Redes Convolucionales en Comprensión de Escenas

Trabajo Final por

Pablo D. Pusiol

Trabajo final realizado para la Facultad de Matemática, Astronomía y Física (Fa.M.A.F), de la Universidad Nacional de Córdoba, Argentina, en cumplimiento parcial para la obtención del grado de Licenciado en Ciencias de la Computación.

Trabajo dirigido por
Guido T. Pusiol, PhD.
Daniel E. Fridlender, PhD.

17 de febrero de 2014

Resumen

Presentamos y aplicamos una técnica de extracción no-supervisada de *features* de imágenes (Redes Neuronales Convolucionales) al problema de comprensión de escenas. Abarcamos las sutilezas tanto para el diseño como para el entrenamiento de estos modelos.

Mostramos el modelo, el entrenamiento y los resultados obtenidos para un problema específico (determinar posición discreta de jugadores de tenis), proveyendo también posibles estrategias para generalizar a otros problemas.

Keywords: *Computer Vision, Scene Undestanding, Deep Learning, Redes Neuronales, Redes Neuronales Convolucionales (ConvNets, CNN, Deep Neural Networks), Tecnología en el deporte, Extracción no-supervisada de features (UFL).*

De acuerdo con 1998 ACM Computing Classification System:

- I.5 PATTERN RECOGNITION
- I.5.1 Models
- I.5.2 Design Methodology
- I.5.4 Applications

Agradecimientos

Agradecer a mi familia, en especial a mis padres Nora y Daniel, y a mis hermanos Dante y Guido: sin su constante apoyo no estaría aqui.

Agradecer a mi directores por tomarse el tiempo para evacuar mis dudas, en especial a mi hermano Guido por sus consejos y por mantenerme motivado desde lejos.

A mis amigos por sus palabras de aliento y por interesarse en estas cosas raras y, por último, también quiero agradecer a todas las personas a las que molesté por correo, como Andrej Karpathy: su ayuda fue necesaria y esclarecedora.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. El problema de visión

La visión por computadora es un dominio abierto ya que abarca principalmente *problemas inversos*, en donde buscamos reconstruir incógnitas a partir de información insuficiente para especificar la solución. Parte del problema está relacionado con la forma en la que se representa una imágen en el disco. Como se mostrará más adelante, una imágen es una colección de valores, donde cada valor representa la intensidad captada por un sensor de un determinado punto espacial, representados como píxeles. La Figura 1.1 muestra como, lo que un humano ve claramente como una escena de un partido de tenis, la computadora lo ve como números.

La representación de los píxeles es muy sensible a cambios en la iluminación, ángulo, contraste y tamaño que pueda existir. Las dos imágenes de la figura 1.2 representan dos partidos del mismo torneo, sólo que en diferentes canchas y a diferente momento del día, afectando el ángulo, la luz/sombra, tamaño de la imágen. Otros desafíos se identifican con la **escala**, la **deformación del objeto**, la **oclusión del objeto**, **confusión con el fondo** y **variación entre clases** [47][46].

(a) Imágen original

139	149	151	127	80	
168	170	168	140	88	
126	164	174	133	87	
106	145	157	119	82	
112	148	155	117	89	1(
144	167	159	118	95	1:
169	176	155	112	94	1(
183	184	157	116	100	1(
187	189	163	121	103	1:

(b) Representación numérica de algunos píxeles de (a)

Figura 1.1: Representación de los píxeles

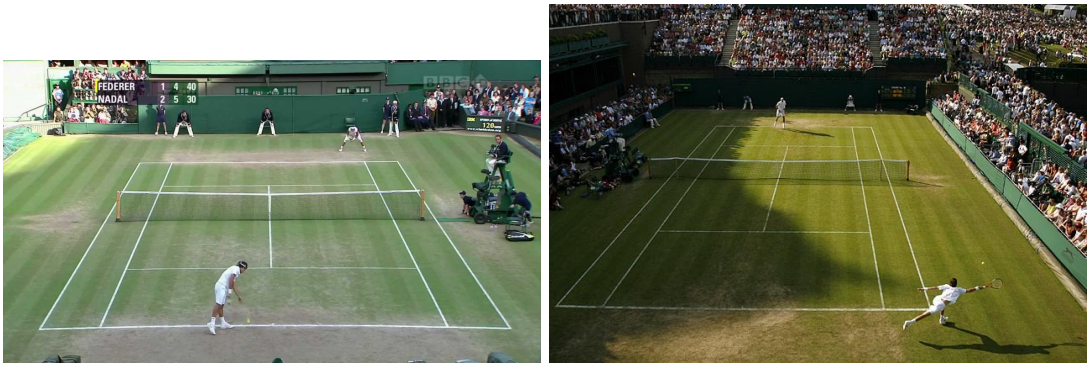


Figura 1.2: Desafíos de visión

1.1.1. El concepto de *"feature"*

Si tomamos el caso de la Figura 1.3, claramente ambas imágenes corresponden a personas jugando al tenis. El problema de visión consiste en hacer que la computadora interprete los píxeles de ambas imágenes de la figura 1.3 como jugadores de tenis, a pesar de que las intensidades representadas por los píxeles son diferentes en ambas. **Necesitamos de alguna función compleja que comprenda el significado de la imágen más allá de la representación numérica de la misma**.

Figura 1.3: Comparando píxel a píxel

Es aquí donde entran en juego las features: representan/describen características de los objetos de interés. Existen features de diferentes niveles de abstracción: desde bordes, características de color, ángulos, hasta capaces de representar el concepto de una persona o de una raqueta de tenis.

Tomemos por ejemplo el problema de reconocer un tenista en una imágen (Figura 1.4). Comparando píxel a píxel, obtendríamos una disposición sin una estructura que permita realizar una clasificación donde los "+" representan los casos donde hay tenistas en la imágen y los "-" las imágenes que no tienen tenistas (Cuadro 1 de la Figura 1.4). En cambio, si en lugar de comparar píxeles, utilizamos el concepto de **feature** y comparamos features como la presencia de

una raqueta y de una persona, obtendríamos resultados similares a los que muestra el Cuadro 2 de la misma figura. Estos resultados permiten encontrar una función que clasifique ambas categorías, pudiendo así tomar decisiones sobre imágenes no conocidas.

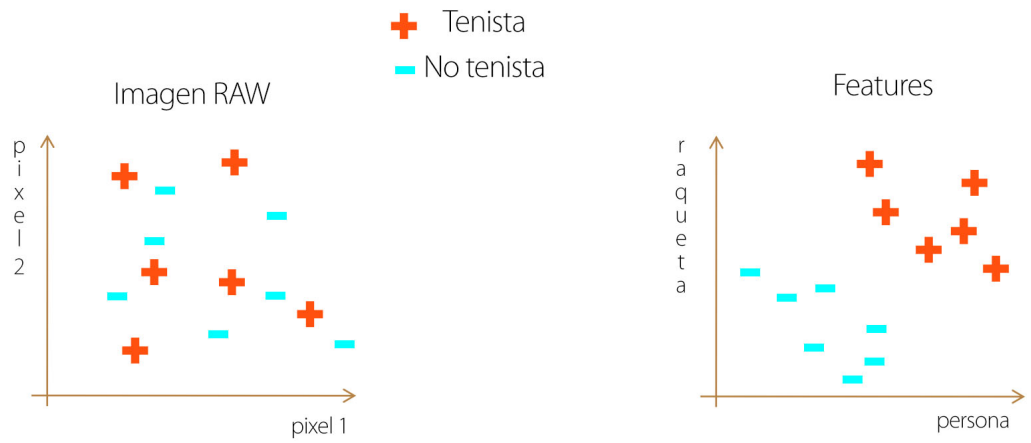


Figura 1.4: Comparando features vs. píxeles

Tenemos ahora en teoría un concepto que nos abstrae de lidiar píxel a píxel, sin embargo el concepto de un descriptor de features no es original para una pc por lo que hay que proveer las herramientas para que la pc pueda encontrarlas.

A lo largo del desarrollo de la visión por computadora se distinguen dos enfoques para la generación de descriptores capaces de reconocer features: El enfoque basado en técnicas superficiales (o "shallow" en inglés) y el enfoque de aprendizaje no-supervisado (UFL[1]).

1.2. Enfoque Superficial

El enfoque superficial responde a la manera tradicional de hacer visión por computadora. Se busca generar una representación invariante a la posición, iluminación, fondo, etc. utilizando principalmente técnicas estadísticas a partir de conocimiento anterior del objeto y del contexto. Los descriptores superficiales son diseñados a mano a partir de características físicas, hipótesis y observaciónes.

El enfoque entonces no aprende features de los objetos sino que se procesa la imágen de interés con extractor de **features prefijadas** y luego se pasa la nueva representación a un clasificador entrenable. El proceso de clasificación es independiente al proceso de extracción de features de la imágen. (Figura 1.4)

En el apéndice A, sección A.4 se profundiza sobre dos descriptores comunmente usados.

[1]UFL (del Inglés): Unsupervised Feature Learning.

Figura 1.5: Estructura de técnicas del Enfoque Superficial (Shallow)

1.3. Enfoque No-Supervisado

La filosofía de estas técnicas se basa en la hipótesis de "un único algoritmo de aprendizaje": El cerebro usa esencialmente el mismo algoritmo para comprender diferentes formas de entrada [53]. Es entonces que en este enfoque se contempla que la complejidad de la percepción no está en el algoritmo ni en las estructuras de aprendizaje, sino en los datos.

El enfoque no supervisado consiste de técnicas, algoritmos y estructuras inspiradas en el cerebro que son capaces de aprender features por sí mismas, en lugar de que sean diseñadas a mano y prefijadas en el detector. El problema asociado a esto es que no sabemos qué features el modelo aprende durante el entrenamiento.

Como muestra la figura 1.6, el extractor de features pasa a ser parte del componente entrenable del detector, haciendo que el módulo de clasificación deje de ser independiente al resto de la estructura.

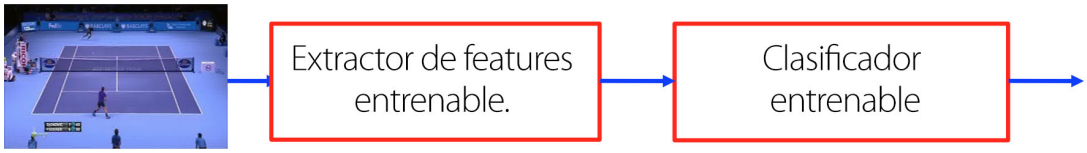


Figura 1.6: Estructura de técnicas del Enfoque No-Supervisado (Deep)

Las estructuras de aprendizaje no-supervisado de features contemplan el concepto heredado de la neurobiología de que las features son jerárquicas: desde features de bajo nivel, hasta estructuras y objetos. Las Redes Neuronales Convolucionales soportan el aprendizaje de features [10, 30] y es el modelo que analizaremos (Ver Figura 1.7).

1.4. Organización del trabajo

En el capítulo siguiente presentamos las redes neuronales convolucionales: tratamos el concepto de la aplicación en visión, la evolución del modelo, arquitectura general, casos de uso exitosos que inspiraron el desarrollo de este trabajo y salvedades generales del entrenamiento del modelo. En el capítulo 3 presentamos nuestro caso de estudio y proponemos un modelo simple de red para la comprensión de escenas, mostrando los resultados obtenidos para la solución del problema planteado. El capítulo 4 concluye el trabajo y seguido a este se encuentran los apéndices que proveen el conocimiento teórico para comprender en más profundidad lo desarrollado.

