Índice general

Capítulo 1

Preliminares

Antes de poder hablar del Aprendizaje profundo, o de Redes Neuronales Convolucionales, es importante sentar las bases matemáticas y computacionales requeridas para su estudio. A continuación, se presentan algunas definiciones y resultados conocidos de temas tales como cálculo, álgebra y ciencias de la computación.

1.1. Resultados de Cálculo y Álgebra lineal

Definición 1.1 (Polinomio de Taylor)**.** *Sea f una función n veces diferenciable, y sean*

$$a_k = \frac{f^{(k)}(a)}{k!}.$$

Definimos el Polinomio de Taylor de grado n para f en a como

$$P_{n,a}(x) = a_0 + a_1(x-a) + \cdots + a_n(x-a)^n.$$

El polinomio $P_{n,a}(x)$ ha sido definido de manera que $P_{n,a}(a) = f^{(k)}(a)$ para $0 \leq k \leq n$.

Definición 1.2. *Sea f una función tal que $P_{n,a}(x)$ existe, definimos el término residual $R_{n,a}(x)$ como*

$$R_{n,a}(x) = f(x) - P_{n,a}(x). \tag{1.1}$$

Teorema 1.3 (Teorema de Taylor)**.** *Supóngase que* $f^{(1)}$, $f^{(2)}$, ..., $f^{(n+1)}$ *están definidos en* $[a, x]$. *Entonces*

$$R_{n,a}(x) = \frac{f^{n+1}(t)}{(n+1)!}(x-a)^{n+1} \tag{1.2}$$

para algún $t \in (a, x)$.

Definición 1.4. *Sea* x *una característica en* $\mathbb{R}^{h\times w\times d}$. *La vectorización de* x, *denotada* $X = vec(x)$ *es un vector en* $\mathbb{R}^{hwd}$ *tal que*

$$X_{(k-1)hw+(i-1)w+j} = x_{i,j,k}, \tag{1.3}$$

para todos $i = 1, 2, ..., h$, $j = 1, 2, ..., w$, $k = 1, 2, ..., d$.

explicar que es una característica (feature). Además, en la definición de convolución uso una imagen I, tal vez sea más consistente escribir una característica x.

Definición 1.5. *Una matriz* H *es llamada Circulante si es de la forma*

$$H = \begin{bmatrix} h_0 & h_1 & \cdots & h_{N-1} \\ h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & & \vdots \\ h_1 & h_2 & \cdots & h_0 \end{bmatrix} \tag{1.4}$$

ya que H *queda completamente determinada con la primera fila, es posible escribir* $H = circ(h_0, ..., h_{N-1})$.

Definición 1.6. *Una matriz* H *es llamada Doblemente Circulante por Bloques si es de la forma*

$$H = \begin{bmatrix} H_0 & H_1 & \cdots & H_{N-1} \\ H_{N-1} & H_0 & \cdots & H_{N-2} \\ \vdots & \vdots & & \vdots \\ H_1 & H_2 & \cdots & H_0 \end{bmatrix} = circ(H_0, ..., H_{N-1}) \tag{1.5}$$

dónde $H_i = circ(h_{i,0}, ..., h_{i,N-1})$.

Definición 1.7. *Sean A y B dos conjunto. El producto cartesiano $A \times B$ se define como*

$$A \times B \quad = \{(a, b) : a \in A \text{ y } b \in B\}. \tag{1.6}$$

De igual manera sean $A_1, ..., A_s$ son conjuntos, el producto cartesiano se define como

$$A_1 \times A_2 \times ... \times A_s = \{(a_1, ..., a_s) : a_i \in A_i, \quad i = 1, ..., s\}. \tag{1.7}$$

De modo natural, se define $A^n := A \times A \times ... \times A$ donde A se repite n veces.

A su vez, nos es conveniente destacar otra notación

Notación 1.8. *Sean $a_1, ..., a_n \in \mathbb{N}$. Definimos $\mathbb{R}^{a_1 \times a_2 \times ... \times a_s} := \mathbb{R}^{a_1} \times ... \times \mathbb{R}^{a_s}$*

Definición 1.9. *Sean $u, v \in \mathbb{R}^n$, el producto punto de u con v denotado como $u \cdot v$ es*

$$u_1 v_1 + u_2 + v_2 + ... + u_n v_n \tag{1.8}$$

donde u_i y v_i representan la i-ésima entrada de los vectores u y v respectivamente.

Definición 1.10. *Una función vectorial es una función de la forma $f : \mathbb{R} \to \mathbb{R}^n$.*

Definición 1.11. *La derivada f' de una función vectorial $f : \mathbb{R} \to \mathbb{R}^n$ se define como*

$$f'(t) = \lim_{h \to 0} \frac{f(t+h) - f(t)}{h \to 0}. \tag{1.9}$$

Sin embargo, en la práctica, no es necesario utilizar la definición, pues

Teorema 1.12. *La derivada f' de una función vectorial $f = (f_1, f_2, ..., f_n)$, con $f_i : \mathbb{R} \to \mathbb{R}$ es*

$$f'(t) = (f_1'(t), f_2'(t), ..., f_n'(t)). \tag{1.10}$$

Definición 1.13 (Gradiente)**.** *Sea $f : \mathbb{R}^n \to \mathbb{R}$. El gradiente de f es el vector conformado por las derivadas parciales en cada una de las n variables:*

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n} \right). \tag{1.11}$$

Proposición 1.14 (Regla de la cadena)**.** *Sea $f : \mathbb{R}^n \to \mathbb{R}$ y $v : \mathbb{R} \to R^n$. Es posible hallar la derivada de la composición.*

$$\frac{d}{dt}(f \circ g) = \nabla f \cdot v'(t) \tag{1.12}$$

Definición 1.15 (Notación O-grande (Big O))**.** *Sean f y g funciones $f, g : \mathbb{N} \to \mathbb{R}^+$. Decimos que $f(n) = \mathcal{O}(g(n))$ si existen enteros positivos c y n_0 tales que para cualquier entero $n \geq n_0$,*

$$f(n) \leq cg(n). \tag{1.13}$$

Cuando $f(n) = \mathcal{O}(n)$, decimos que $g(n)$ es una cota asintótica superior de $f(n)$,

1.2. Visión Computacional

El problema de clasificación de imágenes, pertenece a la rama de la visión computacional. La siguiente definición de imagen fue extraída del libro [1].

Definición 1.16. *Una imagen digital es una función $I : D \subset \mathbb{Z}^2 \to U$. Dónde el domino es un rectángulo con coordenadas enteras $D = [1, M] \cap \mathbb{Z} \times [1, N] \cap \mathbb{Z}$. La terna (x, y, u) se conoce como pixel.*

- *Cuando las imágenes son a escala de grises se tiene que $U = [0, 255] \cap \mathbb{Z}$. Es decir que $u \in U$ es un valor de 0 a 255.*
- *Si las imágenes son a color, consideramos $U = ([0, 255] \cap \mathbb{Z})^3$. Es decir que $u \in U$ es una terna de valores de 0 a 255. A cada entrada de la terna se conoce como un canal de color.*

Es posible representar una *imagen digital* I como un *Tensor*, de la siguiente manera.

$$T_{i,j,k} = u_k, \quad u = I(i, j) \tag{1.14}$$

Definir en algún punto lo que es un tensor

1.3. Inteligencia artificial y Aprendizaje profundo

El *Aprendizaje de Máquina* es un caso particular de la inteligencia artificial, en donde la máquina aprende de los datos proporcionados. Para entender este concepto es importante definir qué significa aprender [2].

Definición 1.17. *Se dice que un programa de computadora aprende de la experiencia E respecto a un conjunto de tareas T y medida de rendiminto P si el rendimiento en las tareas en T, medido con P mejora gracias a la experiencia E.*

Ya que aprender, implica mejorar el rendimiento en una tarea específica, sería bueno remarcar algunas de las tareas más comunes en el campo del aprendizaje automático.

1. **Clasificación.** Consiste en que el modelo reconozca que elementos pertenecen a ciertas clases, teniendo en cuenta sus características. Por ejemplo, distinguir perros de gatos, o en el caso de este proyecto podría ser diferenciar entre polen y polvo

2. **Regresión.** Aquí lo que se busca es usar la información existente para tener una predicción aproximada de algún escalar. Por ejemplo, predecir el precio de una casa, dadas sus características (ubicación, número de cuartos, antiguedad, etc).

3. **Agrupamiento (Clustering).** Este es un méto de aprendizaje no supervisado, que tiene como objetivo detectar similitudes entre las instancias, y separarlas en grupos de elementos similares entre sí.

1.3.1. Sobreajusto y Desajuste

En el aprendizaje automático, nuestro modelo aprende de un conjunto de entrenamiento $\mathcal{X}$. Sin embargo, una vez que nuestro modelo ha aprendido ¿Cómo podemos determinar que aprendió correctamente o que desempeñó de manera satisfactoria su tarea? La manera de determinar que nuestro modelo es bueno, es probando su capacidad de hacer predicciones con datos que no haya utilizado durante el entrenamiento. Para ello, es necesario un conjunto de imágenes nuevas $\mathcal{X}'$ al cual llamaremos conjunto de prueba.

El proceso de entrenamiento consiste en optimizar una función en nuestro conjunto $\mathcal{X}$, donde es posible encontrar una medida de error, la cual se conoce como *error de entrenamiento*. Sin embargo, nuestro objetivo es reducir el error en nunestro conjunto de prueba, el cuál se denomina *error de prueba*. El desempeño correcto de un modelo en el conjunto de prueba se conoce como *generalización* y por ello el error de entrenamiento también es conocido como *error de generalización*

Asumimos que los elementos en nuestros conjuntos de datos son *independientes* y además que el conjunto de entrenamiento y el de prueba están *identicamente distribuidos*, lo cual se conoce como las suposiciones IID. Con estas suposiciones y debido a la naturaleza del entrenamiento, se espera que el error de prueba se reduzca a la par que el error de entrenamiento, y lo más normal es que el error de prueba sea mayor que el de entrenamiento.

