UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Sign2Text - Transcripción de lenguaje de signos (a nivel de palabra) mediante DL

Presentado por Iván Ruiz Gázquez
en Universidad de Burgos — 1 de julio de 2022
Tutores: Dr. Daniel Urda Muñoz y Dr. Bruno Baruque Zanon

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Iván Ruiz Gázquez, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 1 de julio de 2022

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de tablas

Introducción

Descripción del contenido del trabajo y del estrucutra de la memoria y del resto de materiales entregados.

Objetivos del proyecto

La principal motivación del proyecto es:

Mejorar la interacción entre personas que necesitan comunicación por signos y personas que no conocen el lenguaje de signos, intentando aumentar la calidad de vida de las primeras.

Objetivos teóricos

1. Creación de un modelo de DL[1] capaz de clasificar ASL[2] a nivel palabra.

2. Poner en producción un método para que los usuarios puedan probar el modelo de forma libre y transparante.

3. Generar una API de código abierto para que otros desarrolladores puedan crear plataformas de cliente, con el objeto de aumentar la audencia y alcance del proyecto.

4. Liberar y contenedorizar el modelo para su libre distribución.

5. Creación de herramientas que permitan el tratamiento de datos y la transformación entre distintos formatos[3].

6. Estudio del comportamiento, estructura e hiperparametrización de redes neuronales convolucionales.

[1] *Deep Learning*
[2] *American Sign Language*
[3] i.e.: extracción de fotogramas de un video, concatenación de imágenes

7. Aprender a fondo el uso de `PyTorch`[26], un *framework* para acelerar la creación y prototipado de redes neuronales.

8. Exportación del modelo entrenado a un formato estándar que facilite la compatibilidad con el mayor número de librerías en distintos lenguajes de programación.

9. Mantenimiento de una *codebase* que favorezca la integración continua (*linting*[4], *formatting*[5] y *type-checking*[6]) de código, siguiendo así los estándares en contribuciones *Open Source*.

Estos objetivos representan en general la filosofía y los objetivos teóricos del proyecto. Si entramos más en detalle sobre los aspectos técnicos y las *features* que se esperan obtener al finalizar el proyecto, obtenemos los siguientes puntos:

Features **esperadas**

- Debemos ser capaces de inferir resultados del modelo entrenado a tiempo real.
- Se desarrollará un modelo fácilmente escalable y adaptable a distintos dataset de cualquier tamaño.[7]
- Se realizará una fase de *data augmentation* sobre los datos iniciales.
- Se implementarán distintas redes neuronales para los ditintos formatos de datos (imagen y video). Deben ser fácilmente refactorizables y mantenibles.
- Cada vez que se estudie el uso de un nuevo formato de dataset, se creará una nueva red, manteniendo la usabilidad de la anterior intacta.
- A lo largo de las pruebas y entrenamientos de la red, vamos a probar distintos *schedulers*, *optimizers* y *criterions*[8] buscando el que mejor se adecúe al formato de los datos y la estructura de la red.

[4] Voz ingl. Estudio de limpieza, orden, calidad y redundancia
[5] Formatear: Correcta estructura y legibilidad
[6] Comprobación de tipados de variables y funciones
[7] El formato de los datasets debe ser el mismo. Un modelo entrenado para clasificación de imágenes no podrá clasificar en formato video o audio.
[8] Funciones de cálculo de pérdida

- Se mantendrán unas estadísticas a tiempo real de la fase de *trainning* y *test* mediante el uso de `Tensorboard`, una herramienta analítica que permite mantener un *log* de imágenes generadas en la ejecución; así como gráficas de costes y *accuracies*.

Conceptos teóricos

Este proyecto es un proyecto de *machine learning*. Dentro del *machine learning*, es de aprendizaje supervisado. Más concretamente intenta resolver un problema de clasificación mediante *deep learning*. Las redes neuronales aplicadas para la clasificación serán CNN (*Convolutional Neural Networks*), y más concretamente *ResNets*, una arquitectura sobre las CNN. Veamos en detalle todos estos terminos en los siguientes apartados.

3.1. Machine Learning

Según Tom Mitchell [29], un programa de ordenador aprende de una experiencia E con respecto a alguna tarea T si su medida de desempeño P (del inglés *performance*) mejora con E.

El uso del machine learning se origina en la búsqueda de soluciones a problemas que no podemos resolver programáticamente. Determinar las soluciones se centra en la recogida y análisis de datos[9] con la finalidad de buscar relaciones entre ellos.

Para aclarar este concepto, veamos un ejemplo [1]: Imaginemos que debemos estimar el precio de un coche usado pero no tenemos la fórmula exacta para valorar su estado. Lo que si sabemos es que el precio del coche aumenta y disminuye dependiendo de sus propiedades, como la marca, el kilometraje, o el desgaste. No sabemos tampoco cuales de las propiedades

[9] A lo largo del documento nos referiremos a los datos indistintamente como «datos» o «instancias»

tiene más importancia. Sabemos seguro que cuantos más kilómetros tenga el coche, menor será su precio, pero no sabemos a que escala ocurre esto.

Para determinar la solución necesitamos estudiar el precio de coches en el mercado y anotar sus características. Estos datos son los que daremos a nuestro programa para que busque las relaciones entre propiedades e indique el precio óptimo de los coches.

Saliendo del ejemplo, el machine learning se divide en distintos tipos según la cantidad de datos (o supervisión). Hay más formas de dividir el machine learning, como puedes ver en 3.1 (nombraremos alguna de ella a lo largo de los siguientes apartados), pero nos vamos a centrar en *supervised, unsupervised* y *semi-supervised learning*.

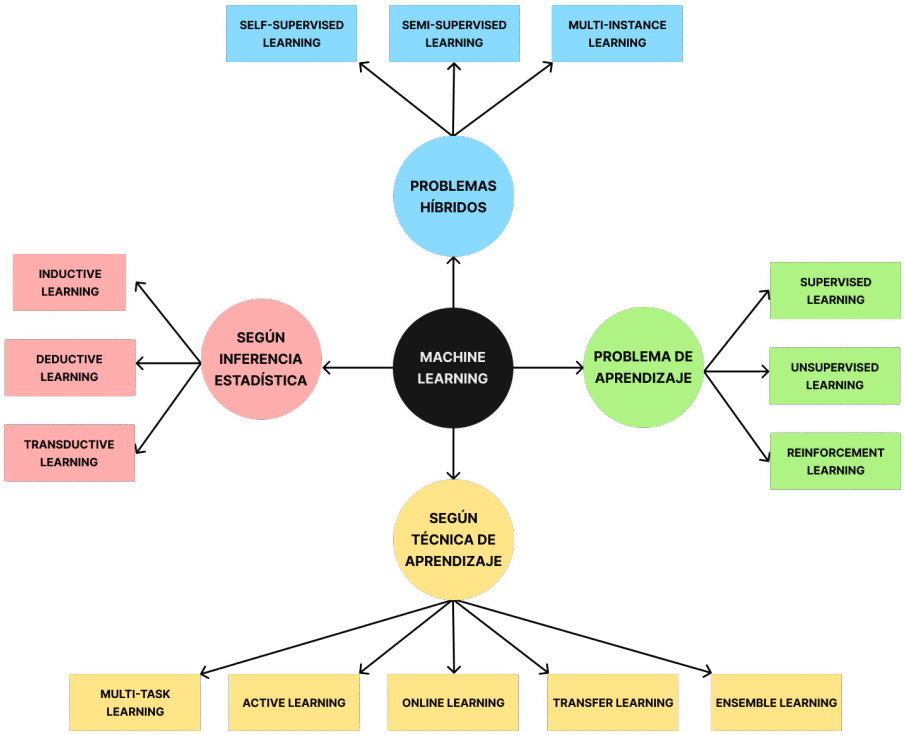

Figura 3.1: Tipos de *machine learning* según datos, inferencia estadística y técnicas de aprendizaje

Unsupervised Learning

El *unsupervised learning* o aprendizaje no supervisado, tiene su principal característica en que es capaz de detectar las relaciones entre los

datos sin ayuda de etiquetas o respuestas externas. Si volvemos al ejemplo de los coches, seríamos capaces de determinar el precio de venta sin necesidad de recibir datos de coches reales.

