Aprendizaje multiclase de videoimágenes deportivas con arquitecturas profundas

Tesina de Grado
Licenciatura en Ciencias de la Computación

Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

Martín Escarrá

Director Guillermo Grinblat

Codirector Pablo Granitto

18 de marzo de 2016

Resumen

Las arquitecturas profundas permiten representar de manera compacta funciones altamente no lineales. Entre ellas, las redes convolucionales han adquirido gran protagonismo en la clasificación de imágenes debido a la invarianza traslacional de sus features. Este trabajo propone investigar un abordaje naïve para la clasificación de videoimágenes con redes profundas, comparar la performance de redes pre-entrenadas con la de redes ad-hoc y finalmente crear un mecanismo de visualización de la representación interna de la arquitectura. Como ejemplo de aplicación se utilizarán segmentos de videos deportivos con diferentes acciones grupales.

Agradecimientos

Agradezco enormemente a Guille, a Pablo, y a todo el grupo de investigación por la buena predisposición para con la realización de este trabajo. Siempre voluntariosos y pacientes, me acompañaron en esta etapa final de la carrera. Gracias a las autoridades, particularmente a Ana, que sistemática y expeditivamente facilitó cada trámite administrativo que he tenido que realizar, ¡incluso durante las vacaciones! A los docentes, especialmente a Dante, por enseñar con pasión y acompañar siempre a los estudiantes. Es un placer aprender con docentes así. A mis amigos y compañeros de estudio: Mauro, Eugenia, Ignacio, Joaquín y Joaquín; que marcaron el ritmo y me tendieron una mano más de una vez para que no pierda el ánimo. A mis amigos: Fede, Romi, Dami y Joaco; invaluables, sinceros y llenos de bondad, me hacen sentir muy afortunado. A Vicky, mi pequeño gran amor, que fue mi apoyo emocional y me ayudó en incontables ocasiones, incluso corrigiendo en esta tesina mi prosa desordenada y repetitiva.

La última mención, y tal vez la más importante, es a mi familia. Nada de esto hubiese sido posible sino fuera por la ayuda incondicional de mi madre, Nora, mis hermanos, Paula y Manuel, y mi abuela Elvira, que todavía piensa que me gradúo de ingeniero. A ellos este trabajo.

Índice general

Capítulo 1

Introducción

Las tecnologías de la información (TICs) están produciendo una nueva revolución tecnológica. El Aprendizaje Automatizado (*Machine Learning*) es fundamental en esta revolución basada en el uso inteligente de la información. Su principal objetivo es la creación de algoritmos que mejoren su eficiencia en la solución de un problema de manera automática, a través de la experiencia acumulada [4], con una mínima cantidad de intervención humana. El Aprendizaje Automatizado integra conocimientos de disciplinas diversas como Inteligencia Artificial, Estadística, Mecánica Estadística, Ciencia Cognitiva, Neurobiología, etc. Uno de los principales problemas que se investigan en esta área es el de reconocimiento de patrones y su clasificación [5], que corresponde a asignar una nueva observación a una clase particular.

La mayoría de estos métodos de análisis de datos se basan en lo que puede definirse como arquitecturas poco profundas (Redes Neuronales con una capa oculta, SVM, Árboles de Decisión, etc.), aunque desde bastante tiempo se sabe que las arquitecturas profundas pueden ser mucho más eficientes a la hora de representar ciertas funciones. Esto se debe, probablemente, a que solo recientemente se pudo desarrollar un método de aprendizaje efectivo para arquitecturas profundas [9]. Estas redes profundas han mostrado gran efectividad resolviendo problemas sobre todo del área de *Machine Vision*: reconocimiento de imágenes [7], [9], secuencias de video [11] y datos de captura de movimiento [14], [15]. En particular, resulta de especial interés para este plan el desarrollo de redes convolucionales: redes profundas con conexiones locales y pesos compartidos que pueden ser utilizadas para la extracción de features robustos a variaciones traslacionales [16], [17], [22]. Con estos modelos se abrió toda una nueva área de estudio dentro de Aprendizaje Automatizado, conocida como *Deep Learning*.

Muchos de los trabajos actuales que usan estas técnicas para reconocimiento de imágenes parten de un modelo pre-entrenado con un conjunto de datos grande (generalmente *ImageNet*) para luego adaptarlo al problema particular que se deba atacar [18], [25]. Esto se debe a dos razones: por un lado, para capturar conceptos que puedan ser importantes para la tarea puede ser necesario un modelo con gran profundidad, y por otro lado, para entrenar un modelo de estas características se necesita una gran cantidad de datos, que generalmente no se tienen, y mucho tiempo de cómputo.

Por otro lado, si bien el análisis de videos deportivos fue atacado anteriormente con técnicas tradicionales [6], [10], son escasos los trabajos que han utilizado técnicas de *Deep Learning* en este problema. Recientemente, [23] propuso un método que hace uso de redes neuronales recurrentes para clasificar videos de distintas disciplinas deportivas, potencialmente con escenarios muy diversos que contienen pistas importantes para la clasificación. En este trabajo, en cambio, usaremos arquitecturas profundas no recurrentes para clasificar distintas acciones de un único deporte.

1.1. Objetivos

El objetivo general de este proyecto es el estudio de técnicas de análisis de datos basadas en arquitecturas profundas especialmente adaptadas al problema de clasificación de videoimágenes deportivas. En particular se investigará si, partiendo de un conjunto de videos relativamente pequeño pero enfocado en un tema acotado, en este caso escenas de partidos de rugby, pueden obtenerse resultados satisfactorios sin recurrir a métodos de *Transfer Learning*, que explotan las características aprendidas por otro modelo (potencialmente más complejo) en otro conjunto de datos.

1.2. Organización del trabajo

El trabajo está organizado de la siguiente manera: El Capítulo 2 explica, a modo de introducción, todos los conceptos propios del Aprendizaje Automatizado necesarios para entender la tesina. El Capítulo 3 muestra la base de los distintos modelos usados en los experimentos, así como los datos con los que fueron entrenados y justifica algunas decisiones de diseño. Luego, en el Capítulo 4 se muestran los modelos obtenidos combinando los antes mencionados y se detallan los ajustes que fueron necesarios aplicar. Además, se compara la performance de los distintos modelos entrenados y se proponen

técnicas para la visualización de los conceptos aprendidos. Por último, el Capítulo 5 compone un resumen final de lo investigado y una lista de posibles proyectos a futuro.

Capítulo 2

Conceptos Generales

En este capítulo se explican todos los conceptos requeridos para poder entender la tesina, ordenados crecientemente por generalidad y fecha de aparición en publicaciones científicas. Primero, se dará una breve introducción al Aprendizaje Automatizado, y luego se exhibirá, detalladamente, la evolución de los modelos y algoritmos del área.

2.1. Aprendizaje Automatizado

A fines de los '50, Arthur Samuel, que fue uno de los primeros en definir *Machine Learning*, dijo que es el campo de estudio que otorga a las computadoras la capacidad de aprender sin ser explícitamente programadas. En 1997, Tom Mitchell dio una definición más formal, ampliamente citada [4]: «*Se dice que un programa* aprende *de la experiencia* E *con respecto a cierto tipo de tarea* T *y una medida de performance* P*, si el desempeño en tareas de* T *según la medida* P*, mejora con la experiencia* E». Mitchell se refiere a los programas que evolucionan con la experiencia acumulada.

Más formalmente, el objetivo de todo proceso de aprendizaje automatizado es encontrar una función

$$h : X \to Y$$

que es alguna hipótesis que tenemos sobre los datos.

El Aprendizaje Automatizado se divide principalmente en:

- Supervisado
  Donde los datos de entrenamiento D constan de una señal supervisora que indica el valor de salida Y deseado para la entrada X.

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \in X \times Y$$

Si Y tiene valores reales entonces se dice que es un problema de *regresión*, mientras que si sus valores son discretos entonces se trata de un problema de *clasificación*. Dentro de los problemas de clasificación podemos distinguir entre los que cuentan con tan solo dos *clases* o *etiquetas*, y aquellos con tres o más. De aquí en adelante los llamaremos problemas *binarios* y *multiclase*, respectivamente.

- No supervisado
  Donde no existe un valor de salida deseado para cada entrada, y por lo tanto se intentan encontrar estructuras ocultas en los datos de entrada X.
  $$D = \{x_1, x_2, \ldots, x_n\} \in X$$
  Algunos ejemplos de estas técnicas son:
  - *Clustering* (p. ej., K-means, Hierarchical clustering, etc.).
  - *Feature learning* (p. ej., RBM, Autoencoders, etc.).

Aprendizaje Supervisado

Esencialmente, todo proceso de Aprendizaje Supervisado consta, a grandes rasgos, de las siguientes cuatro etapas:

- Adquisición de datos
  Se obtiene el conjunto de datos crudo sobre el cual se desea aprender alguna característica.
- Preprocesamiento
  Se procesan los datos crudos, posiblemente eliminando información redundante, reduciendo dimensionalidad, rotando y normalizándolos.
- Entrenamiento
  Se define una métrica para el error del método, y se usa una porción de los datos para ajustar los parámetros del modelo, minimizando el error.
- Testing o Generalización
  Se usa otra porción de los datos (diferente a la que se utiliza en la etapa anterior) para medir la capacidad de generalización del método en datos nunca vistos.

Los métodos más emblemáticos del Aprendizaje Supervisado son: *Árboles de Decisión*, *Support Vector Machine*, *kNN*, *Naive Bayes* y *Redes Neuronales*. Algunos de los cuales se verán con cierto detalle en las siguientes subsecciones.

Sobreajuste

Yendo más en profundidad, podemos diferenciar tres conjuntos de datos: entrenamiento, generalización y validación. Los dos primeros siendo utilizados como se explicó antes, mientras que el de validación servirá durante el entrenamiento para aproximar el error de generalización. Esto es importante porque casi ningún método puede ser aplicado sin antes calibrar algunos parámetros. Para esto, el error de referencia será el de validación. Más aún, en la mayoría de los métodos, el modelo se ajusta demasiado a los datos de entrenamiento, tanto así que aprende conceptos específicos del entrenamiento que no generalizan a casos nunca vistos.

Este último problema se conoce como *sobreajuste* y afecta a gran parte de los métodos de aprendizaje supervisado. Como ejemplo, imagine que necesita aproximar una función con datos extraídos de un muestreo de una función cuadrática. Imagine además que los datos no forman perfectamente una función cuadrática sino que tienen un cierto error, introducido por redondeo.

Su espacio de hipótesis $\mathcal{H}$ es el conjunto de todos los polinomios con coeficientes reales:

$$\mathcal{H} = \{\sum_{i=0}^{n} a_i x^n \mid a_i \in \mathbb{R}, n \in \mathbb{N}\} \tag{2.1}$$

Es claro ver que a medida que vamos aumentando el grado del polinomio iremos reduciendo el error de aproximación (ver Figura 2.1), sin embargo nuestra generalización será cada vez más pobre debido al sobreajuste generado por una hipótesis demasiado específica. Contando entonces con un conjunto de validación podremos aproximar el error de generalización y escoger así la hipótesis que minimice dicho error.

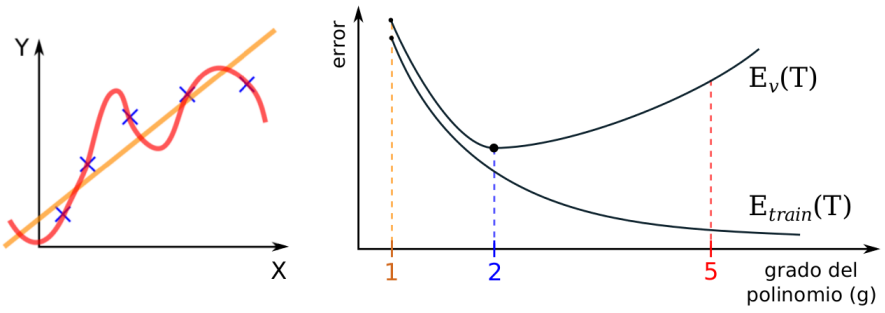


Figura 2.1: La gráfica de la izquierda[1] muestra los puntos que se desean aproximar (cruces azules) y dos posibles hipótesis, una lineal y otra de grado 5. A la derecha vemos el error de entrenamiento y validación.

Sesgo inductivo

Otro concepto importante es el de sesgo inductivo. En términos generales, *sesgo inductivo* es el conjunto de suposiciones que se hacen para predecir una salida sobre casos nunca vistos. Concretamente, todo aquello que supongamos sobre la naturaleza de la función objetivo será nuestro sesgo, y está dado principalmente por el espacio de hipótesis que exploramos, y el algoritmo que usamos para hacerlo.

