



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Trabajo de fin de Grado

Grado en Ingeniería Informática del Software

Adversarial examples

**Realizado por
Iván de los Santos García**

**Dirigido por
Rocío González Díaz
Eduardo Paluzo Hidalgo**

**Departamento
Matemática aplicada I**

Sevilla, 11 de Febrero de 2019

Resumen

A medida que el uso de tecnologías que se benefician de los avances en el campo de las redes neuronales aumentan, se vuelve de vital importancia el diseño de modelos de redes neuronales robustos que sean resistentes a posibles ataques que puedan resultar críticos para la ciberseguridad. El trabajo que aquí presentamos se basa en el artículo (Su, Vargas, y Sakurai, 2017), en el que se analizan cómo perturbaciones en la imagen de entrada de una red neuronal pueden modificar el resultado original de clasificación. La perturbación a realizar conlleva una restricción en el número de píxeles a modificar y, en particular un único píxel puede modificarse. El trabajo se estructura como sigue: Inicialmente, se ha investigado la construcción de las redes neuronales que usaremos como escenario de ataque, redes neuronales de convolución y de perceptrones. Para generar la perturbación de un píxel utilizaremos un algoritmo genético conocido como evolución diferencial. Los resultados muestran que entorno al 10.20 % de las imágenes del conjunto, pueden ser vulnerables, y den lugar a nuevos “adversarial examples”. Con nuestro estudio, se ha creado una red neuronal de convolución que clasifique con un alto grado de confianza imágenes de señales de tráfico. Posteriormente, se ha usado esta red para realizar el ataque de un píxel propuesto por (Su y cols., 2017) y basado, en parte, en la implementación del investigador (Kondratyuk, 2019). Nuestro proyecto ha resultado exitoso, permitiendo probar en un conjunto de datos hasta ahora no estudiado por ataques adversarios, que es posible generar en él de forma rápida nuevos “adversarial examples”.

Agradecimientos

A mi tutora Rocío González por su ayuda, esfuerzo y ánimos para poder llevar a cabo el desarrollo del proyecto y a mi co-tutor Eduardo Paluzo por el esfuerzo realizado y su orden en el trabajo. Ambos me han ayudado a intentar ser mas disciplinado y alcanzar mis objetivos. Gracias por dejarme participar en un trabajo tan interesante.

A mi madre y mis hermanas por siempre estar conmigo cuando las necesito, sin ellas la labor de casa no habría sido posible.

Mis mas sinceras gracias, Iván.

Índice general

Índice general	III
Índice de cuadros	V
Índice de figuras	VI
1 Introducción	1
2 Redes neuronales	3
2.1 perceptrón multicapa	4
2.2 Funciones de activación no lineales	6
2.2.1 Neuronas en la capa oculta	6
2.2.2 Neuronas de la capa de salida	8
2.3 Aprendizaje basado en gradiente	10
2.3.1 Retropropagación	11
2.4 Regularización	14
2.4.1 Regularización L2 (arista) y L1 (Lasso)	15
2.4.2 Dropout	15
2.5 Convolución	16
2.5.1 Pooling o agrupación	19
3 Adversarial examples	21
3.1 Concepto	21
3.2 Importancia	21
3.3 Ataque de un pixel	23
3.3.1 Descripción	23
3.3.2 Evolución diferencial	24
4 Desarrollo del proyecto	27
4.1 Lenguaje y entorno de desarrollo	27
4.2 Conjunto de datos	28
4.3 Diseño del modelo neuronal	30
4.4 Implementación y resultados de la red neuronal	30
4.4.1 Implementación y resultados del ataque a un píxel	32
4.4.2 Experimentación	36
4.4.3 Futuro	38
5 Planificación temporal	41
6 Conclusiones	45

Índice de cuadros

4.1	Red diseñada para el proyecto	33
4.2	Resultados de la experimentación, identificando de la muestra de 500 imágenes cuantas dieron como resultado un adversarial example.	37

Índice de figuras

1.1	A la izquierda, señal detectada correctamente, a la derecha, señal con una modificación que cambia la clasificación original de la red neuronal.	2
1.2	Empresaria china erróneamente denunciada.	2
2.1	Modelo neuronal de perceptrones.	4
2.2	Representación matricial de la matriz de pesos de la capa n -ésima.	5
2.3	Representación gráfica de la función sigmoide.	7
2.4	Representación gráfica de la función RELU.	7
2.5	Operación de maxout para un ejemplo de dos entradas. En este caso se ha omitido el valor de b	8
2.6	Cálculo de la función Softmax para un vector de entrada de tres valores. Como podemos comprobar la suma de las salidas de $S(y_i)$ suman 1.	9
2.7	Valores del gradiente dependiendo de la pendiente de la función.	11
2.8	Proceso de dropout en el que se eliminan algunas conexiones existentes.	16
2.9	Ejemplo del proceso de convolución para un volumen de entrada de profundidad tres, y con la aplicación de dos filtros, produciendo así dos mapas de características que dan lugar al volumen de salida con profundidad dos, como esperábamos, igual al número de filtros utilizados.	17
2.10	Convolución con uso de paso o stride 1.	17
2.11	Convolución con uso de paso o stride 2. Como observamos además, el mapa de características resultante es espacialmente más pequeño que usar un paso menor.	18
2.12	Proceso de zero padding.	18
2.13	Proceso de pooling o agrupación para la reducción del volumen mientras mantiene las características más relevantes del volumen de entrada.	19
3.1	Una perturbación para generar “adversarial examples” en la imagen de entrada, puede provocar una clasificación incorrecta por la red neuronal.	22
3.2	En la imagen de la izquierda una señal de tráfico con un grafiti. En la derecha una imagen con una modificación adversaria de forma que la señal de Stop sea clasificada de forma errónea.	23
3.3	En la figura podemos apreciar como un ataque adversario a una imagen sin limitación alguna, puede dar lugar a imágenes que sean fácilmente reconocibles como modificadas.	23
3.4	Se observa como la modificación de un único píxel puede cambiar el resultado de la red neuronal a una categoría distinta de la original.	25
4.1	Imágenes pertenecientes al conjunto de datos a usar.	28
4.2	Señales del conjunto de imágenes	29
4.3	Señal cargada satisfactoriamente en el cuaderno Jupyter de trabajo	29
4.4	Modelo neuronal desarrollado para el proyecto.	31

4.5	Descripción del procesamiento seguido por la red neuronal diseñada para nuestro proyecto.	32
4.6	En la imagen superior podemos observar la imagen sin modificar, en la inferior, la imagen con el vector de perturbación X de la ecuación 4.2, aplicado.	34
4.7	Resultado de aplicar el vector X , de la ecuación 4.3, a la imagen, modificando así la predicción original de la red, siendo esta límite de 80 Km/h con un 94.22 % de confianza. En la imagen inferior podemos observar como la red tras la perturbación añadida, ha modificado la clasificación original de límite de 80 Km/h por límite de 20 Km/h. Tras la perturbación la confianza de la red en ser una señal de límite 80 Km/h ha disminuido hasta el 23.31 %.	35
4.8	Resultado de realizar un ataque adversario sin éxito. El algoritmo de evolución diferencial al intentar ir generando mejores vectores de perturbación que cambien la categoría original de la imagen, no han conseguido encontrar ninguno que den lugar a “adversarial example”.	36
4.9	Resultado de realizar un ataque adversario a una imagen con valor $d = 3$, obteniendo un resultado satisfactorio. El algoritmo de evolución diferencial fue capaz de generar un vector de perturbación que diese lugar a una gran disminución en la probabilidad de pertenencia de la imagen a su categoría original, dando lugar a un “adversarial example”. ”Attack result” se corresponde con el vector de perturbación X	37
4.10	Se ilustran distintos “adversarial examples” encontrados por nuestro ataque. En todas las imágenes se ha modificado únicamente un píxel.	38

CAPÍTULO 1

Introducción

En este capítulo realizaremos un breve recorrido en el campo de las redes neuronales para poder entender los cambios que han producido un avance tan vertiginoso en este campo y aspectos en el ámbito de la seguridad que servirán como motivación para la investigación propuesta en este proyecto.

En los siguientes capítulos estudiaremos los conceptos troncales que nos permitirán estudiar la creación de redes neuronales de perceptrón y redes neuronales de convolución. Posteriormente estudiaremos en mayor profundidad qué son y en qué consisten los "adversarial examples", sección 3, para finalizar con la investigación del ataque a un píxel propuesto por los investigadores (Su y cols., 2017).

La motivación principal para el uso de las redes neuronales es la de poder obtener patrones e información relevante en un conjunto de datos gracias a un proceso de aprendizaje supervisado. Los factores que han propiciado el creciente uso de este tipo de arquitectura han sido:

1. *Enorme volumen de datos.* Gran parte de universidades y empresas han utilizado sus infraestructuras durante las últimas décadas para almacenar y clasificar grandes volúmenes de información.
2. *Hardware especializado.* Avances en el campo de los microchips han permitido el uso de GPUs (tarjetas gráficas) para el procesamiento paralelizado de la información. Esto ha ayudado a reducir en varios órdenes de magnitud el tiempo de procesamiento de ciertas tareas que involucran operaciones matemáticas.
3. *Software especializado.* Grandes empresas, investigadores y particulares de todo el mundo han puesto su foco en el desarrollo de software que aproveche los avances en el campo de las tarjetas gráficas y la computación paralela mediante el uso de librerías y lenguajes específicos (Tensorflow, Nvidia CUDNN,...), y avances en el campo académico dentro del mundo de la inteligencia artificial con la publicación de artículos de especial repercusión en los últimos años.

Es por ello, que ante la creciente utilización de dispositivos y aplicaciones que hacen uso de modelos basados en redes neuronales artificiales, y todos los factores anteriormente expuestos, den como resultado la necesidad de desarrollar sistemas de seguridad para la prevención de posibles usos maliciosos por parte de atacantes. En este ámbito, los ataques conocidos a redes neuronales se encuentran englobados en ataques que intentan que la red clasifique de manera incorrecta ejemplos o muestras cuya clasificación es conocida y correcta. Posibles aplicaciones maliciosas que hacen vislumbrar por qué son necesarios esfuerzos en este campo pueden ser:

- *Visión por computador para conducción autónoma.* Ataques enfocados a una clasificación incorrecta de las señales de tráfico o semáforos por parte de un sistema de visión por computador que hace uso de redes neuronales de convolución para la segmentación de las imágenes de su alrededor.



Figura 1.1: A la izquierda, señal detectada correctamente, a la derecha, señal con una modificación que cambia la clasificación original de la red neuronal.

- *Diagnóstico clínico.* Ataques que puedan afectar al diagnóstico clínico de un paciente provocando que el modelo clasifique una enfermedad erróneamente o que recomiende medicinas no necesarias o incluso fatales para algunos pacientes específicos.
- *Visión por computador para sistemas de reconocimiento facial.* En la ciudad china de Shenzhen se implantó en el año 2018 un sistema de reconocimiento facial unido a un registro de sus ciudadanos para la denuncia en pantallas públicas de aquellos peatones imprudentes. Esto ha suscitado numerosas controversias, como por ejemplo la clasificación incorrecta y posterior denuncia pública de una empresaria china cuya cara se encontraba en forma publicitaria en un autobús.



Figura 1.2: Empresaria china erróneamente denunciada.

CAPÍTULO 2

Redes neuronales

Las redes neuronales son sistemas de procesamiento de información vagamente inspirado en el funcionamiento de las redes neuronales biológicas. De hecho, las redes neuronales tienen un gran potencial al ser “aproximadores universales” (Cybenko, 1989).

Esto es, para cualquier función $f(x)$ que queramos aproximar con una precisión mayor que $\epsilon \in \mathbb{R}^+ \setminus \{0\}$, podemos garantizar que con una cantidad suficiente de neuronas en la capa oculta podemos encontrar una función $g(x)$ tal que para todo $x \in \mathbb{R}^+$ se cumple que $|f(x) - g(x)| < \epsilon$. Es decir, con una precisión adecuada de ϵ , podemos aproximar la función $f(x)$ mediante $g(x)$ (Nielsen, 2015).

Surgen dos casos de uso principales con respecto a la aproximación de funciones. Por un lado, aproximar una función objetivo f conocida y , por otro lado, aproximar una función de la que se conoce únicamente una muestra, $(x_n, f(x_n))_{n=1}^N$. Este tipo de aprendizaje es conocido como aprendizaje supervisado, siendo el conjunto de entrenamiento $\mathcal{D} := (x_n, y_n)_{n=1}^N$ con $y_n := f(x_n)$, f es desconocida pero será posteriormente estimada mediante el aproximador \hat{f} .

Uno de los objetivos fundamentales del entrenamiento es que el aproximador \hat{f} generalice correctamente, es decir, que replique el comportamiento de f en ejemplos no vistos en \mathcal{D} y con los que, por lo tanto, \hat{f} no ha sido entrenado. Es por eso, que por lo general, siempre se buscan redes, o modelos (su uso es indiferente del de “redes”), cuyo rendimiento a la generalización sea óptimo.

Con esta finalidad, el proceso de encontrar un buen aproximador \hat{f} se basa en un entrenamiento de la red neuronal con respecto a un *conjunto de entrenamiento* \mathcal{D} ; así como una estimación del rendimiento de la red a generalizar con respecto a un *conjunto de pruebas* con el que nunca ha sido entrenada (Balestriero y Baraniuk, 2017).

Para poder entender mejor su funcionamiento debemos comprender sus partes fundamentales. Realizaremos, a continuación, una exposición detallada de la composición fundamental del perceptrón multicapas, prestando especial atención a dos conceptos:

- *La arquitectura de la red neuronal.* Se trata de la propia estructura de la red, teniendo en cuenta neuronas de entrada y salida, número de capas y función de activación en las capas ocultas y de salida.
- *El aprendizaje de la red neuronal.* Se estudia el proceso de aprendizaje de la red, tratando aspectos clave como son funciones de coste, retropropagación y regularización.

2.1– perceptrón multicapa

El perceptrón multicapa se compone de varias capas de neuronas. Se distinguen tres tipos: de entrada, de salida y ocultas. Una capa de entrada, esta compuesta por neuronas que reciben datos o señales procedentes del entorno. Una capa de salida se compone de neuronas que proporcionan la respuesta de la red neuronal. Además, el tipo y cantidad de neuronas de la capa de salida constituirá un elemento de vital importancia para el entrenamiento y evaluación de la red. Las capas ocultas, establecen conexiones entre las neuronas de la capa previa y la capa posterior como podemos apreciar en la figura 2.1.

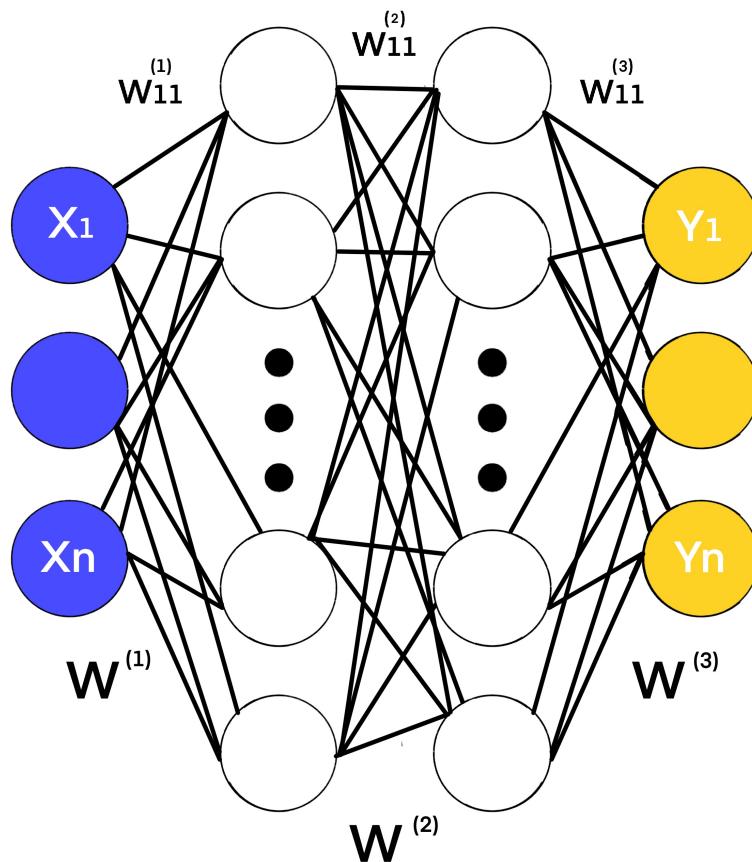


Figura 2.1: Modelo neuronal de perceptrones.

