Clasificador de Señales de Tránsito empleando Redes Neuronales Convolucionales

Trabajo Práctico Final

Reconocimiento de Patrones y Aprendizaje de Máquina

Universidad Nacional del Sur

Autor: Nicolás Salomón
DNI: 40.086.088
Año: 2021

UNS UNIVERSIDAD NACIONAL DEL SUR

Índice

Índice de Figuras

1. Introducción

El Aprendizaje Automático (del inglés, Machine Learning) se encuentra en pleno auge gracias a la gran cantidad de producción de datos por parte de distintas actividades y aplicaciones, el aumento de la potencia computacional y el desarrollo de mejores algoritmos para resolver múltiples problemas.

Actualmente se utiliza en cualquier lugar, desde la automatización de tareas cotidianas hasta actividades mucho más complejas, por ello, las industrias de todos los sectores intentan beneficiarse de él.

Según Arthur Samuel, pionero en el campo de la inteligencia artificial, los algoritmos de Aprendizaje Automático permiten que las computadoras aprendan de los datos e incluso se mejoren a sí mismas, sin estar programadas explícitamente.

El aprendizaje automático permite que las aplicaciones de software sean más precisas en la predicción de resultados siendo su premisa básica recibir datos de entrada y utilizar el análisis estadístico para predecir una salida actualizándose constantemente a medida que hay nuevos datos disponibles.

1.1. Motivación

El empleo de redes neuronales en el ámbito de la visión artificial trae consigo múltiples ventajas, entre las cuales podemos identificar:

- Aprendizaje Adaptativo: Capacidad de aprender a realizar tareas y tomar decisiones basadas en un entrenamiento o en una experiencia inicial.
- Auto-organización: Crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.
- Operación en tiempo real: Los cómputos pueden ser realizados en paralelo gracias al avance actual en la tecnología, brindando rapidez en la toma de decisiones.
- Fácil inserción dentro de la tecnología existente: Facilidad en la integración de chips especializados para redes neuronales en sistemas existentes, mejorando así su capacidad en ciertas tareas.

La implementación de redes neuronales para detectar y clasificar señales de tránsito representa un avance sumamente importante en materia de seguridad vial. Permitir que una computadora reconozca una señal de tránsito, analice su entorno y en base a ello tomar la decisión correcta permitirá reducir en gran medida los accidentes viales.

Actualmente son varias las empresas automotrices que se encuentran investigando y probando distintos sistemas de conducción autónoma para integrar a sus vehículos.

Una de las más importantes, y que presenta el mayor avance en esta área en la actualidad, es Tesla Motors, con el desarrollo de su sistema "Full Self-Driving", el cual busca crear una conducción totalmente autónoma, siempre bajo supervisión del conductor. En la Figura 1 podemos apreciar en sistema "Full Self-Driving" en su etapa de prueba.

Figura 1. Etapa de prueba del sistema "Full Self-Driving" de Tesla Motors.

Es sumamente importante destacar que la única empresa en la actualidad abocada en el desarrollo de sistemas de conducción autónoma empleando solamente cámaras de vídeo es Tesla Motors. El resto de empresas hace uso de dispositivos LIDAR, los cuales aumentan considerablemente el costo final del proyecto, siendo un factor muy importante a considerar, más aún cuando se busca una implementación masiva de estos sistemas en la vida cotidiana.

Por todo esto, la investigación en sistemas de conducción autónoma que hagan uso únicamente del procesamiento de imágenes provenientes de cámaras, a través de redes neuronales, para tomar decisiones adecuadas y con el menor costo posible, representa una oportunidad muy novedosa, con un gran auge en la actualidad.

2. Objetivos

El presente trabajo se realizó teniendo en cuenta los siguientes objetivos.

2.1. Objetivo general

- Diseño e implementación de un clasificador de imágenes de señales de tránsito empleando redes neuronales.

2.2. Objetivos específicos

- Investigar formas de aumentar los datos del dataset ya existente.
- Analizar distintos algoritmos y arquitecturas para la clasificación de señales de tránsito.
- Comparar los rendimientos entre redes neuronales completamente conectadas y redes neuronales convolucionales.
- Comparar los rendimientos entre diferentes funciones de costo y optimizadores.
- Obtener y analizar distintas métricas para evaluar el modelo.
- Presentar por pantalla los resultados obtenidos.

3. Marco Teórico

A continuación, veremos los conceptos teóricos necesarios para la implementación del clasificador de señales de tránsito empleando redes neuronales convolucionales. Para ello se hará mención a diferentes tipos de redes neuronales, tanto su topología como métodos de entrenamiento, funciones de activación, algoritmos de optimización y dataset necesarios para lograr el resultado más óptimo posible.

Según [1], Machine Learning (desde ahora, ML) es la rama de la Inteligencia Artificial que otorga a los ordenadores y demás máquinas la capacidad de "aprender" por sí mismas sin la necesidad de que la función aprendida sea programada de antemano. Esto lo consigue mediante la aplicación de algoritmos diseñados para cada situación, de modo que, mediante unos datos de entrada, permitirá mediante dichos algoritmos generar unas predicciones de forma que la próxima vez que se encuentre con situaciones de datos similares, podrá predecir y/o reconocer características concretas de dichas situaciones debido al aprendizaje automático previo que ha realizado usando el algoritmo.

3.1. Métodos de Aprendizaje

Los algoritmos de ML pueden variar adaptándose a cada situación, sin embargo, es posible agruparlos en 4 grupos distintos de acuerdo a la forma en que "aprenden" a reconocer los patrones:

3.1.1. Aprendizaje supervisado

En el aprendizaje supervisado, el algoritmo de aprendizaje es alimentado por un set de datos, del inglés *dataset*, donde cada una de las observaciones tiene asociada una etiqueta. El objetivo del entrenamiento es lograr encontrar una función que lleve desde una observación a la etiqueta correcta, y el desafío es lograr que la función aprendida generalice para datos que no han sido vistos por el algoritmo durante el entrenamiento.

En cada paso del entrenamiento, el algoritmo recibe una observación y predice un resultado. Este resultado es comparado con la etiqueta asociada a la observación y de esta manera se puede evaluar el desempeño del modelo para corregirlo de ser necesario.

El aprendizaje supervisado es utilizado ampliamente en problemas de regresión y clasificación, y su modo de operación puede apreciarse en la Figura 2.

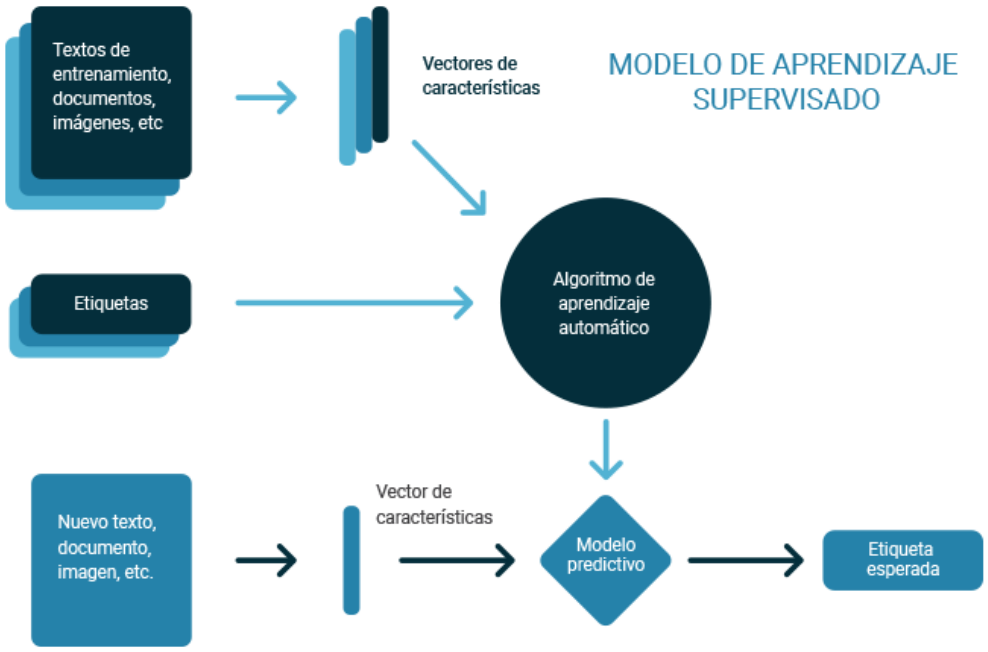

Figura 2. Modelo de funcionamiento de un algoritmo de aprendizaje supervisado

3.1.2. Aprendizaje no supervisado

En el aprendizaje no supervisado, el set de datos que alimenta al algoritmo de aprendizaje no contiene etiquetas, y el algoritmo debe encontrar estructuras o patrones en el set de datos, sin haberle dicho explícitamente como. Debido a que los datos no tienen etiquetas, la evaluación del algoritmo durante el entrenamiento es particularmente difícil. Aun así, es utilizado en gran medida debido a que en la realidad puede ser difícil conseguir muchos datos etiquetados y de calidad para resolver un determinado problema.

Los principales usos del aprendizaje no supervisado son para problemas de agrupamiento de datos (*Clustering*), detección de anomalías, autoencoders, recomendadores, entre otros. El modo de funcionamiento puede apreciarse en la Figura 3.

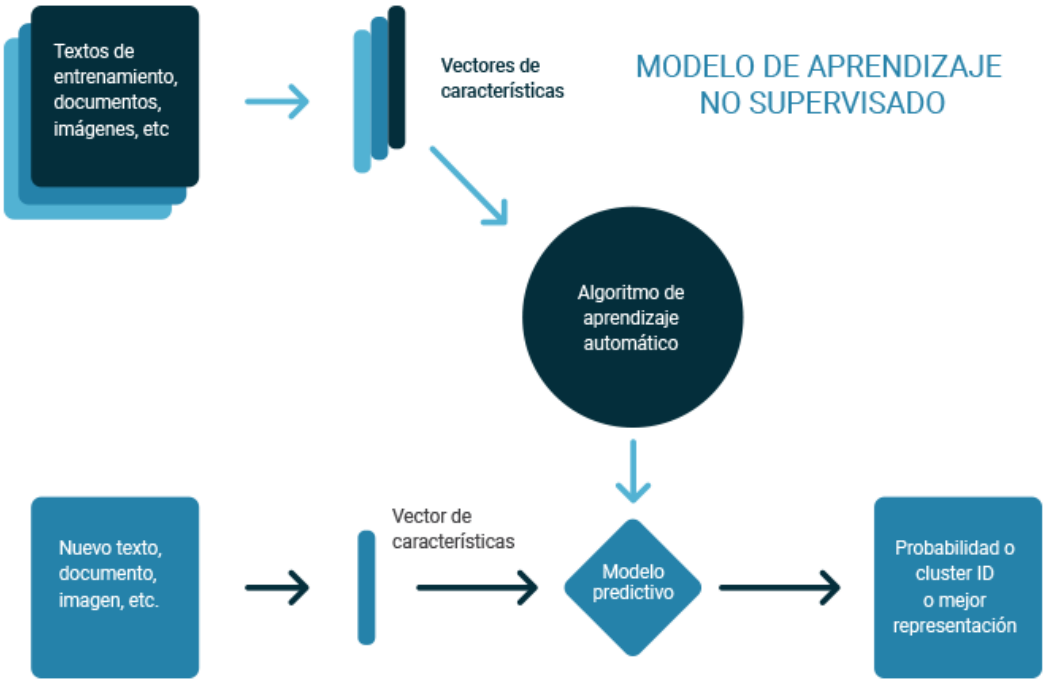

Figura 3. Modelo de funcionamiento de un algoritmo de aprendizaje no supervisado.

3.1.3. Aprendizaje por refuerzos

En el aprendizaje por refuerzo, el objetivo de los algoritmos es decidir qué acción tomar en un entorno con el que interactúan, con el fin de maximizar alguna noción de recompensa, a través de la cual el algoritmo mide su desempeño y corrige su próxima decisión.

Este tipo de aprendizaje tiene una gran cantidad de aplicaciones en la robótica y en los videojuegos (jugadores autónomos). Su funcionamiento se aprecia en la Figura 4.