Sin embargo, el conjunto de funciones que puede adoptar un modelo, depende de la cantidad de parámetros que este tenga, lo cual se conoce como *capacidad*. Con una capacidad suficiente, siempre es posible hacer un mapeo perfecto entre entradas y etiquetas. Sin embargo, no es la función que se adapte mejor a los datos de entrenamiento la que es más conveniente, sino la que generalice mejor. Cuando la diferencia entre el error de entrenamiento y el error de generalización se acrescenta se dice que el modelo sufre de un *sobreajuste*. Cuando la capacidad de un modelo es muy reducida, puede ocurrir que no sea posible reducir el error de entrenamiento con lo que el modelo sufre de un *desajuste*.

1.3.2. Regularización

Cuando se entrena un modelo, son necesarias dos partes:

1. Optimización: En donde, se busca minimizar la función de costo.

2. Generalización: Es importante que las predicciones de nuestro modelo sean aplicables con datos que no hayan sido vistos en el entrenamiento.

Sin embargo, no es fácil conseguir ambos objetivos simultáneamente. En vez de priori-

zar alguno de los dos, la *regularización* pretende lograr que se satisfagan ambos de la mejor manera. Es decir, *la regularización es cualquier modificación en el algoritmo de entrenamiento, cuyo objetivo sea reducir el error de generalización, pero no el error de entrenamiento.*

Insertar imágenes con ejemplos Una de las formas más comunes de regularizar un modelo que aprende la función $f(x, \theta)$, es agregando una penalización $R(\theta)$ a la función de costo, considerando θ como los parámetros.

1.3.3. Función de activación.

Con la intención de ampliar el conjunto de funciones que un modelo puede predecir, en cada capa de nuestras redes neuronales, se compone una función no lineal denominada *función de activación*. En caso de no incluirla, el conjunto de funciones disponibles para nuestro modelo serían sólo funciones lineales, debido a que la composición de dos funciones lineales L_1 y L_2 da lugar a otra función lineal $L_1 \circ L_2$.

Existen muchos ejemplos de funciones no lineales. En un principio se usaban funciones suaves como la función *sigmoidal* $(1 + \exp(x))^{-1}$, la tangente hiperbólica $\tanh(x)$, o la función *Softplus* $\frac{1}{\beta}\log(1 + \exp(\beta x))$. Sin embargo, las tendencias modernas prefieren las funciones no suaves como es el caso de la más popular *ReLU* máx$(x, 0)$.

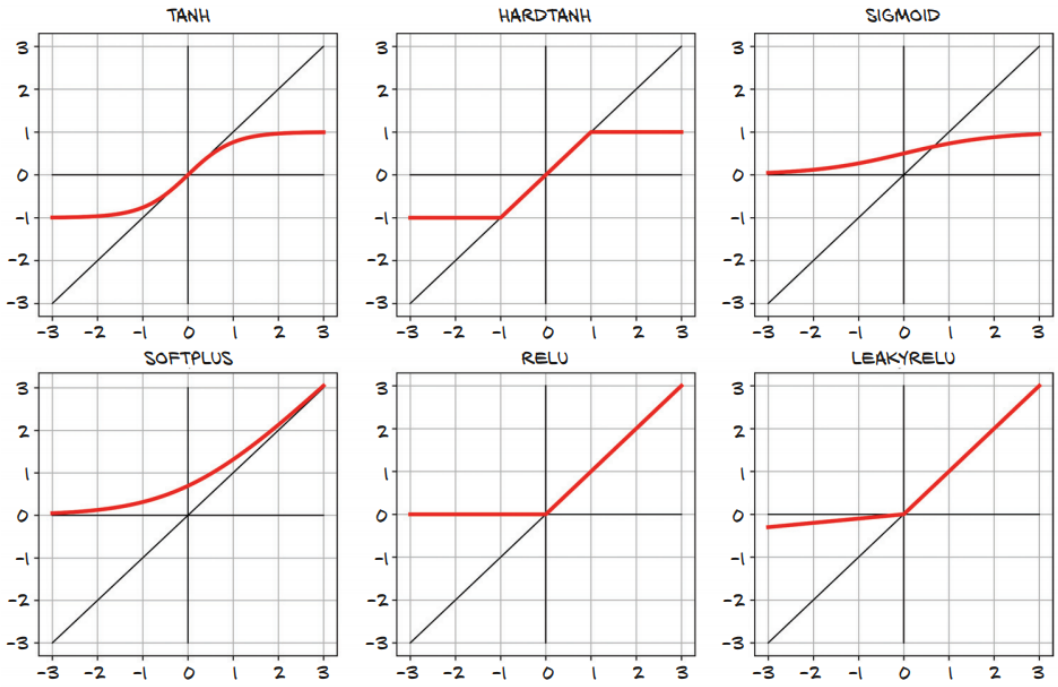


Figura 1.1: Usar una imagen propia

1.3.4. Transferencia de aprendizaje

La transferencia de aprendizaje en el campo del aprendizaje profundo, es un concepto motivado por la psicología, que se basa en usar conocimientos adquiridos para cierta tarea A, con el objetivo de aprender alguna tarea B. Por ejemplo, si una persona sabe bailar salsa, posiblemente pueda usar esos conocimientos para aprender a bailar bachata.

En la mayoría de los casos, los conjuntos de datos tienen una cantidad limitada de imágenes etiquetadas. Una posible solución a este problema, es usar *aprendizaje semisupervisado*, el cuál requiere algunas imágenes etiquetadas, pero también se pueden usar varias imágenes no etiquetadas. Sin embargo, incluso los conjuntos de datos no etiquetados, pueden ser insuficientes. Cuando se tienen conjuntos de datos muy pequeños, es posible apoyarse de modelos pre-entrenados con millones de imágenes, con la esperanza de que las características aprendidas anteriormente, sean de utilidad en el nuevo conjunto de datos.

1.3.5. Conjuntos desbalanceados

Cuando se enfrenta el problema de clasificación, digamos con m clases, uno esperaría que tengamos suficientes ejemplos de cada clase. Más aún, para que nuestro modelo no tenga ningún sesgo por ninguna clase, es preferible que en nuestro conjunto de entrenamiento, todas las clases tengan una cantidad similar de ejemplos. De lo contrario, nuestro modelo podría ser entrenado con demasiados elementos de una clase particular $mathcalC$, y tendería a clasificar muchos elementos en la clase $\mathcal{C}$. Además de esto, si nuestro conjunto de prueba también está desbalanceado, no se vería reflejado el sesgo en métricas comunes como la exactitud y la precisión.

Definición 1.18. *Sea $\mathcal{C}$ un conjunto de datos, cuyas clases son $\mathcal{C}_1, ..., \mathcal{C}_m$. Un conjunto de datos se dice balanceado con tolerancia de ϵ si*

$$1 - \epsilon \leq \frac{|C_i|}{|C_j|} \leq 1 + \epsilon, \quad i, j = 1, 2, ..., m. \tag{1.15}$$

Un conjunto es desbalanceado bajo la tolerancia ϵ si no es balanceado.

Para efectos prácticos, diremos que un conjunto balanceado con tolerancia $\epsilon = 0{,}1$

Ejemplo 1.19. *Supóngase ahora que dada una lista de características, se quisiese clasificar a las personas infectadas de COVID-19. En México el índice de positividad es del 17 % [].*
Por lo que si se toman todas las pruebas realizadas tendríamos la siguiente razón:

$$\frac{|C_{Neg}|}{|C_{Pos}|} = 4{,}8823 > 1{,}1. \tag{1.16}$$

El tamaño de la clase de pruebas negativas es casi 5 veces mayor al tamaño de la clase de pruebas positivas. Por consiguiente, estaríamos en presencia de un conjunto de datos desbalanceado.

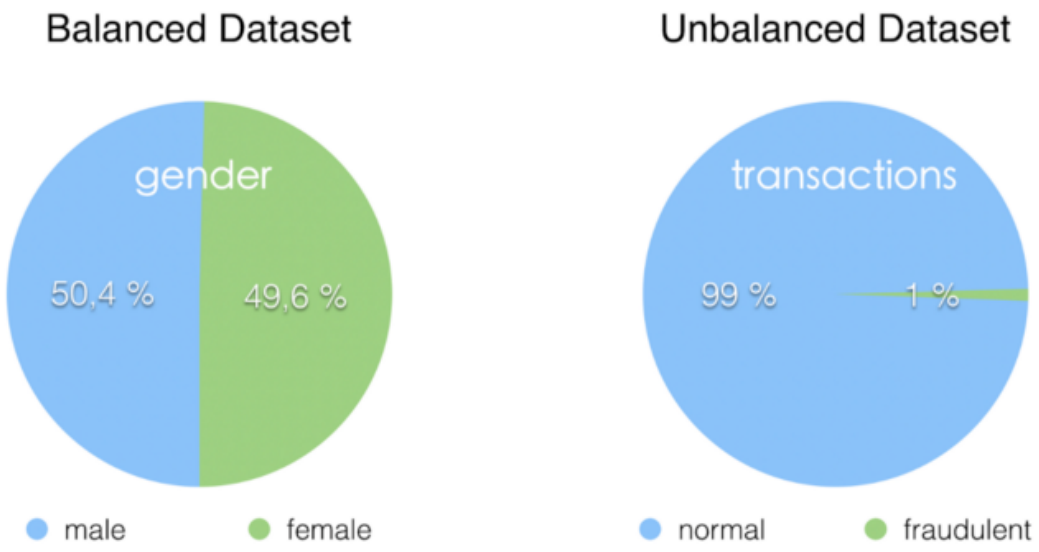


Figura 1.2: Usar una imagen propia

Afrontar un problema con un conjunto desbalanceado

La forma principal de lidiar con un conjunto desbalanceado es, valga la redundancia *balancear el conjunto*. Es decir, forzar al datset a tener clases cuyos tamaños coincidan. Para conseguir esto, es posible seguir distintas estrategias.