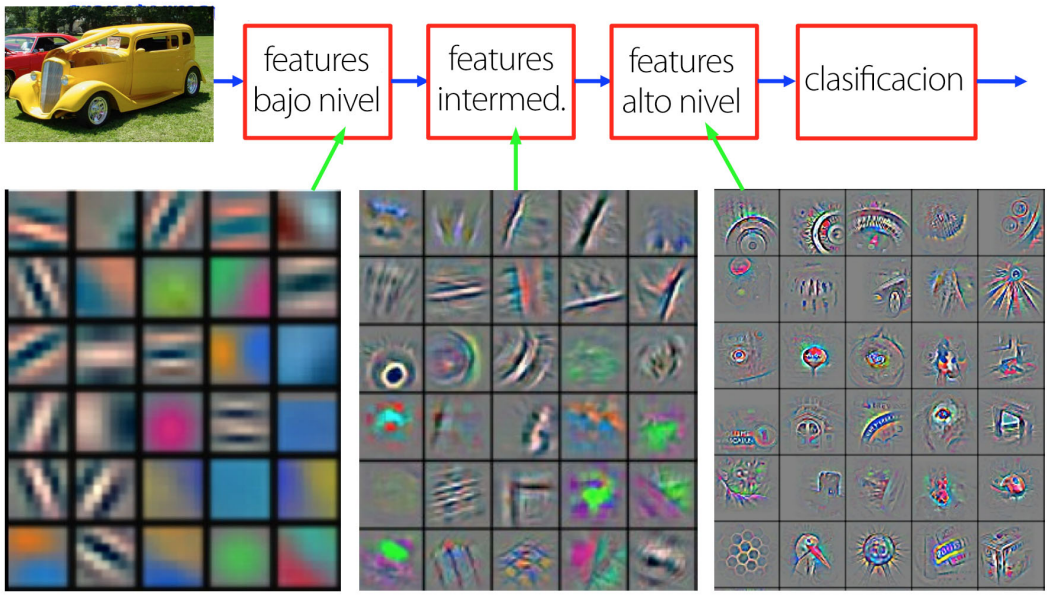


Figura 1.7: Visualización de features de una Red Neuronal Convolucional, Zeiler & Fergus - 2013

Capítulo 2

Redes Neuronales Convolucionales

A pesar de tratarse de un aproximador universal en teoría, las redes FFNN[1] no fueron lo suficientemente buenas al enfrentarse a problemas prácticos, como el reconocer objetos presentados visualmente. En una FFNN todas las neuronas de la capa j están conectadas con todas las de la capa $j+1$, haciendo crecer rápidamente el número de parámetros, lo que se traduce en una solución poco práctica para el dominio de visión. Más aún, dado que cada par de neuronas entre dos capas tienen su propio peso, aprender a reconocer una feature en una ubicación no se translada a la misma feature en otra parte de la imágen.

Las redes convolucionales (o CNN, ConvNets) son un tipo de red neuronal de la familia de **deep learning** específicamente diseñadas para aprender y descubrir características (features) de imágenes [10, 30, 32, 36, 40, 42, 46, 54]. Estan inspiradas en las redes neuronales tradicionales, incorporando operaciones no-lineales de manipulación de imágenes.

2.1. Evolución del arte

Las ConvNets fueron introducidas por Kunihiko Fukushima en 1980 [20][21]. Este modelo, en ese momento bautizado "Neocognitron", se inspira en los descubrimientos en percepción visual de Hubel y Wiesel (1962): los neurocientíficos encontraron dos tipos de células en la corteza primaria de visión a las que llamaron células S y células C, y propusieron un modelo de cascada de relación de estas.

El Neocognitron es una extensión de estos modelos: Consiste en muchas células interconectadas siguiendo una estructura de cascada en donde las features locales son extraídas por células S y las deformaciones son toleradas por células C. Las features locales de la entrada son integradas gradualmente y clasificadas en capas superiores de la cascada.

Recién en 1998 las CNN's volvieron tomar impulso cuando Yann LeCun, con el algoritmo de *Backpropagation*[2], pudo entrenar un modelo similar para el reconocimiento de dígitos en

[1] FFNN = Feed Forward Neural Network. Ver apéndices: B.3.1.
[2] Backpropagation = Retropropagación. Ver apéndice B.

documentos de texto (es con este modelo que nacen las redes neuronales modernas que aplicaremos) de manera totalmente supervisada [37][38][35], donde cada una de estas capas usando el algoritmo dicho, aprende features de las imágenes. El enfoque de Yann LeCun tomo tanta importancia que al cabo de finales de los 90's el 10 % de los cheques circulando en los Estados Unidos eran chequeados por ConvNets. Desde esa época hasta el 2012 se desarrollaron aplicaciones usando CNN's pero no fue hasta el 2012, cuando Geoff Hinton propuso un modelo para clasificar imágenes en 1000 categorías, que se produjo la revolución en el campo de visión computacional.

2.1.1. Aplicaciones exitosas de los últimos años

En los últimos 3 años se han aplicado ConvNets para resolver diferentes problemas de visión, cuyos resultados superan a otras técnicas del estado del arte ganando numerosos concursos (ImageNet, Kaggle para expresiones faciales, aprendizaje multimodal, clasificación de gatos/perros[3], Señales de tránsito alemanas, entre otros). Veamos algunos de estos casos:

Reconocimiento de escritura

Este problema, como ya hemos dicho fue uno de los precursores del uso de redes convolucionales mostrando con Yann LeCun y su LeNet-5 que estas estructuras son útiles para visión. LeNet-5 obtuvo un 99,3 % de efectividad en 1998 [37] sobre el dataset MNIST [39] usando $tanh(x)$ como el componente no lineal y avg-pooling para el submuestreo. Se ha perfeccionado aquella primera red convolucional en una serie de oportunidades, alcanzando un 99,77 % en 2012 superando cualquier otro resultado [7].

MNIST no es el único dataset sobre el cual se han hecho avances en este problema: Ahranjany et al. en [1] alcanzan un 99,17 % de efectividad en caracteres Farsi/Arábicos usando una CNN entrenada con SGD.

OCR en numeración de casas

Para el reconocimiento de los números de las casas, en [19] (2009) y más recientemente en [23] (2014) se han alcanzado resultados del 94,3 % usando redes convolucionales con varias etapas a partir de imágenes de Google Street View.

Señales de tránsito

En [61] se pueden encontrar los resultados para la clasificación de señales de tránsito del concurso GTSRB[27] (The German Traffic Sign Recognition Benchmark). Sermanet en el 2011 [58] logró un 98,31 % de respuestas correctas usando una CNN, próximo al 98,84 % del rendimiento humano. Un año después, Dan Ciresan logra superar el rendimiento humano con un 99,46 % de efectividad [7] también usando CNN's para decidir en más de 40 clases, ambas CNN's dejaron atrás el 96,14 % de las técnicas shallow tradicionales.

[3] Ver http://www.kdnuggets.com/2014/02/deep-learning-wins-dogs-vs-cats-competition.html. Consultado en Febrero 2014 - CNN de Pierre Sermanet.

Detección de peatones

La detección de personas es un problema fundamental en visión. A lo largo de los años se han incorporado nuevas técnicas, nuevos modelos para resolverlo. En el año 2013, Sermanet et al. [57] propusieron un modelo de ConvNets para resolver el problema de forma no supervisada, obteniendo resultados sobresalientes en diferentes datasets (INRIA[8], CALTECH[12], entre otros).

En trabajo previo, Pusiol P. y Pusiol G. [50] (2014) han entrenado un modelo a partir de una ConvNet sobre imágenes del dataset INRIA obteniendo un error del 0,028 % sobre los datos de validación del mismo dataset[4]. El entrenamiento fue totalmente supervisado.

ImageNet

Imagenet es un dataset compuesto de más de 100.000 categorías con cerca de 1.2TB de imágenes, desarrollado por el laboratorio de visión de la Universidad de Stanford[5][11]. Imagenet organiza una competición en la que se desafía a clasificar imágenes de un subconjunto de categorías de ImageNet (ILSVRC) [3][16]. Geoffrey Hinton participó de este concurso con un modelo de ConvNet [34] (SuperVision), bajando el error de un ±25 % a un ±15 % en el desafío usando todas las técnicas aprendidas para el diseño y entrenamiento de CNN's hasta la fecha [16]. SuperVision se trata de una red con 650K neuronas, 832M de sinapsis y 60M de parámetros. Estos resultados significaron una revolución en visión por computadora, haciendo de técnica elemental las redes neuronales para atacar gran parte de los problemas del área. El entrenamiento se realizó durante 1 semana y fue realizado sobre 2 GPU's [2][6]. (También se reportan resultados en otros datasets como Caltech [24]).

Actualmente, Andrej Karpathy quien coolaboró en este trabajo y es miembro del laboratorio de visión de Stanford[6], se encuentra entrenando un modelo capaz de reconocer las 100.000 clases de ImageNet.

2.2. Arquitectura típica

Las ConvNets son estructuras entrenables compuestas de varias etapas, aprendiendo en cada una features de diferente abstracción. La entrada y salida de cada una de estas etapas son conjuntos de arreglos llamados *"mapas de features"*. A la salida, cada mapa de features representa una feature particular extraída de todas las ubicaciones de la entrada.

Cada etapa de una ConvNet está compuesta por las siguientes tres capas: una capa de **convolución**[7], una capa **no lineal** y una capa de **sub-muestreo**.

Una típica ConvNet para clasificación supervisada está contruida a partir de una o varias de estas etapas seguidas de un módulo de clasificación; por ejemplo, la red de Yann LeCun para resolver el problema de los dígitos manuscritos (LeNet), cuenta con dos etapas como las descritas.

[4] Ver más en http://activityrecognition.com
[5] http://www.image-net.org/
[6] http://vision.stanford.edu/people.html
[7] Ver apéndice A.1.2.

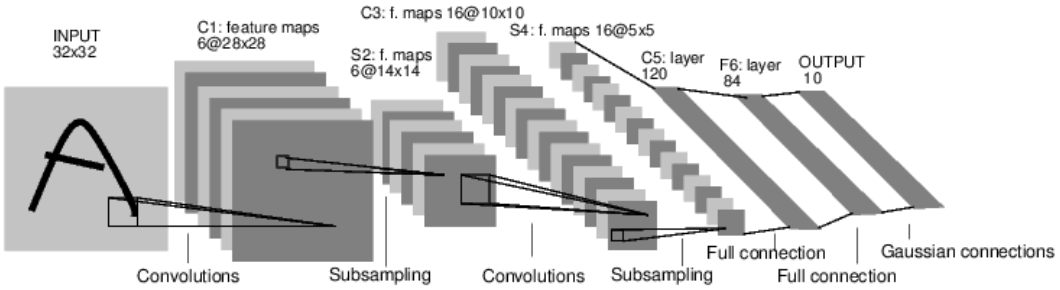


Figura 2.1: Arquitectura de LeNet-5

Sigamos paso a paso el funcionamiento de LeNet-5 (Fig. 3.2): toma como entrada píxeles. A la entrada de cada etapa hay una secuencia de mapas de features (capa de convolución) seguido por una capa de pooling (submuestreo). Entonces, en la capa C1 de 3.2, cada uno de los 6 mapas de features contiene pequeñas features con sus pesos agrupados. A continuación de esta capa está la S2 (pooling) que agrupa las salidas de una serie de features replicadas vecinas en C1, dando como resultado un mapa más pequeño que servirá de entrada a la siguiente etapa dedicada a encontrar features replicadas de mayor abstracción. A medida que se avanza en esta jerarquía de etapas, se aprenden features más complicadas pero más invariantes a posición (por el submuestreo).

Las capas enteramente conectadas se encargan de evaluar las posibles combinaciones de las features aprendidas para salir de la red con el grado de confianza final de la imágen para cada clase.

2.3. Entrenamiento

Las redes neuronales convolucionales están basadas en la idea de que las features se replican: como un objeto se mueve y aparece en diferentes píxeles, un detector de una feature que funciona en una parte de la imágen, es probable que también vaya a funcionar en otra parte de la imágen. La idea es construir varias copias del mismo detector de features en diferentes posiciones y para una misma posición tener muchos detectores de features, de manera que cada parte de la imágen pueda ser representada con features de diferentes tipos.

En la imágen 2.2 se pueden ver tres detectores de features que son réplicas. Cada uno de ellos tiene pesos a 9 píxeles, y esos pesos son idénticos en los tres diferentes detectores: tanto la flecha roja como la verde y la azul tienen el mismo peso para los tres detectores de la figura. Replicar a través de posición ayuda a reducir el número de parámetros a aprender: para los 27 píxeles que muestra la figura sólo hay 9 parámetros diferentes.

El algoritmo de entrenamiento consiste en resolver un problema de optimización, donde se buscan los parámetros que hacen que la función de costo sea mínima: al tratarse de una función multidimensional y no-convexa, existen puntos de silla y mínimos locales que hay que evitar.

Adaptar el concepto de features replicadas a backpropagation no es complicado: se computan los gradientes de la forma usual (SGD) y luego se modifican para que satisfagan las restricciones lineales entre los parámetros, de manera que si los pesos satisfacen las restricciones, tras

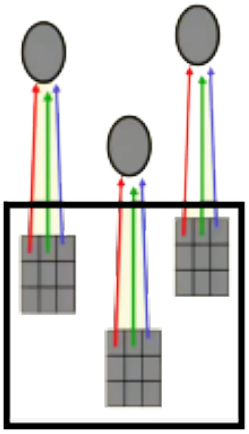


Figura 2.2: Replicación de features.

actualizarlos lo sigan haciendo. Supongamos que queremos restringir los pesos w_1 y w_2 tal que $w_1 = w_2$. Necesitamos que $\Delta w_1 = \Delta w_2$. Usando $\frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2}$ para actualizar w_1 y w_2 forzamos al algoritmo de backpropagation a que aprenda para detectores de features replicadas.

