“Nada hay en la mente que no haya estado antes en los sentidos.”
—Aristóteles

Redes Neuronales Convolucionales

UNA APLICACIÓN PRÁCTICA A OBRAS DE ARTE

JUAN MANUEL GARCÍA CRIADO
FERNANDO MIGUEL HIDALGO AGUILAR

Redes Neuronales Convolucionales – Una aplicación práctica a obras de arte

Juan Manuel García Criado
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
juagarcri@alum.us.es

Fernando Miguel Hidalgo Aguilar
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
ferhidagu@alum.us.es

***Resumen*—El presente documento presenta los resultados obtenidos de la creación de tres redes neuronales para clasificar obras de arte. Con estos dichos datos, se ha llegado a la conclusión de que la red neuronal 2 presenta los mejores resultados.**

***Palabras clave*—Inteligencia Artificial, Red neuronal convolucional, Hiperparámetro,**

I. Introducción

El proyecto, enmarcado en la asignatura de IA en el grado ISW, consiste en la preparación de datos de entrada (en concreto, imágenes de obras de arte divididas en 50 artistas), la creación de 3 redes neuronales y el entrenamiento de estas para ser capaces de identificar a los autores.

Se requiere que 2 de las redes compartan arquitectura, pero no técnicas de regularización, y que 2 tengan una arquitectura esencialmente distinta.

Para llevar a cabo la tarea, se ha optado por el uso del lenguaje Python [1], acompañado de las librerías Tensor Flow Keras [2] para la creación y entrenamiento de las redes, y la librería Split Folder para la preparación de los datos de entrada.

Originalmente se intentó usar Jupyter como entorno de desarrollo, pero una serie de problemas técnicos referentes a la instalación del programa y compatibilidad con ciertas librerías nos hizo desestimar la herramienta.

Para evitar estos problemas, se eligió Google Collaborative como entorno, debido a su sencillo manejo y contar con todas las librerías que necesitamos ya instaladas, así como una comunicación más fluidez entre los miembros, pues se permite la edición simultánea.

II. Preliminares

En lo referente a técnicas empleadas, dado el nombre de trabajo, han sido las redes convoluciones. Asimismo, una miríada de técnicas, algoritmos y funciones han sido usados a lo largo de todo el trabajo.

A. Métodos empleados

- Redes neuronales convolucionales [3]: Tipo de red neuronal artificial, variación de un perceptrón multicapa. Debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, justo el caso que nos ocupa.

  Las redes neuronales convolucionales consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general se añade una función para realizar un mapeo causal no-lineal.

  Como redes de clasificación, al principio se encuentra la fase de extracción de características, compuesta de neuronas convolucionales y de reducción de muestreo.

  Al final de la red se encuentran neuronas de perceptrón sencillas para realizar la clasificación final sobre las características extraídas. La fase de extracción de características se asemeja al proceso estimulante en las células de la corteza visual.

  Esta fase se compone de capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. Según progresan los datos a lo largo de esta fase, se disminuye su dimensionalidad, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, pero al mismo tiempo siendo estas activadas por características cada vez más complejas.

  Existen, a grandes rasgos, 3 tipos de neuronas:

  1) Neuronas convolucionales: En la fase de extracción de características, las neuronas sencillas de un perceptrón son reemplazadas por procesadores en matriz que realizan una operación sobre los datos de imagen 2D que pasan por ellas, en lugar de un único valor numérico.

     El operador de convolución tiene el efecto de filtrar la imagen de entrada con un núcleo previamente entrenado.

Esto transforma los datos de tal manera que ciertas características se vuelven más dominantes en la imagen de salida al tener estas un valor numérico más alto asignados a los píxeles que las representan.

2) *N*euronas de Reducción de Muestreo: Las redes neuronales cuentan con cierta tolerancia a pequeñas perturbaciones en los datos de entrada. Si dos imágenes prácticamente iguales se analizan con una red neuronal, el resultado debería de ser el mismo.

   Esto se obtiene, en parte, dado a la reducción de muestreo que ocurre dentro de una red neuronal convolucional. Al reducir la resolución, las mismas características corresponderán a un mayor campo de activación en la imagen de entrada.

3) *N*euronas de Clasificación: Después de una o más fases de extracción de características, los datos finalmente llegan a la fase de clasificación. Para entonces, los datos han sido depurados hasta una serie de características únicas para la imagen de entrada, y es ahora la labor de esta última fase el poder clasificar estas características hacia una etiqueta u otra, según los objetivos de entrenamiento.