En nuestro caso, la información de las conexiones entre neuronas siempre fluye hacia delante. Sin embargo, existen otros modelos, como las redes neuronales recurrentes, donde se pueden establecer conexiones con neuronas de capas anteriores.

Para definir la arquitectura de la red deberemos comprender sus componentes:

1. *Vector de entrada.* Representa la información que ha de ser procesada por la red neuronal. Se trata, en el caso de las redes neuronales de propagación hacia delante, de la primera capa de la red. Este vector de entrada en general toma valores reales, es decir, puntos en \mathbb{R}^n :

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = (x_1, x_2, \dots, x_n). \quad (2.1)$$

2. *Matriz de pesos.* Se trata de los parámetros ajustables fundamentales de la red neuronal. Modelan el grado de activación de las neuronas ocultas. Sus valores serán de tipo real, $w_{ij} \in \mathbb{R}$. En la figura 2.2, podemos observar representación matricial de la matriz de pesos.

$$\mathbf{W}^{(n)} = \begin{bmatrix} w_{11}^{(n)} & \dots & w_{1j}^{(n)} \\ \vdots & \ddots & \vdots \\ w_{i1}^{(n)} & \dots & w_{ij}^{(n)} \end{bmatrix}$$

Figura 2.2: Representación matricial de la matriz de pesos de la capa n -ésima.

El entrenamiento consistirá, como veremos en la sección 2.3, en un proceso de optimización en el que se intenta minimizar una función de error.

Es usual que los elementos de la capa n -ésima se encuentre totalmente conectados con los elementos de la capa $(n - 1)$ -ésima (o capa anterior). Además, cada conexión tiene un peso que será susceptible a modificaciones durante su entrenamiento.

3. *Función de agregación.* Constituye el proceso de combinar linealmente las entradas de la capa $(n - 1)$ -ésima con la matriz de pesos $\mathbf{W}^{(n)}$ y el uso de una función de activación no lineal. Es habitual encontrar en la bibliografía unos parámetros adicionales denominados umbral o "bias" en inglés, cuyo valor comúnmente es 1 o -1. El parámetro umbral es comúnmente introducido en la matriz de pesos añadiendo la coordenada adicional $x_0 = 1$, al vector de entrada.

Definiremos la activación de un elemento $a_j^{(n)}$ como:

$$a_j^{(n)} = \sum_{i=1}^M w_{ij}^{(n)} z_i^{(n-1)} + b_i^{(n)} \quad (2.2)$$

donde M es el número de elementos en la capa $(n - 1)$ -ésima. El superíndice (n) indica que la activación que estamos tratando es la neurona a_i de la capa n -ésima. El elemento $z_i^{(n-1)}$ se corresponde con el elemento i de la capa $(n - 1)$ -ésima después de que se le aplique la función de activación.

El último paso para completar el proceso de activación es el de la aplicación de una función no lineal. El concepto de utilizar funciones de aplicación no lineales así como sus distintos tipos será explorado en mayor profundidad en la sección 2.2.

La salida final de la neurona oculta dada la activación $a_i^{(n)}$ se define de la siguiente manera:

$$z_j^{(n)} = \sigma(a_j^{(n)}) = \sigma\left(\sum_{i=1}^M w_{ij}^{(n)} a_i^{(n-1)} + b_i^{(n)}\right) \quad (2.3)$$

donde $\sigma(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$ es la función de activación a usar. En la próxima sección, 2.2, se verá más en profundidad el concepto de función de activación con multitud de ejemplos.

Como podemos observar, la red neuronal sigue un proceso de composición de funciones para producir la salida de la red, en base a la figura 2.1. Así, su expresión final es la siguiente:

$$y_j = \sigma\left(\sum_{i=1}^M w_{ij}\sigma\left(\sum_{j=1}^D w_{ij}a_j + b_i\right) + b_i\right). \quad (2.4)$$

2.2– Funciones de activación no lineales

La principal motivación para el uso de funciones de activación no lineales es su capacidad para separar conjuntos de datos a priori no separables linealmente. Esto se trata de algo fundamental en las redes neuronales debido a que una de sus funciones fundamentales es la de clasificar conjuntos de datos. Los datos de entrada de una red neuronal pueden ser de distinta dimensionalidad, por ejemplo, un vector de n -valores reales tiene dimensionalidad \mathbb{R}^n , una imagen en blanco y negro tendrá dimensionalidad $\mathbb{R}^{n \times m}$ (alto y ancho), mientras que una imagen a color $\mathbb{R}^{n \times m \times d}$ (Alto, ancho y profundidad). Es por ello, que una de las funciones principales de las redes neuronales, es la de tratar de resolver problemas de clasificación en espacios de datos de alta dimensionalidad.

Sin el uso de funciones de activación no lineales, la red actuaría como una combinación lineal de sus entradas, es decir como un polinomio lineal. Existen múltiples técnicas basadas en polinomios lineales como regresión lineal. El problema de la alta dimensionalidad de los datos hace que sea imposible encontrar un polinomio lineal que pueda separar los datos y por tanto encontrar una buena clasificación de los datos, es por eso que entran en uso las funciones de activación con características no lineales.

En los últimos años se ha generado un gran debate acerca del tipo de funciones de activación a usar. Pasaremos a describir algunas de las funciones de activación que han cobrado gran popularidad debido a su avance en el estado del arte de distintas arquitecturas neuronales.

2.2.1. Neuronas en la capa oculta

Sigmoide

La función sigmoide es una función real creciente de variable real cuya imagen está contenida en el intervalo $(0, 1)$. Más concretamente,

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad (2.5)$$

donde, $\sigma : \mathbb{R} \rightarrow (0, 1)$ y $x \in \mathbb{R}$. Tiende a 1 en $+\infty$ y a 0 en $-\infty$. Podemos observar su representación gráfica en la figura 2.3.

Además, la expresión de su derivada viene definida como:

$$\sigma(x)' = \sigma(x)(1 - \sigma(x)), \quad (2.6)$$

Se trata ejemplo clásico de función de activación aunque se encuentra actualmente en desuso debido al siguiente factor:

- Las funciones sigmoides saturan el gradiente, esto es, cuando los valores de activación se encuentra en los extremos de la función 0 o 1, el valor del gradiente en estas regiones se vuelve cercano a cero. Como veremos posteriormente en la sección 2.3.1, mediante el uso de retropropagación podremos mejorar el resultado de clasificación de la red en sucesivas iteraciones modificando los pesos de la red. Este método, hace uso de las activaciones de la red neuronal, si su valor es cercano a cero no se producirán modificaciones en los pesos provocando la incapacidad de la red para aprender.

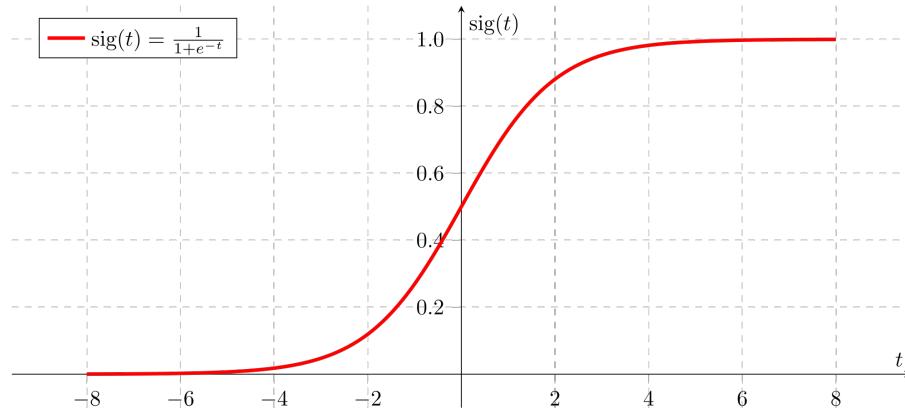


Figura 2.3: Representación gráfica de la función sigmoide.

ReLU o rectificador

Se trata de una función cuyo valor de activación actúa como umbral en el valor cero. Está definida como sigue:

$$f(x) = \max(0, x), \quad (2.7)$$

donde $f : \mathbb{R} \rightarrow \mathbb{R}^+$ y $x \in \mathbb{R}$. Su representación viene dada en la Figura 2.4.

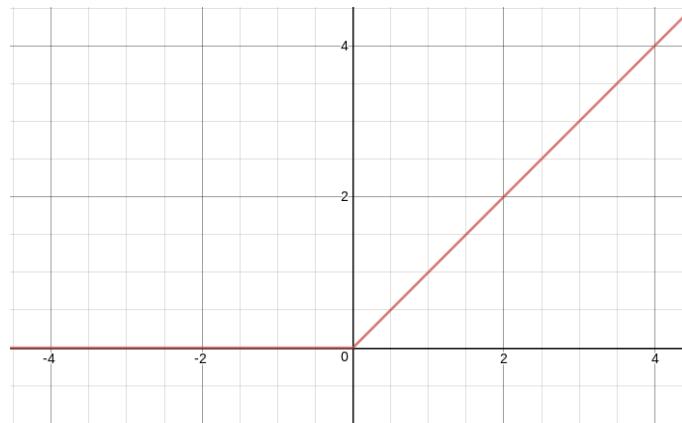


Figura 2.4: Representación gráfica de la función RELU.

Es comúnmente utilizada para redes de convolución, otorgando un mejor resultado que otras funciones de activación como podría ser la sigmoide o la tangente hiperbólica. Algunos los factores que la han llevado a ser tan utilizada son:

- **Acelera la convergencia:** comparado con otras funciones como la sigmoide, utilizando la técnica de optimización de descenso por el gradiente, método de aprendizaje visto en la sección 2.3.
- **Ofrece una implementación poco costosa :** Debido a que no involucra la utilización de operaciones como exponenciación, división, etc., ya que el valor que toma es cero o la parte positiva del dato de entrada x .

- **Inconveniente :** la utilización de funciones ReLU son débiles ante el entrenamiento, esto es, si la función devuelve valores en el dominio negativo su valor será cero y estas neuronas no podrán ser entrenadas correctamente considerándolas como “muertas”. Esto causa, en algunas situaciones, que gran parte de las neuronas de la red no intervengan. Sin embargo, con una elección correcta de los hiper-parámetros de la red esto no debería ser un problema frecuente (Karpathy, 2019a).

Maxout

Esta función de activación fue propuesta en 2013 (Goodfellow et al., 2013) e intenta generalizar el concepto de las funciones ReLU. En lugar de aplicar la función ReLU, $R(x)$, a cada una de las neuronas de una determinada capa, la función maxout divide el valor de x en k grupos, cada uno de estos grupos devolverá el valor máximo de la combinación lineal del valor x con k matrices de peso, es decir, calculamos

$$\max(W^{(n)}x + b_i), \quad (2.8)$$

para $W^{(n)}x \in \mathbb{R}^{(dxmk)}$, $b \in \mathbb{R}^{(mxk)}$ y $x \in \mathbb{R}^d$.

Este concepto, a pesar de parecer abstracto, en la práctica es sencillo. Se encuentra mejor ejemplificado en la Figura 2.5.

La utilización de funciones maxout es comúnmente usada como técnica de regularización obteniendo excelentes resultados. El concepto de regularización constituye una de las principales motivaciones a la hora de monitorizar el entrenamiento de la red neuronal y para la elección de sus hiper-parámetros, es que, a posteriori, la red se generalice de manera adecuada ante un conjunto de datos de prueba no conocidos durante el entrenamiento.

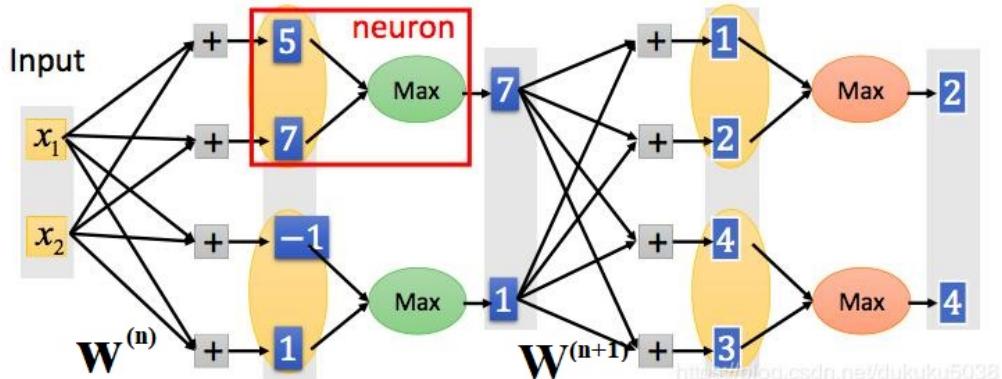


Figura 2.5: Operación de maxout para un ejemplo de dos entradas. En este caso se ha omitido el valor de b .

2.2.2. Neuronas de la capa de salida

La elección del tipo de neuronas de la capa de salida depende, en gran medida, del tipo de problema que necesitamos resolver. Principalmente nos centraremos, como hemos mencionado anteriormente, en la tarea de clasificación o categorización de la red neuronal.

Es común por la facilidad a la hora de decidir si la red esta clasificando de manera satisfactoria nuestras entradas, que la salida se comporte como una función de distribución.

Salida sigmoide para clasificación binaria

Una neurona sigmoide en la capa de salida es usada para predecir el valor de una variable aleatoria binaria, es decir una distribución de Bernoulli:

$$\begin{aligned} P(X = 1) &= p, \\ P(X = 0) &= 1 - p. \end{aligned} \tag{2.9}$$

La salida proporcionada por la función nos indicará la pertenencia del valor de entrada a una categoría binaria.

Función Softmax para multclasicación

En numerosas ocasiones el mínimo de categorías en el que clasificamos las entradas de la red neuronal es mayor de dos, siendo de gran ayuda el uso de funciones Softmax que nos permiten modelar una distribución de probabilidad sobre un conjunto discreto de categorías (distribución multinomial). Está definida como:

$$S(a_i) = \frac{a_i}{\sum_{j=1}^N a_j}, \tag{2.10}$$

Este tipo de funciones no poseen una representación gráfica como la sigmoide. Todos los valores de salida, y_i , de la función Softmax se encuentran en el intervalo $[0, 1]$. Cada salida y_i , representa la probabilidad de pertenencia, dada una entrada cualquiera, a la categoría i . Esta función cumple además con las propiedades de una función de distribución, cada probabilidad de salida $y_j \geq 0$ y la suma de todas ellas debe ser igual a 1.

$$y_i \geq 0, \tag{2.11}$$

$$\sum_{i=1}^N y_i = 1, \tag{2.12}$$

donde N es el número de neuronas en la capa de salida. En la Figura 2.6 se puede ver uso de la función Softmax ante un vector.