El uso de detectores de features replicadas provee **invarianza en el conocimiento** de la red y **equivarianza en las actividades** de las neuronas: Si detectamos una feature en un lugar de la imágen la podremos detectar en otro lugar, pero las neuronas que se activan no son las mismas sino que son equivalentes a las activaciones originales como muestra la Figura 2.3.

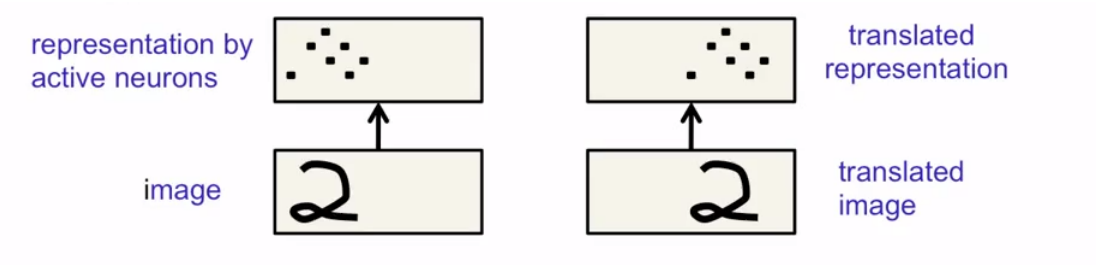


Figura 2.3: Equivarianza de activaciones de neuronas tras transladar feature de imágen.

Si bien tanto la estructura como el algoritmo de entrenamiento de las ConvNets son sencillos, entrenar un modelo no es algo simple. La no-linearidad de las funciones involucradas es explícita, haciendo que la función de costo de cualquiera de estos modelos sea no convexa: mientras más profunda es la red (i.e., más etapas tiene), más compleja va a ser la función de costo.

Capítulo 3

Comprensión de escenas

En este capítulo aplicaremos las ConvNets a un caso de comprensión de escenas; en particular buscaremos extraer semántica de imágenes de transmisiones de partidos de tenis.

3.1. Descripción del caso de estudio

Consideramos que un jugador puede estar en 3 diferentes posiciones de la cancha: A la derecha de la línea de base, a la izquierda de la línea de base o en la red. Proponemos clasificar las posibles posiciones del jugador como muestra la Figura 3.1, teniendo el área azul para la derecha, el área roja para la izquierda y finalmente el área amarillo para la red. Las zonas no cubiertas por esta clasificación, son consideradas ambiguas y pueden ser clasificadas en cualquiera de las áreas vecinas; como por ejemplo, el espacio comprendido entre los rectángulos azul y rojo puede entenderse como posición derecha y posición izquierda indistintamente.

En este caso de estudio sólo analizaremos posiciones discretas, y sólo ubicaremos al jugador de la parte inferior de la cancha (el movimiento del jugador en la parte superior es ruido). Cabe destacar que trabajamos con imágenes de partidos 1 hombre contra 1 hombre.

Si bien el problema ya ha sido estudiado en [22] y [9], los enfoques que usan para abordarlo son superficiales: dependen fuertemente de descriptores para delimitar la cancha, descriptores para detectar los jugadores, hipótesis basadas en color, entre otros.

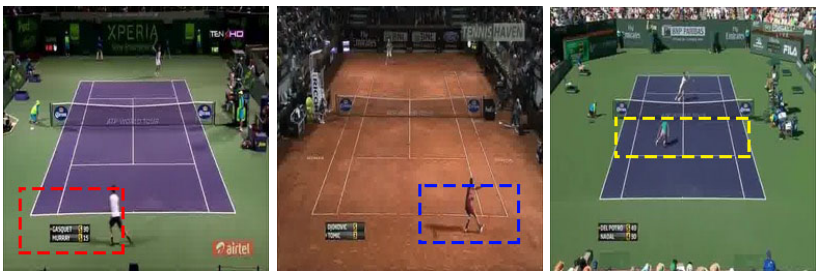


Figura 3.1: Clasificación en las zonas de la cancha

3.2. Descripción nuestra CNN

Para implementar una CNN capaz de extraer semántica de escenas, basamos nuestra arquitectura en la propuesta por Geoffrey Hinton para resolver la clasificación de 1000 objetos de Imagenet [34].

Al tratarse de clasificar sólo en 4 clases en lugar de 1000, proponemos un modelo simple, totalmente supervisado, donde todos los pesos de son inicializados con ruido gaussiano. Este modelo cuenta con una etapa de 3 capas: 1 de convolución, 1 de sub-sampling y 1 para la clasificación (Fig. 3.2), contando también con capas intermedias para normalización:

- C1 (Convolución)
- S1 (Submuestreo)

  N1 (Normalización)
- FC (Clasificador)

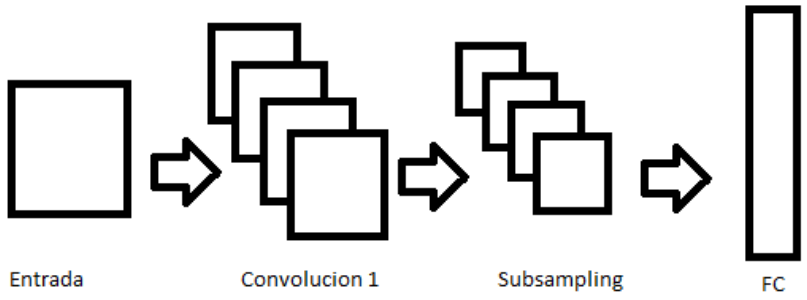


Figura 3.2: Arquitectura de CNN para posición de jugadores

3.3. Características de las capas

3.3.1. Capa 1 : C1

A la red ingresa una imágen cuadrada de 227x227 y 3 canales (RGB), previa sustracción de la media del dataset. La primer capa C1 convoluciona cada una de estas imágenes con 96 núcleos RGB, como propuesto en [34] para resolver Imagenet (Fig. 3.3). Cada kernel es de 11x11. En esta capa se aprendieron features de bajo nivel: bordes, cambios de dirección, cambios de color [31]. La salida consiste en 96x55x55 valores, ya que se necesitan 55 aplicaciones de cada núcleo para cubrir el alto y ancho de la imágen de entrada: En cada aplicación (horizontal y vertical) hay un salto de 4 píxeles. Las salidas de las neuronas se conectan con la capa siguiente usando no-linealidades ReLU's, las que, como se dijo anteriormente, son inicializadas con ruido blanco Gaussiano.

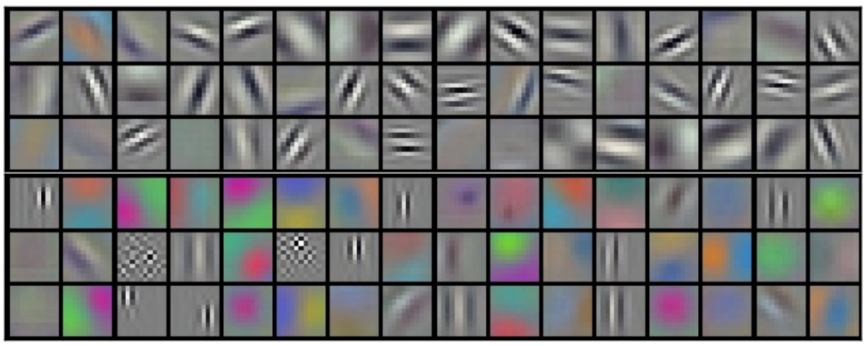


Figura 3.3: 96 filtros de bajo nivel aprendidos en la primer capa de convolución

3.3.2. Capa 2 : S1

Para llevar a cabo el submuestreo, usamos una operación llamada **pooling**, que consiste en tomar el valor máximo (**max-pooling**), mínimo (**min-pooling**) o promedio (**avg-pooling**) dentro de una vecindad, delimitada por un núcleo o tamaño de ventana [4][14][15].

La operación de pooling se usa en visión con dos finalidades: (1) Reduce los costos computacionales de las capas siguientes (C2), y (2) Provee robustez frente a una traslación o desplazamiento: Si consideramos que un sólo píxel puede transladar la imágen original en 8 direcciones, Alrededor de 5 de esas posibles configuraciones producirán el mismo valor en la capa siguiente (Suponiendo un núcleo $k = 3$).

Las capas de pooling en las CNN's sintetizan las salidas de una vecindad de neuronas en el mismo núcleo. Típicamente estas vecindades no se superponen ([28][33][41]). Con el objetivo de disminuir el error y reducir el overfitting, en [34] se propone usar un tamaño de núcleo mayor al salto horizontal y vertical, haciendo que las ventanas de píxeles se superpongan.

En esta capa decidimos usar la operación max-pooling (3.1) con un tamaño de ventana o núcleo $k = 3$ y un paso $s = 2$ (overlapping) siguiendo la estrategia de [34] y contrastamos los resultados con $k = 2 = s$ (no overlapping). De esta manera, a las 96x55x55 valores de la capa C1, se las submuestrea en 96x27x27. El caso de ventanas superpuestas no sólo ayudó a evitar el sobreajuste, sino que también nos dio un rendimiento $\pm 30\,\%$ superior para el mismo tiempo de entrenamiento (Ver Diagrama 3.9: el 2do valor corresponde al porcentaje de predicciones correctas sin overlapping, mientras que el 3ro corresponde al mismo valor pero en un modelo con overlapping).

$$h_{i+1,x,y} = \max_{(j,k)\in N(x,y)} h(i,j,k) \tag{3.1}$$

Subcapa N1

Las neuronas ReLU usadas en la capa de pooling no necesitan normalizar su entrada para evitar la saturación, sólo con que algunos ejemplos de entrenamiento generen una entrada positiva a una de estas neuronas, va a aprender.

Sin embargo, en [34] alcanzan una mejor generalización llevando a cabo una normalización de brillo similar a la normalización de contraste local [28]. Nosotros diseñamos nuestra red con

la siguiente:

Llamemos $a^i_{x,y}$ a la actividad de la neurona computada al aplicar el kernel i en la posición (x, y), obteniendo la respuesta normalizada $b^i_{x,y}$ dada por la siguiente expresión:

$$b^i_{x,y} = \frac{a^i_{x,y}}{(k + \alpha \sum_{j=\text{máx}(0,i-n/2)}^{\text{mín}(N-1,i+n/2)} (a^j_{x,y})^2)^\beta} \tag{3.2}$$

donde la sumatoria recorre n núcleos en la misma posición espacial, y N es el número total de núcleos en la capa. Las constantes toman los siguientes valores $k = 2$, $n = 5$, $\alpha = 10^{-4}$ y $\beta = 0{,}75$.

Aplicando esta normalización no vimos una mejora sustancial sobre el error, lo que creemos que se debe al tamaño de la red ya que sólo impacta en los pesos de una capa al momento de entrenar [64].

3.3.3. Capa 3 : FC

Esta es la última capa que realiza la clasificación de las features para 4 clases mutuamente excluyentes. Para cada neurona x_i calcula la siguiente función:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{4} e^{x_j}} \tag{3.3}$$

La función se llama Softmax y permite que para cada posible clase (o salida del softmax) tengamos una probabilidad de que la entrada a la red neuronal pertenezca a cada clase, donde la suma de las probabilidades es 1.

3.4. Entrenamiento

Aumento del tamaño del dataset

Las CNN funcionan mejor a medida que más datos tienen. Una forma de aumentar el tamaño del dataset es tomar subimágenes del dataset [6].

Las imágenes del dataset TEPOS (Ver 3.5.1) tienen una resolución mayor a la resolución admitida por la red. El motivo de esta diferencia en tamaño es que, para aumentar el tamaño del dataset tomamos ventanas de 227x227x3 del ejemplo de entrenamiento. Aplicar estas ventanas significa multiplicar el tamaño del dataset sin afectar la semántica. Las ventanas que tomamos se muestran en 3.4. En [33] y [34] usan la misma técnica sólo que la posición de las ventanas son aleatorias dentro de la imágen. También proponen tomar las proyecciones horizontales de la imágen, técnica que no pudimos aplicar ya que hacerlo significa romper la clasificación (por ejemplo, la categoría izquierda tendría la imágen original y la proyectada *-que debería estar en derecha-*).