La técnica principal en el aprendizaje no supervisado es el agrupamiento de los datos (o *clustering*) según la similaridad de sus características. Pero hay más técnicas.

Clustering

El clustering consiste en el agrupamiento de datos no etiquetados basándonos en sus similaridades o diferencias. Los algoritmos de *clustering* son usados para procesar datos crudos sin alteraciones en grupos representados por patrones o estructuras de información. Estos algoritmos pueden clasificar en:

- **Mutualmente excluyentes**: En estos algoritmos se estipula que una instancia de datos solo puede existir en un y solo un *cluster* (o grupo). Este tipo de *clustering* también se denomina *«hard» clustering*. Un ejemplo de algoritmo mutualmente excluyente es el algoritmo de *K-means*. En *K-means*, la «K» indica el número de *clusters* basándonos en su centroide[10]. Los puntos, representando a un dato, caeran en el grupo con el centroide más cercano. Un valor de *K-means* mayor indicará mayor número de agrupamientos.

[10] O centro geométrico, es la posición media aritmética entre todos los puntos de una figura

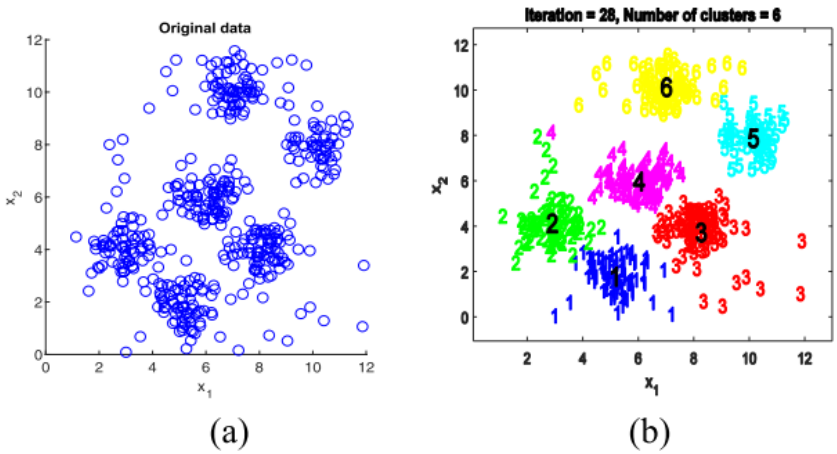

Figura 3.2: Ejemplo de datos antes y después del *clustering* para un algoritmo mutualmente exclutente *K-means* [42].

- **Superpuestos**: En el caso de los algoritmos superpuestos, la diferencia es que un dato puede pertenecer a más de un *cluster* de forma simultánea con distinto grado de pertenencia. Un ejemplo de algoritmo superpuesto es el *«soft» K-means*.

- **Jerárquicos**: También conocido como HCA (*hierarchical cluster analysis*), es un algoritmo no supervisado que se puede categorizar de dos modos, en aglomerativo o divisivo. El primero funciona con acercamiento *«bottom-up»*. Los datos comienzan de forma separada y se van uniendo de forma iterativa según similitud hasta conseguir un único cluster. La similaridad se puede medir de las siguientes formas:

  1. Método de *Ward*: La distancia entre dos clusters se define por el aumento de la distancia cuadrática después de que los clusters sean fusionados.
  2. Promedio de distancias: Este método usa la distancia media entre dos puntos en cada *cluster*.
  3. Distancia máxima: La distancia usada es la distancia mayor entre dos puntos en distintos clusters.
  4. Distancia mínima: Se define por la distancia mínima entre dos puntos de dos clusters diferentes.

  La medida más comun a la hora de calcular estas distancias es la distancia euclídea, aunque otra muy común en la literatura es la distancia

Manhattan. Por otro lado, el acercamiento divisivo, o *«top-down»* funciona dividinendo un único *cluster* basándose en las diferencias entre instancias (puntos del cluster). Visualmente se parece a un dendrograma[11].

- **Probabilísticos**: En un algoritmo probabilístico, los puntos se agrupan basándose en la probabilidad de que pertenezcan a una distribución paramétrica [12]. El GMM (*Gaussian Mixture Model*) es el método más usado en *clustereing* probabilístico.

  - *Gaussian Mixture Model*: Este algoritmo se clasifica como mixto debido a que se compone de un número no especificado de funciones de densidad probabilísitca. Normalmente se encargará de comprobar si un instancia es parte de una distribución Gaussiana o normal. El algoritmo más usado para determinar la distribución es el E-M (Expectation-Maximizarion) [10].

Reglas de asociación

Las reglas de asociación es un método que busca buscar las relaciones entre variables de un *dataset*. Suelen ser usados en *marketing*, sistemas de recomendación, relación entre productos, estudio de hábitos de consumición [32], entre otros. El algoritmo más usado en reglas de asociación es el algoritmo Apriori.

- **Algoritmo Apriori**: Este algoritmo ppermite encontrar de forma frecuente «conjuntos de elementos frecuentes». Estos conjuntos sirven para generar reglas de asociación que permiten extender los sets a sets de mayor tamaño.

[11] Representación gráfica de datos en forma de árbol para organizar datos en categorias y subcategorias hasta alcanzar el nivel de detalle deseado

[12] La estadística paramétrica es una rama de la estadística con la que se deciden valores basándose en distribuciones desconocidas. Estas se determinan según un número finito de parámetros.

Algoritmo 1: Algoritmo Apriori

Input: Para un dataset T, un umbral de soporte ε y un conjunto de datos C_k para el nivel k

```
1  L_1 ← findFrequentOneItemsets(T)
2  k ←2
3  while L_{k-1} not empty do
       /* Generación de Apriori                    */
4      C_k ← empty list
5      forall p ⊆ L, q ⊆ L do
6          c ← p ∪ q_{k-1}
7          forall u ∈ L do
8              if u ⊆ c then
9                  C_k ← C_k ∪ {c}
10             end if
11         end forall
12     end forall
       /* Cálculo de candidatos                    */
13     for t ∈ T do
14         D_t ← {c ∈ C_k : c ⊆ t}
15         for c ∈ D_t do
16             count[c] ← count[c] + 1
17         end for
18     end for
19     k ← k + 1
20     L_k ← {c ∈ C_k : count[c] ≥ ε}
21 end while
22 return Union(L_k)
```

Reducción de la dimensionalidad

Normalmente cuando disponemos de una mayor cantidad de datos, tendemos a obtener mejores soluciones, aunque esto tiene impacto en el rendimiento de nuestros algritmos y puede dificultarnos comprender de forma completa nuestro *dataset*. La reducción de la dimensionalidad se usa cuando el número de características o dimensiones de un *dataset* es muy grande. Reduce el número de instancias a un tamaño manejable manteniendo la integridad de los datos lo máximo posible. Algunos métodos de reducción

de la simensionalidad son [44][15]: PCA (del inglés *Principal Component Analysis*), SVD (*Singular Value Decomposition*) y *Autoencoders*.