- **Submuestreo.** Consiste en tomar la classe mayoritaria (La clase con menor número de elementos) y eliminar algunas instancias, de modo que el conjunto quede balanceado. La manera más sencilla, es tomando elementos aleatorios del conjunto. Sin embargo, esto puede retirar instancias que contengan información esencial, por lo que sólo es recomendable en caso de tener un conjunto de datos muy grande.

  Otros algoritmos, heurísticos se basan en dos diferentes modelos de teoría de ruido. Algunos investigadores consideran que las instancias cercanas a los márgenes de clasificación de dos clases, son consideradas ruido. Por otro lado, algunos investigadores consideran que las instancias presentes en vecindades de varias etiquetas distintas, se pueden considerar como ruido. De modo que en lugar de remover elementos aleatorios, estas estrategias implican hacer el submuestreo tomando en cuenta únicamente

las instancias que no sean ruido.

- **Sobremuestreo.** Al igual que en el submuestreo, existe el sobremuestreo aleatorio, el cual se basa en crear copias idénticas de las instancias de la clase minoritaria, de modo que la nuestra clase minoritaria alcance la magnitud del resto de clases. Al aplicar este método, se incrementa el riesgo de sobreajustar el modelo.

  Otras algoritmos de sobremuestreo, generan instancias sintéticas para incremantar la cantidad de datos en una clase. Existen diversas formas de conseguir esto. Una posibilidad, es usando el aumento de datos únicamente en la clase minoritaria, permitiendo rotaciones, traslaciones y otras transformaciones. Existe también algoritmos como el VAE, SMOTE, MSMOTE.

Figura 1.3: Usar una imagen propia

Métricas distintas a la exactitud

Como hemos mencionado antes, la exactitud no debe ser la única métrica a considerar en un problema con conjunto de datos desbalanceado. La razón es simple: es posible obtener una muy buena exactitud sin realmente hacer predicciones de nuestra clase minoritaria.

Ejemplo 1.20. *Si tenemos un conjunto de datos con dos clases, tal que una clase* C_1 *representa el* $99\,\%$ *de las instancias, y la clase* C_2 *el* $1\,\%$ *restante. Si tomamos el clasificador*

$f : \mathcal{C} \to \{1, 2\}$, $f(x) = 1$. *Claramente nuestra exactitud sería del* $99\,\%$*, pero las predicciones de nuestra clase minoritaria aciertan un* $0\,\%$ *de las veces.*

Una forma de sobrellevar el efecto visto en el ejemplo 1.20 es analizando el desempeño del clasificador para cada clase por separado y luego hacer un promedio. En el caso de nuestro ejemplo todas las instancias que en verdad pertenecen a la clase 1, fueron clasificadas correctamente (1.0), y las instancias que pertenecen a la clase 2, fueron clasificadas todas incorrectamente (0.0). Con lo con esta nueva métrica tendríamos una puntuación de $\frac{1,0+0,0}{2} = 0{,}5$. En el capítulo 6 se definen formalmente las métricas relevantes para este trabajo. Sin embargo, las métricas más utilizadas para este tipo de problemas son las siguientes:

- Confussion matrix: De ésta matriz, se pueden obtener métricas útiles tales como Specificity, Sensitivity, Precision, Recall
- F1-score: La media armónica de precision y recall
- ROC curves
- Logloss
- Kappa: Exactitud de la clasificación, normalizada por el desbalance de clases en nuestros datos.

1.3.6. Redes completamente conectadas

1.3.7. Clasificación de imágenes

Para poder clasificar las imágenes, los algoritmos de inteligencia artificial extraen características importantes de una imagen. ¿Qué puede ser tomado en cuenta como importante? Podría pensarse que las esquinas, los bordes u otros detalles. Sin embargo, los algoritmos de aprenizaje automático suelen ser cajas negras, dónde las caraceterísticas relevantes de una imagen, a simple vista no parezcanr

Definición 1.21 (Característica)**.** *Una característica x es un elemento de $\mathbb{R}^{w\times h\times d}$, dónde $w, h \in \mathbb{N}$ representan las dimensiones espaciales y d la cantidad de canales.*

1.4. Notas del capítulo

1. Añadir la referencia: A Comprehensive Survey on Transfer Learning
2. Falta terminar la sección de transferencia de aprendizaje
3. Uso términos como exactitud y precisión, cuándo estos se definen en la última sección.
4. Añadir la referencia de la población de Hombres y Mujeres en el ejemplo 1 de conjuntos desbalanceados
5. Añadir referencia de positividad de Covid en México en el ejepmlo 2 de conjuntos desbalanceados.

Capítulo 2

Introducción a las redes neuronales convolucionales

La literatura de las redes neuronales data desde inserte fecha. Sin embargo, no fue hasta el 2015 con la introducción de las *redes neuronales convolucionales* (CNN por sus siglas en inglés) que la comunidad científica empezó a prestar especial atención. Para entender a profundidad a las CNNs, definiremos lo que es una convolución. Para un estudio más detallado es posible consultar [3, 4].

2.1. Convoluciones

En matemáticas, se suele referir a la convolución de dos funciones f y g.

Definición 2.1 (Convolución)**.** *Sean $f, g : \mathbb{R} \to \mathbb{R}$. La convolución de f con g, denotada como $f * g$ se define como:*

$$(f * g)(t) = \int_0^t f(\tau)g(t-\tau)d\tau$$

Ésta definición sin embargo no es muy práctica para los paradigmas computacionales, pues las limitaciones físicas nos obligan a traducir la integral como una suma de intervalos muy pequeños. Además, si tomamos f como una imagen, es necesario considerar más de una dimensión.

Definición 2.2 (Convolución de una imagen con un kernel)**.** *Sea I una imagen y K un kernel bidimensional. La convolución de I con K es la imagen $(I * K)$ definida como:*

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

En varias librerías de aprendizaje automático se implementa una correlación cruzada en lugar de una convolución. La única diferencia entre una correlación cruzada y una convolución es la orientación del Kernel. En este trabajo, adoptaremos la convención usual de referirnos a las correlaciones como convoluciones.

Definición 2.3. *Sea I una imagen y K un kernel bidimensional. La correlación cruzada de I con K es la imagen definida por*

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$

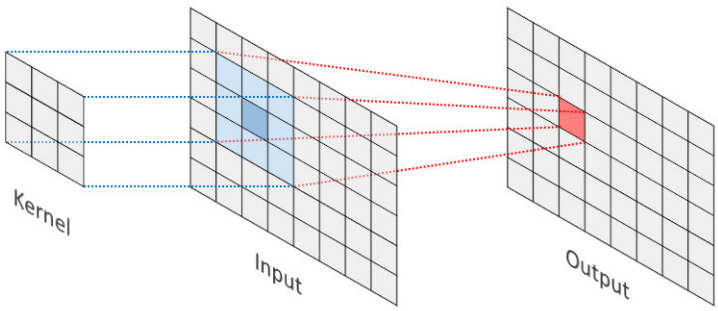


Figura 2.1: Representación gráfica de una convolución. Para cada posición del Kernel sobre la imagen, se realiza una multiplicación *entrada por entrada* del Kernel y la un conjunto de pixeles de la imagen, del mismo tamaño que el Kernel.

Cuando calculamos nuestra correlación cruzada, desplazamos el kenrel 1 pixel a la vez. Sin embargo, es posible aumentar la cantidad de pixeles que avanza el kernel en cada iteración. A esto se le conoce como tamaño de paso

2.1.1. Stride

Definición 2.4. *Sea x una característica, y K un kernel. Denotamos la correlación cruzada con tamaño de paso a como*

$$(I * K)|_a = (I * K)(i,j) = \sum_m \sum_n I(i+m+a*i, j+n+a*j)K(m,n) \tag{2.1}$$

Debido a nuestras convenciones en la notación, a la correlación cruzada con tamaño de paso a, también le llamaremos *convolución con tamaño de paso a*.

2.1.2. Convoluciones con múltiples canales

Las imágenes RGB suelen tener 3 canales, cada uno representa la intensidad de cada color: rojo, verde y azul respectivamente. Más aún, en las redes modernas, se suelen hallar características con múltiples canales, en cada capa. Es por esto, que es imperativo definir convoluciones para características con más de una de un canal. Antes de definir lo que es una convolución para múltiples canales, definiremos un concepto más general de *kernel*.

Definición 2.5. *Decimos que un kernel K multicanal tiene d_1 canales de entrada y d_2 canales de salida cuando $K \in \mathbb{R}^{k\times k\times d_1\times d_2}$.*

$$K = \begin{pmatrix} K_{1,1} & K_{1,2} & \cdots & K_{1,d_2} \\ K_{2,1} & K_{2,2} & \cdots & K_{2,d_2} \\ \vdots & \vdots & \ddots & \vdots \\ K_{d_1,1} & K_{d_1,2} & \cdots & K_{d_1,d_2} \end{pmatrix}$$

Cada $K_{i,j} \in \mathbb{R}^{k\times k}$ representa un subfiltro de K.

Cuando no exista ambiguedad, los kernel multicanal también pueden ser denotados como kernel.