Uso de no-linearidad ReLU

Tradicionalmente en redes neuronales, para modelar la salida de una neurona f, se lo hace como una función de su entrada x, con $f(x) = tanh(x)$ o $f(x) = \frac{1}{1+e^{-x}}$ (función sigmoidea).

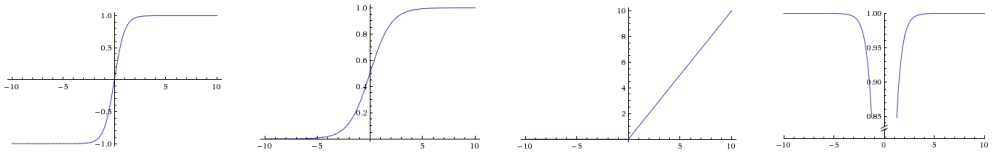
Figura 3.4: Diferentes ventanas del dataset

Usar estas funciones traen dos problemas: (1) Suelen saturar y (2) Son muy lentas para entrenar. Nair y Hinton bautizaron como ReLU a la neurona que computa la función 3.4, probando un mejor rendimiento en CNN's y RBM's [1][45].

$$f(x) = \text{máx}(0, x) \tag{3.4}$$

donde $x = \sum w_i f(z_i)$ es la entrada total a la neurona, y $f(x)$ su salida.

Jarret et al. en [28] proponen otra no-linearidad $f(x) = |tanh(x)|$, sin embargo el uso de esta requiere una normalización diferente a la implementada.



(a) $f(x) = tanh(x)$ (b) $(f(x) = 1 + e^{-x})^{-1}$ (c) $f(x) = \text{máx}(0, x)$ (d) $f(x) = |tanh(x)|$

Figura 3.5: No-linealidades contempladas

Descenso de gradiente estocástico

El problema de aprendizaje de una red neuronal es un problema de optimización: consiste en obtener los parámetros (o pesos) que hacen que la función de costo converja a un mínimo.

El algoritmo básico para entrenar consiste en aplicar **SGD**[2] usando la regla de la cadena (**Backpropagation**) (LeCun [35] y [38]). En CNN's también se entrena con **Batch Learning** (Aprendizaje de a lotes), **Momentum** (aceleración) y **Weight Decay** (decaimiento de los pesos) [44][37][34][33][56]. A continuación explicaremos brevemente como se incorporan cada una de estas estrategias:

Batch Learning: Batch Learning consiste en reducir la cantidad de actualizaciones de pesos. En lugar de actualizar luego de cada propagación, se realizan n propagaciones y luego se modifican los pesos en el modelo. Cabe destacar que, por más que Batch Learning mantenga

[1] RBM = Restricted Boltzmann Machines.
[2] SGD: Stochastic Gradient Descent, inglés para Descenso de Gradiente Estocástico.

pesos constantes dentro de las n propagaciones del ciclo, converge de manera más estable (pero más lenta) al mismo mínimo que si fuese actualizando en tiempo real.

Cada n propagaciones, con batch learning aplicamos la siguiente actualización a un peso w:

$$w := w - \epsilon\Big(\lambda w + \frac{1}{n}\sum_{i=1}^{n}\frac{\partial L}{\partial w}(z_i, w)\Big) \tag{3.5}$$

donde ϵ es la taza de aprendizaje, λw es el término de regularización, y $\frac{\partial L}{\partial w}(z_i, w)$ es la derivada parcial del objetivo en el ejemplo z_i.

Momentum: El momentum (o acelereación) simplemente suma una fracción M de los pesos anteriores al cálculo del nuevo. En [34] sugieren utilizar momentum para prevenir que el entrenamiento converja a un mínimo local o a un punto de Monutra. Un momentum alto (como el propuesto) también ayuda a la velocidad de convergencia, sin embargo esto aumenta el riesgo de exceder al mínimo.

Weight Decay: El Weight Decay (o decaimiento de los pesos), adhiere un término de multa a la función de error. Corresponde a la familia de métodos de regularización; sin embargo para imágenes, Hinton en [34] y en nuestros resultados comprobamos que también reduce el error de entrenamiento del modelo(El primer valor del diagrama 3.9 se corresponde al entrenamiento del modelo sin Weight Decay, mientras que en el 2do y el 3ro se entrena con weight decay).

Configuración final: Para una iteración $i+1$, la actualización del peso w usando Momentum, Weight Decay y Batch Learning, queda configurada como:

$$\begin{aligned} v_{i+1} &:= M \cdot v_i - D \cdot \epsilon \cdot w_i - \epsilon \cdot \Big\langle \tfrac{\partial L}{\partial w}|_{w_i} \Big\rangle_{D_i} \\ w_{i+1} &:= w_i + v_{i+1} \end{aligned} \tag{3.6}$$

donde M es el momentum (tasa constante), v es la variable de momentum, ϵ es la tasa de aprendizaje, D es la constante de Weight Decay, y $\Big\langle \frac{\partial L}{\partial w}|_{w_i} \Big\rangle_{D_i}$ es el promedio sobre el *i-ésimo* lote D_i de la derivada del objetivo con respecto al peso w, evaluado en la *i-ésima* actualización del peso, w_i.

3.5. Resultados

3.5.1. Descripción de los datos

Decidimos trabajar con imágenes generadas a partir de transmisiones deportivas de diferentes canales de televisión.

La resolución de las imágenes originales es de **640x360 píxeles**, siendo la solución escalable a imágenes de alta resolución. Normalmente los sensores capturan con un *frame rate* de 24fps.

Dataset

Tuvimos que diseñar un dataset específico para resolver el problema de la posición usando datos como los descritos en el párrafo anterior. Para esto recurrimos a 2hs de filmaciones, extrayendo entre 10 y 15 cuadros por segundo.

A partir de esas dos horas clasificamos ±80000 imágenes en las categorías izquierda, derecha y red, más una cuarta indicando las imágenes que no corresponden con ninguna de las 3 originales y son propias de las transmisiones deportivas como ser un acercamiento, una repetición, primer plano del público entre otras. Este dataset recibe el nombre de TEPOS y las clases son: *derecha*, *izquierda*, *red* y *no juegan*. El mismo dataset tiene 1600 imágenes de validación. Cabe destacar que entrenamos sobre una submuestra de ±6000 imágenes seleccionadas aleatoriamente de TEPOS.

Si bien las imágenes originales son de 640x360x3, son llevadas en TEPOS a un tamaño de 256x256x3. De esta manera reducimos considerablemente la cantidad de parámetros (tamaño de la red), haciendo que decaiga también el tiempo en que se toma una decisión sin afectar al rendimiento.

3.5.2. Entrenamiento y resultados

Entrenamos nuestro modelo con los siguientes parámetros:

- **Tasa de aprendizaje inicial:** 0.0001
- **Momentum:** 0.9
- **Weight Decay:** 0.0005

Inicializamos los pesos en cada capa usando una distribución Gaussiana de media $\mu = 0$, con desviación estándar $\sigma = 0{,}01$ (igual que [34]). Inicializamos la tasa de aprendizaje en 0.0001 para todas las neuronas, haciendo que descienda en 1/10 cada 5000 iteraciones.

Siguiendo el modelo descrito, los resultados alcanzan una efectividad del 76,8 % sobre el conjunto de validación (±1600 imágenes) tras 20.000 iteraciones (tercer valor del diagrama 3.9). El entrenamiento de esta configuración tomó alrededor de 2 días en GPU iterando cerca de 100.000 veces.

En la Figura 3.7 podemos ver cómo va mejorando el error mientras más se entrena el modelo. Cada 1000 iteraciones ($\pm\frac{1}{6}$ epoch) se valida el entrenamiento, obteniendo el mejor resultado en la iteración 21000, lo se corresponde con el mínimo de la función de costo sobre la cual se entrena (Figura 3.8).

Al no existir otros modelos con diferentes técnicas sobre los mismos datos no podemos efectuar comparaciones de los resultados, sin embargo sabemos que la efectividad del uso de CNN's puede ser mejorada sustancialmente. Discutiremos algunas formas de lograr mejores resultados a continuación.

3.5.3. Posibles mejoras de rendimiento

Pre-procesar los datos Tratar al jugador de la parte superior como ruido introduce sin duda error al modelo, ya que la red está aprendiendo features de ese jugador mientras el dataset lo ignora. Para corregir esto, proponemos cortar la imágen a la altura de la red y entrenar con esas imágenes. De esta manera también aumentamos la cantidad de píxeles que varían, haciendo que más features los detecten.

Aumentar el tamaño del dataset Si bien para modelos de visión tradicionales el dataset que construimos podría ser considerado lo suficientemente grande, para la extracción no atendida de features, mientras más datos tenga el modelo mejores features va a descubrir, haciendo decrecer el error, Imagenet por ejemplo consta de 1.2TB en imágenes. Otras aplicaciones de otras áreas como audio, son entrenadas en $\gg 5000hs$ de grabación [26] , y aplicaciones basadas en texto han sido entrenadas "leyendo" Wikipedia [62].

Pre-entrenamiento de features Hay suficiente evidencia de que, para datasets pequeños se puede direccionar la optimización del modelo haciendo que el entrenamiento de la red comience con pesos próximos a los esperables. Esto técnica la han usado satisfactoriamente en el dataset MNIST para superar el rendimiento de LeNet-5.

Complejizar el modelo Agregar capas de convolución/pooling al modelo hará que aprenda más representaciones de nivel medio y alto logrando una mejor generalización. Sin embargo, para lograr esto también es necesario aumentar significativamente el volumen de datos sobre el cual se entrena, tanto en clases como en cantidad de imágenes.

Durante el desarrollo de este trabajo, probamos con diferentes modelos más complejos (3, 4 y más capas de convolución/pooling) las que terminaron sobreajustando el dataset (overfitting). Para contrarrestar esta situación, incluímos capas de Dropout [2, 5, 25], ajustamos las tasas de aprendizaje, Weight decay, probamos con otras normalizaciones específicas para tratar con overfitting, sin lograr resultados satisfactorios. De tener un mejor dataset estas, estas estrategias significarían un mejor rendimiento en modelos de red más complejos.

En los casos donde pudimos corregir el overfitting siguiendo las estrategias recíen mencionadas y usando un modelo de dos etapas (Fig. 3.10), nos encontramos con un rendimiento inferior al 40 % de clasificaciones correctas (4to resultado del diagrama 3.9): si bien el overfitting se atenúa, el bajo volumen de datos y la baja covarianza entre las categorías hacen que las features intermedias aprendidas sean similares para diferentes categorías. Parte del problema se trata de que, dado que la semántica de nuestras escenas está fuertemente ligada a la posición, las capas de pooling (en cantidad) hacen que la ubicación espacial de las features se pierda.