Semi-supervised Learning

El aprendizaje semi-supervisado se caracteriza por tener la capacidad de aprender de conjuntos de datos cuyas instancias no están completamente etiquetadas. En nuestro ejemplo 3.1«cálculo de precio de un coche según sus propiedades», tendríamos precios reales solo para algunos de los coches, mientras que otros no tendrían una etiqueta establecida.

La característica principal que diferencia el aprendizaje semi-supervisado del aprendizaje no supervisado y del aprendizaje supervisado (que veremos a continuación) es que la capacidad de aprender del algoritmo se nutre del hecho de que algunos datos no estén etiquetados. A la hora de intentar averiguar las etiquetas faltantes en base a las presentes, podemos observar comportamientos de aprendizaje que no se verían en los otros dos tipos de *machine learning*.

De este modo, no es útil solo como algoritmo de aprendizaje si no como algoritmo de generación de nuevos datos en aprendizaje supervisado [49]. Si la obtención de datos y etiquetas nos supone un proceso costoso y/o el acceso a los datos es limitado [13], podemos crear un algoritmo de aprendizaje semi-supervisado para completar la falta de datos.

Algunos de los métodos usados en aprendizaje semi-supervisado son los siguientes:

- Modelos generativos: Con este modelo se busca estimar $p(x \mid y)$, la distribución de los puntos pertenecientes a cada clase. La probabilidad $p(x \mid y)$ de que un punto x pertenezca a una etiqueta y es proporcional a $p(x \mid y) \cdot p(y)$ por la Regla de Bayes.

  Extrayéndolo de [49], los modelos generativos asumen que las distribuciones tienen la forma $p(x \mid y, \theta)$ para metrizada por el vector θ. Si esta suposición es incorrecta, los datos no etiquetados pueden incluso disminuir la capacidad de acierto de la solución. Pero, en el caso contrario, si la suposición es correcta, los datos etiquetados mejoraran el rendimiento del modelo.

[13] i.e.: La recopilación de datos médicos es algo compleja, ya que se debe tener permiso del paciente, así como hay enfermedades cuyos datos son muy escasos y difíciles de conseguir.

Los datos no etiquetados se suelen distribuir mediante una distribución Gaussiana mixta. La distribución de probabilidad conjunta se puede escribir como $p(x, y \mid \theta) = p(y \mid \theta) \cdot p(x \mid \theta)$ mediante el teorema de probabilidad compuesta. Cada vector θ se asocia con una función de decisión:

$$f_\theta(x) = \underset{x}{\operatorname{argmax}} p(y \mid x, \theta)$$

El parámetro se elige entonces basándonos en la adaptación a los datos etiquetados y no etiquetados, usamos un peso de balanceo λ:

$$\underset{\theta}{\operatorname{argmax}} \left(\log p \left(\{x_i, y_i\}_{i=1}^{l} \mid \theta \right) + \lambda \log p \left(\{x_i\}_{i=l+1}^{l+u} \mid \theta \right) \right)$$

- *Support Vector Machines* (SVM) : Según Van Engelen *et al.*[46] un SVM es un clasificador que intenta buscar la frontera de decisión que maximice el margen, que a su vez se define como la distancia entre la fontera de decisión y los puntos cercanos a ella[14]. En el escenario del aprendizaje semi-supervisado, un objeto adicional gana importancia: también queremos minimizar el número de datos «no etiquetados» que viola el margen. Como no podemos saber la etiqueta de los datos «no etiquetados», aquellos puntos que violan el margen son penalizados basándonos en su distancia a la frontera de decisión.

- Modelos basados en gráfos: Este modelo asume un grafo $G = V, E$ tal que los vértices V son las instancias de datos (etiquetados y no etiquetados) y E las aristas no dirigidas que conectan las instancias i, j con un peso $w_i j$. La construcción de un grafo se cumple en los siguientes pasos [43][46] 3.3:

  1. **Construcción del grafo**: Se suele asumir una instanciación inicial del grafo aleatoria
  2. ***Weighting***: Cálculo de pesos de los datos etiquetados para la fase de inferencia.
  3. **Inferencia de etiquetas**: La tarea a cumplir aquí es propagar información de las instancias etiquetadas a las «no etiquetadas» incorporando la estructura de grafo creada en el paso anterior.

[14] A veces el margen también se refiere a la zona delimitada por la fontera de decisión en la que caen los puntos

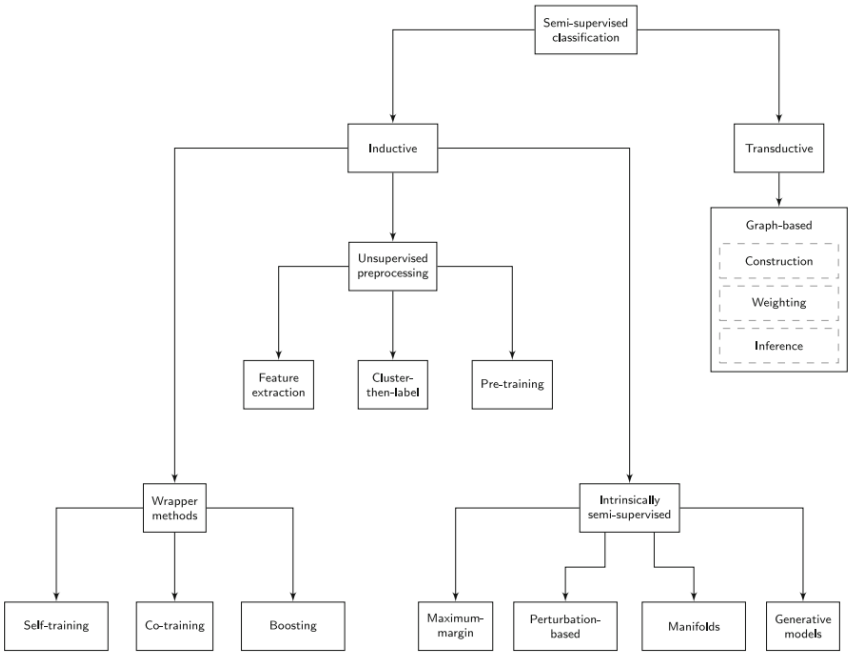

Figura 3.3: Visualización de la taxonomia de los métodos de aprendizaje semi-supervisado. En la rama correspondiente a los métodos basados en grafo podemos observar las distintas fases de clasificación [46]

Supervised Learning

Poco hace falta una definición de aprendizaje supervisado después de haber revisado aprendizaje no supervisado y aprendizaje semi-supervisado, pero por mantener el aislamiento de los apartados, diremos que: tambien llamado *classification*[15] o *inductive learning* (recordemos fig. 3.1), el aprendizaje supervisado es aquel aprendizaje cuyas instancias de datos (usadas para que nuestro programa aprenda) están todas etiquetadas.