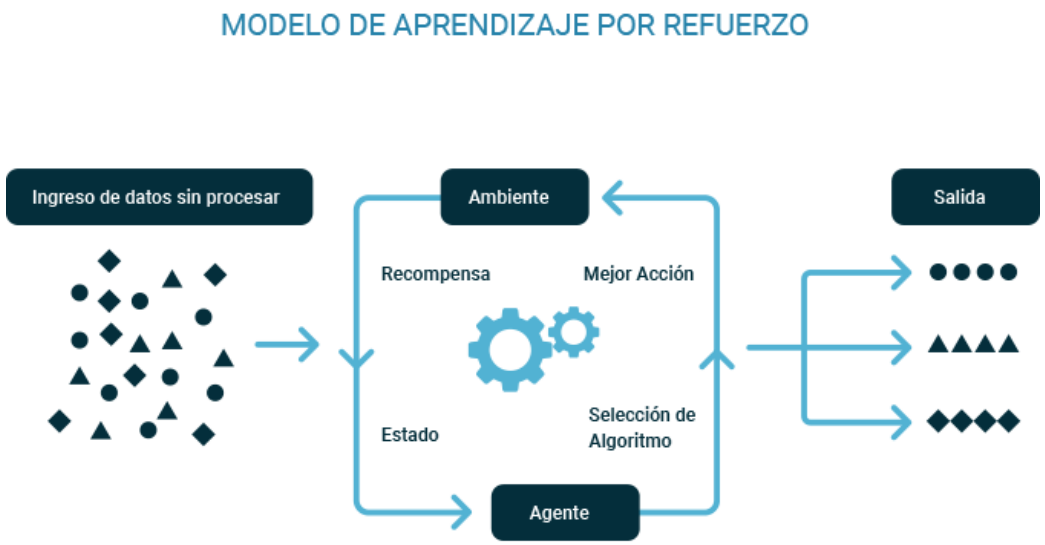

Figura 4. Modelo de funcionamiento de un algoritmo de aprendizaje por refuerzos.

3.2. Perceptrón, la neurona artificial

Al igual que una neurona forma el elemento básico del cerebro, una neurona artificial forma la estructura básica de una red neuronal artificial. Cada neurona recibe información de entrada, la procesa y genera una salida que o bien se envía a otras neuronas para su posterior procesamiento o es el resultado final.

Las partes de una neurona artificial son las siguientes:

- Un conjunto de entradas $x_1,\ldots,x_n$
- Los pesos sinápticos $w_1,\ldots,w_2$ correspondientes a cada entrada,
- Un sesgo b,
- Una función de suma Σ,
- Una función de activación σ,
- Una salida α

La Figura 5 muestra una neurona artificial con sus respectivas entradas y salida

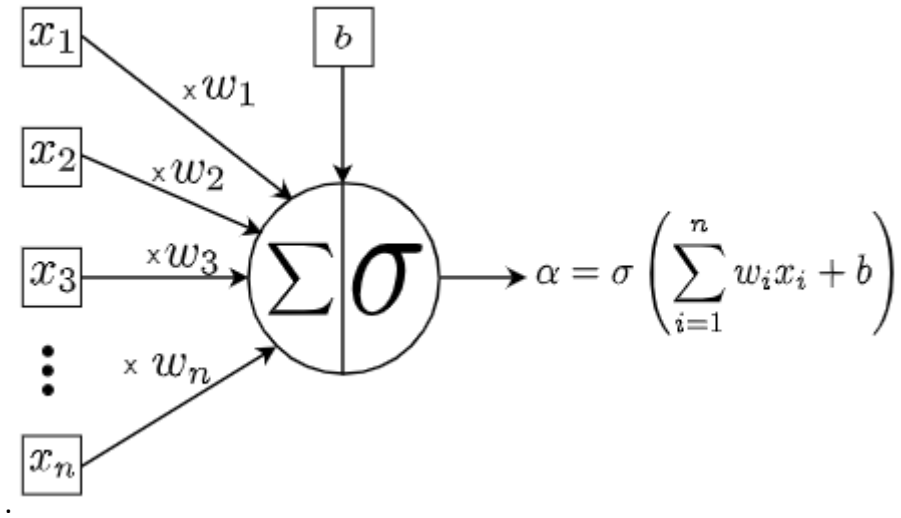

Figura 5. Neurona Artificial

Las entradas son el estímulo que la neurona artificial recibe del entorno que la rodea, y la salida es la respuesta a tal estímulo. Los pesos sinápticos y el sesgo de cada neurona son conocidos como los parámetros libres del modelo, debido a que cambian sus valores y se adaptan para realizar una tarea determinada.

La salida de una neurona está dada por la ecuación 1:

$$\alpha = \sigma(z) \tag{1}$$

Donde,

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2}$$

El perceptrón es la red neuronal más simple en Deep Learning, conformada por una sola neurona. Tal como se dice en [2], uno de sus usos más básicos es el de separar 2 clases, por lo que hace una clasificación de tipo binario, sacando como salida de dicha neurona el valor ‘0’ o ‘1’ según sea una clase u otra. Sin embargo, el perceptrón también se puede usar para distinguir entre varias clases, y que a la salida muestre una probabilidad de bondad de identificación de dicha clase, obteniendo un porcentaje de posibilidad de que sea o no dicha clase.

3.2.1. Pesos

Cada entrada a una neurona tiene un peso asociado. Cuando la entrada ingresa a la neurona, se multiplica por su respectivo peso. Los pesos se inicializan aleatoriamente y se actualizan durante el proceso de entrenamiento del modelo.

Durante el entrenamiento la red neuronal tiende a asignar un mayor peso a la entrada que considera más importante. Un peso de cero denota que la respectiva entrada en particular no aporta información útil.

3.2.2. Sesgo

El sesgo es un término constante que no depende de ninguna entrada, y que se agrega al resultado de la sumatoria de los pesos por las respectivas entradas.

Su finalidad es cambiar el rango de la entrada multiplicada por los pesos.

3.2.3. Función de activación

Luego de multiplicar las entradas por los pesos y sumar el sesgo, se le aplica a la salida una función no lineal, la cual se denomina función de activación. El objetivo de la función de activación es darle la capacidad a una red neuronal de aprender a encontrar relaciones no lineales sobre las entradas.

Funciones de activación más comúnmente usadas [1]:

- **Lineal**

  La función lineal pretende mantener constante el valor de la señal de entrada. Es decir, la salida de la neurona en cuestión le pasará a la siguiente neurona como valores de entrada los mismos datos que tenía la anterior. Es de las más sencillas, y se ilustra en la Figura 6. Un ejemplo de uso de esta función sería el de predicción de un valor de ventas, donde se necesita de una regresión lineal para que la red genere un valor único como salida.

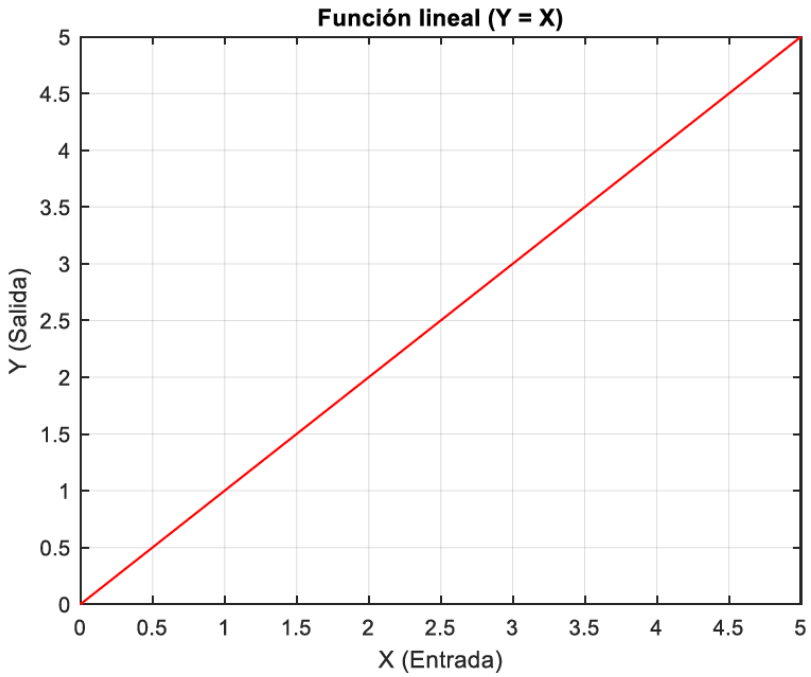

Figura 6. Función de activación Lineal

- **Escalón**

  Indica que el valor que se la pasa a la siguiente neurona será '1' si los datos de entrada superan un umbral concreto, o '0' en caso contrario. Es una función escalón básica, y se suele usar para clasificar directamente en 2 clases, o bien cuando las salidas son categóricas. Se ilustrará a continuación en la Figura 7 para el caso en que el umbral de entrada es el valor '0':

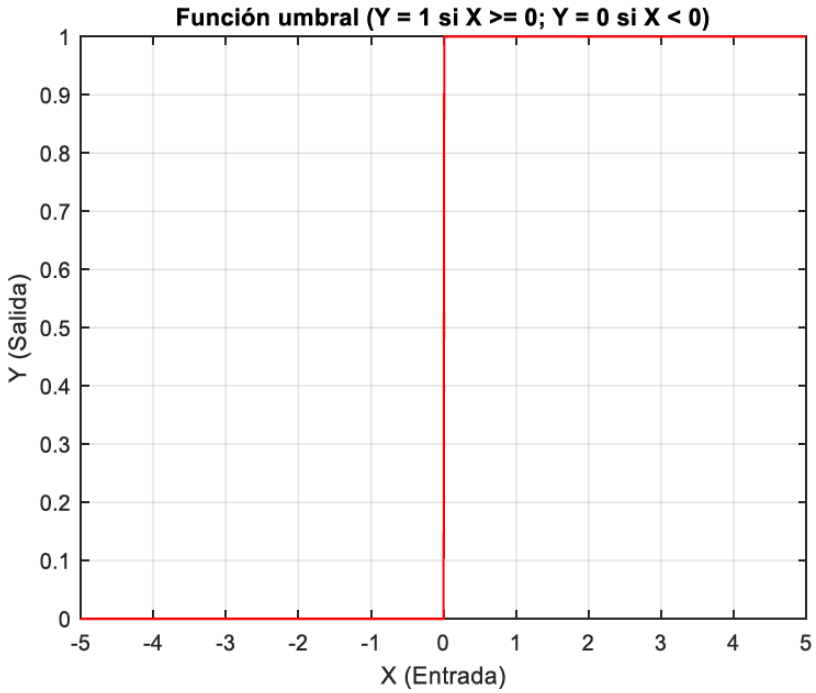

Figura 7. Función de activación Escalón

- **ReLU (unidades lineales rectificadas)**

  Es una de las más usadas en la actualidad ya que su uso ha resultado muy satisfactorio empíricamente. Lo que pretende es que la salida no cambie de valor hasta que la entrada supere un cierto umbral, pasando entonces como salida el mismo valor que la entrada. Así, podría haber una salida que estuviera a '0' mientras que la entrada cambia, y de repente cuando dicha entrada toma valores por encima de un umbral pasa a tener un comportamiento lineal. Se ilustra en la Figura 8.

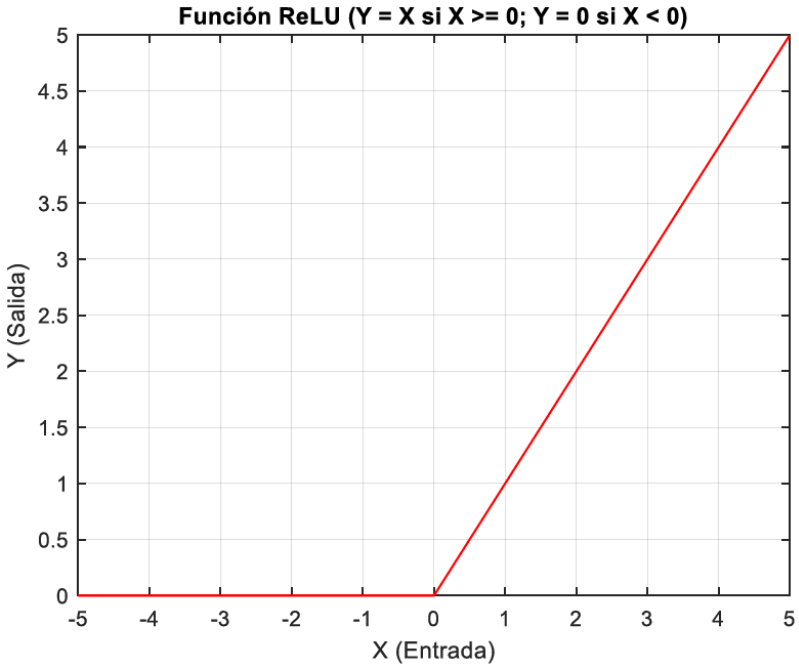

Figura 8. Función de activación ReLU.