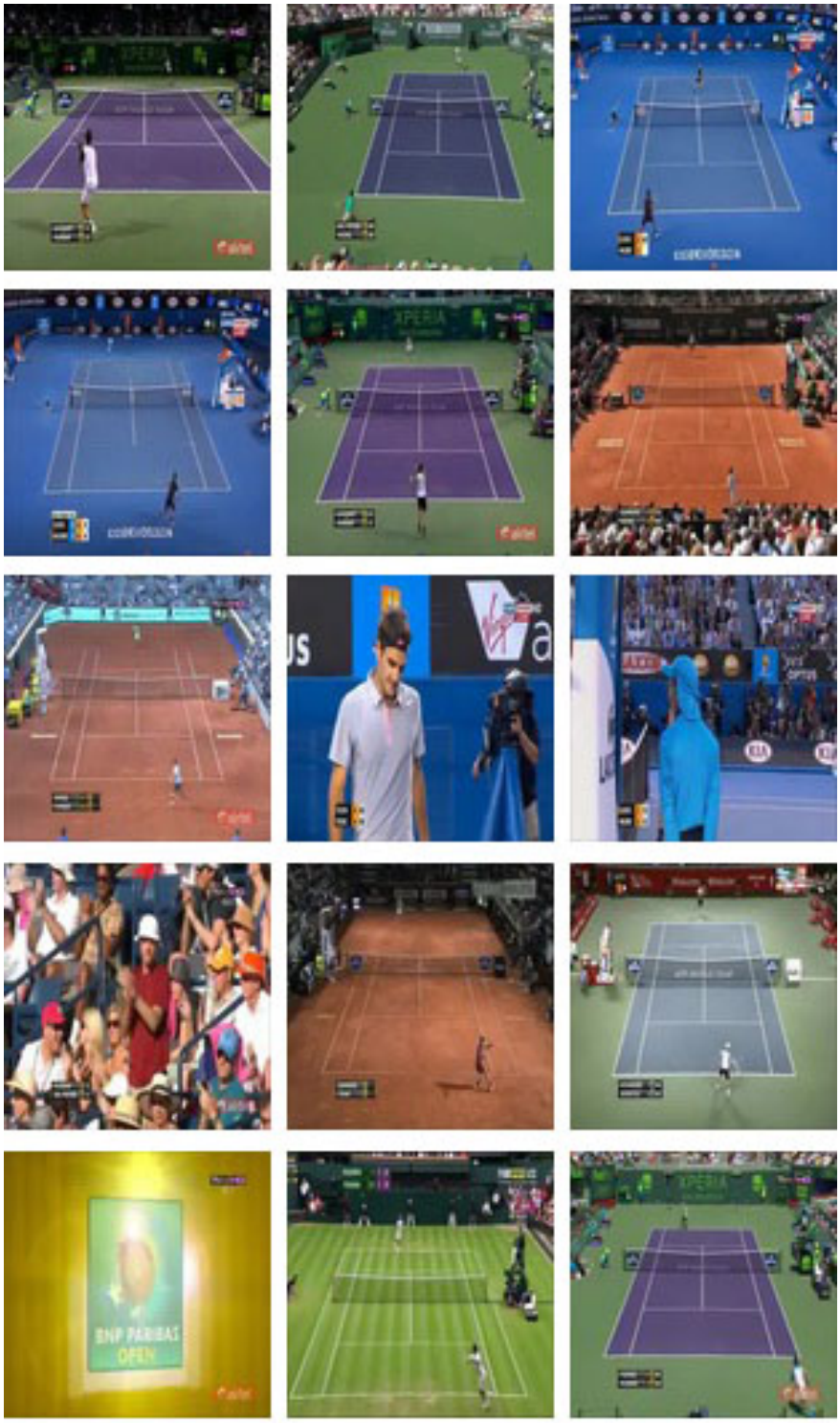


Figura 3.6: Imágenes extraídas de TEPOS para las 4 clases: derecha, izquierda, red y no juegan.

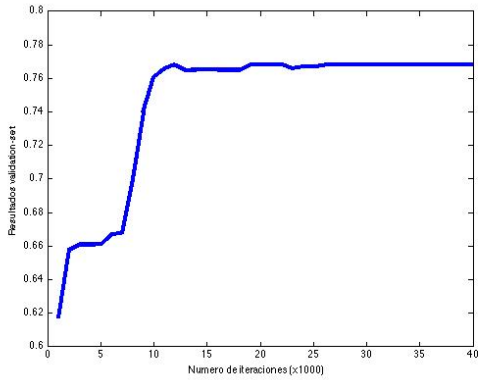


Figura 3.7: Resultados del modelo sobre el set de validación.

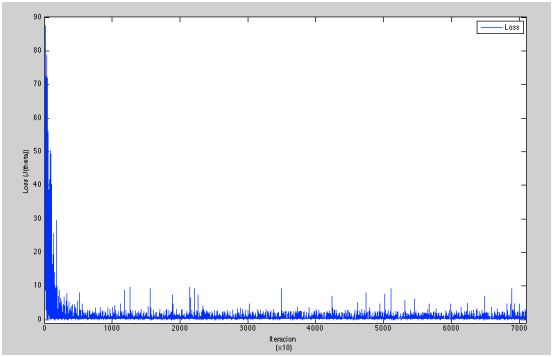


Figura 3.8: Decaimiento de la función de costo.

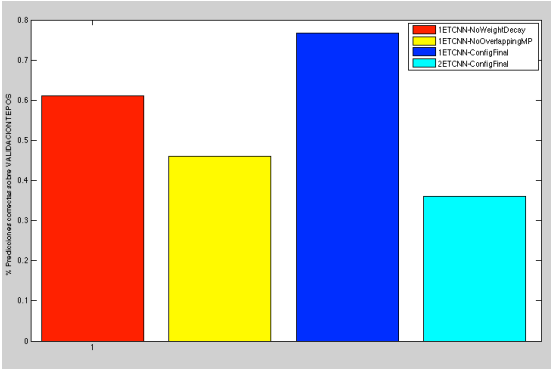


Figura 3.9: Resultados de diferentes modelos entrenados.

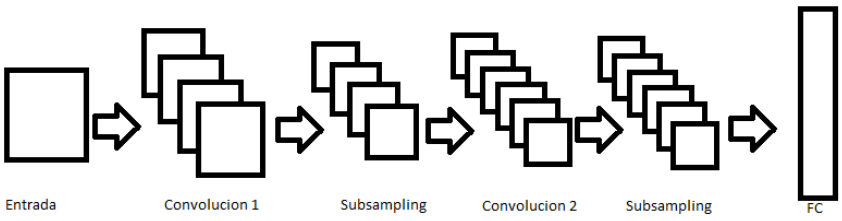


Figura 3.10: Arquitectura de 2 etapas CNN para posición de jugadores

Capítulo 4

Conclusión

4.1. Discusión

Aplicamos técnicas de aprendizaje no supervisado de features (ConvNets) para resolver el problema de comprensión de escenas en visión. Si bien el uso de CNN's ha dado resultados que superan el estado del arte de técnicas tradicionales relativo al reconocimiento de objetos, hasta el desarrollo de este trabajo no se reportan resultados en los que haya sido aplicada a comprensión de escenas, donde obtuvimos rendimientos aceptables y, a nuestro criterio, todavía largamente superables (Sección 3.5.3). Cabe destacar que, si bien los tiempos de entrenamiento a nivel de este trabajo resultan prolongados, se pueden entrenar modelos complejos en tiempos más que razonables, dando como producto tiempos de predicción acordes a sistemas de tiempo real.

Pese a la existencia de soluciones que usan técnicas tradicionales (superficiales) para el caso de estudio planteado, la generalización a otras escenas y otros problemas no es posible sin generar un nuevo algoritmo para cada caso particular.

Es nuestra opinión al igual que con lo sucedido en el reconocimiento de objetos, las CNN's marcarán la nueva tendencia para la comprensión de escenas, proveyendo soluciones generalizables y aplicables a diferentes circunstancias. Sin duda que este tipo de estructuras y modelos acercan la IA de un plano teórico a soluciones de la vida real.

4.2. Objetivos y contribuciones a futuro

Creemos que el paso inmediato para continuar con la aplicación de ConvNets a la comprensión de escenas es mejorar los resultados obtenidos aplicando las estrategias planteadas en este trabajo. Otra posible contribución consiste en aumentar el tamaño de las clases diferenciadas, por ejemplo agregando clases para sacar y recibir.

Sin duda una de las contribuciónes más significativas sería brindarle a la red la capacidad de comprender escenas desarrolladas en dimensión temporal, pasando de comprender posiciones a comprender situaciones dinámicas más complejas de juego (i.e jugadas), pudiendo así crear modelos de juego de jugadores en particular.

Apéndice A

Técnicas superficiales

A.1. Operaciones lineales en imágenes

Definición 1. *Llamaremos **píxel** a p, tal que $p \in \mathbb{R}^K$, donde p representa el vector de intensidad en cada una de las K dimensiones.*

Definición 2. *Llamaremos **imágen** a una matriz $I \in \mathbb{R}^{nxm}$ de píxeles, donde n representa las filas (alto) y m las columnas (ancho).*

Notación 1. *Una imágen será también una función f(x,y) = intensidad del píxel (x,y) de la imágen.*

Definición 3. *Para un píxel (x,y) se define la **vecindad** como una función que enmascara píxeles próximos a (x,y). La Figura A.1 muestra diferentes vecindades para el píxel (3,3).*

Si una imágen se dice que es RGB, significa que cada píxel tiene 3 intensidades: una para el azul, otra para el verde y finalmente una tercera para el rojo. En cambio, si una imágen está en escala de grises, cada píxel tiene sólo valor de gris. Por ejemplo, una imágen RGB de 512x512 se trata de una matriz de $n = 512 = m$, donde cada uno de los valores es un píxel con $K = 3$.

Si bien las definiciones anteriores son correctas para denotar las intensidades de señales, para representar los píxeles se limita el dominio dejándolo en $\mathbb{R}^K_{0/255}$ en lugar de $\mathbb{R}^K$.

A.1.1. Filtro lineal

En este tipo de operación, el valor del píxel de salida está determinado por la suma ponderada de los valores de píxeles de entradas para una vecindad dada.

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \tag{A.1}$$

donde a $h(k,l)$ se lo conoce como *kernel* o *máscara*.

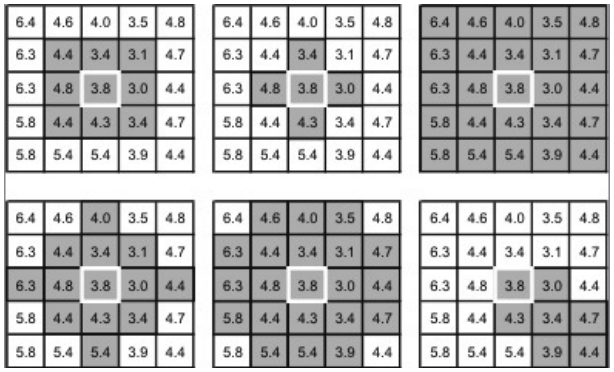


Figura A.1: Diferentes vecinos de una imágen

A.1.2. Convolución

Si revertimos los offsets de f en la ecuación anterior, obtenemos lo que se llama el *operador de convolución*, $g = f * h$:

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l) \tag{A.2}$$

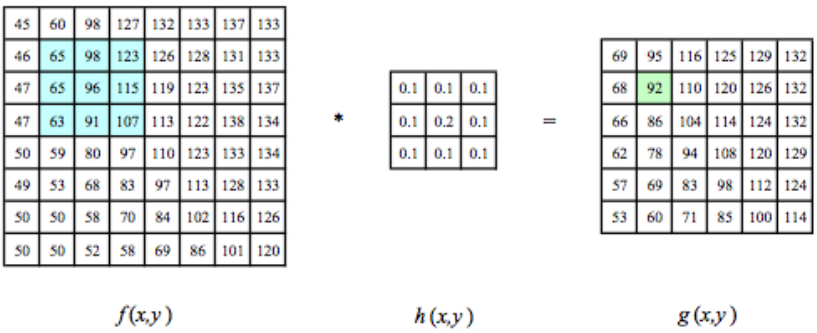
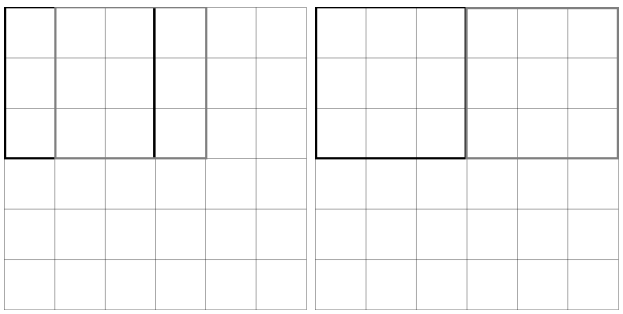


Figura A.2: Convolución en $f(x, y)$ con kernel $h(x, y)$: La imágen f es convolucionada con el filtro h para obtener g. Los píxeles celestes indican la vecindad sobre la cual se opera h para obtener el píxel verde.

Para una imágen I, la aplicación de una de estas operaciones implica deslizar una ventana del tamaño del *kernel* sobre la imágen. Estas aplicaciones del *kernel* pueden solaparse o no entre sí (Figura A.1.2).



(a) (b)

Figura A.3: Aplicación del Kernel. El recuadro negro indica la primer aplicación y el gris la segunda. En el caso (a) la selección es con superposición, en el caso (b) es sin superposición.

A.2. Estructura típica de técnicas superficiales

En el problema del scene undestanding normalmente se sigue la siguiente estructura:

1. Obtención del *frame*.
2. Preprocesamiento del *frame*.
3. Búsqueda de features por medio del uso de algún descriptor.
4. Localización del objeto.
5. Semántica de la localización del objeto.

A.3. Preprocesamiento

La etapa de preprocesamiento por lo general involucra operaciones simples que facilitan la búsqueda de *features* al descriptor. Por lo general se trata de normalizaciones, cambio de color, *cropping*[1], *down/up-scaling.*[2], entre otros.

A.4. Descriptores para detección

Para el problema de la detección/reconocimiento de objetos, en los últimos años se han desarrollado una serie de complejos descriptores.

Estos descriptores consisten de operaciones estadísticas y matemáticas que se aplican sobre la imágen, las que combinadas generan una nueva imágen que busca resaltar las características del objeto (features).