Según Christopher M. Bishop *et al.*[4], el objetivo último del aprensizaje supervisado es ser capaces de observar una variable t (*target*) junto con una variable de entrada x y predecir el valor de t para cualquier nuevo valor de x.

Para conseguir la predicción de nuevos datos debemos crear un modelo que va a ser «entrenado», «testeado» y «validado». Para conseguir datos para las tres fases dividiremos nuestro *dataset* en tres grupos respectivamente.

[15] A pesar que no solo trate problemas de clasificación

Para que el modelo tenga un alto acierto en sus predicciones debe estar bien entrenado, por lo que la mayor fracción de los datos se destinará a esta fase[16]. Pero hay que tener cuidado, porque si asignamos un pocentaje demasiado alto de los datos a la fase de entrenamiento podemos generar un *overfitting* en el modelo.

El overfitting ocurre cuando un modelo se ha sobreentrenado con unos datos, por que lo que a la entrada de nuevos datos (desconocidos) x, generará una predicción t sesgada a los datos de entrenamiento.

Para evitar esto, usamos otra fracción de los datos de entrada como fracción «de control». De este modo iremos comprobando la deriva de las predicciones entre los datos de entrenamiento y los datos (has adivinado) de validación.

Ahora podremos controlar la fase de entrenamiento y parar dicho entrenamiento, valga la redundancia, cuando detectemos este sobreajuste. Podemos ver esto en la figura 3.4. Gracias al *set* de validación, detectamos el momento en el que la red comienza a sobreajustarse. Esto no solo nos evita cometer errores en la fase de test, sino que mediante la eliminación de iteraciones inútiles sobre el modelo, disminuimos en gran cantidad el tiempo de entrenamiento.

[16]Normalmente un 70 % de los datos se asigna a la fase de entrenamiento, aunque depende mucho del problema específico al el que nos enfrentemos

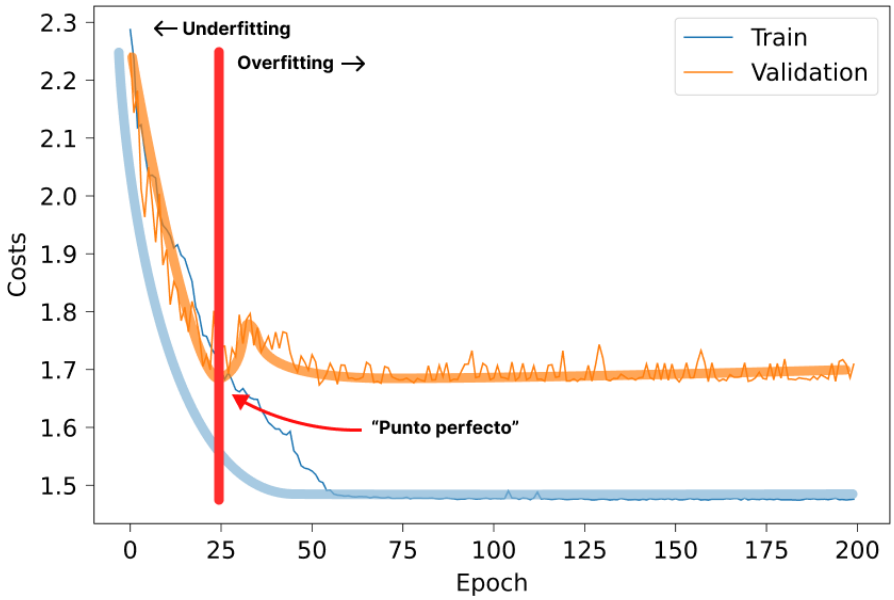

Figura 3.4: Visualización de overfitting en un entrenamiento de una CNN (red neuronal convolucional). Podemos observar en rojo el punto exacto en el que la red comienza a sobre-ajustarse. Este es el momento en el que debemos parar el entranamiento.

En la imagen vemos que el valor del eje de ordenadas tenemos el valor de los costes sobre las iteraciones (*epochs* en inglés). Veremos más sobre esto en el apartado de *fine-tunning* e hiperparametrización de una red neuronal. Veamos ahora los distintos tipos de aprendizaje supervisado.

Regression

En la regresión, el objetivo es predecir un valor numérico y contínuo. Uno de los usos principales de modelos de regresión es el relleno de espacios en los datos o la estimación de valores futuros. Por otro lado, la regresión tambiñen se puede utilizar para determinar realciones casuales entre variables dependientes e independientes [27].

En los modelos de regresión, las variables independientes predicen a las variables dependientes. El análisis de regresión estima la variable dependiente y en base al rango de variables independientes x. En cuanto a los tipos de regresión, puede ser simple o múltiple.

- **Regresión simple**: En las regresiones simples solo tenemos una variable independiente y. Se define la dependencia de la variable como $y = \beta_0 + \beta_1 x + \epsilon$. De forma gráfica, la regresión simple se muestra como una línea entre los puntos cuyo objetivo es minimizar el error cuadrático de las distancias entre dicha línea y los puntos. Es común que en estos modelos tengamos puntos atípicos que se sitúen lejos de la mayoría. Estos puntos se denominan *Outliers* y al distanciarse del resto, provocan desviaciones importantes en la regresión [40].

- **Regresión múltiple**: El objetivo de los modelos múltiples es la predicción del conjunto de variables independientes y según el conjunto de variables dependiente x. El modelo básico es $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m + \epsilon$. La fórmula para determinar la matriz es:

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T y \quad \text{donde}$$

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{bmatrix}, X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1m} \\ 1 & x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_m \end{bmatrix}, Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}$$

Classification

El segundo y último tipo de tipo de aprendizaje supervisado es también el más familiar. La clasificación consiste en etiquetar una variable discreta y dada una variable de entrada x. Según el número de etiquetas disponibles para la clasificación hablamos de clasificación binaria o multiclase.

Aunque no listamos todos los métodos, a continuación se pueden observar los más ampliamente usados[35]:

1. **Árboles de decisión**: Los árboles de decisión son una forma de representar reglas sobre los datos de forma jerárquica con una estructura secuancial recursiva que particiona los datos. Los árboles de decisión se usan para aproximar funciones discretas. Un nodo hoja indica el valor de la etiqueta y, mientras que un nodo de decisión especifica una operación a realizar. La evaluación comienza avaluando el nodo raíz (un nodo de decisión) y moviendonos hasta encontrarnos con un

nodo hoja, con la etiqueta buscada. El algoritmo más común en la construcción de árboles de decisión es una extensión del algoritmo propuesto por J. Quinlan en 1993 *et al.*[33]: ID3 (*Iterative Dichotomiser* 3). El árbol se construye *top-down* de forma recursiva con la técnica de «divide y venceras». Puedes ver el psudocódigo a continuación.

Algoritmo 2: Algoritmo ID3 - TreeBuilding(X, Y)

Input: Para un set de entradas de entrenamiento X y una lista de etiquetas correspondientes Y

```
1  N ← Nodo raíz
2  if ∀ muestra = clase C then
3  |  etiqueta N = C
4  |  return
5  end if
6  if Y = vacío then
7  |  etiqueta N = clase más común C en X(voto mayoritario)
8  |  return
9  end if
10 y ← El atributo que mejor clasifica ejemplos donde y ∈ Y
11 etiqueta N ← y
12 forall v ∈ y do
13 |  Haz crecer una rama desde N con condición y = v
14 |  X_v ← subset de instancias en Y con y = v
15 |  if X_v = vacío then
16 |  |  añade un nodo hoja etiquetado con la clase más común en X
17 |  end if
18 |  else
19 |  |  añade el nodo generado por TreeBuilding(X_v, Y − v)
20 |  end if
21 end forall
```

2. ***Bagging* y *boosting***: Es una técnica de ensamblado que combina varios métodos de *machine learning* en un único modelo predictivo en orden de disminuir la varianza (*bagging*) o el *bias(boosting)*.