- **Leaky-ReLU**

  Si se usan algoritmos de optimización basados en gradiente (serán explicados más adelante), puede surgir un problema a la hora de controlar el Dropout (conexión y desconexión aleatorias de neuronas en la fase de aprendizaje para introducir no linealidades en el entrenamiento), si es que se hace uso de esta técnica, ya que se puede dar el caso de que la función valga '0' y que la derivada de dicha función valga también '0', provocando la muerte (desconexión) de neuronas no deseadas.

  Por eso, hay una variante de la función ReLU, que en el tramo anterior al lineal introduce una pendiente que es prácticamente '0', pero sin llegar a serlo, evitando así este problema. Dicha variación se aprecia en la Figura 9 y es conocida como "Leaky ReLU".

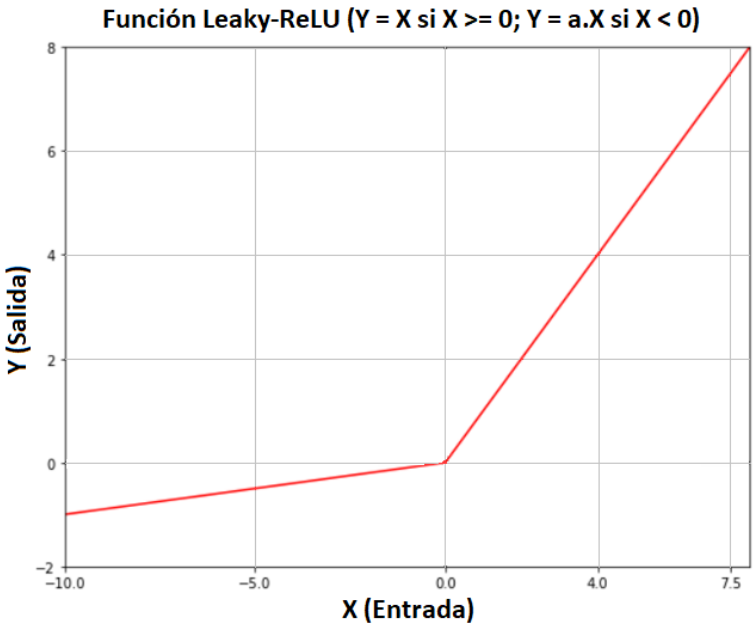

Figura 9. Función de activación Leaky-ReLU.

- **Sigmoide**

  Esta función permite que pequeñas variaciones en la entrada provoquen variaciones muy diferentes en la salida, siendo dichas variaciones poco notables en las zonas extremas de la función, pero muy grandes en comparación en la zona intermedia. En otras palabras, toma una entrada de valor real y la mapea a un rango de valores entre 0 y 1.
  La misma se ilustra en la Figura 10.

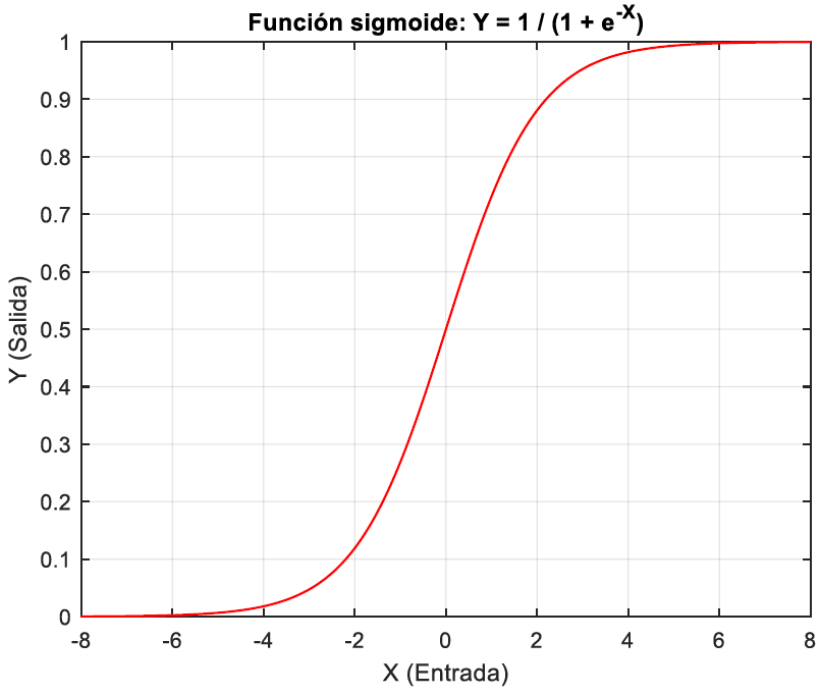

Figura 10. Función de activación Sigmoide.

- **Softmax**

  Esta función es bastante especial y normalmente irá en la última capa de las redes de clasificación con múltiples etiquetas. Es especial porque devuelve el porcentaje de todas las clases posibles, siempre que dichas clases sean mutuamente excluyentes, como un valor entre 0 y 1 según la red crea que los datos de entrada pertenecen a una clase u otra, por lo que se puede usar como capa de decisión de clasificación final. Por tanto, mejora la función sigmoide antes comentada, ya que Softmax introduce varios niveles de decisión, como puede observarse en la Figura 11.

  De este modo, si hay una manzana (clase 1), una pera (clase 2) y un plátano (clase 3), es posible que la función Softmax de la última capa devuelva, si la red está bien entrenada, valores, por ejemplo, 0.8 para la clase 1, 0.1 para la clase 2, y 0.1 para la clase 3 si lo que se ha introducido como dato de entrada era una manzana, correspondiente a la clase 1. Lógicamente, la suma de todos los porcentajes de salida debe dar el valor unidad.

  Se define según la ecuación 3 y un ejemplo de aplicación puede observarse en la Figura 11:

$$\sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3}$$

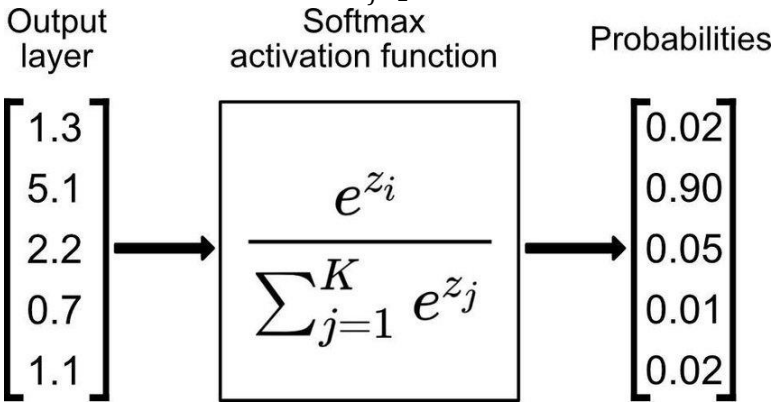

Figura 11. Resultado de aplicar la función Softmax.

3.3. Perceptrón multicapa

Una sola neurona tiene la capacidad de resolver solo problemas linealmente separables. Sin embargo, la mayoría de los casos que son tratados con redes neuronales no son linealmente separables. Por este motivo se crean pilas de neuronas llamadas capas. Una red perceptrón multicapa consiste en una capa de entrada que recibe los estímulos, una capa de salida que toma una decisión o realiza una predicción, y un número arbitrario de capas intermedias, llamadas capas ocultas. Cada neurona en una capa recibe información de todas las neuronas de la capa anterior, la

procesa, y produce una salida que será estímulo de todas las neuronas de la capa posterior.

A las capas de una red perceptrón multicapa se las suele llamar capas totalmente conectadas.

La Figura 12 representa un diagrama de arquitectura de un perceptrón multicapa de 3 capas.

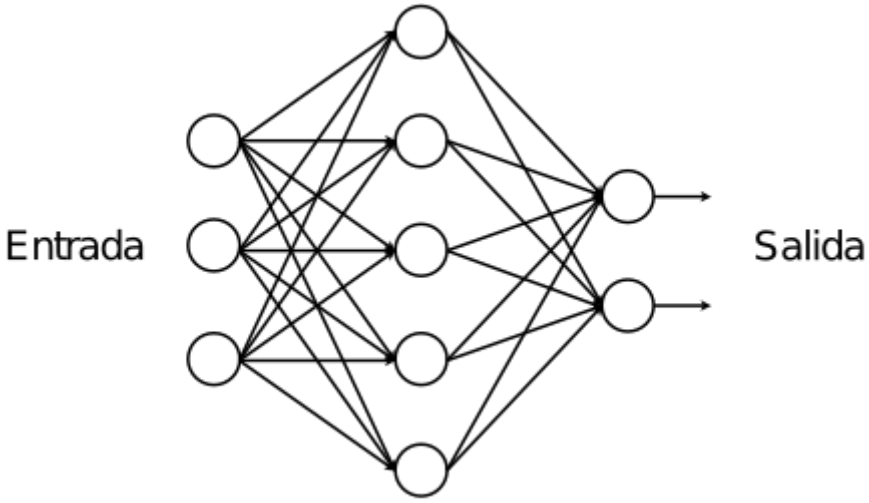

Figura 12. Perceptrón multicapa

Matemáticamente, la salida de una capa *l* se calcula a través de la ecuación 4. La cual entrega como resultado la suma de todas las entradas (pesos multiplicados por el valor de entrada sumado al sesgo) aplicado a la función de activación σ.

$$a^l = \sigma\left(\sum_j w_j^l a_j^{l-1} + b^l\right) \tag{4}$$

3.4. Redes neuronales convolucionales

Las redes neuronales convolucionales son redes que utilizan la operación de convolución en lugar de la multiplicación de matrices general en al menos una de sus capas, con el propósito de extraer características de una imagen.

3.4.1. Capas convolucionales

Cada capa convolucional está compuesta de un conjunto de filtros entrenables. Un filtro es una matriz de pesos que se multiplica con una parte de la imagen de entrada para generar una salida convolucionada. El tamaño del filtro es más pequeño que el tamaño de la imagen original.

Las capas convolucionales tienen pesos y sesgos compartidos, es decir que los pesos y el sesgo del filtro son los mismos para cada sección que este recorre sobre la imagen de entrada.

La salida en la coordenada *j,k* de un filtro de una capa convolucional se calcula de acuerdo a la ecuación 5:

$$a_{j,k} = \sigma\left(\sum_{l=0}^{n}\sum_{m=0}^{n} w_{l,m}x_{j+l,k+m} + b\right) \quad (5)$$

Una capa convolucional puede estar compuesta por uno o por varios filtros, por lo que puede tener una salida o un conjunto de ellas, a las cuales se las llama mapa de características (del inglés, *feature maps*).

En la Figura 13 se ilustra como se aplica la operación convolución a una entrada de 6x6 con un filtro de 3x3.

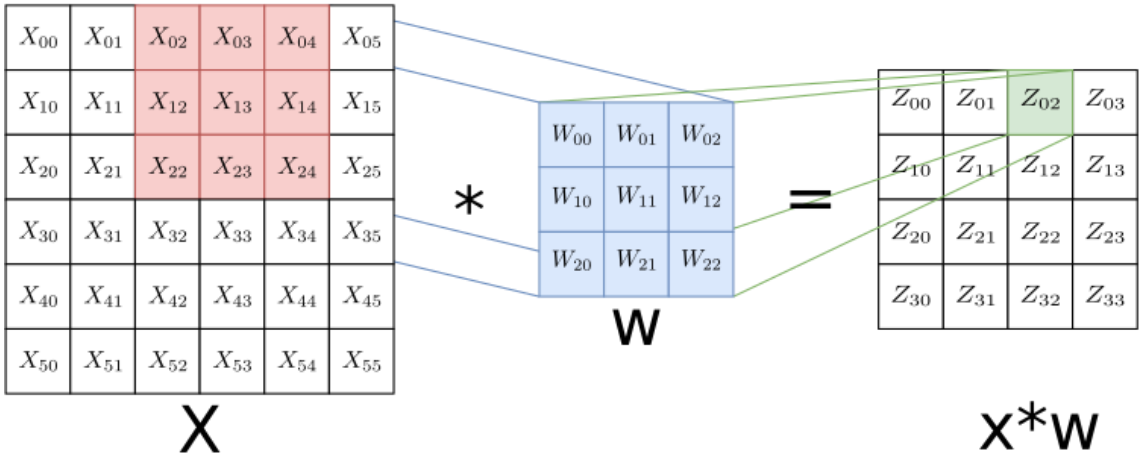

Figura 13. Convolución 2D

Las capas convolucionales aprovechan tres ideas importantes que ayudan a mejorar un sistema de aprendizaje automático: interacciones dispersas, uso compartido de parámetros y equivarianza.

