aspectos, se necesitaría seleccionar una gran cantidad de estas ventanas lo cual es extremadamente costoso computacionalmente y además muchas de estas ventanas serían redundantes. Es por este motivo que se han desarrollado distintos algoritmos para encontrar estas regiones de interés de forma más eficiente. Uno de los primeros casos de éxito es el trabajo de LeCun et al. que en 1998 que aplicó una CNN para el reconocimiento de dígitos manuscritos. Luego, el 2001 Viola y Jones [25] proponen un detector de objetos considerado el primero en alcanzar tasas de detección competitivas en tiempo real, el cual fue aplicado principalmente para la detección de caras. Luego, la

aparición de HOG (*Histogram of Oriented Gradients*) dio lugar a métodos efectivos para la detección de personas. El paradigma de la ventana deslizante fue por mucho tiempo el estándar en detección de objetos, hasta el resurgimiento del *deep learning* con la AlexNet [8]. Es a partir de este punto que empiezan a surgir los primeros detectores de objetos basados en *deep learning*, y rápidamente los detectores de 2 etapas pasaron a ser el estado del arte en la detección de objetos.

Detectores de 2 etapas: Hasta antes de la aparición de RetinaNet, el paradigma dominante en la detección de objetos con *deep learning* estuvo basada en el enfoque de dos etapas, siendo estos los que alcanzaban mejor desempeño en comparación a su par de una etapa. Como uno de los primeros trabajos en usar este enfoque se encuentra *Selective Search* [24]. Los detectores de 2 etapas, como su nombre indica, constan de dos etapas. La primera genera un conjunto disperso de regiones propuestas las cuales debiesen cubrir todos los objetos y al mismo tiempo descartar la mayoría de regiones negativas (el fondo). La segunda etapa clasifica estas propuestas entre las distintas clases y el fondo. R-CNN [5] reemplazó el clasificador de la segunda etapa por una CNN lo cual mejoró considerablemente la precisión y marcó el comienzo de la detección de objetos moderna. Luego aparecieron las *Region Proposal Networks* (RPN) que integraron la generación de propuestas con el clasificador de la segunda etapa en una única red convolucional, creando así Faster R-CNN [17]. Faster R-CNN marcó un precedente en el área principalmente gracias a sus 2 importantes contribuciones: i) el uso de una red de propuesta de regiones o RPN (*regions proposal network*), y ii) el uso de *anchors* para lidiar con el tamaño variable de los objetos.

Detectores de 1 etapa: Estos métodos se caracterizan por ser mucho más rápidos que su contraparte de 2 etapas, pero a la vez alcanzan un menor desempeño que estos últimos. Sin embargo, con la aparición de RetinaNet [11], y en especial de su *focal loss*, esta diferencia se ha reducido significativamente, y los detectores de 1 etapa compiten a la par con los de 2 etapas. Si bien en el último año se han propuesto nuevos métodos de 2 etapas tipo cascada que han vuelto a dejar atrás en términos de desempeño a los de 1 etapa, estos últimos se han popularizado enormemente ya que compensan una pequeña baja del desempeño con un significativo aumento en velocidad de procesamiento. Los métodos de una etapa por excelencia son SSD [13], YOLO [14] y el más reciente RetinaNet [11].

1.8. Principales algoritmos de detección de objetos

1.8.1. Faster R-CNN

Faster R-CNN [17] es un detector de objetos de 2 etapas, está compuesto de 2 bloques que comparten una red convolucional (*backbone*) como se puede observar en la figura 1.10.

Esta red *backbone* tiene como propósito producir características discriminativas las cuales son usadas para estimar candidatos a objetos así como también para predecir su clase respectiva. El primer bloque se denomina *Region Proposal Network* (RPN) el cual se encarga de encontrar regiones con una alta probabilidad de contener objetos. Mientras que el segundo

corresponde a Fast R-CNN [4] (el antecesor de Faster R-CNN) el cual se encarga de predecir la clase de los objetos propuestos.

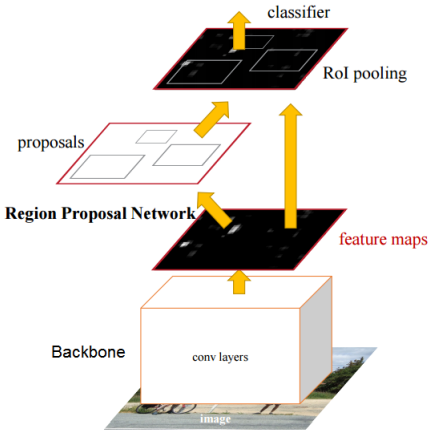


Figura 1.10: Arquitectura de Faster R-CNN.

La *Region Proposal Network* o RPN es una CNN cuya función es la de detectar regiones en la imagen de entrada donde puedan haber objetos independientemente de la clase. La RPN aplica una capa convolucional inmediatamente después del mapa de características producido por la red *backbones*. Su propósito es generar vectores de características (256 características por nodo) los cuales predicen la presencia de un objeto en cierta región de la imagen.

El último componente de la RPN es el predictor. En realidad se tienen 2 predictores, uno encargado de predecir si una región contiene un objeto o no, y el otro a cargo de predecir el *bounding box*, si hay un objeto. Mientras el primero corresponde a un clasificador, el segundo corresponde a un regresor.

Para acelerar el entrenamiento, la RPN usa un conjunto de cajas de referencia por nodo llamadas *anchors*. Estos *anchors* corresponden a regiones rectangulares centradas aproximadamente en los centros de los campos receptivos de cada nodo en el mapa de características. Un *anchor* queda definido por una escala (por ejemplo 32x32) y su razón de aspecto (por ejemplo, 1:1, 1:2, 2:1). La elección de los *anchors* a usar depende del problema a resolver, aunque existen técnicas que permiten automatizar el proceso de encontrar los *anchors* adecuados. La figura 1.11 muestra un conjunto de *anchors* para un mismo nodo.

Por otra parte, la figura 1.12 muestra un ejemplo de como se verían estos *anchors* distribuidos sobre la imagen de entrada. En la práctica, los *anchors* cubren densamente la imagen, especialmente en los métodos de 1 etapa los cuales son descritos más adelante.

1.8.2. YOLO

YOLO [14] (*You Only Look Once*) es un detector de objetos de 1 etapa el cual es considerado uno de los más rápidos en la actualidad. La figura 1.13 muestra una gráfica comparativa entre distintos algoritmos de detección de objetos y como se puede observar, YOLO presenta

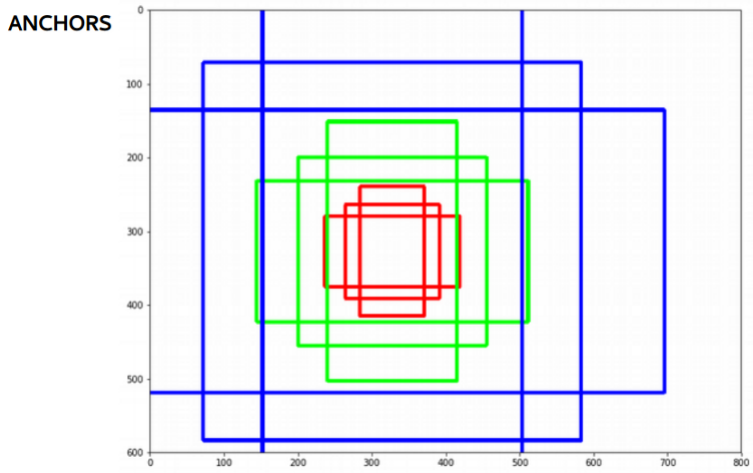


Figura 1.11: Ejemplo de *anchors* para un nodo.

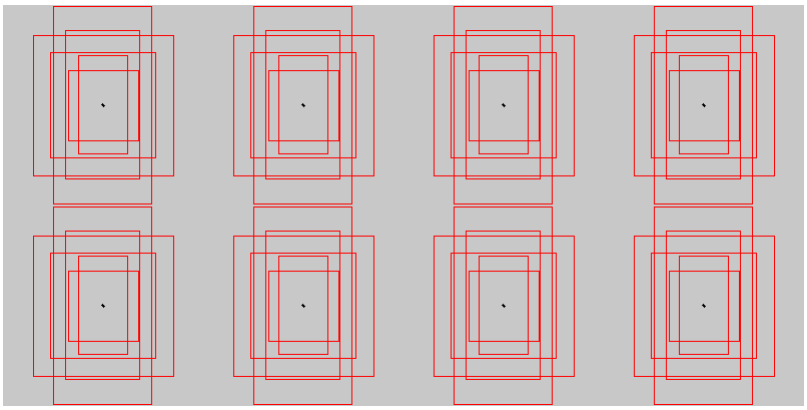


Figura 1.12: Ejemplificación de como se distribuyen los *anchors* en la imagen. En la práctica estos *anchors* cubren densamente la imagen.

el mejor *trade-off* entre precisión y rapidez.

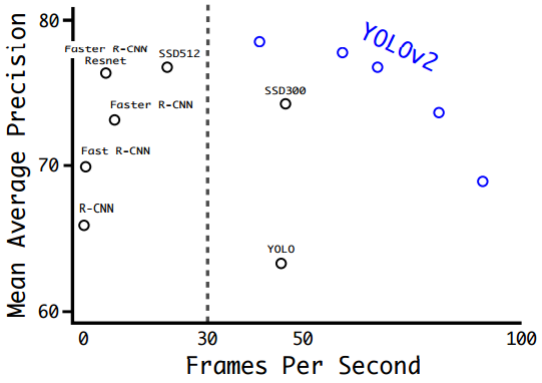


Figura 1.13: Desempeño y velocidad de distintos algoritmos de detección de objetos sobre el dataset VOC 2007.

Desde su primera aparición en 2016 YOLO ha pasado por varias iteraciones. En este trabajo se utilizará la tercera versión llamada YOLOv3 [16] publicada el 2018 la cual es considerada la mejor. En lo que sigue se describe brevemente como funciona YOLOv3. Por simplicidad se omitirá el v3, de modo que YOLO hará referencia a la versión 3, a menos que se mencione lo contrario.

YOLO, al igual que la mayoría de los modelos de detección, involucra capas de *downsampling* para colectar información global y reducir el uso de memoria. Luego, el tamaño de la salida de la red será menor al de la imagen de entrada. Por lo tanto, un punto o píxel (x, y) en la imagen original será mapeado a la posición $(\lfloor \frac{x}{R} \rfloor, \lfloor \frac{y}{R} \rfloor)$, donde R es el factor de *downsampling*. A cada punto en el tensor de salida le corresponderá una región de tamaño $R \times R$ de la imagen de entrada. Estas regiones formarán una grilla del mismo tamaño del tensor de salida $S_H \times S_W$, con $S_H = \lfloor \frac{H}{R} \rfloor$ y $S_W = \lfloor \frac{W}{R} \rfloor$, donde H y W corresponden al alto y ancho de la imagen de entrada. Luego, si el centro de un objeto cae dentro de una celda de la grilla, esta será la responsable de detectarlo, lo cual es equivalente a activar el nodo del tensor de salida asociado a dicha región. En la primera versión de YOLO era la celda la que se activaba, sin embargo, esto cambió en la siguiente versión cuando se introdujeron los *anchors*, los cuales permitieron aumentar considerablemente el *recall* del modelo. Luego, en la versión actual de YOLO cada celda tiene A *anchors* o cajas preestablecidas con diferentes tamaños y razones de aspecto. Son estos *anchors* los que se activan al detectar un objeto, los cuales tendrán asociado un valor llamado *objectness*. El valor predicho de *objectness* debe ser 1 si el *anchor* tiene el mayor traslape con un objeto dado. Dicho traslape es medido por el *Intersection over Union* o IoU el cual es explicado más adelante (ver imágen 1.24). Si el *anchor* no es el mejor, pero sobrelapa a un objeto por más de un cierto umbral (por ejemplo, $IoU \geq 0{,}5$), se ignora la predicción. En cualquier otro caso, debe ser 0. Para que la red aprenda a predecir dicho valor, durante el entrenamiento se utiliza una función de pérdida dada por la entropía cruzada binaria (BCE) también descrita más adelante (ver ecuación 1.12).

Si bien, con este valor de *objectness* es posible predecir en qué celda cae el centro del objeto, aún se requiere localizar precisamente el *bounding box*. Para ello YOLO predice 4 valores t_x, t_y, t_w, t_h de tal forma que la predicción del *bounding box* del objeto detectado está dado por:

$$b_x = \sigma(t_x) + c_x \tag{1.1}$$
$$b_y = \sigma(t_y) + c_y \tag{1.2}$$
$$b_w = p_w \mathrm{e}^{t_w} \tag{1.3}$$
$$b_h = p_h \mathrm{e}^{t_h} \tag{1.4}$$

donde (b_x, b_y) corresponde a la posición del centro del objeto detectado, el par (b_w, b_h) representa el ancho y alto de dicho objeto, (c_x, c_y) corresponde a la posición del borde superior izquierdo de la celda respectiva (cuyo *anchor* detectó el objeto). Por otra parte, el par (p_w, p_h) corresponde al ancho y alto del *anchor* de referencia. Por último, σ representa la función sigmoide definida como $\sigma(x) = \frac{1}{1+\mathrm{e}^{-x}}$. La figura 1.14 muestra como se relacionan las magnitudes anteriores.