Los descriptores que explicaremos brevemente a continuación, comparten la misma estructura funcional: Primero aplican operaciones sobre la imágen, y luego clasifican la imágen en búsqueda de equivalencias con algún modelo aprendido usando técnicas de machine learning.

A continuación presentaremos algunos de los más comunes que conforman el estado del arte.

A.4.1. HOG

Es uno de los más usados. Fue introducido en el 2005 para la detección de personas [13] obteniendo muy buenos resultados. Su nombre significa "Histograma de gradientes orientados". Consiste en:

1. Calcular los gradientes: Para ello se aplica la máscara $D = [-1\ 0\ 1]$ de forma vertical ($D_y = D^T$) y horizontal ($D_x = D$). Para una imágen I, se obtiene $I_x = I * D_x$ e $I_x = I * D_y$.[3]

   La magnitud del gradiente es: $|G| = \sqrt{I_x^2 + I_y^2}$

[1] Recortar la imágen
[2] Achicar/agrandar la imágen.
[3] El operador * es el operador de convolución.

2. Determinar orientación: Para la imágen producto del paso anterior, se generan celdas de píxeles. Para cada celda se define una orientación de gradiente a partir de los gradientes de los píxeles, entre 0 y 360 grados.Una forma de obtener θ, el ángulo del gradiente en la celda es:

$$\theta = \begin{cases} tan^{-1}\frac{I_x}{I_y} & \text{si } x > 0. \\ tan^{-1}\frac{I_x}{I_y} + \pi & \text{si } y \geq 0, x < 0. \\ tan^{-1}\frac{I_x}{I_y} - \pi & \text{si } y < 0, x < 0. \\ +\frac{\pi}{2} & \text{si } y > 0, x = 0. \\ -\frac{\pi}{2} & \text{si } y < 0, x = 0. \end{cases} \tag{A.3}$$

   θ está en radianes, para obtener el ángulo en grados, hacemos $\alpha = \theta\frac{180}{\pi}$, con $\alpha \in (-180, 180)$.

3. Descriptor de bloques: Para tratar cambios en iluminación y contraste, la intensidad del gradiente debe normalizarse localmente. Para ello las celdas se agrupan en bloques vecinos. El descriptor HOG entonces se trata de un vector de las componentes de histogramas de celdas para todos los bloques. Los bloques suelen ser de geometría rectangular (R-HoG) o circular (C-HoG) y típicamente se superponen compartiendo celdas con los bloques vecinos. A partir de esta estructura se construye un **vector de *features***.

4. Normalización de bloques: Sea v el vector no normalizado que contiene todos los histogramas para un bloque, $\|v\|_k$ será la norma k para $k = 1, 2$, sea e una constante. El factor de normalización puede ser cualquiera de los siguientes:

$$L2 - norm : f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}} \tag{A.4}$$

$$L1 - norm : f = \frac{v}{\|v\|_1 + e} \tag{A.5}$$

$$L2 - Sqrt - norm : f = \sqrt{\frac{v}{\|v\|_1 + e}} \tag{A.6}$$

5. Clasificación: Sobre la imágen de entrada se define una ventana o máscara y utilizando SVM (Del inglés Support Vector Machine)[4] se decide la clase a la que pertenece la parte de la imágen cubierta por la ventana por el descriptor, según un modelo previamente entrenado usando los vectores de features.

La Figura A.5 resume secuencialmente el algoritmo para la obtencion de los descriptores de bloque.

A.4.2. DPM

Este descriptor es más moderno y es actualmente muy utilizado para detectar personas y objetos en general [18] [52]. Fue presentado en el 2012, mejorando el rendimiento de HOG en

[4]Las SVM son uno de los dispositivos más usados para resolver problema de clasificación binaria supervisada.

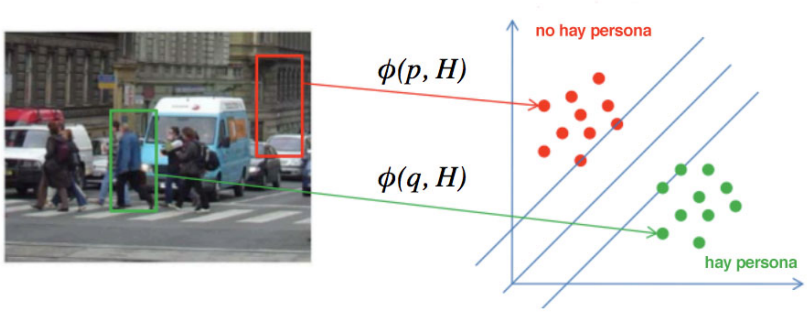


Figura A.4: Clasificación usando SVM

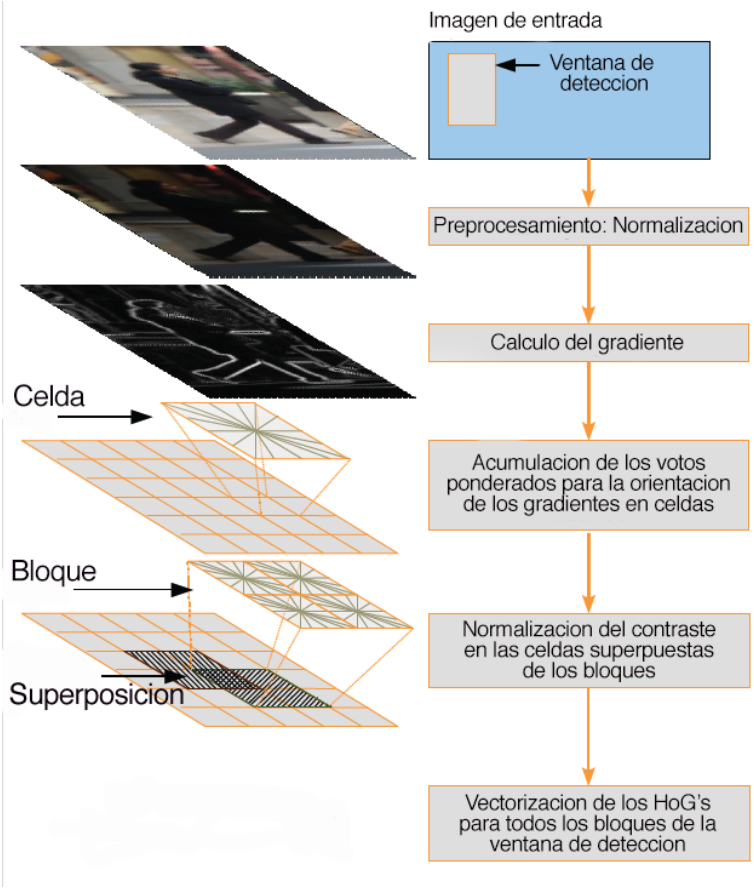


Figura A.5: Obtención de los descriptores HOG

muchos aspectos. Aporta una complejidad superior a HOG, donde a cada objeto se lo trata como una colección de partes articuladas, agregándole a cada una un factor de elasticidad, permitiéndo de esta manera representar deformaciones de las subestructuras en posiciones estándar. Es decir, un objeto se lo representa con modelo de partes deformables [5].

[5]De allí su nombre en inglés: Deformable Parts Models.

A cada parte se la representa con un descriptor de apariencia (por lo general HOG) y se le agrega un factor de deformación derivado de la ecuación del resorte de la física. Luego se busca maximizar sobre diferentes posibles configuraciones para cada una de las partes.

Otra complejidad añadida sobre HOG deriva de la articulación/conexión de las partes. En HOG el objeto se lo trata como un todo, en cambio aquí cada modelo de objeto viene acompañado de una configuración espacial de sus partes, dependiente de la posicion de los píxeles, orientación, entre otros.

Asumamos que tenemos un modelo de K partes. Llamemos l_k a la ubicación de la k-ésima parte. Sea $z = \{l_1, .., l_k\}$ una configuración posible de las K partes. Para una imágen I, llamemos $S(I, z)$ a la puntuación de la configuración de partes z para la imágen I de modo que:

$$S(I, z) = \sum_{i=1}^{K} w_i \cdot \phi(I, l_i) + \sum_{i,j \in E} w_{ij} \cdot \psi(I, l_i, l_j) \tag{A.7}$$

Queremos maximizar la equación anterior sobre z, de manera tal que para una imágen, el modelo devuelve la mejor configuración de partes.

Término de apariencia: Escribimos $\phi(I, l_i)$ para el descriptor de la imágen extraído para la ubicación l_i en la imágen I, y w_i para el descriptor HOG para la parte i.

Término de deformación: Escribiendo $dx = x_j - x_i$ y $dy = y_j - y_i$ podemos definir:

$$\psi(I, L_i, l_j) = [dx\ dx^2\ dy\ dy^2]^T \tag{A.8}$$

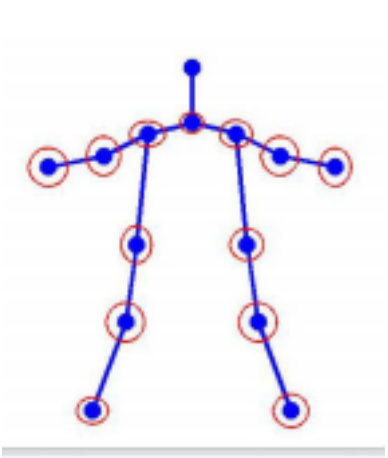
que representa la energía negativa del resorte asociada con alejar las partes j de la i una distancia canónica relativa. El parámetro w_{ij} especifica la posición de reposo y la rigidez del resorte que articula las partes i y j.

La puntuación de la configuración de z puede expresarse en términos de la apariencia y de parámetros espaciales como [51]:

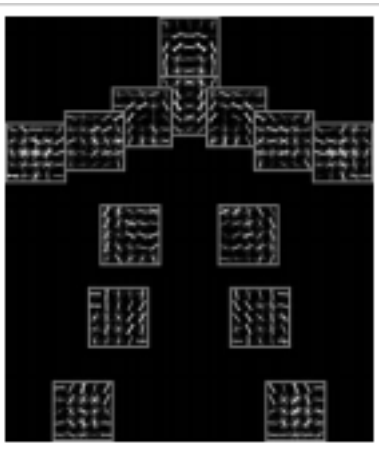
$$S(I, z) = w \cdot \Phi(I, z) \tag{A.9}$$

La figura A.4.2 muestra estos conceptos en la práctica. Cada uno de los rectángulos de (b) muestra una parte del modelo de una persona. En cada uno de los rectángulos pueden verse los descriptores HoG correspondientes. Las figuras (a) y (c) muestra el concepto de articulación de las partes.

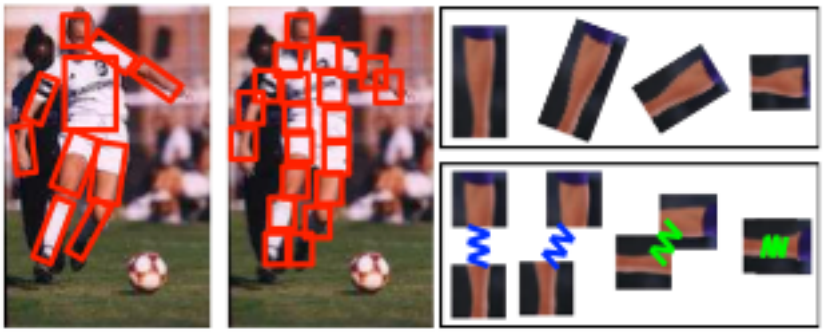
Para aprender estos modelos estructurados, se comienza con una aproximación para la localización de las partes en ejemplos positivos. A partir de esta aproximación, podemos aprender un w que minimiza usando SVM. Para un modelo w, reestimamos las etiquetas en los ejemplos positivos corriendo el modelo $argmax_{z \in Z_n} w \cdot \Phi(I_n, z_n)$. Estos pasos pueden ser comprendidos como un descenso de coordenadas en una función de costo auxiliar que depende de w y de los valores de los ejemplos positivos $Z_{pos} = \{z_n : n \in pos\}$. La prueba está en [17], transcribimos aquí la función auxiliar:



(a) Representación virtual del modelo de partes



(b) HOG de cada parte de un modelo aprendido



(c) Aplicación del modelo a una imágen

Figura A.6: DPM

$$L_{SVM}(w, Z_{pos}) = \frac{\lambda}{2}\|w\|^2 + \sum_{n \in pos} \text{máx}(0, 1 - w \cdot \Phi(I_n, z_n)) + \sum_{n \in neg} \max_{z}(0, 1 + w \cdot \Phi(I_n, z)) \tag{A.10}$$

A.5. Interpretación semántica

Una vez localizado el objeto usando un descriptor, pasa a procesarse la semántica dependiente de ese objeto. Existen métodos supervisados y no-supervisados. La utilización de una técnica o la otra depende de los patrones que se estén buscando y de la escena en particular.