   El *bagging (Bootstrap Aggregating)* comienza creando muestras aleatorias del *set* de entrenamiento. Tras esto, construye un clasificador para cada muestra. Finalmente, se combinan los resultados de los clasificadores y comienza una votación de mayoría. Al construir estos modelos al mismo tiempo, consideramos a Bagging un método de

ensamblado paralelo. Al aumentar el tamaño del *set* de entrenamiento, se disminuye la varianza del modelo final [2].

El *boosting* por otro lado se compone de dos pasos. En el primero usa *subsets* del *set* de entrenamiento para producir modelos con rendimiento medio. Tras esto estimula el rendimiento combinándolos usando el voto mayoritario. En el *boosting*, la creación de *subsets* no es aleatoria y depende del rendimiento de los modelos anteriores.

La principal diferencia entonces entre los dos, es que *bagging* aprende el *dataset* completo de forma paralela mientars que *boosting* aprende usando los resuultados de los modelos anteriores. *Boosting* muestra una mejor *accuracy* que *bagging* [34], pero tambiñen tiene a producir *overfitting* en los modelos.

3. ***Random Forest***: El *random forest* también se puede usar en tareas de regresión. El principio que se sigue es que un conjunto de clasificadores «débiles» se puede unir para crear un clasificador «fuerte». En este caso tenemos un *ensemble* de árboles de decisión que funciona mejor que *bagging* y *boosting* tanto en *accuracy* como en *overfitting*. Cada árbol del *ensemble* tiene un *set* de entrenamiento distinto. Debemos tener en cuenta que si necesitamos una descipción de las relaciones existentes entre los datos, los *random forest* no son la mejor opción, ya que al tener un conjunto de árboles, se hace difícil la compresión de la estructura general.

4. **SVM (*Support Vector Machines*)**: Los SVM también están presentes en la regresión. En el caso de la clasificación, tenemos un conjunto de puntos en un espacio de dimension N. Cada set perteneciendo a una de las posibles clases, el SVM busca los planos separados que maximicen los margenes entre los *sets* de datos. Ya hemos visto en 22 un caso de SVM en aprensizaje semi-supervisado, solo que en este caso funciona distino. A la hora de clasificar un dato, separamos dos planos y calculamos el error cuadrático de los puntos a dichos planos. En algunas ocasiones, los planos pueden no ser linealmente separables; en estos casos, debemos aumentar el número de dimensiones hasta que lo sean. Para esto se usa lo que se demonina función *kernel* [6]

$$K(x, y) = \langle f(x), f(y) \rangle$$

donde x e y son las entradas de dimensión n, y f es un mapa de paso de dimensión-n a dimensión-m. Normalmente nos encontraremos con que m es un número mucho mayor a n.

5. ***Naive Bayes***: Este clasificador probabilístico se basa en el teorema de Bayes y asume que todas las características están independientemente condicionadas por la etiqueta de la clase [30]. Aunque esto no suele ocurrir, el modelo resultante funciona sorprendentemente bien, a veces competiendo con técnicas mucho más sofisticadas [36]. Este modelo ha probado ser muy útil en tareas de clasificación de texto y diagnóstico médico, entre otros. En [11] se prueba que *naive Bayes* funciona de forma óptima en problemas con un alto grado de dependencia entre características.

6. **Redes Neuronales**: Por último, tenemos las redes neuronales. Las redes neuronales ocupan la mayoria de la literatura sobre inteligencia artificial y *machine learning* en la actualidad. Una red neuronal es un conjunto de neuronas artificiales colocadas en capas. Cada capa tiene la tarea de extraer características de los datos de entrada, actualizar sus pesos y «pasar» información a las siguientes capas.

   Una neurona es una función f_j con una entrada $x = (x_1, \ldots, x_d)$ y un vector de pesos $w_j = (w_{j,1}, \ldots, w_{j,d})$, junto con un *bias* b_j y asociado a una función de activación ϕ [3]. La fórmula completa de está función es

   $$y_j = f_j(x) = \phi(\langle w_j, x \rangle + b_j)$$

   Existen numerosas funciones de activación en una red neuronal, algunas son:

   - Función identidad o lineal

     $$\phi(x) = x$$

   - *Binary Step*

     $$\phi(x) = \begin{cases} 0 & \text{para } x < 0 \\ 1 & \text{para } x \geq 0 \end{cases}$$

   - Función sigmoide

     $$\phi(x) = \frac{1}{1 + \exp(-x)}$$

   - Tangente hiperbólica

     $$\phi(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1}$$

- ReLU (*Rectified Linear Unit*)

$$\phi(x) = \text{máx}(0, x)$$

- *Leaky* ReLU

$$\phi(x) = max(0{,}1 \cdot x, x)$$

- ELU (*Exponential Linear Unit*)

$$\phi(x) = \begin{cases} x & \text{para } x \geq 0 \\ \alpha(\exp(x) - 1) & \text{para } x < 0 \end{cases}$$

- Función softmax: descrita como una combinación de múltiples sigmoides, calculamos su salida con

$$\phi(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

  siendo x el conjunto de las salidas de una neurona en un problema multiclase.

Tanto la función de *binary step* como la función de identidad (ambas funciones lineales) no permiten *backpropagation* ya que la derivada de la función es una constante y no tiene relación alguna con la variable de entrada x (no aporta datos adicionales) [19].

Por otro lado, en regresión se usan funciones de activación lineales, mientras que en clasificación, usaríamos sigmoides en clasificación binaria, o softmax en clasificación multietiqueta.

Una vez tenemos nuestra función de activación elegida, podemos calcular la salida de la red dada una variable de entarda x. Y una vez caculada la salida de la red, podemos comparar el resultado y con la etiqueta de *ground truth* correspondiente con dicha entrada. Según sea igual o no, modificaremos el vector de pesos de nuestra neurona.

Repetiremos este proceso hatsa que la *accuracy* de la neurona (o la red de neuronas) sea superior al *threshold* buscado [35].

Podemos medir el *accuracy* de una neurona dividiendo el número de predicciones correctas entre el número total de predicciones [14]. Pero el *accuracy* no es el único método de medida de acierto en las redes neuronales. También tenemos la precisión, el *recall*, el *F-measure*, la especificidad, el *fallout* o la Coeficiente de Correlación de *Mathew* (MCC) [7].

Con

$$tp = true\ positives$$
$$tn = true\ negatives$$
$$fp = false\ positives$$
$$fn = false\ negatives$$

- Accuracy: $Acc = \frac{tp+tn}{tp+tn+fp+fn}$
- Precisión: $P = \frac{tp}{tp+fp}$
- *Recall*: $R = \frac{tp}{tp+fn}$
- *F-measure*: $F = \frac{2P \cdot R}{P+R}$
- Especificidad: $S_p = \frac{tn}{tn+fp}$
- *Fallout*: $\frac{fp}{tn+fp}$
- Coeficiente de Correlación de *Mathew*: MCC $= \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp+fp)(tn+fn)(tp+fn)(tn+fp)}}$

Deep Learning

Hemos visto en el último apartado las características de una red neuronal de forma individual: como se compone, que tipos de funciones de activación tenemos y como podemos evaluar y medir la salida de una neurona.