En el caso que ilustra la Figura 2.1, el espacio de hipótesis definido en la Ecuación 2.1 es parte de nuestro sesgo inductivo dado que representa una cota para el espacio de búsqueda. Si los datos hubiesen sido tomados del muestreo de una función exponencial, entonces nunca habríamos obtenido una buena aproximación. Esta es entonces, la decisión que más sesga el aprendizaje, pero no es la única. Imagine, a modo de ejemplo, que los errores de validación arrojados por dos polinomios de distinto grado son muy similares, ¿qué polinomio deberá elegir nuestro algoritmo? Aquí podría aparecer otro caso muy común de sesgo inductivo, *la navaja de Occam*, que reza:

> «Debe preferirse la hipótesis más simple consistente con los datos.»

En el ejemplo de la Figura 2.1, si el aprendizaje está sesgado por la navaja de Occam entonces dará prioridad al polinomio de menor grado.

Todas estas suposiciones que hace el sistema sobre el concepto a aprender, conforman lo que se denomina la *base inductiva* del mismo.

2.2. Métodos actuales

Support Vector Machine

SVM o *Support Vector Machine*, inventado por *Vladimir N. Vapnik et al* [3], es uno de los métodos de Aprendizaje Supervisado más reconocidos históricamente y es ampliamente utilizado hoy en día. El planteo original aplicaba solo para problemas binarios, separadores lineales y clases linealmente separables, tal como se ilustra en la Figura 2.2.

Intuitivamente, SVM busca el hiperplano que mejor separa los puntos pertenecientes a cada clase. Cada uno de estos puntos es un vector en un espacio p-dimensional y el hiperplano que se busca tiene dimensión $p-1$.

[1] Fuente: Tesina de grado de Leonardo Morelli, *Adaptación de arquitecturas profundas a problemas no estacionarios* (2013).

Formalmente, considere un conjunto de tuplas:

$$D = \{(x, y) \mid x \in \mathbb{R}^p, y \in \{-1, 1\}\}$$

de cardinalidad n, donde y representa la clase o etiqueta correspondiente a x. Para algún hiperplano:

$$\hat{w}^T x + \tilde{b} = 0$$

dado por $\hat{w}$ y $\tilde{b}$, que separe las clases, la distancia del hiperplano a un vector x_i es:

$$r = \frac{|\hat{w}x_i + \tilde{b}|}{\|\hat{w}\|}.$$

Llamamos entonces *vectores soporte* a los vectores de cada clase más próximos al hiperplano, es decir, para los cuales r es la mínima.

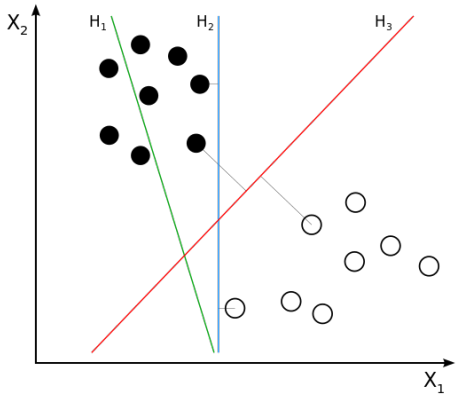


Figura 2.2: En este gráfico[2] tenemos dos clases de puntos (negros y blancos) y tres ejemplos de hipótesis: H_1, que no separa la clases, H_2, que separa las clases pero con un pequeño margen, y H_3, que separa las clases con el margen más amplio.

Imagine que luego hacemos pasar un hiperplano, paralelo al original, por cada vector soporte, como se muestra en la Figura 2.3 y considere que ρ es la distancia entre estos hiperplanos, también llamado *margen*. Entonces para cada par $(x_i, y_i) \in D$ tenemos que:

$$\begin{cases} \hat{w}^T x_i + \tilde{b} \geq \rho/2 & \text{si } y_i = 1 \\ \hat{w}^T x_i + \tilde{b} \leq -\rho/2 & \text{si } y_i = -1 \end{cases} \iff y_i(\hat{w}^T x_i + \tilde{b}) \geq \rho/2. \qquad (2.2)$$

Notar que para cada vector soporte, la Inecuación 2.2 es una ecuación. Podemos escalar convenientemente $\hat{w}$ y $\tilde{b}$, dividiendo miembro a miembro por $\rho/2$ en

[2] Fuente: https://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes_(SVG).svg

dicha ecuación. Esto es:

$$y_s(\hat{w}^T x_s + \tilde{b}) = \rho/2 \Rightarrow \frac{y_s(\hat{w}^T x_s + \tilde{b})}{\rho/2} = 1 \Rightarrow y_s(\overbrace{\frac{2\hat{w}^T}{\rho}}^{w^T} x_s + \overbrace{\frac{2\tilde{b}}{\rho}}^{b}) = 1$$

y luego la distancia entre cada vector soporte x_s y el hiperplano es:

$$r = \frac{y_s(w^T x_s + b)}{\|w\|} = \frac{1}{\|w\|}.$$

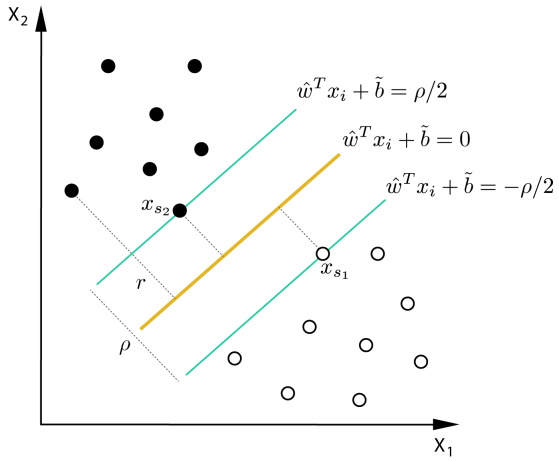


Figura 2.3: x_{s_1} y x_{s_2} son vectores soporte, ρ es el margen a maximizar y r es la distancia entre el hiperplano y un vector.

Finalmente, podemos expresar el margen en función de w como:

$$\rho = 2r = \frac{2}{\|w\|}$$

y entonces podemos pensar nuestro problema como uno de optimización. Queremos maximizar el margen ρ, sujetos a las restricciones que separan a nuestras clases. Equivalentemente, queremos encontrar w y b que resuelvan:

$$\begin{aligned} &\text{minimizar} && \|w\| \\ &\text{sujeto a} && y_i(w^T x_i + b) \geq 1 \quad i = 1, ..., n. \end{aligned} \tag{2.3}$$

Sin entrar en detalles, es interesante mencionar que para resolver este problema se debe construir el problema dual asociado y luego traducir las soluciones al espacio primal.

Quedan por ver ciertos aspectos que esta formulación original no considera, entre ellos:

1. Clases no separables linealmente.
2. Problemas multiclase.
3. Separadores no lineales.

En primer lugar, para los problemas donde las clases no son linealmente separables, [3] sugiere una modificación del problema de optimización original que agrega variables *slack* para relajar las restricciones, permitiendo casos mal etiquetados. Reformulando tenemos:

$$\begin{aligned} &\text{minimizar} \quad \|w\| + C\sum_{i=1}^{n}\epsilon_i \\ &\text{sujeto a} \quad y_i(w^T x_i + b) \geq 1 - \epsilon_i \quad i = 1, ..., n \end{aligned} \tag{2.4}$$

donde C es un parámetro que sirve para regular la importancia relativa entre maximizar el margen y ajustar los datos de entrenamiento.

En segundo lugar, para atacar el problema multiclase existen varias alternativas, entre las cuales las más usadas resuelven el problema de clasificación binaria entre:

- Una clase y todo el resto (*one-versus-all*), o
- Cada par de clases (*one-versus-one*).

El tercer problema que mencioné se resuelve mediante una proyección de los datos a algún espacio de mayor dimensionalidad donde las clases sí sean separables. Para esta tesina, sólo las dos primeras modificaciones al método original serán relevantes.

Redes Neuronales

Las *redes neuronales artificiales* [4] (*RNA*, de aquí en adelante) son una familia de modelos que provee un enfoque robusto para aproximar funciones a valores reales, discretos y vectoriales. Los primeros modelos datan de 1943 pero no tuvieron gran alcance recién hasta la aparición del algoritmo de *backpropagation* [26] en los años '80.

Las RNA están compuestas por unidades celulares llamadas *neuronas* o *perceptrones* (ver Figura 2.4). Cada neurona toma como entrada un vector a valores reales, computa una combinación lineal de estos valores y luego produce una salida, que es una función de dicha combinación. Esta salida representa el grado de activación de la neurona.

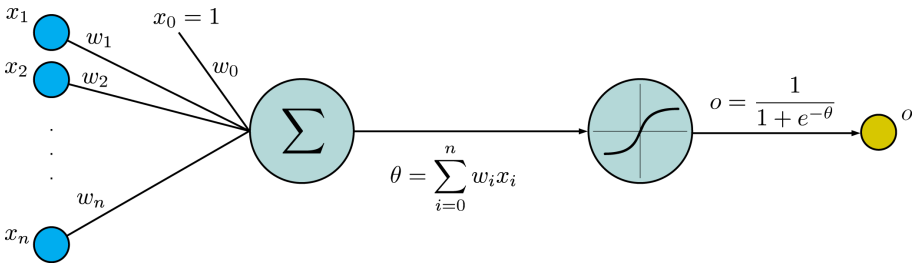


Figura 2.4: Neurona o perceptrón de una RNA. La información fluye de izquierda a derecha. La función de activación es la sigmoidea. $x_0 = 1$ no es la salida de ninguna neurona sino que sirve para entrenar un w_0 constante, llamado *bias*.

Formalmente, una neurona computa una función:

$$y = h(\sum_{i=0}^{n} w_i x_i)$$

donde h es una función de activación, usualmente alguna de las siguientes:

- $h(x) = \frac{1}{1+e^{-x}}$ (*sigmoidea*)
- $h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ (*tangente hiperbólica*)
- $h(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases}$ (*lineal rectificada*)

Una RNA, entonces, no es más que la composición de neuronas, dispuestas en capas, desde la entrada hacia la salida (ver Figura 2.5). Toda capa que no es la de entrada ni es la de salida se denomina *capa oculta*.

Entrenar una RNA involucra elegir un vector de pesos $\vec{w} = (w_0, w_1, ..., w_n)$ para cada neurona de la red, por lo que el espacio de hipótesis a explorar será:

$$\mathcal{H} = \{(\vec{w_1}, ..., \vec{w_m})\} \tag{2.5}$$

donde m es la cantidad de neuronas total de la red. Decimos que los vectores $\vec{w}$ de cada neurona son los *parámetros* que debemos ajustar durante el entrenamiento de una RNA.

Descenso por el gradiente

Para aprender los pesos, podemos pensar en una asignación inicial aleatoria y un método que iterativamente los actualice minimizando el error de entrenamiento, es decir, la diferencia entre la salida actual de la red y la esperada.

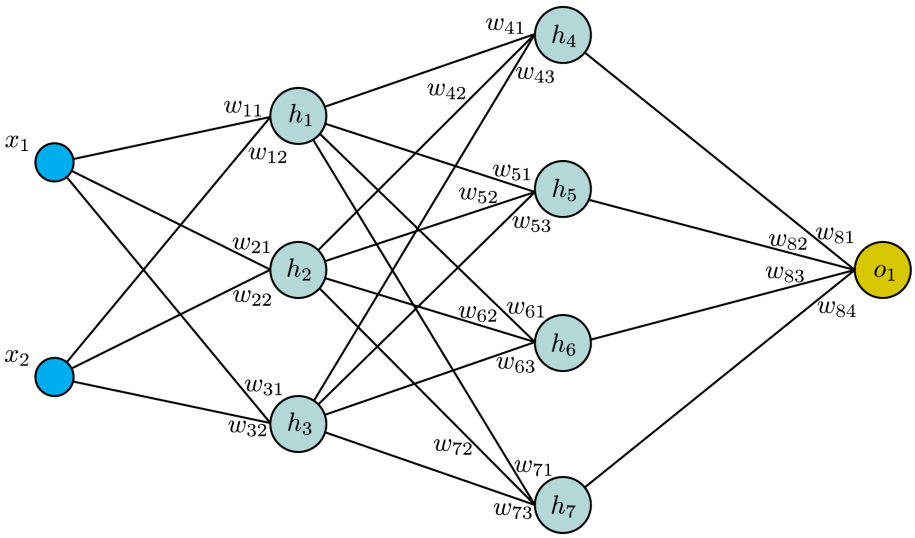


Figura 2.5: RNA *completamente conectada* o *full* con dos entradas, dos capas ocultas (cada una con tres y cuatro neuronas, respectivamente) y una única salida. Está implicito que cada neurona i posee además un peso w_{i0}, también conocido como *bias*, que no depende de ninguna neurona de la capa anterior.