Definición 2.6. *Sean $x \in \mathbb{R}^{h\times w\times d_2}$ una característica y $K \in \mathbb{R}^{k\times k\times d_1\times d_2}$ un kernel con d_1 canales de entrada y d_2 canales de salida. Denotamos la convolución 2D de x y K como:*

$$y_j := \sum_{i=1}^{d_1} K_{i,j} \otimes x_i, \quad j = 1, ..., d_2. \tag{2.2}$$

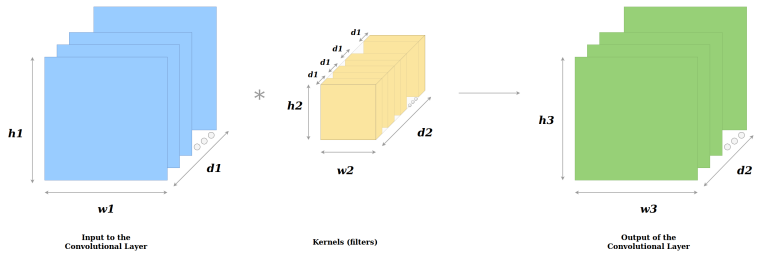


Figura 2.2: La convolución 2D puede ser usada con características de d_1 canales y devolver una característica de d_2 canales.

2.1.3. Padding

Debido a que la operación de convolución utiliza pixeles adyacentes, no es posible calcular los bordes de la imagen resultante. Suponiendo que se tiene una imagen de $n \times n$ y un kernel de $k \times k$ con k impar. Sólo es posible calcular la parte interior de la imagen resultante de tamaño $n - k + 1$.

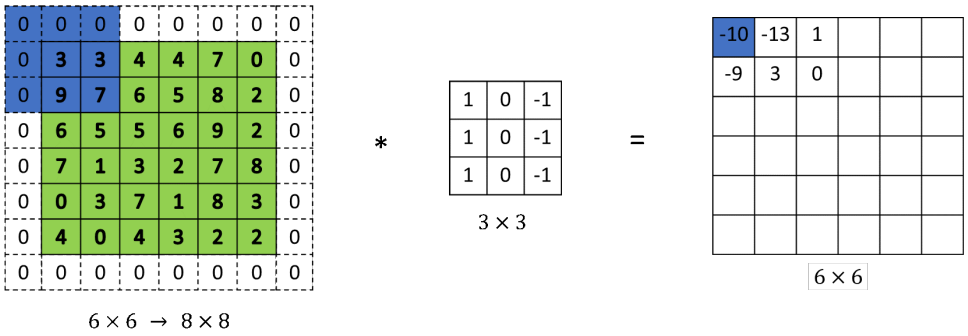


Figura 2.3: Usar una imagen propia

Para solucionar este problema, en vez de hacer la convolución con nuestra imagen original I, extendemos nuestra imagen hacia todos lados $\frac{k-1}{2}$ pixeles y realizamos nuestra nueva convolución con nuestra imagen extendida $\tilde{I}$. Nuestra nueva imagen $\tilde{I}$ contiene la misma información que la imagen original I, por tanto no importa la información que se añada en los nuevos pixeles. Sin embargo varias estrategias heurísticas han sido desarrolladas con el paso de los años:

- **Constant Padding.** Cuando se agrega un valor c en todo el borde. Cuando $c = 0$ se conoce como Zero Padding.

- **Zero Padding.** Una posible opción y de las más utilizadas, es agregar ceros en el borde. En [5] se descubrió que utilizando este tipo de padding codifica cierto grado de información de las posiciones absolutas. Efecto que no ocurre cuando no se utiliza ningún tipo de padding.

- **Reflection Padding.** Este padding se basa en reflejar los valores con respecto al borde [6].

- **Symmetric Padding.** Este padding es muy similar al reflection padding. En cada fila toma todos los valores y los invierte. Siendo esto lo que aparece en los bordes.

Definición 2.7. *Sea $x \in \mathbb{R}^{h\times w}$ una caracetrística. Sea $\tilde{x} \in R^{(h+2s)\times(w+2r)}$ con $s < h$ y $r < w$ nuestra caracteerística extendida. Definimos cada padding como sigue:*

1. *Zero Padding:*

$$\tilde{x}_{i,j} = \begin{cases} x_{i-s,j-r} & si \quad s < i < h+s \quad y \quad r < j < w+r \\ 0 & en\ otro\ caso \end{cases}. \tag{2.3}$$

2. *Reflection Padding:*

$$\tilde{x}_{i,j} = \begin{cases} x_{i-s,j-r} & si \quad s < i < h+s \quad y \quad r < j < w+r \\ x_{i-s,r-j+2} & si \quad j < r \\ x_{s-i+2,j-r} & si \quad i < s \\ x_{2h+s-i,j-r} & si \quad i > h+s \\ x_{i-s,2w+r-j} & si \quad j > w+r \end{cases}. \tag{2.4}$$

3. *Symmetric Padding:*

$$\tilde{x}_{i,j} = \begin{cases} x_{i-s,j-r} & si \quad s < i < h+s \quad y \quad r < j < w+r \\ x_{i-s,r-j+2} & si \quad j < r \\ x_{s-i+2,j-r} & si \quad i < s \\ x_{2h+s-i,j-r} & si \quad i > h+s \\ x_{i-s,2w+r-j} & si \quad j > w+r \end{cases} . \tag{2.5}$$

En [6] tras un estudio exhaustivo en el Image Net concluyen que el symmetric padding y el reflection padding obtienen peores resultados que el Zero padding.

2.1.4. Pooling

Una vez que nuestra red encuentra características en una imagen, es posible que algunas secciones relevantes sean más grandes que otras. Por eso mismo, se implementa el Downsampling, es decir reducir el tamaño de nuestras imágenes, para que nuestro kernel pueda encontrar cosas de distintos tamaños conforme nuestras capas van reduciendo el tamaño de la imagen. La pregunta que surge es, ¿Cómo reducimos el tamaño de una imagen? Sea como sea, siempre algo de información se pierde. Sin embargo existen métodos para resumir la información de varios pixeles en un sólo pixel. Algunos de estos métodos son

- **Max Pooling**. Dada una vecindad rectangular de pixeles, el maxpooling consiste en tomar el máximo valor dentro de esta vecindad.
- **Average Pooling**. Al igual que en el anterior caso, se toma una vecindad rectangular de pixeles. Sin embargo, en este caso se toma el promedio de todos los pixeles.
- **Una punto intermedio**. Considérese v el vector de pixeles que debe reducirse a un sólo pixel . En [7] se analiza el uso de diferentes poolings, y se obtienen distintas parametrizaciones para hallar el valor del pixel $f_P(v)$ que sintetiza la información de . Una posible parametrización es $f_P(v) = \left(\frac{1}{N}\sum_{i=1}^{N} v_i^P\right)^{\frac{1}{P}}$. Donde $P = 1$ es el average

y si $P \to \infty$ es el maxpooling. Por tanto, es posible utilizar otros valores de P para obtener poolings diferentes.

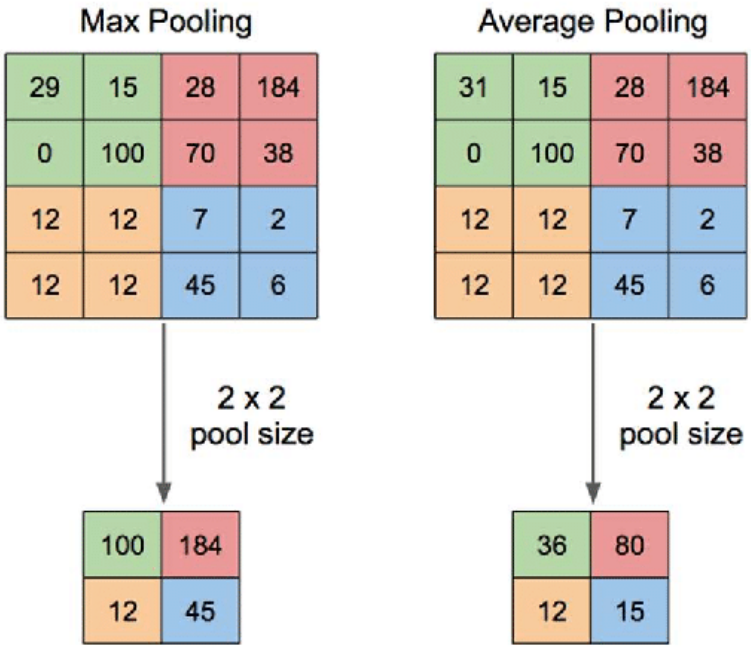


Figura 2.4: Ejemplo de maxpooling y averagepooling con regiones rectangulares de 2×2.

También es posible en vez de hacer un pooling hacer una convolución con cierto tamaño de paso, con la intención de reducir el tamaño de la imagen. Esto no ha remplazado al maxpooling pero en la literatura se observan resultados prometedores.

Global pooling layers

Además de hacer un downsampling, también es común que en algún punto de nuestra red, se reduzca una característica a un valor escalar, o en caso de una imagen con d canales es posible obtener un vector en $\mathbb{R}^d$ [3].