   Las neuronas en esta fase funcionan de manera idéntica a las de un perceptrón multicapas.

   En la fase de extracción de características, las neuronas sencillas de un perceptrón son reemplazadas por procesadores en matriz que realizan una operación sobre los datos de imagen 2D que pasan por ellas, en lugar de un único valor numérico.

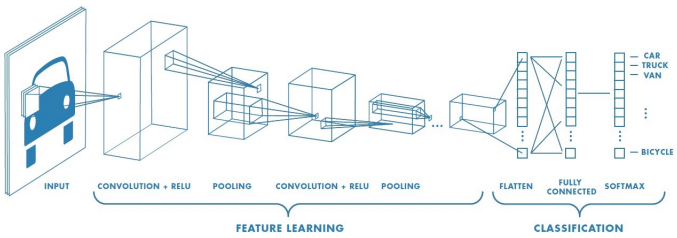


Fig. 1. Funcionamiento de una red neuronal convolucional

- Función de activación [4]: La función de activación se encarga de devolver una salida a partir de un valor de entrada, normalmente el conjunto de valores de salida en un rango determinado como (0,1) o (-1,1).

  Se buscan funciones que las derivadas sean simples, para minimizar con ello el coste computacional.

  Para las capas ocultas, ReLU ha sido la elegida, se caracteriza por transformar los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran.

  Para la capa de salida, se ha optado por Softmax, dado que está diseñada para admitir entradas no binarias

$$f(x) = \max(0, x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$$

Fig. 2. Función ReLU

(1,2,3...), como es nuestro caso, que contamos con 50 artistas de entrada.

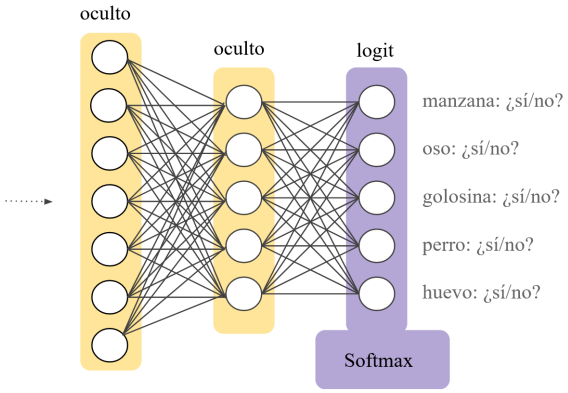
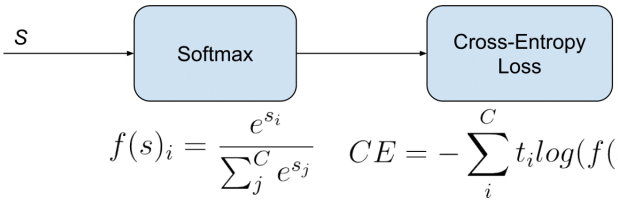


Fig. 3. Función Softmax

- Entropía Cruzada Categórica [5]: También conocida como Pérdida Softmax. Es la combinación entre la activación softmax y la entropía cruzada Usando este tipo de pérdida, entrenamos a la red para que saque una prohibida sobre las C clases para cada imagen. Muy usado en la clasificación de imágenes en varias clases, como es nuestro caso.



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

Fig. 4. Formula de la Entropía Cruzada Categórica

- Maxpooling [6]: Encuentra el valor máximo entre una ventana de muestra y pasa este valor como resumen de características sobre esa área. Como resultado, el tamaño de los datos se reduce por un factor igual al tamaño de la ventana de muestra sobre la cual se opera.
- Descenso de gradiente [7]: Algoritmo que estima numéricamente dónde una función genera sus valores más bajos. Eso significa que encuentra mínimos locales. Además, todo lo que necesita para ejecutarse es la salida numérica de una función, no requiere ninguna fórmula.

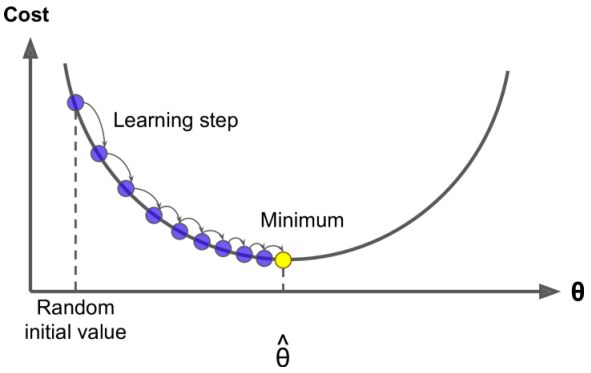


Fig. 5. Descenso por Gradiente

En el caso concreto de nuestro proyecto, Adam ha sido el subtipo de algoritmo elegido, dado que es el que mejor resultados presenta en la clasificación de imágenes.