Inicialmente, imaginemos que debemos entrenar una sola neurona cuya función de activación es la identidad, entonces la *salida* de la neurona no es más que:

$$o(\vec{x}, \vec{w}) = \vec{w}\vec{x}.$$

En términos generales, queremos entrenar un modelo que dé la mayor probabilidad $p(t|\vec{x})$ para cada tupla $(\vec{x}, t)$ en el conjunto D de entrenamiento. Si suponemos que las n observaciones del conjunto D son independientes, entonces queremos maximizar:

$$V(D) = \prod_{(\vec{x},t)\in D} p(t|\vec{x}).$$

Esta función es conocida como la función de *verosimilitud* o *likelihood*. Además, suponemos que para algún valor $\vec{x}$, su valor t asociado proviene de una distribución gaussiana con media $o(\vec{x}, \vec{w})$ y varianza β. Entonces:

$$V(D) = \prod_{(\vec{x},t)\in D} \mathcal{N}(t|o(\vec{x}, \vec{w}), \beta).$$

Además, sabemos que maximizar la verosimilitud es idéntico a maximizar su logaritmo natural, ya que ln es una función monótona creciente. Esto es:

$$\ln V(D) = -\frac{\beta}{2}\sum_{i=1}^{n}(t_i - o(\vec{x_i}))^2 + \frac{n}{2}\ln\beta - \frac{n}{2}\ln(2\pi). \tag{2.6}$$

donde los últimos dos términos son constantes dado que no dependen de $\vec{w}$. Luego, encontrar un máximo para la verosimilitud es equivalente a minimizar la función de error:

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \tag{2.7}$$

donde $o_d = o(\vec{x_d}, \vec{w})$ es la salida que produce el perceptrón dada la entrada $\vec{x_d}$ y t_d es el valor objetivo para $\vec{x_d}$.

La función E define una superficie continua que nosotros queremos explorar para encontrar el mínimo. Para lograr esto podemos derivar E respecto de cada componente de $\vec{w}$, obteniendo:

$$\nabla E(\vec{w}) = (\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}). \tag{2.8}$$

$\nabla E(\vec{w})$ es él mismo un vector, llamado *gradiente* que apunta en la dirección de mayor incremento del error. Por ende, el vector con dirección opuesta $-\nabla E(\vec{w})$ indica hacia dónde el error decrece más empinadamente. Entonces, podemos ir actualizando los pesos, descendiendo paso a paso por esa pendiente, calculando en cada paso:

$$w_i \leftarrow w_i - \eta\frac{\partial E}{\partial w_i}$$

para cada w_i, donde η es la *tasa de aprendizaje* o *learning rate*[3]. Calcular este gradiente es sencillo. Esto es:

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\ &= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}.\vec{x_d}) \\ &= \sum_{d\in D}(t_d - o_d)(-x_{id})\end{aligned} \tag{2.9}$$

donde x_{id} es la componente i del ejemplo d. Finalmente, el algoritmo se reduce a repetir el cálculo de:

$$\Delta w_i = \eta\sum_{d\in D}(t_d - o_d)x_{id}$$

[3]El learning rate es el hiperparámetro más importante de las RNA y en general, fijar su valor tiene una importancia crítica en el aprendizaje.

y luego actualizar los pesos:

$$w_i \leftarrow w_i + \Delta w_i$$

hasta que se cumpla una cierta condición de terminación. Está probado que este algoritmo, para neuronas lineales (activación identidad) o con una función de activación derivable, siempre converge al mínimo global en un número finito de pasos [4].

Descenso por el gradiente estocástico

El algoritmo anterior tiene un problema muy importante que es la velocidad de convergencia. Allí, los pesos se actualizan una vez por iteración y para ello es necesario propagar a través de la neurona cada uno de los ejemplos de entrenamiento. Para subsanar este problema, se propone actualizar incrementalmente los pesos a medida que se evalúa cada caso de entrenamiento. En pseudocódigo es:

- Inicializar cada valor w_i de $\vec{w}$ en algún número aleatorio pequeño
- Hasta que se cumpla la condición de terminación: (1)
  - Por cada $(\vec{x}, t) \in D$: (2)
    - Se pasa como entrada $\vec{x}$ a la neurona y se obtiene o
    - Se actualizan todos las componentes w_i de $\vec{w}$:

$$w_i \leftarrow w_i + \eta(t - o)x_i \quad (3)$$

donde cada iteración del bucle (1) se conoce como *época de entrenamiento*.

Esta versión *estocástica* o *incremental* mostró ser mucho más rápida que la anterior [4]. Es importante remarcar que su convergencia no depende de la separabilidad de las clases.

Un posible retoque que mejora aún más la velocidad de convergencia, es la utilización de *mini-batches* para la actualización de los pesos. Esto es, en vez de recorrer cada ejemplo de D y por cada uno recomputar los pesos, dividimos a D en lotes de cardinalidad k y luego actualizamos los pesos una sola vez por lote. Esto se logra reemplazando (2) por:

$$\text{Por cada } l = \{(\vec{x}^{(1)}, t^{(1)}), \ldots, (\vec{x}^{(k)}, t^{(k)})\} \in L$$

y (3) por:

$$w_i \leftarrow w_i + \frac{\eta}{k} \sum_{j=1}^{k} (t^{(j)} - o^{(j)}) x_i^{(j)}$$

donde L es una partición de D.

Backpropagation

El descenso por el gradiente estocástico es extendido por el algoritmo de *backpropagation* [26] para RNA de tamaño arbitrario, cuyas neuronas cuenten con alguna función de activación *derivable*. En cada época de entrenamiento, backpropagation hará una propagación ordinaria a través de la red y luego actualizará los pesos de cada neurona, desde la salida hasta la entrada, un paso en la dirección del gradiente de la función de error asociada. Una ventaja del algoritmo de backpropagation es que puede ser aplicado a otras funciones de error, que no sean la simple suma de cuadrados (vista en la Ecuación 2.7), en problemas de clasificación y regresión.

Capa Softmax

Para resolver problemas multiclase con redes neuronales es necesario producir una salida que indique la probabilidad de cada clase para la entrada suministrada. Para esta tarea, se elige típicamente una capa *Softmax*[4] [8] que cuenta con una neurona de salida por cada clase, y se cumple siempre que para cualquier entrada x, la suma de la salida de cada neurona de la Softmax es igual a 1. Formalmente, la probabilidad de la clase j dada la entrada x es:

$$P(y = j \mid \vec{x}) = \frac{e^{\vec{w_j}\,\vec{z}}}{\sum_{i=1}^{n} e^{\vec{w_i}\,\vec{z}}} \tag{2.10}$$

donde n es la cantidad de clases, $\vec{w_i}$ es el vector de pesos de la neurona de salida i y $\vec{z}$ es la propagación de $\vec{x}$ hasta la penúltima capa.

Momentum

Cabe destacar que la superficie de error para redes neuronales de tamaño arbitrario puede ser no-convexa y contar entonces con varios mínimos locales donde el aprendizaje se puede quedar atrapado. Para solventar este problema, el algoritmo introduce un nuevo hiperparámetro que se debe fijar llamado *momentum* [4]. Intuitivamente, podemos pensar que lanzamos una bola en la superficie de error. Esa bola se mueve a una velocidad igual a la tasa de

[4] La capa Softmax es una aplicación para RNA de la función Softmax, que a su vez es una generalización de la función logística.

aprendizaje por el módulo del gradiente y en cada instante cuenta con una inercia o momento lineal asociado. Un pequeño pozo en la superficie no la atrapará porque la inercia le hará continuar su movimiento. Formalmente, corregiremos el cómputo de los pesos visto en la referencia (3) del algoritmo de descenso por el gradiente agregando un término que depende de la modificación de los pesos en la iteración anterior, de la siguiente manera:

$$\Delta w_i(n) = \eta(t - o)x_i + \alpha \Delta w_i(n - 1) \tag{2.11}$$

para la i-ésima componente de entrada de la j-ésima neurona en la n-ésima iteración.

Weight Decay

Otro problema muy común en las redes neuronales con activaciones sigmoideas (en general, cualquier función con asíntotas horizontales) es el de *saturación* de las neuronas. Esto es cuando el valor absoluto de $\theta = \sum_{i=0}^{n} w_i x_i$ tiende a producir siempre la misma activación de la neurona, independientemente de la entrada x. La saturación se debe al sobreajuste de los pesos en las iteraciones tardías. Si consideramos que los pesos son inicializados en un valor aleatorio pequeño, entonces valores casi idénticos de pesos describirán superficies de decisión suaves. A medida que el entrenamiento continúa, algunos pesos tienden a crecer para reducir el error en los datos de entrenamiento, y la complejidad de la superficie de decisión crece. Eventualmente, luego de muchas épocas, backpropagation aprenderá superficies por demás de complejas para ajustar el ruido en los datos, o incluso particularidades irrelevantes del muestreo de entrenamiento. Una forma de abordar este problema consiste en controlar el crecimiento de los pesos, decrementándolos por un factor pequeño en cada iteración. Esto es equivalente a cambiar la definición de E incluyendo un término de penalización con la suma de todos los pesos de la red. Esto sesgará la red hacia las superficies de decisión menos complejas. Este término puede estar definido como la suma del cuadrado de los pesos o una suma del valor absoluto de ellos. Estas dos variantes son conocidas como *términos de regularización* $\mathcal{L}_2$ y $\mathcal{L}_1$. Esto es:

$$\mathcal{L}_2 = \gamma \sum_{i,j} w_{ij}^2 \tag{2.12}$$

y

$$\mathcal{L}_1 = \gamma \sum_{i,j} |w_{ij}| \tag{2.13}$$

donde el coeficiente de penalización γ se popularizó en la literatura con el nombre de *Weight Decay* [4].

Dropout

Dropout es una técnica de regularización, como Weight Decay, que evita el sobreajuste proveyendo un mecanismo equivalente a combinar grandes cantidades de redes neuronales de manera eficiente. Fue introducido por *Srivastava et al* en [21] y la idea es bastante simple: tirar aleatoriamente algunas neuronas de una o varias capas durante el entrenamiento. Esto es, en cada época de entrenamiento se eliminan al azar algunas unidades neuronales y con ellas sus entradas y salidas. Luego, para la generalización se usa la red completa.

En el caso más simple, cada neurona es conservada en una iteración con una probabilidad p. Entonces, aplicar dropout significa entrenar un conjunto de muchas redes más delgadas que la original y luego combinarlas para predecir.

Esta técnica resultó ser muy conveniente en redes neuronales de gran tamaño que requieren de muchos datos para ajustar los parámetros.

Arquitecturas profundas

En las redes neuronales introducidas en la sección anterior, el término *profundidad* hace referencia a la cantidad de capas ocultas presentes en el modelo. Mientras más capas, más profunda diremos que es la red. Las RNA de una o dos capas ocultas y otras arquitecturas poco profundas, tienen problemas para representar funciones complejas. Se dice que el poder de representación crece exponencialmente con la cantidad de capas (podríamos pensar que crece linealmente con la cantidad de combinaciones de parámetros de la red), que a su vez trae aparejado un crecimiento en la complejidad de búsqueda en dicho espacio. Es por esto que no resultaron útiles hasta que la ciencia dio con una mejor manera de entrenarlas. En 2006, *Hinton et al* [9] propusieron un método para entrenar *Deep Belief Networks*, un tipo de redes neuronales, que incluía un pre-entrenamiento *greedy* de los parámetros de las capas ocultas utilizando tan solo datos no clasificados. Esta inicialización de los pesos resultó funcionar mucho mejor que la aleatoria porque así fue posible extraer información *a priori* de los patrones presentes en los datos de

entrada (no etiquetados), y luego desde ese punto, realizar el ajuste fino de la red utilizando datos etiquetados.

Con estos modelos se abrió toda una nueva área de estudio dentro del Aprendizaje Automatizado, conocida como *Deep Learning*. Las redes profundas han mostrado gran efectividad resolviendo problemas sobre todo del área de *Machine Vision*: reconocimiento de imágenes [7], [9], secuencias de video [11] y datos de captura de movimiento [14], [15].

Autoencoders

Un *Autoencoder* es simplemente un método de Aprendizaje No Supervisado conformado por una RNA (usualmente con una única capa oculta) entrenado para reconstruir su propia entrada. Un Autoencoder sin ninguna otra restricción y con una gran capacidad (p. ej., con una capa oculta con más neuronas que la capa de entrada) podría aprender la representación trivial. Un Autoencoder ordinario debe entonces forzar que las computaciones fluyan a través de un cuello de botella para evitar que su representación interna sea el mapeo identidad (ver Figura 2.6).

Entrada
1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1

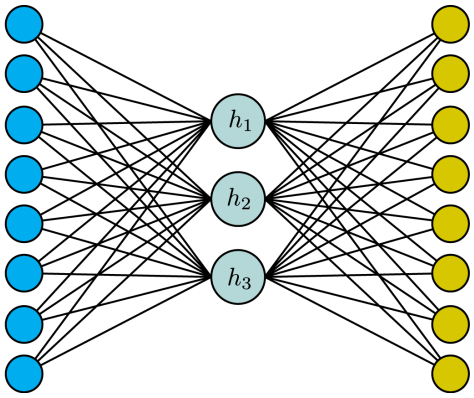


Figura 2.6: Ejemplo de un Autoencoder ordinario. La entrada consta de ocho ejemplos, cada uno con un único bit activado. Esta red intentará codificar en su capa oculta la representación binaria de los números entre 1 y 8.