Definición 2.8. *Sea $x \in \mathbb{R}^{h \times w}$ una característica. El operador global average pooling, $P_g : \mathbb{R}^{hw} \to \mathbb{R}$ se define como:*

$$P_g(x) = \frac{1}{hw} \sum_{i=1}^{h} \sum_{j=1}^{w} x_{i,j}. \tag{2.6}$$

Más aún, sea $x \in \mathbb{R}^{h\times w\times d}$ *una característica. El operador global average pooling,* $P_g : \mathbb{R}^{hw} \to \mathbb{R}$ *se define como:*

$$P_g(x) = (P_g(x_1), ..., P_g(x_d)). \tag{2.7}$$

falta aclarar la notación de x_i, $x_{i,j}$ y $x_{i,j,k}$

2.1.5. Convolución como una operación lineal

Es posible ver una convolución como una multiplicación por una matriz poco densa, en donde varios elementos de la matriz están restringidos a ser iguales a otros. Para las convoluciones de una variable, tienen que ser una *matriz de toeplitz*. En lo que refiere a dos dimensiones, una convolución es equivalente a una *Doble Matriz Circulante por Bloques* [Definición 1.6]. Es decir, se tiene lo siguiente

Corolario 2.9. *Sea* I *una imagen, y* K *un kernel. Existe una matriz* $\hat{K}$

$$I * K = vec(I)\hat{K}. \tag{2.8}$$

Falta explicar que las convoluciones son invariantes a las traslaciones.

2.2. Redes Neuronales Convolucionales

Consideremos el problema de clasificación. Sea m la cantidad de clases posibles. Dados $y_1, y_2, ..., y_s \in \mathbb{R}^n$ y sus etiquetas $c_1, c_2, ..., c_s \in \mathbb{R}^m$, en donde la k-ésima entrada de c_i representa la probabilidad de que y_i pertenezca a la clase k.

Sean

$$Y_0 = [y_1, y_2, ..., y_s]^T \quad \text{y} \quad , \quad C = [c_1, ..., c_s]^T. \tag{2.9}$$

El objetivo es poder clasificar correctamente datos no etiquetados. Para ello se han desarrollado diferentes arquitecturas de redes neuronales, y en el 2015 las *Redes Neuronales Convolucionales* obtuvieron el primer lugar en el Image Net, consiguiendo resultados sin precedenes.

2.2.1. Descenso del Gradiente Estocástico

Ya que el problema de aprendizaje, es un problema de optimización, podemos recurrir a la teoría de optimización conocida. El algoritmo más común en el aprenizaje automático es el *descenso de gradiente estocástico* (SGD por sus siglas en inglés), el cuál es un algoritmo numérico para encontrar mínimos locales de una función. La intuición se basa en que si se tiene un punto $x \in D_f$, la dirección en dónde más decrece es $-\nabla f$. Éste método consiste en dos pasos

1. Se selecciona un punto inicial x_0 en el dominio.

2. Se actualiza nuestro valor $x_{i+1} = x_i - \alpha \nabla f(x_i)$ para $i = 1, ...N$

donde N es el número de iteraciones totales y α es un escalar, el cuál se conoce como ratio de aprendizaje (learning rate). Es importante seleccionar un ratio de aprendizaje apropiado, pues en caso de seleccionar uno muy grande, el algoritmo podría diverger, por el contrario si se selecciona uno muy pequeño, entonces el entrenamiento podría ser demasiado lento.

Para calcular el gradiente de manera exacta, es necesario utilizar todo el conjunto de datos, lo cuál puede traducirse a tiempos elevados de cómputo. Es por eso, que en la práctica, se utiliza únicamente un subconjunto aleatorio de datos, calculando así, una aproximación al gradiente ∇J de nuestra funcion de costo. Debido a este artificio que reduce el tiempo de entrenamiento, es que nuestro algoritmo es considerado estocástico.

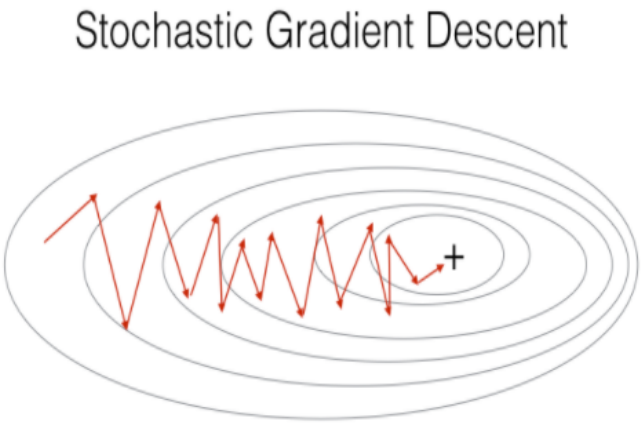


Figura 2.5: Seleccionar su propia imagen

2.2.2. Propagación hacia atrás

Ahora que se tiene un algoritmo para optimizar nuestra función de costo, la pregunta natural es ¿Cómo obtenemos el gradiente? Siendo la propagación hacia adelante de una red, una función tan complicada y con tantos parámetros, es muy difícil calcular de manera analítica nuestro gradiente. Por buena suerte para nosotros, existe una técnica conocida como *propagación hacia atrás*, encargada de calcular numéricamente el gradiente de la función de costo.

2.2.3. Normalización por Lotes

Entre las téncicas de regularización más utilizadas se encuentra la *normalización por lotes* (BN por sus siglas en inglés). En el 2015 en un paper de google [8] se describe esta técnica y se constata que tiene múltiples beneficios tales como incrementarl el learning rate, conseguir que el entrenamiento sea más independiente de la inicialización y además actuar como regularizador.

Definición 2.10. *Sea* $\mathcal{B} = \{x^{(1)}, x^{(2)}, \cdot\cdot, x^{(m)}\}$ *un lote de características. La normalización*

por lotes de $\mathcal{B}$ se define como:

$$B(x^{(i)};\gamma,\beta) := \frac{\gamma(x^{(i)-\mu})}{\sigma} + \beta, \quad i = 1,2...,m, \tag{2.10}$$

donde $\mu := \sum_{i=1}^{m} x^{(i)}/m$ *y* $\sigma^2 = \sum_{i=1}^{m}(x^{(i)} - \mu)^2/m$.

Los parámetros γ y β usualmente se aprenden en la optimización. Cuando se usa la normalización por lotes, no es necesario agregar biases pues son calculados explícitamente a través de β.

Aquí va la justificación de por qué usar convoluciones

Definición 2.11. *Consideremos el problema de clasificación (2.9). La propagación hacia adelante de una CNN está determinada por el siguiente esquema recursivo*

$$Y_{i+1} = \sigma(Y_i * K_i + b_i). \tag{2.11}$$

donde σ *es una función de activación,* K_i *representa un Kernel y* $b_i \in \mathbb{R}^n$ *es el bias.*

Imagen de una Lee Net o AlexNet para entender una arquitectura simple.

2.2.4. Desvanecimiento del gradiente

2.2.5. CNNs en el estado del arte

En 2012 en el ImageNet ocurrió algo sin precedentes en dicha competencia. El ganador del concurso se había llevado el primer lugar absoluto, obteniendo una ventaja de 10.9 % de error por debajo del segundo lugar. Es así como la *AlexNet* [9] consiguió reputación para las CNNs. Para 2013, la red ganadora del ImageNet(2013) fue la Zf-net [10], la cuál era una modificación de los hiperparámetros de la AlexNet.

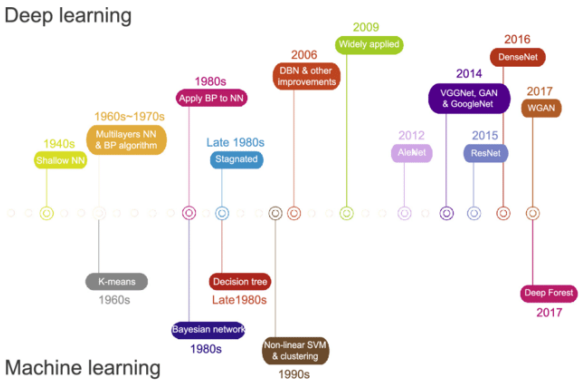


Figura 2.6: Crear imagen propia

2.3. ResNet

Las *Redes Neuronales Residuales* fueron presentadas en 2015 por Kaiming He et. al. en [11]. Uno de los mayores retos en el diseño de redes profundas, es el *desvanecimiento del gradiente*. Es decir, por la naturaleza del backpropagation, las redes muy profundas, implican el cálculo de gradientes con entradas muy pequeñas, y debido a las limitaciones del aritmética de punto flotante, ésto provoca que el gradiente se desvanezca. Es decir, $\|\nabla_\theta L\| = 0$ dónde L es la función de pérdida y θ son los parámetros.

Antes del 2015 existían dificultades para alcanzar redes con profundidades mayores a 20 capas. Para solucionar este problema, se crearon las *conexiones de salto*.

Definición 2.12. *Consideremos el problema de clasificación (2.9). La propagación hacia adelante de la ResNet está determinada por el siguiente esquema recursivo*

$$Y_{i+1} = Y_i + \sigma(Y_i * K_i + b_i). \tag{2.12}$$

donde σ es una función de activación, K_i representa un Kernel y $b_i \in \mathbb{R}^n$ es el bias.

Ya que las matrices

2.3.1. Estabilidad en las redes Neuronales

En clasificación de imágenes, una red debe ser robusta contra el ruido, o pequeñas modificaciones en la imagen. La salida debe ser continua con respecto a la entrada.