Durante el entrenamiento se invierten las expresiones en 1.1 para obtener los valores t_x, t_y, t_w, t_h de *ground thruth* y se comparan con los predichos por la red $\hat{t}_x, \hat{t}_y, \hat{t}_w, \hat{t}_h$ por medio

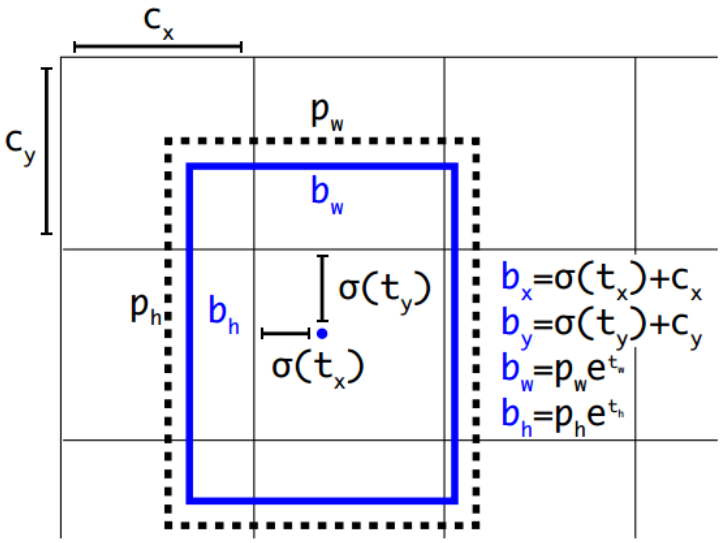


Figura 1.14: Relación entre cajas reales y predichas.

de una función de pérdida cuadrática sobre el error (ℓ_2 *loss*) dada por $\mathcal{L}(t, \hat{t}) = \sum_*(t_* - \hat{t}_*)^2$.

Cada *anchor* tendrá también asociado la clase del objeto respectivo con mayor traslape. Para la predicción de la clase se usan clasificadores logísticos independientes. Así, la función de pérdida en el entrenamiento está dada por la entropía cruzada binaria (BCE). Se opta por no usar *softmax* ya que no es necesaria para obtener un buen desempeño, además que esta formulación resulta ser más generalizable para otros problemas donde hay clases que se traslapan (por ejemplo, animal y perro). De modo que un enfoque multi-clase modelará mejor la data.

Por último, durante el entrenamiento si un *anchor* no tiene asignado un objeto entonces no contribuye ni a la pérdida de clasificación ni localización, solo de *objectness*.

1.8.3. RetinaNet

RetinaNet [11] es un detector de objetos de 1 etapa el cual, junto a su innovadora función de pérdida focal (*focal loss*), se presenta como uno de los mejores hasta la fecha. En la figura 1.15 se muestra una comparación entre distintos algoritmos de detección de objetos, donde la métrica corresponde al mAP y el dataset es COCO [12].

Como se puede notar de la figura 1.15, RetinaNet resulta ser el mejor método de los expuestos. Otra de las ventajas de RetinaNet es que, al ser de 1 etapa (sin *region proposals*), es lo suficientemente rápido para aplicaciones en tiempo real (a diferencia de los métodos de 2 estapas como Faster R-CNN).

La arquitectura de RetinaNet consta de una red *backbone* y dos subredes para tareas específicas. El *backbone* es responsable de calcular los mapas de características sobre toda la imagen de entrada. La primera subred consiste en un clasificador que recibe la salida del

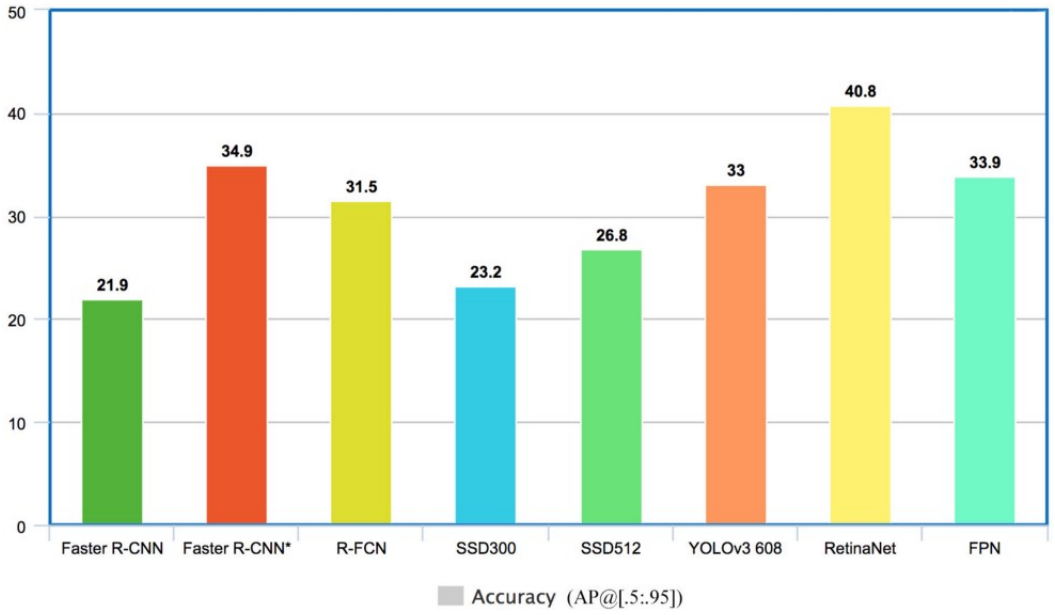


Figura 1.15: Comparación de modelos de detección de objetos.

backbone. Mientras que la segunda subred corresponde al regresor de los *bounding boxes*. La figura 1.16 muestra la arquitectura general de RetinaNet.

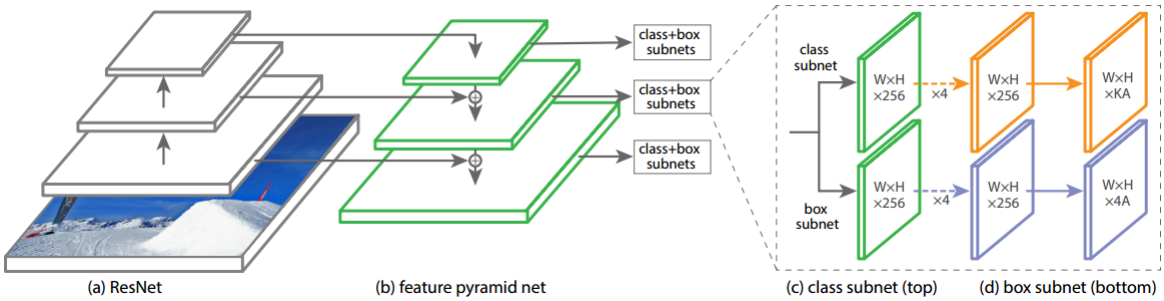


Figura 1.16: Arquitectura general de RetinaNet.

El *backbone* en RetinaNet consiste en una *Feature Pyramid Network* (FPN) [10]. La FPN extiende la red convolucional estándar con un camino de arriba a abajo y conexiones laterales de modo que la red construye eficientemente una pirámide de características multiescala, donde cada nivel puede ser usado para detectar objetos a diferentes escalas. La ventaja de este diseño es que los mapas de características de alta resolución de las primeras capas se benefician de la alta información semántica de los mapas en las últimas capas.

RetinaNet se basa en *anchors* para hacer sus predicciones. Para ello, a cada nivel se le asocian *anchors* de diferentes escalas. Cada *anchor* tiene asignado un vector *one-hot* de largo K para clasificación, donde K es el número de clases. Además, cada *anchor* tiene asociado un vector de largo 4 correspondiente a los parámetros de la caja a predecir.

Durante el entrenamiento, se clasifican los *anchors* como positivos si sobrelapan a alguna caja de objeto en más un cierto umbral, en cuyo caso se le asigna la clase del objeto respectivo.

Por lo general, la regla de asignación es si el IoU del *anchor* y el objeto es mayor a 0.5 se asigna la clase del objeto al *anchor*, en cambio, si es menor a 0.4 corresponde a fondo (*background*). A los *anchor* positivos se les configura el vector de clasificación de largo K de modo que la entrada correspondiente a su clase sea 1, y 0 para el resto. Los *anchors* no asignados, o sea, aquellos con IOU entre [0.4, 0.5), son ignorados durante el entrenamiento.

Para predecir los *bounding boxes* se usa la parametrización de [17] dada por las siguientes expresiones:

$$t_x = (x - x_a)/w_a \qquad t_x^* = (x^* - x_a)/w_a \tag{1.5}$$
$$t_y = (y - y_a)/h_a \qquad t_y^* = (y^* - y_a)/h_a \tag{1.6}$$
$$t_w = \log(w/w_a) \qquad t_w^* = \log(w^*/w_a) \tag{1.7}$$
$$t_h = \log(h/h_a) \qquad t_h^* = \log(h^*/h_a) \tag{1.8}$$

donde x, y, w, h denotan las coordenadas del centro de la caja y su respectivo ancho y alto. Las variables x, x_a y x^* representan a la caja predicha, el *anchor* y el *groundtruth*, respectivamente. Análogamente para y, w, h.

A cada nivel del *backbone* definido por la FPN se acoplan 2 subredes: el clasificador y el regresor. La subred de clasificación predice la probabilidad de la presencia de un objeto en cada posición espacial para cada uno de los A *anchors* y K clases. Para ello se toma un mapa de características de C canales (típicamente $C = 256$) de un nivel de la pirámide y se aplican 4 convoluciones de 3x3 con C filtros, para luego aplicar una convolución final de 3x3 con KA filtros con función de activación sigmoide. Cada una de las KA entradas representa un clasificador binario entre la clase respectiva y el fondo.

Por otra parte, la subred de regresión, al igual que la de clasificación, se acopla en paralelo a cada nivel de la pirámide de características, y al igual que la subred de clasificación consiste en una pequeña red FCN (*fully convolutional network*) de 4 capas, seguida de una convolución con $4A$ filtros, correspondiente a los 4 parámetros t_x, t_y, t_w, t_h que modelan el *offset* entre el *anchor* y el *ground truth*.

Los parámetros de estas subredes son compartidos entre los niveles, pero distintos entre clasificadores y regresores, o sea, los clasificadores tienen pesos compartidos comunes, pero distintos a los de los regresores los cuales también son compartidos entre niveles.

1.8.4. FCOS

En la actualidad, la mayoría de los métodos de detección de objetos del estado del arte tales como Faster R-CNN, YOLOv3 y RetinaNet se basan en el uso de cajas predefinidas llamadas *anchors*. Por mucho tiempo se ha creído que el uso de estos *anchors* ha sido la clave del éxito de tales detectores. Sin embargo, en el último tiempo han surgido novedosas propuestas que buscan resolver el problema de detección de objetos de un modo completamente *fully convolutional* en la que no hayan *anchors* responsables de detectar los objetos, sino que la detección sea realizada por los mismos píxeles de la imagen de forma análoga

al problema de segmentación semántica. Una de estas propuestas del presente año 2019 se llama FCOS (*fully convolutional one-stage object detector*) [23]. Los autores de FCOS afirman tener un desempeño mejor que RetinaNet, pero con la salvedad que su método resulta mucho más simple. Esto pues FCOS es libre de *anchors* y *region proposals* por lo que evita los costosos cómputos asociados a estos como el cálculo de IoU durante el entrenamiento reduciendo significativamente el uso de memoria. Incluso más importante aún es el hecho que reduce significativamente el número de parámetros de diseño a ajustar (la mayoría de forma heurística) pues no requiere de los típicos hiper-parámetros asociados a los *anchors* como, por ejemplo, sus tamaños y razones de aspecto, o los umbrales que deciden si un *anchor* es positivo o negativo. El problema con estos hiper-parámetros es que el desempeño del modelo resulta ser muy sensible a estos. Por ejemplo, en RetinaNet, variar levemente estos hiper-parámetros afecta el desempeño hasta en un 4 %. Como resultado, además de añadirle complejidad al algoritmo (la cual, como pretenden demostrar los autores, resulta innecesaria) estos hiper-parámetros requieren de un cuidadoso proceso de ajuste. Aún con un cuidadoso diseño de estos, debido a que las escalas y razones de aspecto de los *anchors* son fijas, los detectores tienen dificultades con los objetos con variaciones extremas de forma, en especial los pequeños. Por lo tanto, los anchors limitan de cierto modo la habilidad de generalización de los detectores, ya que por cada problema diferente se deben diseñar nuevos *anchors* con escalas y razones de aspecto distintas.

A fin de tener un *recall* alto, un detector basado en *anchors* requiere localizarlos densamente sobre la imagen. Así, por ejemplo, para una imagen de 800x800 se requiere del orden de 180K *anchors* para cubrirla por completo. Luego, al momento de etiquetar dichos *anchors* resulta que la mayoría son negativos (pudiendo llegar a la proporción de 1:1000). Esto agrava el desbalance entre muestras positivas y negativas en el entrenamiento, además que incrementa considerablemente el costo computacional del algoritmo al calcular el IoU entre todos los *anchors* y el *ground truth*.

Recientemente, las redes *fully convolutional* (FCNs) han tenido un gran éxito en problemas como segmentación semántica, estimación de profundidad, detección de puntos clave (*keypoints*), entre otros. Sin embargo, a pesar del éxito mencionado, el estado del arte de los detectores de objetos sigue basándose en el uso de *anchors*. Luego, la propuesta de FCOS consiste en diseñar un *framework fully convolutional* basado en píxeles para el problema de detección de objetos.

