Introducción a las redes neuronales de convolución. Aplicación a la visión por ordenador.

Eugenio García Sánchez
Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Directores del trabajo:
José Tomás Alcalá Nalvaiz
Lidia Orellana Lozano
13 de septiembre de 2019

Prólogo

La inteligencia artificial es una herramienta en desarrollo en el mundo moderno, la cual puede ser el gran avance tecnológico de los últimos años. Una componente de la inteligencia artificial son las redes neuronales artificiales incluidas dentro del machine learning (o aprendizaje automático).

Las redes neuronales artificiales son modelos computacionales, formados por unos elementos conocidos como neuronas las cuales están conectadas mediante unos valores variables llamados pesos sinápticos a otras neuronas. Mediante las redes neuronales podemos realizar trabajos de predicción o clasificación.

Los trabajos de clasificación han sido utilizados para resolver una amplia gama de problemas como puede ser el reconocimiento de imágenes o el reconocimiento de voz. Este tipo de procesos se realizan a partir de un tipo de redes neuronales conocidas como redes neuronales convolucionales.

El ordenador "aprende" a tomar decisiones a partir de unos datos de entrenamiento modificando los pesos mediante un proceso de ensayo y error, buscando reducir el error cometido. Las ventajas que ofrece el tipo de estructura de las redes neuronales es el hecho de tratarse de un sistema que trabaja en paralelo, es decir, varias neuronas pueden estar procesando al mismo tiempo, lo que reduce considerablemente el tiempo computacional.

Aunque las redes neuronales requieren de un gran número de parámetros los cuales debemos ajustar, una vez ajustados los parámetros ofrecen grandes resultados gracias a los cuales se continua invirtiendo en este tipo de modelos.

Las redes neuronales han conseguido grandes logros, como en el caso de Google, el cual ha conseguido derrotar a su propio reCAPTCHA con redes neuronales o en Stanford, donde se ha conseguido crear una red neuronal que genere pies de fotos automáticamente.

Por este tipo de hechos, las redes neuronales son un campo el cual está siendo estudiado actualmente ya que se prometen grandes avances a corto plazo.

En este trabajo se intentará ofrecer de una forma concisa los conocimientos básicos para la comprensión de las redes neuronales artificiales y las redes neuronales convolucionales desde sus comienzos hasta sus aplicaciones hoy en día. Después de lo cual se finalizará con la creación de una red neuronal capaz de realizar clasificación de imágenes.

Summary

Artificial neural networks, ANNs, are mathemathical models involve inside the part of machine learning. An ANN is a computacional technique with prediction and classification capacities.

The first concept of neural network appears when McCulloch and Pitts created the neuron a model that takes n input values, $\mathbf{x} = (x_1, x_2, ..., x_n)$, and generates an output, y, thanks to some values that fit the values of the input called weights, $\omega = (\omega_1, \omega_2, ..., \omega_n)$, and a bias, θ. The neuron is able to make different classifications with the following function:

$$y = sgn(\omega \mathbf{x}^T - \theta).$$

Usually the data is introduce in the sapace $[-1,1]^n$ or in the $[0,1]^n$ and the output in the space $\{-1,1\}$ or $\{0,1\}$

In general, a single neuron makes a partition of the $\mathbb{R}^n$ space, taking the wheigthed sum of the input values and introducing it in a function, called activation function. The objective of a neural network is to fit the wheigts in order to classify in a correct form the input data.

One of the most famous neural network is the perceptron created by Rosenblatt in 1958, offers good results when the problem consists in clasifying problems that are linearly separable. But whith the rest of problems the perceptron can not achive the convergence (like the problem with the logic function or exclusive). This problem was solved with the multilayer perceptron, this model consits in put more than a single layer neuron (groups of neuron with the same caracteristics), calling this kind of layers hidden layers.

There is two different kind of neural networks depending on the conection between the layers:

- Feed-forward propagation: a layer only can be conected with depper layers.
- Feed back-forward propagation: a layer can be conected with the rest of layers and a neuron can be conected with neurons of the same layer including itself.

There are a huge number of different structures, the structure depends on the selection of the propagation rule (relatioship between weights and output data of the neurons), the activation function, the number of neurons and layers and the learning rule (method that consists in the reduction of error fitting the weights).

The training of a neural network is the process used for the decrease of the error, for that we need a data set with the correct label, known as training data. Introducing this data with a random initial weights, the learning rule decrease the error. A posible problem is to reduce the error in a local minima istead of a global one. To avoid this problem the training is initialized with different random weights or it exists some learning rules that ocasionally can avoid this problem, like the stochastic gradient descendent algorithim.

There are diferent measures to compare the quality of our neural network like acuraccy, the curve ROC and the AUC, this measures provide a comparative to choose between diferent models

In section 1.5, it is shown a comparative between classic statistical methods and neural networks for classification and prediction, where we can see results teorical and practical the conclusion is that ANNs provide little better results in classification than statistical models and similar in the prediction part, but the computacional cost of neural networks is bigger.

In chapter 2, we introduce a more advance kind of neural network, the convolutional neural networks, CNNs, this kind of neural network is usefull to process data with a big shape, like images or voice patterns. The idea started when Kunihiko Fukushima developed the neoconigtron a backpropagation network that imitates the visual cortex process, in 1980. But was in 2012 with the creation of the AlexNet when the potencial of this kind of network to process images was developped.

The neural network starts with a convolutional layer. The convolutional layer is focus on taking different characteristics from the imput data, and after that, there is a pooling layer. The pooling layer reduce the volumen of data. The most important pooling layers are the Max-Pooling and the Average-Pooling:

- Max-Pooliing: dividing the data in differen region. The Max-Pooloing takes the higguer value of each region.
- Average-Pooling: for each region instead of taking the maximun value takes the average of all valuues.

We can repeat convolutional layers and pooling layers very times and after that we link the structure of a forward full connected ANN.

The final part is a practical example about ticket classification through images. The task is given by 4 diferent kind of tickets (restaurant, parking, taxi and fuel station), we have generated a computer program based on CNNs with the capacity of recognising the kind of ticket.

For carrying out the solution to this problem was generated a data base of 239 tickets. The train data was of 191 tickets and the testing data was of 48.

The model selected was made of 50 filters of convolution with an hiperbolic tangent activation function, followed by a filter of Max-Pooling, after that the program has 50 hidden neurons with an linear activation and 4 output neuron with softmax activation function. The learning rule is an adaptative gradient.

The acuraccy given by the model is 0.9583 with a micro-average AUC of 0.98 and a macro-average AUC of 0.99, althoug the number of parameters needed to achive these results are over than 24.000.000.

Índice general

Capítulo 1

Redes neuronales artificiales

En este capítulo introduciremos las **redes neuronales artificiales** y desarrollaremos los conceptos básicos para entender el funcionamiento de una red neuronal.

1.1. Introducción

En esta sección desarrollaremos una idea general de la utilidad de las redes neuronales artificiales o simplemente redes neuronales (**RNA** o artificial neural networks ANN en inglés), realizaremos también, un breve repaso a la historia de las redes neuronales y ventajas que proporcionan.
Aunque procuramos alejarnos de la idea clásica de la compración de las redes neuronales con la cognición humana, es inevitable comentar que en su topología se intenta imitar la estructura del cerebro humano y la sinapsis neuronal de este.
Las redes neuronales son modelos matemáticos que se incluyen en la familia de algoritmos conocida como deep learning (aprendizaje profundo en castellano), donde una red neuronal es un modelo computacional, paralelo, compuesto de unidades procesadoras adaptativas (llamadas neuronas) con una alta interconexión entre ellas. Visto como un proceso de reconocimiento de patrones, las redes neuronales son una extensión de los métodos estadísticos.
Se trata de un proceso de machine learning (o aprendizaje automático), el cual es usado para dos posibles objetivos, clasificación y predicción:

- Clasificación: una red neuronal puede ser usada para diferenciar unos elementos de otros, puede ir desde ofrecer la solución de funciones lógicas (como la función *or*, *not* o &) a la clasificaión de imágenes (como realizar distinciones entre tipos de animales o reconocimiento facial).
- Predicción: a partir de resultados previos se realiza una aproximación númerica a situaciones diferentes de las cuales se desconoce su resultado.

Aunque se realizaron estudios anteriores, se considera el primer modelo, el presentado por McCulloch y Pitts [1], quienes en 1943 presentaron un elemento conocido como **neurona**, el cual tomaba n valores de entrada y generaba una salida mediante una función, que mediante unos valores que ajustan los valores de entrada y un sesgo podían realizar ciertas clasificaciones, la forma general de esta función era (1.1) (siguiendo lo sugerido por Zhang, L. y Zhang, B. [2]):

$$y = sgn(\omega x - \theta), \tag{1.1}$$

donde:

$$sgn(v) = \begin{cases} -1 & \text{si } v \leqslant 0 \\ 1 & \text{si } v > 0 \end{cases},$$

$\omega = (\omega_1, \omega_2, ..., \omega_n)$	valores que ajustan la entrada o pesos sinápticos
$x = (x_1, x_2, ..., x_n)$	valores de entrada
θ	es un sesgo introducido para modificar la salida

.

En 1949, Donald Hebb desarrolla una de las de las **reglas de aprendizaje**, método de reducción del error, más usada, The Hebbian rule aparece publicada en su libro [3]. En 1958 Rosenblatt presenta el **perceptrón** inspirado en el trabajo de McCulloch y Pitts, una red neuronal que toma varias entradas binarias y produce una única salida binaria. En 1960 Widrow y Hooff presentan el **ADALINE**, un modelo que ofrecía buenos resultados cuando el resto de modelos se habían estancado. El perceptrón multicapa creado en 1965 también supo resolver los problemas existentes (para más detalles ver [4]).
Años después en 1989 Yann LeCun crea la primera red neuronal convolucional, LeNet [5], la cual abrió un gran abanico de posibilidades en el reconocimiento de datos con más de una dimensión (reconocimiento de imágenes o de voz) y en 2012 con la creación de la red AlexNet por Hinton, Sutskever y Krizhevsky se observó todo el potencial que pueden llegar a ofrecer este tipo de redes neuronales.

1.1.1. Ventajas de las redes neuronales

Para poder comprender el gran avance que suponen las redes neuronales debemos comprender las ventajas que ofrecen (introducidas en [4]):

- Son sistemas distribuidos no lineales, una neurona al operar de forma no lineal implica que la red neuronal no opera de forma lineal, permitiendo operar con sistemas no lineales o caóticos.
- Son tolerantes a fallos, es decir que el fallo en una neurona no obteniendo la salida optima no tiene porque afectar al buen funcionamiento de nuestra red.
- Adaptabilidad, el sistema ofrece cierta capacidad a los cambios en el entorno de trabajo, presencia de ruido, modificaciones en la entrada..., aunque tampoco ofrece una gran capacidad de adaptabilidad ya que esto afectaría a la convergencia de nuestro modelo.
- Tiene la capacidad de establecer relaciones complejas entre los datos de entrada.
- Aunque sus objetivos sean los mismos que los de la estadística clásica, puede haber variación en los resultados obtenidos siendo mejor las redes en la clasificación y actuando de forma similar en la predicción, aunque las redes neuronales convolucionales presentan un coste mayor de programación (ver sección 1.5).

Todas estas propiedades generan la gran capacidad que poseen las redes neuronales, su buen funcionamiento y su alta aplicabilidad en el mundo moderno.

1.2. Componentes de una red neuronal

En esta sección vamos a desarrollar la idea general de como funciona una red neuronal. Nuestro objetivo será dada una función, f, la cual realiza una partición del espacio $\mathbb{R}^n$, buscar una aproximación de f:

$$f : [-1,1]^n \longrightarrow \{-1,1\}^r,$$

de la cual conocemos la salida deseada, $\mathbf{d} \in \{-1,1\}^r$ aplicada a un patrón o entrada $\mathbf{x} \in [-1,1]^n$, donde r representa el número de clases en las que deseamos separar nuestro patrón de entrada. Análogamente podremos definir la entrada en $[0,1]^n$ y la salida en $\{0,1\}^r$. La infornación que introduciremos en la red serán p patrones de entrenamiento:

$$\{\mathbf{x}^1, \mathbf{d}^1\}, \{\mathbf{x}^2, \mathbf{d}^2\}, ..., \{\mathbf{x}^p, \mathbf{d}^p\}.$$

Estas entradas se irán modificando dentro de la red en cada neurona hasta formar una salida, la cual al principio del entrenamiento producirá una salida que no coincidirá con la salida deseada, pero a través de la modificación de los valores correspondientes a las conexiones, o **pesos** (denotados por ω_{ab} el valor de la de la neurona a con la neurona b), obtendremos la salida deseada.
Para conseguir que los datos entren en el espacio $\mathbf{x} \in [-1,1]^n$ realizaremos un proceso de normalización, y si todos los datos son positivos entonces, dividiendo por el máximo x_i viable, la entrada será $\mathbf{x} \in [0,1]^n$.