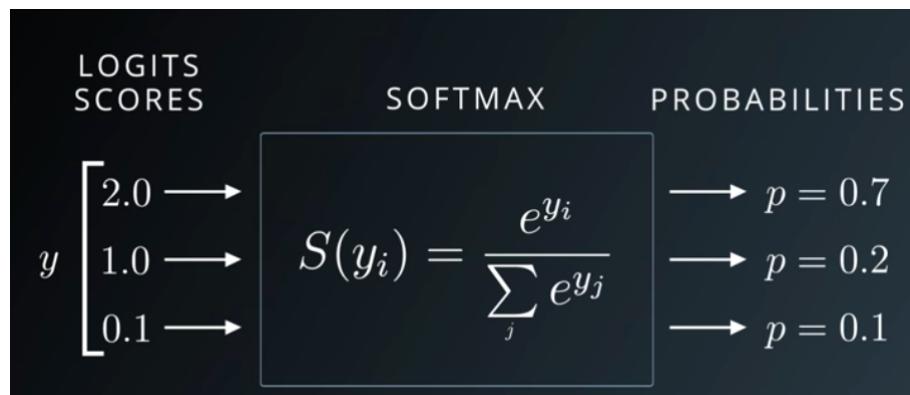


Figura 2.6: Cálculo de la función Softmax para un vector de entrada de tres valores. Como podemos comprobar la suma de las salidas de $S(y_i)$ suman 1.

Esta sección ha tratado de explorar las distintas opciones disponibles para la decisión acerca de que función de activación utilizar. Sin embargo, existen un gran variedad de funciones de activación distintas que dependiendo del tipo de red neuronal a construir nos podrán ofrecer un mejor rendimiento.

2.3– Aprendizaje basado en gradiente

Las redes neuronales son generalmente entrenadas por un proceso iterativo basado en el descenso por el gradiente. Este proceso trata de disminuir el valor de una función de coste, E . Los distintos métodos de descenso por gradiente no garantizan el encontrar un mínimo global de la función de coste debido a su naturaleza no convexa por el uso de funciones de activación no lineales en la arquitectura de la red.

Para comprender mejor este concepto debemos recordar la noción de derivada. La derivada es un concepto fundamental del cálculo y en particular mide la pendiente de una función en un punto concreto, mas rigurosamente, la derivada se establece como la medida de variación entre el resultado de la función con respecto a una ligera variación en sus entradas.

$$f'(x) = \lim_{m \rightarrow 0} \frac{f(x + m) - f(x)}{m}, \quad (2.13)$$

Para funciones dependientes de más de una variable se extiende el concepto de derivadas y se utilizan las derivadas parciales, estas miden el cambio con respecto a una única variable de la función.

Para una función $f(x_1, \dots, x_n)$ la derivada parcial con respecto a una de sus variables está denotada como:

$$\frac{\partial f}{\partial x} \text{ o también, } \partial x_i f, \quad (2.14)$$

Si bien no queremos aplicar la derivada con respecto a una de sus variables sino a todas, el gradiente generaliza la noción de derivadas y derivadas parciales para aplicarlo a un vector. Esto quiere decir que el gradiente de $f(x)$ es un vector que contiene las derivadas parciales con respecto a todas las variables del vector de entradas, se define como:

$$\nabla f(x_1, \dots, x_n) = \frac{\partial f}{\partial x_1} e_1 + \dots + \frac{\partial f}{\partial x_n} e_n, \quad (2.15)$$

donde e_i pertenece a una base en un espacio vectorial de dimensión n .

El gradiente es la pendiente de una recta tangente al gráfico de la función, más en particular indica la **dirección de máximo aumento de la función**. En la Figura 2.7 podemos observar un ejemplo del valor del gradiente.

Sabiendo que el gradiente representa la dirección de máximo aumento de la función, podemos realizar un procedimiento para llevar a cabo una actualización sistemática de los pesos en la dirección contraria al gradiente para así disminuir el valor de la función de coste asociada, E hasta encontrar un mínimo.

$$w_{ij}^{(n)\prime} = w_{ij}^{(n)} - \eta \left(\frac{\partial E}{\partial w_{ij}^{(n)}} \right), \quad (2.16)$$

Donde, $w_{ij}^{(n)\prime}$ representa el nuevo valor del peso $w_{ij}^{(n)}$ tal que disminuye el valor de E , es decir, **contrario al gradiente**. η representa un pequeño valor denominado ratio de aprendizaje, por ejemplo 0.01. Este valor será usado para regular la actualización de los antiguos pesos de forma que nos aproximemos a un mínimo por medio de sucesivas actualizaciones. Un valor demasiado alto en el ratio de aprendizaje podría hacer que "nos saltemos" un mínimo y uno demasiado pequeño ralentizaría el aprendizaje.

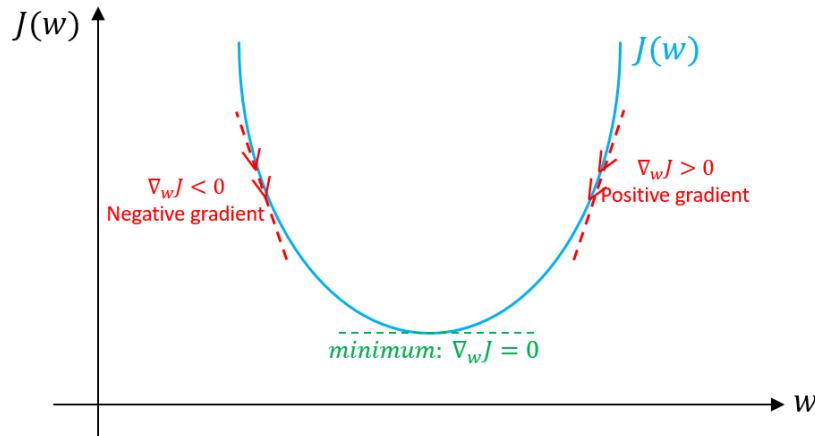


Figura 2.7: Valores del gradiente dependiendo de la pendiente de la función.

2.3.1. Retropropagación

La red neuronal trata de propagar su señal de entrada por la arquitectura de la red para obtener una salida. Esta salida será comparada con el valor esperado y finalmente el error obtenido será retropropagado para ajustar los parámetros de la red en la dirección de menor error haciendo uso del método de gradiente visto en la sección 2.3.

Retropropagación resumen

La técnica de retropropagación para la red neuronal se resume por medio de cuatro ecuaciones para la actualización de sus parámetros en todas las capas.

$$\delta_j^{(n)} = \frac{\partial E}{\partial a_j^{(n)}} \quad (2.17)$$

$$\delta_i^{(n-1)} = \sigma'(a_i^{(n-1)}) \sum_{k=0}^N \delta_k^{(n)} w_{kj}^{(n)} \quad (2.18)$$

$$\frac{\partial E}{\partial w_{ij}^{(n)}} = \delta_i^{(n)} z_j^{(n-1)} \quad (2.19)$$

$$\frac{\partial E}{\partial b_i^{(n)}} = \delta_i^{(n)}. \quad (2.20)$$

Estas ecuaciones serán desarrolladas en esta sección con objeto de conocer y comprender su origen.

Gradiente de la función de error

Para llevar a cabo la actualización de los pesos deberemos calcular la derivada parcial de la función de error E con respecto al peso $w_{ij}^{(n)}$.

$$w_{ij}^{(n)\prime} = w_{ij}^{(n)} + \Delta w_{ij}^{(n)}, \quad (2.21)$$

Donde Δw_{ij} es equivalente a:

$$-\eta \left(\frac{\partial E}{\partial w_{ij}^{(n)}} \right), \quad (2.22)$$

Actualización de los pesos de la última capa

La aplicación del algoritmo de retropropagación comienza con la utilización de la regla de la cadena para derivar parcialmente la función de error con respecto al peso $w_{ij}^{(n)}$.

$$\frac{\partial E}{\partial w_{ij}^{(n)}} = \frac{\partial E}{\partial a_j^{(n)}} \frac{\partial a_j^{(n)}}{\partial w_{ij}^{(n)}}, \quad (2.23)$$

donde $a_j^{(n)}$ corresponde con la activación de la neurona j en la capa $l^{(n)}$.

Esta ecuación anterior se divide en dos términos, el primer término es denominado **error** y es denotado como:

$$\frac{\partial E}{\partial a_j^{(n)}} \equiv \delta_j^{(n)}. \quad (2.24)$$

El segundo término se calcula en base a la expresión del estado de activación $a_j^{(n)}$,

$$a_j^{(n)} = \sum_{i=0}^M w_{ij}^{(n)} z_i^{(n-1)} + b_i^{(n)}. \quad (2.25)$$

Entonces,

$$\frac{\partial a_j^{(n)}}{\partial w_{ij}^{(n)}} = \frac{\partial}{\partial w_{ij}^{(n)}} \left(\sum_{i=0}^M w_{ij}^{(n)} z_i^{(n-1)} + b_i^{(n)} \right) = z_i^{(n-1)}. \quad (2.26)$$

Finalmente, el resultado de la Ecuación 2.17 viene dado por:

$$\frac{\partial E}{\partial w_{ij}^{(n)}} = \delta_j^{(n)} z_i^{(n-1)}. \quad (2.27)$$

Hasta ahora se ha presentado la función de error de forma genérica con independencia del tipo de función usada. Sin embargo, el término $\delta_j^{(n)}$ es dependiente de la misma.

Debido a las motivaciones de este trabajo utilizaremos una función de error denominada **entropía cruzada**, la cual nos sirve para establecer cómo de buena o aproximada es la salida de la red con respecto a la clasificación real del elemento de entrada. La función E se define formalmente como:

$$E : \mathbb{R}^{(i+j)+i} \rightarrow \mathbb{R} \quad (2.28)$$

$$(w_{11}, \dots, w_{1j}, \dots, w_{i1}, \dots, w_{ij}, b_1, \dots, b_i) \rightarrow E(w_{11}, \dots, w_{1j}, \dots, w_{i1}, \dots, w_{ij}, b_1, \dots, b_i),$$

$$E(\theta; X, y) = - \sum_{d=1}^N t_d \log z_i^{(n)} = - \sum_{d=1}^N t_d (a_d^{(n)} - \log \sum_{c=1}^N e^{a_c^{(n)}}), \quad (2.29)$$

donde N se corresponde con el número de neuronas en la última capa y t_d es un vector categórico: con un 1 en la clase o categoría correcta y 0 en el resto. Este tipo de codificación es conocida como One-hot. θ corresponde con los parámetros de la red. La función E , depende de los pesos de la red

y, los parámetros bias o umbral (si estos últimos se encuentran presentes).

Determinada la función de error podemos proceder al cálculo del error $\delta_j^{(n)}$, esto es:

$$\begin{aligned}
 \frac{\partial E}{\partial a_j^{(n)}} &\equiv \delta_j^{(n)} \\
 &= - \sum_{d=1}^N t_d \left(1_{d=j} - \frac{a_j^{(n)}}{\sum_{c=1}^N a_c^{(n)}} \right) \\
 &= - \sum_{d=1}^N t_d \left(1_{d=j} - z_j^{(n)} \right) \\
 &= \sum_{d=1}^N t_d z_j^{(n)} - \sum_{d=1}^N t_d 1_{d=j} \\
 &= z_j^{(n)} \sum_{d=1}^N t_d - t_j \\
 &= z_j^{(n)} - t_j,
 \end{aligned} \tag{2.30}$$

donde $1_{d=j}$ se corresponde con una función delta de Kromecker:

$$1_{d=j} = \begin{cases} 1 & \text{si } d = j \\ 0 & \text{e.o.c.} \end{cases}. \tag{2.31}$$

Conociendo además que t_d en un vector que solo puede almacenar un 1 en la clase a la que pertenece, esto se expresa mediante la siguiente ecuación:

$$\sum_{d=1}^N t_d = 1, \tag{2.32}$$

debido a la mencionada codificación One-hot del vector t_d .

El resultado de realizar la actualización de los pesos de la última capa viene dado por:

$$\begin{aligned}
 \frac{\partial E}{\partial w_{ij}^{(n)}} &= \delta_j^{(n)} z_i^{(n-1)} \\
 &= (z_j^{(n)} - t_j) z_i^{(n-1)}.
 \end{aligned} \tag{2.33}$$

Si el elemento “bias” no se encontrase en la matriz de pesos incluido, deberíamos por tanto calcular la influencia del parámetro “bias” con respecto a la función de error E , esto es:

$$b_j^{(n)\prime} = b_j^{(n)} - \eta \left(\frac{\partial E}{\partial b_j^{(n)}} \right), \tag{2.34}$$

conociendo que,

$$\frac{\partial E}{\partial b_j^{(n)}} = \delta_j^{(n)}(1) = \delta_j^{(n)}. \tag{2.35}$$

Actualización de los pesos de capas anteriores

De forma análoga a la anterior deberemos realizar una actualización de los parámetros con la aplicación de la regla de la cadena. Tomemos, por lo tanto, la actualización de los pesos de la capa $(n - 1)$ -ésima:

$$\frac{\partial E}{\partial w_{ij}^{(n-1)}} = \frac{\partial E}{\partial z_i^{(n-1)}} \frac{\partial z_i^{(n-1)}}{\partial a_i^{(n-1)}} \frac{\partial a_i^{(n-1)}}{\partial w_{ij}^{(n-1)}}. \quad (2.36)$$

La primera de las derivadas parciales se calcula como:

$$\frac{\partial E}{\partial z_i^{(n-1)}} = \sum_{k=1}^M \frac{\partial E}{\partial a_k^{(n)}} \frac{\partial a_k^{(n)}}{\partial z_i^{(n-1)}} = \sum_{k=1}^M \delta_k^{(n)} w_{kj}^{(n)}, \quad (2.37)$$

La equivalencia de la expresión $\frac{\partial E}{\partial a_k^{(n)}} \equiv \delta_k^{(n)}$ proviene de la fórmula (2.22).

$$\frac{\partial z_i^{(n-1)}}{\partial a_i^{(n-1)}} = \sigma'(a_i^{(n-1)}), \quad (2.38)$$

donde $\sigma'(a_i^{(n-1)})$ se corresponde con la derivada de la función de activación usada.

$$\frac{\partial a_i^{(n-1)}}{\partial w_{ij}^{(n-1)}} = \frac{\partial}{\partial w_{ij}^{(n-1)}} \left(\sum_{i=1}^N w_{ij}^{(n-1)} a_i^{(n-2)} \right) = a_i^{(n-2)}, \quad (2.39)$$

El valor N corresponde con el número de neuronas de la capa $l^{(n-1)}$.

Combinando las anteriores expresiones obtenemos:

$$\frac{\partial E}{\partial w_{ij}^{(n-1)}} = a_i^{(n-2)} \sigma'(a_i^{(n-1)}) \sum_{k=1}^M \delta_k^{(n)} w_{kj}^{(n)}. \quad (2.40)$$

Podemos definir análogamente, $\delta_j^{(n-1)} \equiv \sigma'(a_i^{(n-1)}) \sum_{k=1}^M \delta_k^{(n)} w_{kj}^{(n)}$. Finalmente y reescribiendo la expresión (2.33) obtenemos:

$$\frac{\partial E}{\partial w_{ij}^{(n-1)}} = \delta_j^{(n-1)} a_i^{(n-2)}. \quad (2.41)$$

Si contásemos con el parámetro umbral, su expresión en el cálculo del error vendría dado por la siguiente expresión (I-Ta Lee, 2017):

$$\frac{\partial E}{\partial b_i^{(n-1)}} = \frac{\partial E}{\partial z_i^{(n-1)}} \frac{\partial z_i^{(n-1)}}{\partial a_i^{(n-1)}} \frac{\partial a_i^{(n-1)}}{\partial b_i^{(n-1)}} = \delta_i^{(n-1)}(1) = \delta_i^{(n-1)}. \quad (2.42)$$

2.4– Regularización

Como se introdujo en la introducción de esta capítulo, la capacidad de la red neuronal para generalizar ante nuevas imágenes con las que la red no ha sido entrenada, es uno de los principales problemas de las redes neuronales. Una red neuronal entrenada para ajustarse, o clasificar correctamente, al conjunto de entrenamiento con la imposibilidad de categorizar nuevos ejemplos adecuadamente, se le conoce como una red sobreajustada. Siendo primordial que la red responda de manera adecuada ante nuevas entradas, se hace de vital importancia la utilización de regularizadores. Las estrategias usadas se basan en reducir el coste de la función de error para el conjunto

de datos de prueba, con el cual la red no ha sido entrenada, a costa de aumentar el error para el conjunto de entrenamiento.