Posición Se trata de ubicar espacialmente el objeto a partir de un sistema de referencia específico para la escena. Existen formas de posicionar discretas (decidir por ejemplo si el objeto está arriba o abajo de alguna referencia), como técnicas que posicionan teniendo en cuenta distancias.

Trayectoria A partir de diferencias de posición del objeto cuadro a cuadro, se traza la trayectoria para un período de tiempo t.

Heat-maps Los heatmaps permiten, para tiempos prolongados, proveer información estadística de la escena. La semántica derivada directamente de estos contempla casos como espacios más usados, trajectorias más comunes, tiempos de espera en determinado sector, etc.

Reconocimiento de actividades Es una de las formas más abstractas de semántica en video, donde se buscan coincidencias para actividades aprendidas en un modelo [49][48].

Apéndice B

Redes Neuronales

Las Redes Neuronales Artificiales, ANN por su nombre en inglés, son dispositivos de aprendizaje inspirados en el cerebro. Normalmente se las representa como un conjunto de neuronas interconectadas, en donde cada una contribuye afectando su entrada para generar una salida.

La Figura B.1 muestra el esquema típico de una neurona. Las dentritas corresponden con las entradas a la célula, el núcleo opera las entradas y el Axón es el encargado de enviar los mensajes a otras neuronas. Una persona tiene alrededor de 10^{11} neuronas, cada una con alrededor de 10^4 salidas.

La estructura de neuronas de la corteza cerebral es modular: si bien todas las partes del cerebro son relativamente similares, diferentes partes de hacen diferentes cosas; a partir de una estructura general, según la experiencia se generan nuevas estructuras específicas al problema a resolver.

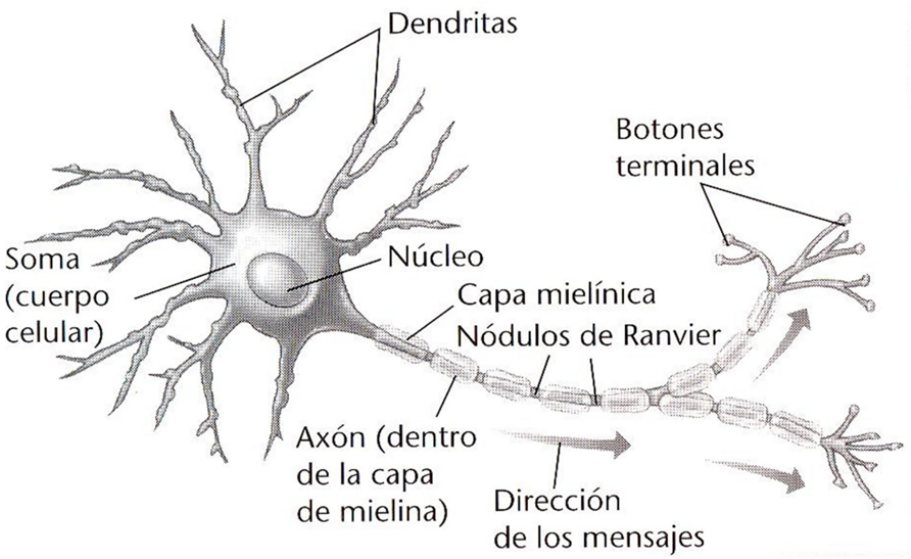


Figura B.1: Esquema de una neurona del sistema nervioso central

B.1. Representación

La palabra **red** corresponde a la existencia de interconexiones (sinapsis) de las neuronas en las diferentes capas de la estructura. Un ejemplo simple se puede ver en la Figura B.2. La primer capa tiene las neuronas de entrada quienes se conectan por sinapsis a la capa oculta, y luego via más sinapsis a la tercer capa, la capa de salida. Cada conexión entre neuronas tiene un parámetro llamado en la literatura parámetro, peso o peso de activación que influye en los cálculos que se realizan en la neurona para generar su salida. La propiedad de aprendizaje de la red neuronal viene dada por la capacidad de variar que tienen los pesos.

Notación 2. *Al peso de una sinapsis lo denominamos con la letra griega θ_i o con la letra w_i, por el inglés **weight**. El subíndice i indica que el peso se corresponde con la entrada (o actividad) i-ésima, x_i.*

Definición 4. *Llamaremos $\Theta^{(j)}$ a la matriz de pesos correspondientes a las sinapsis de la capa j con la capa $j+1$.*

Cada salida y de una neurona viene dada por tres funciones:

1. Una **funcion de propagación** $g(\theta)$, que por lo general consiste suma de cada entrada multiplicada por el peso de su interconexión. Si el peso es positivo, la conexión se denomina excitatoria; si es negativo, se denomina inhibitoria.

2. Una **función de activación** $a_i^{(j)}$, donde (j) indica la capa e i la neurona, que modifica a la anterior. Puede no existir, siendo en este caso la salida la misma función de propagación.

3. Una **hipótesis** $h_\theta(x)$, que se aplica al valor devuelto por la función de activación. Se utiliza para acotar la salida de la neurona y generalmente viene dada por la interpretación que queramos darle a dichas salidas. Por lo general representa la salida de la última capa de una red neuronal.

B.2. Modelos

Una ANN se la define generalmente por tres parámetros:

- La estructura de interconexión de las capas de neuronas.
- El proceso de aprendizaje de actualización de Θ.
- La función de activación.

A continuación se enumeran algunos modelos de neuronas. Estos modelos son mucho más simples que las neuronas reales, sin embargo permiten crear ANN's capaces de aprender cosas interesantes.

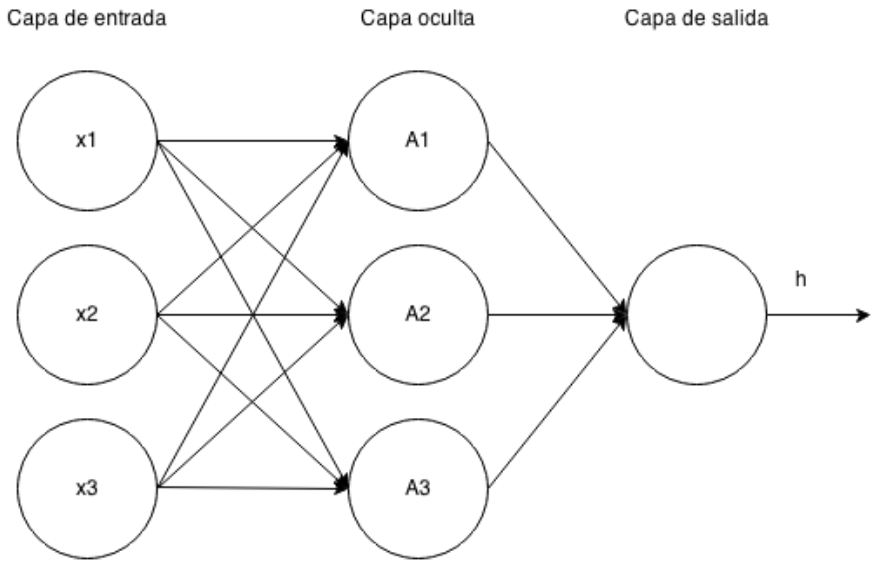


Figura B.2: Esquema de un perceptrón multicapa con una capa oculta

Neuronas lineales Son una de las estructuras más simples de neuronas. La salida y de la neurona queda definida como:

$$y = b + \sum_i x_i w_i \tag{B.1}$$

Donde b es una constante (*bias*), x_i es la i-ésima activación y w_i es el peso correspondiente. Ver Figura B.3.

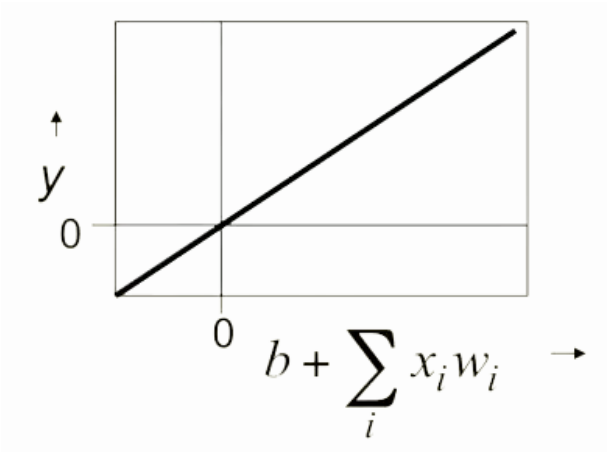


Figura B.3: Salida de una neurona lineal

Neuronas McCulloch-Pitts Introducidas en 1943, se tratan de neuronas con salida binaria. La particularidad es que su activación queda condicionada por un umbral. Los inventores tra-

taron las salidas como valores de verdad de lógica proposicional. Las neuronas se combinaban para construir proposiciones más grandes. Hay dos formas equivalentes de escribir la ecuación de y para estas neuronas:

$$z = \sum_i x_i w_i \quad y = \begin{cases} 1, & \text{si } z \geq umbral. \\ 0, & \text{caso contrario.} \end{cases} \tag{B.2}$$

o también se puede escribir

$$z = b + \sum_i x_i w_i \quad y = \begin{cases} 1, & \text{si } z \geq 0. \\ 0, & \text{caso contrario.} \end{cases} \tag{B.3}$$

Donde la equivalencia esta dada por $umbral = -b$

Rectified Linear neurons (ReLU) Computan una suma lineal ponderada de sus entradas, y su salida y es una función no-lineal.

$$z = b + \sum_i x_i w_i \quad y = \begin{cases} z, & \text{si } z \geq 0. \\ 0, & \text{caso contrario.} \end{cases} \tag{B.4}$$

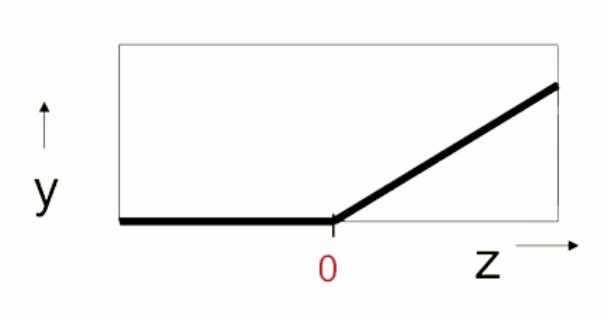


Figura B.4: Salida de una neurona ReLU

Neuronas Sigmoideas Son muy utilizadas en la práctica. Un ejemplo simple de neuronas sigmoideas, es la neurona que calcula la función logística, donde para valores grandes de z, y tiende a 1, en cambio para valores pequeños, y tiende a 0. En la Figura B.5 se muestra este comportamiento.

$$z = b + \sum_i x_i w_i \quad y = \frac{1}{1 + e^{-z}} \tag{B.5}$$

Este tipo de neuronas tienen la propiedad de que son derivables fácilmente algo, como veremos, valioso a la hora de entrenar el modelo.

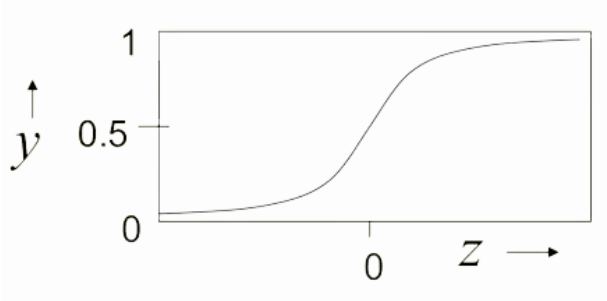


Figura B.5: Salida de una neurona sigmoidea

B.3. Arquitecturas

B.3.1. Redes "Feed-Forward" (FFNN)

Son la estructura más simple de redes neuronales. La Figura B.2 es un ejemplo de un Perceptrón Multicapa. La característica que define a esta categoría es que las sinapses siempre ocurren de capa inferior a capa superior, desde la capa de entrada hasta la de salida. Las redes Feed Forward, computan una serie de transformaciones que cambian las similaridades de los posibles casos, obteniendo asi, una nueva representación de la entrada para cada capa. El cambio en las similaridades de los casos se logra haciendo que, la salida de la capa j-ésima sea una función no lineal de la salida de la capa (j-1)-ésima.