La idea de FCOS es considerar los mismos puntos sobre la imagen de entrada que en el caso de RetinaNet eran los centros de los *anchors* y en el de YOLO los centros de las celdas de la grilla. Estos puntos tiene la propiedad de estar cercanos a los centros de los campos receptivos de cada neurona del mapa de características de salida de la red. Si s es el *stride* total o razón entre el tamaño de la imagen de entrada y el mapa de características, entonces, dichos puntos se pueden expresar como el siguiente arreglo: $P = (xs + \lfloor \frac{s}{2} \rfloor, ys + \lfloor \frac{s}{2} \rfloor)$ para $x \in \{0, ..., W-1\}, y \in \{0, ..., H-1\}$, donde W y H son el ancho y alto del mapa de características, respectivamente. A diferencia de un detector basado en *anchors* en que las muestras a clasificar y *regresionar* son los *anchors*, para FCOS las muestras son los nodos. Luego, un punto $(x, y) \in P$ es considerado positivo si cae dentro de cualquier objeto, y se le asocia la clase del mismo. En otro caso, el punto es negativo (clase *background*). Para predecir el *bounding box* se recurre al vector $\mathbf{t} = (l, t, r, b)$, donde l, t, r y b (*left, top, right, bottom*) son

las distancias desde el punto (x, y) a los cuatro lados de la caja de *ground truth*. Si un punto cae

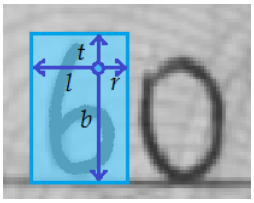


Figura 1.17: Para cada centro de celda (las mismas de YOLO) FCOS predice las distancias l, t, r, b.

dentro de múltiples cajas de *ground truth* se considera la caja de área mínima. Formalmente, si un punto (x, y) está asociado una caja de *ground truth* $B = (x_1, y_1, x_2, y_2)$, donde (x_1, y_1) y (x_2, y_2) representan la esquina superior izquierda y la inferior derecha respectivamente. Entonces, el vector objetivo a predecir está dado por $\mathbf{t} = (l, t, r, b) = (x - x_1, y - y_1, x_2 - x, y_2 - y)$

En cuanto a la arquitectura de la red, esta es similar a RetinaNet con red *backbone* una ResNet y FPN en caso de entrenamiento multiescala. La diferencia reside en las capas de salida, que en el caso del clasificador es de N, donde N es el número de clases. Mientras que para el regresor es de solo 4 para el vector $\mathbf{t} = (l, t, r, b)$. Nótese como se reduce considerablemente la cantidad de variables en la salida en comparación a los otros modelos (RetinaNet y Yolo)

Durante el entrenamiento, la función de pérdida del clasificador, al igual que en los otros modelos, está dada por la entropía cruzada binaria (BCE). En cuanto al regresor, dado que FCOS predice directamente las dimensiones del objeto no recurriendo a transformaciones de ningún tipo, puede aprovechar las propiedades geométricas del *bounding box* predicho. Así, en lugar de considerar las 4 variables independientemente como lo hacen los modelos anteriores (YOLO con una ℓ_2 *loss* y RetinaNet con la *Huber loss*) en que la pérdida total está dada por la suma de las pérdidas por componente, FCOS las considera correlacionadas, de modo que su función de pérdida de localización está dada por la llamada *IoU loss* [26]. La *IoU loss* considera estas 4 componentes como un todo no separable en una suma de sub-pérdidas, lo cual conduce a una convergencia más rápida y una mayor precisión en la localización del objeto. La *IoU loss* está dada por $IoU\ loss = -\ln(IoU(box_{gt}, box_{\text{pred}}))$ como se ilustra en la figura 1.18.

1.9. Redes recurrentes

Una red neuronal recurrente (RNN) es un tipo de red neuronal especialmente diseñada para el procesamiento de data secuencial como, por ejemplo, en el reconocimiento de voz, texto, etc. Mientras que en una red neuronal tradicional (incluyendo las CNNs) la salida depende exclusivamente de la entrada, las RNNs mantienen un estado interno de modo que la salida dependerá tanto de la entrada como de dicho estado. La gracia de guardar un estado es que, al ser variable, permite a la red ser persistente, o sea, tener cierta memoria de las

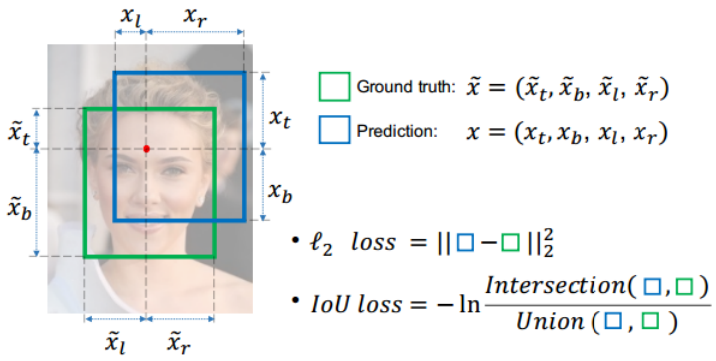


Figura 1.18: Comparación entre la *IoU loss* y la ℓ_2 *loss*.

entradas pasadas, de modo que la salida de la red no solo es dependiente directamente de la entrada actual, sino que indirectamente también de las pasadas. Para lograr esto, estas redes poseen retroalimentaciones o auto-conexiones (ciclos dirigidos) de modo que la información de una entrada dada persista en la siguiente ejecución. La figura 1.19 muestra la estructura de una red recurrente. A la izquierda se observa el *loop* que es el que permite guardar el contexto de las entradas previas durante el entrenamiento. A la derecha se observa la misma red desenrollada.

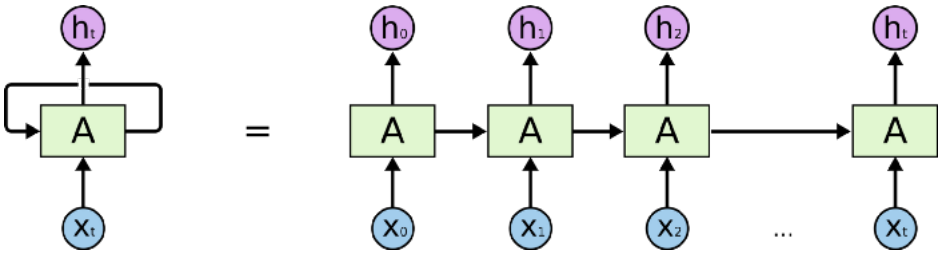


Figura 1.19: Diagrama de una RNN.

En términos generales, una RNN se puede expresar por la ecuación 1.9

$$\mathbf{h}^{(t)} = g^{(t)}(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, ..., \mathbf{x}^{(2)}, \mathbf{x}^{(1)}) \tag{1.9}$$

$$= f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta) \tag{1.10}$$

donde $\mathbf{x} \in \mathbb{R}^N, \mathbf{h} \in \mathbb{R}^M$. La función g corresponde a la versión enrollada y toma como argumento todas las entradas pasadas produciendo un nuevo estado que a la vez actúa como salida de la red. En cambio, la función f representa la versión desenrollada, y solo necesita de la entrada actual y el estado previo para producir el estado actual. La ventaja de la versión desenrollada es que permite factorizar $g^{(t)}$ en repetidas aplicaciones de la función f, haciendo el modelo independiente del largo de la secuencia.

Una RNN tradicional tiene una capa oculta auto-conectada entre su capa de entrada y salida. Cada vez que recibe una nueva entrada x_t de la secuencia de entrada, la RNN actualiza su estado interno h_t con una función no lineal g la cual recibe como argumento

tanto la entrada x_t como el estado anterior h_{t-1}. Formalmente se tiene $h_t = g(x_t, h_{t-1})$. De este modo, las entradas pasadas son capturadas y usadas como contexto para la predicción.

En la práctica las RNNs tradicionales sufren de memoria a corto plazo la cual limita considerablemente el tamaño del contexto que pueden almacenar. Este problema se relaciona directamente con el del desvanecimiento del gradiente visto anteriormente. Un tipo especial de red recurrente llamada LSTM (*Long-Short Term Memory*) está especialmente diseñada para resolver dicho problema.

Como se mencionó anteriormente, la salida de una RNN se puede expresar como la aplicación sucesiva de la función f, la cual se representa como un módulo o celda base y que, al concatenarlas, formarán la red completa. La figura 1.20 muestra la arquitectura más básica de una celda recurrente, la cual consta simplemente de una capa *tanh*.

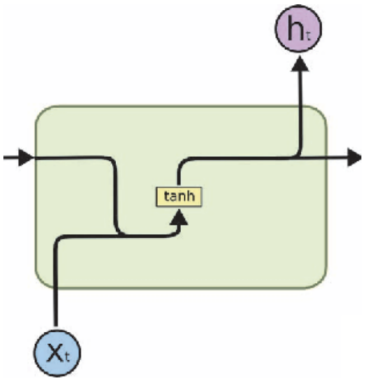


Figura 1.20: Arquitectura básica de una celda recurrente.

Lo que hace entonces la LSTM es modificar la estructura de este módulo. La idea clave es hacer de este módulo una celda de memoria. Para ello se introducen las llamadas compuertas las cuales ayudan a regular el flujo de información que entra a la celda de estado. Estas compuertas pueden aprender cual información en una secuencia es relevante y cual no. Con este mecanismo se logra traspasar la información en una secuencia larga. La figura 1.21 muestra la arquitectura de una celda LSTM.

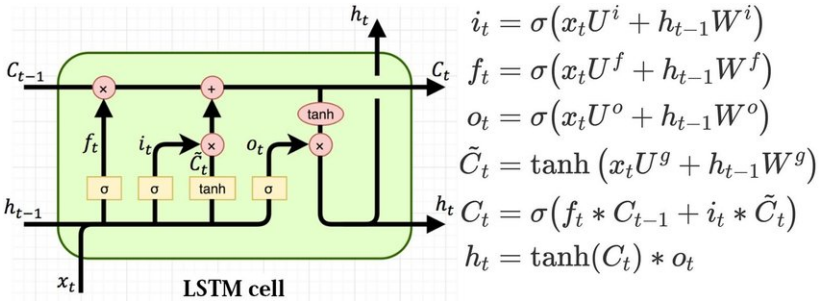


Figura 1.21: Arquitectura básica de una celda LSTM.

Como se puede observar en la figura 1.21, una LSTM consiste en una celda de memoria y 3 compuertas llamadas de entrada i_t, salida o_t y de olvido f_t. La salida en el tiempo t

esta dada por h_t y el estado por C_t. Conceptualmente, la celda de memoria almacena los contextos pasados, y las compuertas de entrada y salida permiten almacenar contextos por un periodo largo de tiempo. Por otra parte, la compuerta de olvido permite a la celda limpiar su memoria. Actualmente, la LSTM se ha convertido en una de las RNN más usadas dado su diseño especial el cual permite capturar dependencias de alto rango las cuales ocurren frecuentemente en secuencias basadas en imágenes.

La LSTM es unidireccional, es decir, solo usa contextos pasados. Sin embargo, en secuencias provenientes de imágenes, contextos de ambas direcciones (de izquierda a derecha y viceversa) son útiles y complementarios. Por lo tanto, es común combinar 2 LSTMs en direcciones opuestas para crear una LSTM bidireccional (BiLSTM) la cual permite a la red tener acceso tanto a los contextos pasados como a los futuros. Además, se pueden apilar múltiples LSTM bidireccionales, dando como resultado una BiLSTM profunda como muestra la figura 1.22. Esto permite mayores niveles de abstracción, lo cual puede mejorar el desempeño de la red.

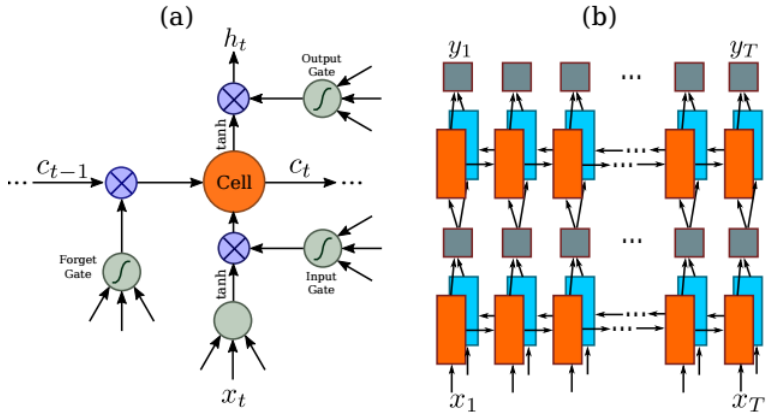


Figura 1.22: (a) Estructura de una celda LSTM. (b) Estructura de una red BiLSTM profunda.

1.10. *Connectionist Temporal Classification*

Cuando una red recurrente se combina con una capa de salida especial llamada CTC [6] (*connectionist temporal classification*), se obtiene un excelente *framework* para tareas de etiquetado secuencial. Esto gracias a la habilidad de la CTC de modelar el alineamiento entre entradas y etiquetas directamente, lo cual la hace especialmente adecuada para tareas de reconocimiento de secuencias, tales como reconocimiento de texto en imágenes o voz en audio.

Si la data de entrada a la RNN proviene de una imagen, lo que se suele hacer es primero procesarla con una CNN para extraer sus características, las cuales posteriormente son ordenadas de forma secuencial (transposición de dimensiones) y se hacen pasar por una RNN, y cuya salida es procesada por la CTC para realizar la predicción final. De este modo, la combinación CNN-RNN-CTC se convierte en un novedoso *framework* que integra la extracción automática de características (capas convolucionales), el modelamiento secuencial (capas

recurrentes) y la transcripción (traducción de la salida de la RNN a la secuencia de *labels* final), y el cual resulta especialmente útil para las tareas de reconocimiento de secuencias basadas en imágenes, como por ejemplo, el reconocimiento de texto manuscrito. Las ventajas de este tipo de arquitecturas unificadas es que son entrenables de extremo a extremo y no requieren de ningún tipo de pre o postprocesamiento costoso como segmentación, localización de componentes, etc.

A este tipo de arquitecturas que combinan CNN y RNN se les suele denominar arquitecturas CRNN donde, como se mencionó anteriormente, las capas convolucionales se encargan de extraer una representación de características secuenciales de la imagen de entrada. Dicha representación se obtiene al dividir secuencialmente el último mapa de características producido por la CNN en vectores de características formados por las columnas de este, a este proceso se le denomina transposición dimensional de las características. Así, el i-ésimo vector está dado por la concatenación de las i-ésimas columnas de todos los mapas de características, de este modo, si el tensor de salida es de $H \times W \times C$, se tendrán W vectores de tamaño HC. Nótese que, como las capas convolucionales operan sobre regiones locales, cada columna de los *feature maps* está asociada a una región rectangular o *frame* de la imagen de entrada, la cual es denominada *campo receptivo*. Esta secuencia de vectores (ordenados de izquierda a derecha) constituye la entrada a la RNN y cuya salida predecirá una distribución de probabilidad de los *labels* para cada *frame*.