Estas estrategias son conocidas como **regularización**. Se basan en añadir un término de penalización a la función de error con el objetivo de restringir el valor de los pesos de la red. Con una elección correcta de la penalización se obtendrá una mejor generalización ante nuevas entradas.

2.4.1. Regularización L2 (arista) y L1 (Lasso)

Una regularización L2 es conocida como regularización de Tikhonov o de arista. Se trata de una de las formas más comunes de regularización la cual se implementa añadiendo un término que consiste en el producto de un parámetro de penalización λ y la suma al cuadrado de los pesos.

Mediante la definición de $E(\theta; X, y)$ (ecuación 2.28) podemos añadir el término regularizador a la función de error, conociendo a la nueva función como \hat{E}

$$\hat{E}(\theta; X, y) = E(\theta; X, y) + \frac{\lambda}{2} \sum_{i=1}^M \sum_{j=1}^N w_{ij}^2, \quad (2.43)$$

L1 es otro tipo común de regularización. Se basa en añadir el valor absoluto de los pesos junto con un término regularizador λ . Su uso combinado con L2, se conoce como "Regularización elastic net", $\lambda_1|w_{ij}| + \lambda_2 w_{ij}^2$. La regularización L1 tiene la propiedad que provoca una caída generalizada en el valor de los pesos. Esto lo convierte en una opción menos efectiva que utilizar la regularización L2. Se le conoce con el nombre de regularización Lasso.

$$\hat{E}(\theta; X, y) = E(\theta; X, y) + \frac{\lambda}{2} \sum_{i=1}^M \sum_{j=1}^N |w_{ij}|. \quad (2.44)$$

La idea intuitiva detrás de la utilización de esta técnica es evitar grandes valores en algunos pesos y por tanto preferir una distribución de menor valor y más uniforme de los pesos. Esto hace que la red interaccione en menor medida pero con todos los valores del vector de entrada X , en lugar de interaccionar enormemente con algunos valores (Karpathy, 2019c).

2.4.2. Dropout

Corresponde con una técnica simple y efectiva de regularización, comúnmente usada con otros métodos de regularización como L2 o L1. Este método trata de regular la activaciones de las neuronas mediante una probabilidad p . Para describir el proceso tenemos,

$$\begin{aligned} r_j^{(n)} &\sim Bernoulli(p) \\ \hat{y}_j^{(n)} &= r_j^{(n)} y_j^{(n)} \\ a_j^{(n+1)} &= \sum_{i=1}^M w_{ij}^{(n+1)} \hat{y}_j^{(n)} + b_i^{(n+1)} \\ z^{(n+1)} &= \sigma(a_j^{(n+1)}) \end{aligned} \quad (2.45)$$

Dropout introduce un hiper-parámetro extra que puede ser controlado para moderar la intensidad de eliminación de las conexiones. Una probabilidad $p = 1$, implica que no se produce dropout. Los valores recomendados para las capas ocultas varían entre 0.5 y 0.8. Un valor de p demasiado pequeño puede dar lugar a una falta de aprendizaje por la red ante los ejemplos entrenados. Un valor muy alto en el dropout podría no ser suficiente para prevenir el overfitting. En la Figura 2.8 se aprecia el proceso de eliminación de neuronas (Srivastava et al., 2014).

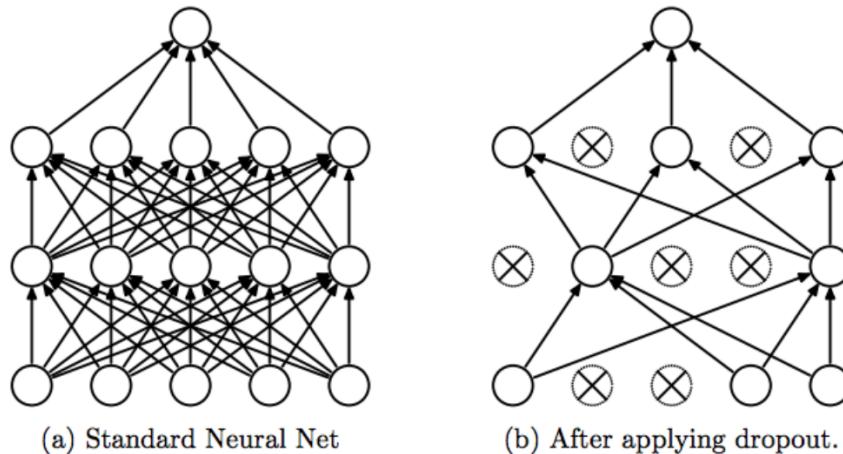


Figura 2.8: Proceso de dropout en el que se eliminan algunas conexiones existentes.

2.5- Convolución

La operación de convolución esta formada fundamentalmente por dos componentes: un volumen de entrada $H \times W \times D$, donde H se corresponde con la altura, W con la anchura y D con la profundidad, y un kernel o filtro el cual tiene un tamaño menor espacialmente (alto y ancho) $H' \times W' \times D$, donde $H' \leq H$ y $W' \leq W$, pero se desplaza a través de todo el volumen de entrada.

Durante el proceso, convolucionaremos, o desplazaremos, el filtro a través del ancho y alto del volumen de entrada, obteniendo, como resultado, el producto escalar de los valores del filtro por los del volumen. Esto produce un mapa de características bidimensional, que corresponde con la activación del filtro en cada localización espacial del volumen de entrada. Intuitivamente, la red neuronal de convolución aprenderá filtros que se activarán cuando detecten distintos tipos de patrones, como pueden ser bordes o ejes, y bloques de color. Antes de definir distintos parámetros modificables dentro del proceso de convolución, hago referencia a la figura 2.9 para ilustrar el proceso descrito anteriormente, de forma más compacta, en la figura se puede observar como d filtros, para la figura $d=2$, serán convolucionados con la imagen de entrada para producir mapas de características bidimensionales de tamaño $n \times m$, normalmente de menor tamaño espacial (alto y ancho) que la imagen de entrada. La combinación de estos mapas de características dan lugar a la salida del proceso de convolución de volumen $n \times m \times d$.

Existen tres parámetros que permiten modificar el volumen de salida. Estos parámetros son compartidos por toda una misma capa de convolución:

- 1. Profundidad (Depth):** Se corresponde con el número de filtros a usar. Cada uno de ellos buscará distintos patrones en el volumen de entrada.
 - 2. Paso (Stride):** Especifica un valor entero que cuantifica el desplazamiento del filtro. Un valor de paso a uno recorrerá la imagen al completo aplicando el filtro en cada posición espacial. Un valor mayor de paso hará que el desplazamiento sea mayor. Al producirse menos operaciones de convolución el mapa de características de salida será menor. Se ilustra una convolución con paso a uno en la figura 2.10 y a continuación una con paso dos en la figura 2.11.
 - 3. Zero padding:** Se trata del proceso de añadir un margen de ceros al volumen de entrada de forma que se producen dos beneficios.
 - a) Permite que podamos usar las capas de convolución sin necesariamente disminuir el tamaño del volumen de entrada. Esto es algo de vital importancia para redes neuro-

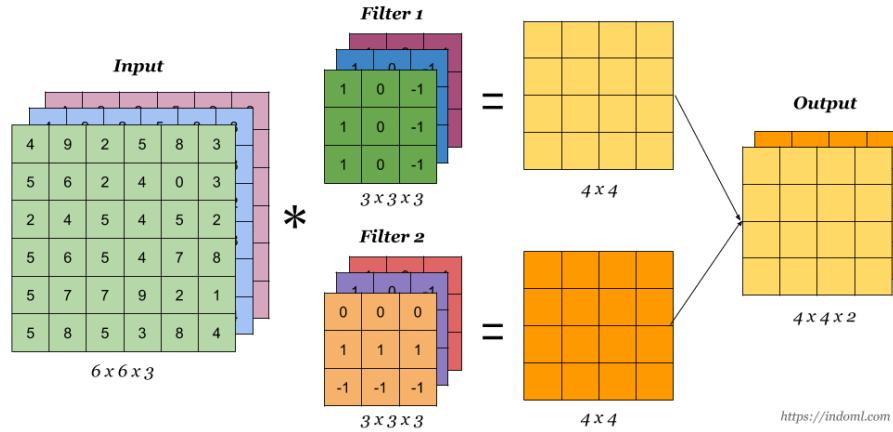


Figura 2.9: Ejemplo del proceso de convolución para un volumen de entrada de profundidad tres, y con la aplicación de dos filtros, produciendo así dos mapas de características que dan lugar al volumen de salida con profundidad dos, como esperábamos, igual al número de filtros utilizados.

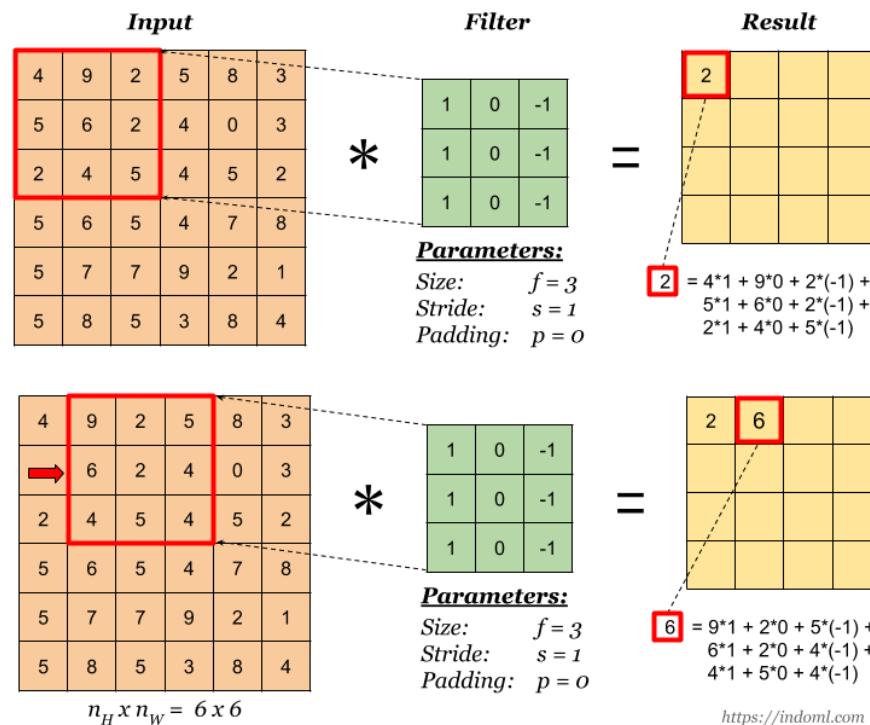


Figura 2.10: Convolución con uso de paso o stride 1.

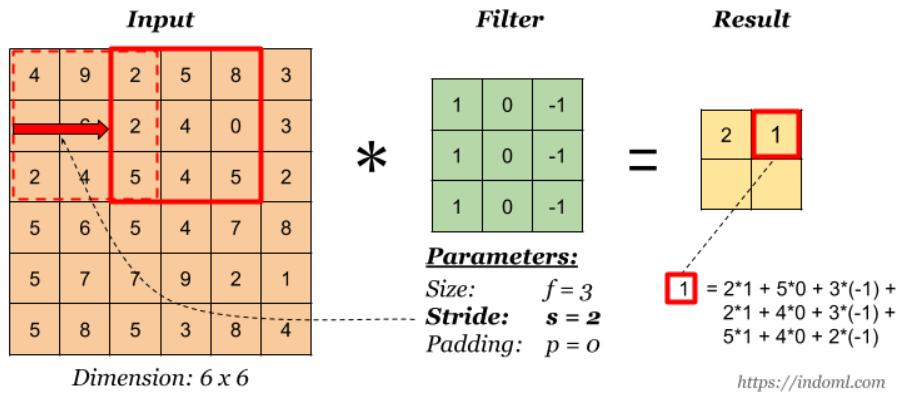


Figura 2.11: Convolución con uso de paso o stride 2. Como observamos además, el mapa de características resultante es espacialmente más pequeño que usar un paso menor.

nales profundas, con un gran número de capas, donde si no se aplicase zero padding disminuiríamos demasiado el volumen de entrada en muy pocas convoluciones.

- b) Se almacena más información de los bordes de la imagen que de otra forma serían prácticamente irrelevantes. La figura 2.12 ilustra la colocación de los ceros en los bordes de la entrada.

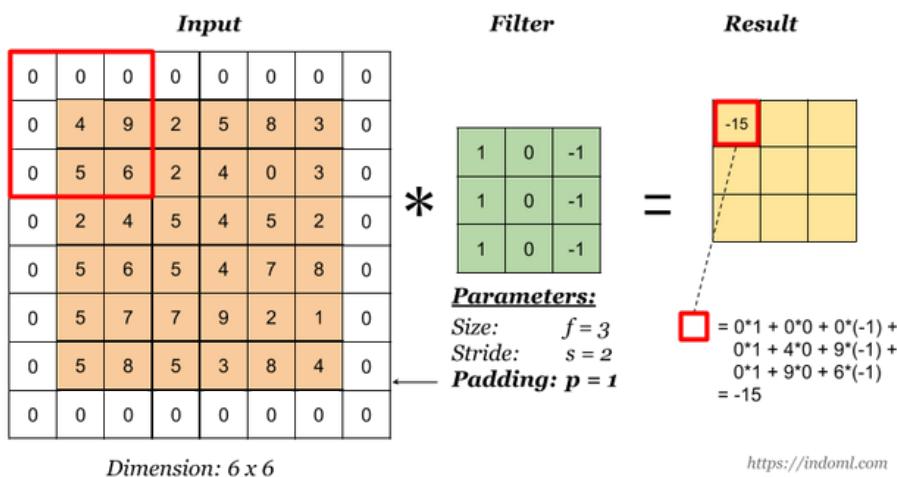


Figura 2.12: Proceso de zero padding.

Para la técnica de zero padding existen dos terminologías que aparecen recurrentemente: “**valid**” la cual se identifica con no usar padding y “**same**” que corresponde con usar padding para que el volumen de salida sea igual que el de entrada.

En base al volumen de entrada se puede calcular el volumen de salida si se conocen los parámetros de la convolución a utilizar como se indica a continuación:

- El proceso de convolución acepta un volumen de entrada de tamaño $\mathbf{W} \times \mathbf{H} \times \mathbf{D}$
- Requiere cuatro parámetros
 - Número de filtros a usar, \mathbf{K} ,
 - Su longitud espacial del filtro, \mathbf{F} , por lo general suele ser $\mathbf{F} \times \mathbf{F}$,
 - El paso o stride, \mathbf{S} ,

- La cantidad de zero padding a usar, \mathbf{P} .
- Producirá de salida un volumen $\mathbf{W}' \times \mathbf{H}' \times \mathbf{D}'$, donde:
 - $\mathbf{W}' = (\mathbf{W} - \mathbf{F} + 2\mathbf{P}) / \mathbf{S} + 1$,
 - $\mathbf{H}' = (\mathbf{H} - \mathbf{F} + 2\mathbf{P}) / \mathbf{S} + 1$,
 - $\mathbf{D}' = \mathbf{D}$.
- Teniendo en cuenta que el valor de \mathbf{W}' y \mathbf{H}' debe resultar entero, valores comunes para esta operación suelen ser $\mathbf{F} = 3$, $\mathbf{S} = 2$ y $\mathbf{P} = 1$.

2.5.1. Pooling o agrupación

Esta operación trata de reducir, tras una capa de convolución, el tamaño espacial del volumen de entrada, así como el número de parámetros y cálculos de la red. Esta operación se aplica a un volumen de salida $H \times W \times D$, y es independiente para cada capa D . Al igual que la operación de convolución, hará uso de un filtro F de tamaño $F \times F$, normalmente 2×2 , y un paso o stride con valor entero, comúnmente dos. A diferencia de convolución, se utilizará la operación MAX. Sin embargo, existen otras opciones como la media aritmética, para calcular el máximo valor dentro del espacio 2×2 del filtro y quedarnos con ese valor. En la figura 2.13 se puede comprender mejor el funcionamiento de esta operación.