Definición 5. *Una ANN del tipo Feed-Forward será una* ***Deep Network****, si tiene más de una capa oculta entre la de entrada y la de salida.*

B.3.2. Redes recurrentes

Las redes recurrentes son mucho más poderosas que las Feed Forward. Si se las representa gráficamente, cuentan con ciclos dirigidos (Figura B.6). Por lo general, estas redes cuentan con una dinámica muy compleja, lo que hace que el entrenamiento sea difícil, sin embargo, proveen una forma muy natural de modelar datos secuenciales.

B.4. Entrenamiento

En los años '80, las redes FFNN tomaron fuerza dentro del ambiente científico debido al descubrimiento del algoritmo de aprendizaje llamado *"Backpropagation"*. El entrenamiento se realiza minimizando una función de costo $J(\Theta)$ propia del modelo, con Θ vector de parámetros del mismo.

Notación 3. *Para problema el de clasificación en K clases, llamaremos* $(h_\Theta(x))_i$ *a las i-ésima salida, donde* $h_\Theta(x) \in \mathbb{R}^K$.

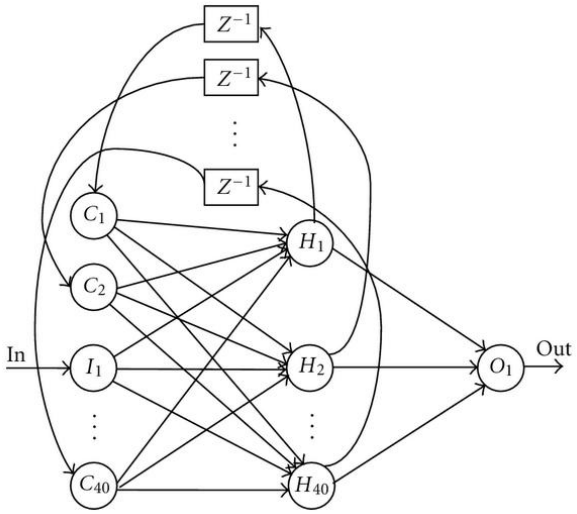


Figura B.6: Ejemplo de una red recurrente

Definición 6. *Un caso de entrenamiento para un problema supervisado será un par* (x, y)*, donde* x *es la entrada e* y *es la clase a la que* x *pertenece* [1]*. Un training set es un conjunto de casos de entrenamiento* $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})$*.*

Definición 7. *Para ANN's, la función de costo será*

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)}))_k) + (1 - y_k^{(i)}) log(1 - h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2 \tag{B.6}$$

donde L *es la cantidad de capas de la red neuronal y* s_l *el numero de neuronas de la capa* l*-ésima.*

El segundo sumando de la ecuación de la función de costo se lo llama factor de regularización. Por ahora lo ignoraremos, esto es $\lambda = 0$[2].

Definición 8. *Llamaremos* $\delta_j^{(l)}$ *al error del costo para* $a_j^{(l)}$*. En efecto,*

$$\delta_j^{(l)} = \frac{\partial}{\partial \Theta_{ij}^{(l)}} y^{(i)} log h_\theta(x^{(i)}) + (1 - y^{(i)}) log h_\theta(x^{(i)}) \tag{B.7}$$

Se puede comprobar que

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} = D_{ij}^{(l)} \tag{B.8}$$

Podemos ahora esquematizar el **Algoritmo de Backpropagation** como:

1. $\Delta_{ij}^{(l)} := 0$ `/*para todos los l,i,j*/`
2. `for i=1,...,m {`
3. $a^{(1)} := x^{(i)}$ `/*termino b de cada capa*/`
4. `computar` $a^{(l)}$ `para l=2,3,..,L`
5. $\delta^L := a^{(L)} - y^i$
6. `computar` $\delta^{(L-1)}, \delta^{(L-2)}, .., \delta^{(2)}$
7. $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(L+1)}$
8. }
9. `if (j == 0){`
10. $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$
11. `}else{`
12. $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$
13. }

[1] En la literatura, a y suele identificársela con la letra t, por su nombre en inglés (target).

[2] El factor de regularización ayuda a resolver problemas de sobreajuste. Existen varios tipos de factores de regularización [60].

Bibliografía

[1] S.S. Ahranjany, F. Razzazi, y M.H. Ghassemian. A very high accuracy handwritten character recognition system for farsi/arabic digits using convolutional neural networks. *Bio-Inspired Computing: Theories and Applications (BIC-TA)*, 2010.

[2] R.M. Bell y Y. Koren. Lessons from the netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 2007.

[3] A. Berg, J. Deng, y L. Fei-Fei. Large scale visual recognition challenge. 2010.

[4] Y-Lan Boureau, Nicolas Le Roux, Francis Bach, Jean Ponce, y Yann LeCun. Ask the locals: multi-way local pooling for image recognition. En *Proc. International Conference on Computer Vision (ICCV'11)*. 2011.

[5] L. Breiman. Random forests. *Machine Learning 45(1):5-32*, 2001.

[6] D.C. Ciresan, U. Meier, J. Masci, L.M. Gambardella, y J. Schmidhuber. High performance neural networks for visual object classification. *Arxiv preprint arXiv:1102.0183*, 2011.

[7] Dan Cireşan, Ueli Meier, y Juergen Schmidhuber. Multi-column deep neural networks for image classification. *CVPR*, 2012.

[8] Navneet Dalal. Inria's pedestrian dataset. 2014. URL http://pascal.inrialpes.fr/data/human/.

[9] Bao Dang, An Tran, Tien Dinh, y Thang Dinh. A real time player tracking system for broadcast tennis video. *ACIIDS'10 Proceedings of the Second international conference on Intelligent information and database systems: Part II*, 2010.

[10] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, y A. Ng. Large scale distributed deep networks. *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, y Fei-Fei L. ImageNet: A Large-Scale Hierarchical Image Database. *Computer Vision and Pattern Recognition*, 2009.

[12] P. Dollar. Caltech's pedestrian dataset. 2014. URL http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/.

[13] Navneet Dalal et al. Histograms of oriented gradients for human detection. 2005. URL http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf.

[14] Clement Farabet, Camille Couprie, Laurent Najman, y Yann LeCun. Scene parsing with multiscale feature learning, purity trees, and optimal covers. En *Proc. International Conference on Machine learning (ICML'12)*. 2012.

[15] Clement Farabet, Camille Couprie, Laurent Najman, y Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013. In press.

[16] L. Fei-Fei, R. Fergus, y P. Perona. Learning generative visual models from a few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding, 106(1):59-70*, 2007.

[17] P. Felzenszwalb, D. McAllester, y D. Ramanan. A discriminatively trained, multiscale, deformable part model. *Computer Vision and Pattern Recognition*, 2008.

[18] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, y D. Ramanan. Object Detection with Discriminatively Trained Part Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[19] A. Frome, G. Cheung, A. Abdulkader, M. Zennaro, B. Wu, A. Bissacco, H. Adam, H. Neven, y L. Vincent. Large-scale privacy protection in street-level imaginery. *ICCV*, 2009.

[20] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. 1980.

[21] K. Fukushima, S. Miyake, y T. Ito. Neocognitron: a neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-13(Nb. 3)*, 1983.

[22] Guangyu Zhu Changsheng Xu Qingming Huang Wen Gao y Liyuan Xing. Player action recognition in broadcast tennis video with applications to semantic analysis of sports game. 2006.

[23] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, y Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. 2014.

[24] G. Griffin, A. Holub, y P. Perona. Caltech 256 object category dataset. *Technical Report 7694. California Institute of Technology*, 2007.

[25] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutsjever, y R.R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[26] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, y Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. 2012.

[27] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, y Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. En *International Joint Conference on Neural Networks*, 1288. 2013.

[28] K. Jarret, K. Kavukcuoglu, M.A. Ranzato, y Y. LeCun. What is the best multi-stage architecture for object recognition? *International Conference on Computer Vision*, 2009.

[29] A. Karpathy. ConvNetJS: deep learning library. supports convolutional (and ordinary) neural networks. 2014.

[30] Koray Kavukcuoglu, Pierre Sermanet, Y-Lan Boureau, Karol Gregor, Michaël Mathieu, y Yann LeCun. Learning convolutional feature hierachies for visual recognition. En *Advances in Neural Information Processing Systems (NIPS 2010)*, tomo 23. 2010.

[31] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. 2009.

[32] A Krizhevsky. Convolutional deep belief networks on Cifar-10. *Unpublished Manuscript*, 2012.

[33] A. Krizhevsky y G. Hinton. Using very deep autoencoders for content-based image retrieval. *ESANN*, 2011.

[34] A. Krizhevsky, I. Sutskever, y G. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. 2012.

[35] Y. LeCun. A Theoretical Framework for Back-Propagation. 1988.

[36] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, y L.D. Jackel. Handwritten digit recognition with a backpropagation network. *Advances in neural information processing systems*, 1990.

[37] Y. LeCun, L. Bottou, Y. Bengio, y P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[38] Y. LeCun, L. Bottou, G.B. Orr, y K. Müller. Efficient backprop. 1998.

[39] Y. LeCun, C. Cortes, y C. Burges. Mnist: A dataset of handwritten digits. 2014. URL http://yann.lecun.com/exdb/mnist/.

[40] Y. LeCun, F.J. Huang, y L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. *Computer Vision and Pattern Recognition*, 2004.

[41] Y. LeCun, K. Kavukcuoglu, y C. Farabet. Convolutional networks and applications in vision. *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium*, 2010.

[42] H. Lee, R. Grosse, R. Ranganath, y A.Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

[43] T. Mensink, J. Verbeek, F. Perronnin, y G. Csurka. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. *ECCV - European Conference on COmputer Vision, Florence Italy*, 2012.

[44] M. Moreira y E. Fiesler. Neural networks with adaptive learning rate and momentum terms. *IDIAP Technical report*, 1995.

[45] V. Nair y G. Hingon. Rectified linear units improve restricted boltzmann machines. *Proc 27th International Conference on Machine Learning*, 2010.

[46] N. Pinto, D.D. Cox, y D. DiCarlo, J.J. adn Doukhan. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLos computational biology*, 2009.

[47] N. Pinto, D.D. Cox, y J.J. DiCarlo. Why is real-world visual object recognition hard? *PLos computational biology*, 2008.

[48] G. Pusiol, F. Bremond, y M. Thonnat. Trajectory Based Discovery. *7th IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS), Boston*, 2010.

[49] G. Pusiol, F. Bremond, y M. Thonnat. Unsupervised Discovery and Recognition of long term Activities. *The 8th International Conference on Computer Vision Systems. (ICVS)*, 2011.

[50] P. Pusiol y G. Pusiol. Pedestrian detection using convnets and INRIA dataset. *Unpublished*, 2014.

[51] D. Ramanan. Visual Analysis of Humans. 2011.

[52] D. Ramanan. Articulated Human Detection with Flexible Mixtures-of-Parts. 2013.

[53] A.W. Roe, S.L. Pallas, Y.H. Kwon, y M. Sur. Visual projections routed to the auditory pathway in ferrets: receptive fields of visual neurons in primary auditory cortex. *J. Neurosci. 12: 3651-3664.*, 1992.

[54] B.C. Russell, A. Torralba, K.P. Murphy, y W.T. Freeman. Lableme: a database and web-based tool for image annotation. *International journal of computer vision*, 2008.

[55] J. Sanchez y F. Perronnin. High-dimensional signature compression for large-scale image classification. *Computer Vision and Pattern Recognition, (CVPR'11)*, 2011.

[56] Tom Schaul, Sixin Zhang, y Yann LeCun. No more pesky learning rates. En *Proc. International Conference on Machine learning (ICML'13)*. 2013.

[57] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, y Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. 2013.

[58] Pierre Sermanet y Yann LeCun. Traffic sign recognition with multi-scale convolutional networks. 2011.

[59] P.Y. Simard, D. Steinkraus, y J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, 2003.

[60] S. N. Srihari. Regularization in neural networks. 2012.

[61] J. Stallkamp, M. Schlipsing, J. Salmen, y C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, (0):–, 2012. ISSN 0893-6080. doi:10.1016/j.neunet.2012.02.016. URL http://www.sciencedirect.com/science/article/pii/S0893608012000457.

[62] Ilya Sutskever, James Martens, y Geoffrey Hinton. Generating text with recurrent neural networks. 2011.

[63] S.C. Turaga, F.J. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk, y H.S. Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural Computation,22(2):511-538*, 2010.

[64] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, y Rob Fergus. Regularization of neural networks using dropconnect. En *Proc. International Conference on Machine learning (ICML'13)*. 2013.