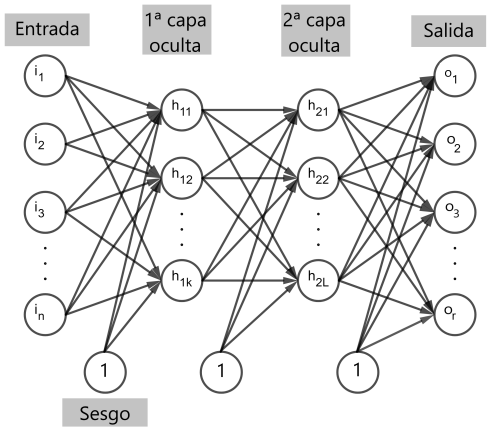


Figura 1.1: Red neuronal con dos capas ocultas y n neuronas de entrada y r de salida.

1.2.1. Unidades de procesamiento

Una neurona o unidad de procesamiento es el elemento básico de una red neuronal. Una neurona se encarga de forjar un **valor de salida**, y_j, a partir de las otras neuronas que enlazan, es decir tienen **conexiones**, con ella e información obtenida del exterior, **sesgo**.
Podemos discernir tres tipos de neuronas:

1. Neuronas de entradas, i_k: su función es procesar la información procedente del exterior e introducirla en el sistema, en particular introduce la componente k-ésima del vector $\mathbf{x}^j$.

2. Neuronas ocultas, h_{jk}: son neuronas que se encuentran en el centro de nuestra red, tienen una conexión θ_{jk} llamada **sesgo**, o bias en inglés, y esta conectada con otras neuronas de capas previas.

3. Neuronas de salida, o_k: son las neuronas que se encargan de mostrar la predicción o clasificación realizada por nuestra red, las cuales ofrecerán valores de salida en $\{-1,1\}$ o en $\{0,1\}$.

1.2.2. Conexiones entre unidades

Ofrecer una conexión entre un grupo de neuronas $a_1, a_2, ..., a_k$ en una neurona b equivale a dar una **regla de propagación** de las neuronas $a_1, a_2, ..., a_k$ a la neurona b, donde una regla de propagación es una función que relaciona la salida de las neuronas $a_1, a_2, ..., a_k$ y la entrada de la neurona b.
En la mayoría de los casos asumimos que cada unidad ofrece una contribución aditiva a la entrada de las unidades a las cual esta conectada. La regla de propagación usual es aquella en la cual la entrada total en la unidad b es simplemente la suma de las salidas de las diferentes neuronas a las cuales esta conectada, ponderada por los pesos de dichas componentes junto con la entrada externa, o sesgo θ_b

(según lo sugerido por Kröse y Smagt en [7, pág. 16]).

$$s_b(t) = \sum_{j=1}^{k} \omega_{jb}^t y_j^t + \theta_b^t.$$

Si la contribución ω_{jb} es positiva se denomina excitación, en caso contrario se denomina inhibición. Existen otras reglas de propagación más complejas y menos frecuentes, por ejemplo, la introducida por Feldman y Ballard, conocida coma la regla sigma-pi unit:

$$s_b(t) = \sum_{j=1}^{k} \omega_{jb}^t \prod_{i=1}^{k} y_i^t + \theta_b^t.$$

1.2.3. Reglas de activación y salida

A lo largo largo del proceso de aprendizaje de una red neuronal se deben aplicar otros factores como las **reglas de activación** y desactivación de las neuronas, se trata de una regla que da el efecto de las entradas en la activación de la unidad. Para una neurona cualquiera k del modelo la función, F_k, que toma todas las entradas s_k^t y el estado actual de la neurona y_k^t produciendo un nuevo estado de activación de esta (como se puede ver en [7, pág. 16-17]):

$$y_k^{t+1} = F_k(y_k^t, s_k(t)).$$

Suele ser una función no decreciente del total de entradas en la unidad:

$$y_k^{t+1} = F_k(s_k(t)) = F_k(\sum_j \omega_{jk}^t y_j^t + \theta_k^t).$$

Algunas de las funciones de activación más comunes son las siguientes:

- la función ReLU (Rectified Lineal Unit):
  es una función que anula los valores negativos y deja los positivos tal como entran.

  $$y_k^{t+1} = F(s_k(t)) = \text{máx}\{0, s_k(t)\}.$$

  La cual ha demostrado un buen desempeño en tipos de problemas como el reconocimiento de imágenes (tal y como se desarrollará en el cápitulo 2), aunque puede dejar muchas neuronas desactivadas y no está acotada.

- la función lineal:

  $$y_k^{t+1} = a \cdot s_k(t),$$

  donde a es una constante real. Esta función destaca por tener cierto parecido con la ReLU, aunque destaca los valores negativos. Puede ser utilizada para exagerar las características que más destacan ($a > 1$) o suavizarlas ($a < 1$).

- la función sigmoidal:

  $$y_k^{t+1} = F(s_k(t)) = \frac{1}{1 + e^{-s_k(t)}}.$$

  Una de las bondades de esta función reside en la posibilidad de escribir su derivada en términos de

ella misma, lo cual será muy importante para optimizar el coste computacional, lo cual explicamos a continuación en mayor detalle.

$$F(x) = \frac{1}{1+e^{-x}},$$
$$F'(x) = F(x)(1-F(x)).$$

En nuestro caso la función F dependerá de diversas variables $\theta, \omega_1, \omega_2, ..., \omega_m$,

$$F = F(\theta + \omega_1 y_1 + \omega_2 y_2 + ... + \omega_m y_m)$$

.
Por lo cual, aplicando la regla de la cadena:

$$\frac{\delta F}{\delta \theta} = F(1-F),$$
$$\frac{\delta F}{\delta \omega_i} = y_i F(1-F) \qquad i = 1, ..., m.$$

1.2.4. Reglas de aprendizaje

Las **reglas de aprendizaje** son los algoritmos utilizados para la modificación de los pesos y el umbral con la itención de disminuir el error cometido por nuestra red.
Las **funciones de error**, E, son aquellas que ofrecen una medida de la diferencia entre la salida de nuestra red y la salida deseada, algunas de las funciones que podemos encontrar son:

- Error cuadrático:
$$E = \frac{1}{2}\sum_{j=1}^{p}\sum_{k=1}^{r}(d_j^k - y_j^k)^2.$$
- Entropía cruzada:
$$E = -\sum_{j=1}^{p}\sum_{k=1}^{r}[d_j^k log(y_j^k) + (1-d_j^k) log(1-y_j^k)].$$

A continuación introduciremos algunas de las reglas de aprendizaje más importantes:

Hebbian rule

The hebbian rule (Donald Hebb, 1949, [8]). Fue la primera regla de aprendizaje, podemos utilizarla para identificar como mejorar los pesos de las neuronas de la red.
La regla de aprendizaje de Hebb asume que si dos neuronas vecinas activadas y desactivadas al mismo tiempo, el peso que las une debería incrementarse. En caso contrario debería disminuirse. Y en caso de que no hubiera correlación no deberíamos cambiar el peso de su conexión.
Si las entradas de ambas neuronas son positivas entonces un alto peso positivo existe entre ellas: en caso de que una salida sea negativa y la otra positiva, un fuerte peso negativo existe entre los nodos.

La regla es la siguiente:

$$\omega_{ij} = y_i y_j.$$

Al comienzo del entrenamiento todos los pesos son puestos a cero, los valores absolutos de los pesos son proporcionales al tiempo de entrenamiento, lo cual no es deseable ya que se alcanza un fenómeno denominado sobreajuste, el cual se desarrollará más adelante.

Regla de aprendizaje del perceptrón simple

Es el método de aprendizaje usado por el perceptrón simple, un tipo de red que explicaremos en profundidad más adelante. La regla es expresada de la siguiente manera:

$$\omega_j^{t+1} = \omega_j^t + \Delta\omega_j^t,$$

siendo

$$\Delta\omega_j^t = \mu(t)[d^t - y^t]x_j^t,$$

lo cual indica que el peso es proporcional al error producido multiplicado por el valor de la entrada. $\mu(t) \in (0,1)$ es un parámetro positivo, llamado **tasa de aprendizaje**, que limita la magnitud del cambio de los pesos sinápticos.

Regla del método delta

Desarrollada por Widrow y Hoff, la regla del método delta, es una de las más usadas. Funciona en el aprendizaje supervisado (se compara dado un patrón de entrada si su salida, y, coincide con la salida deseada, d,). Se basa en que el peso sináptico de una neurona debe ser modificada en función del error multiplicado por su entrada y una tasa de aprendizaje.
Primero definimos W_1 como la matriz de pesos de la capa de entrada a la primera capa oculta, incluyendo el sesgo, de la misma manera definimos W_2, la matriz que representa los pesos de la segunda capa oculta con la tercera, siguiendo sucesivamente denotamos W_K los pesos que relacionan la última capa oculta y la capa de salida, denominamos por Θ el tensor formado por $(W_1,...,W_K)$.
Si en la capa i disponemos de la función de activación F_i, podemos definir la salida ofrecida por la red de la siguiente manera:

$$y = F_K(...F_3(F_2(F_1(x,W_1),W_2),W_3)...,W_K).$$

Debido a este hecho podemos expresar la función de error, como una función $E(\Theta,x)$, puesto que la salida deseada, d, depende de la entrada, x, por tanto, la regla del método delta se expresa de la siguiente manera:

$$\omega_j^{t+1} = \omega_j^t - \mu\frac{\delta E(\Theta,x)}{\delta\omega_j}(\omega_j^t, x^t), \tag{1.2}$$

Por ser supervisada compara si el vector de salida coincide con la respuesta deseada. Si la diferencia fuera cero podemos observar que no habrá modificación de los pesos, en caso contrario, ajusta los pesos para reducir su diferencia.

Método del gradiente estocástico

El método del gradiente estocástico es un proceso que permite hallar mínimos en funciones $J : \mathbb{R}^d \longrightarrow \mathbb{R}$ diferenciables.
Se puede formular el método del gradiente estocástico de la siguiente manera:

1. Se toma $u^0 \in \mathbb{R}^d$, arbitrario.
2. Conocido u^t, se calcula u^{t+1} de la siguiente manera:

$$u^{t+1} = u^t - \mu\nabla J(u^t),$$

donde μ es la tasa de aprendizaje.
Por otro lado se tiene que para toda función elíptica el método del gradiente estocástico converge en un número finito de pasos, donde una función, $J : \mathbb{R}^d \longrightarrow \mathbb{R}$, se dice elíptica si es continua y diferenciable y existe $\alpha > 0$ tal que:

$$(\nabla J(v) - \nabla J(u))(v-u) \geq \alpha\|v-u\|^2.$$

Una demostración de este hecho se puede ver en [9, pág. 44-49].