Definición 2.13. *Supóngase f la propagación hacia adelante de una red neuronal. Sea x una imagen y $\hat{x}$ la misma imagen perturbada. Decimos que f es estable cuando*

$$\|f(x) - f(\hat{x})\| < \epsilon \tag{2.13}$$

donde epsilon es muy pequeño

En [12] se agrega un término $h \in \mathbb{R}$ a la ResNet con la intención de añadirle estabilidad a la red

$$Y_{i+1} = Y_i + h\sigma(Y_i K_i + b_i). \tag{2.14}$$

2.4. Notas del capítulo

1. Añadir la definición de stride en la sección de convoluciones
2. El capítulo de ResNet podría llamarse Redes Neuronales convolucionales
3. No estoy seguro de si dedicarle un capítulo completo al desvanecimiento de gradiente, o sólo mencionarlo en el capítulo de ResNets. Por lo pronto, lo menciono en el capítulo de resNets
4. Debo mencionar lo que significa el gradiente en los preliminares
5. Cuando definimos el problema de clasificación, decimos que $y_i \in \mathbb{R}^n$. Me gustaría que pudiese hablar de y_i como imágenes, es decir $y_i \in \mathbb{R}n_1 \times n_2 \times d$.
6. En preliminares falta escribir un poco de visión computacional. ¿Qué es una imagen? ¿Qué significa perturbar una imagen o meterle ruido?
7. Traducir Batch Normalization
8. Aún falta definir matemáticamente el symmetric padding

Capítulo 3

Ecuaciones diferenciales

Una ecuación diferencial se representa de la siguiente manera:

$$\frac{dy}{dx} = f(x, y(x)) \tag{3.1}$$

Donde $f : \mathbb{R}^m \to \mathbb{R}^n$. En caso de que no se genere ambiguedad, es posible escribir y en lugar de $y(x)$ y y' para referirse a la derivada de y con respecto a x:

$$y' = f(x, y) \tag{3.2}$$

3.1. Problemas de valor inicial

Considérese el siguiente problema de valor inicial (IVP por sus siglas en inglés)

$$y'(t) = f(t, y), \quad t > 0 \tag{3.3}$$

$$y(t_0) = \nu, \tag{3.4}$$

donde $t \in [t_0, t_f]$.

3.1.1. Existencia, unicidad y continuidad de soluciones

Teorema 3.1. *Sea $f : [a_1, b_1] \times [a_2, b_2]$ una una función Lipschitz continua en la segunda entrada y $a_2 < y_a < b_2$. Entonces existe un $c \in [a_1, b_1]$ tal que el IVP*

$$\begin{cases} y' = f(t, y) \\ y(a_1) = y_a \\ t \in [a_1, c] \end{cases} \tag{3.5}$$

tiene exactamente una solución $y(t)$. Aún más, si f es Lipschitz continua en $[a_1, b_1] \times (-\infty, \infty)$, entonces existe exactamente una solución en $[a_1, b_1]$

3.1.2. Método de Euler

La manera más elemental para resolver el IVP (3.3)- (3.4) es con el método de Euler. Particionemos el intervalo $[t_0, t_f]$ en n partes iguales, obteniendo así la sucesión

$$t_0, t_0 + h, t_0 + 2h, ..., t_0 + nh,$$

donde

$$h = \frac{t_f - t_0}{n}.$$

Denotaremos $t_i = t_0 + ih$ y $y_i = y(t_i)$. De modo que tenemos la siguiente sucesión:

$$t_0 < t_1 < t_2 < \cdots < t_n = t_f. \tag{3.6}$$

Lo que haremos será aproximar las soluciones y_i, y a estas aproximaciones les llamaremos u_i, teniendo en cuenta que $u_0 = y_0$. Para ello, requerimos la expansión de Taylor de $y(t+h)$.

$$y(t + h) = y(t) + hy'(t) + R_1(t), \tag{3.7}$$

donde $R_1(t)$ es el error de truncamiento local. Sustituyendo $y'(t) = f(t, y(t))$, obtenemos

$$y(t + h) = y(t) + hf(t, y(t)) + R_1(t). \tag{3.8}$$

Sustituyendo $t = t_i$ con $i < n$ obtenemos que

$$y(t_i + h) = y(t_i) + hf(t_i, y(t_i)) + R_1(t_i) \tag{3.9}$$

. Por lo tanto, substrayendo el error de truncamiento local, podemos obtener una sucesión que aproxime nuestra solución $y(t)$:

$$u_{i+1} = u_i + hf(t_i, u_i), \quad u_0 = y_0. \tag{3.10}$$

Error de truncamiento local

En general, el error de truncamiento local de un método numérico, es el error generado a partir de una iteración.

Asumiendo que en la identidad (3.7), $y(t)$ es doblemente diferenciable en el intervalo (t_0, t_f), ocurre que

$$R_1(t) = \frac{1}{2!}h^2 y''(\xi), \quad \xi \in (t, t+h) \tag{3.11}$$

Error de truncamiento global

El error de truncamiento global de un método numérico, es el error generado por multiples iteraciones. En este caso, asumiremos una cota superior mh^2 para la norma del error de truncamiento local. Además asumimos Lipschitz continuidad para f, cuya constante de Lipschitz es L.

Denotemos a_i y b_i como los siguientes errores:

$$\alpha_i = y_i - u_i \tag{3.12}$$

$$\beta_i = f(t_i, y_i) - f(t_i, u_i) \tag{3.13}$$

Debido a la Lipschitz continuidad, debe pasar que

$$\|f(t_i, y_i) - f(t_i, u_i)\| \leq L\|y_i - u_i\|$$

$$\Rightarrow \|\beta_i\| \leq L\|\alpha_i\|$$

Falta terminar de escribir esta prueba

Definición 3.2. *Un método numérico se dice que es de orden ρ cuando $|u_n - y_n| = \mathcal{O}(h^\rho)$.*

Escribir la notación bigO en los preliminares

3.2. Estabilidad

3.3. Métodos de Runge-Kutta

Es posible generalizar el método de Euler, evaluando la derivada multiples veces en un paso. Esta idea se le atribuye a Runge [13]. y en 1901 Kutta contribyó a las ideas. Dando paso a métodos de orden 4 y el primer método de orden 5, denominados métodos de Runge-Kutta.

3.3.1. Método de Euler modificado

El siguiente método, es un ejemplo de un método de Runge-Kuta, el cuál usaremos para desarrollar las ideas tras los métodos generales.

$$\begin{aligned} k_1 &= f(t_n, u_n), \\ k_2 &= f(t_n + \tfrac{1}{2}h, u_n + \tfrac{1}{2}hk_1), \\ u_{n+1} &= u_n + hk_2, \\ t_{n+1} &= t_n + h. \end{aligned}$$

Este es un método de segundo orden. Falta hablar de qué significa que algo sea de orden n en algún punto del capítulo. Además falta cambiar la notación, porque en algunos casos n es un subíndice y en algunos casos es el número total de etapas

3.3.2. Método general

El método de Runge-Kutta general con s etapas se puede definir usando s^2+2s números

$$a_{i,j}, \quad i,j = 1,2,...,n. \qquad b_i, c_i, \quad i = 1,2,...,n, \tag{3.14}$$

usando el siguiente esquema

$$u_{n+1} = u_n + h\sum_{i=1}^{s} b_i k_i. \tag{3.15}$$

En donde la sucesión $\{k_i\}$ es calculada usando la función f:

$$k_i = f\left(t_n + c_i h, u_n + h\sum_{i=1}^{s} a_{i,j}k_i\right), \quad i = 1, 2, ..., s. \tag{3.16}$$

Es posible determinar si el método es explícito o implícito, basándose en la matriz $A = \{a_{i,j}\}$. En caso de que sea una matriz triangular inferior, con todos los elementos de la diagonal iguales a cero ($a_{i,j} = 0$ para $j = i, i+1, ..., s$) el esquema es explícito.

Las siguientes relaciones, son conocidas como condiciones de orden

$$\sum_{i=1}^{s} b_i = 1, \quad \sum_{i,j=1}^{s} b_i a_{i,j} = \frac{1}{2}, \quad \sum_{i,j,k=1}^{s} b_i a_{i,j} a_{j,k} = \frac{1}{6}, \quad \sum_{i,j,k=1}^{s} b_i a_{i,j} a_{i,k} = \frac{1}{3} \tag{3.17}$$

y aseguran esquemas de al menos orden 3.

3.3.3. Runge Kutta de orden 4 (RK4)

Uno de los métodos más conocidos es el de orden 4

$$u_{i+1} = u_i + \frac{h}{6}(s_1 + 2s_2 + 2s_3 + s_4), \tag{3.18}$$

donde

$$\begin{aligned} s_1 &= f(t_i, w_i) \\ s_2 &= f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}s_1\right) \\ s_3 &= f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}s_2\right) \\ s_4 &= f\left(t_i + h, w_i + hs_3\right). \end{aligned}$$

. La simlicidad de este método, hace que sea muy fácil de programar, y al ser de orden 4, se prefiere por sobre otos métodos como Euler, o el Trapezoide. Como se puede apreciar, RK4 es un método de orden 4 y además tiene 4 etapas (i.e. $s = 4$). Sin embargo, no se conoce cuántas etapas son necesarias para esquemas de órdenes mayores a 8.

3.3.4. Métodos de Runge Kutta simplécticos

Definición 3.3. *Un esquema de Runge Kutta se dice simpléctico si satisface la relación*

$$b_i a_{i,j} + b_j a_{j,i} - b_i b_j = 0, \quad i, j = 1, ..., s. \tag{3.19}$$

Falta explicar la importancia de la propiedad simplectica

3.3.5. Métodos implícitos

Los métodos numéricos analizados hasta este momento se conocen como métodos explícitos, debido a que es posible calcular u_{i+1} conociendo el valor u_i.

Euler hacia atrás

Una versión implícita de Euler se conoce como Euler hacia atrás.

$$u_0 = y_0 \tag{3.20}$$

$$u_{i+1} = u_i + hf(t_{i+1}, u_{i+1}) \tag{3.21}$$

Difiere del método de Euler tradicional en que la "pendiente"que se usa involucra u_{i+1}. No sé si escribir la versión intuitiva de Euler, para que tenga sentido este término de "pendiente".