Las redes neuronales más básicas: FFNN (*Feed Forward Neural Netowrks*) y Perceptrones, se componen únicamente de una capa de entrada y salida, con ocasionalmente una o más capas ocultas entre ambas. El método para entrenar estas redes suele ser el *backpropagation*.

El término *backpropagation* viene tras ser empleado por *Rosenblatt* (1962) [5] a la hora de intentar generalizar el entrenamiento de un perceptrón a múltiples capas ocultas. No fué muy exitoso hasta que *Rumelhart* y *Williams* (1986) [39] extendieron la idea y la hicieron popular entre investigadores [38]. La idea del *backpropagation* es minimizar el error entre la salida correcta e incorrecta mediante el *gradient descent*. Para esto necesitamos calcular las derivadas parciales de nuestras funciones de activación [17]. Calculamos la salida de la derivada en el *forward pass*, y con

[17] Recordemos que solo las funciones de activación no lineales pueden ser usadas para el *backpropagation*, ya que la derivada de una función lineal es una constante

el resultado obtenido, realizamos un *backward pass*, con el que propagamos y actualizamos los pesos de la red.

Esto que ahora nos parece básico, pero fué una «revolución» en su momento y es la base principal del *deep learning*. Pero, el *deep learning* no solo trata el aprendizaje supervisado, (también tenemos *deep reinforcement learning*, *unsupervised learning*[18] y *semi-supervised learning*). Por simplificar vamos a tratar solo redes neuronales de aprendizaje supervisado o DNN (*Deep Neural Networks*). Una lista extensa de redes neuronales se puede ver en [21].

Las *deep neural networks* (DNN) se clasifican en dos tipos principales: Redes Neuronales Recurrentes (RNN) y Redes Neuronales Convolucionales (CNN). Se diferencia de las redes neuronales artificiales «clásicas» (ANR) [19] en que en vez de tener una capa de entrada seguida de una o más capas escondidas, con una capa de salida a continuación; las DNN tienen más de una capa entre la salida y la entrada, de ahí que se denominen profundas [21].

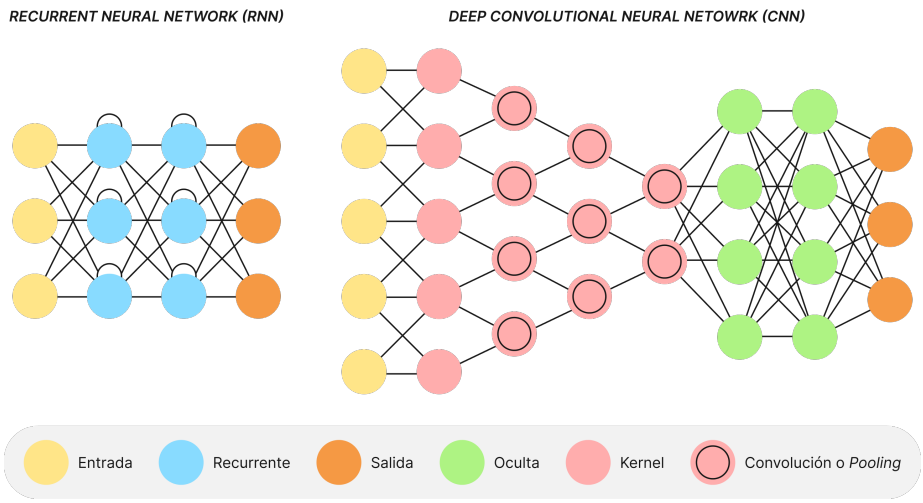

Figura 3.5: Estructura de una red RNN y CNN

En la figura 3.5, se puede observar la estrutura de los dos tipos mencionados. La red recurrente debe su nombre a las neuronas que están conectadas consigo mismas (neuronas en azul).

[18] Una de las redes neuronales en *unsupervised learning* más habladas en la literatura son las redes GAN (*Generative Adversarial Neural Networks*). Estas redes usan la estructura de discriminador-generador [16] para extraer información de los datos de entrada

[19] Como FNNN o perceptrón

Las redes neuronales convolucionales son más complejas y su particularidad, como veremos más en profundidad en el siguiente apartado, es la extracción de características y reducción de la dimensionalidad. Vemos que la capa de entrada (en amarillo) tiene 5 neuronas, y que en las siguientes capas, el número de neuronas se va reduciendo hasta llegar a dos en la capa justo anterior a la oculta (en verde). Esto se produce mediante la convolución o el *pooling*. Vamos a estudiar estas dos redes y sus términos con un poco de profundidad a continuación.

- **RNN**: Las Redes Recurrentes son redes que tuvieron su auge en investigación en los años 90. *Fausett* (1994) [13] las define como redes neuronales con conexiones *feedback* (retroalimentación). Ejemplos de estas redes son las redes BAM (*Bidirectional associative memory*), redes *Hopfield* y máquinas de *Boltzmann* [28].

  La característica principal de las redes recurrentes es que tienen un estado que mantienen entre distintas iteraciones de la red. Las neuronas no solo reciben información de la capa anterior si no que también reciben información de si mismas y de la iteración anterior [12]. Por esta razón, el orden con el que «alimentamos» datos a la red es importante y puede cambiar la salida generada de forma radical. Esta característica las hace muy útiles con formatos de datos como video o audio [20]. Su principal aplicación es el avance y completado de información, como por ejemplo el autocompletado de texto. En [37] podemos ver un ejemplo de una RNN aplicada para enseñar a la red a contar.

- **CNN**: Y llegamos a las redes convolucionales. Introducidas por *Le Cun* (1998) [23], estas redes son uno de los tópicos más hablados en la literatura actual en el mundo del *machine learning* y la inteligencia artificial. Estas redes se usan en muchísimas aplicaciones y dominios, sobre todo en procesamiento de video e imagen.

  Las redes convolucionales tienen una estructura muy diferente a cualquier otra red neuronal (figura 3.6). Se componen de capas de convolución, capas de *subsampling* y capas «*fully-connected*».

[20] Aunque también podemos tratar texto o imagen como concatenación de letras o píxeles en el tiempo

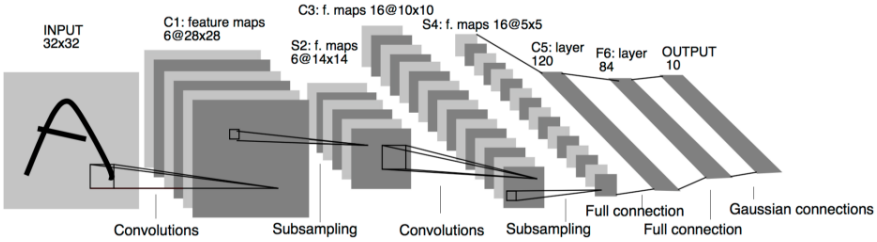

Figura 3.6: Estructura de una CNN. Figura de *Le Cun et al.*[25]

A continuación vamos a hablar de filtros o *kernels*, *pooling*, *padding*, *stride*, *ResNet*, entre otros términos.