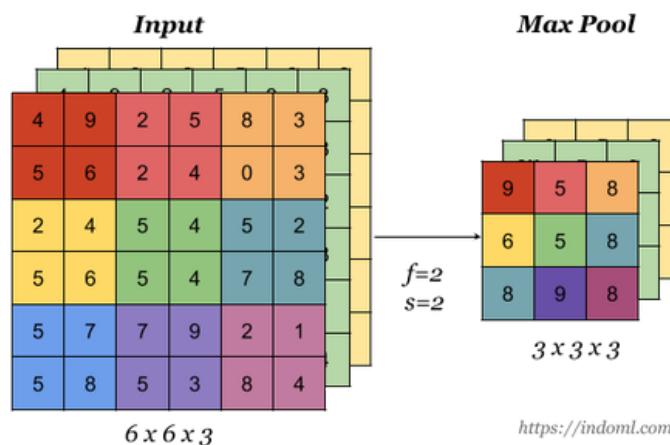


Figura 2.13: Proceso de pooling o agrupación para la reducción del volumen mientras mantiene las características más relevantes del volumen de entrada.

Para la operación de agrupación también podemos conocer el volumen de salida en función de sus parámetros al igual que ocurre con la operación de convolución.

- El proceso de agrupación acepta un volumen de entrada de tamaño $\mathbf{W} \times \mathbf{H} \times \mathbf{D}$
- Requiere cuatro parámetros
 - Su longitud espacial del filtro, \mathbf{F} , por lo general suele ser $\mathbf{F} \times \mathbf{F}$,
 - El paso o stride, \mathbf{S} .
- Producirá de salida un volumen $\mathbf{W}' \times \mathbf{H}' \times \mathbf{D}'$, donde:
 - $\mathbf{W}' = (\mathbf{W} - \mathbf{F}) / \mathbf{S} + 1$,
 - $\mathbf{H}' = (\mathbf{H} - \mathbf{F}) / \mathbf{S} + 1$,
 - $\mathbf{D}' = \mathbf{D}$.

- Teniendo que tener en cuenta que el valor de \mathbf{W}' y \mathbf{H}' debe resultar entero. Valores comunes para esta operación suelen ser $F = 2$ o 3 y $S = 2$.

Textos basados en los apuntes del profesor Andrej Karpathy de la Universidad de Stanford, (Karpathy, 2019b), por otro lado las imágenes han sido tomadas de la web (Mesin, 2018).

CAPÍTULO 3

Adversarial examples

Un número cada vez mayor de aplicaciones en el mundo han sido impulsadas por redes neuronales. Por ejemplo, empresas de TI y de la industria automovilística (Google, Tesla, Uber, ...) están probando los coches con piloto autónomo, que requieren del uso de técnicas de aprendizaje profundo como el reconocimiento de objetos, aprendizaje reforzado y otra variedad de técnicas específicas para la tarea. Otro ejemplo sería la autenticación facial por parte de los terminales Apple y Android para desbloquear teléfonos móviles.

A pesar de los grandes éxitos en numerosas aplicaciones, muchas de estas aplicaciones que hacen uso de redes neuronales son cruciales para la vida de otros seres humanos, lo que plantea grandes preocupaciones en el campo de la seguridad y la protección. Estudios recientes han encontrado que el aprendizaje profundo es vulnerable frente a un conjunto especialmente diseñado de muestras. Estas muestras pueden engañar fácilmente a un modelo neural bien entrenado, realizando pequeñas perturbaciones imperceptibles para humanos en sus entradas a la red.

3.1– Concepto

Con el objetivo de generar muestras que ofreciesen una clasificación errónea de la red, los investigadores (Szegedy y cols., 2014) en su trabajo “Propiedades intrigantes de las redes neuronales”, consiguieron generar por primera vez pequeñas perturbaciones imperceptibles en las imágenes de entrada de la red haciendo que estas fuesen erróneamente clasificadas, este proceso se llevó a cabo mediante la generación de perturbaciones que maximicen la función de coste o error, E , asociada al sistema. Estas muestras mal clasificadas recibieron el nombre “adversarial examples”.

Los primeros algoritmos para la generación de “adversarial examples” necesitaban acceso al modelo. En particular, estos algoritmos debían ser capaces de calcular los gradientes para las entradas dadas. Esta restricción acabó siendo innecesaria ya que con el vector de probabilidades o la clasificación de la entrada proporcionada por el atacante, es información suficiente para poder generar “adversarial examples”. En particular, aquellos “adversarial examples” que son generados sin acceso al modelo se conocen como ataques de caja negra. Además, los “adversarial examples”, en ocasiones, son transferibles entre modelos, es decir, es posible encontrar un “adversarial example” que genere una clasificación incorrecta en distintas arquitecturas neuronales.

3.2– Importancia

A medida que las redes neuronales se integran más en sistemas como vehículos autónomos o dispositivos médicos, también se están convirtiendo en puntos de entrada para los posibles atacantes. Incluso si las predicciones de un modelo neuronal ante un conjunto de datos de prueba

son 100 % correctas, se pueden encontrar “Adversarial examples” que consigan engañar a la red neuronal. La construcción de modelos neuronales que sean seguros contra ataques es una nueva parte en el campo de la ciberseguridad que esta cobrando vital importancia.

Se prevé y se están utilizando aplicaciones basadas en las redes neuronales, en entornos críticos para la seguridad. Mientras tanto, recientes estudios muestran que los “adversarial examples” pueden aplicarse en el mundo real. Un atacante con malas intenciones puede construir “adversarial examples” físicos y confundir a vehículos autónomos manipulando la señal de tráfico en reconocimiento o eliminando la segmentación de los peatones en un sistema de reconocimiento de objetos. En la figura 3.1 podemos observar un ejemplo de las implicaciones que podría tener para la seguridad, la utilización de sistemas de reconocimiento visual mediante redes neuronales.

Esto deja la investigación en “adversarial examples” en un punto de suma importancia para la seguridad de sistemas que utilicen redes neuronales.

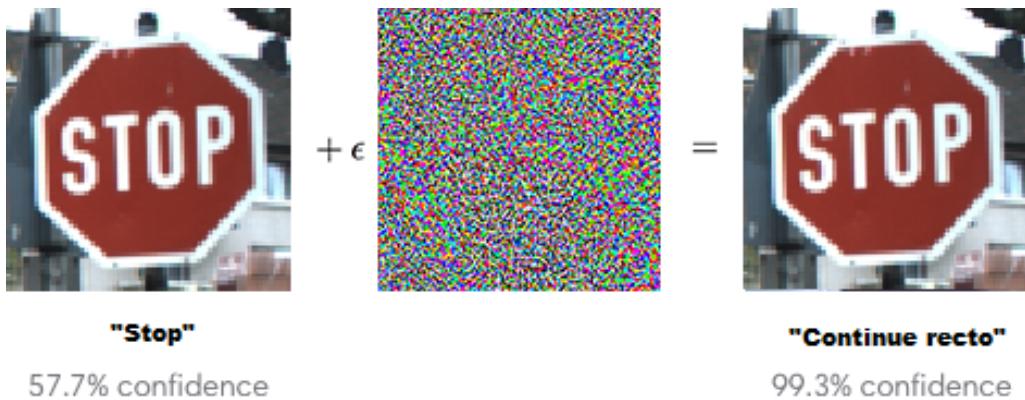


Figura 3.1: Una perturbación para generar “adversarial examples” en la imagen de entrada, puede provocar una clasificación incorrecta por la red neuronal.

Los investigadores (Kurakin, Goodfellow, y Bengio, 2016), han demostrado que si toman una imagen que la red pueda clasificar, generan un “adversarial example”, lo imprimen en un papel, en físico, y ahora toman una fotografía del elemento físico generado que contiene el “adversarial example” y lo envían a un modelo neuronal entrenado para reconocer imágenes, el clasificador producirá clasificaciones incorrectas, siendo por tanto posible generar “adversarial examples” por medio de objetos físicos, demostrando la vulnerabilidad de las redes neuronales que hagan uso de sistemas de reconocimiento por video. Posteriormente, surgen impresiones de adversarial examples en tres dimensiones que permiten realizar ataques a modelos que reconocen señales de tráfico urbano (Evtimov y cols., 2017). En la figura 3.2 podemos observar un adversarial example físico que dio lugar a una clasificación incorrecta de la señal por medio de una modificación al objeto físico. En marzo de 2019, los investigadores de Tencent lograron que un Tesla se dirigiera hacia el tráfico en sentido contrario engañando al asistente de detección de los carriles con un ataque de adversario (Tencent Keen Security Lab, 2019).

Los ejemplos de “adversarial examples” no son específicos de la visión por computador. Por ejemplo, estos ataques también se pueden realizar en modelos de procesamiento de lenguaje natural, modelos de detección de malware, modelos de reconocimiento automático de voz con sonido grabado por un micrófono, etc.

Con estos ejemplos se pretende visibilizar la importancia a la hora de implantar arquitecturas neuronales en sistemas que se encuentren en producción y puedan tener un impacto negativo en los usuarios y la imagen corporativa de una empresa.



Figura 3.2: En la imagen de la izquierda una señal de tráfico con un grafiti. En la derecha una imagen con una modificación adversaria de forma que la señal de Stop sea clasificada de forma errónea.

3.3– Ataque de un pixel

3.3.1. Descripción

Los “adversarial examples” se corresponden con un problema real de las redes neuronales. La idea principal para atacar una red neuronal de clasificación, y en particular para clasificación de imágenes, es crear imágenes adversarias en las que se añaden una pequeña perturbación diseñada para cambiar el resultado de clasificación original proporcionado por la red neuronal ante la imagen de entrada. Además, que esta perturbación sea imperceptible para el ojo humano.

Muchos de estos ataques no consideran escenarios extremadamente limitados para la generación de imágenes adversarias, por ejemplo algunos algoritmos no tienen una limitación en el número de píxeles a modificar, provocando que la imagen adversaria generada pueda ser reconocida como una imagen alterada. En la figura 3.3, se puede observar una clara modificación en el fondo de la imagen. Este tipo de detalles son los que pueden dar lugar a que un humano pueda darse cuenta de que la imagen se encuentra modificada.

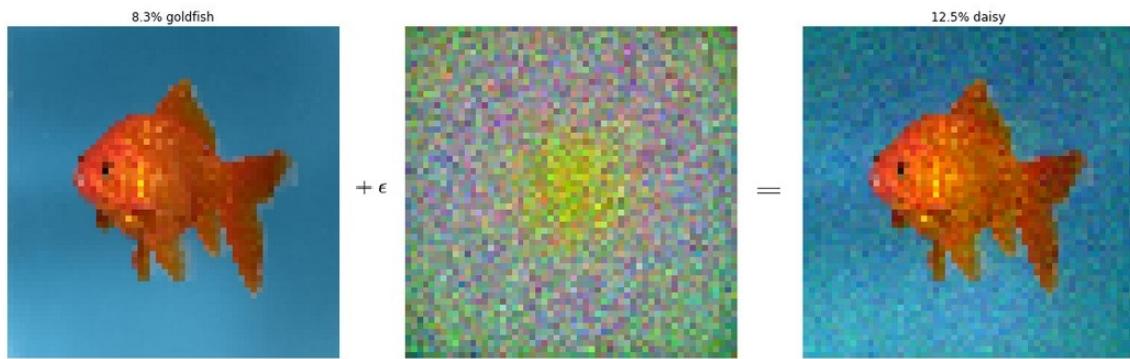


Figura 3.3: En la figura podemos apreciar como un ataque adversario a una imagen sin limitación alguna, puede dar lugar a imágenes que sean fácilmente reconocibles como modificadas.

En este proyecto se propone una forma de generar imágenes adversarias teniendo una clara limitación en el escenario de ataque: generar adversarial examples mediante una cantidad limitada

de píxeles a modificar en la imagen original. En particular, se propone perturbar un solo píxel de la imagen de entrada mediante un algoritmo de evolución diferencial, siguiente sección 3.3.2. Este ataque se encuentra dentro de la categoría de ataque de caja negra, es decir, solo se conocen las probabilidades de pertenencia a cada categoría proporcionadas por la red neuronal y no hace falta acceso o conocimiento de la arquitectura interna.

Un ataque a un píxel puede ser un candidato excelente para esconder modificaciones adversarias en imágenes. Hasta la fecha, ninguna investigación ha permitido garantizar si una perturbación será imperceptible. Es por esto que reduciendo el número de modificaciones a las mínimas posibles se mitiga parte de este problema.

Generar imágenes adversarias puede ser formalizado como un problema de optimización con restricciones, si consideramos que la imagen de entrada puede ser representada como un vector en el que cada escalar representa a un píxel.

De esta forma, f se define como la categoría objetivo del clasificador neuronal dado una entrada n-dimensional. Sea $\mathbf{x} = (x_1, \dots, x_n)$, la imagen de entrada sin modificación alguna y correctamente clasificada en la categoría t . La probabilidad de pertenencia de \mathbf{x} a la categoría t es por tanto $f_t(\mathbf{x})$.

El vector $e(\mathbf{x}) = (e_1, \dots, e_n)$, es un vector de perturbaciones adversarias en función de \mathbf{x} . La categoría objetivo adv indica una clasificación distinta de t . d se identifica con las modificaciones máximas a realizar, donde d es un pequeño valor entero. Para el caso del ataque de un pixel, $d = 1$. El objetivo es encontrar una solución $e(\mathbf{x})^*$ que responda a las dos cuestiones siguientes:

- Las dimensiones que deberán ser perturbadas.
- Identificar cómo será la intensidad de la modificación en cada dimensión.

La ecuaciones que responden a nuestro problema planteado son:

$$\text{maximizar}_{e(x)^*} \quad f_{adv}(x + e(x)), \quad (3.1)$$

$$\text{sujeto a} \quad \|e(x)\|_0 \leq d. \quad (3.2)$$

A diferencia de otras investigaciones las cuales realizan una modificación en todas las dimensiones, en nuestro caso, solo d dimensiones se verán modificadas quedando el resto de dimensiones de $e(\mathbf{x})$ a cero. En la figura 3.4, podemos apreciar el resultado de aplicar el ataque de un pixel a unas imágenes, junto con su clasificación antes y después de aplicar la perturbación.

3.3.2. Evolución diferencial

Para realizar una optimización de caja negra, como es en nuestro caso, conociendo únicamente la probabilidad de pertenencia a una categoría, puede ser poco eficiente utilizar un método de optimización basado en gradiente. Sería conveniente utilizar método para encontrar buenas soluciones sin importar la diferenciabilidad en la superficie de la función. Es conveniente por tanto el uso de un algoritmo genético denominado evolución diferencial.

Se trata de un método de optimización evolutivo. Una parte de la población es evaluada ante un nuevo individuo para generar nuevos posibles candidatos para la población sustituta en base a una función de fitness que evalúa la calidad de cada individuo. En el contexto del proyecto cada solución candidata es un vector

$$\mathbf{X} = (x_i, y_i, r_i, g_i, b_i), \forall i \in [1, d], \quad (3.3)$$

siendo d el valor mínimo de perturbaciones como mencionamos en la sección 3.3.1.