- **Modificación de los pesos**
  Por ser una regla de aprendizaje debemos tener en cuenta como se modifican los pesos.
  Siguiendo la notación expresada en (1.2). Podemos aplicar el método del gradiente estocático de la siguiente manera:
  $$\Theta^{t+1} = \Theta^t - \mu \nabla_\Theta E(\Theta^t, x^t).$$

Método AdaGrad

El gradiente adaptativo, AdaGrad, es igual que la regla del aprendizaje del gradiente estocástico, salvo que la tasa de aprendizaje, μ_t se ajusta después de algunas iteraciones.

$$\Theta^{t+1} = \Theta^t - \mu_t \nabla_\Theta E(\Theta^t, x^t).$$

Ejemplo del algoritmo de retropropagación

El algoritmo de retropropagación es el método utilizado para la modificación de los pesos. Daremos un ejemplo práctico de como se calculan las derivadas para una red neuronal con dos capas ocultas. Denominando por y_1 la salida producida por la primera capa oculta, y_2 la producida por la segunda e y_3 la producida por la capa de salida, se tiene:

$$y_1 = F_1(x, W_1),$$
$$y_2 = F_2(y_1, W_2) = F_2(F_1(x, W_1), W_2),$$
$$y_3 = F_3(y_2, W_3) = F_3(F_2(y_1, W_2), W_3) = F_3(F_2(F_1(x, W_1), W_2), W_3).$$

Por lo cual aplicando la regla de la cadena:

$$\nabla_{W_3} y_3 = \frac{\delta y_3}{\delta W_3} = \frac{\delta F_3(y_2, W_3)}{\delta W_3},$$
$$\nabla_{W_2} y_3 = \frac{\delta y_3}{\delta W_2} = \frac{\delta F_3(y_2, W_3)}{\delta W_2} = \frac{\delta F_3(y_2, W_3)}{\delta y_2}\frac{\delta y_2}{\delta W_2} = \frac{\delta F_3(y_2, W_3)}{\delta y_2}\frac{\delta F_2(y_1, W_2)}{\delta W_2},$$
$$\nabla_{W_1} y_3 = \frac{\delta y_3}{\delta W_1} = \frac{\delta F_3(y_2, W_3)}{\delta W_1} = \frac{\delta F_3(y_2, W_3)}{\delta y_2}\frac{\delta y_2}{\delta W_1} = \frac{\delta F_3(y_2, W_3)}{\delta y_2}\frac{\delta F_2(y_1, W_2)}{\delta W_1}$$
$$= \frac{\delta F_3(y_2, W_3)}{\delta y_2}\frac{\delta F_2(y_1, W_2)}{\delta y_1}\frac{\delta y_1}{\delta W_1}.$$

Por lo cual, podemos calcular $\dfrac{\delta E(\Theta, x)}{\delta \Theta}$, con cierta facilidad.

1.3. Topologías de redes neuronales

Una vez conocidas las distintas componentes de una red neuronal, procedemos a explicar los distintos tipos de redes las cuales se clasifican según su topología:
La topología de una red viene dada por como están definidas sus funciones de activación, la función de propagación, su regla de aprendizaje y su número de capas. Según sus conexiones podemos definir dos tipos de redes neuronales:

- Redes Feed-forward propagation (o simplemente forward), donde son estrictamente hacia delante. Los ejemplos clásicos de este tipo de red son el perceptrón y el Adaline los cuales presentaremos en mayor detalle.

- Redes recurrentes (Feed back-forward propagation o back-forward), se permiten conexiones entre redes de la misma capa, incluída la misma neurona y hacia capas anteriores. En este tipo de redes se pueden alcanzar estados de estabilidad para ciertas neuronas, donde la activación de esta neurona no cambiará.

1.3.1. Perceptrón

Esta sección será desarrollada siguiendo lo expresado en [10]. El perceptrón es la red neuronal más básica y antigua, es usado para la separación de patrones linealmente separables. Consiste en una única capa neuronal, capa de salida, con pesos sinápticos, o simplemente pesos, y un umbral ajustables, ver figura (1.2).

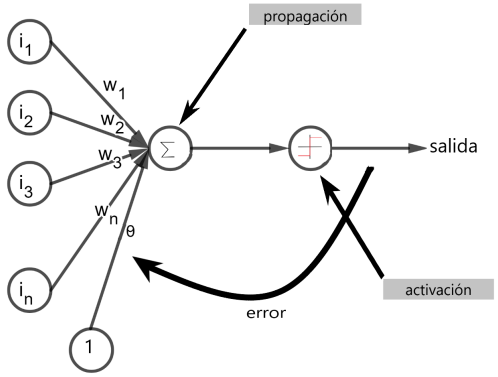


Figura 1.2: Estructura del perceptrón.

Convergencia del perceptrón

Fue un procedimiento desarrollado por Rosenblatt en 1958, quien demostró que si los patrones necesarios para entrenar la red son obtenidos de dos clases linealmente separables entonces el algoritmo es convergente y la superficie de decisión es un hiperplano separando las dos superficies (Teorema de convergencia del perceptrón).
Si el perceptrón tiene una única neurona, perceptrón simple, estará limitado a la separación en dos clases, en cualquier caso las clases deben ser linealmente separables para la convergencia del algoritmo.

Teorema 1.1. *Si el conjunto de patrones de entrenamiento:*

$$\{\boldsymbol{x}^1,\boldsymbol{d}^1\},\{\boldsymbol{x}^2,\boldsymbol{d}^2\},...,\{\boldsymbol{x}^p,\boldsymbol{d}^p\}.$$

es linealmente separable entonces el algoritmo del perceptrón simple converge en número finito de pasos.

Demostración. El objetivo es la separación de las observaciones $x^1,x^2,...x^p$ en dos clases C_1, C_2: Por ser los patrones de entrada linealmente separables existen unos valores $\omega_1^*,\omega_2^*,...,\omega_n^*$ y θ^* tales que:

$$\sum_{x_j\in C_1}\omega_j^* x_j > \theta,$$

$$\sum_{x_i\in C_2}\omega_i^* x_i < \theta.$$

Supongamos que la iteración t tiene que modificar los pesos sinápticos según la regla de aprendizaje utilizada, como y^t no coincide con la salida deseada d^t. Se tiene:

$$\sum_{j=1}^{n}(\omega_j^{t+1}-\omega_j^*)^2+(\theta^{t+1}-\theta^*)^2=\sum_{j=1}^{n}(\omega_j^t+\mu[d^t-y^t]x_j^t-\omega_j^*)^2+(\theta^t+\mu[d^t-y^t]-\theta^*)^2=$$

$$=\sum_{j=1}^{n}(\omega_j^t-\omega_j^*)^2+(\theta^t-\theta^*)^2+\mu^2[d^t-y^t]^2(\sum_{j=1}^{n}(x_j^t)^2+1)+2\mu[d^t-y^t](\sum_{j=1}^{n}(\omega_j^t-\omega_j^*)x_j^t+\theta^t-\theta^*)=$$

$$= \sum_{j=1}^{n} (\omega_j^t - \omega_j^*)^2 + (\theta^t - \theta^*)^2 + \mu^2 [d^t - y^t]^2 (\sum_{j=1}^{n} (x_j^t)^2 + 1) +$$

$$+ 2\mu [d(t) - y(t)] (\sum_{j=1}^{n} \omega_j^t x_j^t + \theta^t) - 2\mu [d^t - y^t] (\sum_{j=1}^{n} \omega_j^* x_j^t \theta^*). \tag{1.3}$$

Debemos notar que, $[d^t - y^t](\sum_{j=1}^{n} \omega_j^t x_j^t + \theta^t) < 0$, ya que si $\sum_{j=1}^{n} \omega_j^t x_j^t + \theta^t > 0$ entonces $y^t = 1$ y como la salida es errónea $d^t = -1$ y si $\sum_{j=1}^{n} \omega^t x_j^t + \theta^t < 0$ entonces $y^t = -1$ y como la salida es errónea $d^t = 1$
Por ello se puede escribir

$$[d^t - y^t](\sum_{j=1}^{n} \omega_j^t x_j^t + \theta^t) = -2|\sum_{j=1}^{n} \omega_j^t x_j^t + \theta^t|.$$

Por un razonamiento similar se tiene que:

$$[d^t - y^t](\sum_{j=1}^{n} \omega_j^* x_j^t + \theta^*) = 2|\sum_{j=1}^{n} \omega_j^* x_j^t + \theta^*|.$$

Por tanto, sustituyendo en (1.3) y eliminando un termino, tenemos:

$$\sum_{j=1}^{n} (\omega_j^{t+1} - \omega_j^*)^2 + (\theta^{t+1} - \theta^*)^2 \leqslant \sum_{j=1}^{n} (\omega_j^t - \omega_j^*)^2 + (\theta^t - \theta^*)^2 + 4\mu^2 (\sum_{j=1}^{n} (x_j^t)^2 + 1) - 4\mu |\sum_{j=1}^{n} \omega_j^* x_j^t + \theta^*|.$$

Llamando:

$$D(t+1) = \sum_{j=1}^{n} (\omega_j^{t+1} - \omega_j^*)^2 + (\theta^{t+1} - \theta^*)^2,$$

$$D(t) = \sum_{j=1}^{n} (\omega_j^t - \omega_j^*)^2 + (\theta^t - \theta^*)^2,$$

$$L = \max_{1 \leqslant t \leqslant p} \{\sum_{j=1}^{n} (x_j^t)^2 + 1\},$$

$$T = \min_{1 \leqslant t \leqslant p} \{\sum_{j=1}^{n} \omega_j^* x_j^t + \theta^*\}.$$

Por lo cual tenemos:

$$D(t+1) \leqslant D(t) + 4\mu^2 L - 4\mu T,$$

$$D(t+1) \leqslant D(t) + 4\mu(\mu L - T).$$

Tomando μ, tal que, $\mu < \dfrac{T}{L}$ se obtiene que $D(t+1) < D(t)$.
Por ello, podemos hacer disminuir $D(t)$ en cada iteración y por ello sabemos que hay convergencia en un número finito de pasos, puesto que en caso contrario obtendríamos un $D(t) < 0$, lo cual es imposible. □

Problema XOR (o exclusivo lógico)

El XOR, es una función lógica que actúa de acuerdo con el cuadro (1.1):
El perceptrón separa mediante hiperplanos, pero en la función XOR, sus imágenes no pueden ser separadas mediante dichos hiperplanos (figura 1.3).

En el gráfico (1.3) se puede observar que los cuatros puntos no se pueden separar linealmente, lo cual, fue determinante a la hora de formular el modelo del perceptrón multicapa, que ofrece solución a este tipo de problemas.

A	B	A OR B
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Cuadro 1.1: función XOR.

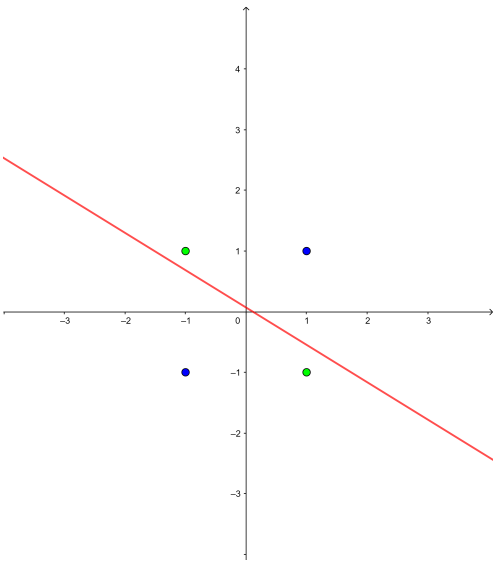


Figura 1.3: Problema XOR.

1.3.2. Adaline

Es una red de tipo forward que suele ser explicada junto al perceptrón debido a sus similitudes. La únicas diferencias son que Adaline usa la regla de aprendizaje del método delta y como función de activación la función signo.

1.4. Entrenamiento en redes neuronales

Las redes neuronales cuentan con dos fases, entrenamiento y predicción. Para el ajuste de los pesos de las conexiones (fase de entrenamiento), se necesitan dos conjuntos de datos. Un conjunto de entrenamiento y otro de validación.
Al comienzo de nuestra red contendrá unos pesos puestos al azar. Mediante el conjunto de datos de entrenamiento se procederá al ajuste de los datos minimizando el error producido por estos. Los datos de entrenamiento se pueden introducir varias veces llamando a cada reiteración **época**.
Una vez realizas todas las épocas deseadas, habiendo ajustado los pesos de forma que nuestros datos de entrenamiento son predichos de forma correcta, se procede a la introducción de los datos de validación. Si la predicción de los datos de validación es correcta, podemos dar por finalizado la fase de entrenamiento. En caso contrario si hemos escogido una topología de red adecuada, la falta de precisión en la validación se puede deber a dos causas principales, una mala elección en los datos de entrenamiento o un sobreajuste.
La mala elección de los datos de entrenamiento puede ser debida a que el conjunto es poco representativo, los datos no suficientes o no se ajustan de manera equilibrada a todas las clases.
El sobreajuste se produce cuando en nuestra red se ha producido un sobreentrenamiento. La red se centra en detalles poco relevantes, los cuales impiden distinguir casos que estén fuera de los datos de entrenamiento, dicho de otra manera es incapaz de generalizar.