- Flatten [8]: Convierte los elementos de la matriz de imágenes de entrada en un array plano.

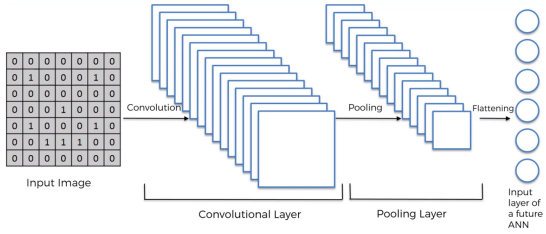


Fig. 6. MaxPooling y Flatten combinados

- Dense [9]: Capa totalmente conectada, por lo que todas las neuronas de una capa están conectadas a las de la siguiente capa. Se usa como capa de salida

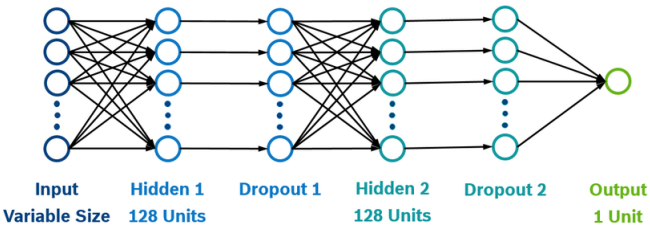


Fig. 7. Funcionamiento Dense

- Algoritmo Adam [10]: El algoritmo de estimación de movimiento adaptativo, o Adam para abreviar, es una extensión del algoritmo de optimización del descenso de gradientes. Adam está diseñado para acelerar el proceso de optimización.

  Esto se logra calculando un tamaño de paso para cada parámetro de entrada que se optimiza. Es importante destacar que cada tamaño de paso se adapta automáticamente al rendimiento del proceso de búsqueda en función de los gradientes (derivadas parciales) encontrados para cada variable.

Algorithm 1: Adam Optimizer

Initialize $\theta_0, m_0 \leftarrow 0\ , v_0 \leftarrow 0, t \leftarrow 0$
While θ_t not converged
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
Bias Correction
$\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}, \widehat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$
Update
$\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{v_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon} \right)$

Fig. 8. Algoritmo Adam

- Stride [11]: Parámetro filtro que modifica la cantidad de pixeles de una imagen que se desplazan, haciendo que el resultado se encoja. En conjunción con Padding, este parámetro ayuda a evitar el sobreajuste.

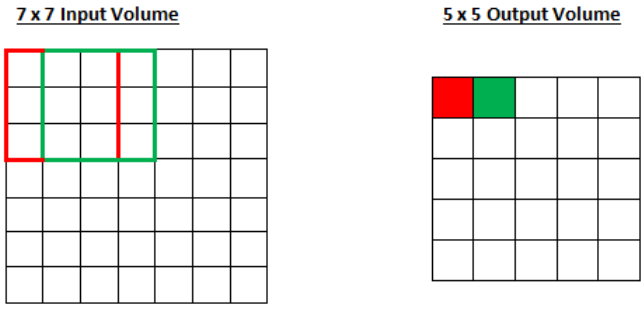


Fig. 9. Ejemplo de Stride

- Regularización L1L2 [12]: También conocida como Regularización ElasticNet, es una combinación de dos técnicas.
  1) *L*asso: Regularización que fomenta que la solución sea poco densa. Es decir, que algunos de los coeficientes acaben valiendo 0. Esto es útil para marcar que atributos de entrada son relevantes obteniendo un modelo que generaliza mejor.

$$J = \frac{1}{M} \sum_{i=1}^{M} (real_i - estimado_i)^2 + \alpha \frac{1}{N} \sum_{j=1}^{N} |w_i|$$

Fig. 10. Fórmula Lasso

  2) *R*idge: Regularización que hace que los coeficientes acaben siendo más pequeños. Esta disminución minimiza el efecto de la correlación entre los atributos de entrada y hace que el modelo generalice mejor.

$$J = \frac{1}{M}\sum_{i=1}^{M}(real_i \ - \ estimado_i)^2 + \alpha\frac{1}{2N}\sum_{j=1}^{N} w_i^2$$

Fig. 11. Fórmula Ridge

B. Trabajo Relacionado

Para documentarnos a la hora de abordar el proyecto, hemos optado principalmente por el trabajo realizado por Free Code Camp para YouTube [13].