El *Denoising Autoencoder* (*Vincent et al., 2008* [12]) es una variante estocástica de la versión original con una propiedad distintiva: evita aprender la función identidad aún cuando el modelo tiene capacidad alta. Un Autoencoder de este tipo se entrena intentando quitar el ruido de una versión corrupta de su entrada. Se ha probado en un arreglo de datasets que su performance es significativamente mejor que la de un Autoencoder ordinario [12].

Este modelo puede ser usado para inicializar los parámetros de las capas ocultas de una RNA. Para esto, se entrenará un Autoencoder por cada capa oculta de la red, donde el i-ésimo Autoencoder intentará reconstruir su entrada, que es la salida de la capa $i-1$, para la cual sus parámetros ya fueron inicializados por el Autoencoder $(i-1)$-ésimo. Esta técnica de *Stacked Denoising Autoencoders* o *SDAE* sirve para incializar los pesos de cualquier RNA tradicional y será utilizada en el Capítulo 4.

Redes Convolucionales

Las *redes neuronales convolucionales* (*RNC*, de aquí en adelante) fueron desarrolladas originalmente por *Yann LeCun et al* a fines de los '80 [1] y olvidadas algún tiempo por la comunidad de *Machine Vision* porque no se creyó que escalarían a imágenes del mundo real. El *statu quo* permaneció hasta el 2012, año en que *Krizhevsky et al* [16] ganan la competencia *2012 ImageNet Challenge* por amplio margen: (15.3 % de error de generalización del top-5[5] contra 26.2 % del segundo mejor). Este hecho en particular revivió el interés en las redes convolucionales, y durante los años subsiguientes (hasta la actualidad) se presentaron numerosos artículos al respecto (p. ej., [16], [17], [20]) y son hoy en día el foco de atención principal de la comunidad de *Machine Vision*.

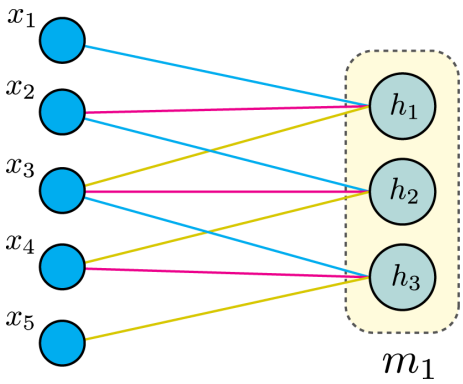


Figura 2.7: Capa convolucional con m_1 como único *feature map*. Cada neurona en m_1 se conecta con tan solo tres neuronas de entrada. Las conexiones del mismo color comparten el peso w_i.

Una RNC es una red neuronal con pesos compartidos y conexiones locales. Como se ilustra en la Figura 2.7, un *feature map* o *mapa* es un conjunto de

[5] Un ejemplo se considera bien clasificado durante la generalización top-5 si su clase real está dentro de las cinco más probables para el modelo.

neuronas ocultas cuyas conexiones se limitan a una porción de la entrada y cuyos vectores de pesos $\vec{w} = (w_1, w_2, ..., w_n)$ son idénticos. Cada capa convolucional cuenta con uno o varios mapas. Típicamente, las primeras capas cuentan con una gran cantidad de mapas que servirán para representar *características* o *features* de bajo nivel, y las sucesivas capas superiores aprenderán conceptos más refinados y abstractos, producto de la combinación de los mapas de los niveles anteriores. Es importante observar que las conexiones locales limitan el rango de visión de una neurona dada. Por ejemplo, en la Figura 2.7, h_1 sólo depende de x_1, x_2 y x_3. Por lo tanto, las primeras capas tendrán una visión localizada, mientras que las capas superiores desarrollarán una visión más global del espacio de entrada.

RNC en imágenes

Imagine ahora que el espacio de entrada $x_1, ..., x_n$ está formado por los n píxeles que componen una imagen, los cuales están acotados en un rango que depende de la representación de colores. Los mapas aprendidos por la red funcionarán como *filtros* de ciertas características relevantes en las imágenes, como ser aristas y esquinas en mapas de capas inferiores, y formas complejas en capas superiores. Es decir, detectarán características interesantes para resolver el problema de aprendizaje. Para visualizar estos filtros, podemos graficar la imagen (del tamaño del filtro) que maximice su activación. Piense, por ejemplo, en alguna neurona del mapa m_1 de la Figura 2.7 (no importa cual, todas comparten los mismos pesos). Encontrar la imagen que maximice su activación es posible, porque es simplemente encontrar un vector $\vec{x} = (x_1, \ldots, x_n)$ tal que la sumatoria $\sum_{i=0}^{n} w_i x_i$ sea máxima, lo cual se traduce en elegir el mínimo x_i posible para las componentes negativas de $\vec{w}$ y el máximo x_i posible para las componentes positivas de $\vec{w}$. Esto se puede hacer en general para cualquier mapa de cualquier capa, sin embargo el vector $\vec{x}$ tiene una representación gráfica con sentido solo cuando este es la imagen de entrada. Es decir, los filtros que valen la pena visualizar (al menos de esta manera) son los pertenecientes a la primera capa. Un ejemplo de esto se puede ver más adelante en la Figura 3.3.

Las RNC son especialmente útiles en la tarea de reconocer patrones en imágenes por varias razones, entre ellas:

- Invarianza traslacional: Cada neurona dentro de un mapa comparte el vector de pesos pero se enfoca en un lugar distinto de la entrada. Esto permite que las neuronas aprendan un feature sin importar la posición que este ocupe en el campo visual de entrada. Se dice que las conexiones

locales y los pesos compartidos explotan la propiedad *estacionaria* de las imágenes naturales.

- EFICIENCIA TEMPORAL: Las RNC tienen una eficiencia superior a las RNA tradicionales durante el entrenamiento, dado que reducen ampliamente la cantidad de parámetros a aprender. Los vectores $\vec{w}$ tienen muchísimas menos componentes debido a las conexiones locales y además son compartidos por todas las neuronas de un mismo mapa. Este factor es clave a la hora de trabajar con imágenes dado que ciertos conceptos que resultan de interés para los humanos son solo visibles a partir de cierta resolución.

Podemos distinguir entre dos tipos de capas utilizadas en redes convolucionales

- CAPAS DE CONVOLUCIÓN: Se encargan de propagar, a través de toda la entrada, el filtro (tambien conocido como *ventana* o *parche* de convolución) antes mencionado. Este filtro se desplazará con un paso determinado generando así un mapa donde cada neurona representa el grado de activación de un mismo feature en distintas regiones de la entrada. Por ejemplo, en la Figura 2.8, la capa convolucional propaga una ventana de 5×5 con un paso de 1×1. Es decir, la ventana se desplazará 1 unidad en cada iteración en horizontal, y cada vez que cambie de fila lo hará desplazándose también 1 unidad. El tamaño final de la capa puede calcularse de la siguiente manera:

  Sea $i = (i_1, i_2) \in \mathbb{N}^2$ la dimensión de la imagen de entrada, $d = (d_1, d_2) \in \mathbb{N}^2$ la dimensión de la ventana y $s = (s_1, s_2) \in \mathbb{N}^2$ el paso. Entonces la capa convolucional tendrá una dimensión:

  $$o = (\frac{i_1 - d_1}{s_1} + 1, \frac{i_2 - d_2}{s_2} + 1). \tag{2.14}$$

  En el caso de la Figura 2.8, $o = (\frac{14-10}{1} + 1, \frac{14-10}{1} + 1) = (10, 10)$ es la dimensión de m_2.

- CAPAS DE POOLING: Se encargan de reducir progresivamente el tamaño espacial de la representación para decrementar la cantidad de parámetros de la red y las computaciones asociadas, además de ayudar a controlar el sobreajuste. Usualmente, luego de cada capa convolucional se coloca una capa de pooling que realiza una operación simple para reducir la dimensionalidad. La manera es similar a la ventana corrediza de la convolución, sólo que en cada una de esas

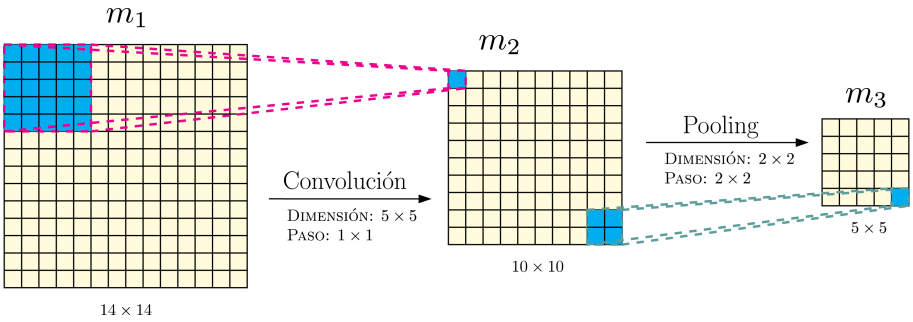


Figura 2.8: m_1 es la imagen de entrada, m_2 es una capa convolucional y m_3 es una capa de pooling.

ventanas se computa el **máximo** o el **promedio**. Generalmente, la operación que se lleva a cabo es el **máximo**, y particularmente en esta tesina es la que usaremos en todas las capas de pooling.

Cabe señalar que la Ecuación 2.14 sirve también para el tamaño de las capas de pooling dado que este depende únicamente de i, d y s, siendo irrelevante el cómputo que realice la capa.

Capítulo 3

Datasets y arquitecturas

Este capítulo sirve de preparación preliminar para los experimentos del Capítulo 4. Aquí se mostrarán en detalle los datos y modelos específicos que servirán de base sobre la cual se construirán los experimentos.

3.1. Datasets

Para la realización de esta tesina se utilizaron datos provenientes de dos datasets que se detallan a continuación.

ImageNet

ImageNet [13] es un dataset de imágenes, organizado de acuerdo a *WordNet* [2], una base de datos léxica que cuenta con conjuntos de sinónimos. ImageNet provee un compendio de imágenes tomadas de internet para un gran número de grupos de sinónimos. Hoy en día cuenta con 14 197 122 imágenes provenientes de 21 841 grupos de sinónimos distintos.

Las imágenes etiquetadas son, en cambio, una cantidad menor: 1,2 millones de imágenes pertenecientes a 1000 clases distintas (ver Figura 3.1). Cada año, desde el 2010, ImageNet organiza una competencia mundial llamada *ILSVRC* (*ImageNet Large Scale Visual Recognition Contest*) donde participan investigadores y aficionados de todo el planeta con el objetivo de obtener el mejor resultado clasificando el conjunto de test de ImageNet.

[1] Estas imágenes pertenecen específicamente al dataset *STL-10*, que es un subconjunto de ImageNet: `http://cs.stanford.edu/~acoates/stl10`

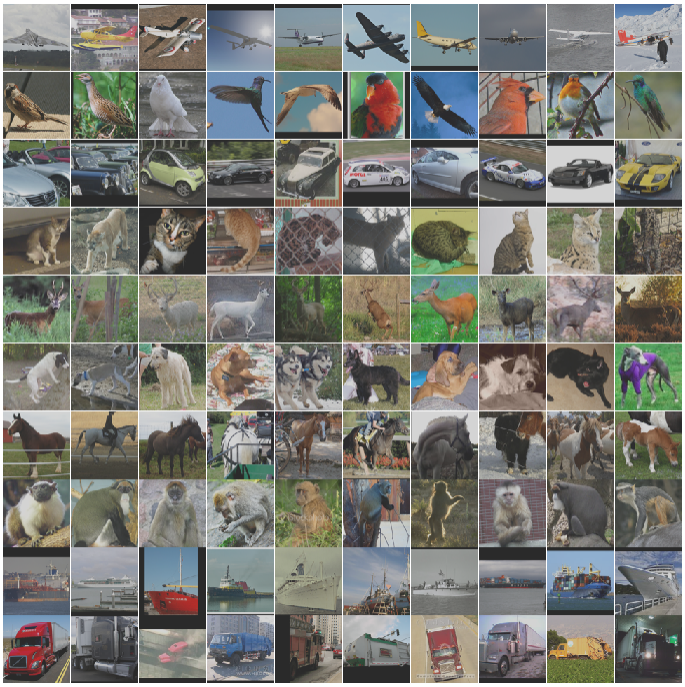


Figura 3.1: Ejemplo de diez clases de ImageNet[1]. Cada fila exhibe imágenes de una de las clases.