- Interacciones dispersas significa que cada salida de la capa convolucional no depende de todas sus entradas, sino de un conjunto pequeño de ellas, a diferencia de las capas totalmente conectadas.
- Uso compartido de parámetros se refiere a que el valor del peso aplicado a una entrada está vinculado al valor del peso aplicado en otra parte de la entrada, es decir aprende un conjunto de parámetros en lugar de aprender parámetros separado para cada ubicación.
- Equivarianza a la traslación quiere decir que un desplazamiento en la entrada implica una salida igual con el mismo desplazamiento aplicado. Esta propiedad es resultado del uso compartido de parámetros. La convolución no es equivariante para otras transformaciones como la rotación o el escalado.

El resultado de aplicar dichas capas convolucionales puede apreciarse en la Figura 14, la cual muestra la salida de una capa convolucional de la red convolucional desarrollada en el presente trabajo, compuesta por 32 filtros. Se observa como existen

filtros que se centran en detectar los bordes de la señal de tránsito, mientras que otros prestan más atención al color de la señal y de su fondo.

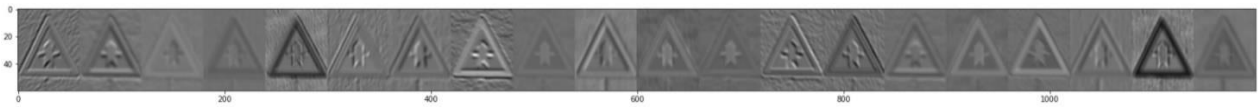

Figura 14. Salida de una capa convolucional de una red neuronal

3.4.2. Capas de Pooling

La capa de pooling reemplaza la salida de la red en una ubicación determinada con un resumen estadístico de las salidas cercanas. En detalle, una capa de pooling toma cada una de las salidas de la capa convolucional y genera una nueva salida condensada utilizando algún criterio particular para realizar dicha reducción (ver Figura 15). Los criterios más comúnmente aplicados son tomar el elemento máximo o realizar un promedio entre un conjunto de elementos [1].

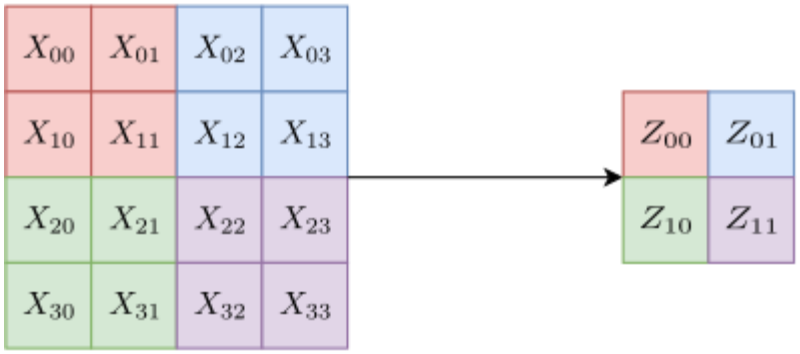

Figura 15. Operación de Pooling

En el ejemplo de la Figura 15, tomando el criterio de que el elemento resultante tomará el valor del máximo elemento de cada ventana, los resultados son los siguientes:

$$Z_{00} = \max(X_{00}, X_{01}, X_{10}, X_{11})$$

$$Z_{01} = \max(X_{02}, X_{03}, X_{12}, X_{13})$$

$$Z_{10} = \max(X_{20}, X_{21}, X_{30}, X_{31})$$

$$Z_{11} = \max(X_{22}, X_{23}, X_{32}, X_{33})$$

El pooling generalmente ayuda a que la salida tenga una cierta invarianza sobre cambios locales de traslación en la entrada, esto significa que, si desplazamos la entrada en una cantidad pequeña, los valores de la mayoría de las salidas luego del pooling no cambian considerablemente. La invarianza a la traslación local puede ser una propiedad útil si en el modelo se tiene más interés si una característica está presente a que la ubicación exacta de la característica.

Además, ayuda a reducir la cantidad de parámetros finales de la red neuronal convolucional, dado que con cada capa de pooling se reduce, al menos a la mitad, la dimensión de los datos de entrada a dicha red.

3.4.3. Normalización por lotes (Batch Normalization)

Con el avance en la infraestructura para admitir redes neuronales profundas, la investigación se desvía para aprovechar al máximo el hardware disponible al hacer que la red sea más amplia y profunda y mejorar la precisión. Un aspecto clave de las redes neuronales profundas que hace factible profundizar sin comprometer la velocidad de entrenamiento es la normalización por lotes (del inglés, Batch Normalization.

En la mayoría de los modelos de estado del arte actuales para el reconocimiento de objetos, hay una capa de normalización por lotes después de una capa de convolución dentro de la arquitectura de red. Pero, ¿qué tan importante es tener una normalización de lotes?

La normalización es una técnica de preparación de datos para garantizar que los datos sean consistentes y tengan una escala uniforme. En Imágenes, las técnicas de preparación de datos comúnmente utilizadas incluyen la normalización de píxeles y la normalización de imágenes.

- Normalización de píxeles: Esto se logra dividiendo cada píxel por 255 para que estén en un rango de [0,1].
- Normalización de imagen: esto altera el rango de intensidad de píxeles para que cada imagen en el conjunto de datos utilice un rango completo de valores de píxeles.

Las técnicas anteriores pueden ayudar a normalizar la imagen hasta que comience el entrenamiento. Una vez que convolucionamos sobre la imagen, no hay fuerza impulsora que restrinja el cambio de escala y se mantenga constante. Esto formula la necesidad de incluir otra técnica de normalización que pueda usarse después de cada operación de convolución. Ahí es cuando interviene la normalización por lotes. Ayuda a reducir el cambio de covariable interno.

En términos simples, un cambio de covariable se refiere a la distribución de las características que es diferente en diferentes partes del conjunto de datos. La misma diferencia cuando se observa dentro de las capas de la red debido a la distribución de activaciones se denomina cambio de covarianza interna. Usando normalización por lotes, agregamos un paso de normalización que corrige las medias y las variaciones de las entradas de capa, lo que ayuda a una convergencia más rápida y un flujo de gradiente mejorado a través de la red, al reducir la dependencia de los gradientes en la escala de los parámetros o de sus valores iniciales [3].

3.5. Entrenamiento

Existen distintas variables y funciones que influyen en el entrenamiento de la red neuronal, las cuales analizaremos a continuación. En necesario destacar que no existe una configuración única para resolver una problemática empleando redes neuronales convolucionales, sino que es necesario adatar el problema y evaluar distintas redes para obtener aquella que brinde un mejor resultado.

3.5.1. Dataset

Un set de datos (del inglés, *dataset*) en el ámbito del Aprendizaje Automático puede definirse como un conjunto de datos (imágenes, archivos .csv, .json, entre otros) que sirven de base para entrenar un algoritmo con el objetivo de que una máquina pueda tomar decisiones. Estas decisiones serán tomadas justamente de los datos provistos por el Dataset.

Por lo general, un set de datos se usa no solo con fines de entrenamiento de la red, sino que se divide en varias partes (ver Figura 15), para verificar qué tan bueno fue el entrenamiento del modelo y evaluar sus resultados a través de diferentes métricas (precision, recall, f1-score, matriz de confusión, entre otras).

Para este propósito, un set de datos generalmente se separa en datos de entrenamiento (para entrenar la red, dándole un conjunto de datos correctamente etiquetados, de esta manera la red "aprenderá" como debe de clasificar los datos recibidos), validación (para determinar que tan bien funciona la red) y prueba (para comprobar cuánto generaliza la red al finalizar el entrenamiento y la validación).

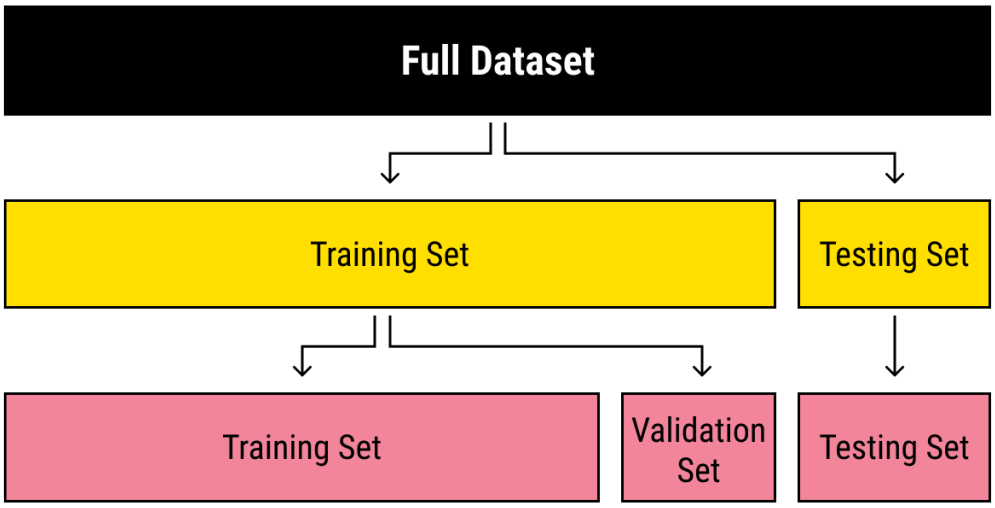

Figura 16. División de datos en un Dataset

3.5.2. Función de costo o de pérdida

Durante el proceso de entrenamiento se busca lograr que la salida de la red se aproxime lo máximo posible al valor real deseado para una entrada determinada. Para

cuantificar qué tan bien se está logrando este objetivo, se define una función de costo, también conocida como función de pérdida.

Si las predicciones se desvían demasiado de los resultados reales, la función de pérdida en ML arrojaría un número muy grande. Poco a poco, con la ayuda de alguna función de optimización, la función de pérdida aprende a reducir el error en la predicción.

No existe una función de pérdida para todos los algoritmos en ML. La elección de una función de pérdida para un problema específico, como el tipo de algoritmo, la facilidad de cálculo de las derivadas y, en cierta medida, el porcentaje de valores atípicos en el conjunto de datos son diversos.

En general, las funciones de pérdida pueden clasificarse en dos categorías principales dependiendo del tipo de tarea de aprendizaje con la que estamos tratando: Pérdidas por regresión (cuando se trata de predecir un valor continuo) y Pérdidas por clasificación (cuando se trata de predecir el resultado del conjunto de valores categóricos finitos).

- **Funciones de pérdida para regresión:**

  Error cuadrático medio (MSE): se mide como el promedio de la diferencia al cuadrado entre las predicciones y las observaciones reales. Solo se centra en la magnitud promedio del error, independientemente de su dirección.

  Sin embargo, debido a la que su término esta elevado al cuadrado, las predicciones que están muy lejos de los valores reales son penalizadas fuertemente en comparación con las predicciones menos desviadas.
  La fórmula matemática involucrada con esta función de perdida se aprecia en la ecuación 6, donde y_i es el valor real e $\hat{y}_i$ es el valor predicho, n es el número total de datos.

$$MSE = \frac{1}{n} * \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \qquad (6)$$

  Error Absoluto Medio (MAE): El error absoluto medio por otro lado, se mide como el promedio de la suma de las diferencias absolutas entre las predicciones y las observaciones reales.
  Al igual que MSE (error cuadrático medio), también mide la magnitud del error sin considerar su dirección.
  A diferencia de MSE (error cuadrático medio), MAE (error absoluto medio) necesita herramientas más complicadas, como la programación lineal para calcular los gradientes. Además, MAE es más robusto para los valores atípicos, ya que no utiliza el cuadrado.
  Esta función se define según la ecuación 7.

$$MAE = \frac{1}{n} * \sum_{i=1}^{n} |y_i - \hat{y}_i| \quad (6)$$

- **Funciones de pérdida para clasificación**

  Pérdida de entropía cruzada (Binary Cross Entropy Loss): La entropía es la medida de aleatoriedad en la información que se procesa, y la entropía cruzada es una medida de la diferencia de aleatoriedad entre dos variables aleatorias.