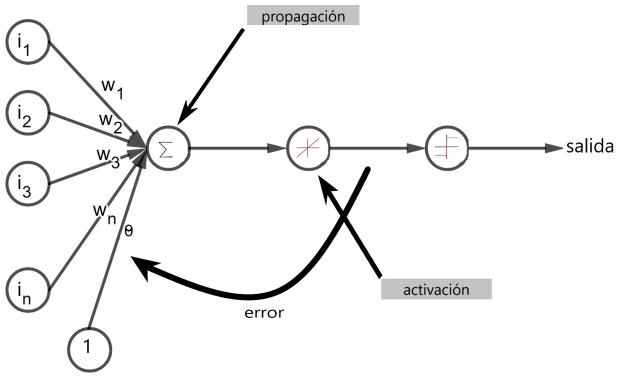


Figura 1.4: Estructura Adaline.

Las siguientes secciones se basan en el trabajo de Svozil, Kvasnicka y Popischal ([12, Cap.4]).

1.4.1. Disminución del error

Durante la fase de entrenamiento podemos diferenciar dos tipos de disminución de error:

- Pattern mode: el error se modifica después del procesamiento de cada dato de entrenamiento. También conocido como entrenamiento online.
- Batch mode: el error se modifica después del procesamiento de un conjunto de datos, a veces incluso tras cada época. También conocido como entrenamiento off line.

En la práctica se prefiere el batch mode puesto que aunque requiere un mayor uso de memoria reduce el coste computacional en la modificación de los pesos, además si los datos son presentados de una forma aleatoria entonces se reduce la probabilidad de caer en un mínimo local de la función de error, en vez de en un mínimo global del error el algoritmo se estaciona en un mínimo local. El batch mode permite una aproximación más precisa del vector gradiente. El análisis de la efectividad de cada método solo se puede observar una vez resuelto el problema de ambas formas.
Introduciendo en la red los datos de validación se observa el error cometido por la red y se utiliza como estimación de la calidad de nuestra red.
Una red se dice que generaliza bien cuando incluimos datos en nuestra red, no usados en el entrenamiento, y se obtiene la respuesta deseada.

1.4.2. Selección del modelo

Para la selección de las variables de entradas, se suele realizar un estudio previo de la importancia de las variables mediante métodos estadísticos, manteniendo las que posean una mayor relevancia.
Otra complejidad se haya en la selección del modelo es el número de capas ocultas que debemos introducir, pero según [12], no es necesario el uso de más de dos capas. Siendo más específicos según, Hornik [13], se puede observar que se debe usar el menor número de capas ocultas posibles para la resolución del problema. El uso de dos capas ocultas se suele utilizar para la aproximación de funciones continuas con algunas discontinuidades, aunque el uso de dos capas ocultas ya aumenta las posibilidades de obtener un mínimo local del error en vez del global. En caso de usar varias capas deberemos inicializar el proceso con distintos pesos aleatorios o realizar distintos algoritmos que disminuyan las posibilidades de acabar convergiendo en mínimos locales del error.
Para evitar la convergencia en mínimos locales podemos emplear la validación cruzadas de k-capas. Este método consiste en la separación del conjuntos en k subconjuntos del mismo tamaño, a continuación

se escoge un subconjunto de estos k y se entrena la red con los $k-1$ conjuntos restantes y se utiliza el conjunto restante para la validación de esta red, realizamos k entrenamientos diferentes eligiendo cada vez un conjunto diferente y elegimos el modelo con mínimo error en la validación.
Otra forma de la selección del número de capas ocultas y de nodos es entrenar varias redes al mismo tiempo con distintas topologías y ver cuales cometen un menor error en la evalución de los datos de validación.
Para la observación del error cometido por nuestra red se puede observar el porcentaje de acierto de nuestra red o podemos realizar un análisis de la curva ROC (Receiver Operating Characteristic) que ofrece una representación gráfica de la sensibilidad frente a la especibilidad de un sistema de clasificación binario:

- sensibilidad (VPR): $VPR = \dfrac{VP}{VP+FN}$,
- especifidad (SPC): $SPC = \dfrac{VN}{FP+VN}$,
- valor predictivo positivo o precisión (PPV): $PPV = \dfrac{VP}{VP+FP}$,

donde, VP es el número de verdaderos positivos (datos que son predicho como correctos y realmente son correctos), VN es el número de verdaderos negativos o también rechazos correctos, FP el número de falsos positivos y FN el número de falsos negativos.

Por lo cual se define la matriz de confusión a la siguiente matriz:

$$cm = \begin{pmatrix} VP & FP \\ FN & VN \end{pmatrix},$$

la cual nos da una idea general del número de errores cometidos por nuestra red.
Otra medida muy útil es la exactitud o precisón (ACC): $\dfrac{VP+VN}{N}$, siendo N el número de datos totales.

La curva ROC es un indicador para discriminar entre dos categorías $C1$ y $C2$ las cuales tendrán dos distribuciones de probabilidad (figura 1.5). En ella, se puede ver en sombreado la probabilidad de clasificar como positivo un caso al azar para un umbral dado.
Para cualquier valor se puede realizar una clasificación errónea de algunos de los casos de las zonas

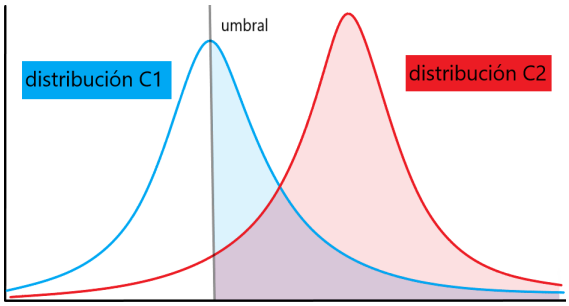


Figura 1.5: Distribuciones de C1 y C2

solapadas. La curva ROC dibuja para todos los posibles umbrales los pares de positivos para ambas poblaciones. Obteniendo un gráfico similar a la figura (1.6):

Una medida de la calidad de nuestra curva ROC es la medida AUC (area under of curve), la cual denominando t al umbral, la podemos definir de la siguiente manera:

$$AUC = \int_0^1 ROC(t)dt. \tag{1.4}$$

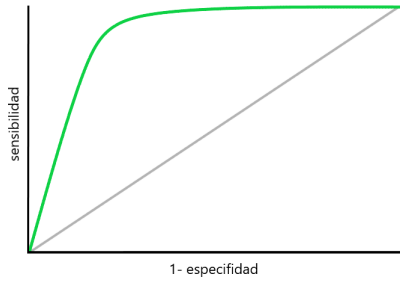


Figura 1.6: Curva ROC.

Los valores de (1.4) van desde $\frac{1}{2}$, en el peor de los casos, hasta 1 en el mejor.
En el caso de tener más de una neurona de salida, se realiza la separación de las diferentes clases y la matriz de confusión pasa a ser una matriz de dimensión *nxn* donde el elemento cm_{ij} pasa a ser el numero de observaciones de la clase i que se clasifican como elementos de la clase j. La curva ROC también es modificada, ya que se calcula para cada categoría la curva ROC de esa categoría frente a todas las demás (1-vs-all), lo que equivale a tener n matrices de confusión. Se podrían desarrollar otros métodos como la comparativa de dos categorías diferentes, lo que equivaldría a tener $n(n-1)$ matrices de confusión.
También aparecen otras medidas como la curva micro-average ROC y la curva macro-average ROC:

- la curva macro-average ROC es la media de las ROC resultantes de calcular las matrices de confusión de 1-vs-all.
- la curva micro-average ROC es el resultado de las medias de la curva ROC resultantes de calcular las las matrices de confusión de 1-vs-all ponderada mediante los tamaños de la muestra de cada categoría de los datos de validación.

1.5. Breve comparación entre RNA y métodos clásicos de estadística

Realizando un resumen de lo presentado en *Redes Neuronales Aplicadas al Análisis de datos*[14], realizaremos una comparativa de los métodos básicos de regresión y las redes neuronales artcifíales. Finalizando con ejemplos prácticos reales donde se usaron redes neuronales y métodos estadísticos comparando los resultados finales obtenidos.

1.5.1. Comparación teórica

Teniendo en cuenta la equivalencia entre el concepto de una función de enlace en un modelo de regresión lineal generalizado (MLG), función que relaciona el predictor lineal y la media de la función de distribución, y la función de activación de la salida en un modelo con el perceptrón.
En este caso también podemos suponer una equivalencia entre el error producido por la red neuronal y la discrepancia proporcionada por la regresión ya que en ambos casos la función es el error cudrático medio:

$$E = \frac{1}{2}\sum_{j=1}^{p}\sum_{k=1}^{r}(d_j^k - y_j^k)^2.$$

La diferencia reside en el método de minimización del error. El perceptrón usa el método de mínimos cuadrados para la estimación de los pesos cuando el MLG utiliza el criterio de máxima verosimilitud aunque podemos intercambiar los métodos empleados en ambos modelos.

En el caso del perceptrón si empleamos el método de máxima verosimilitud la función de error a minimizar es la entropía cruzada:

$$E = -\sum_{j=1}^{p}\sum_{k=1}^{r}[d_j^k log(y_j^k) + (1-d_j^k) log(1-y_j^k)].$$

Otra equivalencia se establece entre el modelo adaline con una sola neurona y la regresión lineal. Ya que su funcion de activación sigue la siguiente regla:

$$y_p^k = \theta^k + \sum_{i=1}^{n} \omega_i^k x_i^k.$$

En caso de que hubiera más de una neurona de salida la red se convierte en un modelo de regresión multivariante.
Un perceptrón simple con función de activación sigmoidal es similar a un modelo de regresión logística. Además, el perceptrón simple con función de activación umbral, es similar a la función lineal discriminante de Fisher, cuando las observaciones a clasificar son normales, con misma covarianza y probabilidades iguales a priori.

Una diferencia entre los modelos de redes neuronales y los modelos estadísticos es que en las redes actúan como “cajas negras", es decir, no se puede obtener información de como actúan las variables ni la importancia de estas, aunque se están realizando estudios mediante la función de error, siendo un área todavía en desarrollo.

1.5.2. Comparación práctica

En esta parte nos centraremos en estudios desarrollados mediante redes neuronales y métodos estadísticos para comparar los resultados obtenidos. Encontraremos tanto estudios reales, como simulados. Realizando una breve explicación de la motivación del estudio y los resultados obtenidos. Una parte que debemos de considerar antes de profundizar, es el hecho de que al igual que en estadística en las redes neuronales debemos tener en cuenta que el modelo obtenido no es universal, otra persona con otro modelo diferente podría obtener resultados diferentes.

Comparación por simulación

Siguiendo lo expresado por Pitarque, Roy y Ruiz [15] podemos encontrar un estudio realizado por simulación donde se intenta realizar una compresión de que método es mejor según la correlación entre las variables de entrada y de salida. Nos encontramos en que podemos clasificar los datos en cuatro categoías:

- Categoría A: a la cual pertenecen los casos cuyas variables de entrada disponen una baja correlación entre ellos pero una alta con la respuesta.
- Categoría B: agrupa a los casos en los que hayamos una alta correlación entre las variables de entrada y baja con las variables de salida.
- Categoría C: se compone de los modelos con baja correlación tanto entre las variables de entrada como las variables de entrada con las de salida.
- Categoría D: formada por los modelos en los que las variables de entrada y las de salida tienen una alta correlación entre si.

En el estudio se realizó una separación de dos objetivos la predicción en la clasificación. Para los modelos de redes se utilizó un modelo de perceptrón multicapa backpropagation y para los modelos de regresión la regresión logística. Se observó que únicamente en la tarea de predicción bajo la categoría A la regresión obtiene un resultado poco mejor, en el resto de las tareas de predicción funcionaron de forma similar. Mientras que en la tarea de clasificación las redes ofrecen mejores resultados en todas ellas.

Torneado en seco

Ahora vamos a dar un ejemplo más concreto y real en el que se usaron modelos para predecir la rugosidad del acero tras su torneado en seco [16]. La rugosidad del acero, *Ra*, es una medida altamente relacionada con la calidad de este material, puesto que afecta altamente a su desgaste, por ello en la industria se puede encontrar diversos estudios para predecir la *Ra* en función de variables como velocidad de corte, angulo de ataque, vibración de la herramienta... . En [16] se proponen el algoritmo de perceptrón multicapa backpropagation y la regresión no lineal múltiple como método estadístico.
En nuestro caso particular se realizó un estudio estadístico previo mediante un análisis tipo Anova para buscar los factores más influyentes en relación con la *Ra*. Las variables escogidas fueron el avance por revolución (mm/rev), calidad y material de la herramienta, velocidad (m/min) y tiempo (min).
Bajo estas condiciones ambos métodos ofrecen resultados satisfactorios en relación con lo obtenido experimentalmente, aunque los resultados ofrecidos por las redes neuronales son ligeramente superiores a los ofrecidos por la regresión.