Rugby

El grupo de investigación de *Aprendizaje Automatizado y Aplicaciones* del *CIFASIS-CONICET* cuenta con un conjunto importante de videos de rugby, provenientes de cámaras personales y grabaciones televisivas. De todo este conjunto de videos intentaremos aprender tres clases que representan acciones grupales reconocibles fácilmente por el ojo humano. Estas acciones son[2] (ver Figura 3.2):

- Scrum: El scrum o la melé, una de las formaciones más reconocibles del rugby, es una puja frente a frente, de un grupo de cada equipo formado por un máximo de ocho y un mínimo de cinco jugadores en tres líneas, que se enfrentan agazapados y asidos entre sí, para comenzar a empujar con el fin de obtener el balón que ha sido lanzado en medio de ellos y sin tocarlo con la mano.

- Line: Cuando el balón, o el jugador que lo lleva, salen del campo por la línea de touch (línea lateral), el juego se reinicia mediante un saque

[2]Definiciones simplificadas tomadas de https://es.wikipedia.org/wiki/Rugby

de banda llamado line out (o simplemente line) que debe arrojarse recto y superando la línea ubicada a cinco metros campo adentro del touch entre dos hileras de jugadores, una de cada equipo y separadas por una distancia de un metro. Los jugadores deben saltar para obtener la pelota, pudiendo ser impulsados y sostenidos por sus compañeros.

- Juego: En esta clase se aglomeran todas las otras situaciones de juego, típicamente el avance continuo de un equipo, aunque también puede incluír penales y momentos donde el árbitro detiene el juego para sancionar a algún jugador.

El problema resulta de vital importancia para los entrenadores de rugby, dado que actualmente deben separar estas clases manualmente para luego utilizarlas durante los entrenamientos tácticos en pos de corregir defectos de una formación en particular. De esta manera, el entrenador puede, por ejemplo, mostrar todos los lines, uno tras otro, sin tener que revisar todo el video del partido.

Intrínsecamente, este problema requiere de un modelo robusto a cambios irrelevantes (p. ej., colores de las camisetas, tipo de campo de juego, presencia de tribunas, etc.). Sin embargo, en estos videos aparece una alta gama de variaciones escénicas: diferentes ángulos, intensidad de luz, sombras, cortes de cámara y zoom. Algunas filmaciones personales incluso introducen obstáculos entre la cámara y la acción grupal (ver Figura 3.2). Todas estas variaciones en el video representan dificultades adicionales para aprender el problema.

Del gran conjunto de videos de rugby, se toman algunos, se dividen en pequeños videoclips de tres segundos y se los separa en dos datasets, que mostraremos a continuación. Cada videoclip exhibe imágenes de una única clase. Es decir, en ningún videoclip aparecen las transiciones entre diferentes acciones (p. ej., de *line* a *juego*, o de *scrum* a *juego*), lo que facilita el proceso de etiquetarlos, y no da lugar a confusiones. Todos los videos comparten las siguientes especificaciones:

- Dimensiones: 232x232 píxeles
- Duración de cada clip: 3 segundos
- FPS: 15
- Audio: No

main

El dataset principal (*main*, de aquí en adelante) tiene las siguientes características:

Figura 3.2: Imágenes de ejemplo, extraídas de videos del dataset principal (*main*). Las etiquetas, por filas de arriba hacia abajo, son: *scrum, scrum, line, line, juego, juego.*

.

- Videoclips por clase: 80
- Cantidad de partidos: 10

Este fue el dataset creado originalmente, con el cual se entrenaron y testearon todos los modelos de la tesina. Algunas imágenes de este conjunto se exhiben en la Figura 3.2.

oneMatch

Para poder dar una mejor estimación del error de generalización se separó un partido más, nunca visto durante el entrenamiento. Esto servirá para evaluar el sobreajuste y tener una medición confiable del error de generalización. Este dataset se compone de:

- Videoclips por clase: 10
- Cantidad de partidos: 1

3.2. Arquitecturas

En esta sección veremos detalladamente las arquitecturas que se usaron durante la investigación.

Overfeat

Overfeat es un framework integral que permite usar redes convolucionales profundas para las tareas de clasificación, localización y detección [17]. Nuestro interés está puesto en la red para clasificación, cuya arquitectura es similar a la ganadora del *ILSVRC12* presentada por *Krizhevsky et al* en [16]. El framework cuenta con dos versiones de dicha arquitectura, un modelo *rápido* y otro *preciso*. En esta tesina se utilizó, por cuestiones de performance, únicamente el modelo rápido, que de aquí en adelante llamaremos simplemente Overfeat.

Capa	1	2	3	4	5	6	7	8
Dimensión de entrada	231x231	24x24	12x12	12x12	12x12	6x6	1x1	1x1
Procesos	conv + max	conv + max	conv	conv	conv + max	full	full	full
# mapas	96	256	512	1024	1024	3072	4096	1000
Tamaño de filtro	11x11	5x5	3x3	3x3	3x3	-	-	-
Paso de convolución	4x4	1x1	1x1	1x1	1x1	-	-	-
Tamaño de pooling	2x2	2x2	-	-	2x2	-	-	-
Paso de pooling	2x2	2x2	-	-	2x2	-	-	-

Tabla 3.1: Arquitectura de Overfeat. Las líneas dobles verticales separan la región convolucional de la región full.

En la Tabla 3.1 se puede ver la arquitectura de Overfeat. Cada columna representa una capa del modelo. Las capas de la 1 a la 5 son capas convolucionales con unidades con activaciones lineales rectificadas (*rectified linear unit* o *relu*). Algunas de ellas cuentan con un max-pooling luego de la convolución. Las capas 6, 7 y 8 están completamente conectadas, siendo la última capa una Softmax, útil para la clasificación.

El modelo fue entrenado con ImageNet (ver Sección 3.1). Cada imagen del dataset fue escalada hasta que su dimensión más pequeña fuera 256, y luego se extrajeron cinco recortes aleatorios de 231×231 y se presentaron a la red junto a sus espejados horizontalmente en mini-batches de 128. Los pesos se inicializaron aleatoriamente tomados de una distribución normal con $(\mu, \sigma) = (0, 1 \times 10^{-2})$. Luego se actualizaron utilizando el algoritmo de descenso por el gradiente estocástico (explicado en la Sección 2.2), acompañado por un término de momentum de 0,6 y un término de weight decay $\mathcal{L}_2$ de 1×10^{-5}. El

learning rate fue inicialmente 5×10^{-2} y posteriormente fue decrementando a un factor de 0,5 luego de $(30, 50, 60, 70, 80)$ épocas de entrenamiento. Dropout fue usado en las capas completamente conectadas (capas 6 y 7 del clasificador).

El error de generalización top-5 de Overfeat en el conjunto de testing de ImageNet es de 16,39 %. Su versión más robusta obtuvo un resultado un poco mejor (14,18 %), aunque requiere de casi el doble de conexiones.

Overfeat será de vital importancia para componer modelos en una forma especial de aprendizaje, llamado *Transfer Learning*. La idea es aprovecharse de los features aprendidos por Overfeat en ImageNet, que hasta cierta capa serán características relevantes en todo tipo de imágenes naturales, y de allí en adelante entrenar una red propia para distinguir las clases de Rugby. Concretamente, propagaremos nuestro dataset a través de Overfeat hasta cierta profundidad, y luego almacenaremos esas propagaciones como un conjunto de datos intermedio que servirá para entrenar y validar otros modelos.

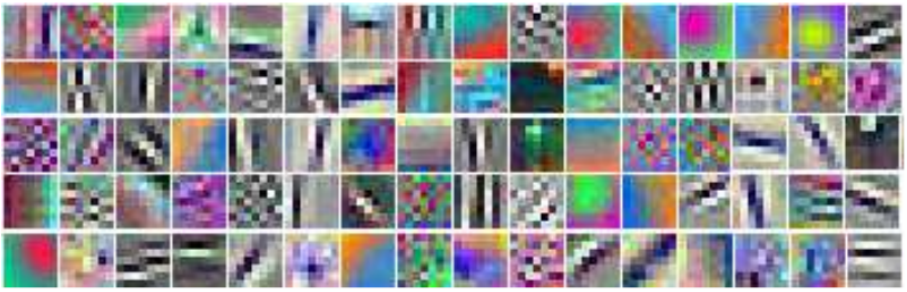


Figura 3.3: Los 96 filtros entrenados de la primera capa convolucional de Overfeat.

Convnet

Convnet es el nombre corto que se le suele dar a las redes convolucionales. De aquí en adelante, usaremos este término para hacer referencia a la arquitectura convolucional profunda propuesta por este trabajo (ver Tabla 3.2).

Esta arquitectura cuenta con cuatro capas convolucionales, de las cuales las primeras tres están seguidas de un max-pooling, y luego dos capas full, de las cuales la última es una Softmax. La cantidad de mapas en la primera capa fue determinante para que el entrenamiento y la búsqueda en el espacio de hiperparámetros fuera posible. Este modelo, al igual que todos los otros que veremos en esta tesina, fue entrenado utilizando el framework *pylearn2* que compila código *CUDA* y ejecutado sobre una GPU *nVidia GeForce GTX*

Capa	1	2	3	4	5	6
Dimensión de la entrada	128x128	31x31	13x13	5x5	2x2	1x1
Procesos	conv + max	conv + max	conv + max	conv	full	full
# mapas	16	32	64	128	256	3
Tamaño de filtro	7x7	5x5	3x3	4x4	-	-
Paso de convolución	2x2	1x1	1x1	1x1	-	-
Tamaño de pooling	2x2	2x2	2x2	-	-	-
Paso de pooling	2x2	2x2	2x2	-	-	-

Tabla 3.2: Arquitectura de Convnet. Las líneas dobles verticales separan la región convolucional de la región full.

770[3]. Con ese poder de cómputo, pasar de 16 a 32 mapas dio un salto en el tiempo consumido por una única época de entrenamiento de **2m30s** a **45m15s**. Esta demanda restringió en buena medida el porte de la red convolucional (comparar con la arquitectura de Overfeat en la Tabla 3.1). Otra cota importante fue el tamaño de entrada de las imágenes. El *trade-off* entre eficiencia en la clasificación y eficiencia temporal del entrenamiento se explicará en detalle en el Capítulo 4.

[3]1536 núcleos de 1.046Ghz, 2GB de RAM.

Capítulo 4

Experimentos

En este capítulo condensaremos todo el trabajo experimental de la tesina. Particularmente, analizaremos si disponiendo únicamente de un conjunto de videos relativamente pequeño es posible obtener resultados satisfactorios. Para evaluar esta hipótesis de trabajo, en la Sección 4.1 utilizaremos los features propagados a través de Overfeat (opcionalmente agregando algunas capas full) para intentar aprender otros de más alto nivel, útiles al problema particular de rugby. Veremos allí, además, cómo será el entrenamiento de Convnet. Luego, en la Sección 4.2 se exibirán las técnicas que permiten una buena aproximación del error de generalización, y métodos alternativos de clasificación de video. En la Sección 4.3 se podrán comparar los resultados de los distintos modelos y finalmente, en la Sección 4.4 se echará un ojo a los conceptos aprendidos por las redes a través de distintas técnicas de visualización.

4.1. Entrenamiento

El enfoque que tendrá el entrenamiento de las redes será *naïve*, en tanto la clasificación de cada clip de video dependerá de la clasificación de cada frame individual que lo compone. Esto significa que, en principio, no será posible extraer información temporal de los videos y que contaremos únicamente con un canal de features espaciales (a diferencia del trabajo propuesto por *Wu et al* en [24] que además extrae características temporales a través de un flujo óptico generado entre cada par de frames). Entendemos que esto será posible dado que las acciones grupales que estamos interesados en identificar son reconocibles a partir de imágenes estáticas.

Entrenando modelos basados en Overfeat

Usaremos las siguientes abreviaciones para referirnos a los modelos (o a partes de ellos) vistos en la secciones 3.2 y 2.2:

- OF7: Overfeat hasta la capa 7.
- OF6: Overfeat hasta la capa 6.
- DNN: RNA con capas full e inicializaciones aleatorias.
- SDAE-DNN: RNA con capas full e inicializaciones mediante Stacked Denoising Autoencoders preentrenados usando datos no etiquetados.
- SM: Capa Softmax

Tanto DNN como SDAE-DNN contarán con una cantidad de capas ocultas y una cantidad de neuronas en las capas ocultas determinadas durante la búsqueda de hiperparámetros. En particular se probó con todas las posibles combinaciones entre:

- # Capas ocultas: 1, 2 o 3
- # Neuronas ocultas por capa: 500, 1000, 2000 o 4000

Dicho esto, los modelos compuestos que compararemos serán los siguientes:

- OF7 + SDAE-DNN + SM
- OF7 + DNN + SM
- OF7 + SM
- OF6 + SDAE-DNN + SM
- OF6 + DNN + SM
- OF6 + SM
- Convnet

donde la relación $A + B$ representa una conexión completa de la salida de A con la entrada de B.