1. **Extracción de características o convolución**: En esta fase se busca reducir la dimensionalidad de la entrada manteniendo las características espaciales de los datos. De esta forma no perdemos información y es más fácil procesarla en las siguiente capas. En redes neuronales tradicionales, transformaríamos una entrada (imagen o video) en un vector de dimensión uno. Al hacer esto perderíamos información de formas en la imágen, o de movimientos de objetos o personas en los videos. Por esa razón, la extracción de características de una imagen se hacía antes de forma manual en la fase de preparación de los datos.

   Para eliminar esta fase manual, las CNN actuan directamente en matrices de datos multidimensionales (o tensores [21]), en vez de actuar sobre vectores. Pero, ¿cómo extraemos información de un tensor?.

   Para extraer información de un tensor mantendiendo sus características espaciales vamos a usar *kernels* o filtros. Un filtro es un tensor con las mismas dimensiones que la entrada, pero con tamaños mucho menores. i.e.: si tenemos como entrada una imagen de $(200x200)$ pixeles, tendremos un filtro de dos dimensiones, por ejemplo, de $(3x3)$ pixeles. Si en vez de una imagen, estamos clasificando un video con 10 frames $(10x200x200)$ tendremos un filtro que por ejemplo puede tener el tamaño $(2x3x3)$, siendo 2 el número de frames.

   Una vez tenemos nuestro tensor de entrada y nuestro *kernel*, podemos iniciar el proceso de convolución. Para hacer esto, vamos a

[21] Un tensor es un objeto algebraico que representa un espacio vectorial. Puede mapear vectores y matrices multidimensionales [9]

«mover» nuestro *kernel* por nuestro tensor de entrada, generando un «mapa de caracterísitcas». Veamos un ejemplo:

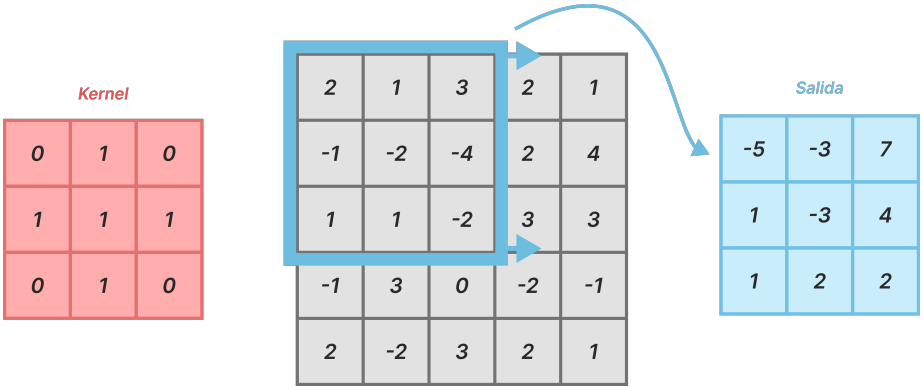

Figura 3.7: Ejemplo de convolución con *kernel* $3x3$ sin *strading* ni *padding*

Vemos en la figura 3.7 que tenemos un tensor bidimensional (i.e.una imagen) de tamaño $(5x5)$ con un *kernel* de tamaño $(3x3)$. El mapa de caracterísitcas resultante tiene un tamaño de $(3x3)$[22].

En este caso no hemos tenido ningún problema y hemos podido «encajar» nuestro *kernel* en la matriz perfectamente. Esto no siempre es así, ya que si queremos adaptarnos a cualquier tipo de entrada, es posible que tengamos tamaños cuyo *kernel* no «encaje» de forma perfecta. Si nuestra matriz fuera de $(4x4)$ en vez de $(5x5)$, tendríamos que para el tercer cálculo, nuestro *kernel* se situa fuera de la matriz en el lado derecho. Para solucionar este problema usamos *padding* o *stride*.

- El *padding* se usa para añadir información a los bordes de nuestro tensor y así mantener el tamaño de salida deseado. Podemos elegir añadirlo en todos los bordes o solo en algunos. El valor óptimo del *padding* es 0. Esto es porque toda información que añadamos a los bordes, «contamina» la información importante ubicada en el tensor de entrada [31].
- El *stride* por otro lado indica la cantidad de unidades que movemos nuestro *kernel* sobre nuestra entrada en cada iteración de la convolución. Valores comunes de stride son 2 o 3 [17]. Tenemos que tener cuidado aquí, porque con un

[22]Da la casualidad de que el tamaño de la salida es igual al tamaño del kernel, pero no es la norma. Si tuviéramos una entrada de $(5x7)$ obtendríamos una salida de $(3x5)$

stride demasiado alto, podemos saltarnos extracciones de características importantes de nuestro tensor.

Podemos calcular el tamaño de salida con la siguiente fórmula [31]:

$$\delta = \frac{D + 2p - K}{s} + 1$$

donde D es cada una de las dimensiones un vector de tamaño d con las dimensiones de nuestro tensor de entrada, K es un vector de tamaño d con las dimensiones del *kernel*, y p y s son los valores de *padding* y *stride* respectivamente. La salida será un número δ. Si el número es un entero, entonces la convolución será correcta.

Así en nuestro caso $\delta = [\frac{5+2\cdot0-3}{1} + 1, \frac{5+2\cdot0-3}{1} + 1] = [3, 3]$. Como vemos, en el proceso de convolución, «alteramos» las dimensiones del tensor de entrada con la finalidad de extraer las carcaterísitcas espaciales del mismo.

2. ***Pooling***: Muy similar a la fase de convolución, el pooling reduce el tamaño del mapa de caracterísitcas. El objetivo es disminuir la capacidad computacional necesaria para procesar gran cantidades de datos [8]. Hay dos tipos de *pooling* principales: *Max Pooling* y *Average Pooling*. El primero devuelve el valor máximo de la «porción» del tensor cubierta por el *kernel* mientras que el segundo devuelve la media de todos los valores en dicha «porción». *Max pooling* funciona normalmente mejor que *average pooling* porque no solo lleva a cabo la labor de reducción de la dimensionalidad, sino que también actua como mecanismo de eliminación de ruido. [23].

3. **Capa *fully connected***: Una vez hemos extraido las características necesarias de nuestro tensor de datos, hemos terminado con un tensor con dimensiones reducidas. Esta fase se ve muy bien gráficamente en la figura 3.5. Las capas de color rosa son las capas en las que realizamos la convolución y el *pooling*. El tensor resultante debe ahora sí, ser convertido en un vector (tensor de dimensión uno), ya que este es la entrada ahora de una capa *fully connected*. Este proceso se denomina *flatten*.

   Las capas *fully connected* se asemejan mucho a estructuras tradicionales de una red neuronal, con la diferencia de que las neuronas

[23] Denotar que lo importante del *kernel* en el *pooling* son sus dimensiones ya que no vamos a realizar operaciones con él

que forman esta capa aplican una transformación lineal del vector de entrada primero, seguido de una transformación «no lineal».
La salida de esta capa se calcula como

$$y_{jk}(x) = f\left(\sum_{i=1}^{n} w_{jk}x_i + w_{j0}\right)$$

donde w_0 es el *bias*, w nuestra matriz de pesos y x el vector de entrada.
El nombre de esta capa [24] se debe a que las neuronas están totalmente conectadas entre si entre las distintas capas (no entre neuronas de la misma capa ni consigo mismas) [20].
La salida de la capa *fully conneted* será la salida que produce la función de activación no lineal que se aplica tras las funciones lineales. Esta función puede ser una función *softmax* para clasificación multiclase o puede ser una sigmoide para otro tipo de salida. Por ejemplo, en este proyecto, se ha usado una *fully connected* con *softmax* para clasificar los signos y una *fully connected* con sigmoide para extraer un vector de poses de gestos en un video. Puedes avanzar a la sección «Aspectos relevantes» para más información.