Este videotutorial nos proveyó con una base en cuanto a las posibilidades de la librería Keras, así como proporcionarnos los primeros resultados tangibles, permitiendo una rápida iteración sobre las redes hasta alcanzar las mejores

Además, se ha consultado regularmente la documentación de Keras, para entender que había detrás de cada función de la librería.

Por último, la teoría del tema 3 de la propia asignatura ha sido útil para conocer la teoría que sustenta el proyecto [14].

III. Metodología

Para todos los experimentos, se ha seguido la misma metodología. En un primer lugar, se han cargado el conjunto de imágenes. Este conjunto se subdivide en 2: entrenamiento y validación.

Notamos que la cantidad de imágenes que conformaban el conjunto de entrenamiento no era el suficiente, cayendo la red en el sobreajuste, esto es, es el efecto de sobreentrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado.

Cuando un sistema se sobreajusta, el algoritmo de aprendizaje puede quedar ajustado a unas características muy específicas de los datos de entrenamiento que no tienen relación causal con la función objetivo.

Durante la fase de sobreajuste el éxito al responder las muestras de entrenamiento sigue incrementándose mientras que su actuación con muestras nuevas va empeorando.

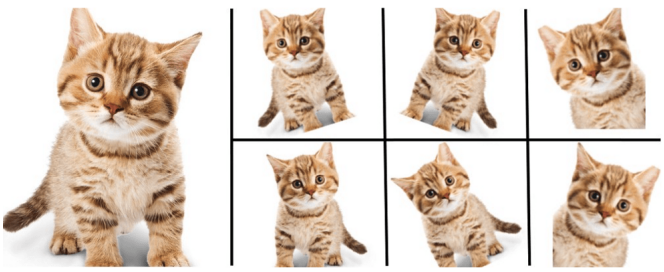


Fig. 12. Modificaciones a las imágenes

Para evitar el mencionado sobreajuste, decidimos realizar un preprocesado de las imágenes, en la que cada una de ellas era duplicada varias veces, y cada una de estas copias recibía una modificación, ya sea rotación, espejo vertical y horizontal, zoom, movimiento a la izquierda y movimiento a la derecha.

De esta manera, nos aseguramos tener imágenes en cantidad suficiente para un correcto entrenamiento.

Una vez solventado el problema, comenzamos a construir redes, iterando entre distintas arquitecturas hasta encontrar aquellas más óptimas para la presente tarea.

Se establecieron dichas arquitecturas, sin embargo, los resultados obtenidos eran muy pobres, siendo los mejores 0.15 de Accuray aproximadamente. Además, las iteraciones eran muy lentas y poco eficientes, legando incluso a colgar en entorno de desarrollo.

Tras realizar una investigación sobre las opciones que el entorno ofrece, se decidió cambiar el entorno de ejecución a uno basado en el uso de GPUs (graphics processing unit), comúnmente conocidas como tarjetas gráficas.

Estos dispositivos ofrecen una potencias de cálculo muy superior, llegando en nuestro caso, al nivel de acelerar el entrenamiento de una red de 2h a tan solo 10 minutos, una mejora considerable que nos permitió alcanzar los resultados que mostramos.

Además, tras y como se explica en la propia definición de Red Neuronal Convolucional, realizamos un redimensionado de las imágenes a 224x244px, para favorecer la lectura de patrones.

A continuación mostramos la arquitectura de cada una de las redes.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 244, 244, 32)      896
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 122, 122, 32)      0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 122, 122, 64)      18496
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 61, 61, 64)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 61, 61, 128)       73856
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 30, 30, 128)       0
_________________________________________________________________
flatten_4 (Flatten)          (None, 115200)            0
_________________________________________________________________
dense_4 (Dense)              (None, 50)                5760050
=================================================================
```

Fig. 13. Arquitectura Red 1

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 244, 244, 32)      896
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 122, 122, 32)      0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 122, 122, 64)      18496
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 61, 61, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 61, 61, 128)       73856
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 30, 30, 128)       0
_________________________________________________________________
flatten_3 (Flatten)          (None, 115200)            0
_________________________________________________________________
dense_3 (Dense)              (None, 50)                5760050
=================================================================
```