Previsión consumo eléctrico

Kaytez, Taplamacioglu, Çam y Hardalac en [17], estudian el consumo futuro de electricidad en Turquía, ya que se necesita la importación de esta, una sobreestimación del consumo desemboca en un gasto económico mayor de lo necesario, y subestimar el consumo produce que los costes de la electricidad necesaria para completar el cupo sean mayores y se generan cortes de energía.
Para la predicción de consumo se usan redes neuronales y regresión lineal múltiple junto con el método LS-SVM, pero solo nos centraremos en las RNA y la RLM. Las variables utilizadas en este estudio fueron la capacidad instalada, cantidad de electricidad bruta generada, población y la cantidad total de gente suscrita a la red, estas variables fueron tomadas como independientes.
Para las redes se utiliza un algoritmo de tipo backpropagation con dos capas ocultas 10 y 9 neuronas en cada una. En la primera capa tenemos la identidad como función de activación y en la segunda la función sigmoidal. Para la función de aprendizaje se utiliza el algoritmo Levenberg-Mardquart (Moré [18]).
Las medidas para observar el error fueron el error máximo, el error cuadrático medio (junto con la raíz de este) y la suma de los cuadrados del error y se tuvieron en cuenta los resultados proporcionados por la curva ROC. Se pudo observar una gran superioridad de las redes neuronales frente a la regresión lineal múltiple.

Capítulo 2

Redes neuronales convolucionales: reconocimiento de imágenes por ordenador.

Una vez introducida la idea general de las redes neuronales vamos a profundizar la idea introduciendo una técnica más avanzada, como es el caso de las redes neuronales convolucionales. Desarrollaremos la parte teórica y finalizaremos con una parte práctica de reconocimiento de imágenes por ordenador.

2.1. Teoría de las redes neuronales convolucionales

Las redes neuronales convolucionales (convolutional neural networks, CNN o ConvNets) son una herramienta reciente de la tecnología. Son un tipo de red neuronal, que surge como variación del perceptrón multicapa, la diferencia se encuentra en el uso de capas de convolución y de subsampling, ideas que desarrollaremos más adelante.
Están especialmente diseñadas para el procesamiento de datos de dos dimensiones, sobretodo reconocimiento de imágenes y señales de voz en espectogramas, también pueden ser modificadas para la realización de tareas en una dimesión o varias dimensiones.
La idea comenzó a ser desarrollada con Kunihiko Fukushima, quien en 1982 desarrolló el neocognitron una red neuronal de tipo backpropagation que imita el proceso del cortex visual, como se puede ver en [19]. En 1998 Yann LeCun entrenó la red denominada LeNet (presentada en [5]), la cual emplea una estructura de red neuronal convolucional y consiguiendo clasificar imágenes de dígitos escritos a mano con una precisión del 99,3 %. Pero fue en 2012, con la creación de la red AlexNet (Geoffrey Hinton, Ilya Sutskever y Alex Krizhevsky) para el concurso anual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) cuando se observó la potencia de las redes neuronales para la clasificación de imágenes ([20]).
También podemos destacar las redes RGAs (redes generativas antagónicas o GANs, generative adversial networks en inglés) son un tipo de red convolucional, creada por Ian Goodfellow, desarrollada para generar imágenes capaces de engañar a una red neuronal. Para conseguir este objetivo se enfrentan dos redes neuronales en un "juego de suma cero" una generando imágenes con el objetivo de hacer que la otra red realice una mala clasificación.
Antes de la explicación del funcionamiento de las CNN vamos realizar una explicación de las capas de convolución y de las capas de subsampling.

2.1.1. Capas de convolución

Primero procederemos a introducir la convolución y la adaptaremos al tipo de convolución utilizada en las redes neuronales, para lo cual nos basaremos en [22].
La convolución es un tipo de operación lineal, en su forma más general es la operación de dos funciones,$x(t)$ y $w(t)$, denotada por $(x*w)(t)$ o $s(t)$:

$$s(t) = (x*w)(t) = \int x(a)w(t-a)da.$$

Si enlazamos con la nomenclatura de las CNN entonces $x(t)$ sería la entrada y $w(t)$ el núcleo o kernel y $s(t)$ la salida producida por la convolución la denominaremos **mapa de características**.
Por el hecho de discretizar los datos si t toma valores enteros y w solo esta definido en valores enteros, entonces:

$$s(t) = (x*w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$

Una cosa importante que debemos tener en cuenta en las CNN es la dimensión de entrada que estamos trabajando, por lo cual deberemos adaptar el núcleo a la dimensión necesaria (tres en el caso de imágenes a color, dos en imágenes en blanco y negro).
En el caso de las bidimensiones la fórmula de convolución, con entrada I y con núcleo K, es:

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

La cual es conmutativa:

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

Computacionalmente se suele utilizar esta segunda fórmula ya que hay menos variación en el rango de valores m y n (por ser de menor dimensión el núcleo que la entrada). Aunque en la inteligencia articifial se utiliza una fórmula llamada correlación cruzada, pero por convenio también se denomina convolución:

$$s(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \qquad i,j = 0,1...,$$

$$s(i,j) = \sum_m \sum_n I(i+m-1, j+n-1)K(m,n) \qquad i,j = 1,2....$$

Para acelerar el proceso podemos realizar la convolución con paso (stride en inglés) p definida de la siguiente forma:

$$s(i,j) = \sum_m \sum_n I([i-1]p+m, [j-1]p+n)K(m,n) \qquad i,j = 1,2....$$

En el caso de las imágenes a color tenemos tres dimensiones, tres capas bidimensionales (las correspondientes a las capas de los canales RGB). En este caso la convolución equivaldría a la realización de tres convoluciones bidimensionales, una a la capa de rojo, una a la verde y otra a la azul, y sumar los resultados.
Un problema en la convolución con paso p, es encajar las dimensiones de la entrada con la dimensión núcleo. Una solución es el proceso de zero-padding (rellenar con ceros), para aumentar la dimensión de la entrada se empieza añadiendo una columna de ceros en la derecha o abajo donde fuera necesario, en caso de necesitar más se rellenaría con ceros en la izquierda o arriba. Vamos a esclarecer este hecho con dos ejemplos:

1. Un ejemplo de una matriz de entrada 3x3 y un núcleo 2x2 con paso 2.

Entrada:

a	b	c
d	e	f
g	h	i

Núcleo:

1	-1
2	0

Para realizar la convolución debemos poner en la entrada una columna de ceros y una fila inferior también de ceros.

a	b	c	0
d	e	f	0
g	h	i	0
0	0	0	0

Obteniendose el siguiente mapa de caracteríticas:

$a-b+2d$	$c+2f$
$g-h$	i

2. Un ejemplo de una matriz de entrada 4x4 y un núcleo 3x3 con paso 3.

Entrada:

-1	0	0	-1
1	1	1	-1
0	-1	1	0
0	1	1	0

Núcleo:

1	0	0
0	1	0
0	0	-1

Realizamos el proceso de zero-padding:

0	0	0	0	0	0
0	-1	0	0	-1	0
0	1	1	1	-1	0
0	0	-1	1	0	0
0	0	1	1	0	0
0	0	0	0	0	0

Obteniendo como resultado:

-2	-1
0	1

En los ejemplos expuestos podemos observar que la dimensión de la matriz resultante del proceso de convolución es menor que la de entrada, por ello se suele igualar las dimensiones realizando un proceso de zero-padding, o se completa manteniendo los bordes de la entrada original. Una capa convolucional consiste en realizar varias convoluciones añadiendo un término de sesgo a cada una de las entradas obteniendo diferentes mapas de características.

2.1.2. Pooling.

Una vez realizada la convolución nos interesa realizar una reducción del volumen de datos, de esta tarea se encarga el submuestreo o pooling, en cierta manera interactúa dentro de la red del mismo modo que una capa convolucional, haciendo operaciones en pequeñas regiones de la matriz de entrada, acumulando los elementos de las capas de características.

Max-Pooling.

Dada una matriz A_{axa} podemos definir el proceso de Max-Pooling con una amplitud k y un stride p como la matriz $P(i,j)$ tal que:

$$P(i,j) = \max_{n,m=1,\dots,k} A[(i-1)p+m,(j-1)p+n].$$

Si fueran necesarias más filas o columnas para el desarrollo del algoritmo se pueden incluir más filas y columnas mediante el métododе zero-padding
El algoritmo de Max-Pooling usualmente utilizado es el aquel con una amplitud y stride 2.

Average-Pooling

Es similar al método de Max-Pooling, salvo que en vez de ser la salida el máximo se trata de la media aritmética.

$$P(i,j) = \frac{1}{k^2} \sum_{n,m=1,\dots,k} A[(i-1)p+m,(j-1)p+n].$$

2.1.3. Estructura de una red convolucional.

La arquitectura de una red neuronal convolucional empieza aplicando una capa convolucional y una capa de pooling, siguiendo este proceso repetidamente hasta obtener un conjunto de matrices, tal que al poner todos los elementos como un vector (proceso de flattering), tenga una cantidad computacionalmente viable de elementos como para ser introducido como entrada de una red neuronal.

2.1.4. Procesamiento de imágenes por ordenador.

En esta parte se ha desarrollado como trabajar con matrices y tensores en redes neuronales convolucionales, vamos a ofrecer una idea general de como se transforman las imágenes en matrices y tensores. La base reside en sacar las características de color, para ello se realiza un mallado de la imagen; a cada cuadrado procedente de dicho mallado se denomina pixel.
Si la imagen fuera en blanco y negro a cada pixel se le asocia un número que coincide con la tonalidad del pixel, 0 es asociado al color negro y 255 al blanco. Por tanto tenemos una matriz con tantos elementos como pixeles tiene la imagen.
Si la imagen fuera en color entonces nuestra imagen no es una combinación de blancos y negros si no que se trata de una combinación de tres colores, rojo verde y azul (canal RGB). Por ello podemos definir cada pixel como un vector de tres dimensiones, poniendo en la primera componente del vector la intensidad de rojo, de verde en la segunda y azul en la tercera, todas las componentes se mueven entre valores de 0 a 255 donde el 0 representa el negro o color más oscuro correspondiente y el 255 el color blanco o más claro correspondiente. También esta definido el canal CMYK, un caso similar al caso RGB, pero con los canales cian, magenta, amarillo y negro que también actúan con el mismo comportamiento, pero en el espacio $[0,254]^4$, este será el formato usado en la parte práctica para el procesado de las imágenes.

2.2. Parte práctica: clasificación de tickets

Dentro de la empresa Efor, donde he desarrollado la parte práctica del trabajo, los empleados en ocasiones deben realizar viajes a otras ciudades, lo cual supone gastos (comidas, autopistas, taxis, ...) que son abonados por el empleado hasta final de mes, cuando se entregan los tickets a la empresa junto con una hoja excel donde se detallan los conceptos de gastos y así recupera el importe invertido.
La idea consiste en realizar una clasificación de los tickets a través de imágenes obtenidas mediante el

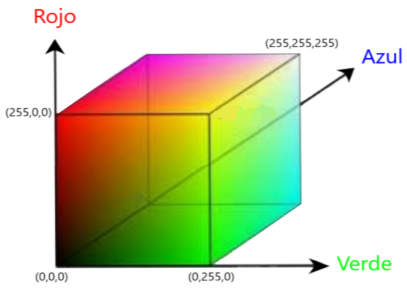


Figura 2.1: Espacio de color RGB (imagen modificada a partir de [23]).

escaneado de estos, con el objetivo de automatizar el proceso y generar un ahorro de tiempo y comodidad a los empleados.
Realizaremos una primera prueba con la clasificación de cuatro grupos de tickets. Tickets de un restaurante (Las galias), de taxi, de parking y de gasolinera (Repsol). Para ello se realiza un escaneado de estos tickets para generar nuestra base de datos, la cual consta de 63 tickets del restaurante, 63 de parking, 56 de gasolinera y 57 de taxi, un total de 239 tickets. Necesitamos tener una carpeta con los datos de entrenamiento, en la cual estarán nuestros datos metidos en otras carpetas separados según a la categoría a la que pertenecen. Además dispondremos de 9 tickets extra (uno de comida, cuatro de parking, dos de gasolinera y dos de taxi) para desarrollar un posible programa de cómo funcionaría una aplicación de clasificación de tickets.