Las capas recurrentes tienen 3 ventajas: (i) las RNNs tienen gran capacidad para capturar información de contexto dentro de una secuencia. Usar dicha información contextual para el reconocimiento de secuencias basadas en imágenes es más estable y útil que tratar cada símbolo independientemente, ya que, por ejemplo, pueden haber símbolos que requieran varios *frames* sucesivos para describirlos. (ii) La RNN puede propagar el error durante el *backpropagation* hacia las capas convolucionales, permitiendo entrenar de forma conjunta tanto capas convolucionales como recurrentes en una misma red. (iii) Las RNNs son capaces de operar con secuencias de largo variable.

La última capa del modelo RCNN consiste en una capa de transcripción. La transcripción es el proceso de interpretar y convertir las predicciones para cada *frame* (campos receptivos de la imagen de entrada) realizadas por la RNN en la secuencia de *labels* final. Más formalmente, la transcripción consiste en encontrar la secuencia de *labels* con mayor probabilidad condicionada sobre las predicciones por *frame*.

En la práctica es común adoptar la probabilidad condicional definida por la CTC. La CTC es un tipo de capa de salida común en redes recurrentes la cual cumple dos funciones. La primera es el cálculo de la *loss* durante el entrenamiento y la segunda es decodificar la salida de la RNN en la inferencia.

Sea A el diccionario del cual se extraen los *labels* (que en el caso de reconocimiento CAR, A son los 10 dígitos). Se define un *label* extra llamado *blank* y denotado como $-$ con el cual se obtiene un nuevo conjunto $A' = A \cup \{-\}$. Sean las predicciones por *frame* $y = y_1, ..., y_T$, donde cada $y_t \in \mathbb{R}^{|A'|}$ representa una distribución de probabilidad sobre A' y T es el largo de la secuencia. Estas predicciones corresponden a la entrada de la CTC y se obtienen al pasar la salida de la RNN por una capa *fully connected* con función de activación *softmax*. Sea $I \in A^*$ la secuencia de *labels* de *ground truth* asociado a una imagen de entrada, donde

$A^* = \bigcup_{\mathrm{i}\geq 0} A^{\mathrm{i}}$. Se define el mapeo $\mathscr{F} : A'^T \mapsto A^{\leq T}$ que elimina el *label* blanco y los *labels* repetidos. Por ejemplo, $\mathscr{F}(-1-4-88000) = 1480$. Luego, la probabilidad de I condicionada a y está dada por la suma de las probabilidades de todos los $\pi \in A'^T$ tales que $\mathscr{F}(\pi) = I$

$$p(I|y) = \sum_{\pi \in \mathscr{F}^{-1}(I)} p(\pi|y) \tag{1.11}$$

donde la probabilidad condicional de π está definida como:

$$p(\pi|y) = \prod_{t=1}^{T} y_{\pi_t}^t$$

donde $y_{\pi_t}^t$ denota la probabilidad de tener el *label* π_t en el tiempo t. Calcular 1.11 directamente es en la práctica infactible debido a la cantidad exponencial de sumas, por lo que dicha probabilidad es calculada por medio de programación dinámica (algoritmo *forward-backward*) [6].

Sea $S = (X, I)$ el conjunto de entrenamiento, donde X es el conjunto de imágenes e I el conjunto de las secuencias de *labels* respectiva. Luego, la función de pérdida CTC se define como:

$$CTC\ loss = -\sum_{(x,\mathrm{i})\in S} \log p(I|y)$$

donde y es la secuencia producida por la RNN.

De este modo, la combinación CNN-RNN-CTC, y en particular CNN-LSTM-CTC, provee un *framework* entrenable de extremo a extremo puesto que solo requiere de imágenes y sus secuencias de *labels* respectivas.

1.11. Supresión no máxima (NMS)

La supresión no máxima o NMS por sus siglas en inglés es un método comúnmente usado por los detectores de objetos en la etapa de postprocesamiento. La NMS consiste en suprimir todas aquellas cajas predichas redundantes, o sea que, corresponden al mismo objeto detectado. El algoritmo de la NMS supone que cada caja tiene un *score* asociado. La métrica usada para comparar *bounding boxes* corresponde al IoU (ver figura 1.24). Luego, el algoritmo es el siguiente:

1. Descartar todas las cajas con *score* bajo ($\leq$ umbral de *score*).
2. Seleccionar la caja con mayor *score* y descartar todas aquellas con IoU $\geq$ un cierto umbral (típicamente 0.5) con la caja seleccionada.
3. Repetir 2 hasta que no queden más cajas.

La figura 1.23 muestra la NMS en acción.

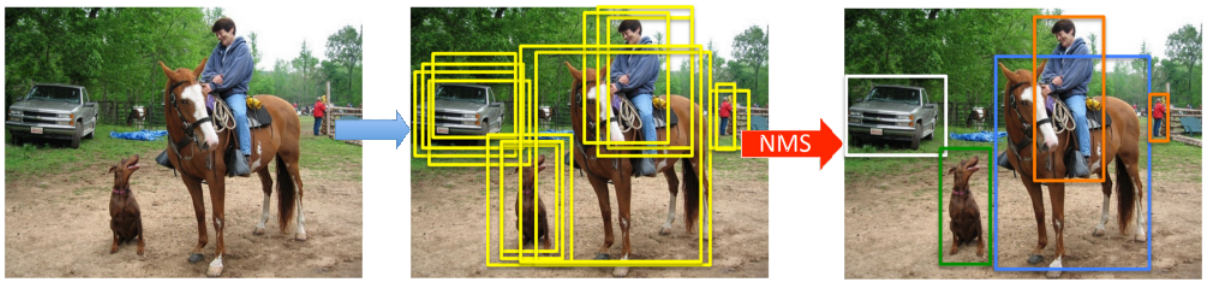


Figura 1.23: NMS en acción.

1.12. Funciones de pérdida comunes en detectores de objetos

1.12.1. Entropía cruzada binaria

La entropía cruzada binaria o BCE (*Binary Cross-Entropy*) está dada por:

$$\mathrm{CE}(y,p) = -y\log(p) - (1-y)\log(1-p) = \begin{cases} -\log(p) & \text{si } y = 1 \\ -\log(1-p) & \sim \end{cases} \tag{1.12}$$

donde $y \in \{0,1\}$ especifica la clase real (*ground truth*) y $p \in [0,1]$ corresponde a la probabilidad estimada por el modelo para la clase $y = 1$. Está función se puede reescribir como $\mathrm{CE}(y,p) = \mathrm{CE}(p_t) = -\log(p_t)$, donde p_t se define como:

$$p_t = \begin{cases} p & \text{si } y = 1 \\ 1-p & \sim \end{cases} \tag{1.13}$$

1.12.2. *Focal loss*

La *focal loss* [11] se propuso como una solución al desbalance extremo presente durante el entrenamiento en los detectores de 1 etapa entre las clases correspondientes a objetos y el fondo (*background*). Para ello introduce a la entropía cruzada estándar un factor modulador $(1-p_t)^\gamma$ el cual le baja el peso a la pérdida de los ejemplos bien clasificados de modo que la red pueda centrarse en aquellos casos más difíciles y previniendo que la vasta mayoría de muestras negativas fáciles acaparen el entrenamiento del detector degenerando el modelo.

La *Focal loss* se define como:

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^\gamma \log(p_t)$$

donde γ es un hiperparámentro, p_t está dado por 1.13 y α_t se define de forma análoga a p_t con $\alpha \in [0,1]$ otro hiperparámetro.

1.12.3. L_2 *loss*

Esta función de pérdida corresponde a la distancia euclidiana entre el valor predicho y el real:

$$L_2(y_{true}, y_{\text{pred}}) = (y_{true} - y_{\text{pred}})^2$$

La pérdida L_2 mide el *mean squared error* (MSE).

1.12.4. *Smooth* L_1

La función de pérdida *smooth* L_1, o también llamada *Huber loss*, se define como:

$$L_\delta(y_{true}, y_{\text{pred}}) = \begin{cases} 0{,}5x^2 & \text{if } |x| \leq \delta \\ \delta\,|x| - 0{,}5\delta^2 & \text{if } |x| > \delta \end{cases}$$

donde $x = y_{true} - y_{\text{pred}}$ y δ es un hiperparámetro y corresponde al punto en que la pérdida cambia de cuadrática a lineal. Esta función de pérdida es menos sensible a *outliers* que la pérdida L_2.

1.13. Métricas

- **Intersection over Union (IoU)**: mide el traslape entre 2 cajas o *bounding boxes*, los cuales, en el contexto de detección de objetos, serían la caja de *ground truth* (la que encierra el objeto real) y la caja predicha. Con esta métrica se decide si una detección es valida (*True Positive*) o no (*False Positive*). El IoU está dado como la razón entre el área de la intersección y de la unión de ambas cajas (ver figura 1.24)

Figura 1.24: Ejemplo de Intersection-over-Union (IoU)

Las predicciones realizada por un modelos se clasifican en:

- Verdadero Positivo (TP): si predice correctamente la clase (clase predicha = clase real) y IOU $\geq$ *threshold*.

- Falso Positivo (FP): si la predicción es incorrecta (clase predicha $\neq$ clase real) o IOU $<$*threshold*.

- Falso Negativo (FN): un objeto no detectado.

donde *threshold* denota un umbral para el IoU a partir del cual una detección se considera invalida. Lo más común es fijar dicho umbral entre 0.5 y 0.7.

- ***Accuracy***: Se define como el número de imágenes reconocidas correctamente (*string* predicho = *string* real) dividido entre el total de imágenes.

Capítulo 2

Hipótesis y objetivos

El presente tema de memoria consiste en una investigación acerca de un nuevo enfoque para la resolución del problema de reconocimiento de secuencias de dígitos de largo variable en imágenes de cheques reales (reconocimiento CAR), utilizando técnicas de *deep learning*.

El enfoque propuesto se basa en abordar el problema como uno de detección de objetos, en donde cada dígito representa una clase a detectar. La hipótesis es que los modelos de detección de objetos mejoran la efectividad del reconocimiento de secuencias numéricas manuscritas.

De este modo, el objetivo general de este trabajo es investigar el impacto de los modelos de detección de objetos basados en CNNs para el problema reconocimiento de secuencias de dígitos manuscritos en imágenes, en particular, montos en cheques bancarios reales (reconocimiento CAR). Cabe señalar que en este trabajo solo interesa reconocer dígitos, no otros símbolos como peso, punto o coma.

Los objetivos específicos son:

1. Diseñar arquitecturas basada en redes convolucionales (CNN) específicas para reconocimiento CAR.
2. Programar una herramienta utilizando tensorflow (un *framework* de *deep learning*) la cual permita entrenar y probar los distintos modelos a utilizar. La idea es no depender de una implementación ya hecha, lo cual puede sesgar la comparación entre los métodos, ya que cada implementación (oficial o no oficial) tiene su propia forma de procesar la data, distinta red *backbone*, etc. Luego, con un *framework* común para los distintos algoritmos la comparación es más justa.
3. Construir un dataset para entrenar modelos de detección de objetos aplicados a reconocimiento CAR.
4. Evaluar modelos de detección de objetos para reconocimiento CAR.

Capítulo 3

Desarrollo

3.1. Diseño de arquitecturas

Al igual que la mayoría de modelos basados en CNN para extraer características de las imágenes de entrada, la arquitectura utilizada como red *backbone* corresponde a una ResNet de 17 capas como se muestra en la tabla 3.1, donde la tupla $[\#\text{maps:c}, ka \times a, sb \times b]$ denota una capa convolucional con kernel $a \times a$, c filtros y *stride* b. Si no se especifica el *stride* es 1. Cabe señalar que solo el primer bloque de cada grupo aplica el *stride* s2x2 (bloque proyectado).

Se añade una capa de *batch normalization* a todas las capas convolucionales. Esto estabiliza el entrenamiento, acelera la convergencia y ayuda a regularizar el modelo evitando el sobreajuste. Como función de activación se usa ReLU, a menos que se mencione lo contrario.

En las siguientes secciones se describe como, a partir de la arquitectura de la tabla 3.1 se pueden acoplar distintas subredes las cuales realizarán tipos de predicciones diferentes como clasificación, regresión, secuencial, etc. Se probaron otras arquitecturas ResNet más grandes,

Tabla 3.1: Red residual usada como *backbone* para extraer características de la imagen de entrada.

Layer	**Output size**	**17-layer**
Input image	80×400	
conv1	40×200	[#maps:64, k5×5, $s1 \times 1$]
C1	40×200	$\begin{bmatrix}\#\text{maps:64}, k3 \times 3, s1 \times 1 \\ \#\text{maps:64}, k3 \times 3, s1 \times 1\end{bmatrix} \times 2$
C2	20×100	$\begin{bmatrix}\#\text{maps:128}, k3 \times 3, s2 \times 2 \\ \#\text{maps:128}, k3 \times 3, s1 \times 1\end{bmatrix} \times 2$
C3	10×50	$\begin{bmatrix}\#\text{maps:256}, k3 \times 3, s2 \times 2 \\ \#\text{maps:256}, k3 \times 3, s1 \times 1\end{bmatrix} \times 2$
C4	5×25	$\begin{bmatrix}\#\text{maps:512}, k3 \times 3, s2 \times 2 \\ \#\text{maps:512}, k3 \times 3, s1 \times 1\end{bmatrix} \times 2$

pero no se observó una mejora significativa (mayor a 1 %), si no más bien que el tiempo de procesamiento era mayor.

Las arquitecturas que se probaron en este trabajo son las siguientes:

- CNN-LSTM-CTC / CNN-CTC [27]
- RetinaNet [11].
- YOLO [16].
- FCOS [23].

3.1.1. Arquitectura 1: CNN-LSTM-CTC / CNN-CTC

Esta arquitectura corresponde al estado del arte en el reconocimiento de secuencias en imágenes. La ventaja de esta es que es entrenable extremo a extremo, lo cual significa que la red se alimenta de imágenes y sus respectivas secuencias de *labels*, en este caso dígitos.

La figura 3.1 muestra la arquitectura CNN-LSTM-CTC diseñada. La red consiste de una red ResNet de 4 bloques residuales (ver tabla 3.1) para extraer características discriminativas de la imagen de entrada en forma de mapas de características. Estos mapas son convertidos en una secuencia de vectores de características por medio de lo que se denomina transposición de dimensionalidad, que no es más que la concatenación por columna de los mapas de características (ver figura 3.2). Esta secuencia de vectores ingresa a la red recurrente formada por 2 capas BiLSTM, para finalmente pasar por una capa de transcripción dada por la CTC la cual computará la *loss* y realizará la predicción final.