Existen numerosas arquitecturas de redes neuronales convolucionales que han demostrado ser muy útiles en la resolución de distintos problemas. Algunas son: AlexNet [22], LeNet[24], VGGNet [41], GoogLeNet [45], ZFNet [48] o ResNet [18].

Todas estas no son ni mucho menos todas las redes neuronales del campo del *deep learning*. Cada pocos meses se publica un *paper* que muestra estructuras nuevas con oportunidades futuras impresionantes. En la actualidad, otro de los modelos de *deep learning* que más atención (nunca mejor dicho) genera, son los modelos *transformers* expuestos por primera vez en el famoso *paper* «*Attention is all you need*», parte de un libro de (2017) [47]. Estos modelos usan el llamado «mecanismo de atención» que da pesos distintos a las distintas partes de nuestro dato de entrada. Los principales campos en los que se usan los *transformers* son el campo del *natural planguage processing* (NLP) y el campo de *computer vision* (CV).

Nota del autor: Los *transformrs* se alejan un poco del tema principal de este proyecto, pero invito a leer el fantástico paper y descubrir

[24] También se suele llamar capa densa

sobre los modelos que usan esta red y que están revolucionando el mundo de la *deep learning* (i.e.: GPT-3 o DALL-E)

Técnicas y herramientas

Aspectos relevantes del desarrollo del proyecto

5.1. Carga de datos

Extracción de frames de videos

5.2. Tratamiento de los datos

Normalización

Data Augmentation

Transformación de datos a tensores

Batching

Subsampling

Eliminación de datos innecesarios

5.3. Red Neuronal ResNet

5.4. Estructura de la red convolucional

BatchNormalization

Dense layers

Capas de salida

Sigmoide

Softmax

5.5. Entrenamiento e hiperparametrización

Pipeline del proceo

Optimizer

Scheduler

Criterion

5.6. Salida de la red

Dos salidas en la misma red

Las salidas se controlan entre ellas

5.7. Comprobación de resultados

¿Cómo podemos saber si el resultado incorrecto se debe a la estructura de la red o a los datos introducidos?

5.8. Exportación del modelo

Formatos estándares

ONNX

5.9. Copresión del modelo

Consecuencias

Velocidad

Accuracy

Métodos de compresión

Quantization

Prunning

5.10. Estudio de escalabilidad del modelo

Vamos a comprobar hasta que punto la red neuronal es capaz de mantener la precisión de clasificación con el aumento de las etiquetas. Para todo lo anterior el número de etiquetas a clasificar han sido 8. Vamos a aumentar esto hasta 100 etiquetas.

En la siguiente tabla blabla

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] Ethem Alpaydin. *Machine learning*. MIT Press, 2021.

[2] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1):105–139, 1999.

[3] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[4] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[5] H Dv Block, BW Knight Jr, and Frank Rosenblatt. Analysis of a four-layer series-coupled perceptron. ii. *Reviews of Modern Physics*, 34(1):135, 1962.

[6] Steven Busuttil. Support vector machines. 2003.

[7] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):1–13, 2020.

[8] Yin Cui, Feng Zhou, Jiang Wang, Xiao Liu, Yuanqing Lin, and Serge Belongie. Kernel pooling for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2930, 2017.

[9] Colaboradores de. Tensor, Feb 2004.

[10] Chuong B Do and Serafim Batzoglou. What is the expectation maximization algorithm? *Nature biotechnology*, 26(8):897–899, 2008.

[11] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2):103–130, 1997.

[12] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[13] L. Fausett and L.V. Fausett. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall international editions. Prentice-Hall, 1994.

[14] Peter Flach. Performance evaluation in machine learning: the good, the bad, the ugly, and the way forward. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9808–9814, 2019.

[15] Ali Ghodsi. Dimensionality reduction a short tutorial. *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada*, 37(38):2006, 2006.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Nets*. Universite de Montréal, 2014.

[17] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. Dec 2015.

[19] Geoffrey Hinton, Simon Osindero, Max Welling, and Yee-Whye Teh. Unsupervised discovery of nonlinear structure using contrastive backpropagation. *Cognitive science*, 30(4):725–731, 2006.

[20] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[21] Tha Asimov Institute. the neural network zoo the asimov institute 2016, Sep 2016.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2012.

[23] Yann Le Cun, Lawrence D Jackel, Brian Boser, John S Denker, Hans Peter Graf, Isabelle Guyon, Don Henderson, Richard E Howard, and William Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46, 1989.

[24] Yann Lecun, L Eon Bottou, Yoshua Bengio, and Patrick Haaner Abstract|. Gradient-based learning applied to document recognition. *PROC. OF THE IEEE*, 2006.

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

[26] Pytorch main Mantainer. Pytorch developers guideline and design philosophy, 2022.

[27] Dastan Maulud and Adnan M Abdulazeez. A review on linear regression comprehensive in machine learning. *Journal of Applied Science and Technology Trends*, 1(4):140–147, 2020.

[28] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[29] Tom M Mitchell and Tom M Mitchell. *Machine learning*. Number 9. McGraw-hill New York, 1997.

[30] Kevin P Murphy et al. Naive bayes classifiers. *University of British Columbia*, 18(60):1–8, 2006.

[31] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[32] Suprianto Panjaitan, Muhammad Amin, Sri Lindawati, Ronal Watrianthos, Hengki Tamando Sihotang, Bosker Sinaga, et al. Implementation of apriori algorithm for analysis of consumer purchase patterns. In *Journal of Physics: Conference Series*, volume 1255, page 012057. IOP Publishing, 2019.

[33] J Quinlan. Building classification models: Id3 i c4. 5. *Dane udostepnione pod adresem: http://yoda. cis. temple. edu*, 8080, 1993.

[34] J Ross Quinlan et al. Bagging, boosting, and c4. 5. In *Aaai/Iaai, vol. 1*, pages 725–730, 1996.

[35] Shoba Ranganathan, Kenta Nakai, and Christian Schonbach. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*. Elsevier, 2018.

[36] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[37] Paul Rodriguez, Janet Wiles, and Jeffrey L Elman. A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40, 1999.

[38] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34, 1995.

[39] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[40] Seldon. Outlier detection and analysis methods - seldon, Jul 2021.

[41] Karen Simonyan and Andrew Zisserman. *Published as a conference paper at ICLR 2015 VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION*. 2015.

[42] Kristina P Sinaga and Miin-Shen Yang. Unsupervised k-means clustering algorithm. *IEEE access*, 8:80716–80727, 2020.

[43] Zixing Song, Xiangli Yang, Zenglin Xu, and Irwin King. Graph-based semi-supervised learning: A comprehensive review. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[44] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014.

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. *Going deeper with convolutions*. 2014.

[46] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[48] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks, 2013.

[49] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.