2.2.1. Abordaje del problema.

Las imágenes serán introducidas y redimensionadas a $325x120$ pixels con los cuatro canales de color y procederemos a normalizarlas al espacio $[0,1]^{325x120x4}$.
Para realizar el programa nos basaremos en una librería denominada Keras (un manual se puede encontrar en [24]), la cual junto con TensorFlow es la más usada para la programación de redes neuronales. Probaremos diversas estructuras de redes neuronales y tendremos en cuenta la matriz de confusión, la precisión y el AUC para seleccionar el modelo que creamos conveniente.
En lo referente a las librerías, Keras es una librería básica en la programación de redes neuronales, ofrece una gran variedad y versatilidad, las estructuras empleadas son sencillas e intuitivas, las cuales nos permiten generar las capas de convolución y max pooling y las capas del modelo con su función de activación y regla de aprendizaje. Otra librería es sklearn la cual permite la implemntación de los datos de entrenamiento(gracias a la creación de las etiquetas y la separción de los datos de entrenamiento) y también la validación del entrenamiento (la matriz de confusión y las curvas ROC, AUC, micro y macro), una introducción la podemos hallar en [25].
Todas las redes tendrán un elemento en común y es que la capa de salida tendrá una función de activación softmax, o exponencial normalizada, con cuatro neuronas, la cual nos da una probabilidad de pertenencia a cada clase. También la función de aprendizaje sera común, el AdaGrad, al igual que la regla de propagación, la cual será la usual y la función de error será la entropía cruzada (explicada en 1.5.1).
Ha habido una gran variedad en los modelos probados:

- Respecto a las capas ocultas se han probado de dos a tres capas ocultas y el número de neuronas en cada capa ha estado entre 8 y 64.
- Los filtros de convolución se ha probado cantidades entre 16 y 100 filtros de convolución.
- Las funciones de activación se han escogido entre la tangente hiperbólica, la función lineal, la ReLU y la sigmoidal.

Para considerar qué modelos fueron mejores se tuvo en cuenta tanto la precisión en los datos de validación como la Micro-AUC y la Macro-AUC.
La salida obtenida mediante el modelo será un vector con cuatro componentes dando los valores producidos por la capa de salida, de la cual redondeando los valores obtendremos el valor 1 para la clase a la que se considera que debería pertenecer el ticket y 0 para el resto.
Vamos a realizar una explicación del funcionamiento de las principales funciones utilizadas en nuestro programa:

- Image.open(): se encarga de abrir cada imagen.
- .resize((r,s), Image.ANTIALIAS): redimensiona la imagen a una dimensión de sxr, gracias a Image.ANTIALIAS conservamos la estructura de la imagen, hay otros filtros como NEAREST, el cual recorta la imagen, BILINEAR o BICUBIC los cuales realizan interpolación al redimensionar la imagen.
- np.asarray(): nos permite transformar la imagen en un conjunto de $325x120$ vectores de 4. dimensiones.
- train_test_split(): permite realizar la separación de los datos, en los datos de entrenamiento y los datos de validación. Nosotros al realizar una separación del 20 % nos quedamos con 191 datos de entrenamiento y 48 de validación. Para poder realizar la validación cruzada realizamos una nueva separación de los datos de entrenamiento del 20 % quedando los datos de entrenamiento divididos en 152 y 39.
- Sequential(): es la arquitectura más sencilla ya que implica que todas las neuronas de una capa estarán conectadas con todas las neuronas de la capa siguiente, para estructuras más complejas se debe usar *keras functional API*.
- .add(Conv2D(50, kernel_size=$(3,3)$, activation='tanh', padding='same', input_shape=$(s,r,4)$)): representa la capa de convolución, obtendremos 50 filtros diferentes los cuales serán obtenidos con un núcleo $3x3$ y una función de activación tangente hiperbólica, el padding indica la dimensión de los filtros de salida same es que tengan la misma dimensión que la entrada (otras opciones son valid, la cual no realiza un proceso de zero-padding, o casual, con la cual la salida no depende de la entrada).
- .add(LeakyReLU($alpha = 0{,}1$)): permite una reducción de la salida de la neurona cuando no este activa:

$$y = \begin{cases} alpha * F(s_k) & si \quad F(s_k) \leq 0 \\ \\ F(s_k) & si \quad 0 < F(s_k) \end{cases}$$

- .add(MaxPooling2D((2, 2),padding='same')): nos permite realizar el proceso de max-pooling con un núcleo de dimensión $2x2$.
- .add(Dropout$(1-q)$): desactiva las neuronas con una probabilidad q para evitar el sobreajuste.
- .add(Flatten()): pone los datos como un vector.
- .add(Dense(50, activation='linear')): provee de una capa de 50 neuronas con activación lineal.
- .compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adagrad(lr=INIT_LR, decay=INIT_LR / 100),metrics=['accuracy']): con esta función establecemos que la regla de aprendizaje es el AdaGrad con una caída INIT_LR/100 para cada época, la función de perdida es la proporcionada por la entropía cruzada y como medida de acierto utilizaremos la precisión.

- .fit(train_X, train_label, batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X, valid_label)): realiza el entrenamiento de tipo batch mode con un tamaño batch_size, y número de épocas epochs, verbose indica la salida por pantalla (verbose=0 no ofrece nada por pantalla, 1 permite ver la barra de progreso y 2 solo ofrece la salida final) y validation_data son los datos previamente separados para el entrenamiento.
- .evaluate(test_X, test_Y_one_hot, verbose=1): nos permite evaluar nuestro modelo en los datos de validación ofreciendo la precisión obtenida.
- confusion_matrix(test_Y_one_hot.argmax(axis=1),Y_pred.argmax(axis=1)): ofrece la matriz de confusión.
- roc_curve(test_Y_one_hot[:,i],Y_pred[:,i]): calcula la curva roc para la categoría *i*.
- auc(fpr[i], tpr[i]): calcula para la categoría *i* el AUC (fpr, false positive ratio y tpr, true positive ratio).

Al final del anexo A mostramos un programa, al cual le introducimos los tickets y genera una salida en la que expresa a que categoría pertenece cada ticket introducido y otra salida con la imagen de cada ticket pero escrito en la esquina superior izquierda la categoría a la que pertenece.
A continuación expondremos los elementos de aquellos modelos que obtuvieron mejores resultados:

	modelo 1	modelo 2	modelo 3	modelo 4
Capa Convolución	45 filtros + ReLU	50 filtros + tanh	80 filtros + lineal	45 filtros + lineal
1ª capa oculta	39 neuronas + ReLU	50 neuronas + lineal	63 neuronas + lineal	39 neuronas + lineal
2ª capa oculta	39 neuronas + ReLU	-	63 neuronas + ReLU	-
Épocas	60	20	40	30
Precisión	0.8333	0.9583	0.9375	0.8958
Micro-AUC	0.95	0.98	0.97	0.98
Macro-AUC	0.99	0.99	0.99	0.99

Cuadro 2.1: Comparativa de modelos

El modelo 2 es el que mejor resultados ha ofrecido, aunque su coste computacional era de los más bajos. El modelo 2 consta de más de 24.000.000 de parámetros ajustables, pero gracias al ordenador disponible (ThinkPad L390) y una alta velocidad de conexión a internet el tiempo de entrenamiento rondaba los tres minutos, a no ser que se entrenaran varias redes a la vez. Su programación detallada se puede ver en el anexo A.

2.2.2. Futuros trabajos

El objetivo sería avanzar en el desarrollo realizado abordando la clasificación en más categorías, para ello se deberá acceder a una base de datos mayor que la generada en este trabajo. También se podría implementar reconocimiento de texto a las imágenes, con la intención de obtener información como los importes y los conceptos que aparecen en los tickets para así poder automatizar el proceso manual que se realiza actualmente.

Bibliografía

[1] MCCULLOCH, WARREN S AND PITTS, WALTER, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics, volumen 5, número 4, 1943, págs. 115-133, disponible en http://aiplaybook.a16z.com/reference-material/mcculloch-pitts-1943-neural-networks.pdf.

[2] ZHANG, LING AND ZHANG, BO, *A geometrical representation of McCulloch-Pitts neural model and its applications*, IEEE Transactions on Neural Networks, IEEE, volumen 10, número 4, 1999, págs. 925-929, disponible en http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.459.7236&rep=rep1&type=pdf.

[3] HEBB, DONALD OLDING, *The organization of behavior: A neuropsychological theory* , Psychology Press, 2005, disponible en https://pure.mpg.de/rest/items/item_2346268/component/file_2346267/content.

[4] SORIA, EMILIO AND BLANCO, ANTONIO *Redes neuronales artificiales*, Autores científico-técnicos y académicos, 2001, págs. 25-33, disponible en https://www.acta.es/medios/articulos/informatica_y_computacion/019023.pdf.

[5] LE CUN, YANN AND JACKEL, LD AND BOSER, B AND DENKER, JS AND GRAF, HP AND GUYON, I AND HENDERSON, D AND HOWARD, RE AND HUBBARD, W, *Handwritten digit recognition: applications of neural net chips and automatic learning*, IEEE Communications Magazine,IEEE, volumen 27, número 11, 1989, págs. 41-46.

[6] https://esacademic.com/dic.nsf/eswiki/994505.

[7] KRÖSE, BEN AND SMAGT, PATRICK *An introduction to neural networks*, 1993, disponible en http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=08EDFDFD11FF58F70D99E18C2D95787B?doi=10.1.1.18.493&rep=rep1&type=pdf.

[8] HEBB, DONALD OLDING, *The organization of behavior: A neuropsychological theory*, Psychology Press, 2005.

[9] FERRAGUT CANALS, LUIS *Optimización numérica*, Universidad de Salamanca, 2017, disponible en https://gredos.usal.es/bitstream/handle/10366/136962/OptimizacionNumerica.pdf;jsessionid=AF3BC6133C54FDD13BCFE1C64BB42C9F?sequence=1.

[10] http://bibing.us.es/proyectos/abreproy/11084/fichero/Memoria+por+cap%C3%ADtulos+%252FCap%C3%ADtulo+4.pdf+

[11] ANDREJ, K AND JANEZ, B AND ANDREJ, K, *Introduction to the Artificial Neural Networks*, Artificial Neural Networks-Methodological Advances and Biomedical Applications, 2011, disponible en https://www.intechopen.com/books/artificial-neural-networks-methodological-advances-and-biomedical-applications/introduction-to-the-artificial-neural-networks.

[12] SVOZIL, DANIEL AND KVASNICKA, VLADIMIR AND POSPICHAL, JIRI, *Introduction to multilayer feed-forward neural networks*, Chemometrics and intelligent laboratory systems, Elsevier, volumen 39,número 1, 1997, disponible en https://www.sciencedirect.com/science/article/abs/pii/S0169743997000610.

[13] HORNIK, KURT, *Approximation capabilities of multilayer feedforward networks*, Neural networks, Elveiser, volumen 4, número 2, 1991, págs 251-257, disponible en http://www.vision.jhu.edu/teaching/learning/deeplearning18/assets/Hornik-91.pdf.

[14] MONTAÑO MORENO, JUAN JOSÉ AND OTHERS, *Redes neuronales artificiales aplicadas al análisis de datos*, Universitat de les Illes Balears, 2017, págs. 40-61, disponible en http://dspace.uib.es/xmlui/bitstream/handle/11201/2511/Montano_Moreno_JuanJose.pdf?sequence=1.

[15] PITARQUE, ALFONSO AND ROY, JUAN FRANCISCO AND RUIZ, JUAN CARLOS, *Redes neurales vs modelos estadísticos: Simulaciones sobre tareas de predicción y clasificación*, Psicológica, 1998, págs. 387-400, disponible en https://www.uv.es/PSICOLOGICA/articulos3.98/pitarque.pdf.

[16] MORALES-TAMAYO, YOANDRYS AND ZAMORA-HERNÁNDEZ, YUSIMIT AND VÁSQUEZ-CARRERA, PACO AND PORRAS-VÁSCONEZ, MARIO AND BÁRZAGA-QUESADA, JOAO AND LÓPEZ-BUSTAMANTE, RINGO, *Comparación entre redes neuronales artificiales y regresión múltiple para la predicción de la rugosidad superficial en el torneado en seco*, 2018, disponible en http://dspace.ups.edu.ec/handle/123456789/15124.