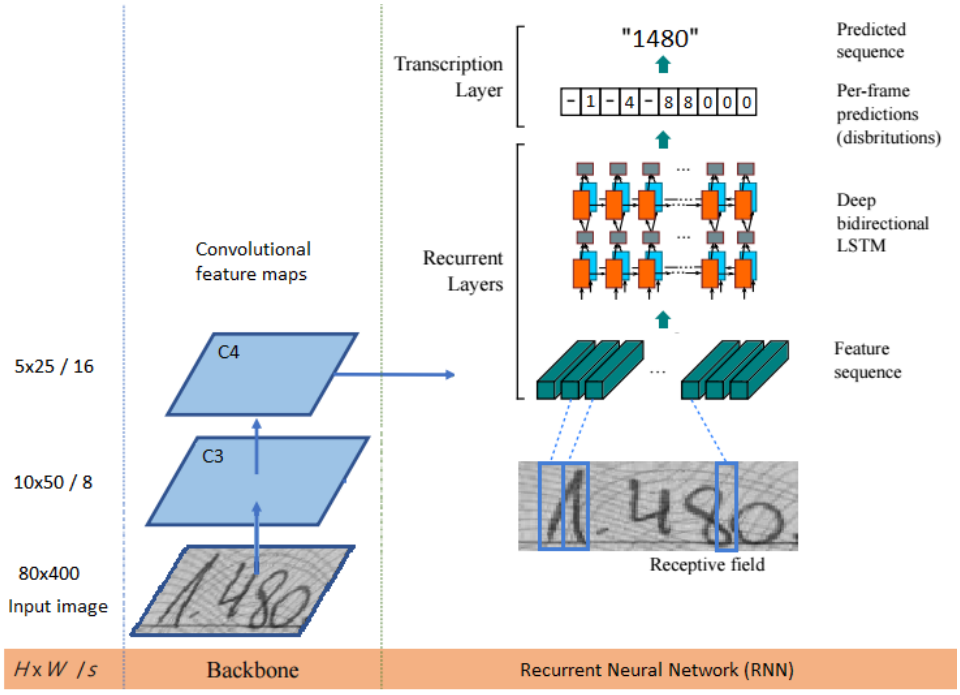


Figura 3.1: Arquitectura CNN-LSTM-CTC para reconocimiento CAR.

La elección de los hiperparámetros de la red es similar al de [27]:

- Tamaño de la imagen de entrada: 80×400 ($H \times W$).
- El número de filtros de cada uno de los 4 bloques residuales es de 64, 128, 256 y 512 respectivamente.
- El tamaño del kernel de todas las convoluciones es 3×3 a excepción de la primera que es de 5×5.
- Número de unidades ocultas de las BiLSTMs (LSTM bidirecional): 100.
- Número de capas BiLSTM: 2
- Tamaño de batch: 100.
- Optimizador: Adam.
- Tasa de aprendizaje: 0.001.

Se realizó un experimento consistente en probar el efecto de la LSTM en el desempeño del modelo, para lo cual esta fue removida de la red. Este nuevo modelo se denomina CNN-CTC el cual consta de 3 partes: las capas convolucionales (CNNs), la capa de transposición de dimensionalidad de las características y la capa de salida. La figura 3.2 ilustra la capa de transposición de dimensionalidad de las características.

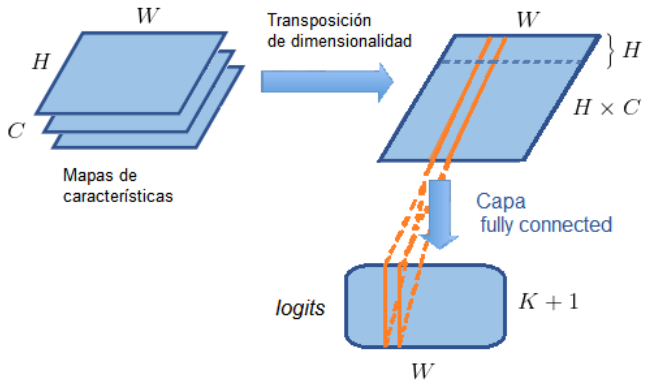


Figura 3.2: Esquema de la capa de transposición de dimensionalidad de las características del modelo CNN-CTC.

En la figura 3.2, los mapas de características representan la salida de la red convolucional y los *logits* corresponden a la salida de la red. Estos *logits* se pueden representar como una matriz de $(K+1) \times W$ o una secuencia de W vectores de largo $(K+1)$, donde $K+1$ es el número de clases más el símbolo blanco (-) y W es el ancho del mapa de características. Son estos *logits* los que pasan por la capa de salida CTC para calcular la *loss* en el entrenamiento y decodificar la salida en la inferencia.

3.1.2. Arquitectura 2: RetinaNet

La figura 3.3 muestra la arquitectura de la red propuesta para reconocimiento CAR basada en RetinaNet. En ella se puede observar la red *backbone* compuesta de los *feature maps* C1,

C2, C3 y C4. A esta red se acoplan 2 subredes: el clasificador y el regresor, las cuales retornan un tensor de $H \times W \times KA$ y $H \times W \times 4A$, respectivamente, donde H y W son el alto y ancho del *feature map* respectivo, K es el número de clases, que para este problema es $K = 10$, y por último, A es el número de *anchors* por nodo, el cual está dado por el número de variaciones de estos.

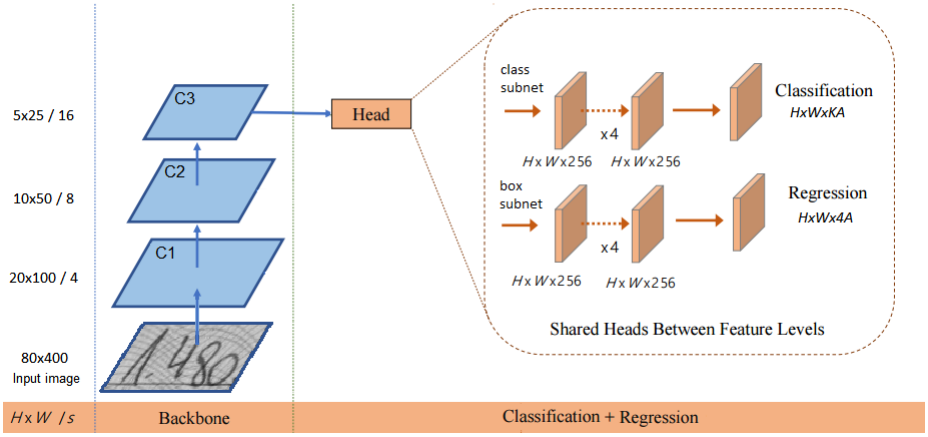


Figura 3.3: RetinaNet adaptada a reconocimiento CAR.

Durante el entrenamiento estos tensores se comparan con el *ground truth*, el cual, en el caso del clasificador consiste en un tensor de 1s y 0s, donde los 1s corresponden a los índices de las clases asignadas a los *anchors* positivos, y los 0s al resto. Para el regresor, en cambio, se usa la parametrización de las 4 coordenadas dada por la ecuación 1.5

Durante el entrenamiento se computa la pérdida de clasificación, dada por la *focal loss* [11], donde se usa $\alpha = 0{,}25$ y $\gamma = 2$ siguiendo [11]. Mientras que la pérdida del regresor está dada por la *Huber loss* con $\delta = 0{,}1$, aplicada sobre el error. Luego, la pérdida total está dada por la suma de ambas pérdidas: $L = L_{cls} + L_{reg}$.

En cuanto a los *anchors*, estos fueron diseños a partir de un cuidadoso análisis por medio de *k-means* entre otras técnicas, obteniéndose que con 3 escalas y 2 razones de aspecto (1:1, 1:2) por nivel es suficiente, lo cual da un total de $A = 6$ *anchors* por nodo.

A modo de ilustración, la figura 3.4 muestra como se ven dichos *anchors* (en rojo) en acción sobre una imagen real. Por claridad solo se muestran los *anchors* positivos, o sea, aquellos asignados a un objeto ($IoU \geq 0{,}5$).

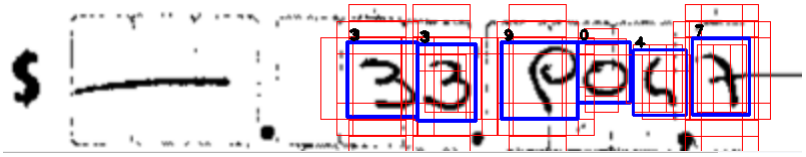


Figura 3.4: *Anchors* positivos en imagen de cheque real en rojo y *ground truth* en azul.

3.1.3. Arquitectura 3: YOLO

Se entrenó YOLO con *backbone* similar al de la tabla 3.1, donde el *feature map* para hacer la predicciones corresponde a C3. Este mapa de características pasa por 3 convoluciones más con kernel 3x3 y 256 filtros, y una última convolución con $A(5 + K)$ filtros, donde A es el número de *anchors* por nodo y K el número de clases.

Los *anchors* se obtienen por medio de *k-means* como se describe en [15]. En total se utilizaron $A = 6$ *anchors* por nodo. Luego, como son $K = 10$ clases, se tiene que la salida de la red consiste en un tensor de dimensión $10 \times 50 \times 90$, donde $H = 10$, $W = 50$ son las dimensiones del mapa de características y $A(5 + K) = 90$ es el número de canales.

En el entrenamiento la *loss* está dada por la suma de la *loss* de clasificación (regresión logística), la *loss* del regresor (ℓ_2 *loss*) y la de *objectness* (entropía cruzada binaria o BCE).

En la inferencia, el *score* final de las detecciones se calcula multiplicando la probabilidad predicha con su correspondiente valor de *objectness*.

Algunos hiperparámetros de la red son los siguientes:

- Tamaño de la imagen de entrada: 80×400 ($H \times W$).
- El número de filtros de cada uno de los 3 bloques residuales es de 64, 128, 256 respectivamente.
- El tamaño del kernel de todas las convoluciones es 3×3 a excepción de la primera que es de 5×5.
- Función de activación Leaky ReLU.
- Tamaño de batch: 100.
- Optimizador: Adam.
- Tasa de aprendizaje: 0.001.
- Umbral de NMS: 0.6
- Umbral de score: 0.3

3.1.4. Arquitectura 4: FCOS

La arquitectura de FCOS es similar a la de RetinaNet (ver figura 3.3), con la diferencia que ahora la salida del clasificador es de $H \times W \times K$, donde K es el número de clases, mientras que el regresor es de tamaño $H \times W \times 4$. Nótese que H y W son las dimensiones del mapa de características que en este caso es de $H = 10$ y $W = 50$.

Además, como se señala en [23], se agregó una rama más para predecir la *centerness*, la cual, como se afirma, reduce considerablemente las detecciones de baja calidad. La *centerness* predice la desviación de un píxel al centro del *bounding box* correspondiente. Su valor predicho es luego usado en la inferencia para bajarle el peso a las detecciones de baja calidad. La

centerness se define como:

$$\text{centerness}^* = \sqrt{\frac{\text{mín}(l^*, r^*)}{\text{máx}(l^*, r^*)} \times \frac{\text{mín}(l^*, r^*)}{\text{máx}(l^*, r^*)}}$$

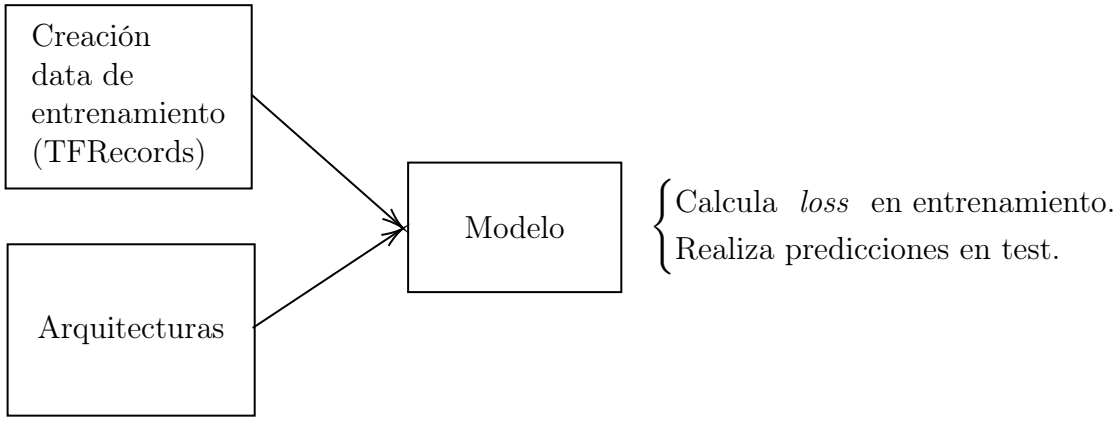
donde (l^*, t^*, r^*, b^*) corresponde al vector objetivo para una cierta posición. Así, el valor de la *centerness* decae de 1 a 0 conforme el punto se aleja del centro del objeto, y es entrenado con la entropía cruzada binaria (BCE). Luego, en el entrenamiento, la pérdida total está dada por la suma de la pérdida de clasificación (*focal loss*), la pérdida del regresor (*IoU loss*) y la de *centerness* (BCE).

En la inferencia, el *score* final de las detecciones se calcula multiplicando la probabilidad predicha con su correspondiente valor de *centerness*.

3.2. Construcción de herramienta

Se programó una herramienta (mini *framework*) en Python usando tensorflow 1.2 para entrenar y probar los distintos modelos diseñados para el reconocimiento CAR.