Recuerde que Overfeat es una red ya entrenada, por lo que el entrenamiento de los modelos compuestos que involucren a OF6 u OF7 constarán de una etapa preliminar que simplemente propagará los datos de entrada a través de Overfeat, produciendo una salida que será la entrada para el submodelo restante.

Entrenando Convnet

Convnet fue entrenado usando únicamente el dataset *Rugby* descripto en la Sección 3.1. En primer lugar, se redujo la cantidad de fps de los videos a 5. Luego, se extrajeron los *frames* o *fotogramas* de cada video resultando en

3500 imágenes. Las imágenes fueron escaladas a un tamaño de 128×128 y más tarde propagadas a través de la red en mini-batches de 100. Los pesos se inicializaron aleatoriamente tomados de una distribución uniforme en el rango $(-5 \times 10^{-2}, 5 \times 10^{-2})$. Para computar los pesos se utilizó la técnica del descenso por el gradiente estocástico, con un término de momentum de 0,5 y un learning rate constante de 0,1. Dropout fue usado solamente en la capa 5 con $p = 0{,}5$.

Es importante mencionar que tanto los hiperparámetros más relevantes (learning rate y momentum) como la morfología de la arquitectura (cantidad de capas convolucionales, cantidad de mapas en capas convolucionales, cantidad de capas full y cantidad de neuronas en las capas full) fueron optimizados haciendo una búsqueda en ese espacio y evaluando resultados. Además, cabe destacar que conservar el tamaño original de las imágenes producía un modelo intratable, debido a la cantidad de conexiones, y reducirlo en demasía empeoraba la precisión, ya que los features relevantes para identificar las clases no eran reconocibles a partir de imágenes pequeñas. Por estas razones, se utilizaron imágenes de 128×128 con este modelo.

4.2. Generalización

Las mediciones que hagamos sobre el conjunto de generalización pueden estar sesgadas por la partición que se consideró. Para tener una medición más exacta del error (o la precisión) de generalización, independiente de la partición, usaremos la técnica de validación cruzada de k-folds. Esto es, dividir los datos en k conjuntos y luego usar $k-2$ para entrenar (*train*), uno para validar (*valid*) y otro para generalizar (*test*). Este proceso tiene k iteraciones, y en cada una se elige un nuevo fold para test, un fold al azar para valid y el resto para entrenar. Acto seguido, se entrena el modelo con esa partición y se evalúa la precisión de test allí. Finalmente, la precisión de generalización se calcula como la media de la precisión de test en cada iteración (μ). Esta media, tiene un error estadístico (llamado *error estándar* o *desvío estándar de la media*), que se calcula de la siguiente manera:

$$\sigma([a_1, a_2, \ldots, a_k]) = \frac{\sqrt{\sum_{i=1}^{k} (a_i - \mu)^2}}{\sqrt{k}} \tag{4.1}$$

donde a_i es la medición de precisión en el fold i-ésimo.

Los resultados, presentados en la Sección 4.3, mostrarán la media y el desvío estándar de la media con el siguiente formato:

$$\mu \pm \sigma. \tag{4.2}$$

Para estas mediciones hemos usado validación cruzada de 8-folds en los modelos compuestos a partir de Overfeat y de 5-folds en Convnet. La razón de esta elección está fuertemente ligada a la complejidad temporal asociada al entrenamiento de cada modelo. Overfeat es una red convolucional considerablemente más compleja que Convnet, sin embargo no tenemos que volver a entrenarla en cada fold. Este sí es el caso de Convnet, por lo que se decidió disminuir allí el número de folds para aligerar el cómputo.

Clasificación de video

Imagine que disponemos de la predicción de alguna red para un videoclip en particular. Esto es, las probabilidades p_0, p_1 y p_2, que corresponden a las clases *juego*, *line* y *scrum*, respectivamente. Consideraremos entonces algunas variantes para clasificar el video:

1. Votación simple

Se considera la clase predicha para cada frame como la clase con la probabildad más alta y luego se suman los votos de la predicción de cada frame. La clase más votada es la clase predicha para el video. Formalmente, suponga que el video v tiene n frames, y la probabilidad de que el frame i pertenezca a la clase j es p_{ji}. Entonces la predicción del video v será c_v:

$$c_v = g(S_0, S_1, S_2) \tag{4.3}$$

donde

$$S_k = \sum_{i=1}^{n} f(p_{0i}, p_{1i}, p_{2i}, k) \quad , \forall k = 0 \dots 2 \tag{4.4}$$

,

$$f(x, y, z, k) = \begin{cases} 1 & \text{si } g(x, y, z) = k \\ 0 & \text{si } g(x, y, z) \neq k \end{cases} \tag{4.5}$$

y

$$g(x, y, z) = \begin{cases} 0 & \text{si } x \geq y \wedge x \geq z \\ 1 & \text{si } y \geq x \wedge y \geq z \\ 2 & \text{sino} \end{cases} \tag{4.6}$$

2. Suma de probabilidades

Este método de clasificación de videos es similar a la votación simple, solo que reemplaza las componentes del vector U, que aparecen en la Ecuación 4.4 por:

$$U_k = \sum_{i=1}^{n} p_{ki} \quad , \forall k = 0 \dots 2 \tag{4.7}$$

Es decir, para cada clase, se suma la probabilidad de cada frame y la clase con la suma más alta gana. Este método es especialmente útil si sucede que las predicciones son muy certeras cuando son correctas (probabilidad alta de la clase real) y muy inciertas cuando son incorrectas (es decir, cuando gana la clase incorrecta, lo hace por un pequeño margen).

3. Series temporales

Por último, intentaremos incorporar información temporal a la predicción de un videoclip. Con tal fin, tomaremos las probabilidades de una clase en particular para cada frame como una función en el tiempo, como podemos ver en la Figura 4.1.

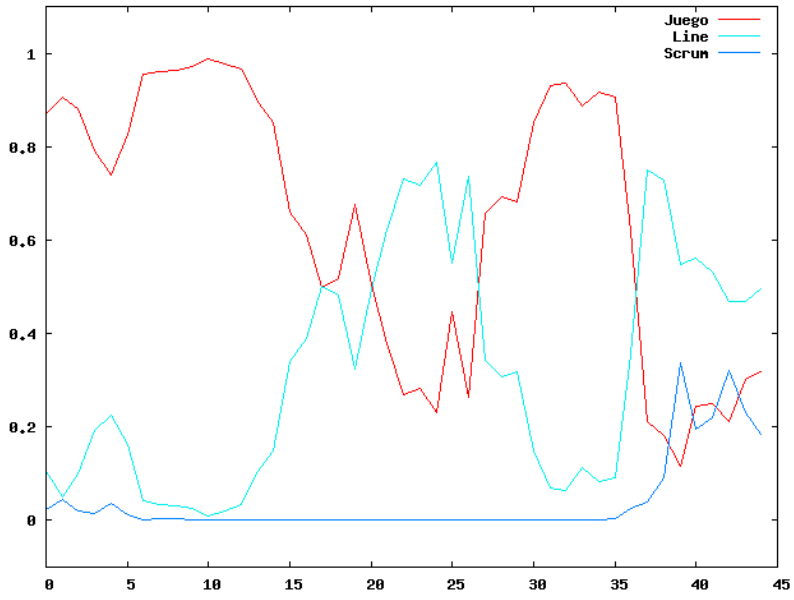


Figura 4.1: Ejemplo de series temporales producidas por CONVNET para un video cuya clase real es *juego*.

Posteriormente, se intentará predecir utilizando SVM sobre datos recolectados de las series, con alguno de los siguientes enfoques:

- VENTANA CORREDIZA + SVM: Por cada clase, se suman las probabilidades en una ventana de tiempo de tamaño fijo. Eso da una suma de probabilidades por clase por ventana. Moviendo la ventana de principio a fin, generamos tres nuevas curvas de probabilidad, notablemente más suavizadas (ver Figura 4.2).

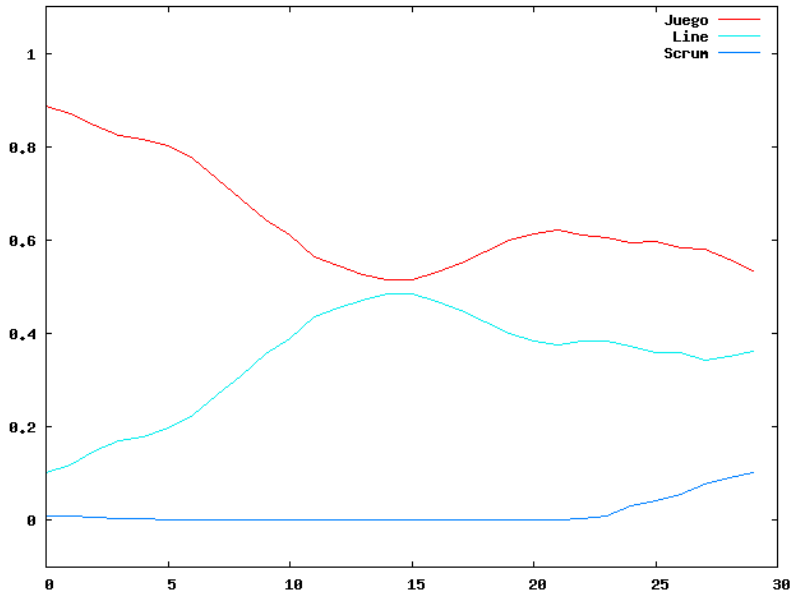


Figura 4.2: Versión procesada con ventana corrediza y luego normalizada de la serie de la Figura 4.1.

  Podemos pensar entonces a cada video procesado de esta manera como un vector de dimensión $k(f - w)$, donde k es la cantidad de clases, f la cantidad de frames y w el tamaño en frames de la ventana. Finalmente, se utiliza SVM con separadores lineales para clasificar dichos vectores.

- VENTANA CORREDIZA + DETECCIÓN DE EVENTOS: Suponiendo que nuestras nuevas series temporales son ahora aquellas suavizadas mediante el proceso de ventana corrediza, surge ahora una alternativa aún más sencilla de clasificación: la de detección de picos que sobrepasen cierto umbral. Podemos pensar que la creencia en una clase en particular tiene relevancia solo a partir de cierta probabilidad fija. Luego, este método no es más que contar la cantidad de frames que

superaron esos umbrales, para cada clase, y ver qué clase los superó en más oportunidades. De cualquier modo, la detección de eventos también precisa de una etapa de entrenamiento, donde se realiza una búsqueda que permite fijar valores para los umbrales de cada clase.

4.3. Resultados

Haciendo uso de las técnicas descriptas en las secciones anteriores para entrenar y validar nuestros modelos, obtuvimos los resultados que aparecen en la Tabla 4.1 para la versión *main* del dataset de rugby.

Arquitectura	**Precisión Frames** (%)	**Precisión Videos** (%)
OF7 + SDAE-DNN + SM	$81{,}4 \pm 2{,}1$	$84{,}2 \pm 2{,}1$
OF7 + DNN + SM	$81{,}5 \pm 1{,}6$	$86{,}7 \pm 2{,}1$
OF7 + SM	$81{,}4 \pm 1{,}9$	$84{,}2 \pm 2{,}2$
OF6 + SDAE-DNN + SM	$82{,}3 \pm 1{,}4$	$84{,}2 \pm 2{,}1$
OF6 + DNN + SM	$\mathbf{84{,}8} \pm 2{,}2$	$\mathbf{88{,}7} \pm 2{,}0$
OF6 + SM	$\mathbf{84{,}6} \pm 1{,}8$	$\mathbf{88{,}8} \pm 2{,}7$
Convnet	$72{,}7 \pm 1{,}7$	$76{,}5 \pm 2{,}2$

Tabla 4.1: Resultados para la versión *main* de rugby, clasificación los videos mediante votación simple.

Por un lado, es importante notar que la utilización de autoencoders para inicializar los pesos de las DNN encima de Overfeat, no resultó ser mejor que la inicialización aleatoria. Por otro lado, las propagaciones hasta la capa 6 arrojaron mejores resultados que aquellas hasta la capa 7. Entendemos que esto se debe a que los features de la capa 7 son más específicos de las clases de ImageNet y por lo tanto menos útiles para resolver el problema de rugby. Finalmente, la red convolucional entrenada únicamente con los datos de rugby (Convnet) no mostró buenos resultados, rindiendo peor que cualquier modelo basado en Overfeat. Los filtros que aparecen en la Figura 4.3 son claramente más difusos que aquellos de Overfeat. Algunos tienen ruido de alta frecuencia, otros tienen el color del césped, y el resto muestra un leve gradiente de color en algún ángulo (comparar con la Figura 3.3 donde las regiones con bordes tienen un alto contraste).

Seguidamente, se sospechó que mezclar videoclips de un mismo partido en train y test podría representar una ventaja resolviendo el problema, debido a la similitud de las escenas (campo de juego, camisetas, etc.), por lo que se decidió evaluar los modelos en la otra versión de rugby (*oneMatch*), que incluye un partido nunca visto durante el entrenamiento. Allí los resultados fueron los ahora exhibidos en la Tabla 4.2.