[17] KAYTEZ, FAZIL AND TAPLAMACIOGLU, M CENGIZ AND CAM, ERTUGRUL AND HARDALAC, FIRAT, *Forecasting electricity consumption: A comparison of regression analysis, neural networks and least squares support vector machines*, International Journal of Electrical Power & Energy Systems, Elveiser, volumen 67, 2015, págs. 431-438, disponible en https://www.researchgate.net/profile/Ertugrul_Cam/publication/270006584_Forecasting_electricity_consumption_A_comparison_of_regression_analysis_neural_networks_and_least_squares_support_vector_machines/links/5a840903aca272d6501f606b/Forecasting-electricity-consumption-A-comparison-of-regression-analysis-neural-networks-pdf.

[18] MORÉ, JORGE J, *The Levenberg-Marquardt algorithm: implementation and theory*, Numerical analysis, Springer, 1978, apágs. 105-116, disponible en https://www.osti.gov/servlets/purl/7256021/.

[19] FUKUSHIMA, KUNIHIKO, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Biological cybernetics, Springer, volumen 36, número 4, 1980, págs. 193-202, disponible en https://link.springer.com/article/10.1007/BF00344251.

[20] https://ml4a.github.io/ml4a/convnets/.

[21] ORTEGA ASENSIO, ERNESTO AND OTHERS, *Sistema de reconocimiento de gestos de la mano basado en procesamiento de imagen y redes neuronales convolucionales*, 2017, disponible en http://repositorio.upct.es/handle/10317/6564.

[22] GOODFELLOW, IAN AND BENGIO, YOSHUA AND COURVILLE, AARON, *Deep learning*, MIT press, 2016, págs. 326-366, disponible en http://www.deeplearningbook.org/contents/convnets.html.

[23] http://bibing.us.es/proyectos/abreproy/11875/fichero/Proyecto+Fin+de+Carrera%252F3.Espacios+de+color.pdf

[24] GULLI, ANTONIO AND PAL, SUJIT, *Deep Learning with Keras*, Packt Publishing Ltd, 2017.

[25] ABRAHAM, ALEXANDRE AND PEDREGOSA, FABIAN AND EICKENBERG, MICHAEL AND GERVAIS, PHILIPPE AND MUELLER, ANDREAS AND KOSSAIFI, JEAN AND GRAMFORT, ALEXANDRE AND THIRION, BERTRAND AND VAROQUAUX, GAËL, *Machine learning for neuroimaging with scikit-learn*, Frontiers in neuroinformatics, Frontiers, volumen 8, pág. 14, 2014, disponible en https://www.frontiersin.org/articles/10.3389/fninf.2014.00014/full.

Anexo A

1 Modelo 2

Generamos un modelo.

Descargamos las librerias necesarias.

[1]:
```
import numpy as np
import os
import sys
import re
import matplotlib.image as img
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from sklearn.metrics import  confusion_matrix
from sklearn.utils.multiclass import unique_labels


import keras
from keras.utils import to_categorical
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
from resizeimage import resizeimage
import scipy.misc
from PIL import Image, ImageDraw, ImageFont
```

```
Using TensorFlow backend.
```

[2]:
```
import chart_studio.plotly as py
import plotly.graph_objects as go
from itertools import cycle
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
```

```
from scipy import interp
```

[3]:
```
>>> os.getcwd()
'/home/usuario'
```

[3]:
```
'/home/usuario'
```

Leemos las imágenes y las redimensionamos.

[4]:
```
dirname = os.path.join(os.getcwd(), 'tickets3')
imgpath = dirname + os.sep

images = []
directories = []
dircount = []
prevRoot=''
cant=0

print("leyendo imagenes de ",imgpath)
r=int(120)
s=int(325)
for root, dirnames, filenames in os.walk(imgpath):
    for filename in filenames:
        if re.search("\.(jpg|jpeg|PNG|bmp|tiff)$", filename):
            cant=cant+1
            filepath = os.path.join(root, filename)
            img= Image.open(filepath)


            reducida = img.resize((r,s), Image.ANTIALIAS)
            img1=np.asarray(reducida,dtype=np.float32)
            images.append(img1)
            b = "Leyendo..." + str(cant)
            print (b, end="\r")
            if prevRoot !=root:
                print(root, cant)
                prevRoot=root
                directories.append(root)
                dircount.append(cant)
                cant=0
dircount.append(cant)


dircount = dircount[1:]
dircount[0]=dircount[0]+1

print('Directorios leidos:',len(directories))
print("Imagenes en cada directorio", dircount)
print('suma Total de imagenes en subdirs:',sum(dircount))
```

```
leyendo imagenes de  C:\Users\egarcia\Prueba_3\tickets3\
C:\Users\egarcia\Prueba_3\tickets3\las galias 1
C:\Users\egarcia\Prueba_3\tickets3\par_mat 62
C:\Users\egarcia\Prueba_3\tickets3\repsol 63
C:\Users\egarcia\Prueba_3\tickets3\taxi2 56
Directorios leidos: 4
Imagenes en cada directorio [63, 63, 56, 57]
suma Total de imagenes en subdirs: 239
```

[5]:
```
labels=[]
indice=0
for cantidad in dircount:
    for i in range(cantidad):
        labels.append(indice)
    indice=indice+1
print("Cantidad etiquetas creadas: ",len(labels))

deportes=[]
indice=0
for directorio in directories:
    name = directorio.split(os.sep)
    print(indice , name[len(name)-1])
    deportes.append(name[len(name)-1])
    indice=indice+1

y = np.array(labels)
X = np.array(images, dtype=np.uint8)


classes = np.unique(y)
nClasses = len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)
```

```
Cantidad etiquetas creadas:  239
0 las galias
1 par_mat
2 repsol
3 taxi2
Total number of outputs :  4
Output classes :  [0 1 2 3]
```

Creamos el conjunto de entrenamiento y el de validación.

[6]:
```
train_X,test_X,train_Y,test_Y = train_test_split(X,y,test_size=0.2)
print('Training data shape : ', train_X.shape, train_Y.shape)
print('Testing data shape : ', test_X.shape, test_Y.shape)
```

```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255.
test_X = test_X / 255.


train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

print('Original label:', train_Y[0])
print('After conversion to one-hot:', train_Y_one_hot[0])

train_X,valid_X,train_label,valid_label = train_test_split(train_X,␣
 ↪train_Y_one_hot, test_size=0.2, random_state=13)

print(train_X.shape,valid_X.shape,train_label.shape,valid_label.shape)
```

```
Training data shape :  (191, 325, 120, 4) (191,)
Testing data shape :  (48, 325, 120, 4) (48,)
Original label: 2
After conversion to one-hot: [0. 0. 1. 0.]
(152, 325, 120, 4) (39, 325, 120, 4) (152, 4) (39, 4)
```

Ponemos las estructura de nuestra red.

[7]:
```
INIT_LR = 1e-3
epochs = 20
batch_size = 10

ticket_model = Sequential()
ticket_model.add(Conv2D(50, kernel_size=(3,␣
 ↪3),activation='tanh',padding='same',input_shape=(s,r,4)))
ticket_model.add(LeakyReLU(alpha=0.1))
ticket_model.add(MaxPooling2D((2, 2),padding='same'))
ticket_model.add(Dropout(0.5))

ticket_model.add(Flatten())
ticket_model.add(Dense(50, activation='linear'))
ticket_model.add(LeakyReLU(alpha=0.1))
ticket_model.add(Dropout(0.5))
ticket_model.add(Dense(nClasses, activation='softmax'))

ticket_model.summary()

ticket_model.compile(loss=keras.losses.categorical_crossentropy,␣
 ↪optimizer=keras.optimizers.Adagrad(lr=INIT_LR, decay=INIT_LR /␣
 ↪100),metrics=['accuracy'])
```

```
WARNING:tensorflow:From
C:\Users\egarcia\AppData\Local\Continuum\anaconda3\lib\site-
packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From
C:\Users\egarcia\AppData\Local\Continuum\anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 325, 120, 50)      1850
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 325, 120, 50)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 163, 60, 50)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 163, 60, 50)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 489000)            0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                24450050
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 50)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 204
=================================================================
Total params: 24,452,104
Trainable params: 24,452,104
Non-trainable params: 0
_________________________________________________________________
```

Realizamos el entrenamiento.

[8]:
```
ticket_train_dropout = ticket_model.fit(train_X, train_label,␣
↪batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,␣
↪valid_label))
```

```
WARNING:tensorflow:From
```

```
C:\Users\egarcia\AppData\Local\Continuum\anaconda3\lib\site-
packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Train on 152 samples, validate on 39 samples
Epoch 1/20
152/152 [==============================] - 31s 201ms/step - loss: 10.7021 - acc:
0.2434 - val_loss: 8.1781 - val_acc: 0.1795
Epoch 2/20
152/152 [==============================] - 31s 204ms/step - loss: 11.0300 - acc:
0.2237 - val_loss: 8.2073 - val_acc: 0.2564
Epoch 3/20
152/152 [==============================] - 30s 195ms/step - loss: 4.1996 - acc:
0.4145 - val_loss: 0.8554 - val_acc: 0.6923
Epoch 4/20
152/152 [==============================] - 29s 189ms/step - loss: 0.7084 - acc:
0.7566 - val_loss: 0.8491 - val_acc: 0.7179
Epoch 5/20
152/152 [==============================] - 28s 187ms/step - loss: 0.4460 - acc:
0.8684 - val_loss: 0.8056 - val_acc: 0.7436
Epoch 6/20
152/152 [==============================] - 28s 184ms/step - loss: 0.3050 - acc:
0.8750 - val_loss: 0.6504 - val_acc: 0.7692
Epoch 7/20
152/152 [==============================] - 29s 193ms/step - loss: 0.1832 - acc:
0.9408 - val_loss: 0.6822 - val_acc: 0.8462
Epoch 8/20
152/152 [==============================] - 30s 195ms/step - loss: 0.1360 - acc:
0.9605 - val_loss: 0.8781 - val_acc: 0.7436
Epoch 9/20
152/152 [==============================] - 29s 190ms/step - loss: 0.1217 - acc:
0.9737 - val_loss: 0.7202 - val_acc: 0.8462
Epoch 10/20
152/152 [==============================] - 29s 191ms/step - loss: 0.1018 - acc:
0.9605 - val_loss: 0.6915 - val_acc: 0.8718
Epoch 11/20
152/152 [==============================] - 29s 191ms/step - loss: 0.0914 - acc:
0.9737 - val_loss: 1.0366 - val_acc: 0.7436
Epoch 12/20
152/152 [==============================] - 27s 180ms/step - loss: 0.0651 - acc:
0.9868 - val_loss: 0.7345 - val_acc: 0.8462
Epoch 13/20
152/152 [==============================] - 29s 188ms/step - loss: 0.0576 - acc:
0.9803 - val_loss: 0.7550 - val_acc: 0.8462
Epoch 14/20
152/152 [==============================] - 29s 191ms/step - loss: 0.0565 - acc:
```

```
0.9803 - val_loss: 0.7361 - val_acc: 0.8718
Epoch 15/20
152/152 [==============================] - 29s 188ms/step - loss: 0.0442 - acc:
1.0000 - val_loss: 0.7239 - val_acc: 0.8718
Epoch 16/20
152/152 [==============================] - 31s 203ms/step - loss: 0.0567 - acc:
0.9868 - val_loss: 0.6965 - val_acc: 0.8718
Epoch 17/20
152/152 [==============================] - 28s 183ms/step - loss: 0.0487 - acc:
0.9934 - val_loss: 0.7926 - val_acc: 0.7949
Epoch 18/20
152/152 [==============================] - 27s 180ms/step - loss: 0.0310 - acc:
0.9934 - val_loss: 0.7373 - val_acc: 0.8718
Epoch 19/20
152/152 [==============================] - 27s 178ms/step - loss: 0.0475 - acc:
0.9934 - val_loss: 0.7334 - val_acc: 0.8462
Epoch 20/20
152/152 [==============================] - 23s 150ms/step - loss: 0.0241 - acc:
1.0000 - val_loss: 0.7161 - val_acc: 0.8718
```

Le aplicamos el modelo obtenido en el entrenamiento a los datos de validación.