El programa consta de los siguientes módulos principales:



El módulo de *Creación de data de entrenamiento* crea los llamados *TFRecords* que no son más que la data necesaria para entrenar los modelos almacenada en un formato binario que permite la lectura eficiente por parte del modelo durante el entrenamiento. El esquema de la data depende de la arquitectura. Por ejemplo, para el modelo RetinaNet la data consta de imágenes y *anchors* definidos como un arreglo de su clase y su vector objetivo $[t_x, t_y, t_w, t_h]$. El módulo *Arquitecturas* define la arquitectura del modelo a entrenar. Las arquitecturas programadas son: ResNet, FPN (*Feature Pyramid Network*), RetinaNet, YOLO, FCOS, CNN-LSTM-CTC, CNN-CTC, entre otras.

Por último, el módulo *Modelo* es el que entrena y realiza las predicciones. Durante el entrenamiento se alimenta de los *TFRecords*, calcula la *loss* y actualiza los pesos con el algoritmo de *backpropagation*.

3.3. Datasets

Una de los aspectos que diferencia a los métodos basados en *deep learning* de los tradicionales basados en segmentación es la cantidad de datos requerida para entrenarlos. En este trabajo se usaron diferentes datasets para entrenar y testear los modelos. Estos datasets son los siguientes:

- **ORAND-CAR-A**: Este dataset cuenta con 2009 y 3984 imágenes de entrenamiento y prueba respectivamente. Las imágenes de este dataset se obtuvieron del campo CAR (*Courtesy Amount Recognition*) de cheques reales (banco uruguayo). Al ser este dataset público la partición entre *train* y *test* estaba ya definida.
- **ORAND-CAR-B**: Este dataset cuenta con 3000 y 2926 imágenes de entrenamiento y prueba, respectivamente. Las imágenes de este dataset pertenecen a cheques reales (banco chileno). Al ser este dataset público la partición entre *train* y *test* estaba ya definida. Este dataset es similar a ORAND-CAR-A, pero con imágenes de calidad un poco más baja.
- **CVL**: El dataset *Computer Vision Lab Handwritten Digit String* o CVL consta de 6698 imágenes las cuales solo son usadas para test. Este datasets tiene un fondo limpio, pero la variabilidad de estilos de escritura (300 escritores distintos) hacen de este un verdadero desafío.
- **Ban1**: Este dataset cuenta con 3150 y 463 imágenes de entrenamiento y prueba, respectivamente. Las imágenes de este dataset pertenecen a cheques reales (banco chileno). Este dataset es privado perteneciente a ORAND S.A. y del cual no se contaba con las etiquetas de las imágenes, de modo que hubo un trabajo de etiquetado de estas así como del desarrollo de un pequeño *software* que facilitara dicho etiquetado.
- **Ban2**: Este dataset cuenta con 500 imágenes de cheques reales (banco chileno). Este dataset solo será de prueba.
- **Ban3**: Este dataset cuenta con 4000 imágenes de entrenamiento y 500 de test. Las imágenes de este dataset pertenecen a cheques reales (banco uruguayo). Dado que las imágenes no estaban etiquetadas, parte del trabajo consistió en generar las anotaciones respectivas.
- **Data-sintética-MNIST**: Se creó un dataset de imágenes sintéticas con dígitos tomados de MNIST, la razón para incluir este conjunto de datos es para tener más variabilidad en los dígitos y considerar otros estilos de escritura (en este caso el americano en que, por lo general, el 1 es una raya y el 7 suele no llevar la raya del medio), siendo este el más parecido a CVL. En total son 1000 imágenes extra para entrenar.
- **Data-sintética-printed**: Se creó un dataset de imágenes sintéticas con dígitos tipo impreso, la razón para incluir este conjunto de datos es para no sesgar al modelo con números manuscritos. En total son 1000 imágenes más para entrenar.

La tabla 3.2 resume la información anterior. En total, se usaron 14159 imágenes para entrenar la red. La figuras 3.5, 3.6, 3.7, 3.8, 3.9, 3.10, 3.11 y 3.12 ilustran muestras de cada uno de los datasets anteriores.

Tabla 3.2: Datasets utilizados para entrenar y testear los modelos.

	CAR-A	CAR-B	CVL	Ban1	Ban2	Ban3	Generated MNIST	Generated Printed	Total
Training	2009	3000	0	3150	0	4000	1000	1000	14159
Testing	3984	2926	6698	463	500	500	0	0	15071

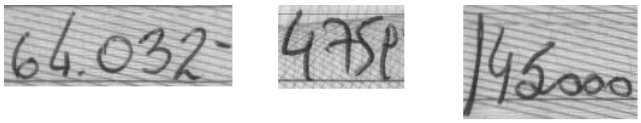


Figura 3.5: Muestras de CAR-A.

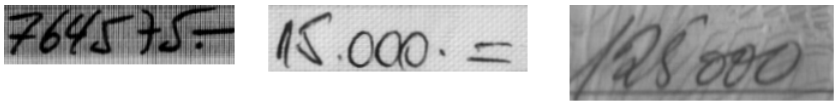


Figura 3.6: Muestras de CAR-B.

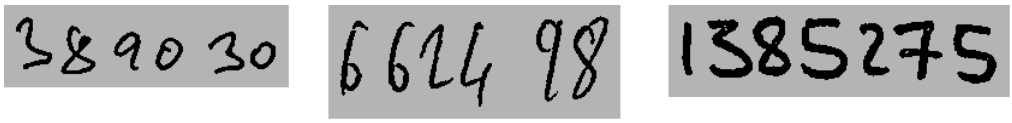


Figura 3.7: Muestras de CVL.

Figura 3.8: Muestras de Ban1.

Figura 3.9: Muestras de Ban2.

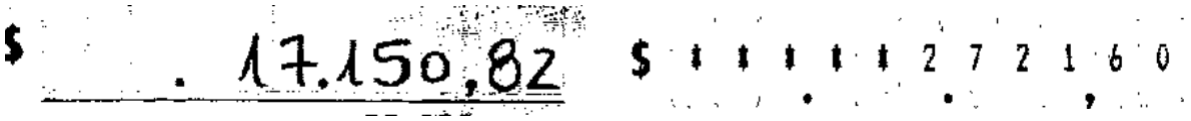


Figura 3.10: Muestras de Ban3.

Figura 3.11: Muestras de data generada en base a MNIST.

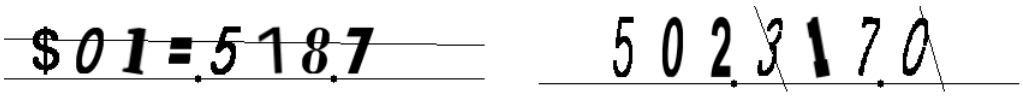


Figura 3.12: Muestras de data generada de números impresos.

3.4. Procesamiento de la data

Como preprocesamiento todas las imágenes son convertidas a escala de grises, se redimensionan a 80x400 (alto x ancho) y se les resta la imagen promedio, además se divide por 255 para que sus valores estén entre [-1,1]. A fin de tener más data se aplicó *data augmentation offline*, o sea, antes de entrenar, por cada imagen se crearon 10 más con distintas variaciones aleatorias. Se aplicaron transformaciones afines tales como rotación, translación, torsión (*shear*) y escalamiento. Además, se aplicaron variaciones aleatorias de brillo y contraste. Se decidió no hacerlo *online* o dinámicamente ya que el entrenamiento se hacía considerablemente más lento (especialmente con RetinaNet en que hay que volver a recalcular los sobrelapes de los *anchors* con el *ground truth*) lo cual es costoso computacionalmente. Así, con el *data augmentation* el tamaño del conjunto de entrenamiento crece a 141590 imágenes. Cabe señalar el efecto positivo del *data augmentation* que aumenta el desempeño en hasta un 4 %.

Dependiendo del método se aplicó un postprocesamiento diferente:

- En los modelos de detección de objetos (RetinaNet, YOLO, FCOS) se aplica *Non-max Suppression* (NMS) para filtrar los *bounding boxes* predichos. Posteriormente, se ordenan estos *bounding boxes* junto a sus *labels* asociadas en orden creciente de su coordenada x, de modo que la predicción final viene dada por la concatenación de los *labels* ordenados.
- En los métodos basados en la CTC el postprocesamiento viene dado por el algoritmo de decodificación de la salida de la red. Este algoritmo es de tipo *greedy* y consiste en seleccionar la clase con mayor probabilidad para cada *logit* (*best path*), a continuación se eliminan los caracteres repetidos consecutivos y finalmente se eliminan los símbolos blancos (-). Por ejemplo, para la secuencia $\{1-22--211\}$ se tiene $\{1-22--211\} \rightarrow \{1-2-21\} \rightarrow \{1221\}$. Esta última secuencia corresponde a la predicción final. Nótese que el orden de eliminación es importante (primero los repetidos y luego el símbolo blanco).

Cabe señalar que se realizó una pequeña modificación al algoritmo NMS la cual consiste en reemplazar la definición del IoU como comparador entre *bounding boxes* por el IoM (*Intersection over Minimum*), es decir, en lugar de dividir por el área de la unión se divide por la menor área. Se comprueba que esta nueva definición mejora un poco la tasa de aciertos (en torno a un 0.8 %) lo cual se explicaría por el hecho que el algoritmo original permite que hayan objetos dentro de otros, pero este no es el caso de los dígitos ya que no debiesen haber dígitos dentro de otros.

Capítulo 4

Experimentos y resultados

4.1. Resultados experimentales

En los experimentos realizados se consideró un tamaño de imagen de entrada de 400x80 (ancho x alto). El código está implementado con Tensorflow 1.2. Los experimentos fueron realizados en una GPU GeForce GTX TITAN X con 11363 MB de memoria. Todas las redes se entrenaron con el algoritmo de optimización ADAM y una tasa de aprendizaje inicial de 0.001. Por otra parte, la métrica a utilizar para evaluar los modelos corresponde al *accuracy* definido como el número de imágenes reconocidas correctamente entre el total. Los resultados obtenidos con los distintos modelos entrenados se muestra en la tabla 4.1.

Tabla 4.1: Resultados del *accuracy* en el conjunto de test de los distintos modelos entrenados.

Método	CAR-A	CAR-B	CVL	Ban1	Ban2	Ban3	Tiempo
RetinaNet	95.4 %	95.0 %	77.4 %	89.0 %	78.4 %	87.1 %	106 FPS
CNN-LSTM-CTC	94.6 %	95.5 %	77.5 %	85.5 %	83.2 %	91.2 %	31 FPS
CNN-CTC	96.1 %	95.8 %	78.9 %	89.0 %	86.5 %	91.7 %	118 FPS
YOLO	96.8 %	96.4 %	81.5 %	92.4 %	82.0 %	93.2 %	104 FPS
FCOS	96.1 %	96.3 %	76.4 %	89.8 %	77.2 %	90.0 %	84 FPS

4.2. Análisis del error

Se realizó un análisis de error para comprender que tipo de errores cometieron los distintos modelos. En las siguientes figuras las imágenes de la izquierda corresponden a predicciones correctas, mientras que las de la derecha a predicciones fallidas. Por cada *bounding box* predicho (en rojo) se muestra el dígito predicho y el *score* asociado. La figura 4.1 muestra algunos resultados obtenidos con RetinaNet. La figura 4.2 muestra algunos resultados obtenidos con YOLO. En la figura 4.3 se muestran ejemplos de predicciones realizadas por FCOS. Por último, la figura 4.5 muestra ejemplos de imágenes reconocidas por el modelo CNN-LSTM-CTC.

A modo de comparación, la figura 4.4 muestra las predicciones realizadas por YOLO y FCOS. La primera columna corresponde a las predicciones de YOLO, mientras que la segunda corresponde a las de FCOS. Las dos primeras filas muestran imágenes predichas correctamente por YOLO, pero no por FCOS, mientras que las dos últimas muestran imágenes predichas correctamente por FCOS, pero no por YOLO.

La figura 4.6 muestra una comparación entre predicciones de YOLO vs. CNN-CTC. La primera columna corresponde a las predicciones de YOLO, la segunda a las de CNN-CTC y la tercera al *grund truth*. Las primeras 2 filas corresponden a predicciones correctas de YOLO, pero fallidas por CNN-CTC, mientras que las dos siguientes corresponden a predicciones correctas de CNN-CTC, pero fallidas por YOLO, y por último, las 2 últimas corresponden a predicciones fallidas tanto por YOLO como por CNN-CTC.

En base a los resultados presentados y revisando algunas muestras de los errores cometidos por los distintos modelos, se puede concluir que, en general, todos los modelos tienen un desempeño similar, siendo algunos ligeramente mejores que otros, y que los errores que cometen son similares entre sí. Dichos errores se asocian a casos difíciles incluso para una persona como, por ejemplo, en la figura 4.1 las imágenes incorrectas de las filas 2 y 3. En el primer caso la predicción es 102266, pero el real es 102246, mientras que en el segundo apenas se distingue el 9 siendo reconocido como 8. Claramente muchos de estos errores, aunque no todos, son debido a una mala calidad de las imágenes.

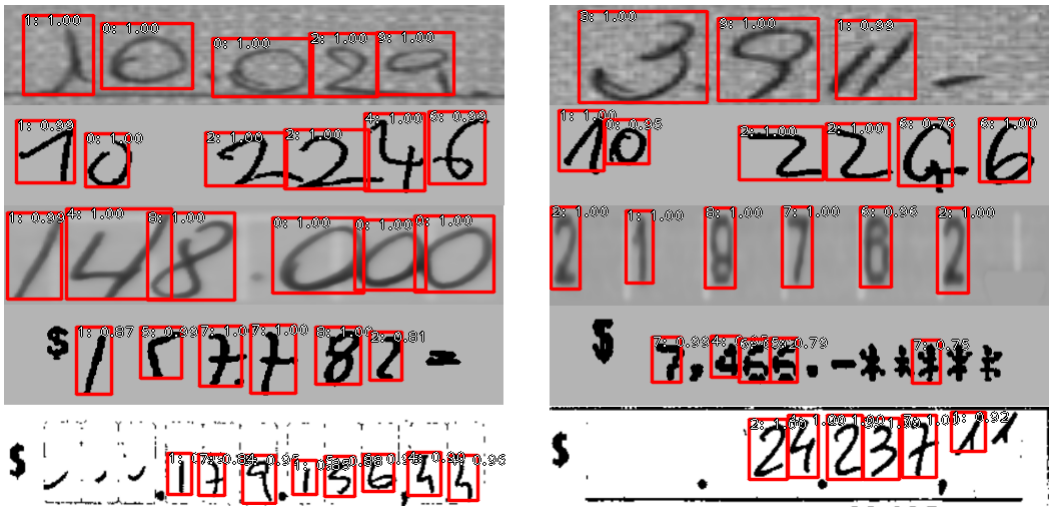


Figura 4.1: Predicciones de RetinaNet. A la izquierda las correctas y a la derecha las incorrectas.