  Esta es la configuración más común para los problemas de clasificación. La pérdida de entropía cruzada aumenta a medida que la probabilidad prevista diverge de la etiqueta real, como puede apreciarse en la ecuación 7.

$$J = -\sum_{i=1}^{n} y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i) \quad (7)$$

  donde y_i es el valor real e $\hat{y}_i$ es el valor predicho, n es el número total de datos.

  Dado que la clasificación binaria significa que las clases toman 0 o 1, si y_i = 0, ese término deja de existir y si y_i = 1, el término (1-y_i) se convierte en 0.

  Pérdida de entropía cruzada categórica (Categorical Cross Entropy Loss): La pérdida de entropía cruzada categórica es esencialmente una pérdida de entropía cruzada binaria expandida a múltiples clases.

  Un requisito cuando se utiliza la función de pérdida de entropía cruzada categórica es que las etiquetas deben ser de la forma one-hot encoded, de esta manera, solo un elemento será distinto de cero ya que otros elementos en el vector se multiplicarían por cero.

  Sin embargo, existe una función adicional denominada **Pérdida de entropía cruzada categórica dispersa** que admite etiquetas numeradas, es decir sin tener una codificación one-hot.

3.5.3. Descenso del gradiente

El objetivo del algoritmo de entrenamiento es minimizar el costo en función de los pesos y los sesgos. En otras palabras, se busca encontrar un conjunto de pesos y sesgos que hagan que el costo sea lo más bajo posible. Esto se logra mediante un algoritmo conocido como descenso de gradiente [1].

La forma de ajustar los pesos y sesgos es mediante pequeñas actualizaciones de la siguiente manera:

$$w \to w' = w - \eta \nabla_w C \quad (8)$$

$$b \to b' = b - \eta \nabla_b C \quad (9)$$

Donde η es un parámetro positivo conocido como tasa de aprendizaje, (del inglés, *learning rate*), C es la función de costo, $\nabla_w C$ y $\nabla_b C$ son los vectores de

derivadas parciales de C respecto de w y b, conocidos como vectores gradientes.

Se debe tener en cuenta que, para obtener una buena aproximación con el algoritmo de descenso de gradiente, se debe elegir un valor de tasa de aprendizaje η lo suficientemente pequeño. Al mismo tiempo, hay que tener cuidado de que el valor de η no sea extremadamente pequeño, ya que esto hará que los cambios sean mínimos, y por lo tanto el algoritmo de descenso de gradiente funcione de manera muy lenta.

Un resumen del algoritmo es el siguiente:

1. Introducimos un mini-lote de entrada con N muestras aleatorias provenientes de nuestro dataset de entrenamiento, previamente etiquetado.
2. Después de los cálculos pertinentes en cada capa de la red, obtenemos como resultado las predicciones a su salida. A este paso se le conoce como *forward propagation* (de las entradas).
3. Evaluamos la función de pérdida para dicho mini-lote. El valor de esta función de coste es lo que se trata de minimizar en todo momento mediante el algoritmo, y hacia ello se orientan los siguientes pasos.
4. Calculamos el gradiente como la derivada multivariable de la función de coste con respecto a todos los parámetros de la red. Gráficamente sería la pendiente de la tangente a la función de coste en el punto donde nos encontramos (evaluando los pesos actuales). Matemáticamente es un vector que nos da la dirección y el sentido en que dicha función aumenta más rápido, por lo que deberíamos movernos en sentido contrario si lo que tratamos es de minimizarla (de allí el signo negativo de la ecuación 7).

   El cálculo del vector gradiente en una red neuronal profunda no es para nada algo trivial. Se complica bastante debido a la gran cantidad de parámetros y a su disposición en múltiples capas. ¿Cómo saber lo que influye realmente la variación de un parámetro de la primera capa en el coste final si esa variación repercute en todas las neuronas de todas las capas sucesivas? Para ello existe un algoritmo conocido como back-propagation.

   Dicho algoritmo consiste en comenzar calculando las derivadas parciales de la función de coste a la salida con respecto únicamente a los parámetros de la última capa (que no influyen sobre ningún otro parámetro de la red). No es un cálculo muy complicado gracias a la regla de la cadena.
   Una vez obtenidas estas derivadas, pasamos a la capa anterior, y calculamos nuevamente las derivadas parciales de la función de coste, pero ahora con respecto a los parámetros de esta capa, algo que en parte ya tenemos resuelto precisamente por la regla de la cadena.

Y así seguiremos progresando hacia atrás, hasta llegar al inicio de la red.

5. Una vez obtenido el vector gradiente, actualizaremos los parámetros de la red restando a su valor actual el valor del gradiente correspondiente, multiplicado por una tasa de aprendizaje que nos permite ajustar la magnitud de nuestros pasos.

6. Repetiremos todos los pasos mientras el valor de la función de coste y las métricas de salida no empiecen a empeorar de forma sostenida.

3.5.4. Optimizadores

El descenso del gradiente es un método de optimización de primer orden, ya que toma las primeras derivadas de la función de coste. Eso nos da información puntual sobre la pendiente de la función, pero no sobre su curvatura. Podríamos calcular las segundas derivadas, de forma que pudiéramos conocer también cómo varía el gradiente, pero eso supondría un elevado coste computacional.

Para resolver esta problemática se emplean optimizaciones sobre el descenso del gradiente. A continuación, se detallan los optimizadores empleados en el presente trabajo [4].

Momentum: La clave de esta optimización reside en actualizar los parámetros de la red añadiendo un término adicional que tenga en cuenta el valor de la actualización aplicada en la iteración anterior, de tal forma que se estarán teniendo en cuenta los gradientes anteriores además del actual. El gradiente actual se multiplicará por la tasa de aprendizaje (η) y el valor de la actualización anterior por una constante conocida como coeficiente del momentum (γ)**.**

Propagación cuadrática media (Root Mean Square Propagation, RMSProp): En este método la tasa de aprendizaje se adapta para cada parámetro incluyendo la media móvil exponencial del gradiente al cuadrado. También aparece una constante ρ, que se conoce como factor de olvido:

Adam (Adaptive Moment Estimation): Se trata de una combinación de RMSProp con el Momentum. Por un lado, tendremos la media móvil exponencial del gradiente al cuadrado, y por otro la media móvil exponencial de los pasos anteriores. Suele ser el más rápido de los vistos hasta ahora, y el que se usa por defecto a día de hoy.

3.5.5. Regularización y Dropout

La regularización es cualquier modificación que se le hace al algoritmo de aprendizaje para reducir el error de generalización, pero no necesariamente el error de entrenamiento. Puede ocurrir que debido a un dataset desbalanceado (diferentes cantidades de datos o imágenes para cada clase), o poco variado (posee siempre los mismos datos o tipos de imágenes), la red neuronal reproduzca muy bien el comportamiento de dichos datos, pero no el de datos nuevos, es decir, datos que nunca antes pasaron por la red. Esto se conoce como sobreajuste.

Existen diferentes modos de evitar el sobreajuste. Uno de ellos es obtener más datos para el entrenamiento que, aunque esto no es siempre posible, a veces se pueden aumentar artificialmente.

Otra posibilidad es reducir el tamaño de la red (menor número de parámetros), de modo que la red neuronal sea menos flexible y más robusta, aunque si se reduce demasiado, puede que no sea capaz de aprender o aproximar la función objetivo.

El método empleado en este trabajo para evitar este problema (existen una gran variedad de métodos) fue el aumento del dataset en conjunto con el Dropout.

Dropout

El *Dropout* es una técnica de regularización, que no se aplica modificando la función de costo, y puede ser aplicado en cualquier capa de la red. Consiste en perturbar en cada paso de entrenamiento las activaciones de la capa, apagando aleatoriamente cierta cantidad de neuronas antes de que pasen a la siguiente capa, como se muestra en la Figura 16. Luego de que la red ha sido entrenada, se usan todas las neuronas normalmente.

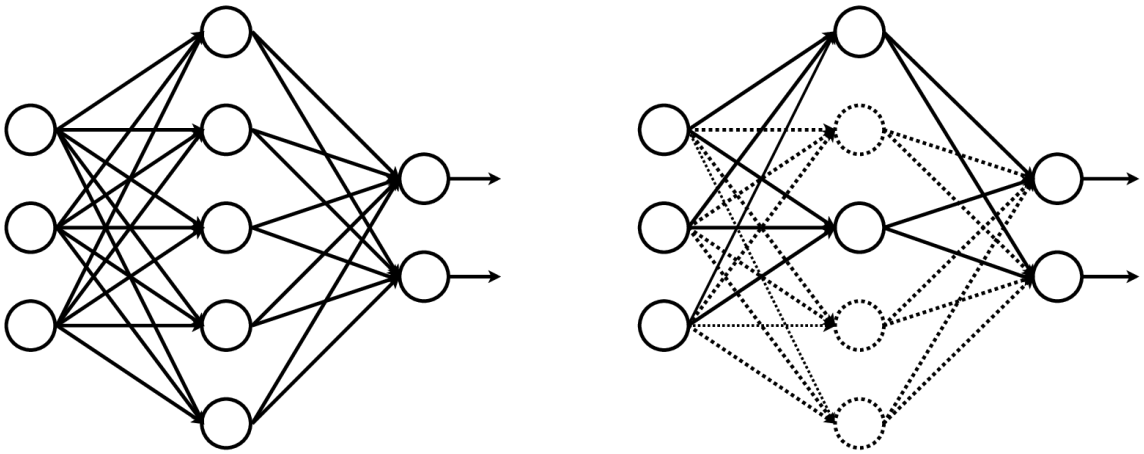

Figura 17. Ejemplo de Dropout

Esto hace que, durante la inferencia, la red no dependa únicamente de activaciones aisladas, sino que las decisiones de la red estarán sostenidas por varias activaciones al mismo tiempo.

Aplicar *dropout* puede ser interpretado como una manera equivalente a entrenar un conjunto de modelos en forma paralela y tomar como decisión final el promedio de los resultados de cada modelo. Esta técnica se conoce como ensamble de modelos.

En el caso de *dropout*, el conjunto de modelos a ensamblar consiste en todas las subredes que se pueden formar removiendo aleatoriamente neuronas de la capa oculta donde se aplica.

Usar *dropout* normalmente implica usar mayor cantidad de parámetros y un mayor tiempo de entrenamiento.

3.6. Métricas de medición

Existen diferentes métricas que se pueden emplear, y cuya finalidad es evaluar el desempeño de una red neuronal determinada. Entre las métricas más usadas se puede nombrar, el *accuracy, precision, recall, F1,* entre otros.

- Accuracy: se define como la relación existente entre el número de predicciones que el modelo ha hecho correctamente frente al número total de predicciones [1].

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

  Donde TP es el número de verdaderos positivos, TN el número de verdaderos negativos, FP el número de falsos positivos, y FN el número de falsos negativos. Sin embargo, el accuracy no siempre es suficiente, por lo que hay otro tipo de métricas centradas en otros aspectos, como se indica en [5].

- Precision: También se conoce como "Valor Predictivo Positivo", y mide el cociente de los verdaderos positivos que registra el modelo frente a todas las predicciones positivas de dicho modelo. Es una estimación del número de veces que el modelo consigue acertar respecto al número de veces que cree que ha aceptado.

$$precision = \frac{TP}{TP + FP}$$

- Recall: También se conoce como "Sensibilidad". Da una medida del número de veces que la red detecta positivos reales respecto a todos los positivos que hay realmente.

$$recall = \frac{TP}{TP + FN}$$

- F1: Engloba al precision y al recall. Es el promedio armónico de esas dos, y se representa como el cociente entre recall y la suma de recall y precision.

$$F1 = \frac{recall}{precision + recall}$$

4. Implementación

Una vez que se conocen todas las herramientas necesarias para empezar a resolver el problema del proyecto, el siguiente paso es explicar en qué consiste exactamente dicho problema. Se quiere diseñar un sistema de clasificación de señales de tráfico para vehículos autónomos que se base en el uso de Visión Artificial implementada con redes neuronales convolucionales de manera que, al proporcionarle una imagen con una señal de tráfico a la red, ésta sea capaz de identificar de qué señal se trata proporcionando al vehículo un estímulo esencial para que pueda actuar en consecuencia con dicha señal. El entorno de programación que se usará será Python con Keras, que usa Tensorflow como back-end.