Capítulo 4

Teoría de control óptimo

Cuando hablamos de sistemas, nos referimos a un proceso que cambia de estado conforme pasa el tiempo. En nuestro caso pensemos en un sistema dinámico definido en el intervalo $[0, T]$ como

$$\dot{x} = f(x(t), t), \quad x(0) = x_0, \tag{4.1}$$

donde $x(t)$, conocida como la *variable de estado* puede representar la producción de alimento en un tiempo t, el dinero recaudado en un tiempo t, o en general el estado de un proceso en un tiempo t. En logística y física, existen variables que podemos controlar, como la cantidad de publicidad, o la reinversión de capital cada cierto momento. Con lo que podemos extender nuestro sistema (4.1) al siguiente:

$$\dot{x} = f(x(t), u(t), t), \quad x(0) = x_0, \tag{4.2}$$

donde la variable $u(t)$ es denominada *variable de control*. Conociendo la variable de control, es posible determinar la solución para x en el sistema (4.2). El problema de control óptimo reside en maximizar la *función objetivo*, definida como

$$J = \int_0^T F(x(t), u(t), t)dt + S[x(T), T]. \tag{4.3}$$

Donde F puede es conocida como cómo es conocida y la función S determina el traducir salvage en el estado final $x(T)$ en el tiempo T. Se denotará al conjunto de calores posibles de $u(t)$ como $\Omega(t)$.

Definición 4.1. *El problema de control óptimo es encontrar un valor admisible u^* tal que*

$$u^* = \max_{u(t)\in\Omega(t)} \left\{ \int_0^T F(x,u,t)dt + S(x(T),T) \right\} \tag{4.4}$$

con la condición

$$\dot{x} = f(x,u,t), \quad x(0) = 0. \tag{4.5}$$

La trayectoria óptima denotada como x^* es la trayectoria que se obtiene cuando $u = u^*$.

4.1. La ecuación de Hamilton-Jacobi-Bellman

Sea $V : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ una función definida por

$$V(x,t) = \max_{u(s)\in\Omega(s)} \left\{ \int_t^T F(x(s),u(s),s)ds + S(x(T),T) \right\} \tag{4.6}$$

donde $s \geq t$. Sea δt un diminuto incremento en el tiempo, nótese que

$$\begin{aligned} V(x,t) - V(x(t+\delta t),t+\delta t) = & \max_{u(s)\in\Omega(s)} \left\{ \int_t^T F(x(s),u(s),s)ds + S(x(T),T) \right\} \\ & - \max_{u(s)\in\Omega(s)} \left\{ \int_{t+\delta}^T F(x(s),u(s),s)ds + S(x(T),T) \right\} \end{aligned}$$

Lo cuál, gracias al principio de optimalidad [14] se tiene que

$$V(x(t),t) - V(x(t+\delta t),t+\delta t) = \max_{u(\tau)\in\Omega(\tau)} \left\{ \int_t^{t+\delta t} F(x(\tau),u(\tau),\tau) \right\}, \tag{4.7}$$

y por consiguiente

$$V(x(t),t) = \max_{u(\tau)\in\Omega(\tau)} \left\{ \int_t^{t+\delta t} F(x(\tau),u(\tau),\tau) + V(x(t+\delta t),t+\delta t) \right\}. \tag{4.8}$$

Debido a que F es una función continua, la integral en (4.8) es igual a $F(x,u,t)\delta t$. Por consiguiente:

$$V(x,t) = \max_{u\in\Omega(t)} F(x,u,t)\delta t + V[x(t+\delta t),t+\delta t] + o(\delta t) \tag{4.9}$$

falta explicar lo que es $o(\delta t)$. Podría ser lo mismo que $O(\delta t)$ pero aún no estoy seguro. El libro dice que por definición $\lim_{\delta t\to 0} \frac{o(\delta t)}{\delta t} = 0$. Asumiendo que nuestra función V es

continuamente diferenciable en ambas entradas podemos hacer uso de la expansión de Taylor de V en δt:

$$V[x(t+\delta t), t+\delta t] = V(x,t) + [V_x(x,t)\dot{x} + V_t(x,t)]\delta t + o(\delta t) \tag{4.10}$$

Sustituyendo (4.2) en (4.9) obtenemos que

$$V(x,t) = \max_{u\in\Omega(t)} \{F(x,u,t)\delta t + V(x,t) + V_x(x,t)f(x,u,t)\delta t + V_t(x,t)\delta t\} + o(\delta t) \tag{4.11}$$

Cancelando $V(x,t)$ de ambos lados y dividiendo entre δt se deduce que

$$0 = \max_{u\in\Omega(t)} \{F(x,u,t) + V_x(x,t)f(x,u,t) + V_t(x,t)\} + \frac{o(\delta t)}{\delta t}, \tag{4.12}$$

con lo que si $\delta t \to 0$ se tiene que

$$0 = \max_{u\in\Omega(t)} \{F(x,u,t) + V_x(x,t)f(x,u,t) + V_t(x,t)\} \tag{4.13}$$

dónde si se evalúa $t = T$ en V se obtiene que

$$V(x,T) = S(x,T). \tag{4.14}$$

Cuando nuestra función V es aplicada en el estado óptimo $x*$ obtenemos un nuevo vetor

Definición 4.2. *(Vector de retorno marginal) Sea V la función definida en (4.6), el vector de retorno marginal $\lambda(t)$ se define como*

$$\lambda(t) = V_x(x^*(t),t) := V_x(x,t)|_{x=x^*} \tag{4.15}$$

A continuación describiremos el concepto de Hamiltoniano.

Definición 4.3. *Considérese el sistema (4.4)-(4.5). Diremos que el hamiltoniano $H : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ del sistema como*

$$H(x,u,\lambda,t) = F(x,u,t) + \lambda f(x,u,t). \tag{4.16}$$

La ecuación (4.13) puede reescribirse como

$$\max_{u\in\Omega(t)} [H(x,u,V_x,t) + V_t] = 0 \tag{4.17}$$

la cuál se conoce como *Ecuación de Hamilton-Jacobi-Bellman*(HJB). Más aún, ya que V_t no depende de u es posible retirar el V_t del operador máx.

$$\max_{u\in\Omega(t)}[H(x,u,V_x,t)]+V_t=0 \tag{4.18}$$

Gracias a la definición de nuestro control óptimo u^*, cualquier $u\in\Omega(t)$, satisface que

$$H[x^*(t),u^*(t),\lambda(t),t]+V_t(x^*(t),t)\geq H[x^*(t),u(t),\lambda(t),t]+V_t(x^*(t),t) \tag{4.19}$$

$$\Rightarrow H[x^*(t),u^*(t),\lambda(t),t]\geq H[x^*(t),u(t),\lambda(t),t] \tag{4.20}$$

4.2. Ecuación adjunta

El lado izquierdo de (4.18) se maximiza con $x=x^*$ y $u=u^*$ y el valor máximo que alcanza es cero. Justificar esta parte. Sea $x(t)=x^*(t)+\delta x(t)$, un camino distinto al camino óptimo. Sea $t\in[0,T]$ se tiene que

$$0=H[x^*(t),u^*(t),V_x(x^*(t),t),t]+V_x(x^*(t),t) \tag{4.21}$$

$$\geq H[x(t),u^*(t),V_x(x(t),t),t]+V_x(x(t),t). \tag{4.22}$$

también justificar esta parte. En teoría, tiene lógica porque x^* es el camino óptimo, pero hay que analizarlo bien.

4.3. Principio del Máximo

4.4. Notas del capítulo

1. Añadir el libro de Suresh P. Sethi. Optimal Control Theory a las referencias.

2. De la ecuación 4.7 a la ecuación 4.8 no tengo muy claro como se llega. Falta definir anteriormente qué es el principio de optimalidad y describir claramente por qué aplica en este caso.

3. Aún falta justificar varias cositas de control óptimo. Además, la redacción es casi idéntica al libro.

Capítulo 5

Relación entre las ecuaciones diferenciales y las ResNets

5.1. Notas del capítulo

1. -

Capítulo 6

Resultados

6.1. Descripción del problema

Para nuestra fase experimental se utilizaron dos bases de datos. Como punto comparativo al actual estado del arte, se utilizó el CIFAR10 [15]. Además, se hizo uso de una base de datos provista por el *Pollen Challenge* [16]. El objetivo del concurso era clasificar los granos de polen utilizando un conjunto de imágenes bajo microscopio.

Las imágenes de microscopio fueron digitalizadas y clasificadas por expertos aerobiológicos en 4 clases, las cuales incluyen 3 especies de polen y una clase extra que podría ser confundida con polen (burbujas, aire, etc).

Figure 1 – Example of class 1 (Normal Pollen)

Figure 2 – Example of class 2 (Anomalous Pollen)

Figure 3 – Example of class 3 (Alnus)

Figure 4 – Example of class 4 (Debris)

Figura 6.1: Imagen extraída de la página oficial del Pollen Challenge [16]

6.2. Métricas para clasificación Binaria

Analizaremos primero el caso de las métricas en un problema de clasificación binario. Es decir, dada una clase $\mathcal{C}$, es posible que un dato y pertenezca o no pertenezca a $\mathcal{C}$.

Es importante describir primero las métricas en este tipo de problemas, debido a que cuando nos adentremos a la clasificación multiclase, estas métricas nos serán de utilidad.

6.2.1. Matriz de confusión

Considérese un problema de clasificación binarioEn el mejor de los casos, nuestro modelo podría predecir perfectamente qué datos pertenecen a nuestra clase y cuáles no lo hacen. Sin embargo, siendo que es un modelo estadístico, nuestro modelo no siempre acertará, y usando nuestros datos etiquetados podemos detectar en dónde ha habido una *confusión*. El modelo puede confundirse de dos maneras: La primera es afirmando que $y \in \mathcal{C}$ cuando no es el caso, y la segunda es clasificar $y \notin \mathcal{C}$, cuando en realidad y sí pertenecía a la clase. Por tanto, existen 4 posibilidades:

1. **Verdaderos positivos**: Aquellos datos que sí pertenecen a la clase, y fueron clasificados dentro de la clase. A la cantidad de verdaderos positivos se le denota T_P por sus siglas en inglés.