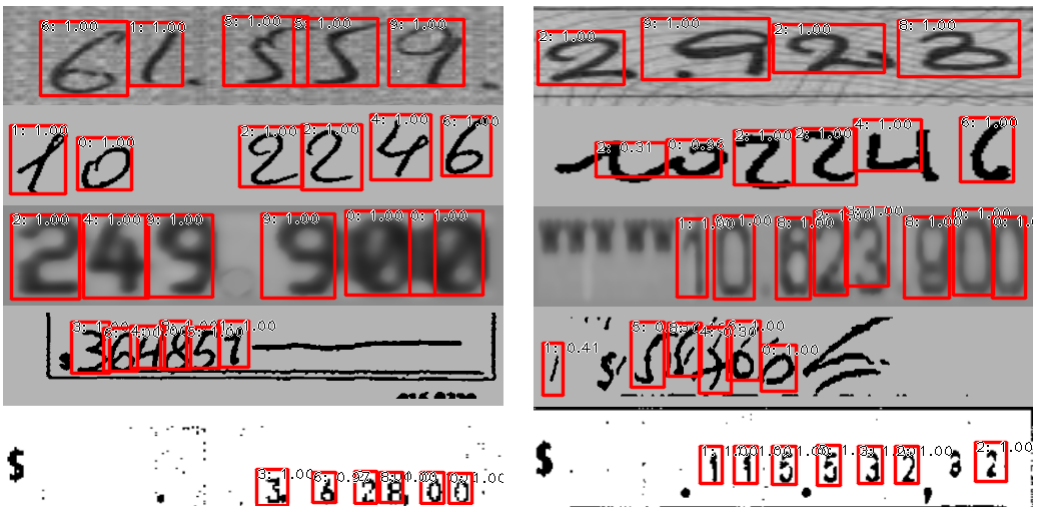


Figura 4.2: Predicciones de YOLO. A la izquierda las correctas y a la derecha las incorrectas.

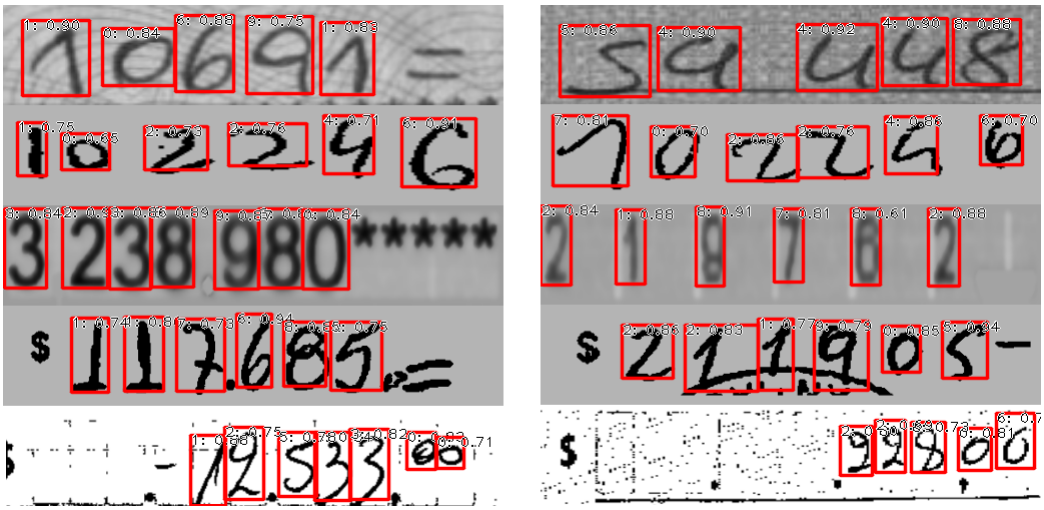


Figura 4.3: Predicciones hechas por FCOS. A la izquierda las correctas y a la derecha las incorrectas.

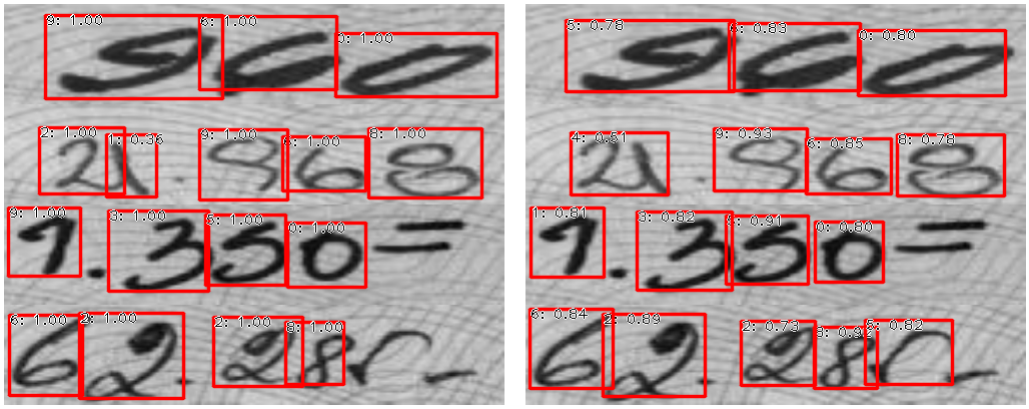


Figura 4.4: Comparación entre predicciones de YOLO (a la izquierda) y FCOS (a la derecha).

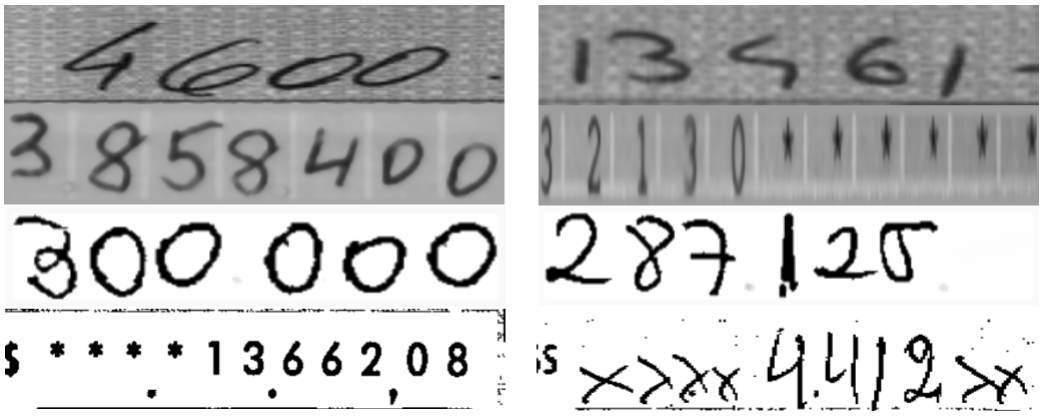


Figura 4.5: Predicciones hechas por CNN-LSTM-CTC. A la izquierda las correctas y a la derecha las incorrectas.

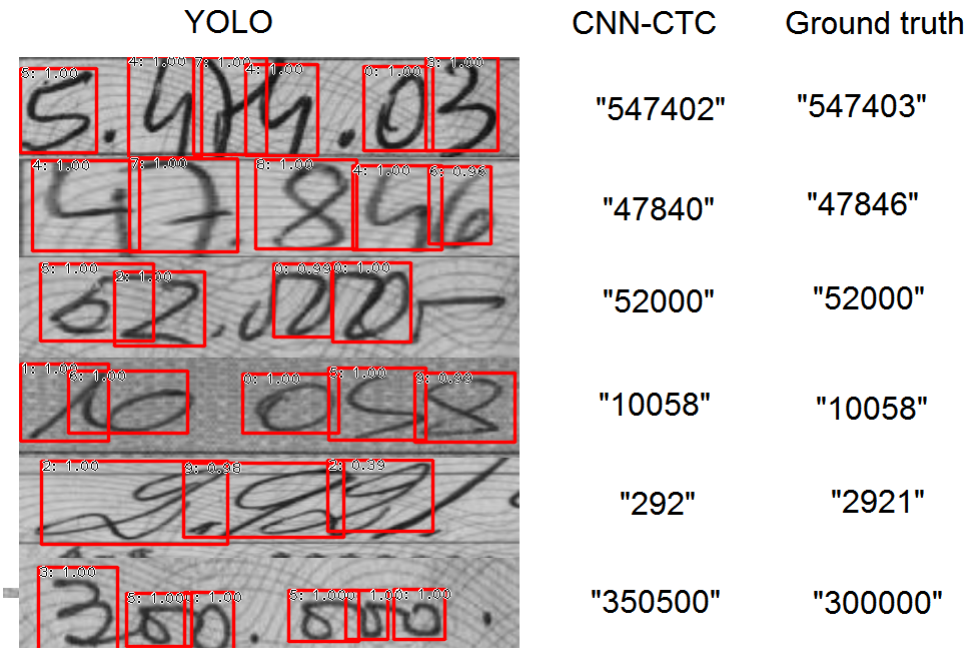


Figura 4.6: Comparación entre predicciones de YOLO vs. CNN-CTC y el *ground truth*.

Capítulo 5

Análisis y discusión

La figura 5.1 resume el *performance* de los detectores/reconocedores analizados, donde el *accuracy* (medido en porcentaje) corresponde al promedio en los 6 *datasets*. La tabla 5.1 muestra las tasas de reconocimiento de diferentes modelos del estado del arte y los métodos propuestos en los datasets públicos CAR-A, CAR-B y CVL. En particular, se seleccionaron los métodos con mejor desempeño, siendo estos: YOLO y CNN-CTC.

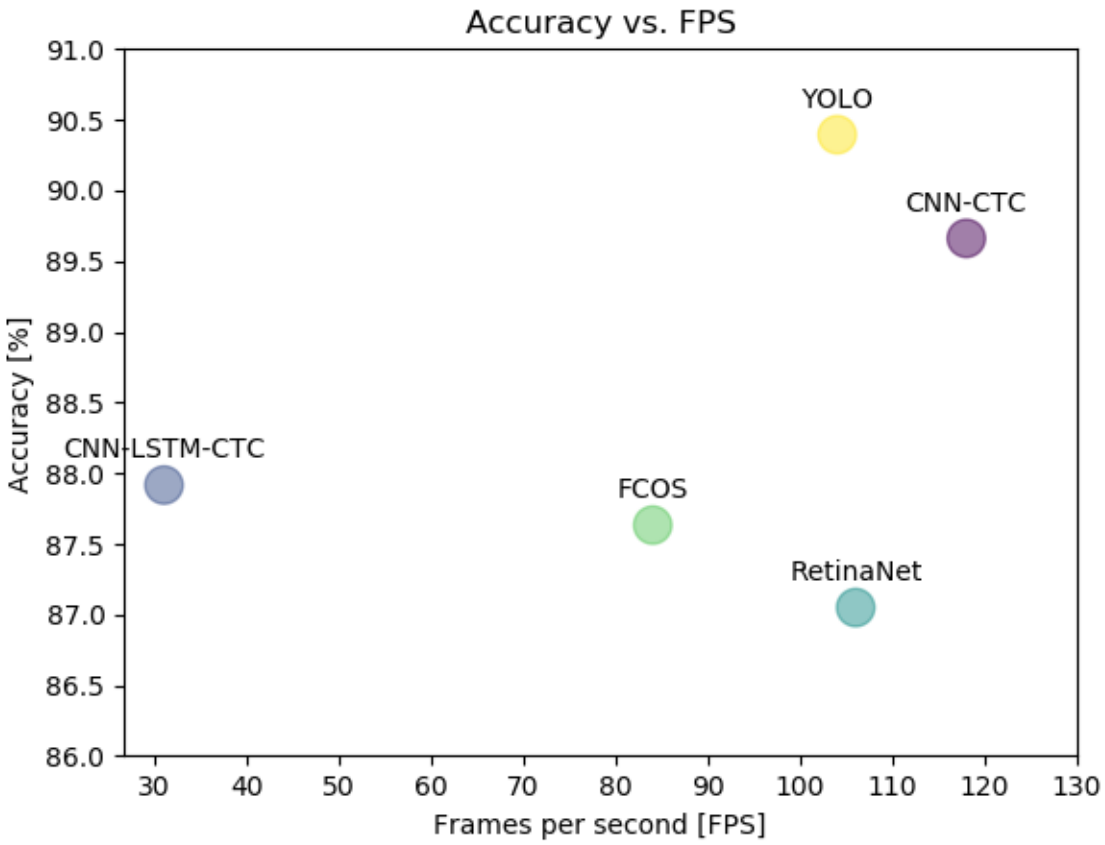


Figura 5.1: Desempeño de los métodos analizados para reconocimiento CAR.

Tabla 5.1: Comparación con el estado del arte (en negrita las propuestas).

Métodos	CAR-A	CAR-B	CVL
Tebessa I [2]	0.3705	0.2662	0.5930
Tebessa II [2]	0.3972	0.2772	0.6123
Singapore [2]	0.5230	0.5930	0.5040
Pernambuco [2]	0.7830	0.7543	0.5860
BeiJing [2]	0.8073	0.7013	**0.8529**
CRNN [19]	0.8801	0.8979	0.2601
Saabni [18]	0.8580		-
Zhan [27]	0.8975	0.9114	0.2707
YOLO	**0.9678**	**0.9645**	0.8150
CNN-CTC	**0.9614**	**0.9583**	0.7887

En lo que sigue se realizará un análisis del desempeño de los modelos entrenados tanto por el *accuracy* alcanzado como por el tiempo de inferencia medido en *frames* por segundo o FPS. Este último factor es también muy importante ya que en la práctica no es útil un método con excelente *accuracy*, pero demasiado lento, de modo que es preferible un método con un poco menos de precisión, pero mayor rapidez. Dicho esto, cabe señalar que los tiempos son solo una referencia ya que son completamente dependientes del computador utilizado, así como de la implementación de los algoritmos usados. Por ejemplo, una implementación eficiente del algoritmo NMS puede incrementar la rapidez en hasta 30 FPS.

Como se puede observar en la figura 5.1, el mejor método en cuanto a *accuracy* alcanzado corresponde a YOLO (v3) con un 90.39 % y velocidad de 104 FPS. Por otra parte, el método más rápido corresponde a CNN-CTC con 118 FPS alcanzando un *accuracy* de 89.66 %. Nótese que CNN-CTC tiene un *accuracy* 0.73 % menos que YOLO, pero supera su rapidez en 14 FPS.

Por otra parte, RetinaNet no logra superar a YOLO, ni tampoco a FCOS. Esto sin duda resulta sorprendente por cuanto era de esperar que RetinaNet tuviera un mejor desempeño que YOLO a costa de una menor rapidez. Si bien RetinaNet resultó ser igual de rápido que YOLO, este último lo superó en *accuracy* 3.34 %.

Se probó la versión multiescala de RetinaNet con 2 escalas usando una FPN, pero no se observó una mejora significativa en el desempeño (87.05 % sin multiescala vs. 87.48 % con multiescala), sino más bien una disminución en la rapidez de hasta 30 FPS. Esto en cierto modo valida el supuesto que este problema no ameritaba de un análisis multiescala ya que, por lo general, los dígitos no presentan grandes variaciones de tamaño, además que el tamaño de la imagen de entrada (400x80) es relativamente pequeño si se compara al de los de problemas como COCO [12] en que, por lo general, son de entre 800 y 1333.

En cuanto a FCOS, este método logró un desempeño escasamente superior a RetinaNet (0.58 %), pero de igual modo inferior a YOLO (2.76 %). Dado que es un método nuevo requiere quizás de algunas mejoras para poder competir con YOLO y demostrar que los *anchors* realmente no son necesarios.

El caso más interesante es el de la red recurrente. Se probaron dos modelos: i) CNN-LSTM-CTC y ii) CNN-CTC. Se sabe que el estado del arte en el problema de reconocimiento manuscrito consiste en una red que combina una CNN para extraer características, una LSTM para el modelamiento secuencial y una capa de transcripción dada por la CTC para calcular la *loss* y la predicción final. Con este *framework* se reportan tasas de reconocimiento cercanas al 90 %. Sin embargo, pocos autores se han cuestionado cuanto realmente aportan las capas recurrentes en el desempeño del reconocedor. Para ello, se experimentó sacando la LSTM y conectando la capa CTC a la salida de la CNN (por medio de un mecanismo de transposición de dimensiones), modelo que se llamó CNN-CTC. Sorprendentemente se obtuvo que el modelo sin LSTM es capaz de aprender mejor que su versión recurrente, superándola en un 1.74 % en *accuracy*, y a una tasa considerablemente mayor pasando de 31 FPS a 118 FPS. Esto de por si representa un hallazgo importante en el área de reconocimiento de secuencias basadas en imágenes en la que se suponía clave el rol de la LSTM para modelar correctamente la secuencia dada por la imagen de entrada.

Del experimento anterior se puede concluir que no es necesario un modelamiento secuencial dado por la LSTM en el problema de reconocimiento de dígitos, ya que no mejora el desempeño final y solo hace más lenta la red. Las redes recurrente, y en particular la LSTM, tienen justamente la desventaja de ser costosas para calcular su salida lo cual las hace mucho más lentas en comparación a las redes convolucionales cuyas operaciones son paralelizables y por lo tanto aprovechan muy bien la capacidad de las GPUs, en cambio, las recurrentes al ser secuenciales no permiten tal *speed-up*. Cabe señalar que en los experimentos se usó una implementación eficiente de la LSTM que sí aprovecharía la GPU (función `CudnnCompatibleLSTMCell` de tensorflow), pero aún así sigue siendo considerablemente más lenta que una CNN. Si bien las redes recurrentes, y en particular las LSTMs, han reportado numerosos éxitos en la literatura, solo por mencionar que el traductor de Google las utiliza, pero en el caso de reconocimiento de secuencias de dígitos en imágenes se demuestra que no son realmente necesarias como la mayoría de trabajos que han abordado el problema afirman. Quizás sea por el hecho de que al no existir una dependencia entre dígitos no se justifica el modelamiento de los contextos pasados y/o futuros.

Otra desventaja del modelamiento recurrente es que la memoria explícita añade más pesos a cada nodo los cuales deben ser entrenados. Esto incrementa la dimensionalidad del problema, y potencialmente hace más difícil encontrar una solución óptima, lo cual explicaría en cierto modo la diferencia de 1.74 % en *accuracy*.

Otro análisis interesante es comparar el desempeño de los métodos de detección de objetos, en particular YOLO, el cual es el que alcanzó el mejor de todos, con los métodos basados en la CTC, en particular CNN-CTC. YOLO alcanzó un mayor *accuracy* que CNN-CTC, con una diferencia de 0.73 %, pero tiene una menor rapidez, siendo CNN-CTC 14 FPS más rápido. En parte esto se explicaría porque YOLO necesita de una etapa de postprocesamiento dada por la supresión no-máxima (NMS) la cual corre solo en CPU, mientras que con CNN-CTC el postprocesamiento viene dado por la decodificación de la salida de la red la cual tiene un costo considerablemente menor.

Cada enfoque tiene sus ventajas y desventajas. YOLO como detector de objetos tiene la ventaja de no solo predecir el número final, sino que también localizar cada dígito en la

imagen. Sin embargo, su desventaja es que es un poco más lento que CNN-CTC y además, para entrenarlo requiere las anotaciones de *ground truth*, o sea que, por cada imagen necesita las etiquetas de las clases y *bounding boxes* de cada dígito, lo cual puede ser problemático al ser un proceso más lento que solo etiquetar el número final. Por otra parte, CNN-CTC tiene como principal ventaja su rapidez la cual supera a la de todos los métodos probados, y además, como se mencionó anteriormente, el etiquetado de las imágenes es mucho más fácil y rápido, ya que solo necesita imágenes y sus correspondientes secuencias de *labels*, evitando tener que etiquetar cada dígito por separado.

Un análisis interesante es suponer que el enfoque secuencial dado por la CTC vendría siendo el óptimo por cuanto la *loss* penaliza explícitamente los fallos en la secuencia predicha, no así el enfoque basado en detección de objetos cuyas funciones de pérdida penalizan indirectamente dichos fallos a través de la detección de los dígitos. De este modo, se puede ver el desempeño alcanzado por la CTC como una referencia de desempeño para los detectores de objetos. Por lo tanto, se puede concluir que YOLO es un excelente detector de objetos al superar en 0.73 % al óptimo dado por la CTC, mientras que RetinaNet y FCOS se quedan atrás en un 2.61 % y 2.03 %, respectivamente.

Por último, la tabla 5.1 muestra una comparación de los métodos del estado del arte y las propuestas con mejor desempeño: YOLO y CNN-CTC. En ella se aprecia que ambos métodos propuestos superaron con amplio margen al estado del arte en los datasets CAR-A y CAR-B, mientras que en CVL solo son superados por BeiJing.

Conclusión

En este trabajo se investigó el impacto de los modelos de detección de objetos basados en redes convolucionales (CNN) para el problema de reconocimiento de secuencias de dígitos manuscritos en imágenes, en particular, montos en cheques bancarios reales (reconocimiento CAR).

Para ello se diseñaron arquitecturas basadas en CNN específicas para reconocimiento CAR. Las arquitecturas diseñadas consisten en adaptaciones de los métodos del estado del arte en detección de objetos, así como de modelamiento secuencial. Los algoritmo de detección de objetos utilizados son: YOLO, RetinaNet y FCOS. Mientras que los de modelamiento secuencial son: CNN-LSTM-CTC y CNN-CTC. Se programó una herramienta basada en tensorflow para entrenar y testear los modelos anteriores. Se creó un dataset el cual consta de datasets públicos: ORAND-CAR-A, ORAND-CAR-B y CVL, y datasets privados (propiedad de ORAND S.A.): Ban1, Ban2 y Ban3, donde estos últimos corresponden al campo CAR de cheques reales de 3 bancos distintos. Por último, se evaluaron los distintos modelos entrenados para reconocimiento CAR.

Los resultados obtenidos muestran una mejora significativa en comparación al estado del arte [27] en los datasets: CAR-A y CAR-B con un incremento de 6.95 % y 5.16 % en el *accuracy*, respectivamente, con el modelo basado en YOLO (ver tabla 5.1). Este resultado valida la hipótesis de que los algoritmos de detección de objetos basados en redes convolucionales, y en particular YOLO, son una alternativa competitiva con los actuales métodos de *deep learning* basados en redes recurrentes y superan considerablemente a los métodos tradicionales basados en segmentación (16 % de incremento).

Se compararon distintos métodos de los cuales el mejor en cuanto a desempeño fue YOLO con un promedio de 90.36 % de *acuracy* en los 6 datasets considerados. Mientras que el mejor en cuanto a rapidez corresponde a CNN-CTC con aproximadamente 118 FPS. También se analizaron los casos de falla de los distintos métodos comprobándose que muchos de ellos corresponden a casos difíciles incluso para una persona debido principalmente a la calidad de las imágenes.

En cuanto a la aplicación de este trabajo cabe señalar que debido a los buenos resultados alcanzados se pusieron en producción (para ORAND S.A.) algunos de los modelos entrenados (particularmente YOLO y CNN-CTC) en un entorno más eficiente basado en C++.

Como conclusión, en este trabajo se cumplieron todos los objetivos planteados al comienzo siendo el principal el de validar la hipótesis de que un enfoque basado en detección de

objetos es capaz de superar al estado del arte en el problema de reconocimiento CAR. Una consecuencia de este trabajo es que se demuestra que los algoritmos de detección de objetos son eficientes en reconocer dígitos lo cual debiese aplicar para los futuros algoritmos que se presenten.

Dado esto, es natural preguntarse qué sucede con LAR (reconocimiento del monto escrito en palabras). La respuesta no es tan directa, ya que si bien se podría extender la metodología de este trabajo de modo de aplicar un algoritmo de detección de objetos como YOLO para reconocimiento LAR, no es del todo claro cuales vendrían siendo los objetos a detectar. Una posibilidad es que sean las letras (*a,b,c,..*) y la otra es que sean las palabras (*uno, veinte, mil, cien*, etc.). Dado que las letras en una misma palabra están demasiado conectadas como para poder separarlas correctamente, sumado al hecho que que la traducción de palabras a números es mucho más directa que en el caso de las letras, de modo que resulta más factible la segunda opción, es decir, detectar solo las palabras. Ahora bien, un problema que se presenta sería el número de clases a reconocer (en el caso de dígitos era de 10) el cual corresponde al tamaño del diccionario utilizado para describir números. Además hay que sumar el hecho que, a diferencia de CAR en que los dígitos son completamente independientes entre sí, en el caso de LAR las palabras si tienen una cierta dependencia secuencial, por ejemplo, el hecho que exista *ciento doce* y no *doce ciento* indica que en este caso sí ayudaría considerar una memoria para los contextos pasados y/o futuros, de modo que un modelo CNN-LSTM-CTC puede que sea la mejor opción. Dada la cantidad de interrogantes aún sin responder las cuales requieren de una cuidadoso tratamiento, es preferible abordar el problema de reconocimiento LAR por medio de un algoritmo de detección de palabras como parte de un trabajo futuro, el cual incluye la creación de un dataset (con el cual aún no se cuenta) lo bastante grande para entrenar los modelos a utilizar.

Por último, en cuanto al cumplimiento de los objetivos de este trabajo: (i) se diseñaron arquitecturas basada en redes convolucionales (CNN) específicas para reconocimiento CAR, (ii) se programó una herramienta utilizando tensorflow la cual permita entrenar y probar los distintos modelos a utilizar, (iii) se construyó un dataset para entrenar modelos de detección de objetos aplicados a reconocimiento CAR, (iv) se evaluaron modelos de detección de objetos para reconocimiento CAR. De este modo, se valida que se cumplieron todos los objetivos planteados al inicio de este trabajo.

Bibliografía

[1] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12):3207–3220, 2010.

[2] Markus Diem, Stefan Fiel, Florian Kleber, Robert Sablatnig, Jose M Saavedra, David Contreras, Juan Manuel Barrios, and Luiz S Oliveira. Icfhr 2014 competition on handwritten digit string recognition in challenging datasets (hdsrc 2014). In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 779–784. IEEE, 2014.

[3] Abdeljalil Gattal, Youcef Chibani, and Bilal Hadjadji. Segmentation and recognition system for unknown-length handwritten digit strings. *Pattern Analysis and Applications*, 20(2):307–323, 2017.

[4] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[6] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[11] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[14] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[15] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[16] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[17] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[18] Raid Saabni. Recognizing handwritten single digits and digit strings using deep architecture of neural networks. In *2016 Third International Conference on Artificial Intelligence and Pattern Recognition (AIPR)*, pages 1–6. IEEE, 2016.

[19] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] C Soon-Man and O Il-Seok. A segmentation-free recognition of two touching numerals using neural network. In *Proc. of the 5th International Conference on Document Analysis and Recognition*, 1999.

[22] Ilya Sutskever, Geoffrey E Hinton, and A Krizhevsky. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012.

[23] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. *arXiv preprint arXiv:1904.01355*, 2019.

[24] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[25] Paul Viola, Michael Jones, et al. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1(511-518):3, 2001.

[26] Jiahui Yu, Yuning Jiang, Zhangyang Wang, Zhimin Cao, and Thomas Huang. Unitbox: An advanced object detection network. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 516–520. ACM, 2016.

[27] Hongjian Zhan, Qingqing Wang, and Yue Lu. Handwritten digit string recognition by combination of residual network and rnn-ctc. In *International conference on neural information processing*, pages 583–591. Springer, 2017.