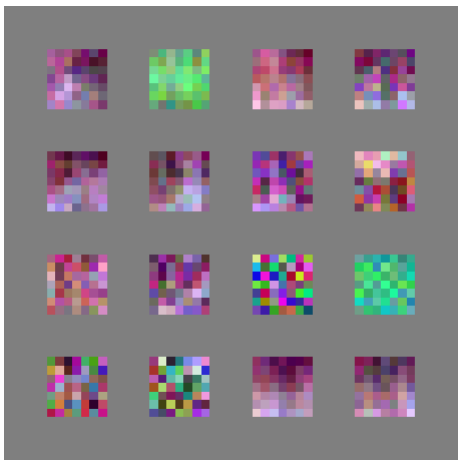


Figura 4.3: Los 16 filtros entrenados de la primera capa convolucional de Convnet.

Arquictectura	**Precisión Frames** (%)	**Precisión Videos** (%)
OF7 + SDAE-DNN + SM	$76{,}7 \pm 0{,}5$	$81{,}7 \pm 1{,}0$
OF7 + DNN + SM	$74{,}7 \pm 0{,}9$	$80{,}3 \pm 1{,}0$
OF7 + SM	$76{,}2 \pm 0{,}7$	$82{,}7 \pm 0{,}8$
OF6 + SDAE-DNN + SM	$80{,}0 \pm 0{,}9$	$83{,}0 \pm 1{,}0$
OF6 + DNN + SM	$80{,}0 \pm 0{,}8$	$83{,}0 \pm 1{,}2$
OF6 + SM	$\mathbf{83{,}9} \pm 0{,}8$	$\mathbf{88{,}3} \pm 1{,}2$
CONVNET	$70{,}3 \pm 1{,}4$	$70{,}0 \pm 2{,}1$

Tabla 4.2: Resultados para la versión *oneMatch* de rugby, clasificación los videos mediante votación simple.

Comparando las tablas 4.1 y 4.2 podemos pensar que nuestra sospecha era válida, al menos para la mayoría de los modelos. Aun así, el modelo con mejor performance para *main*, resultó ser también el mejor para *oneMatch*, en ambos casos con una precisión más alta de lo esperado, considerando el problema a resolver.

Suma de probabilidades y series temporales

Los resultados que aparecen anteriormente son satisfactorios, incluso superando nuestras expectativas, teniendo en cuenta la falta de una componente temporal y la sencillez del método de clasificación por votación. En vista de ajustar un poco más el modelo exitoso, se experimentó con otras maneras de predecir los videos, solo para encontrar los resultados que figuran en la Tabla 4.3.

Con ninguna de estas técnicas alternativas se encontraron mejoras en

Arquictectura	**Clasificación**	**Precisión Video** (%)
main		
OF6 + SM	Ventana corrediza + Detección de eventos	$80{,}9 \pm 3{,}8$
OF6 + SM	Ventana corrediza + SVM	$81{,}5 \pm 3{,}3$
OF6 + SM	Suma de probabilidades	$\mathbf{88{,}8} \pm 2{,}7$
CONVNET	Ventana corrediza + SVM	$76{,}0 \pm 2{,}4$
oneMatch		
OF6 + SM	Ventana corrediza + Detección de eventos	$75{,}9 \pm 2{,}2$
OF6 + SM	Ventana corrediza + SVM	$85{,}7 \pm 1{,}6$
OF6 + SM	Suma de probabilidades	$\mathbf{88{,}3} \pm 1{,}2$
CONVNET	Ventana corrediza + SVM	$68{,}0 \pm 1{,}7$
CONVNET	Suma de probabilidades	$69{,}3 \pm 2{,}7$

Tabla 4.3: Precisión de la predicción de videoclips para el dataset de rugby.

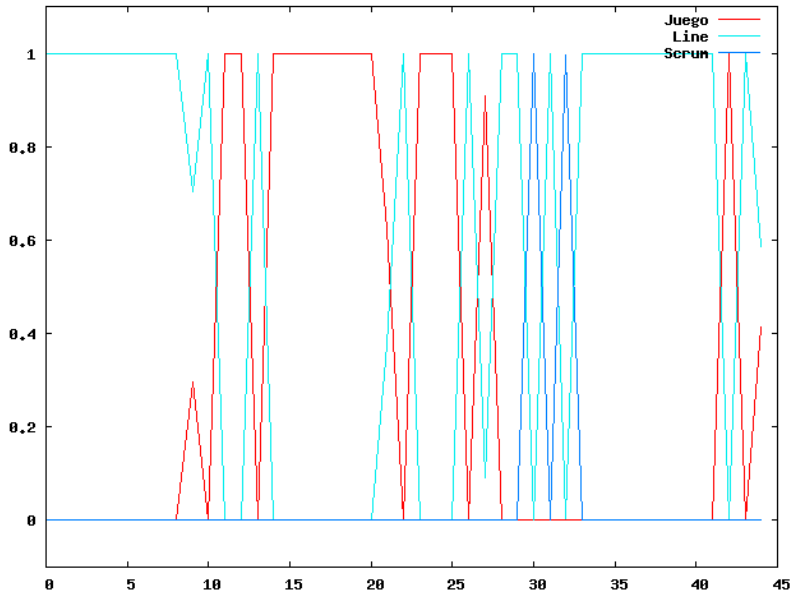


Figura 4.4: Ejemplo de serie temporal de probabilidades para un video de *main* producida por el modelo OF6 + SM.

performance, comparando con votación simple. El caso de OF6 + SM está íntimamente relacionado con las series de probabilidad que este produce (ver Figura 4.4). La función exponencial normalizada de Softmax directamente sobre los features de salida de Overfeat resulta en valores muy polarizados (casi siempre es 0 ó 1), y por ende la suma de probabilidades suele ser idéntica a la suma de los votos y las series temporales contienen una baja cantidad de información aprovechable. Ahora bien, viendo el caso de CONVNET, cuyas series temporales son ricas en variaciones de probabilidad (véase Figura 4.1), podemos pensar que las similitudes entre todos los

videoclips de una misma acción grupal no se traducen en similitudes entre sus series temporales de probabilidad.

4.4. Visualizaciones

Como parte final del trabajo, se proponen algunos mecanismos gráficos para revelar las representaciones internas generadas durante el aprendizaje de las redes convolucionales profundas. Esto aquí, es otro aporte del trabajo (útil en sí mismo, para cualquier modelo de clasificación) que echa luz sobre la hipótesis planteada.

Simplify

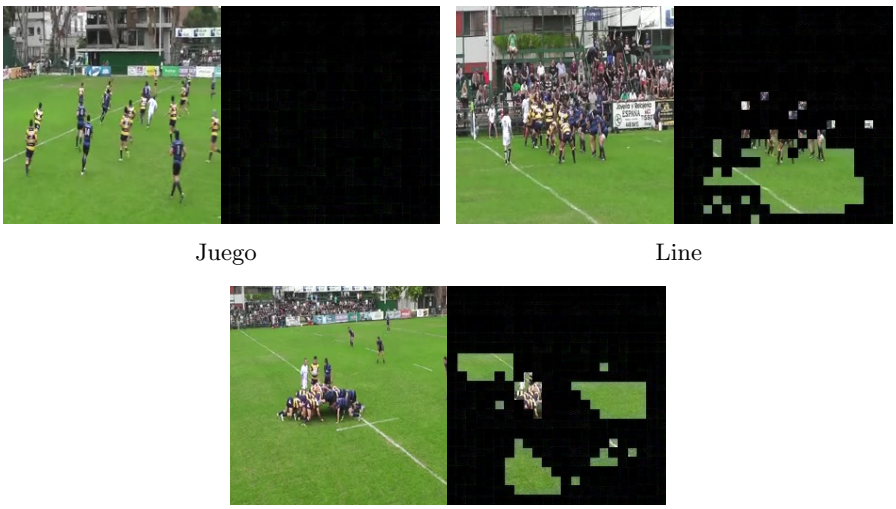
El primer enfoque intenta reducir progresivamente la información necesaria para clasificar una imagen. Esto es, partiendo de la imagen completa, se generan $\frac{wh}{s_x s_y}$ nuevas imágenes, donde w y h son el ancho y el alto en píxeles de la imagen, ubicando un rectángulo negro de tamaño $r = s_x \times s_y$ en cada posible región de una cuadrícula de $\frac{w}{s_x} \times \frac{h}{s_y}$ con el objetivo de ocultar una pequeña porción de la imagen. Luego, todas estas imágenes generadas son clasificadas por un modelo ya entrenado y la oclusión que menos decrementa la probabilidad de la clase real se queda. Entonces en la segunda iteración, la imagen contará con un pequeño rectángulo negro y ahora se generarán, de la misma manera, $\frac{wh}{s_x s_y} - 1$ nuevas imágenes. Siempre una menos que en la iteración anterior, dado que en cada iteración removemos un rectángulo de la imagen. Este proceso se repite hasta que la probabilidad de la clase real esté por debajo de un umbral fijo. La complejidad temporal de este método es:

$$T(n,t) = t \sum_{i=1}^{n/r} i = t\,\frac{\frac{n}{r}(\frac{n}{r}+1)}{2} = \frac{1}{2r^2}(n^2 t + rnt) = \mathcal{O}(n^2 t) \qquad (4.8)$$

donde $n = wh$ es la cantidad de píxeles de las imágenes de entrada y t es el tiempo que consume el modelo para producir un resultado. Este proceso toma aproximadamente cinco días en una computadora moderna[1], con imágenes del dataset de rugby (de tamaño 232×232) y $r = 10 \times 10$ ($t \approx 2{,}9s$).

En la Figura 4.5 se pueden observar, a la izquierda, imágenes de distintas clases sometidas al proceso de *simplify* y a la derecha, el resultado de dicho proceso. Cada una de estas imágenes es tomada de la version *oneMatch* del dataset de rugby, el tamaño del rectángulo es en todos los casos de $r = 10 \times 10$, y el modelo que se está estudiando es OF7 + SDAE_DNN + SM.

[1] Quadcore 2.4GHZ, 4GB Ram



Juego Line

Scrum

Figura 4.5: Imágenes de la versión *oneMatch* del dataset de rugby (a la izquierda) y sus simplificaciones (a la derecha). El modelo aquí utilizado es OF7 + SDAE_DNN + SM.

Las imágenes de la Figura 4.5 sugieren que para clasificar un Scrum es imprescindible ver al menos una parte de los jugadores que lo conforman y algunas regiones aledañas con nada más que césped. Similarmente, para detectar un line es necesario contar mínimamente con las piernas alineadas de los jugadores y una región de césped. Por último, la clase juego es bien clasificada aún cuando todo fue borrado de la imagen. Esto nos muestra que la clase *juego* es una suerte de clase descarte para la red, donde cae todo aquello que no parece *line* ni *scrum*.

Occlude

Otro mecanismo, presentado en [19], consiste en ocultar disitintos sectores de la imagen e iterativamente predecir utilizando algún modelo entrenado con el fin de entender qué partes de la imagen son realmente importantes para la correcta clasificación de la misma, según ese modelo. El proceso toma un parche gris de tamaño p y lo mueve a través de toda la imagen con un salto de $s_x \times s_y$ (ver Figura 4.6).

Cada imagen generada de esta manera, se propaga a través de la red y se obtiene la probabilidad de la clase real, generando así un mapa de $\frac{w}{s_x} \times \frac{h}{s_y}$ predicciones, siendo w y h, el ancho y la altura de la imagen en píxeles. Luego para cada pixel de la imagen real se promedian las predicciones de todas las

Figura 4.6: Iteración de Occlude sobre la 27ava fila con un tamaño de oclusión de 69px.

imágenes que lo ocultaron obteniendo así un mapa de probabilidades con las mismas dimensiones que la imagen original. Finalmente, escalamos los valores del mapa para poder apreciar el resultado, lo ploteamos como un mapa de calor y lo ubicamos sobre la imagen original. La complejidad temporal de este método es una función del tamaño de las imágenes del dataset y el tiempo que consume la predicción de una de ellas. Esto es:

$$T(n,t) = \frac{nt}{s_x s_y} = \mathcal{O}(nt) \tag{4.9}$$

donde $n = wh$ es la cantidad de píxeles de una imagen. Este programa se ejecuta en menos de dos horas en la misma computadora en la que *Simplify* tarda casi cinco días, con imágenes del dataset de rugby (de tamaño 232×232) y $s_x = 4$; $s_y = 4$.