2. **Falsos positivos**: Aquellos datos que sí pertenecen a la clase, y fueron clasificados fuera de la clase. A la cantidad de falsos positivos se le denota F_P por sus siglas en inglés.

3. **Verdaderos negativos**: Aquellos datos que no pertenecen a la clase, y fueron clasificados fuera de la clase. A la cantidad de verdaderos negativos se le denota T_N por sus siglas en inglés.

4. **Falsos negativos**: Aquellos datos que no pertenecen a la clase, y fueron clasificados dentro de la clase. A la cantidad de falsos negativos se le denota F_N por sus siglas en inglés.

Es posible agrupar toda esta información en una matriz conocida como *matriz de confusión*.

Definición 6.1. *Considérese el problema de clasificación (2.9) con $m = 2$. Sea $f : \mathbb{R}^n \to C$*

nuestro modelo. Definimos los siguientes valores:

$$T_P = |\{y_i : c_i = (1,0)^T \ y \ f(y_i) = (1,0)^T\}|$$
$$F_P = |\{y_i : c_i = (0,1)^T \ y \ f(y_i) = (1,0)^T\}|$$
$$T_N = |\{y_i : c_i = (0,1)^T \ y \ f(y_i) = (0,1)^T\}|$$
$$F_N = |\{y_i : c_i = (1,0)^T \ y \ f(y_i) = (0,1)^T\}|.$$

La matriz de confusión se define como

$$D = \begin{bmatrix} T_P & F_P \\ F_N & T_N \end{bmatrix}. \tag{6.1}$$

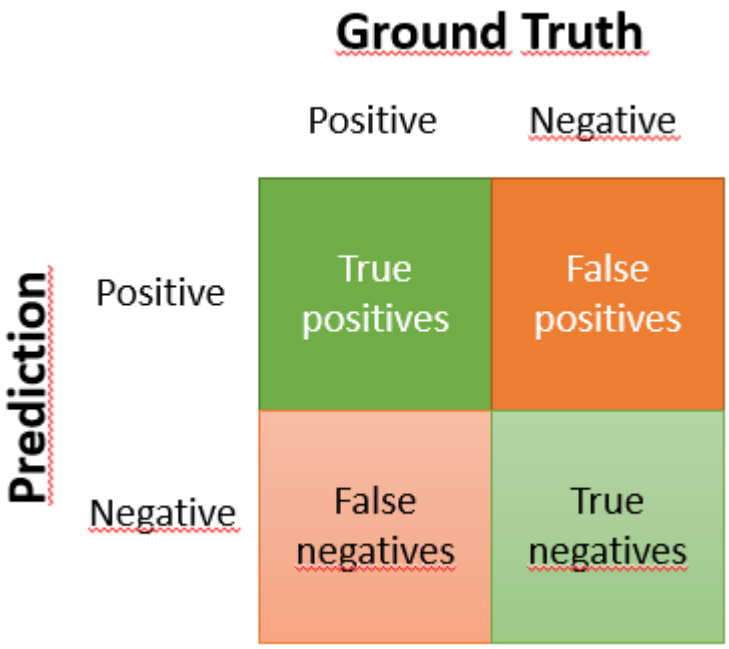


Figura 6.2: Conseguir imagen propia

6.2.2. Exactitud

La exactitud se define como la razón de las predicciones acertadas y las predicciones totales. En el aprendizaje automático suele ser la métrica principal. Sin embargo, no es el único factor a tomar en cuenta.

Definición 6.2 (exactitud). *Considérese los valores T_P, F_P, T_N, F_N de la definición (6.1). La exactitud A se define como*

$$A = \frac{T_P + T_N}{T_P + F_P + T_N + F_N} \tag{6.2}$$

6.2.3. Precisión

Definición 6.3 (Precisión). *Considérese los valores T_P, F_P, T_N, F_N de la definición (6.1). La precisión P se define como*

$$P = \frac{T_P}{T_P + F_P} \tag{6.3}$$

6.2.4. Sensibilidad

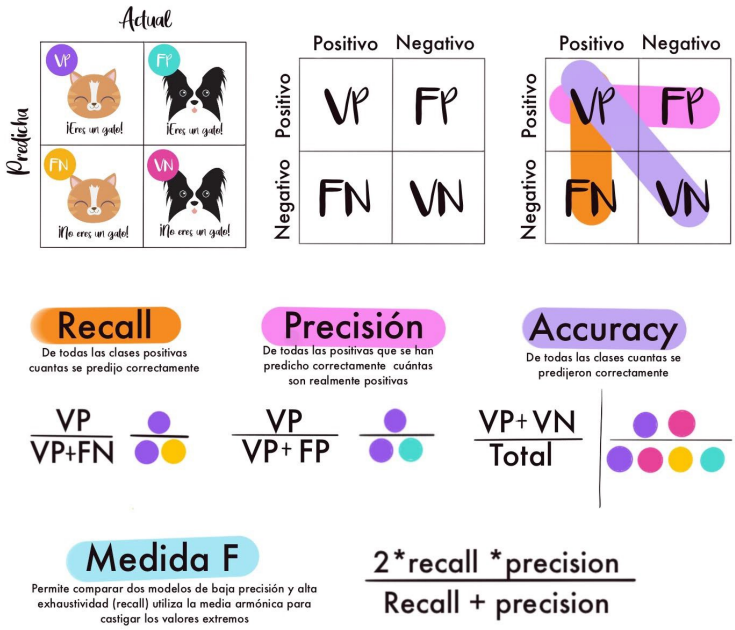
Definición 6.4 (Sensibilidad). *Considérese los valores T_P, F_P, T_N, F_N de la definición (6.1). La sensibilidad R se define como*

$$R = \frac{T_P}{T_P + F_N} \tag{6.4}$$

6.2.5. F1-score

Definición 6.5 (F1-score). *Considérese los valores T_P, F_P, T_N, F_N de la definición (6.1). La F1-score F_1 se define como*

$$F_1 = \frac{2}{\frac{1}{R} + \frac{1}{P}} = \frac{2RP}{R + P} = \frac{T_P}{T_P + \frac{1}{2}(F_P + F_N)} \tag{6.5}$$



6.3. Métricas para clasificación multiclase

En cuanto a la problema de clasificación con múltiples clases, no podemos hacer uso directo de las fórmulas anteriores. Sin embargo, es posible conseguir una matriz de confusión. En este caso, un eje determina la verdadera etiqueta, y el otro eje determina la predicción de nuestro modelo.

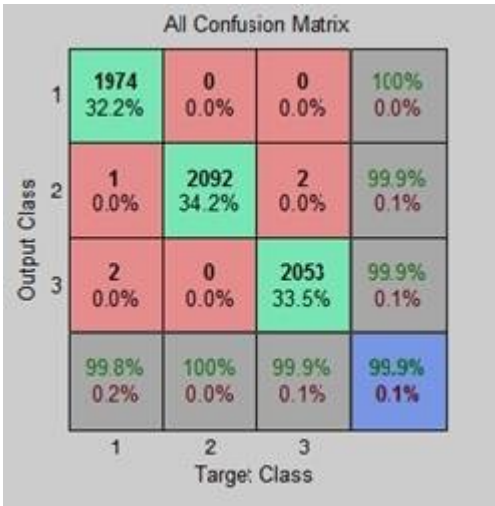


Figura 6.3: Conseguir imagen propia

6.4. Notas del capítulo

1. Añadir el libro de Suresh P. Sethi. Optimal Control Theory a las referencias.
2. Podría añadir en la accuracy el famoso ejemplo de la predicción de cancer. donde nuestro modelo es $f(x) = 0$.
3. Falta agregar una pequeña introducción en cada métrica de la clasificación binaria
4. los datos de las diapositivass en la parte experimental
5. En la referencia del pollen Challenge no aparece el URL

Bibliografía

[1] Reinhard Klette. *Concise Computer Vision - An Introduction into Theory and Algorithms*. 01 2014.

[2] Tom M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, 1 edition, 1997.

[3] Linan Zhang and Hayden Schaeffer. Forward stability of resnet and its variants. *CoRR*, abs/1811.09885, 2018.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[5] Md. Amirul Islam, Matthew Kowal, Sen Jia, Konstantinos G. Derpanis, and Neil D. B. Bruce. Position, padding and predictions: A deeper look at position information in cnns. *CoRR*, abs/2101.12322, 2021.

[6] Guilin Liu, Kevin J. Shih, Ting-Chun Wang, Fitsum A. Reda, Karan Sapra, Zhiding Yu, Andrew Tao, and Bryan Catanzaro. Partial convolution based padding. *CoRR*, abs/1811.11718, 2018.

[7] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 111–118, Madison, WI, USA, 2010. Omnipress.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[10] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[12] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *CoRR*, abs/1705.03341, 2017.

[13] John C. Butcher. *Numerical methods for ordinary differential equations*. Wiley, 2nd ed edition, 2008.

[14] Richard Bellman. *Dynamic Programming*. Princeton Landmarks in Mathematics and Physics. Dover Publications, 2003.

[15] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[16] Pollen challenge, 2020.

Capítulo 7

Notas

1. La notitas rojas son cosas que aún deben investigarse o cambiarse. Las notas azules representan textos que no están mal, pero que pueden escribirse de una mejor manera.

2. En preeliminares pretendo definir lo que es una imagen (una función) y un kernel.