Primero, generamos una población con n perturbaciones candidatas

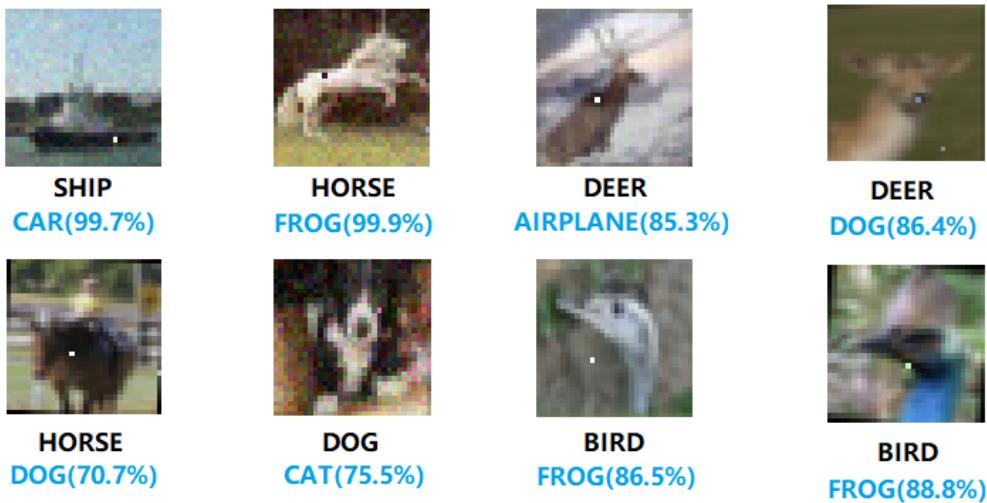


Figura 3.4: Se observa como la modificación de un único píxel puede cambiar el resultado de la red neuronal a una categoría distinta de la original.

$$\mathbf{P} = (X_1, X_2, \dots, X_n). \quad (3.4)$$

Para cada iteración, calcularemos n nuevos hijos mutantes que pertenecerán a la futura población. Estos hijos son generados en base al algoritmo de mutación usual de evolución diferencial

$$X_i(g+1) = X_{r1}(g) + F(X_{r2}(g) - X_{r3}(g)), \quad (3.5)$$

tal que,

$$r1 \neq r2 \neq r3, \quad (3.6)$$

donde $r1$, $r2$ y $r3$ son índices aleatorios de la población \mathbf{P} , y $F = 0.5$ es un parámetro de mutación. El término g representa el índice de población actual dentro de la ejecución del algoritmo. En resumen, Utilizamos tres individuos de la población anterior para generar un nuevo individuo. Si este nuevo candidato al realizar la perturbación de sus valores en las posiciones x_i , y_i , y valores RGB r_i , g_i y b_i , obtiene una menor probabilidad de pertenencia a la categoría original de la imagen, este nuevo candidato $X_i(g+1)$ reemplaza al candidato $X_i(g)$.

Este método al ser meta-heurístico y no depender de optimizaciones basadas en gradiente, lo hacen un método efectivo para evitar mínimos locales en nuestro problema. Tratándose como en nuestro caso de un problema de optimización complejo debido a sus restricciones, no es posible modificar todos los píxeles que deseemos de la imagen hasta encontrar un “adversarial example”, sino que únicamente es posible la modificación de un píxel en la imagen de entrada. Estas características hacen al algoritmo de evolución diferencial un método idóneo para resolver este tipo de problemas con optimizaciones complejas debido a multitud de mínimos locales y duras restricciones que disminuyen el espacio de búsqueda.

El enfoque tomado en este proyecto hace que sea independiente del modelo neuronal seleccionado, haciendo que sea factible para el ataque el conocimiento probabilístico de las salidas de la red. Es por ello, que un grupo de investigadores pudieron demostrar que es posible realizar ataques de caja negra ante páginas web que dispongan de sistemas de clasificación basados en redes neuronales de reconocimiento visual (Liu, Chen, Liu, y Song, 2016)

CAPÍTULO 4

Desarrollo del proyecto

Este capítulo se divide en dos partes fundamentales. En primer lugar, se trata de diseñar una red neuronal para la clasificación de imágenes. Posteriormente, la generación de un ataque a la red neuronal mediante la búsqueda de una imagen modificada que dé lugar al concepto conocido como adversarial example, visto en el capítulo 3. Se generará un “ataque”, basado en la investigación propuesta por (Su y cols., 2017), mediante un algoritmo de evolución diferencial, realizando una modificación de la imagen original mediante la alteración de una cantidad de píxeles d . De esta forma, la categoría de clasificación original de la imagen será alterada, obteniendo por la red una categorización errónea ante la imagen. Este proceso se trató en mayor profundidad en el capítulo 3.

4.1– Lenguaje y entorno de desarrollo

Se recogen las herramientas usadas para poder completar el proyecto, así como las principales librerías del lenguaje de programación usado.

- **Python:** Se define como un lenguaje de programación interpretado y multiparadigma, ya que soporta la orientación a objetos, programación imperativa y programación funcional.

En general es un lenguaje de programación con una activa comunidad, lo que favorece al mantenimiento de librerías especialmente útiles para el manejo de matrices, tratamiento de datos, visualización de información, etc...

Jupyter Notebook: Para el desarrollo de este proyecto se ha optado por la utilización de un entorno de programación denominado como “notebook” o “cuaderno”. Jupyter es un entorno de trabajo que nos permite la interpretación de nuestro código Python de manera interactiva y visual desde nuestro navegador. Permite la inclusión de texto en formato Markdown para la creación de presentaciones junto con partes del código. Por estos motivos, muchos trabajos realizados en Python optan por el uso de este entorno.

- **Tensorflow:** Es biblioteca de código abierto para la creación, entrenamiento y evaluación de redes neuronales, en particular este software creado por Google permite su ejecución en distintos entornos como podrían ser CPU, GPU, servidores o dispositivos móviles. Tensorflow permite trabajar de forma eficiente con las recurrentes operaciones matriciales que suelen ser usadas en el desarrollo de redes neuronales.
- **Keras:** Es una librería de código abierto para Python que proporciona, de forma sencilla, la creación de modelos de Deep Learning a través de una interfaz de programación de aplicaciones (API), usando como apoyo principal la librería TensorFlow de Google. Fue desarrollado

y es mantenido por François Chollet, ingeniero de Google. Keras contiene varias implementaciones de los bloques constructivos de las redes neuronales como capas, funciones objetivo, funciones de activación, optimizadores matemáticos,etc...

- **Scipy:** Al igual que Keras, Scipy es una librería de código abierto para Python. Se compone principalmente de herramientas, implementaciones y algoritmos matemáticos. Scipy contiene módulos orientados al procesamiento de imágenes y señales, integración, álgebra lineal, computación evolutiva,etc,... Dentro de Scipy haremos uso de un algoritmo necesario para nuestro proyecto en la generación de “adversarial examples” denominado “Evolución diferencial”.

El investigador (Kondratyuk, 2019) en su implementación del ataque a un píxel hace uso del algoritmo de evolución diferencial de Scipy, con una ligera modificación que él mismo aplicó, para una ejecución del algoritmo más rápida y eficiente. Estará será la implementación del algoritmo de evolución diferencial que se usará, debido a que las diversas pruebas con otras implementaciones no han generado una perturbación de forma adecuada y eficiente.

4.2– Conjunto de datos

Las imágenes que se van a utilizar para el entrenamiento de la red y posterior realización del ataque se han obtenido del conjunto Alemán de imágenes de señales de tráfico, (Stallkamp, Schlipsing, Salmen, y Igel, 2011). En la figura 4.1 podemos apreciar ejemplos de imágenes que se encuentran dentro del conjunto de datos.



Figura 4.1: Imágenes pertenecientes al conjunto de datos a usar.

La motivación para utilizar este conjunto de imágenes para su posterior búsqueda de “adversarial examples” surge de la importancia, en el reciente campo de la seguridad de las redes neuronales, de intentar conocer cómo de resistentes son ante ataques de este tipo. Hasta ahora no se han generado “adversarial examples” para este conjunto de imágenes, lo que lo ha convertido en un excelente candidato para explorar esta técnica de ataque en nuestro proyecto. Se trata de un conjunto compuesto por 43 clases de un total de más de 35.000 imágenes con un tamaño de

48 x 48 píxeles en color, dividido en conjunto de entrenamiento y conjunto de pruebas. Podemos observar un ejemplo de las imágenes que aparecen en el conjunto de datos en la figura 4.2.



Figura 4.2: Señales del conjunto de imágenes

Cada imagen, tanto en el conjunto de entrenamiento como en el de prueba, cuenta con un valor entero que identifica la clase a la que pertenece (por ejemplo, el valor 2 identifica a Rotonda). La red neuronal durante su entrenamiento jamás puede ser entrenada con imágenes del conjunto de prueba si se esperan obtener resultados no sesgados.

Por su facilidad de uso, las imágenes se han almacenado en un formato reconocible por el lenguaje de programación con el que se ha desarrollado el proyecto (Python) el cual se conoce como Hierarchical Data Format 5 (.h5), esto facilita la portabilidad del conjunto de entrenamiento y pruebas, ya que permite almacenar las imágenes y el valor de la categoría perteneciente de forma que puedan ser cargados rápidamente en un entorno de programación Python. En la figura 4.3 podemos observar una imagen cargada en el cuaderno Jupyter del conjunto de imágenes que estamos utilizando.

Example of test image after processing

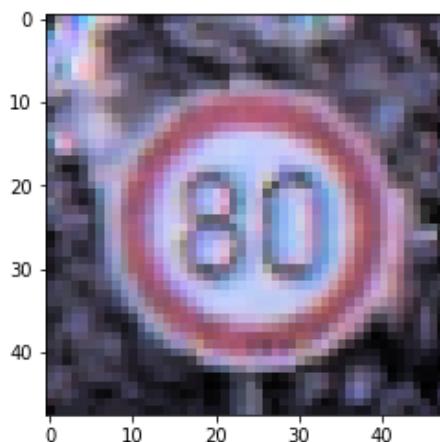


Figura 4.3: Señal cargada satisfactoriamente en el cuaderno Jupyter de trabajo

La motivación a usar este formato es la de no tener que procesar las imágenes una por una cada vez que se utilice el “cuaderno” y se ofrezca la opción de cargar las imágenes ya procesadas o procesarlas por uno mismo.

4.3– Diseño del modelo neuronal

El sistema diseñado se encargará de clasificar imágenes estará formado por una red neuronal compuesta de dos partes:

- 1. Capas de convolución como generador de mapa de características:** Las capas de convolución se encargarán de aplicar una serie de filtros a las imágenes de entrada de la red de forma que reduzcan su tamaño espacial, a la vez que mantengan una serie de características fundamentales de cada imagen, esto es algo conocido como mapa de características. Ayudará a la capa de clasificación debido a que el proceso de convolución mantiene características de similitud entre imágenes como podrían ser los bordes, rectángulos o patrones que componen la imagen. El proceso de convolución se ha tratado en profundidad en la sección 2.5.
- 2. Capas de perceptrón completamente conectadas como clasificador:** El volumen producido por la red de convolución es vectorizado para poder introducirlo en la red de perceptrones, este proceso es conocido como Flatten y se puede observar en el cuadro 4.1. Este vector de valores reales alimentarán a la red de perceptrones para producir una salida mediante neuronas de tipo Softmax, como se vio en la sección 2.2.2, éstas se tratan de neuronas para clasificación con múltiples categorías obteniendo la probabilidad de la pertenencia a cada posible categoría, siendo en nuestro proyecto la categoría un valor numérico entero que indica la clase a la que pertenece la imagen.

En la figura 4.4, se puede observar con mejor detalle las dos partes que compone la estructura de nuestra red neuronal.

La arquitectura de la red neuronal utilizada para nuestro proyecto no se basa en ninguna de las utilizadas en el artículo original del ataque a un píxel, (Su y cols., 2017), debido a que se utilizan modelos ya pre-establecidos y el objetivo, en parte, de este proyecto es el desarrollo de un red neuronal de convolución para la clasificación de señales de tráfico.

4.4– Implementación y resultados de la red neuronal

Las redes neuronales de convolución aprovechan que comúnmente las entradas a la red son imágenes. Es por ello que la arquitectura de las redes de convolución están orientadas a este tipo de datos. En particular, y a diferencia de las redes de perceptrón, las capas de las redes de convolución tiene sus neuronas organizadas en tres dimensiones: altura, ancho y profundidad (profundidad se refiere a la capa en particular y no a la profundidad de la red al completo, en concreto, la profundidad de una capa se identifica con el número de filtros a aplicar en ese misma capa).

Por ejemplo, las imágenes de entrada utilizadas en nuestro proyecto constituyen un volumen de entrada con dimensiones 48x48x3 (altura, ancho y profundidad respectivamente, siendo 3 el número de canales (RGB) de la imagen de entrada). Como podremos ver a continuación, y a diferencia de las redes de perceptrón, las neuronas de una capa no se encuentran completamente conectadas con las de la capa anterior, sino con una pequeña región de las neuronas de la capa anterior. Además, la capa de salida será un vector de 43 valores debido al proceso de reducción espacial producido por las capas de convolución y por el posterior procesamiento de las capas de perceptrón, generando un vector de salida con las probabilidades de pertenencia a cada categoría posible.

Como se ha descrito anteriormente, las redes de convolución están compuestas por una secuencia de capas, estas capas transforman un volumen de entrada en otro volumen de salida. Para

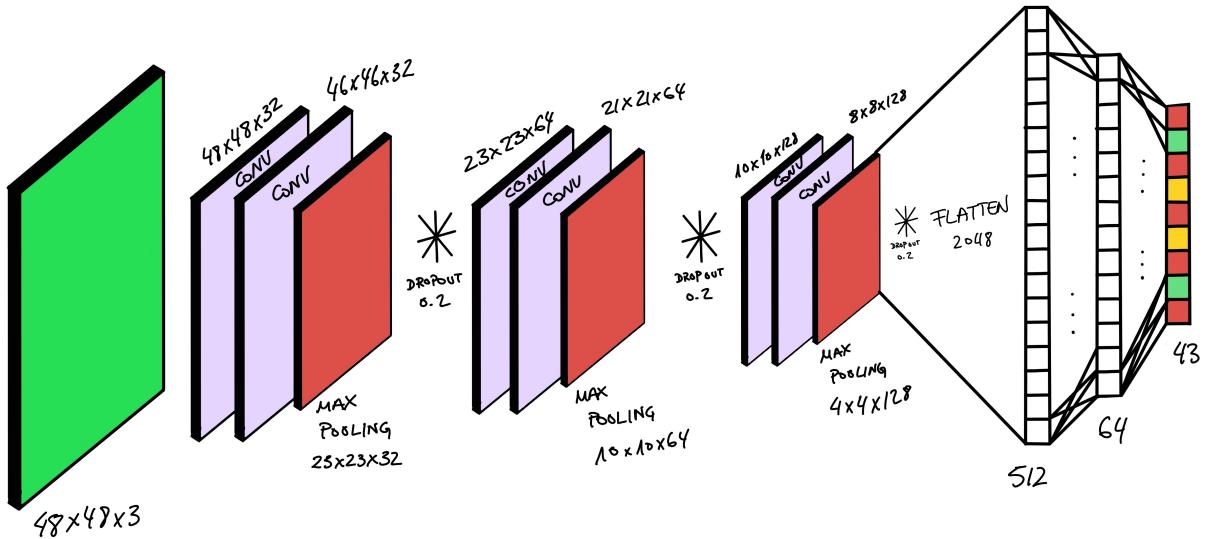


Figura 4.4: Modelo neuronal desarrollado para el proyecto.

su creación se usan tres tipos de capas principalmente: capas de convolución, capas de Pooling o agrupación, y capas de perceptrones completamente conectadas con función de activación Softmax en la capa de salida.

- **Entrada [48x48x3]:** Almacena los valores originales de la imagen de entrada de altura 48 píxeles, ancho 48 píxeles y con tres canales RGB.
- **Convolución:** Se ocupa de calcular la salida de las neuronas que se encuentran localmente conectadas con otras. El producto escalar entre los pesos del filtro de la capa de convolución y la región de la imagen de entrada producirán el volumen de salida de la capa. En nuestro caso, en la primera convolución utilizamos 32 filtros produciendo un volumen de salida tras la primera capa con dimensiones $[48 \times 48 \times 32]$. El proceso de convolución fue estudiado en la sección 2.5.
- **Activación:** Esta capa aplicará una función de activación elemento a elemento en la red. En particular usaremos la función RELU como ya vimos en la sección 2.2.1. Está aplicará a cada valor la siguiente función

$$f(x) = \max(0, x). \quad (4.1)$$

Esta acción no produce modificación alguna en el volumen, $[48 \times 48 \times 32]$.

- **Pooling o agrupación:** Se aplicará una operación de remuestreo a lo largo de las dimensiones espaciales alto y ancho. Esta operación reduce las dimensiones del volumen. Por ejemplo, la salida tras una aplicación de pooling puede ser un volumen de dimensiones $[24 \times 24 \times 32]$. Esta capa se ha explicado en detalle en la sección 2.5.1.

- **Capas de perceptrones completamente conectados:** Trata de calcular las probabilidades de pertenencia a cada categoría posible para la imagen de entrada. Su salida se corresponde con un vector de 43 valores reales posibles, donde cada valor es la probabilidad de pertenencia a cada categoría. Las neuronas de una capa, como su propio nombre indica, se mantienen completamente conectadas con todas las neuronas de la capa anterior. En la figura 4.5 podemos observar las capas descritas anteriormente así como su distribución en el caso del modelo neuronal de nuestro proyecto.

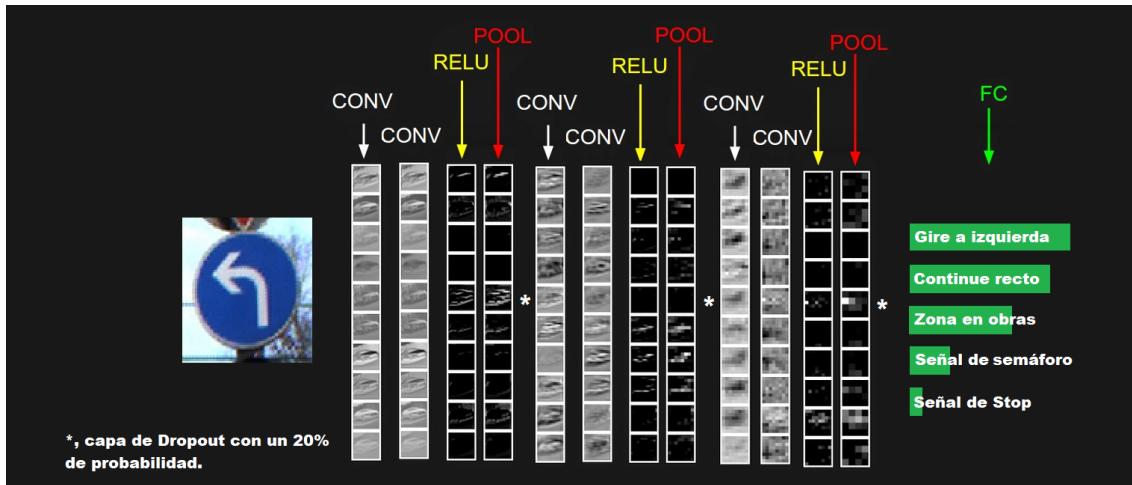


Figura 4.5: Descripción del procesamiento seguido por la red neuronal diseñada para nuestro proyecto.

De esta forma, las redes de convolución, mediante una serie de transformaciones aplicadas a la imagen de entrada, consiguen producir una clasificación de pertenencia por categorías. Es importante indicar que algunas capas contienen parámetros y otras no, en particular, las capas de activación y pooling aplican una función preestablecida. El resto de parámetros que sea posible entrenar, serán entrenados y actualizados mediante descenso por gradiente vistos en la sección 2.3.

La red neuronal final diseñada para la tarea de clasificación de señales de tráfico ha sido la que aparece en el cuadro 4.1. Tras entrenar a la red neuronal se procedió a estudiar su rendimiento con respecto al conjunto de imágenes de prueba. Esto consiste en que la red realice su predicción acerca de la categoría de pertenencia ante el conjunto imágenes de prueba, las cuales nunca ha visto, y luego comparar esas predicciones con las categorías reales de la imágenes, a fin de conocer cómo de bien está clasificando nuestra red ante nuevos ejemplos. El resultado del entrenamiento de la red utilizada fue del **97.57 % de acierto**.

4.4.1. Implementación y resultados del ataque a un píxel

La implementación del ataque a un píxel está basada en el trabajo del investigador Dan Kondratyuk (Kondratyuk, 2019) para la generación de “adversarial examples” ante el conjunto de datos CIFAR-10. Se han modificado partes del código donde ha sido necesario, aparecen debidamente señaladas en el cuaderno de Github del proyecto, (de los Santos García, 2019). Estas modificaciones en general son debidas a que el conjunto de imágenes CIFAR-10 almacena los valores RGB en formato [0-255] y el conjunto de las imágenes usadas para nuestro proyecto lo hacen en formato [0-1]. Además surgieron otros problemas como por ejemplo que no era posible realizar ataques con más de un pixel, los cuales fueron solucionados para nuestra implementación.

Para comenzar, realizamos la implementación de la función de perturbación que se encargue de modificar la imagen original dado un vector de perturbaciones X_s . Esta función se encarga de modificar los valores RGB r_i, g_i y b_i , en las posición x_i e y_i .

```

conv2d layer(kernel=3, stride=1, padding=same, profundidad=32, activacion=relu)
conv2d layer(kernel=3, stride=1, padding=valid, profundidad=32, activacion=relu)
    max pooling layer(kernel=2, stride=2)
    dropout(0.2)
conv2d layer(kernel=3, stride=1, padding=same, profundidad=64, activacion=relu)
conv2d layer(kernel=3, stride=1, padding=valid, profundidad=64, activacion=relu)
    max pooling layer(kernel=2, stride=2)
    dropout(0.2)
conv2d layer(kernel=3, stride=1, padding=same, profundidad=128, activacion=relu)
conv2d layer(kernel=3, stride=1, padding=valid, profundidad=128, activacion=relu)
    max pooling layer(kernel=2, stride=2)
    dropout(0.2)
    flatten layer
    fully connected(size=512)
    dropout(0.5)
    fully connected (size=64)
    dropout(0.5)
    softmax classifier

```

Cuadro 4.1: Red diseñada para el proyecto

En la figura 4.6, se observa primero que la imagen original perteneciente al conjunto de pruebas y segundo, la modificación en esa misma imagen con un vector de perturbación de prueba

$$X = (16, 16, 1, 1, 0), \quad (4.2)$$

siendo 1 equivalente al valor 255 en la escala RGB.

Fue necesario implementar una función que indique el éxito del ataque ante un vector de perturbaciones, así como el resultado probabilístico de pertenencia de la imagen a su categoría original. En la figura 4.7, se observa el resultado, en el cuaderno del proyecto, de realizar una perturbación a una imagen con un vector de perturbaciones de prueba con valores

$$X = (30, 23, 1, 1, 0). \quad (4.3)$$

En esta misma figura, podemos observar como la predicción original de la red se ha modificado a otra categoría por lo que consideramos el ataque como exitoso.

El algoritmo de evolución diferencial usado es propio de la librería Scipy, pero con ligeras modificaciones específicas para este problema, realizadas por el investigador Dan Kondratyuk (Kondratyuk, 2019).

Con las funciones implementadas para nuestro proyecto y el algoritmo de evolución diferencial usado, podemos asegurar la funcionalidad de nuestra implementación realizando ataques con el número de píxeles d arbitrario.

A continuación se listan las funciones implementadas en el cuaderno Jupyter y las modificaciones aplicadas con respecto a la implementación de Dan Kondratyuk (Kondratyuk, 2019):

1. `plot_image`

- **Modificación:** Nueva función
- **Descripción:** Mostrar imagen

2. `perturb_image`

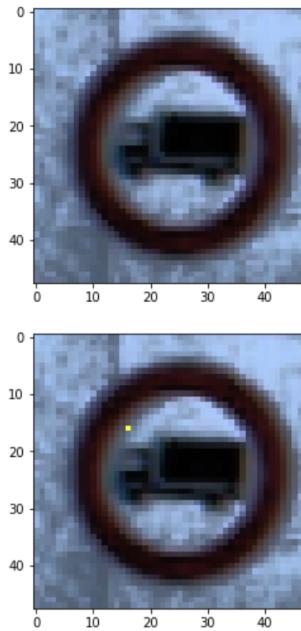


Figura 4.6: En la imagen superior podemos observar la imagen sin modificar, en la inferior, la imagen con el vector de perturbación X de la ecuación 4.2, aplicado.

- **Modificación:** Tomada de Dan con ligeras modificaciones
- **Descripción:** aplicar el vector de perturbación X

3. predict_classes

- **Modificación:** Basada en la implementación de Dan con una ligera modificación.
- **Descripción:** Devolver la probabilidad de pertenencia proporcionada por la red para la categoría original de la imagen.

4. attack_success

- **Modificación:** Ninguna
- **Descripción:** Determina si el ataque ha sido exitoso o no.

5. attack

- **Modificación:** La implementación de Scipy esta tomada de Dan, se han probado otras implementaciones ninguna dando como resultado “adversarial examples” con la modificación de píxeles deseados.
- **Descripción:** Función principal para lanzar el ataque con la implementación de Scipy. Surgió un problema en el que no era posible el conjunto de imágenes seleccionado para nuestro problema y la implementación de Dan para su conjunto de imágenes, realizar ataques adversarios con más de un píxel. Modifique el código para que permitiese realizar ataques con un número arbitrario de píxeles.

6. attack_all

- **Modificación:** Nueva.
- **Descripción:** Permite probar con un subconjunto de las imágenes, cuantas de ellas son vulnerables a ataques adversarios con la cantidad de píxeles deseados a modificar.

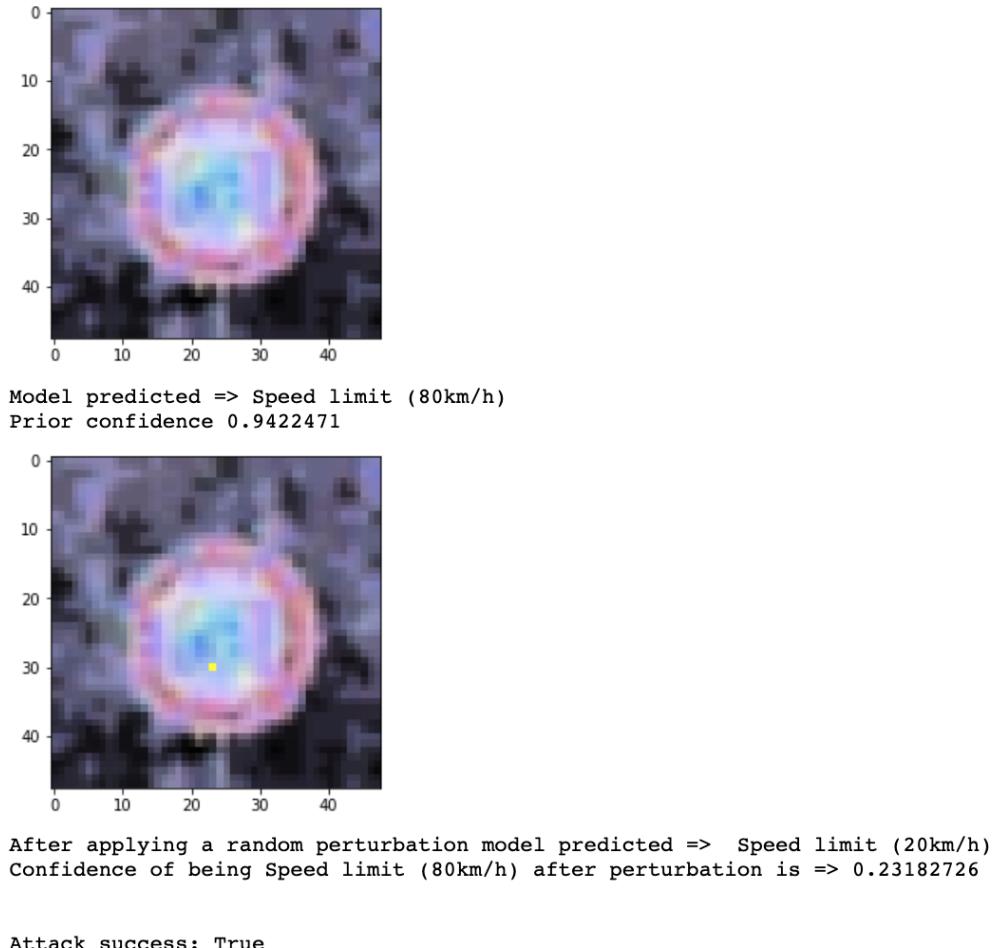
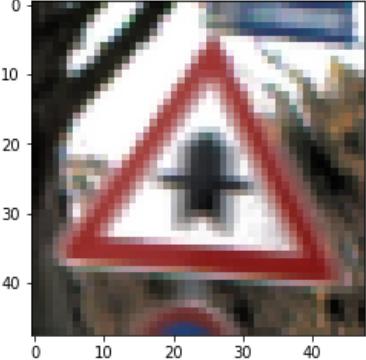
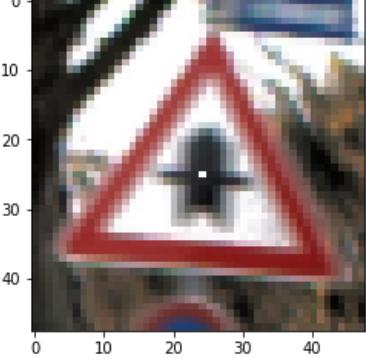


Figura 4.7: Resultado de aplicar el vector X , de la ecuación 4.3, a la imagen, modificando así la predicción original de la red, siendo esta límite de 80 Km/h con un 94.22 % de confianza. En la imagen inferior podemos observar como la red tras la perturbación añadida, ha modificado la clasificación original de límite de 80 Km/h por límite de 20 Km/h. Tras la perturbación la confianza de la red en ser una señal de límite 80 Km/h ha disminuido hasta el 23.31 %.

En la figura 4.8, se aprecia un ataque con valor $d = 1$ cuyo resultado no es satisfactorio, mientras que, en la figura 4.9, estamos realizando un ataque con valor $d = 3$. El resultado de este ataque en cambio fue satisfactorio y podemos observar el adversarial example encontrado.

```

Confidence: 0.99998486
Confidence: 0.99998415
-----
Attack result [25.82867132 24.32586847 0.99616478 0.9994214 0.99612964]


Model predicted => Right-of-way at the next intersection
Prior confidence 1.0
-----

After applying a random perturbation model predicted => Right-of-way at the next intersection
Confidence of being Right-of-way at the next intersection after perturbation is => 0.99998415
-----
Attack success: False

```

Figura 4.8: Resultado de realizar un ataque adversario sin éxito. El algoritmo de evolución diferencial al intentar ir generando mejores vectores de perturbación que cambien la categoría original de la imagen, no han conseguido encontrar ninguno que den lugar a “adversarial example”.

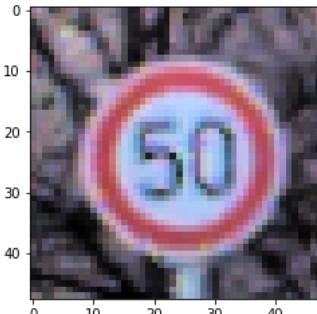
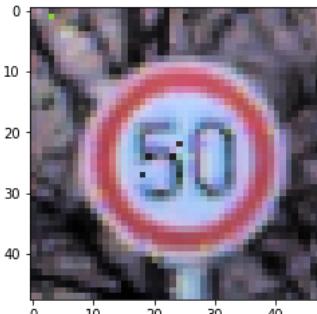
4.4.2. Experimentación

El ratio de éxito se muestra en el cuadro 5.2. Este ratio de éxito es obtenido en base un subconjunto de 500 imágenes arbitrarias del conjunto de prueba. Estas imágenes son sometidas al proceso de ataque con valores $d = [1, 3, 5]$.

Definimos el éxito de un ataque al hecho de encontrar una perturbación \mathbf{X} tal que modifique la categoría de pertenencia original de la imagen a otra. El ratio de éxito proviene de calcular que proporción de las 500 imágenes usadas del conjunto de prueba son vulnerables a ataques adversarios. Se observa que al aumentar el número de píxeles, como es de esperar, la probabilidad de encontrar adversarial examples aumenta. Se ha podido comparar con los resultados del artículo

```

Confidence: 0.91416615
Confidence: 0.7720716
Confidence: 0.06599495
-----
Attack result [ 1.2040326  3.05031972  0.53710248  0.83227669  0.18316248 22.24478394
24.92355355  0.25778937  0.17179703  0.17828779 27.76766172 18.11541165
0.12157127  0.17201998  0.26824693]


Model predicted => Speed limit (50km/h)
Prior confidence 1.0
-----

After applying a random perturbation model predicted => Speed limit (80km/h)
Confidence of being Speed limit (50km/h) after perturbation is => 0.06599495
-----
Attack success: True

```

Figura 4.9: Resultado de realizar un ataque adversario a una imagen con valor $d = 3$, obteniendo un resultado satisfactorio. El algoritmo de evolución diferencial fue capaz de generar un vector de perturbación que diese lugar a una gran disminución en la probabilidad de pertenencia de la imagen a su categoría original, dando lugar a un “adversarial example”. ”Attack result” se corresponde con el vector de perturbación X.

	1 píxel	2 píxeles	3 píxeles
Adversarial examples encontrados	51	132	134
Ratio de éxito	10.2 %	26.4 %	26.8 %

Cuadro 4.2: Resultados de la experimentación, identificando de la muestra de 500 imágenes cuantas dieron como resultado del ataque un adversarial example.

original de este proyecto, (Su y cols., 2017), en él los investigadores, hacen uso del conjunto de imágenes CIFAR-10 (Krizhevsky, Nair, y Hinton, s.f.) el cual utiliza unas imágenes de menor tamaño que las usadas en nuestro proyecto además de usar redes de convolución que están a la vanguardia en la visión por computador. En nuestro proyecto se han obtenidos unos ratios de éxitos menores indicando una alta dificultad para la generación de “adversarial examples” con nuestro modelo. Esto puede estar motivado por el alto número de categorías posibles y el bajo subconjunto de la población de prueba a usar. Un concepto existente en el mundo de las redes neuronales es la robustez, se dice que un sistema es robusto si es difícil encontrar en él ataques adversarios. Podremos decir con un ratio del 10 % indicando que solo una de cada diez imágenes de nuestro conjunto es vulnerable a ataques adversarios, que contamos con un sistema robusto ante ataques de un píxel.

Podemos observar en la siguiente figura, 4.10, distintos “adversarial examples” generados con nuestro proyecto.



Figura 4.10: Se ilustran distintos “adversarial examples” encontrados por nuestro ataque. En todas las imágenes se ha modificado únicamente un píxel.

4.4.3. Futuro

Como futuras mejoras se plantea un estudio más extenso de los resultados de generación de adversarial examples. Por un lado realizando un estudio para conocer qué clases son más y menos susceptibles a perturbaciones adversarias, para así encontrar estructuras perceptibles en la imagen que puedan indicar su susceptibilidad ante los ataques.

Mediante el uso de mapas de calor, como ya fue usado para el artículo en el que se inspira este proyecto (Su y cols., 2017), se puede representar el número de ataques exitosos con respecto a la clasificación objetivo original de la imagen y la clasificación proporcionada por la red. Esto permite darnos cuenta de que ciertas imágenes de entrada pertenecientes por ejemplo a una categoría t_1 , son comúnmente transferidas a una categoría a una cierta categoría t_2 distinta con alta frecuencia. Esto nos puede indicar que existe cierta “proximidad” de una categoría t_1 a otra t_2 .

Como posible mejora adicional al proyecto, podemos considerar que, siendo uno de los objetivos de este proyecto limitar el número de perturbaciones en la imagen de entrada, de forma que las perturbaciones sean poco perceptibles en la imagen original, sería interesante considerar si es

possible encontrar “adversarial examples” con un límite de posibles perturbaciones d y cuya perturbación para un píxel en la posición x_i e y_i con color RGB r_i , g_i y b_i , cuyo vector de perturbaciones sea

$$\mathbf{X} = (x_i, y_i, r'_i, g'_i, b'_i), \quad (4.4)$$

donde r'_i , g'_i , b'_i se corresponde con los valores obtenidos por una función de distribución gaussiana para cada color, con media, μ , igual al valor r_i , g_i o b_i , y desviación típica, σ , uno. Lo que se pretende con esto es estudiar si es posible generar “adversarial examples” cuyos píxeles modificados sean “similares.^a los originales, permitiendo así que estos “adversarial examples” sean confundidos con las imágenes originales debido a su semejanza.

CAPÍTULO 5

Planificación temporal

Este capítulo contempla la evolución temporal que ha seguido el proyecto. Se realizó una división de tareas al inicio del proyecto a las que se le asignaron una fecha inicio y unas horas estimadas en su realización. Se recoge la desviación de las horas estimadas, así como la fecha de inicio final, fecha final y una sección de comentarios con respecto a opiniones que se hayan tenido en referencia a la tarea.

Tarea 1: Redacción del capítulo 1, introducción.

- Fecha inicio: 11-02-2019
- Fecha inicio (final): 11-02-2019
- Horas estimadas: 15 horas.
- Fecha fin: 16-02-2019
- Fecha fin (final): 16-02-2019
- Horas finales: 18 horas
- Comentarios: Aprender a adaptarme con el entorno de Latex.

Tarea 2: Redacción del capítulo 2, redes neuronales.

- Fecha inicio: 16-02-2019
- Fecha inicio (final): 17-02-2019
- Horas estimadas: 45 horas.
- Fecha fin: 13-03-2019
- Fecha fin (final): 15-03-2019
- Horas finales: 55 horas.
- Comentarios: La parte más compleja fue encontrar bibliografía que permitiese comprender retropropagación para la red de perceptrones de forma simple y rigurosa, y que además, se aproximase a la nomenclatura utilizada en nuestro proyecto.

Tarea 3: Selección y preparación de las imágenes del conjunto de entrenamiento y pruebas.

- Fecha inicio: 14-03-2019
- Fecha inicio (final): 16-03-2019
- Horas estimadas: 30 horas.
- Fecha fin: 02-04-2019
- Fecha fin (final): 31-03-2019
- Horas finales: 25 horas.
- Comentarios: El conjunto utilizado fue seleccionado ya que nunca se han realizado ataques adversarios al mismo. Además, ilustra la importancia en ciberseguridad de utilizar sistemas de visión por computador basados en redes neuronales. Su procesamiento fue sencillo, se utilizó un sistema de almacenamiento de las imágenes llamado H5 para poder compartirlas y cargarlas de forma sencilla.

Tarea 4: Creación de la red neuronal para el proyecto. Una red de convolución junto con una red de perceptrones usando Keras.

- Fecha inicio: 03-04-2019
- Fecha inicio (final): 02-04-2019
- Horas estimadas: 30 horas.
- Fecha fin: 13-04-2019
- Fecha fin (final): 13-04-2019
- Horas finales: 30 horas.
- Comentarios: Keras ofrece una interfaz sencilla con un ajuste de los hiperparámetros fue fácil obtener un buen resultado por la red neuronal después del entrenamiento.

Tarea 5: Construcción del ataque a un píxel para generar “adversarial examples”.

- Fecha inicio: 13-04-2019
- Fecha inicio (final): 13-04-2019
- Horas estimadas: 85 horas.
- Fecha fin: 08-05-2019
- Fecha fin (final): 13-05-2019
- Horas finales: 100 horas.
- Comentarios: Fue complejo encontrar una implementación que diese lugar a “adversarial examples” y que además fuesen rigurosas con los detalles propuestos por los investigadores del artículo en el que se basa el proyecto, (Su y cols., 2017).

Tarea 6: Finalizar el capítulo 2, acabar con la redacción de las redes neuronales de convolución

- Fecha inicio: 10-05-2019
- Fecha inicio (final): 14-05-2019
- Horas estimadas 15 horas.

- Fecha fin: 16-05-2019
- Fecha fin (final): 20-05-2019
- Horas finales: 15 horas.
- Comentarios: Resulto interesante encontrar imágenes que permitiesen comprender convocatoria de forma gráfica.

Tarea 7: Redacción del concepto de “adversarial example” y el ataque a un píxel, capítulo 3.

- Fecha inicio: 17-05-2019
- Fecha inicio (final): 21-05-2019
- Horas estimadas: 25 horas.
- Fecha fin: 28-05-2019
- Fecha fin (final): 30-05-2019
- Horas finales: 25 horas.
- Comentarios: Una vez finalizado este capítulo me fue sencillo poder explicar a otras personas de que trataba exactamente el artículo original de este proyecto, abarca una pequeña introducción al problema para la seguridad que involucran los “adversarial example”, además de tratar el problema principal para nuestra investigación.

Tarea 8: Redacción del capítulo 4, correspondiente a la implementación y experimentación del proyecto.

- Fecha inicio: 29-05-2019
- Fecha inicio (final): 01-06-2019
- Horas estimadas: 55 horas.
- Fecha fin: 15-06-2019
- Fecha fin (final): 19-06-2019
- Horas finales: 60 horas.
- Comentarios: Este capítulo condensa la información relacionada con el trabajo realizado, las imágenes intentan ser lo más precisas con la implementación.

Tarea 9: Finalización de la memoria del trabajo de fin de grado con la redacción del resumen, conclusiones y arreglo de mejoras.

- Fecha inicio: 17-06-2019
- Fecha inicio (final): 18-06-2019
- Horas estimadas: 15 horas.
- Fecha fin: 20-06-2019
- Fecha fin (final): 22-06-2019
- Horas finales: 15 horas.
- Comentarios: Correcciones relacionadas con errores de comprensión en algunas expresiones, figuras, orden de algunos textos y añadir información extra.

El total de horas invertidas en el proyecto fue de **354 horas**.

CAPÍTULO 6

Conclusiones

Con respecto a la parte de investigación tome la decisión de llevar a cabo este proyecto debido a ser novedoso a la vez que interesante y estar relacionado con mi campo de interés, inteligencia artificial. Me ha permitido poner en práctica tanto conocimientos de recopilación de información y síntesis, como de desarrollar habilidades técnicas. En general, la planificación se ha mantenido, a excepción de ligeros retrasos consecuencia de la dificultad en la implementación. Me siento satisfecho por poder haber llevado a cabo este proyecto y generar “adversarial examples” ante un conjunto de datos nunca antes atacado.

Desde la parte experimental, este proyecto nos ha permitido estudiar en profundidad los componentes de una red neuronal de perceptrones y de convolución, así como demostrar que es posible encontrar “adversarial examples” que generen una clasificación errónea en nuestra red neuronal. Como aspectos interesantes del proyecto, se demuestra que es posible encontrar “adversarial examples” de forma rápida y con el uso de pocos recursos. Siendo éste un ataque de caja negra, la única información imprescindible es el resultado de clasificación de la red. Además, este ataque es independiente del modelo neuronal y conjunto de imágenes usado. La forma de generar el ataque adversario para este problema conlleva una modificación mínima de píxeles en las imágenes de entrada de la red, método propuesto por (Su y cols., 2017). Esto provoca que parte de los “adversarial examples” generados sean prácticamente imperceptibles al ojo humano, considerándolo por tanto, un método ideal para estudiar la robustez de un sistema neuronal.

Referencias

- Balestrieri, R., y Baraniuk, R. G. (2017). *Deep neural networks*. Rice University.
- Cybenko, G. (1989). *Approximation by superpositions of a sigmoidal function*. Springer-Verlag New York Inc. Descargado de <https://pdfs.semanticscholar.org/05ce/b32839c26c8d2cb38d5529cf7720a68c3fab.pdf> (fecha de consulta: 16 de Febrero de 2019)
- de los Santos García, I. (2019). *Crafting adversarial examples with one pixel attack*. Descargado de <https://github.com/drexpp/Adversarial-examples-One-pixel-attack> (fecha de consulta: 17 de Junio de 2019)
- Evtimov, I., Eykholt, K., Fernandes, E., Kohno, T., Li, B., Prakash, A., ... Song, D. (2017). Robust physical-world attacks on machine learning models. *CoRR, abs/1707.08945*. Descargado de <http://arxiv.org/abs/1707.08945> (fecha de consulta: 15 de Junio de 2019)
- Goodfellow et al. (2013). *Maxout networks*. Département d'Informatique et de Recherche Opérationnelle, Université de Montréal. Descargado de <https://arxiv.org/pdf/1302.4389.pdf> (fecha de consulta: 07 de Junio de 2019)
- I-Ta Lee. (2017). *Notes on backpropagation with cross entropy*. Descargado de <https://doug919.github.io/notes-on-backpropagation-with-cross-entropy/> (fecha de consulta: 15 de Marzo de 2019)
- Karpathy, A. (2019a). *Commonly used activation functions*. Stanford.edu. Descargado de <http://cs231n.github.io/neural-networks-1/#actfun> (fecha de consulta: 8 de Marzo de 2019)
- Karpathy, A. (2019b). *Convolutional networks*. Stanford.edu. Descargado de <http://cs231n.github.io/convolutional-networks/> (fecha de consulta: 12 de Junio de 2019)
- Karpathy, A. (2019c). *Regularización*. Stanford.edu. Descargado de <http://cs231n.github.io/neural-networks-2/#reg> (fecha de consulta: 19 de Marzo de 2019)
- Kondratyuk, D. (2019). *Keras implementation of one pixel attack for fooling deep neural networks using differential evolution on cifar10 and imagenet*. Descargado de <https://github.com/Hyperparticle/one-pixel-attack-keras> (fecha de consulta: 16 de Junio de 2019)
- Krizhevsky, A., Nair, V., y Hinton, G. (s.f.). Cifar-10 (canadian institute for advanced research). Descargado de <http://www.cs.toronto.edu/~kriz/cifar.html> (fecha de consulta: 22 de Junio de 2019)
- Kurakin, A., Goodfellow, I. J., y Bengio, S. (2016). Adversarial examples in the physical world. *CoRR, abs/1607.02533*. Descargado de <http://arxiv.org/abs/1607.02533> (fecha de consulta: 15 de Junio de 2019)
- Liu, Y., Chen, X., Liu, C., y Song, D. (2016). Delving into transferable adversarial examples and black-box attacks. *CoRR, abs/1611.02770*. Descargado de <http://arxiv.org/abs/1611.02770> (fecha de consulta: 15 de Junio de 2019)
- Mesin, B. P. (2018). *Student notes: Convolutional neural networks*. Descargado de <https://indoml.com/2018/03/07/student-notes-convolutional>

- neural-networks-cnn-introduction/ (fecha de consulta: 11 de Junio de 2019)
- Nielsen, M. A. (2015). *A visual proof that neural nets can compute any function*. Determination Press. Descargado de <http://neuralnetworksanddeeplearning.com/chap4.html> (fecha de consulta: 17 de Febrero de 2019)
- Srivastava et al. (2014). *Dropout: A simple way to prevent neural networks from overfitting*. University of Toronto. Descargado de <http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf> (fecha de consulta: 19 de Marzo de 2019)
- Stallkamp, J., Schlipsing, M., Salmen, J., y Igel, C. (2011). *The German Traffic Sign Recognition Benchmark: A multi-class classification competition*. (fecha de consulta: 11 de Junio de 2019)
- Su, J., Vargas, D. V., y Sakurai, K. (2017). One pixel attack for fooling deep neural networks. *CoRR, abs/1710.08864*. Descargado de <http://arxiv.org/abs/1710.08864> (fecha de consulta: 16 de Junio de 2019)
- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., y Fergus, R. (2014). Intriguing properties of neural networks. En *International conference on learning representations*. Descargado de <http://arxiv.org/abs/1312.6199> (fecha de consulta: 18 de Junio de 2019)
- Tencent Keen Security Lab. (2019). *Experimental security research of tesla autopilot*. Descargado de <https://keenlab.tencent.com/en/2019/03/29/Tencent-Keen-Security-Lab-Experimental-Security-Research-of-Tesla-Autopilot/> (fecha de consulta: 14 de Junio de 2019)