Fig. 14. Arquitectura Red 2

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 244, 244, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 122, 122, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 122, 122, 64)      18496
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 122, 122, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 61, 61, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 61, 61, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 115200)            0
_________________________________________________________________
dense (Dense)                (None, 50)                5760050
=================================================================
```

Fig. 15. Arquitectura Red 3

IV. Resultados

Se han realizado 3 experimentos, uno por cada red neuronal creada. En la siguiente tabla se recogen sus respectivas configuraciones y mediciones.

El objetivo en todos los experimentos es obtener los mayores valores posibles en las mediciones, pues eso se traduce en un mejor desempeño de la red en la tarea de clasificar las imágenes.

La Red 3 se observa fácilmente y sin lugar a duda que presenta peores resultados que las otras. Sin embargo, las redes Red 1 y Red 2 presentan valores muy similares. Hemos optado por la red 2, por contar con menor Accuracy, es decir, está menos entrenada.

Esto se traduce en que con un menor entreno, obtiene prácticamente los mismos resultados, lo que la hace una red mucho más prometedora en caso de entrenarse más.

A continuación se muestran una tabla de los valores obtenidos de las redes, así como gráficas que representan visualmente los valores de Accuracy respecto a la epoch de cada red:

TABLA I
Tabla de resultado y análisis

Experimento	Accuracy	Val Accuracy
Red 1	0.62	0.38
Red 2	0.59	0.37
Red 3	0.49	0.34

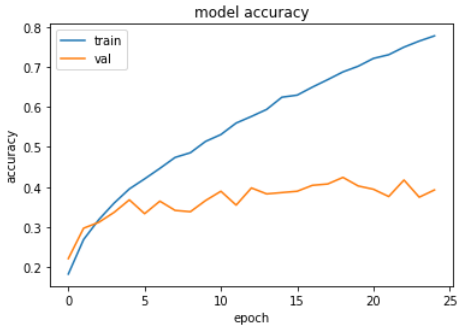


Fig. 16. Gráfica Accuracy/Epoch Red 1

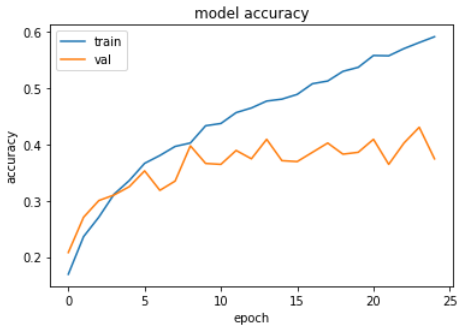


Fig. 17. Gráfica Accuracy/Epoch Red 2

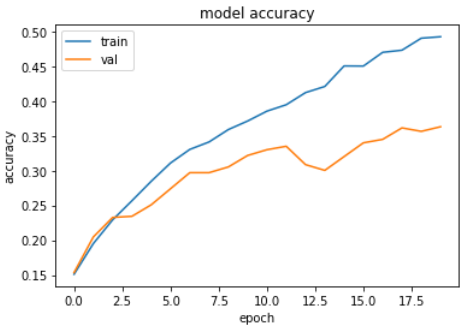


Fig. 18. Gráfica Accuracy/Epoch Red 3

V. Conclusiones

Recapitulando, se han realizado 3 experimentos de clasificación de imágenes, empleando en cada uno de ellos una red neuronal convolucional distinta, siendo la mejor la Red 2.

El trabajo ha presentado unos resultados aceptables, pero se podrían llegar a mejorar implementando nuevas técnicas, como el Fine-Tuning, tal y como se menciona en capítulos posteriores a los seguidos en la guía de Free Code Camp.

Asímismo, un mayor número de imágenes de entrenamiento facilitaría el desarrollo de las redes, obteniendo además de ellas unos resultados mucho más satisfactorios.

Referencias

[1] General Python FAQ
[2] Tensor Flow Keras Official Docs
[3] Redes Neuronales Convolucionales, Wikipedia
[4] Función de activación, Diego Calvo
[5] Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names, Raúl Gómez
[6] MaxPooling2D layer, Keras API Docs
[7] Descenso de gradiente, Khan Academy
[8] Flatten layer, Keras API Docs
[9] Dense layer, Keras API Docs
[10] Optimización de descenso de gradiente de Code Adam Top Big Data
[11] Stride (Machine Learning), DeepAI
[12] Regularización Lasso L1, Ridge L2 y ElasticNet, IArttificial
[13] Keras with TensorFlow Course - Python Deep Learning and Neural Networks for Beginners Tutorial
[14] Tema 3 - Redes Neuronales, CS, Universidad de Sevilla