4.1. Dataset de señales de tránsito

Notebook con el tratamiento del dataset:
https://colab.research.google.com/drive/1O4USuolQsSy0BnmS7bz-rgb3jB46sXAr?usp=sharing

Para el clasificador de señales de tránsito desarrollado en el presente trabajo, se empleó el dataset usado en un desafío de clasificación de imágenes llevado a cabo en la *International Joint Conference on Neural Networks* (IJCNN) en el año 2011. El mismo tiene el nombre de "GTSRB" (German Traffic Sign Recognition Benchmark) [6] y cuenta con 43 clases diferentes, correspondiéndose cada tipo de señal con una clase numérica entre 0 y 42, acumulando un total de más de 60000 imágenes de señales de tránsito cuyos tamaños varían entre 15x15 y 250x250 píxeles.

Las 43 señales a reconocer se pueden ver en la Figura 17, donde las clases de 0 a 42 se asignan a cada señal de izquierda a derecha y desde arriba hacia abajo. Por otro lado, en la Tabla 1 se observan las diferentes clases con sus etiquetas numéricas.

Figura 18. Señales de tránsito empleadas

Labels	Names	Labels	Names
0	Speed limit (20km/h)	22	Bumpy road
1	Speed limit (30km/h)	23	Slippery road
2	Speed limit (50km/h)	24	Road narrows on the right
3	Speed limit (60km/h)	25	Road work
4	Speed limit (70km/h)	26	Traffic signals
5	Speed limit (80km/h)	27	Pedestrians
6	End of speed limit (80km/h)	28	Children crossing
7	Speed limit (100km/h)	29	Bicycles crossing
8	Speed limit (120km/h)	30	Beware of ice/snow
9	No passing	31	Wild animals crossing
10	No passing heavy vehicle	32	End speed + passing limits
11	Right-of-way at intersection	33	Turn right ahead
12	Priority road	34	Turn left ahead
13	Yield	35	Ahead only
14	Stop	36	Go straight or right
15	No vehicles	37	Go straight or left
16	Veh >3.5 tons prohibited	38	Keep right
17	No entry	39	Keep left
18	General caution	40	Roundabout mandatory
19	Dangerous curve left	41	End of no passing
20	Dangerous curve right	42	End no passing veh >3.5 tons
21	Double curve		

Tabla 1. Etiquetas de cada señal de tránsito

El dataset empleado presenta dos problemas principales:

1. Las imágenes poseen una gran variedad de tamaños o resoluciones, las cuales pueden variar entre los 15x15 y 250x250 píxeles, por lo cual será necesario redimensionar las mismas a un único tamaño de 64x64 pixeles antes de usarlas para entrenar nuestra red neuronal.
2. Existe un marcado desbalance en la cantidad de imágenes para cada etiqueta del dataset. Mientras que la señal de límite de velocidad máxima de 50 [km/h] posee 2250 imágenes, la señal de límite de velocidad máxima de 20 [km/h] posee 210 imágenes. Se puede apreciar este desbalance en el histograma de la Figura 18.

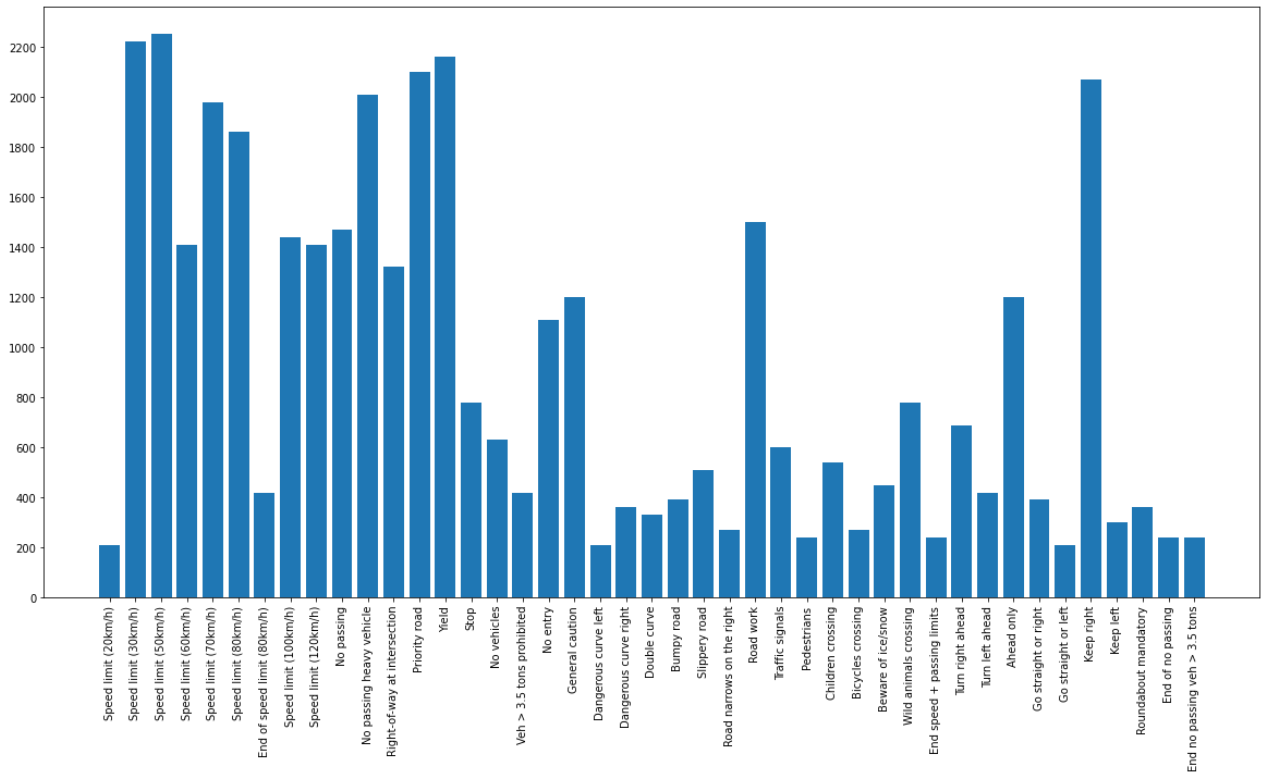

Figura 19. Histograma de clases de señales de tránsito

Debido a estos problemas será necesario modificar el dataset antes de entrenar nuestra red neuronal.

La primera modificación a realizar será dividir nuestro dataset en imágenes para entrenamiento, validación y prueba. Si bien el dataset original (GTSRB) ya se encuentra dividido en set de entrenamiento y prueba, será necesario subdividir el set de entrenamiento para obtener un set de validación a partir de éste. Además, se aprovecha este paso para redimensionar las imágenes a 64x64 pixeles, se escogió este tamaño como un punto intermedio entre las imágenes de menor y de mayor tamaño.

Luego de este paso tenemos:

- Set de entrenamiento: 31.367 imágenes de 64x64 pixeles
- Set de validación: 7.842 imágenes de 64x64 pixeles
- Set de prueba: 12.630 imágenes de 64x64 pixeles

La siguiente modificación será balancear el dataset de entrenamiento de forma que todas las clases tengan la misma cantidad de imágenes. Para realizar esto emplearemos una técnica denominada aumento de datos, la misma consiste en generar nuevas imágenes rotando, trasladando y realizando zoom en imágenes aleatorias. El resultado de realizar esto podemos apreciarlo en el histograma de la Figura 19, en naranja el histograma del set de entrenamiento antes de aumentar los datos y en azul el resultado final.

Además, se agregaron 500 imágenes a todas las clases para tener mayor invarianza a variaciones en la posición o rotación de las imágenes que luego serán detectadas por la red, logrando como resultado final las imágenes de la Figura 20.