Juego
Oclusión: 59 × 59 px

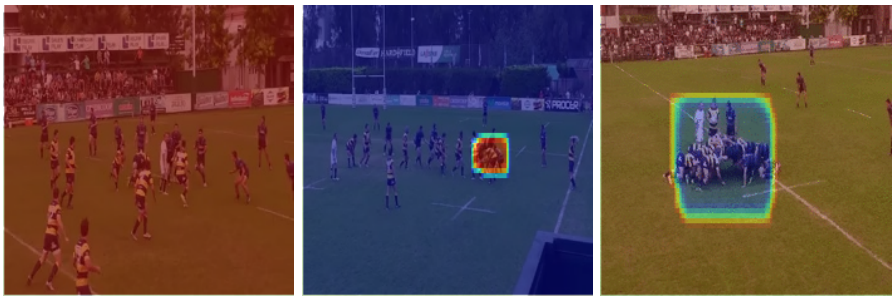
Line
Oclusión: 59 × 59 px

Scrum
Oclusión: 69 × 69 px

Figura 4.7: Visualizaciones *Occlude* de CONVNET para imágenes cuyas clases reales son, de izquierda a derecha, *juego*, *line* y *scrum*.

La Figura 4.7 muestra el resultado de aplicar el proceso *Occlude* utilizando el modelo CONVNET en imágenes tomadas de la versión *oneMatch* del dataset de rugby. El color azul más oscuro indica que tapar esa región resultó en la peor predicción de la clase real, y lo inverso con el color rojo más oscuro. En todas estas imágenes parece haber una franja horizontal crítica para la clasificación, es decir que conservarla puede mejorar o empeorar la clasificación. Esta franja, aunque con límites difusos, está presente en todas

visualizaciones *Occlude* realizadas sobre CONVNET, y posiblemente sea causada por el centrado que hace el camarógrafo sobre la región de interés. Cuando comparamos esto con aquellas realizadas sobre OF6 + SM (ver Figura 4.8) encontramos diferencias sustanciales, que explican la brecha que hay en el error de generalización de ambos modelos. Notar, por ejemplo, que la probabilidad de la correcta clasificación del *scrum* fue baja en OF6 + SM únicamente cuando se ocultó el scrum per sé, y ocultar cualquier otra región no alteró la probabildad de la clase correcta. Por el contrario, el line de la Figura 4.8, fue siempre mal clasificado salvo cuando de la imagen se quitaron los primeros hombres del line, que están en una posición inclinada, típica del scrum, y en cuyo caso la predicción arrojó una probabilidad muy alta de la clase real. Por último, en la clase *juego* es difícil evaluar las porciones relevantes, siendo la clase de descarte, y tal como se ve en la Figura 4.8 es bastante robusta a oclusiones.



Juego	Line	Scrum
Oclusión: 69 × 69 px	Oclusión: 19 × 19 px	Oclusión: 69 × 69 px

Figura 4.8: Visualizaciones *Occlude* del modelo OF6 + SM para imágenes cuyas clases reales son, de izquierda a derecha, *juego*, *line* y *scrum*.

Variando el tamaño de la oclusión

El tamaño de la oclusión es determinante para encontrar visualizaciones interesantes. Esto se debe a que la oclusión tiene que ser lo suficientemente grande como para ocultar una buena parte de lo que sirve para detectar la clase real de la imagen. En las figuras 4.9 y 4.10 podemos ver algunos esbozos de esto.

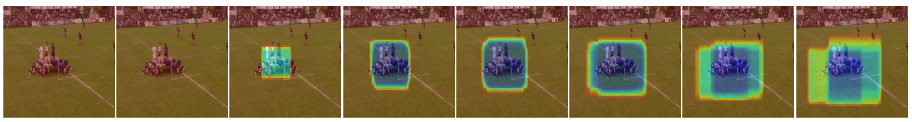


Figura 4.9: De izquierda a derecha, el tamaño de la oclusión aumenta con un paso de 10×10 empezando con una oclusión de tamaño 39×39 px. La clase real es *scrum* y el modelo visualizado es OF6 + SM.

Figura 4.10: De izquierda a derecha, el tamaño de la oclusión aumenta con un paso de 10×10 empezando con una oclusión de tamaño 39×39 px. La clase real es *line* y el modelo visualizado es OF6 + SM.

Capítulo 5

Conclusiones y trabajos futuros

En esta tesina se experimentó con redes convolucionales profundas para resolver el problema de clasificación de acciones grupales en video deportivo, con un abordaje naïve. Se comparó la eficiencia de modelos compuestos por redes pre-entrenadas usando millones de imágenes naturales con la de otros modelos ad-hoc que aprendieron únicamente de un conjunto de datos pequeño y específicio del problema. Se formularon otras funciones para determinar la clase de un video y se experimentó con esto. Finalmente, se propusieron e implementaron métodos de visualización, con el fin de estudiar la representación interna de los modelos. A continuación, haremos un resumen del conocimiento adquirido durante todo el proceso y propondremos posibles líneas de trabajo futuro.

5.1. Conclusiones

El presente trabajo propuso varias arquitecturas profundas que mostraron ser robustas a variaciones escénicas de todo tipo (algunas más que otras) y que establecieron la vara inicial del estado del arte en este dataset. Los modelos formados en parte por redes pre-entrenadas dieron mejores resultados que los modelos específicos. Las razones son diversas, pero entre ellas podemos destacar: la cantidad de mapas (sobre todo en las primeras capas) de Overfeat y la cantidad de datos para entrenar (1,2M contra 3500). Ambas razones marcan diferencias visibles en la calidad de los filtros de la primera capa y consiguientemente, en el error de generalización y en las visualizaciones *Occlude*. Nos propusimos, como objetivo general, investigar si contando únicamente con un conjunto de videos pequeño podríamos obtener resultados satisfactorios sin recurrir a un conjunto extra. No pudimos mostrar

que fuera así: Los resultados del modelo ad-hoc fueron considerablemente peores que cualquiera de los modelos pre-entrenados con *ImageNet*.

Se investigó además cómo etiquetar un video a partir de la probabilidad de cada uno de sus fotogramas. Los distintos métodos estudiados se comportaron igual o peor que la votación simple, que es el método menos costoso computacionalmente, por lo que se conservó ese método como estándar.

Se propuso un método de visualización y se trabajó con otro ya existente. Estos métodos son génericos para cualquier modelo de Aprendizaje Supervisado y revelan cuales son los sectores críticos en una imagen para su correcta clasificación. Ambos funcionan ocultando iterativamente regiones de las imágenes y propagándolas a través de un modelo entrenado. Occlude es considerablemente mejor que Simplify ya que produce un mapa de calor, en vez de una imagen binaria, y además su complejidad temporal es de un orden menor.

Finalmente, es importante destacar que los fotogramas de un video son imágenes estáticas que generan en los modelos características espaciales, únicamente. No obstante, las características presentes en las clases del dataset de rugby son mayormente reconocibles en frames aislados, es por esto que podemos alcanzar una performance satisfactoria. Aún así, entendemos que el problema se podría resolver con mayor eficiencia si considerásemos la componente temporal.

5.2. Trabajos futuros

Para continuar con esta línea de trabajo, podemos mencionar algunos proyectos que serían interesantes:

- Dos canales: Procesar datos temporales (como ser flujo óptico) utilizando otra arquitectura profunda y luego fusionar con el canal espacial (propuesto por esta tesina), como se sugiere en [24].
- Features 3D: Investigar sobre redes convolucionales con filtros tridimensionales, cuya tercera dimensión sea el tiempo.
- Más rugby: Extender el dataset de rugby, con transmisiones televisivas y filmaciones personales.
- Más deportes: Incorporar datos de otros deportes (p. ej., fútbol) para hacer un pre-entrenamiento.

Bibliografía

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard y L. D. Jackel, «Backpropagation applied to handwritten zip code recognition», *Neural Computation*, vol. 1, n.º 4, págs. 541-551, 1989.

[2] G. A. Miller, «WORDNET: A lexical database for english», en *Human Language Technology, Proceedings of a Workshop held at Plainsboro, New Jerey, USA, March 8-11, 1994*, 1994.

[3] V. N. Vapnik, *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995, ISBN: 0-387-94559-8.

[4] T. M. Mitchell, *Machine learning*, ép. McGraw Hill series in computer science. McGraw-Hill, 1997, ISBN: 978-0-07-042807-2.

[5] R. O. Duda, P. E. Hart y D. G. Stork, *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000, ISBN: 0471056693.

[6] D. A. Sadlier y N. E. O'Connor, «Event detection in field sports video using audio-visual features and a support vector machine», *IEEE Trans. Circuits Syst. Video Techn.*, vol. 15, n.º 10, págs. 1225-1233, 2005. DOI: 10.1109/TCSVT.2005.854237. dirección: http://doi.ieeecomputersociety.org/10.1109/TCSVT.2005.854237.

[7] Y. Bengio, P. Lamblin, D. Popovici y H. Larochelle, «Greedy layer-wise training of deep networks», en *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, 2006, págs. 153-160. dirección: http://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, ISBN: 0387310738.

[9] G. E. Hinton, S. Osindero e Y. W. Teh, «A fast learning algorithm for deep belief nets», *Neural Computation*, vol. 18, n.º 7, págs. 1527-1554, 2006. dirección: http://dx.doi.org/10.1162/neco.2006.18.7.1527.

[10] M. D. Rodriguez, J. Ahmed y M. Shah, «Action MACH a spatio-temporal maximum average correlation height filter for action recognition», en *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*, 2008. DOI: 10.1109/CVPR.2008.4587727. dirección: http://dx.doi.org/10.1109/CVPR.2008.4587727.

[11] I. Sutskever, G. E. Hinton y G. W. Taylor, «The recurrent temporal restricted boltzmann machine», en *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, 2008, págs. 1601-1608. dirección: http://papers.nips.cc/paper/3567-the-recurrent-temporal-restricted-boltzmann-machine.

[12] P. Vincent, H. Larochelle, Y. Bengio y P. Manzagol, «Extracting and composing robust features with denoising autoencoders», en *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, 2008, págs. 1096-1103. dirección: http://doi.acm.org/10.1145/1390156.1390294.

[13] J. Deng, W. Dong, R. Socher, L. Li, K. Li y F. Li, «Imagenet: A large-scale hierarchical image database», en *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, 2009, págs. 248-255. dirección: http://dx.doi.org/10.1109/CVPRW.2009.5206848.

[14] G. W. Taylor y G. E. Hinton, «Factored conditional restricted boltzmann machines for modeling motion style», en *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, 2009, págs. 1025-1032. dirección: http://doi.acm.org/10.1145/1553374.1553505.

[15] G. W. Taylor, L. Sigal, D. J. Fleet y G. E. Hinton, «Dynamical binary latent variable models for 3d human pose tracking», en *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR*

2010, San Francisco, CA, USA, 13-18 June 2010, 2010, págs. 631-638. dirección: http://dx.doi.org/10.1109/CVPR.2010.5540157.

[16] A. Krizhevsky, I. Sutskever y G. E. Hinton, «Imagenet classification with deep convolutional neural networks», en *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, 2012, págs. 1106-1114. dirección: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[17] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus e Y. LeCun, «Overfeat: integrated recognition, localization and detection using convolutional networks», *CoRR*, vol. abs/1312.6229, 2013. dirección: http://arxiv.org/abs/1312.6229.

[18] N. Wang y D. Yeung, «Learning a deep compact image representation for visual tracking», en *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, 2013, págs. 809-817. dirección: http://papers.nips.cc/paper/5192-learning-a-deep-compact-image-representation-for-visual-tracking.

[19] M. D. Zeiler y R. Fergus, «Visualizing and understanding convolutional networks», *CoRR*, vol. abs/1311.2901, 2013. dirección: http://arxiv.org/abs/1311.2901.

[20] K. Simonyan y A. Zisserman, «Very deep convolutional networks for large-scale image recognition», *CoRR*, vol. abs/1409.1556, 2014. dirección: http://arxiv.org/abs/1409.1556.

[21] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, «Dropout: a simple way to prevent neural networks from overfitting», *Journal of Machine Learning Research*, vol. 15, n.º 1, págs. 1929-1958, 2014. dirección: http://dl.acm.org/citation.cfm?id=2670313.

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke y A. Rabinovich, «Going deeper with convolutions», *CoRR*, vol. abs/1409.4842, 2014. dirección: http://arxiv.org/abs/1409.4842.

[23] J. Y. Ng, M. J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga y G. Toderici, «Beyond short snippets: deep networks for video classification», *CoRR*, vol. abs/1503.08909, 2015. dirección: http://arxiv.org/abs/1503.08909.

[24] Z. Wu, X. Wang, Y. Jiang, H. Ye y X. Xue, «Modeling spatial-temporal clues in a hybrid deep learning framework for video classification», *CoRR*, vol. abs/1504.01561, 2015. dirección: http://arxiv.org/abs/1504.01561.

[25] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel e Y. Bengio, «Show, attend and tell: neural image caption generation with visual attention», *CoRR*, vol. abs/1502.03044, 2015. dirección: http://arxiv.org/abs/1502.03044.

[26] Y. LeCun, «Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks)», en *Proceedings of Cognitiva 85*, Paris, France, 1985, págs. 599-604.