[9]:
```
test_eval = ticket_model.evaluate(test_X, test_Y_one_hot, verbose=1)

print('Funcion de perdida:', test_eval[0])
print('Precisión:', test_eval[1])
```

```
48/48 [==============================] - 2s 41ms/step
Funcion de perdida: 0.28993281722068787
Precisión: 0.9583333333333334
```

[10]:
```
Y_pred=ticket_model.predict(test_X)
```

[11]:
```
cm = confusion_matrix(test_Y_one_hot.argmax(axis=1),Y_pred.argmax(axis=1))
print('Matriz de confusión:\n', cm)
```

```
Matriz de confusión:
 [[12  1  0  0]
 [ 0 18  0  0]
 [ 0  0 11  0]
 [ 0  0  1  5]]
```

Calculamos las curvas ROC.

[12]:
```
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(nClasses):
```

```
    fpr[i], tpr[i], _ = roc_curve(test_Y_one_hot[:,i],Y_pred[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])


fpr["micro"], tpr["micro"], _ = roc_curve(test_Y_one_hot.ravel(), Y_pred.
 ↪ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(nClasses)]))


mean_tpr = np.zeros_like(all_fpr)
for i in range(nClasses):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])


mean_tpr /= nClasses

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])


data = []
lw=2
trace1 = go.Scatter(x=fpr["micro"], y=tpr["micro"],
                    mode='lines',
                    line=dict(color='deeppink', width=lw, dash='dot'),
                    name='curva micro-average ROC (area = {0:0.2f})'
                           ''.format(roc_auc["micro"]))
data.append(trace1)

trace2 = go.Scatter(x=fpr["macro"], y=tpr["macro"],
                    mode='lines',
                    line=dict(color='navy', width=lw, dash='dot'),
                    name='curva macro-average ROC (area = {0:0.2f})'
                          ''.format(roc_auc["macro"]))
data.append(trace2)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue','green'])
for i, color in zip(range(nClasses), colors):
    trace3 = go.Scatter(x=fpr[i], y=tpr[i],
                        mode='lines',
                        line=dict(color=color, width=lw),
                        name='curva ROC de la clase {0} (area = {1:0.2f})'
                        ''.format(i, roc_auc[i]))
    data.append(trace3)
```

```
trace4 = go.Scatter(x=[0, 1], y=[0, 1],
                    mode='lines',
                    line=dict(color='black', width=lw, dash='dash'),
                    showlegend=False)

layout = go.Layout(title='ROC multiclase',
                   xaxis=dict(title='Ratio de falsos positivos'),
                   yaxis=dict(title='Ratio de verdaderos positivos'))

fig = go.Figure(data=data, layout=layout)
```

[13]:
```
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(nClasses)]))
```

[14]:
```
fig
```

Procedemos a introducir los datos de prueba en el modelo.

[15]:
```
dirname = os.path.join(os.getcwd(), 'validar')
imgpath = dirname + os.sep

images = []
directories = []
dircount = []
prevRoot=''
cant=0

print("leyendo imagenes de ",imgpath)

for root, dirnames, filenames in os.walk(imgpath):
    for filename in filenames:
        if re.search("\.(jpg|jpeg|PNG|bmp|JPEG|tiff)$", filename):
            cant=cant+1
            filepath = os.path.join(root, filename)
            img= Image.open(filepath)

            r=int(120)
            s=int(325)
            reducida = img.resize((r,s), Image.ANTIALIAS)
            img1=np.asarray(reducida,dtype=np.float32)
            images.append(img1)

            R1= np.array(images, dtype=np.uint8)
            test_R1 = R1.astype('float32')
            test_R1= test_R1 / 255.
            predicted_classes = ticket_model.predict(test_R1)
            prediccion = predicted_classes[0]
            prediccion
```

```
leyendo imagenes de  C:\Users\egarcia\Prueba_3\validar\
```

[16]:
```
predicted_classes.round()
```

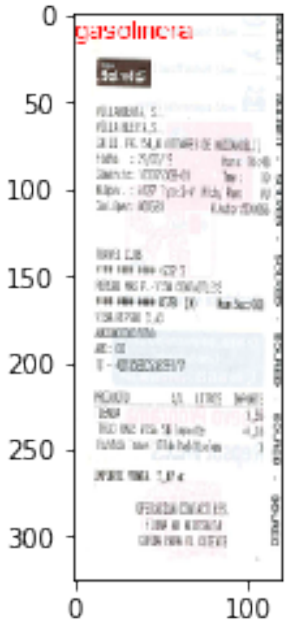
[16]:
```
array([[0., 0., 1., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [1., 0., 0., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.]], dtype=float32)
```

[17]:
```
ticket_model.save("tickets_2.h5py")
```

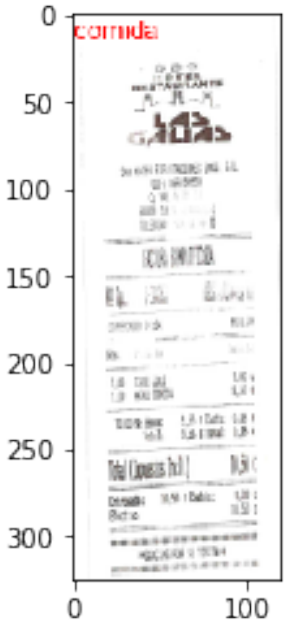
[18]:
```
dirname = os.path.join(os.getcwd(), 'validar')
imgpath = dirname + os.sep

images = []
directories = []
dircount = []
prevRoot=''
cant=0

print("leyendo imagenes de ",imgpath)

for root, dirnames, filenames in os.walk(imgpath):
    for filename in filenames:
        if re.search("\.(jpg|jpeg|PNG|bmp|JPEG|tiff)$", filename):
            cant=cant+1
            filepath = os.path.join(root, filename)
            img= Image.open(filepath)


            reducida = img.resize((r,s), Image.ANTIALIAS)
            img1=np.asarray(reducida,dtype=np.float32)
            images.append(img1)

            R1= np.array(images, dtype=np.uint8)
            test_R1 = R1.astype('float32')
            test_R1= test_R1 / 255.
            predicted_classes = ticket_model.predict(test_R1).argmax(axis=1)
            categoria=predicted_classes[-1]

            if categoria==0:
                clase='comida'
            elif categoria==1:
                clase='parking'
            elif categoria==2:
```

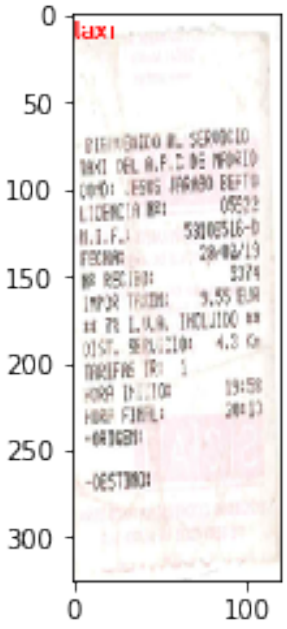
```
            clase='gasolinera'
        else:
            clase='taxi'

        prediccion = predicted_classes[-1]
        draw = ImageDraw.Draw(reducida)
        font = ImageFont.truetype("arial.ttf", 15)
        draw.text((1, 1), clase, font=font, fill="red")
        plt.figure()
        plt.imshow(reducida)
        print('El ticket', cant, 'pertenece a la categoria:',clase)
```

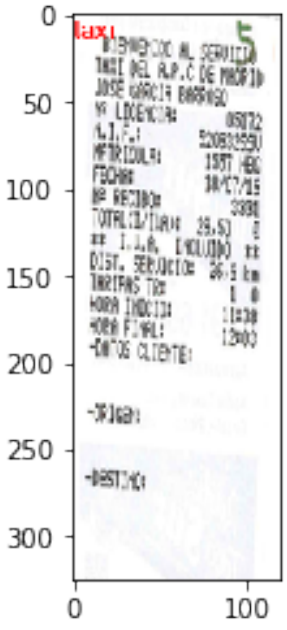
```
leyendo imagenes de  C:\Users\egarcia\Prueba_3\validar\
El ticket 1 pertenece a la categoria: gasolinera
El ticket 2 pertenece a la categoria: parking
El ticket 3 pertenece a la categoria: parking
El ticket 4 pertenece a la categoria: parking
El ticket 5 pertenece a la categoria: parking
El ticket 6 pertenece a la categoria: gasolinera
El ticket 7 pertenece a la categoria: comida
El ticket 8 pertenece a la categoria: taxi
El ticket 9 pertenece a la categoria: taxi
```

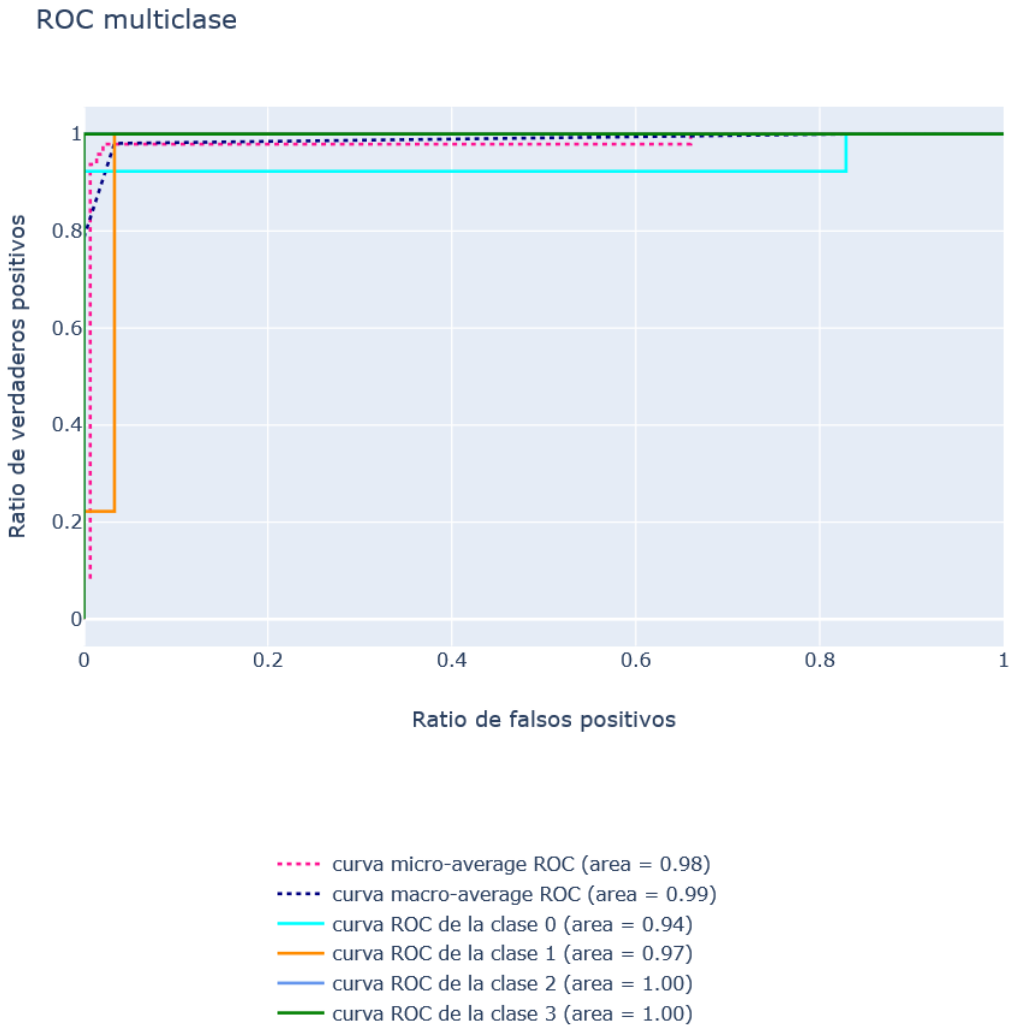



gasolinera
0
50
100
150
200
250
300
0
100


comida
0
50
100
150
200
250
300
0
100


0
50
100
150
200
250
300
0
100


0
50
100
150
200
250
300
0
100


ROC multiclase
Ratio de verdaderos positivos
Ratio de falsos positivos
0
0.2
0.4
0.6
0.8
1
curva micro-average ROC (area = 0.98)
curva macro-average ROC (area = 0.99)
curva ROC de la clase 0 (area = 0.94)
curva ROC de la clase 1 (area = 0.97)
curva ROC de la clase 2 (area = 1.00)
curva ROC de la clase 3 (area = 1.00)