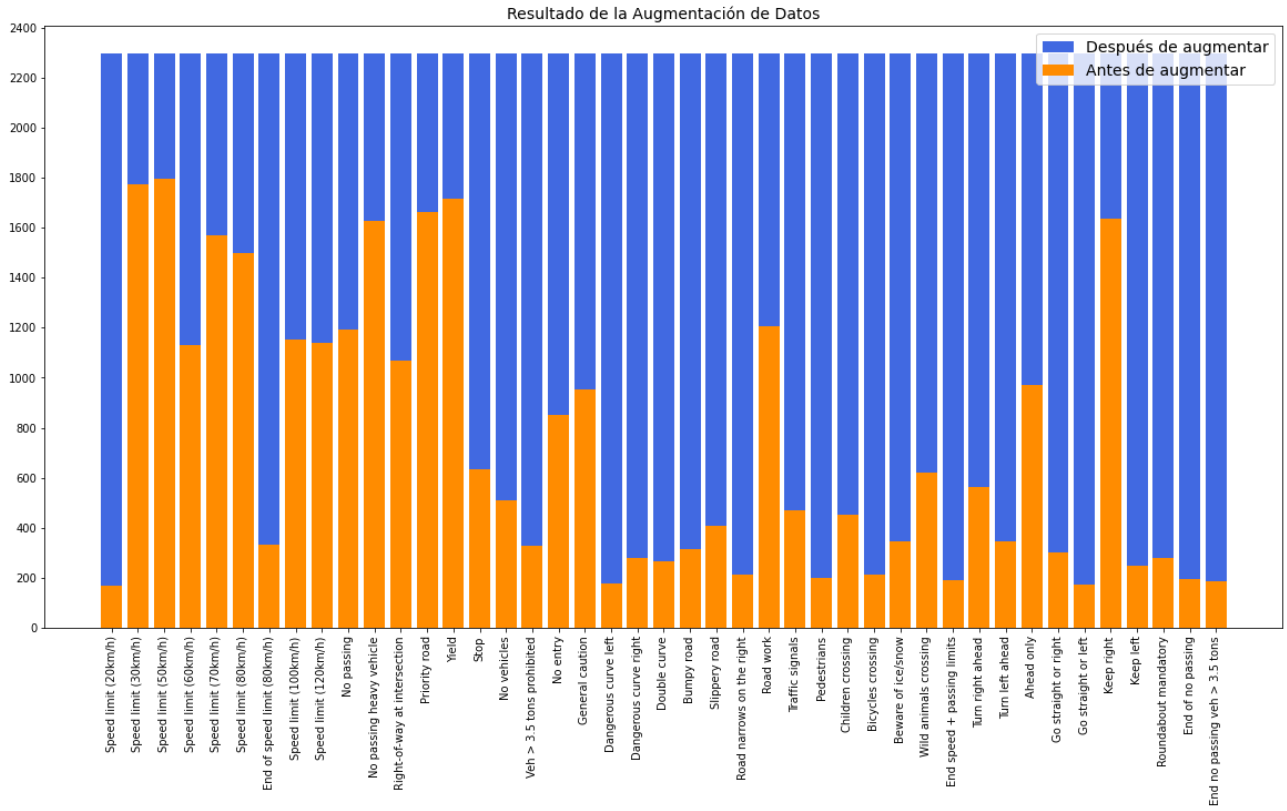

Figura 20. Histograma del resultado del aumento de datos

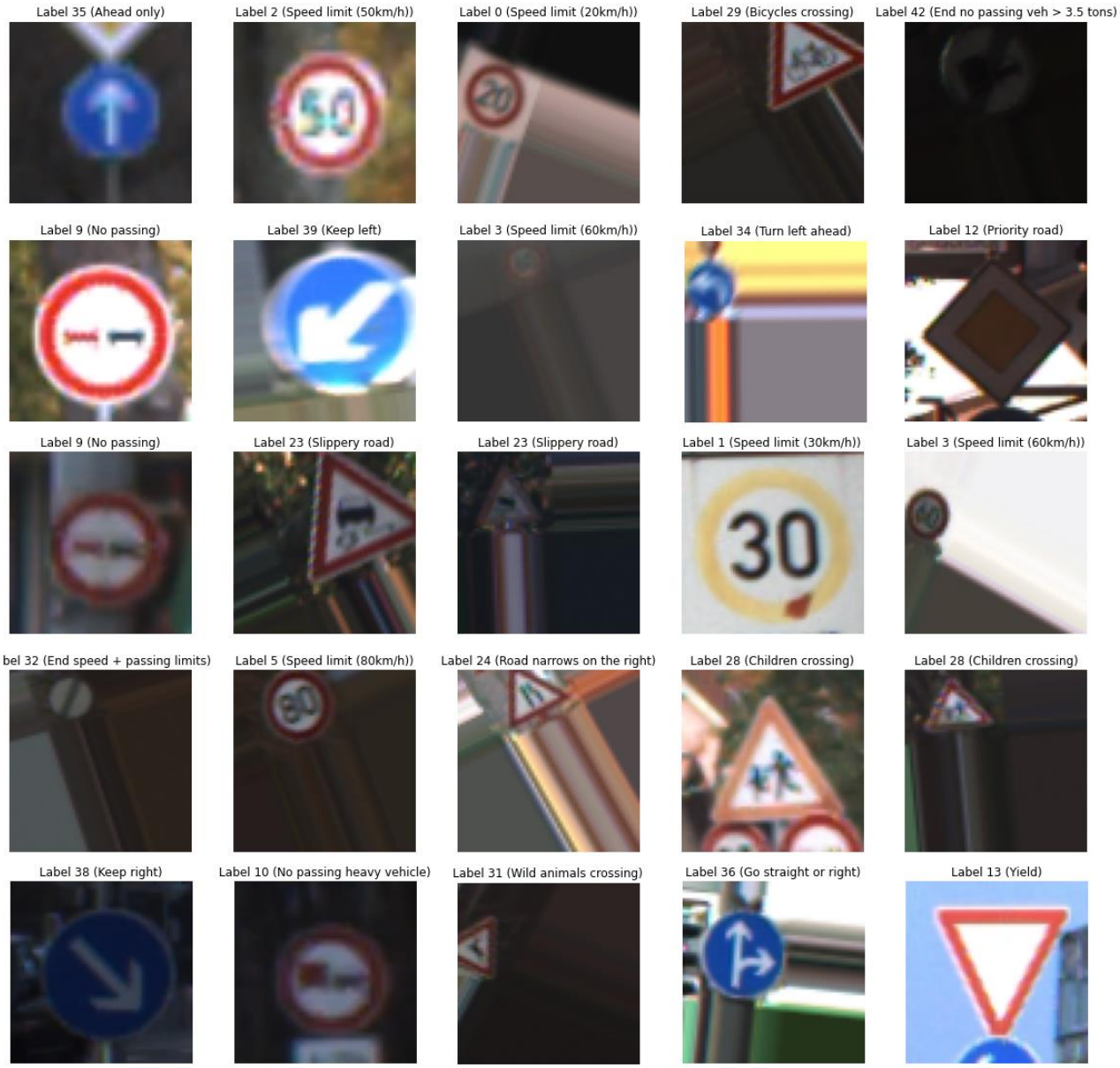

Figura 21. Resultado del aumento de datos en el set de entrenamiento

4.2. Red neuronal completamente conectada

La primera arquitectura puesta a prueba fue una red neuronal completamente conectada. En ella, todas las neuronas de una capa de conectan a todas las neuronas de la capa siguiente (ver Figura 21), lo que introduce ciertas desventajas al modelo. Entre dichas desventajas podemos nombrar:

- Gran cantidad de parámetros a entrenar (pesos y biasses).
- Red poco versátil frente a variaciones en las imágenes de entrada (rotaciones, desplazamientos, entre otros).
- Se pierde la espacialidad de las características en la imagen.

La entrenar dicha red con nuestro set de entrenamiento se obtienen resultados bastante deficientes (ver Figuras 21 y 22), debido a los puntos nombrados anteriormente.

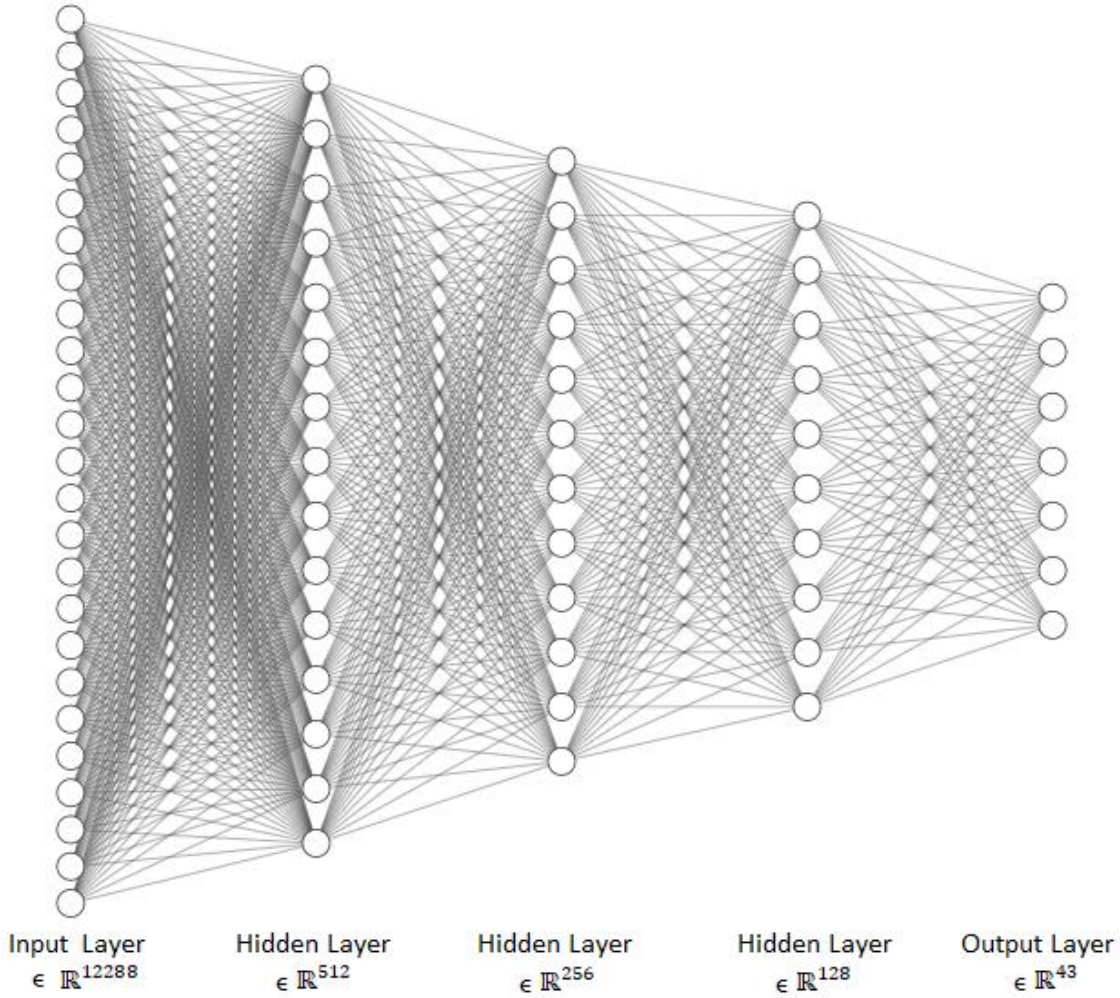

Figura 22. Red neuronal completamente conectada implementada

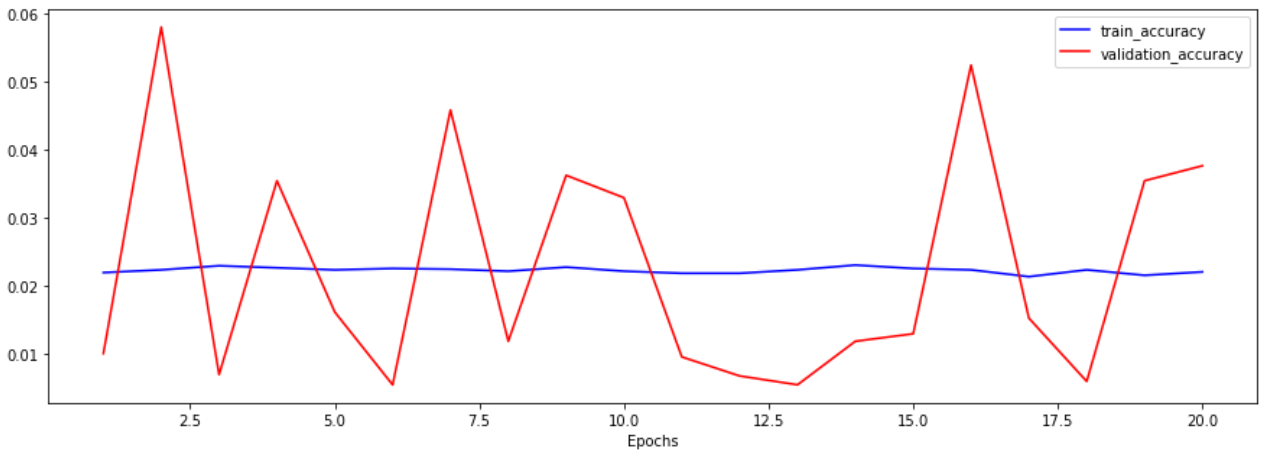

Figura 23. Accuracy red totalmente conectada

4.3. Red neuronal convolucional empleada

La red neuronal puesta bajo prueba en el presente trabajo consta de dos etapas: una primera etapa convolucional, empleando las técnicas descriptas en la sección 3.4, encargada de obtener los distintos mapas de características de las señales.

Además, se agregaron capas de Dropout y de Normalización por lotes luego de cada capa convolucional para evitar el sobreajuste y lograr acelerar el proceso de clasificación, según lo expuesto en las secciones 3.4 y 3.5.

Posteriormente se emplean dos capas totalmente conectadas encargadas de realizar la clasificación propiamente dicha de las señales de tránsito.

La arquitectura de red neuronal empleada para el presente trabajo puede apreciarse en la Figura 23.

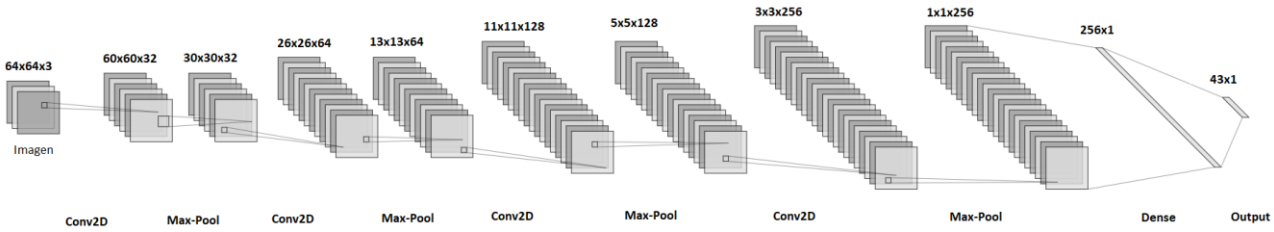

Figura 24. Red neuronal empleada

De acuerdo a lo explicado en las secciones 3.2 y 3.5 existen múltiples funciones de activación, así como también una gran variedad de optimizadores posibles a emplear para obtener el mejor resultado, por lo que, se entrenará la red neuronal mostrada anteriormente empleando las 2 funciones de activación más comunes para problemas de clasificación (ReLU y Leaky-ReLU) cuya descripción encontramos en la sección 3.2.3, en conjunto con 3 optimizadores también muy populares en problemas de clasificación con redes neuronales convolucionales (SGD con Momentum, RMSProp y Adam) cuya descripción encontramos en la sección 3.5.4.

Para realizar un contraste adecuado entre todas las funciones de activación y optimizadores empleadas y elegir la que mejor resultados brinde, entrenamos con cada una a la red neuronal por 20 épocas y medimos sus porcentajes de exactitud. Además, empleamos la misma imagen del set de pruebas para medir el tiempo que demora la red en clasificar la señal (tiempo de inferencia).

4.4. Resultados obtenidos

Notebook con las pruebas realizadas sobre el modelo de mejor resultado:
https://colab.research.google.com/drive/1NAOxDn6Df5QP5nxjbXC36vw8W7T_chzM?usp=sharing

Notebook con los modelos analizados:
https://colab.research.google.com/drive/10SocQKlEXGoomXYCTx_2X8QOkdzdvFun?usp=sharing

Notebook con ejemplo sobre mapas de características:
https://colab.research.google.com/drive/1cSDzDA-iMHCoy9qTeq9wpSBuQ2Nofaky?usp=sharing

Los resultados obtenidos para todos los casos posibles pueden apreciarse en la Tabla 1.

	Train accuracy	Validation accuracy	Test accuracy	Tiempo de inferencia (con imagen de prueba)
Activación: ReLU Optimizador: SGD con Momentum	92.96%	99.41%	93%	0.0561 [s]
Activación: Leaky-ReLU Optimizador: SGD con Momentum	92.21%	99.13%	94%	0.0549 [s]
Activación: ReLU Optimizador: RMSProp	95.69%	99.67%	95%	0.0583 [s]
Activación: Leaky-ReLU Optimizador: RMSProp	95.15%	99.53%	96%	0.1192 [s]
Activación: ReLU Optimizador: Adam	95.90%	99.72%	95%	0.0538 [s]
Activación: Leaky-ReLU Optimizador: Adam	96.28%	99.72%	97%	0.0578 [s]

Tabla 2. Resultados experimentales obtenidos empleando múltiples funciones de activación y optimizadores

Se puede apreciar como la red neuronal que mejor rendimiento tuvo fue aquella que emplea como función de activación a la función Leaky-ReLU en conjunto con el optimizador Adam. En segundo lugar, encontramos a aquella red que emplea como función de activación a la función ReLU en conjunto con el mismo optimizador.

En base a la configuración de red neuronal que mejor resultado obtuvo, se calcularon diferentes métricas para evaluar su desempeño frente al set de pruebas, aclarando que este set de datos posee imágenes completamente nuevas para la red neuronal.

Se calculó la precisión, recall, f1-score, soporte y matriz de confusión. Los resultados obtenidos para todas las clases se muestran en las Figuras 24 y 25.

Aclaración: El *support* es el número de muestras de las respuestas verdadera que se encuentran en esa clase.

	precision	recall	f1-score	support
Speed limit (20km/h)	0.98	0.92	0.95	60
Speed limit (30km/h)	0.98	0.99	0.99	720
Speed limit (50km/h)	1.00	0.99	0.99	750
Speed limit (60km/h)	1.00	0.96	0.98	450
Speed limit (70km/h)	1.00	0.98	0.99	660
Speed limit (80km/h)	0.96	0.99	0.97	630
End of speed limit (80km/h)	1.00	0.98	0.99	150
Speed limit (100km/h)	0.98	0.99	0.99	450
Speed limit (120km/h)	0.99	0.98	0.98	450
No passing	1.00	1.00	1.00	480
No passing heavy vehicle	0.99	1.00	0.99	660
Right-of-way at intersection	0.93	0.98	0.95	420
Priority road	0.99	0.98	0.98	690
Yield	0.99	0.99	0.99	720
Stop	1.00	0.99	0.99	270
No vehicles	0.98	0.98	0.98	210
Veh > 3.5 tons prohibited	0.96	1.00	0.98	150
No entry	1.00	0.97	0.98	360
General caution	0.98	0.91	0.94	390
Dangerous curve left	0.90	1.00	0.94	60
Dangerous curve right	0.94	1.00	0.97	90
Double curve	0.84	0.90	0.87	90
Bumpy road	0.80	0.76	0.78	120
Slippery road	0.83	0.96	0.89	150
Road narrows on the right	0.98	0.90	0.94	90
Road work	0.94	0.99	0.96	480
Traffic signals	0.94	0.96	0.95	180
Pedestrians	0.76	0.58	0.66	60
Children crossing	0.90	0.99	0.94	150
Bicycles crossing	0.96	0.79	0.87	90
Beware of ice/snow	0.97	0.72	0.83	150
Wild animals crossing	0.96	1.00	0.98	270
End speed + passing limits	0.95	1.00	0.98	60
Turn right ahead	0.95	0.98	0.97	210
Turn left ahead	0.98	1.00	0.99	120
Ahead only	0.98	0.98	0.98	390
Go straight or right	0.92	0.99	0.96	120
Go straight or left	0.98	0.95	0.97	60
Keep right	0.99	0.98	0.99	690
Keep left	0.99	0.94	0.97	90
Roundabout mandatory	0.98	0.96	0.97	90
End of no passing	0.83	0.98	0.90	60
End no passing veh > 3.5 tons	0.90	0.98	0.94	90
accuracy			0.97	12630

Figura 25. Métricas resultantes del modelo

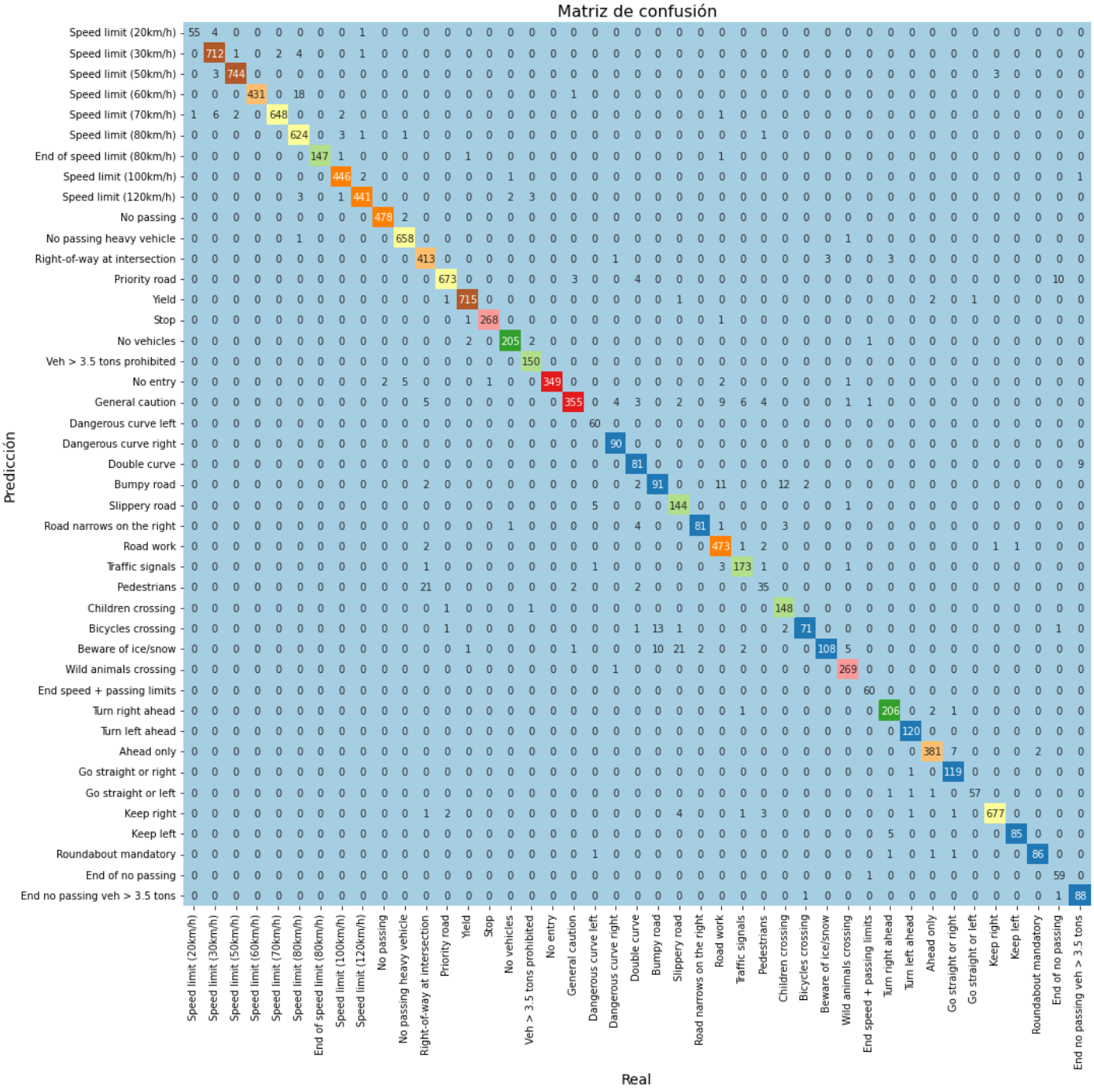

Figura 26. Matriz de confusión del modelo

A modo de prueba final para la red neuronal creada se clasificaron 30 imágenes de señales de tránsito extraídas de internet. Los resultados pueden apreciarse en la Figura 27, además se incluyó las 3 señales con mayor precisión obtenida (se empleó una

escala logarítmica para poder apreciar la precisión de las imágenes que anteceden a la señal predicha.

Figura 27. Resultados sobre imágenes obtenidas de Internet

5. Conclusiones y Mejoras a futuro

Para concluir este trabajo se comentarán los resultados obtenidos contrastándolos con los objetivos planteados en un comienzo.

En primer lugar, se pudo demostrar el deficiente rendimiento que poseen las redes neuronales completamente conectadas en la clasificación de imágenes a través de la extracción de características de las mismas, sumado a la gran cantidad de parámetros que posee dicha red.

En cuanto a la red neuronal convolucional diseñada, se obtuvo un desempeño bastante aceptable para lo simple que es la red en sí misma (únicamente posee 3 capas convolucionales), alcanzando un 97% de *accuracy* empleando el set de prueba. Esto podría explicarse debido al aumento del dataset original realizado y a las pruebas con diferentes funciones de activación y optimizadores para obtener el mejor resultado.

Sobre esto último, es importante destacar como la función de activación Leaky-ReLU obtuvo mejores resultados frente a la función ReLU, dado que evita el fenómeno de muerte o apagado de la neurona descripto en la sección 3.2.3. Además, se debe destacar como el optimizador Adam obtuvo los mejores y más rápidos resultados, lo cual era de esperarse dado que el mismo es una combinación entre el Momentum y RMSProp.

Por último, se observa un resultado bastante aceptable en las imágenes descargadas de Internet. Un caso particular sucede con la señal de "*Bumpy Road*" (4º fila, 4º columna) que es clasificada erróneamente, aún así la segunda opción con mayor precisión (y que no se encuentra muy distante) es la correcta. Esto puede deberse a que en nuestro dataset dicha señal posee un fondo amarillo y la forma negra en su interior es distinta. Esto se confirma en la señal de "*Bumpy Road*" (5º fila, 1º columna), la cual posee una forma negra en su interior igual a la del dataset, y dicha señal es clasificada correctamente.

Para finalizar se proponen las siguientes mejoras a futuro para el presente desarrollo:

- Agregar aún más clases de señales de tránsito, no tan solo señales de tránsito de Alemania como es este caso.
- Realizar un preprocesamiento de las imágenes previo ingreso a la red neuronal para lograr mayor independencia frente a cambios de luz o contraste.
- Investigar y probar redes STN (Spatial Transformer Network) para brindarle al sistema una mayor invariancia (aún mayor a la lograda con el aumento de datos) frente a señales rotadas o trasladadas.

6. Bibliografía

[1] J. Torres, DEEP LEARNING Introducción práctica con Keras., España: Independently published, 2018.

[2] J. Mariano Álvarez, «El perceptrón como neurona artificial,» 10 Junio 2018. [En línea]. Available: http://blog.josemarianoalvarez.com/2018/06/10/el-perceptron-como-neurona-artificial/

[3] Ioffe, S., & Szegedy, C. (2015). Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv Preprint arXiv:1502.03167v3.

[4] Yaqub, M., Jinchao, F., Zia, M. S., Arshid, K., Jia, K., Rehman, Z. U., & Mehmood, A. (2020). State-of-the-Art CNN Optimizer for Brain Tumor Segmentation in Magnetic Resonance Images. *Brain sciences*, *10*(7), 427. https://doi.org/10.3390/brainsci10070427

[5] J. Martínez, «Más Allá del Accuracy: Precisión, Recall y F1,» Diciembre 2019. [En línea]. Available: https://datasmarts.net/es/mas-alla-del-accuracy-precision-recall-y-f1/

[6] http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset