PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

CARRERA DE: INGENIERÍA EN SISTEMAS DE LA INFORMACIÓN



Trabajo de Titulación

Tema: Uso de modelos de aprendizaje profundo para el reconocimiento de señales de tránsito del Ecuador. Casos: Pare, Ceda el paso, límites de velocidad y semáforos

AUTOR:

Leonardo Sebastián Tene Garcés

QUITO DM, ENERO DE 2024

DEDICATORIA

“Me gusta la gente capaz de entender que el mayor error del ser humano es intentar sacarse de la cabeza aquello que sale del corazón” *Mario Benedetti*. Con esta frase dedico mi trabajo de titulación a mi familia y amigos los cuales siempre me supieron dar palabras de aliento, consejos y sobre todo su compañía a lo largo de estos 4 años para alcanzar esta meta. Fueron estos lazos los que, a pesar de saber cómo esto iba a terminar, me ayudaron a disfrutar el camino y espero que perduren para siempre como las palabras escritas aquí.

AGRADECIMIENTO

Quiero agradecer a mis padres, hermano, tías, amigos y profesores que durante estos 8 semestres me enseñaron a cómo lidiar con los nuevos problemas que van surgiendo en este pequeño fragmento de la vida adulta, me enseñaron el peso que tienen las palabras sobre las personas, la importancia de dar siempre lo mejor en todo lo que hago y a ser yo mismo ante el mundo. De verdad muchas gracias por mostrarme que este montón de cosas pequeñas hacen que las cosas grandes en mi vida sucedan

RESUMEN

El desarrollo de un modelo de inteligencia artificial que permita reconocer diferentes tipos de señales de tránsito de Ecuador, con tiempos de procesamiento bajos y con altos porcentajes de precisión es de vital importancia para la introducción de los vehículos inteligentes en el mercado. Para obtener un modelo que cumpla con las características anteriores se optó por utilizar redes neuronales convolucionales las cuales son especializadas en el reconocimiento de imágenes.

La generación del modelo se obtuvo mediante la aplicación de la metodología de CRISP-DM en el este problema de ciencia de datos. Este enfoque estructurado permite abordar todas las fases del proyecto de manera sistemática, desde la comprensión del problema y la preparación de los datos, hasta la evaluación y el despliegue del modelo. Además, fue importante el utilizar Python como principal lenguaje de programación junto con sus diferentes bibliotecas especializadas en redes neuronales como Keras o Tensorflow, que proporcionan herramientas avanzadas para la construcción y entrenamiento de modelos de aprendizaje profundo.

Para determinar si un modelo es útil o no es necesario someterlo a pruebas bajo diferentes criterios de evaluación. En este caso se evaluó al modelo bajo las métricas de precisión y pérdida con la finalidad de observar su desempeño a lo largo de las 300 épocas en las que se entrenó a los modelos

ÍNDICE

ÍNDICE DE FIGURAS 7

ÍNDICE DE FIGURAS 7

CAPÍTULO I: INTRODUCCIÓN 8

1. MARCO DE REFERENCIA 8

1.1. JUSTIFICACIÓN 8

1.2. PLANTEAMIENTO DEL PROBLEMA 8

1.3. OBJETIVOS 9

1.3.1. GENERAL: 9

1.3.2. ESPECÍFICOS: 9

1.4. ALCANCE 9

CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA 11

2. FUNDAMENTACIÓN TEÓRICA 11

2.1. ANTECEDENTES 11

2.2. MARCO TEÓRICO 12

2.2.1. Metodologías para estudios de ciencia de datos 12

2.2.2. CRISP-DM 13

2.2.3. Reconocimiento de imágenes 15

2.2.4. Deep Learning 16

2.2.5. Aprendizaje supervisado 17

2.2.6. Redes Neuronales Convolucionales 18

2.2.7. Funciones de activación 20

2.2.8. Aumento de datos 20

2.2.9. Python 21

2.2.10. TensorFlow 23

2.2.11. Keras 23

CAPÍTULO III: Desarrollo del trabajo 24

3. Metodología para análisis de datos 24

3.1. Comprensión del negocio 24

3.2. Comprensión de los datos 25

3.3. Preparación de datos 29

3.3.1. Selección de datos 29

3.3.2. Limpieza de datos 30

3.3.3. Construcción de nuevos datos 33

3.3.4. Formato de datos 34

3.4. Modelado 35

3.4.1. Selección de técnicas de modelado 35

3.4.2. Generación de modelo simple 36

3.4.3. Generación de modelo sin regularización 38

3.4.4. Generación de modelo con regularización 40

3.5. Evaluación del modelo 42

3.6. Análisis de resultados 44

3.6.1. Evaluación de resultados 44

3.6.2. Proceso de revisión 47

3.6.3. Determinación de los pasos siguientes 47

CAPÍTULO IV: Conclusiones y Recomendaciones 48

Conclusiones 48

Recomendaciones 49

BIBLIOGRAFÍA 51

ANEXOS 53

4. Cuaderno Python 53

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

Figura 1 Ciclo de vida de minería de datos 15

Figura 2 Estructura red CNN 19

Figura 3 Previsualización de imágenes límite de velocidad 27

Figura 4 Previsualización de imágenes ceda el paso 28

Figura 5 Previsualización de imágenes pare 28

Figura 6 Previsualización de imágenes pare 29

Figura 7 Imagen de semáforo no útil 30

Figura 8 Ejemplo imagen no válida por combinación de varias señaléticas.....31

Figura 9 Ejemplo de válida por combinación de varias señaléticas 32

Figura 10 Imagen informativa.....32

Figura 11 Estructura de carpetas del dataset 33

Figura 12 Imagen con etiqueta de clase.....33

Figura 13 Imágenes de entrenamiento y validación 35

Figura 14 Representación gráfica red neuronal tradicional.....37

Figura 15 Visualización de capas red convolucional sin regularización 39

Figura 16 Visualización de capas red convolucional con regularización 41

Figura 17 Precisión por época de redes neuronales sobre datos 42

Figura 18 Pérdida por época de redes neuronales sobre datos 44

Figura 19 Evaluación del modelo bajo datos de prueba 45

Figura 20 Predicciones realizadas por el modelo 46

Figura 21 Predicciones realizadas por el modelo 46

CAPÍTULO I: INTRODUCCIÓN

1. MARCO DE REFERENCIA

1.1. JUSTIFICACIÓN

Con el rápido crecimiento de popularidad de las inteligencias artificiales y sus aplicaciones en diferentes ámbitos resulta incorrecto pensar que el campo automotriz no se verá beneficiado por esta nueva tecnología. Es necesario desarrollar modelos de inteligencia artificial entrenados para escenarios particulares en las calles y carreteras de cada país por la tendencia en aumento de vehículos con piloto automático o asistentes de conducción. De esta manera, se podría poner a disposición de las concesionarias que venden este tipo de automóviles el modelo de inteligencia artificial, lo que favorecería la introducción de dichos vehículos en el mercado ecuatoriano, con mayores niveles de fidelidad.

1.2. PLANTEAMIENTO DEL PROBLEMA

El problema principal radica en la falta de modelos capacitados para reconocer señales de tránsito en Ecuador. Al no poder garantizar la seguridad y el correcto funcionamiento de sus productos en el país debido a las diferencias entre algunas señaléticas del país origen del fabricante, las concesionarias que están comenzando a desarrollar vehículos con piloto automático o asistentes que reconocen los escenarios viales no ven un negocio rentable en el mercado automotriz del país.

Un problema secundario son las alteraciones que se producen en las señales del país. Algunos modelos pueden tener problemas con al momento de reconocer y clasificar las señales debido a grafitis, stickers y otros tipos de vandalismo. Por lo tanto, es necesario crear modelos que no solo identifiquen las principales señales de tránsito, sino que también sean resistentes a diferentes condiciones climáticas, variaciones en la iluminación y problemas de visibilidad.

Para abordar estos problemas se considera que una red neuronal convolucional tendría las capacidades para clasificar con precisión las señales de tránsito del Ecuador. El modelo debe reconocer señales en varias condiciones y con posibles alteraciones para aumentar la seguridad y la eficiencia de los sistemas de conducción asistida y automatizada en el país.

1.3. OBJETIVOS

1.3.1. GENERAL:

Desarrollar un modelo de inteligencia artificial para el reconocimiento de señales de tránsito basado en Deep Learning.

1.3.2. ESPECÍFICOS:

- Investigar y recopilar una base de datos de las señales de tránsito utilizadas en Ecuador, limitándose a señales de Pare, Ceda el paso, semáforos y límites de velocidad.
- Desarrollar modelos de aprendizaje profundo, como redes neuronales convolucionales (CNN) que puedan reconocer y clasificar las diversas categorías de señales de tránsito que se encuentran en el país.
- Evaluar el funcionamiento del modelo bajo la métrica de precisión en sus predicciones sobre conjuntos de datos de validación y pruebas.

1.4. ALCANCE

Este estudio busca entrenar a un modelo de red neuronal convolucional capaz de clasificar las imágenes de las señales de tránsito del Ecuador con un porcentaje de precisión de al menos un ochenta por ciento (80%). Los tipos de señales de tránsito que se utilizarán para entrenar la red neuronal CNN también limitarán el alcance de este estudio los cuales son

- Semáforos
- Pare
- Límites de velocidad

- Ceda el paso

De igual manera, este estudio se limitará a completar las cinco primeras fases de la metodología CISP-DM, sin incluir una fase de despliegue debido a que al no ser un modelo que contemple toda la señalética ecuatoriana no puede ser puesto a funcionar. Es decir, se realizará:

- Comprensión del Negocio: Definir los objetivos del proyecto
- Comprensión de los Datos: Recopilar, describir y explorar los datos de las señales de tránsito
- Preparación de los Datos: Seleccionar y preparar los datos para que sean adecuados para el entrenamiento del modelo
- Modelado: Seleccionar las técnicas de modelado, construir el modelo de CNN y calibrar sus parámetros
- Evaluación: Evaluar el modelo para asegurar que cumple con los objetivos de precisión y rendimiento establecidos

CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA

2. FUNDAMENTACIÓN TEÓRICA

2.1. ANTECEDENTES

Como se mencionó anteriormente, en Ecuador no se han logrado avances significativos en los modelos de reconocimiento de imágenes. Principalmente por la falta de infraestructura tecnológica actual para aplicaciones de estos clasificadores en los escenarios viales del país o en otros escenarios. A esto se suma la falta de datos para el entrenamiento de modelos, lo que retrasa la obtención de modelos aptos para ser implementados en entornos operativos.

Según la investigación de Flores-Calero et al. (2018), que se centró en la detección de señales "Pare" y "Ceda el Paso" y en su segunda entrega, que incluye señales de velocidad, se menciona que las bases de datos liberadas corresponden a países como Alemania, Italia, España, Estados Unidos, etc. La falta de datos específicos en el país dificulta el análisis y el desarrollo de modelos.

Para los estudios mencionados anteriormente se utilizaron técnicas como histogramas orientados por gradientes (HOG), funciones aceleradas robustas (SURF), K-NN, redes neuronales de clasificación y perceptrón de múltiples capas. Estos métodos permitieron la creación de modelos capacitados para localizar e identificar las señales mencionadas. Sin embargo, la posibilidad de utilizar modelos enfocados en la extracción de características, como las redes neuronales convolucionales (CNN), no se estudió. Este método podría presentar mejoras con la precisión del modelo y los tiempos de reconocimiento de imágenes.

2.2. MARCO TEÓRICO

2.2.1. Metodologías para estudios de ciencia de datos

Al igual que el desarrollo de software tuvo su auge el mundo de la ciencia de datos también ha experimentado un alto crecimiento en la última década con la popularización de diferentes lenguajes y librerías especializados en el tema. El principal problema con este crecimiento era la falta de una metodología que permita estandarizar el proceso y las fases a superar para que los resultados de los estudios puedan replicarse y corroborarse por diferentes grupos. Brindando así una mayor validez a los modelos realizados

Por ello, se crearon varias metodologías para guiar los diferentes estudios y proyectos en ciencia de datos. A continuación, se mencionan algunas de las metodologías más populares en el ámbito:

CRISP-DM

Cross Industry Standard Process for Datan Mining o simplemente CRIPS-DM es una de las metodologías más antiguas pero que en la actualidad sigue siendo ampliamente utilizada para proyectos de minería de datos. Está compuesta por 6 fases en donde se maneja un enfoque empresarial para definir objetivos y termina con un despliegue de los modelos.

SEMMA

SEMMA es el acrónimo para Sample, Explore, Modify, Model, Assess. Esta metodología fue desarrollada por el SAS Institute. Al igual que CRIPS-DM, SEMMA está enfocada en solventar problemas de las empresas con resultados similares a un CRM (Customer Relationship Management). Con sus 5 fases abarca menos campo que CRIPS-DM porque comienza desde la selección de datos y termina con la evaluación de los modelos

TDSP

Esta es la metodología sugerida por Microsoft y es acrónimo de Team Data Science Process. Este es muy similar a las metodologías anteriores mencionadas, pero abarca equipos más grandes. Como resultado, ayuda a definir los roles y responsabilidades de los miembros de un proyecto de ciencia de datos. Comienza por establecer objetivos y requisitos. Termina con el monitoreo y mantenimiento de los modelos creados cuando se trata de proyectos interdisciplinarios más grandes.

Para el desarrollo de este trabajo se escogió utilizar la metodología de CRISP-DM con algunas modificaciones. Se decidió utilizar las cinco primeras fases del marco de referencia, con algunas modificaciones en la primera fase para adaptarlo a un proyecto de titulación y no de negocios.

2.2.2. CRISP-DM

Cross-Industry Estándar Process for Data Mining es una metodología para estudios de minería de datos que tengan una aplicación formal fuera de la investigación. De acuerdo con (IBM, n.d.-c) consta de cinco fases de las cuales son:

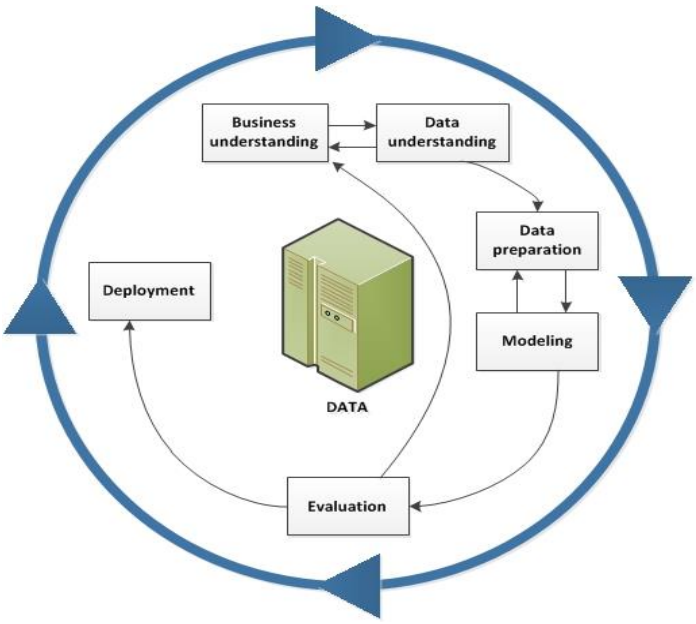
- Comprensión del negocio: En esta etapa inicial, es importante comprender el giro del negocio sobre el cual opera la empresa. Se realiza una evaluación de la situación de la empresa para comprender el alcance y los resultados potenciales. Además, se establecen los objetivos del proyecto tanto comerciales como de minería de datos y, por último, se establece un plan del proyecto.
- Comprensión de los datos: Durante esta fase se busca estudiar más de cerca los datos disponibles. Durante esta se realizan tablas y gráficos para entender o explorar los datos. Gracias a esto se puede llegar a evitar problemas inesperados en el resto de las fases por falta de comprensión.

- Preparación de los datos: Esta es la fase más más extensa de toda la metodología. Dependiendo de los objetivos de minería y comerciales IBM indica que la preparación de datos suele implicar:
    - La fusión de conjunto de datos
    - Selección de muestras
    - Agregar registros
    - Derivación de nuevos atributos
    - Clasificación de los datos de modelado
    - Eliminar o sustituir valores en blanco o perdidos
    - División de conjuntos de entrenamiento y prueba
- Modelado: Esta es la fase más iterativa de toda la metodología ya que en esta se debe experimentar con varios modelos y diferentes parámetros para estos de modo que se obtengan los resultados esperados. No solo prueba los modelos, sino que con los hallazgos que surgen se puede volver a la fase anterior y preparar mejor los datos. Esta fase es el resultado de todas las anteriores.
- Evaluación: En esta etapa, se comparan los resultados obtenidos con los modelos para determinar si cumplen con los criterios de rendimiento establecidos anteriormente. Es fundamental para garantizar que las empresas puedan utilizar los resultados o resultados de los modelos.
- Despliegue: El despliegue consiste en utilizar los conocimientos generados por los modelos en pro de mejoras para la empresa. Gracias a esta fase es posible identificar patrones o tendencias sobre las operaciones de la organización que posiblemente pasaron inadvertidas.

Una característica importante de esta metodología es ser iterativa. Al igual que en el desarrollo de software, la ciencia de datos no promete resultados inmediatos o en un solo ciclo.

Es posible que durante la fase de modelado se identificara que los datos se les puede dar un procesamiento diferente que cambiarían el rendimiento de los modelos. O también es posible que durante la evaluación se identifique que los resultados obtenidos no se ajustan completamente a los objetivos propuestos o que es posible obtener resultados más específicos. Para estas situaciones la metodología no es un limitante y permite regresar en la iteración.

Figura 1
Ciclo de vida de minería de datos



Nota. Adaptado de Conceptos básicos de ayuda de CRISP-DM [Fotografía], por IBM, 2021 Introducción a IBM, (https://www.ibm.com/docs/es/spss-modeler/saas?topic=dm-crisp-help-overview)

2.2.3. Reconocimiento de imágenes

El reconocimiento de imágenes es parte de la inteligencia artificial. Como su nombre lo indica se encarga de identificar, analizar y clasificar un objetivo colocado en imágenes o videos. Utiliza técnicas de aprendizaje automático y redes neuronales para simular la interacción entre los ojos y cerebro humano para reconocer figuras o seres vivos.

La forma de procesar las imágenes es a través de un análisis de pixeles. Dentro de cada pixel se debe identificar características propias del objeto de estudio y posteriormente se une el conjunto de estas para comprender patrones, formas, escenas o textos. El número de pixeles con el cual se desea dividir a la imagen depende del autor, de la infraestructura disponible para realizar el procesamiento y de la nitidez con que se desee trabajar. De estos factores dependerá la eficiencia con la que se entrene el modelo. Según el tamaño se obtendrán las características necesarias para poder realizar predicciones o clasificaciones

Otro factor para considerar es la distinción entre imágenes a blanco y negro, escala de grises o con color. El procesamiento de imágenes a blanco negro es menos costoso en términos computacionales ya que permite categorizar a los pixeles en binario (0 y 1). Esto suele aplicarse con imágenes en donde el fondo no influye en el resultado. Por otro lado, las imágenes a escala de grises ya implican otro tipo de análisis para que las redes puedan aprender. La normalización suele resultar útil en transformar las escalas de grises entre valores comprendidos entre 0 y 1 para agilizar el entrenamiento. Por último, están las imágenes a color. Estás representan un grado de complejidad al obtener los valores de los pixeles por una combinación de matrices. Lo que se realiza es obtener el valor del color en cada canal del RGB y de ahí se obtiene un valor combinado. Este tipo de imágenes suele utilizarse cuando el fondo tiene un grado de influencia sobre el objetivo o cuando el color representa diferencias entre una clase y otra.

2.2.4. Deep Learning

El Deep Learning o aprendizaje profundo es una rama del Machine Learning la cual emplea redes neuronales multicapas para simular al cerebro humano en la toma de decisiones sobre escenarios complejos. Se las denomino como “deep” o “profundo” debido a la gran cantidad de capas de varios niveles que utilizan para aprender características de bajo nivel como bordes hasta las más complejas que permiten identificar formas completas. En el intento de

imitación es necesario que las redes puedan combinar los datos, asignen pesos y sesgos a las características para que puedan ser categorizadas.

Estas capas tienen como mínimo tres capas ocultas con nodos interconectados entre sí y son entrenadas con grandes cantidades de datos. A diferencia de las redes convencionales estás suman algoritmos que favorecen a un mejor aprendizaje. La propagación hacia delante consiste en que cada nueva capa de la red usa información de la anterior para redefinir u optimizar sus predicciones o clasificaciones, comenzando por la capa de entrada hasta la capa de salida. El segundo proceso es la propagación inversa. Son un conjunto de algoritmos que ayudan a calcular errores para modificar los pesos y sesgos desde atrás hacia delante. Es con estos procesos que la red afina su aprendizaje y mejora su precisión

El hardware que utiliza depende del tipo de red que se utilizará. En el caso de las redes convolucionales, especializadas en procesamiento de imágenes, usan GPU o TPU paralelamente para aprovechar núcleos en los espacios de memoria. Esto le ha permitido tener diferentes aplicaciones como:

- Visión por computadora
- Procesamiento del lenguaje
- Reconocimiento de voz
- Juegos
- Simulaciones
- Medicina y bilogía

2.2.5. Aprendizaje supervisado

El aprendizaje supervisado es el método de aprendizaje que usa datos etiquetados para entrenar modelos. Así se indica a los modelos las respuestas sobre una predicción o clasificación. Así estos no tienen que procesar todos los campos y de ahí formar sus propias

conclusiones, sino que más bien se le da los resultados deseados y este identifique patrones o en el caso de imágenes características que le permitan hacer predicciones o clasificaciones que den como resultado la etiqueta que se dio como variable.

Este aprendizaje se usa en 2 problemas: clasificación y regresión. En problemas de clasificación lo que se desea predecir es una variable cualitativa con una etiqueta discreta. Los principales algoritmos utilizados para clasificar son los: árboles de decisión, vecinos cercanos (KNN), regresión logística y redes neuronales. De estos se derivan diferentes aplicaciones como lo son el reconocimiento de imágenes y objetos, analítica predictiva, análisis de opinión del cliente entre otros (IBM, n.d.-b)

En cambio, los problemas de regresión buscan predecir variables cuantitativas como precios, temperatura o demandas de energía. Para estos problemas es mejor utilizar algoritmos como regresión línea o polinómica, árboles de decisión o redes neuronales.

Al igual que todo paradigma este ofrece ventajas y desventajas. Por mencionar algunas de las ventajas se tiene la precisión que se puede obtener. Al conocer las salidas es posible corregir hiper parámetros hasta obtener los niveles deseados. Por este mismo motivo sus algoritmos suelen ser fáciles de interpretar. Sin embargo, la mayor desventaja que posee es lo mismo que lo define, tener los datos etiquetados. Se necesitan muchos datos etiquetados para entrenar los modelos, lo que puede ser laborioso de obtener. De igual manera, al tener una variable con las salidas deseadas es común que los modelos caigan en sobreajuste si no se maneja correctamente los datos de entrenamiento y el preprocesamiento sobre estos.

2.2.6. Redes Neuronales Convolucionales

Es posible usar redes neuronales convencionales para clasificar imágenes con porcentajes altos de rendimiento, sin embargo, esto puede resultar un poco engañoso y se basa en las capas que se manejan por detrás. La red puede funcionar con gran éxito sobre imágenes

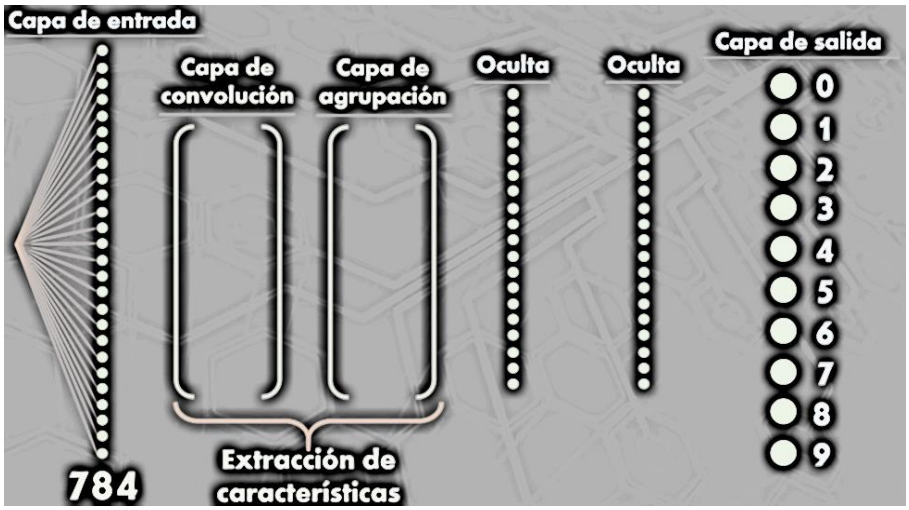
del conjunto de entrenamiento utilizando solo capas ocultas. Sin embargo, en la mayoría de los casos, la red ya no reconoce fotografías si a esta se le cambia de ubicación o se utiliza una externa al conjunto de datos original.

En este punto, las redes neuronales convolucionales ofrecen una opción superior. Según López-Saca et al. (2019), “Las CNN están formadas por dos secciones: extracción de características y clasificación. La ventaja de la red diseñada es que la sección de extracción de características utiliza tres kernel de convolución distintos por donde ingresa una imagen.”

El siguiente es un resumen de cómo funciona la red: Primero debe tomar los pixeles de la imagen; si es una fotografía a color, debe realizar el proceso para cada color RGB. Después, estos pixeles son transferidos a la capa de entrada, luego a la capa de convolución y agrupación, que se encarga de extraer características. El resto del proceso es el normal para una red de clasificación.

Este proceso se explica mejor con la imagen a continuación:

Figura 2
Estructura red CNN



Ringa Tech. (2021). Redes Neuronales Convolucionales- Clasificación avanzada de imágenes con IA / ML (CNN). https://www.youtube.com/watch?v=4sWhhQwHqug

De esta forma le sumamos la capa de características a la red neuronal. Esto favorece al modelo ya que ahora este recibe los pixeles de las imágenes, luego extrae las características y finalmente realiza una clasificación. Con esto, además, se reduce el número de falsos positivos y falsos negativos al tener parámetros definidos que ayuden a categorizar, tal cual lo realiza el cerebro humano.

2.2.7. Funciones de activación

Las funciones de activación son cruciales para el diseño de redes neuronales, especialmente en el aprendizaje profundo porque agregan no linealidad al modelo, lo que permite a la red aprender y modelar las complejas relaciones entre los datos de entrada y salida. Una red neuronal sería esencialmente un modelo lineal sin funciones de activación y se limitaría a tareas básicas que solo requieren ajustes lineales. Estas características ayudan a determinar si una neurona específica debe activarse, es decir, si la información que procesa la neurona es relevante para la tarea en cuestión. Este proceso de activación determina la capacidad de una red neuronal para realizar clasificaciones complejas y otras tareas predictivas.

Existen muchas funciones de activación, cada una con sus propias características y usos. La función sigmoide, la función tangente hiperbólica (tanh) y la unidad lineal rectificada son las más comunes. El tipo de problema, la complejidad del modelo y la necesidad de eficiencia en el entrenamiento son algunos de los factores que pueden afectar la elección de la función de activación.

2.2.8. Aumento de datos

En el campo del aprendizaje automático, el aumento de datos es una técnica clave, especialmente en problemas de visión por computadora y procesamiento del lenguaje natural. Esta técnica utiliza transformaciones aleatorias y controladas en las muestras existentes para aumentar el tamaño y la diversidad del conjunto de datos de entrenamiento. Las rotaciones, las

traslaciones, los cambios de escala, las modificaciones de brillo y contraste son algunas de estas transformaciones.

Esto tiene como objetivo mejorar la capacidad del modelo para hacer predicciones precisas y generalizar en datos de entrada nunca vistos. Al aumentar la variabilidad de las muestras de entrenamiento, se reduce el riesgo de sobreajuste y se mejora la robustez del modelo frente a la variabilidad de los datos del mundo real. El aumento de datos es especialmente útil cuando se dispone de un conjunto de datos limitado. Se puede mejorar la capacidad del modelo para capturar la diversidad y complejidad de los datos subyacentes al generar nuevas muestras a partir de las existentes. Esto resulta en un mejor rendimiento en tareas como clasificación.

El aumento de datos puede realizarse en diferentes tipos de datos. Esto lo hace mediante las diferentes bibliotecas que poseen este tipo de implementación. Entre las principales están

- Keras mediante su clase ImageDataGenerator
- TensorFlow con el módulo tf.image
- Albumentatios es una biblioteca dedicada al aumento de datos con compatibilidad con PyTorch y TensorFlow
- NLTK y spaCY es una herramienta dedicada para el aumento de datos en textos

Las principales consideraciones que se debe tener al momento de usar el aumento de datos es no caer en una sobre generación datos. Al tener demasiados datos aumentados, sobre todo en un conjunto de datos pequeños, se pude crear una distorsión de lo que realmente se desea pronosticar

2.2.9. Python

Python es un lenguaje de programación de alto nivel y de código abierto que fue creado por Guido van Rossum a finales de los 80 y lanzado al mercado en 1991. Este se ha vuelto muy

popular para el desarrollo web, científico, minería de datos entre otros. Lo que caracteriza a Python es que enfatiza la legibilidad del código y la productividad del programador.

Uno de los principales objetivos de diseño de Python es proporcionar una estructura que sea fácil de entender y utilizar, al tiempo que permite la creación de aplicaciones potentes y escalables. Esto se logra con una sintaxis clara y concisa, y una amplia biblioteca estándar que abarca desde el manejo de datos hasta el desarrollo web. Python destaca por su portabilidad, pudiendo ejecutarse en plataformas, desde sistemas operativos de escritorio hasta servidores web y dispositivos embebidos.

Entre las principales bibliotecas que ofrece Python tenemos las siguientes según (Pona, 2022) en su artículo “Las mejores librerías de Python para la ciencia de datos en 2022”

Manipulación de datos

En esta categoría tenemos Pandas y Numpy. Ambas bibliotecas permiten manejar los datos, ya sea mediante matrices multidimensionales para realizar cálculos de alto nivel o mediante estructuras de datos como “DataFrames” que permite manejar datos estructurados.

Gráficos

Dentro de la ciencia de datos es muy importante el poder tener representaciones gráficas de los datos para poder entender sus comportamientos Para existen las librerías de Plotly, Seaborn y Matplotlib. Todas estas librerías son compatibles con las de manipulación de datos y permiten crear gráficos 2D y 3D

Modelado

La librería más popular para realizar modelado, separar conjuntos de datos y evaluar modelos es Scikit-learn. Esta posee una gran variedad de algoritmos que pueden ser

implementados y después ser graficados con las librerías anteriores. De igual manera, se integra muy bien las librerías de TensorFlow y Keras para el desarrollo de modelos de Deep Learning

2.2.10. TensorFlow

TensorFlow es una biblioteca desarrollada por Google, diseñada para facilitar la creación y entrenamiento de modelos de aprendizaje automático y redes neuronales. Surgió como una herramienta para investigadores y desarrolladores en el campo de la inteligencia artificial y el aprendizaje automático, permitiendo la implementación de algoritmos complejos de manera eficiente y escalable. Con TensorFlow se pueden construir una variedad de modelos de aprendizaje automático, incluyendo redes neuronales convolucionales, redes neuronales recurrentes y modelos de aprendizaje profundo. Además, ofrece una amplia gama de herramientas y recursos para facilitar el entrenamiento, la evaluación y la implementación de modelos en producción.

Lo que busca TensorFlow es proporcionar una plataforma flexible y escalable para el desarrollo de aplicaciones de aprendizaje automático. Esto se logra con un modelo de programación simbólico que permite a los desarrolladores definir y manipular grafos computacionales de manera eficiente. También ofrece la capacidad de integrarse fácilmente con otras bibliotecas y herramientas populares.

2.2.11. Keras

Keras es una biblioteca de alto nivel para el desarrollo y la experimentación de modelos de aprendizaje automático, particularmente redes neuronales. Desarrollada originalmente por François Chollet, Keras se destaca por su simplicidad y facilidad de uso, permitiendo a los desarrolladores construir prototipos de modelos de aprendizaje profundo de forma rápida y eficiente. Esta biblioteca funciona como una interfaz para TensorFlow, lo que significa que combina su uso con la potencia y flexibilidad de TensorFlow, facilitando la creación de soluciones sin necesitar un conocimiento profundo de los procesos computacionales subyacentes.

CAPÍTULO III: Desarrollo del trabajo

3. Metodología para análisis de datos

Como se mencionó antes la metodología a usar para el desarrollo de este trabajo es CRISP-DM, especializada en trabajos o investigaciones para análisis de datos. Este tiene un enfoque de negocios que no se adapta a este trabajo académico, pero que puede usar sus guías para un desarrollo organizado. Por este motivo se realizó una modificación en la primera fase de la metodología donde se remplaza los objetivos comerciales y de minería por los objetivos del presente trabajo.

Otra modificación importante al ciclo de CRISP-DM es la omisión de la fase de despliegue del modelo obtenido. Sin importar los resultados, esta fase no se realizará porque el modelo se centrará en analizar solo cuatro señaléticas. Como resultado, no es factible implementar el modelo en un entorno operativo, ya que no sería capaz de responder a todas las señaléticas, sino solo a las cuatro estudiadas. Por lo tanto, no tendría sentido ponerlo en funcionamiento en un contexto real.

Una vez establecidas las modificaciones la primera fase queda de la siguiente manera:

3.1. Comprensión del negocio

En Ecuador, las concesionarias de vehículos se enfrentan a un desafío importante a medida que comienzan a desarrollar tecnologías de piloto automático y asistentes de conducción: la falta de modelos capacitados para reconocer señales de tránsito. Debido a las diferencias en las señalizaciones viales entre Ecuador y los países de origen de los fabricantes, las empresas no pueden garantizar la seguridad y el rendimiento óptimos de sus productos en el país.

Por este motivo se realiza el estudio con los objetivos del trabajo los cuales son:

- Desarrollar un modelo de inteligencia artificial para el reconocimiento de señales de tránsito basado en Deep Learning.
- Investigar y recopilar una base de datos de las señales de tránsito utilizadas en Ecuador, limitándose a señales de “Pare”, “Ceda el paso”, semáforos y límites de velocidad.
- Desarrollar modelos de aprendizaje profundo, como redes neuronales convolucionales (CNN) que puedan reconocer y clasificar las diversas categorías de señales de tránsito que se encuentran en el país.
- Evaluar el funcionamiento del modelo bajo la métrica de precisión y pérdida en sus predicciones sobre conjuntos de datos de validación y pruebas.

3.2. Comprensión de los datos

Para el presente trabajo se usaron datos existentes de sets de datos públicos y datos adquiridos de manera manual gracias a las imágenes que ofrece la navegación de Google Maps y del navegador de Google. La fuente principal o la que proporcionó mayor cantidad de datos fueron los data sets públicos. Lamentablemente, no tenían origen a Ecuador, por lo que se tuvo que realizar otro proceso en la selección para identificar imágenes similares a la señalética ecuatoriana.

La cantidad de datos varía según la categoría, ya que en los diferentes sets de datos no existían muchos para las 4 categorías. En especial para la señalética de “Ceda el Paso” es donde se presentaron las mayores dificultades debido a que esta es muy particular para el caso de Ecuador por incluir la frase “Ceda el paso” dentro de la señalética. La cantidad se reparte de la siguiente manera:

Imágenes para entrenamiento y validación

- Ceda el paso: 1018 imágenes

- Límite de velocidad: 1010 imágenes
- Pare: 993 imágenes
- Semáforos: 2301 imágenes

Total: 5322 imágenes

Con los semáforos se usó el doble de datos que el resto de las categorías porque este es un objeto que no es una placa con una figura fija como el resto de las señaléticas. Sino que este puede cambiar de forma dependiendo del perfil con el que se toma la foto. Con esto en mente se incluyeron más imágenes para que la red tenga más datos con los cuales aprender a reconocer los semáforos.

Datos para pruebas

- Ceda el paso: 100 imágenes
- Límite de velocidad: 100 imágenes
- Pare: 100 imágenes
- Semáforos: 100 imágenes

Total: 400 imágenes

En otras palabras, se destinó un siete por ciento de datos para pruebas y un noventa y tres por ciento para entrenamiento y validación. Como se mostrará más adelante este 93 será dividido en 80-20 lo que destinará un 74.6 por ciento de datos de entrenamiento y 18.6 por ciento para validación en el total. La razón para destinar así los porcentajes es la cantidad limitada de datos con los que se cuenta para el estudio por las causas ya mencionadas anteriormente. Se prefirió por concentrar la mayoría de los datos en entrenamiento y validación para poder dar un mejor entrenamiento del modelo y poder ajustarlo. Tener un 7 por ciento de imágenes de prueba es un buen tamaño para obtener la estimación.

Como parte de esta fase también es necesario realizar una exploración de los datos para poder conocer que tratamiento necesitarán a posterior. De este modo se dispone a continuación una muestra de las imágenes por clase:

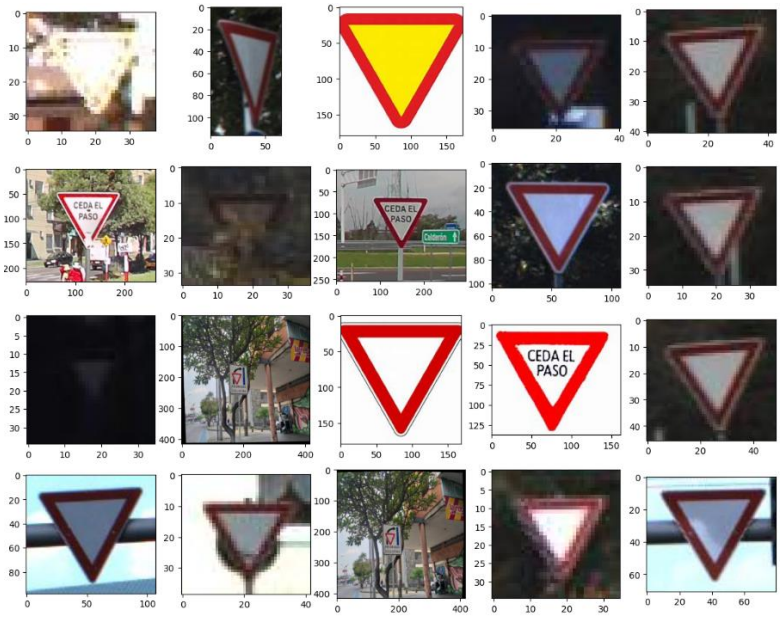
Figura 3
Previsualización de imágenes límite de velocidad



Nota. Imágenes recopiladas para límites de velocidad.

Figura 4

Previsualización de imágenes ceda el paso



Nota. Imágenes recopiladas para ceda el paso.

Figura 5

Previsualización de imágenes pare



Nota. Imágenes recopiladas para pare.

Figura 6
Previsualización de imágenes pare



Nota. Imágenes recopiladas para semáforos.

Como es posible observar las imágenes tienen diferentes resoluciones, principalmente en las imágenes de ceda el paso, pare y límite de velocidad, por esto algunas imágenes salen distorsionada. Esto ayuda a identificar que se necesitará redefinir el tamaño de las imágenes para evitar que esto afecte al reconocimiento.

3.3. Preparación de datos

3.3.1. Selección de datos

Al tratarse de una red CNN para el caso de señalética de “Pare” se usaron imágenes de países con habla inglés ya que su señalética es la misma, de no ser por el texto. Para la red esto no tiene un grado de impacto muy alto debido a que esta no procesa el texto y este ocupa el espacio de manera similar. Sin embargo, para evitar un mal entrenamiento la mayoría de las imágenes poseen el texto en español.

Algo muy similar sucede con las imágenes de límite de velocidad. En otros países estos solo cuentan con el contorno y el número. En Ecuador se suele imprimir el círculo rojo con el número en un rectángulo. Se considera que esta modificación no debería afectar al entrenamiento debido a que lo central de la señalética es común en ambos casos.

Para semáforos no existió un problema con el tipo de imágenes obtenidas debido a que se seleccionaron imágenes solo contornos similares a los del país. Es en las imágenes de “Ceda el paso” donde se encontraron mayores dificultades ya que en Ecuador se incluye la frase de ceda el paso dentro de la señalética, sin embargo, en otros países esto no se realiza. Lo que se optó por hacer en este caso fue utilizar tanto imágenes con la frase como las que no la poseían. Caso contrario el número de imágenes hubiera sido muy bajo y si se poseen algunas con la frase la red podrá aprender sobre esto.

3.3.2. Limpieza de datos

En el caso de imágenes la limpieza de datos consistía en quitar las imágenes que no mantengan una similitud con la normativa ecuatoriana. Por ejemplo, se decidió no utilizar imágenes de semáforos con formas que no se utilizan localmente como se muestra a continuación:

Figura 7
Imagen de semáforo no útil



Nota. Adaptado de Semáforos dataset [Fotografía], por Roboflow, 2022 Semaforos Computer Vision Project, (https://universe.roboflow.com/practica1/semaforos-njeuc)

Además, se prefirió por no usar imágenes con 2 o más señales de tránsito combinadas. Por ejemplo, se descartaron imágenes donde un semáforo y Para eran visibles al mismo tiempo. Para no confundir al modelo. Imágenes con otro tipo de señalética si fue incluida, pero en este tipo de imágenes no superan el uno o dos por ciento del total.

Figura 8

Ejemplo imagen no válida por combinación de varias señaléticas



Nota. Adaptado de Semáforos dataset [Fotografía], por Roboflow, 2022 Semaforos Computer Vision Project, (https://universe.roboflow.com/practica1/semaforos-njeuc)

Figura 9
Ejemplo de válida por combinación de varias señaléticas



Nota. Adaptado de Semáforos dataset [Fotografía], por Roboflow, 2022 Semaforos Computer Vision Project, (https://universe.roboflow.com/practica1/semaforos-njeuc)

Otro tipo de imágenes que si fueron permitidas son las que no son tomadas en un contexto real, sino que son parte del manual de la Agencia Nacional de Tránsito como informativa para la señalética y que han sido reguladas por el INEN

Figura 10
Imagen informativa

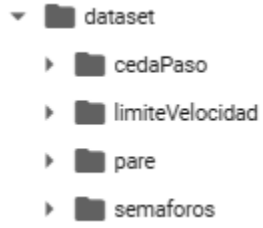


Nota. Adaptado de Señalización Vial. Parte 1. Señalización Vertical [Fotografía], por INEN, 2011, Reglamento Técnico Ecuatoriano, (RTE-4-1 Información (obraspublicas.gob.ec))

3.3.3. Construcción de nuevos datos

Este tipo de redes trabaja con aprendizaje supervisado para identificar las características y después verifica la clase a la que pertenece, así aprende a reconocer. Por este motivo, fue necesario incluir una etiqueta con el tipo de señal de tránsito a la que pertenecía la imagen. Esto fue hecho mediante código gracias a la librería “flow_from_directory” de Keras. Como el data set fue formado a partir de 4 carpetas la librería reconoce a cada carpeta como una categoría diferente, con eso añade una columna de etiqueta al conjunto de pixeles proveniente de cada imagen según la carpeta donde estaba.

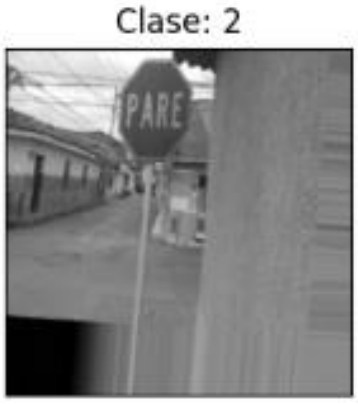
Figura 11
Estructura de carpetas del dataset



Nota. Estructura de carpeta en Google Colab para la asignación de etiquetas

Con esto y tras las transformaciones realizadas que serán explicadas más adelante la imagen queda de la siguiente manera:

Figura 12
Imagen con etiqueta de clase



Nota. Imagen con etiqueta de clase después de aplicar “flow_from_directory”

3.3.4. Formato de datos

El formato de datos va muy de la mano con el modelo que se planea utilizar y el tipo de los datos. Al ser una red neuronal que trabajará con imágenes, todas las imágenes deben tener un tamaño definido para que las capas de la red procesan el mismo número de pixeles por foto. Para este trabajo se definió un tamaño de 224 por 224 pixeles. Este tamaño se definió según la nitidez de las imágenes y el tamaño de algunas provenientes de los data sets públicos. Con este tamaño se mantenía un equilibrio entre la cantidad de pixeles que iban a ser procesador por la red y la claridad de las fotos, evitando así que estás fueran irreconocibles más allá de la etiqueta.

También para reducir la carga de procesamiento se transformó las imágenes a escala de grises. Con esto se reduce la cantidad de pixeles que tendría que procesar la red. Como se mencionó en el marco teórico las imágenes a color necesitan ser procesadas 3 veces para obtener una entrada en la red, una por cada color de RGB con el valor por píxel. Al ser imagen de escala de grises solo es necesario obtener un valor entre 0 y 255. También se descartó el uso de pixeles binarios, blanco o negro. Esto por los datos que incluyen un fondo, diferentes ángulos e iluminación útil para entrenar al modelo.

El último formateo que se realizó fue el aumento de datos. Este consistió en dar una normalización de los valores de los píxeles para que tengan un valor entre 0 y 255 (escala de grises). Siguiendo se dio la libertad de que las imágenes roten en un rango de +/- 15 grados. Otros parámetros de aumento de datos son la traslación horizontal y vertical. Para ambos casos se estableció un rango de +/- 25 porciento de libertad para que se trasladen las imágenes sin que se pierda las señaléticas. También se realizó una distorsión de cizallamiento la cual provoca un desplazamiento relativo en sentido horizontal entre las filas de píxeles de la imagen. Se dio un rango de 0.5 y 1.25 de zoom que se le puede dar a la imagen sobre su tamaño original.

En cuanto a la separación de los conjuntos de entrenamiento, prueba y validación de destinó un 7 por ciento para pruebas y un 93 por ciento para entrenamiento y validación. De este último se dividió en un 80 por ciento para entrenamiento y 20. Con esta repartición las imágenes de entrenamiento quedaron de la siguiente manera:

Figura 13
Imágenes de entrenamiento y validación



Nota. Imágenes de entrenamiento y validación después del aumento de datos

3.4. Modelado

3.4.1. Selección de técnicas de modelado

La técnica de modelo que se adapta a los objetivos del trabajo es la red neuronal convolucional. Sin embargo, para poder hacer una comparación se trabajará con 3 modelos. El primero es una red neuronal tradicional de clasificación. Por tradicional se refiere a una red con capas estándar, sin ninguna de convolución y con varias neuronas por capa.

El segundo modelo consiste en una red con capas de neuronas tradicionales y con capas convolucionales, pero sin ninguna regularización. De esta forma el modelo puede caer en sobreajuste al perder la capacidad de desactivar sus neuronas aleatoriamente. Este proceso ayuda a la red a no aprender demasiado bien los detalles y el ruido en conjunto de datos de entrenamiento.

El tercer modelo es uno con neuronas tradicionales, capas de convolución y con métodos de regularización que permitan a la red alcanzar su rendimiento más alto. Este modelo junto con el aumento de datos y la capacidad de las redes CNN de aprender características se prevé no caer en sobre ajuste del modelo.

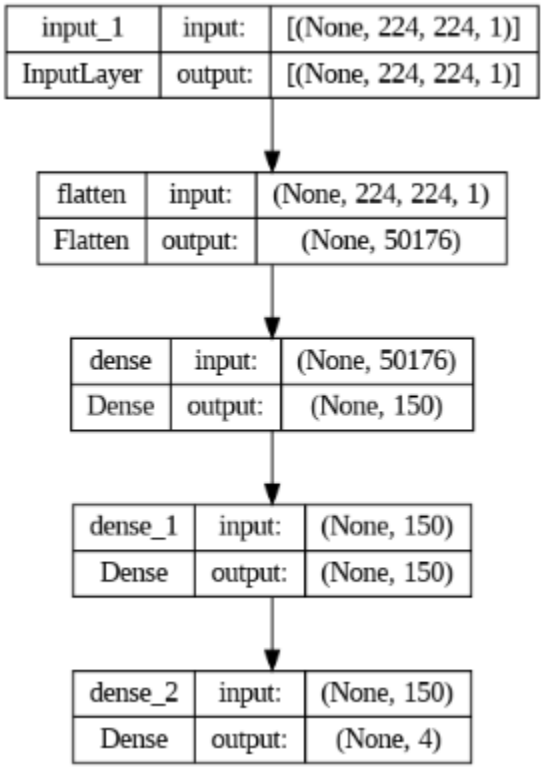
3.4.2. Generación de modelo simple

Una vez definidos los modelos se procede a generar estos en el entorno de Google Colab. El primer en ser implementado es la red neuronal tradicional. Este modelo cuenta con 4 capas de las cuales una es para entradas. Está configurada para recibir imágenes de 224 pixeles de alto y 224 de ancho, además se especifica los canales con los cuales va a tratar la red. Para este caso se establece el parámetro con un valor de 1 al solo tener un canal de color (escala de grises). Las siguientes capas son de procesamiento y estás cuentan con 150 neuronas cada una. La función de activación seleccionada para las capas es “ReLU” que como se mencionó en el marco teórica es una de las funciones más rápidas y eficientes mientras ayuda a la red a no caer en la linealidad y que pueda aprender relaciones complejas en los datos. Finalmente, este modelo posee su capa de salida con 4 neuronas, cada una para representar cada clase y con su función “softmax”. El porqué de esta función radica en los resultados que se esperan por parte de la red. Al tratarse de una respuesta multiclase se prefirió asignar probabilidades a cada una de las clases de salida dando así una distribución de probabilidad en donde las imágenes puedan ser clasificada y asignadas un valor para realizar la predicción

Al tratarse de un modelo simple este tendrá una ejecución de únicamente 100 épocas dentro de las cuales se podrá observar y tener un estimado de su comportamiento. En la compilación del modelo se configuró a este con Adam, un algoritmo muy común y eficaz que ajusta los pesos de la red minimizando el error o pérdida. De igual manera se definió a “categorical_crossentropy” como la función de pérdida del modelo. Esta función de pérdida compara la distribución de probabilidad prevista por el modelo contra la distribución real de las etiquetas, y mide el rendimiento del modelo a través de la entropía cruzada entre estas distribuciones. Por último, se colocó las métricas con las cuáles se va a evaluar el modelo. Para este caso se seleccionó la precisión.

El modelo quedaría definido de la siguiente manera:

Figura 14
Representación gráfica red neuronal tradicional



Nota. Visualización de la red neuronal a través de TensorFlow

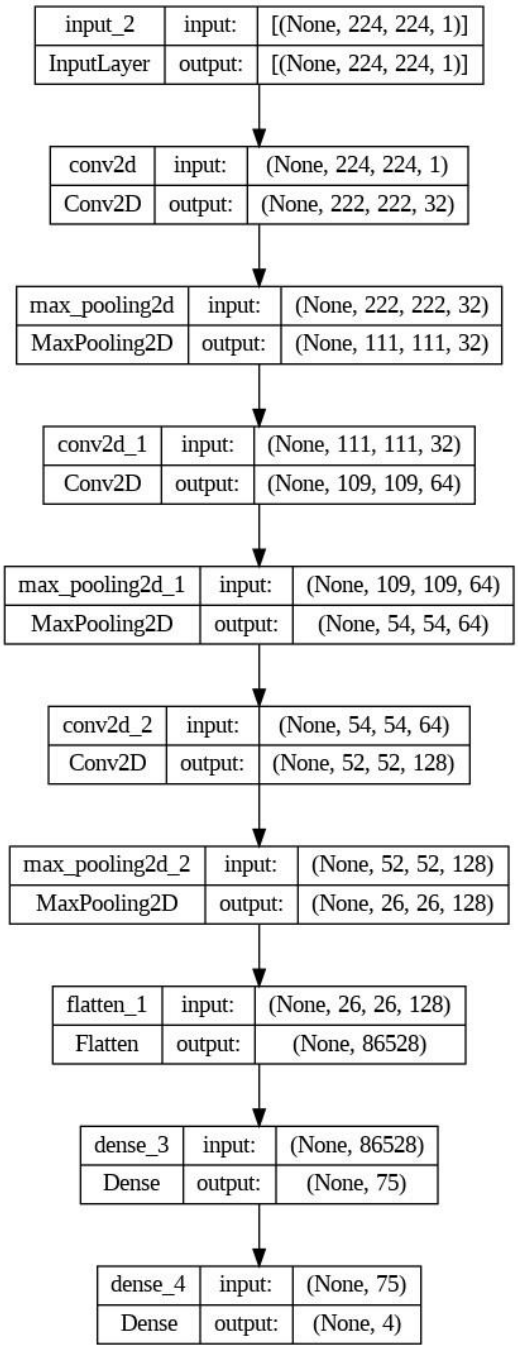
3.4.3. Generación de modelo sin regularización

Este modelo es muy similar al anterior con la configuración de las capas de entrada y de salida. El cambio en este modelo es lo que se incluye capas de convolución. Para ser más específico este posee tres capas de convolución con 32, 64 y 128 neuronas respectivamente. También se incluyó una capa tradicional con 75 neuronas. En cuanto a las funciones de pérdida, activación y optimización estás fueron las mismas que el anterior modelo para tener una comparación entre iguales.

Para este caso se entrenó al modelo durante 300 épocas ya que la muestra de 100 no era lo suficientemente significativa para permitir al modelo aprender y desarrollarse correctamente

Figura 15

Visualización de capas red convolucional sin regularización



Nota. Visualización de la red neuronal a través de TensorFlow

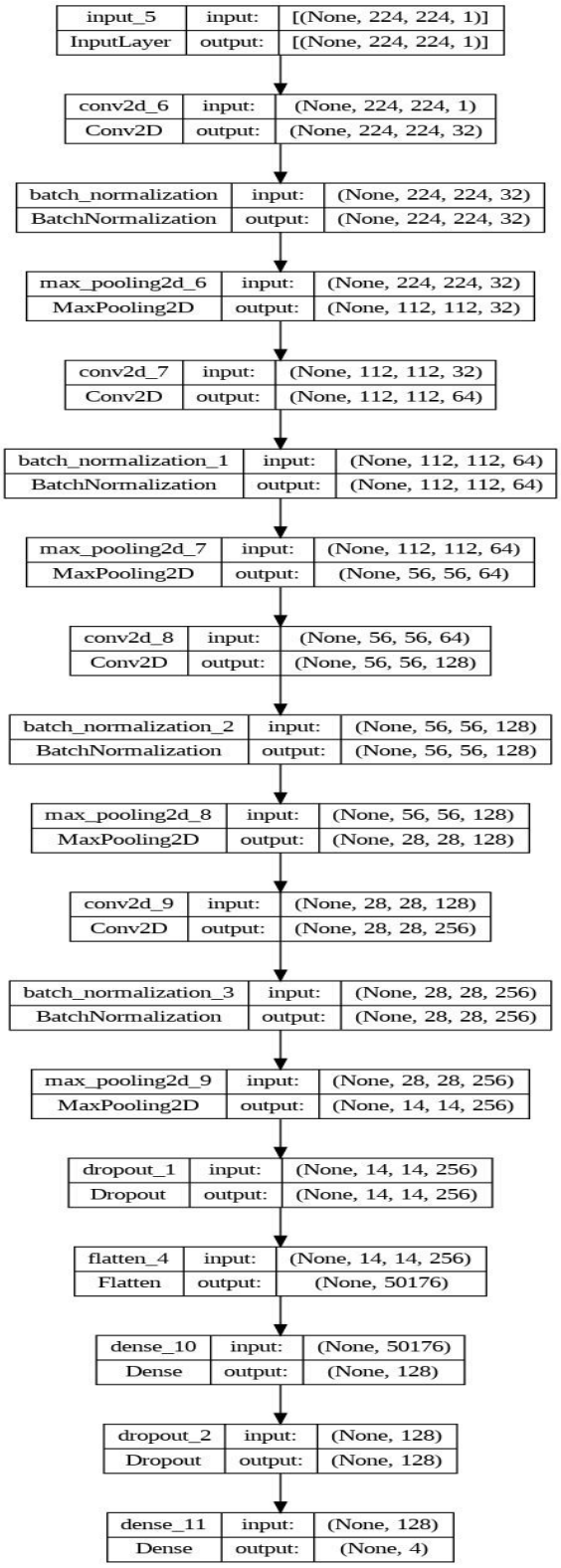
3.4.4. Generación de modelo con regularización

Este modelo es más robusto que los anteriores debido a que al presentar regularización es más resistente al sobre ajuste, pero de igual manera lo vuelve más complejo. En si este modelo posee una estructura muy similar a la anterior, pero con una capa más de convolución de 256 neuronas. Para la regularización se decidió incluir 4 normalizaciones de batch y dos de dropout. Además, en la capa tradicional se incluyó un parámetro de regularización L2. También se modificaron los parámetros de la función de optimización Adam a un valor de 0.001 con el cual se busca un mayor suavizado en los datos.

Al igual que el anterior modelo este será entrenado con 300 épocas para una correcta visualización del desenvolvimiento de este.

Figura 16

Visualización de capas red convolucional con regularización



Nota. Visualización de la red neuronal a través de TensorFlow

3.5. Evaluación del modelo

Cuando los 3 modelos se entrenan, es hora de observar sus desempeños según la métrica de precisión establecida antes sobre los conjuntos de entrenamiento y validación. Para una mejor visualización se utilizó las herramientas de tensorboard para que permita observar el cambio en el tiempo de los modelos según el paso de las épocas como se muestra a continuación:

Figura 17
Precisión por época de redes neuronales sobre datos



Run	Smoothed	Value	Step	Relative
CNN/train	0.9758	0.976	299	5.01 hr
CNN/validation	0.7881	0.7836	299	5.01 hr
CNN2/train	0.9694	0.973	299	11.38 hr
CNN2/validation	0.67	0.54	299	11.38 hr
denso/train	0.452	0.4456	99	46.39 min
denso/validation	0.4738	0.477	99	46.39 min

Nota. Gráfico generado con Tensor Board a partir de los datos obtenidos en la fase de entrenamiento sobre la métrica de precisión

Una de las primeras impresiones que se tiene de la imagen anterior son los altos valores obtenidos durante las fases de entrenamiento en los modelos convolucionales. Estos llegan a tener valores muy cercanos a 1 y su crecimiento es alto en las primeras épocas del modelo y luego se estabilizan. Este comportamiento se explica por la cantidad de datos con los que se entrena el modelo. En total se utilizaron 4255 imágenes para el entrenamiento y 1063 para validación. Si bien los datos de entrenamiento son el 74.6 por ciento de los datos los modelos

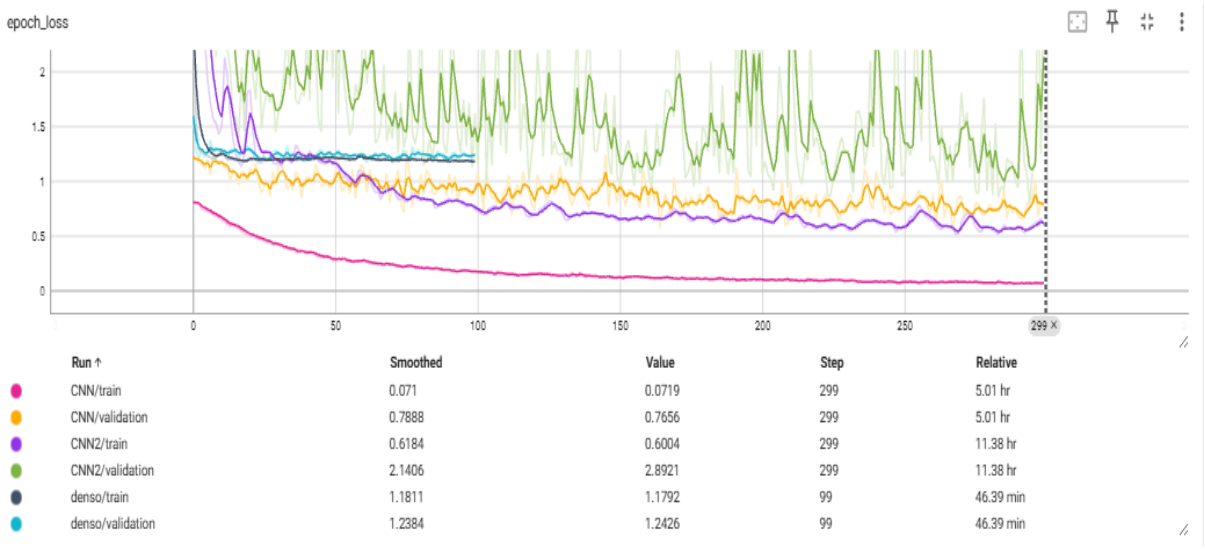
parecen memorizar los datos. Esto es más evidenciado en el modelo sin regularización que comienza con valores más altos en las primeras épocas.

El modelo de red convolucional con regularización obtiene los resultados más altos del resto. Siendo en la época 294 en donde alcanza un máximo de 0.91 de precisión sobre los datos de validación. Este tiene un comportamiento variable, pero con una tendencia de crecimiento, como se evidencia durante las diferentes regularizaciones del modelo. Es por este comportamiento que se descarta la idea de que el modelo caiga en sobre ajuste y que sus resultados sean más fiables.

En cuanto al modelo sin regularización vemos que su conjunto de entrenamiento tiene un comportamiento similar al del modelo con regularización con la diferencia antes mencionada. Su crecimiento es más estable, pero existe una diferencia considerable entre la validación y el entrenamiento, lo que podría significar que este modelo si cae en un sobre ajuste.

Por último, el modelo tradicional es el que tuvo el desempeño más bajo en sus 100 épocas de entrenamiento. Nunca supera el 0.55 de precisión lo que ayuda a demostrar que las redes tradicionales poseen un rendimiento diferente que las convolucionales.

Figura 18
Pérdida por época de redes neuronales sobre datos



Run	Smoothed	Value	Step	Relative
CNN/train	0.071	0.0719	299	5.01 hr
CNN/validation	0.7888	0.7656	299	5.01 hr
CNN2/train	0.6184	0.6004	299	11.38 hr
CNN2/validation	2.1406	2.8921	299	11.38 hr
denso/train	1.1811	1.1792	99	46.39 min
denso/validation	1.2384	1.2426	99	46.39 min

Nota. Gráfico generado con Tensor Board a partir de los datos obtenidos en la fase de entrenamiento sobre la métrica de la pérdida por época

Para complementar la evaluación también se debe considerar la pérdida o loss. En este puede observar que encontró su mejor desempeño igual en la época 294 con un valor de 0.8 lo que indica que en si no existe tanta discrepancia entre las predicciones hechas con el modelo con las reales.

3.6. Análisis de resultados

3.6.1. Evaluación de resultados

Para obtener otra medida de desempeño es la obtenido directamente con los conjuntos de pruebas.

Figura 19
Evaluación del modelo bajo datos de prueba

```
# Obtener las predicciones del modelo para el conjunto de prueba
predictions = modeloCNN2.predict(test_generator)

# Convertir las predicciones a etiquetas de clase
predicted_classes = np.argmax(predictions, axis=1)

# Obtener las etiquetas verdaderas
true_classes = test_generator.classes

# Obtener los nombres de las clases
class_labels = list(test_generator.class_indices.keys())

# Evaluar el modelo y mostrar precisión en el conjunto de prueba
test_loss, test_accuracy = modeloCNN2.evaluate(test_generator)
print(f'Loss en el conjunto de prueba: {test_loss}')
print(f'Accuracy en el conjunto de prueba: {test_accuracy}')
```

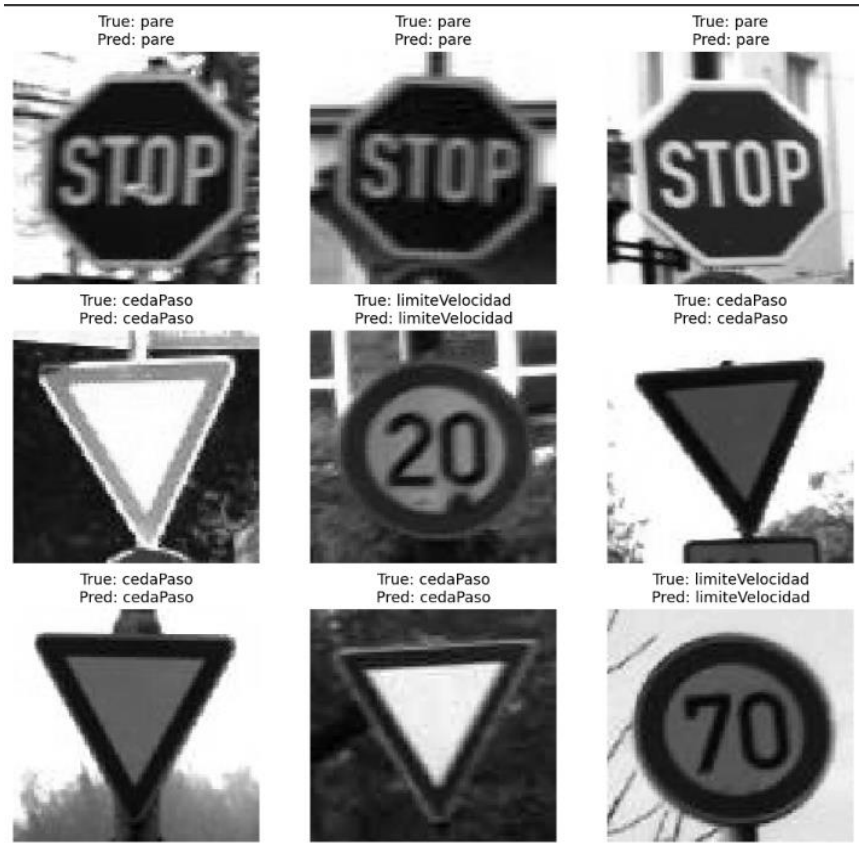
```
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 115ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 114ms/step - accuracy: 0.8800 - loss: 1.1384
Loss en el conjunto de prueba: 1.7527399063110352
Accuracy en el conjunto de prueba: 0.8149999976158142
```

Nota. Se somete al modelo a los datos de prueba para obtener las métricas de precisión y pérdida

Se realizaron dos pruebas de precisión para el modelo. La primera corresponde a realizar directamente las predicciones sobre el conjunto de prueba y comparar con las etiquetas reales. En ese caso el modelo obtuvo un 0.88 de precisión lo que indica que si no es excelente si es apto para las predicciones. En la segunda se usaron las métricas propias de la librería para evaluar el desempeño en términos de precisión. Este devolvió un valor mucho menor de 0.81. A fines del estudio y al superar el 0.80 se considera como válido. Además, en ambos casos la pérdida tiene valores bajos y en comparación con los obtenidos en los valores de validación se observan datos similares. Lo que indica que el modelo no ha caído en sobre ajuste

A continuación, se mostrará el resultado de algunas de las predicciones y las etiquetas reales que debían presentar

Figura 20
Predicciones realizadas por el modelo



Nota. Resultados de las predicciones sobre las imágenes de prueba

Figura 21
Predicciones realizadas por el modelo



Nota. Resultados de las predicciones sobre las imágenes de prueba

3.6.2. Proceso de revisión

Revisando la iteración del ciclo el tiempo de entrenamiento que tomó el modelo convolucional con regularización fue extenso ya que duró un aproximado de doce horas. Para próximas iteraciones se podría plantear utilizar otro componente de hardware para el entrenamiento. Si bien la TPU tuvo un rendimiento mayor a las tarjetas de gráficas básicas ofrecidas por Google Colab tomando en cuenta las unidades de cómputo se podría explorar otras tarjetas gráficas como la NVIDIA A1000

Cada vez que se necesite procesar, el proceso de carga de los datos se demora mucho. Una mejor alternativa sería el cargar las imágenes dentro del Google Drive para poder almacenar ahí y poder consumir los datos directo de la nube y no desde el entorno de ejecución el cual es volátil.

3.6.3. Determinación de los pasos siguientes

Con los resultados anteriores las consideraciones que se deben tener son las siguientes:

- Se necesita una mayor cantidad de imágenes en todas las clases para que el modelo posea más datos de los cuales aprender.
- Con los parámetros actuales se obtuvieron resultados positivos, esto indica que en principio no se necesita modificar estos para la próxima iteración
- Debido a que el modelo mostraba una tendencia de crecimiento se podría entrenarlo por más épocas y obtener su máximo resultado

CAPÍTULO IV: Conclusiones y Recomendaciones

Conclusiones

- El modelo de red neuronal convolucional con regularización tuvo una precisión del 91 % en los datos de validación y del 81 % en los datos de prueba. Estos hallazgos confirman la eficacia del modelo en la clasificación de señales de tránsito en Ecuador. Cumple con el objetivo específico de rendimiento del modelo, que se centra en la identificación y clasificación adecuadas de estas señales. La alta precisión de los datos de validación indica que el modelo conoce bien las características relevantes de las señales de tránsito, mientras que la precisión de los datos de prueba indica su capacidad para generalizar a datos no vistos previamente.
- En comparación con el modelo sin regularización, el comportamiento del modelo con regularización demostró una mayor estabilidad y una menor propensión al sobreajuste. Esto es esencial porque uno de los desafíos en el aprendizaje profundo es evitar que el modelo se ajuste demasiado a los datos de entrenamiento y pierda la capacidad de generalización. Esto destaca la importancia de las técnicas de regularización en la construcción de modelos robustos y confiables.
- El modelo de red neuronal tradicional tenía una precisión del 55%, lo cual era significativamente inferior. Este resultado destaca la superioridad de las redes neuronales convolucionales en tareas de reconocimiento de imágenes, destacando su capacidad para extraer y aprender características complejas. Este resultado respalda el segundo objetivo específico que busca fomentar el desarrollo y la implementación de modelos de aprendizaje profundo avanzados como las redes neuronales convolucionales para mejorar la precisión en las tareas de clasificación de imágenes.
- La cantidad limitada de datos de entrenamiento, particularmente para la señalética de "Ceda el paso", fue un gran problema. A pesar de esto, el modelo funcionó bien, lo que indica que

el desempeño del modelo podría mejorar significativamente con un aumento en la base de datos de entrenamiento. Este elemento está en línea con el primer objetivo específico que enfatiza la necesidad de tener un conjunto de datos amplio y representativo para entrenar de manera efectiva modelos de aprendizaje profundo. Aumentar la cantidad de datos disponibles mejoraría la precisión y ayudaría al modelo a generalizarse a nuevas señales de tránsito.

- La baja y constante pérdida observada en el modelo con regularización durante el entrenamiento y la validación confirma que sus predicciones son confiables. Esta característica muestra la capacidad del modelo para aprender y retener características relevantes sin caer en sobreajuste. La consistencia en la pérdida indica que el modelo no solo es preciso, sino que también funciona bien durante las distintas etapas del proceso de entrenamiento y validación.
- El extenso entrenamiento, que fue de 12 horas, demuestra la necesidad de considerar mejoras en la infraestructura de hardware para futuras iteraciones. La optimización de los recursos de hardware permitiría una evaluación más rápida y efectiva y aceleraría el proceso de desarrollo del modelo. La reducción del tiempo de entrenamiento permitiría la iteración más rápida y la mejora continua del modelo.

Recomendaciones

Se aconseja ampliar la cantidad de imágenes de señales de tránsito, especialmente las de "Ceda el paso". El modelo de red neuronal podrá aprender mejor las características distintivas de cada señal con una base de datos más amplia y variada, lo que aumentará la precisión y robustez del modelo. Actualmente, la limitación de la cantidad de datos para ciertas señales puede causar errores de clasificación, lo que reduce la eficacia del modelo en entornos reales.

El hardware utilizado en el entrenamiento del modelo debe optimizarse para aumentar la eficiencia y reducir el tiempo de procesamiento. Se recomienda probar tarjetas gráficas como las

NVIDIA A100. Este tipo de hardware reduce el tiempo de procesamiento del modelo al procesar más datos, lo que facilita el desarrollo y ajuste de modelos más complejos y precisos.

De igual manera se podría considerar el continuar con el entrenamiento del modelo por más épocas. Esto por los resultados obtenidos durante las iteraciones realizadas en este trabajo, que muestran una tendencia al alta en la precisión y valores más bajos de pérdida, lo que indicaría que el modelo aún posee un margen de mejora con los parámetros actuales.

BIBLIOGRAFÍA

Flores Calero|, M. J., Conlago, C., Yunda, J., Aldás, M., & Flores, C. (2018). Implementación de un algoritmo para la detección de señales de tránsito del Ecuador: Pare, Ceda el paso y Velocidad. *Ingenius*, *20*, 9–20. https://doi.org/10.17163/ings.n20.2018.01

Flores-Calero, M., Conlago, C., Yunda, J., Aldas, M., & Flores, C. (2018). Localización y reconocimiento de señales de tráfico del Ecuador: Casos Pare y Ceda el Paso. *ACI Avances En Ciencias E Ingenierías*, *especial*(1). https://doi.org/10.18272/aci.v10i1.1012

Hotz, N. (2021, May 14). *SEMMA*. Data Science Process Alliance. https://www.datascience-pm.com/semma/

IBM. (n.d.-a). *¿Qué es Deep Learning?* | *IBM*. Www.ibm.com. https://www.ibm.com/es-es/topics/deep-learning

IBM. (n.d.-b). *¿Qué es el aprendizaje supervisado?* | *IBM*. Www.ibm.com. https://www.ibm.com/mx-es/topics/supervised-learning

IBM. (n.d.-c). *¿Qué son las redes neuronales convolucionales?* | *IBM*. Www.ibm.com. Retrieved March 20, 2024, from https://www.ibm.com/mx-es/topics/convolutional-neural-networks

IBM. (2021, August 17). *Introducción al CRISP-DM*. Www.ibm.com. https://www.ibm.com/docs/es/spss-modeler/saas?topic=guide-introduction-crisp-dm

López-Saca, F., Aviles-Cruz, C., & Ferreyra, A. (2019). Clasificación de imágenes usando redes neuronales convolucionales. *Zaloamati.azc.uam.mx*. http://zaloamati.azc.uam.mx/handle/11191/6123?show=full

Microsoft. (n.d.). *¿Qué es el Proceso de ciencia de datos en equipo (TDSP)? - Azure Architecture Center*. Learn.microsoft.com. Retrieved May 27, 2024, from https://learn.microsoft.com/es-es/azure/architecture/data-science-process/overview

Pona, M. (2022, September 28). *Las mejores librerías de Python para la ciencia de datos en 2022*. Maseldata.com. https://www.maseldata.com/post/python-librerias-data-science

Ponomareva, J., Senthil, J., & Cijov, A. (2020). *Road sign detection*. Www.kaggle.com. https://www.kaggle.com/datasets/andrewmvd/road-sign-detection/code

Ringa Tech. (2021, August 16). *Redes Neuronales Convolucionales - Clasificación avanzada de imágenes con IA / ML (CNN)*. Www.youtube.com. https://www.youtube.com/watch?v=4sWhhQwHqug

Roboflow. (2022, November). *Semaforos Object Detection Dataset by Practica1*. Roboflow. https://universe.roboflow.com/practica1/semaforos-njeuc

Tensorflow. (n.d.). *Introducción a TensorFlow*. TensorFlow. https://www.tensorflow.org/learn?hl=es-419

Toledo, J. (2023, July 28). *Origen y evolución de Python: la historia del lenguaje de programación*. Scriptología. https://scriptologia.com/origen-y-evolucion-de-python-la-historia-del-lenguaje-de-programacion/

ANEXOS

4. Cuaderno Python

. .

Pontificie Universidad Católica del Ecuador

Carrera de Ingeniería

Trabajo de titulación

Nombre: Sebastián Tene

Tema: Uso de modelos de aprendizaje profundo para el reconocimiento de señales de tránsito del Ecuador. Casos: Pare, Ceda el paso, Límites de velocidad y Semáforos

Fase 2. Comprensión de los datos

Recopilación de datos iniciales

A continuación se realiza el proceso de la carga de las imágenes en el entorno de Google Colab y su posterior exrtracción en las carpetas correspondientes

```
# Crear las carpetas para subir las imágenes de entrenamiento
!mkdir semaforos
!mkdir pare
!mkdir limiteVelocidad
!mkdir cedaPaso


# Descomprimir imagenes en las carpetas
%cd cedaPaso
!unzip cedaPaso.zip

%cd ..

%cd limiteVelocidad
!unzip limiteVelocidad.zip

%cd ..

%cd pare
!unzip Pare.zip

%cd ..

%cd semaforos
!unzip Semáforos.zip

%cd ..
```

```
Streaming output truncated to the last 5000 lines.
  inflating: cedaPaso/00034_00017.jpg
  inflating: cedaPaso/00034_00027.jpg
  inflating: cedaPaso/00034_00028.jpg
  inflating: cedaPaso/00035_00011.jpg
  inflating: cedaPaso/00035_00019.jpg
  inflating: cedaPaso/00035_00022.jpg
  inflating: cedaPaso/00036_00001.jpg
  inflating: cedaPaso/00036_00005.jpg
  inflating: cedaPaso/00036_00007.jpg
  inflating: cedaPaso/00036_00015.jpg
  inflating: cedaPaso/00036_00016.jpg
  inflating: cedaPaso/00036_00018.jpg
  inflating: cedaPaso/00036_00021.jpg
  inflating: cedaPaso/00036_00026.jpg
  inflating: cedaPaso/00036_00027.jpg
  inflating: cedaPaso/00037_00003.jpg
  inflating: cedaPaso/00037_00005.jpg
  inflating: cedaPaso/00037_00010.jpg
  inflating: cedaPaso/00037_00012.jpg
  inflating: cedaPaso/00037_00017.jpg
  inflating: cedaPaso/00037_00019.jpg
  inflating: cedaPaso/00037_00022.jpg
  inflating: cedaPaso/00037_00023.jpg
```

```
  inflating: cedaPaso/00037_00024.jpg
  inflating: cedaPaso/00037_00026.jpg
  inflating: cedaPaso/00038_00003.jpg
  inflating: cedaPaso/00038_00004.jpg
  inflating: cedaPaso/00038_00007.jpg
  inflating: cedaPaso/00038_00012.jpg
  inflating: cedaPaso/00038_00013.jpg
  inflating: cedaPaso/00038_00015.jpg
  inflating: cedaPaso/00038_00017.jpg
  inflating: cedaPaso/00038_00019.jpg
  inflating: cedaPaso/00038_00022.jpg
  inflating: cedaPaso/00038_00025.jpg
  inflating: cedaPaso/00038_00026.jpg
  inflating: cedaPaso/00038_00028.jpg
  inflating: cedaPaso/00039_00014.jpg
  inflating: cedaPaso/00039_00016.jpg
  inflating: cedaPaso/00039_00018.jpg
  inflating: cedaPaso/00039_00019.jpg
  inflating: cedaPaso/00039_00021.jpg
  inflating: cedaPaso/00039_00023.jpg
  inflating: cedaPaso/00039_00027.jpg
  inflating: cedaPaso/00039_00028.jpg
  inflating: cedaPaso/00039_00029.jpg
  inflating: cedaPaso/00040_00003.jpg
  inflating: cedaPaso/00040_00006.jpg
  inflating: cedaPaso/00040_00008.jpg
  inflating: cedaPaso/00040_00010.jpg
  inflating: cedaPaso/00040_00011.jpg
  inflating: cedaPaso/00040_00013.jpg
  inflating: cedaPaso/00040_00016.jpg
  inflating: cedaPaso/00040_00017.jpg
  inflating: cedaPaso/00040_00020.jpg
  inflating: cedaPaso/00040_00021.jpg
  inflating: cedaPaso/00040_00022.jpg
```

```
#Eliminar archivos zip
!rm -rf /content/cedaPaso/cedaPaso.zip
!rm -rf /content/limiteVelocidad/limiteVelocidad.zip
!rm -rf /content/pare/Pare.zip
!rm -rf /content/semaforos/Semáforos.zip
```

Preparar conjunto de datos de prueba

```
!mkdir semaforos_test
!mkdir pare_test
!mkdir limiteVelocidad_test
!mkdir cedaPaso_test
```

```
%cd pare_test
!unzip pare_test.zip
```

```
[Errno 2] No such file or directory: 'pare_test'
/content/pare_test
Archive:  pare_test.zip
  inflating: pare_test/00000_00003.jpg
  inflating: pare_test/00000_00014.jpg
  inflating: pare_test/00000_00015.jpg
  inflating: pare_test/00000_00020.jpg
  inflating: pare_test/00000_00021.jpg
  inflating: pare_test/00000_00026.jpg
  inflating: pare_test/00001_00017.jpg
  inflating: pare_test/00001_00020.jpg
  inflating: pare_test/00001_00021.jpg
  inflating: pare_test/00001_00022.jpg
  inflating: pare_test/00001_00026.jpg
  inflating: pare_test/00001_00028.jpg
  inflating: pare_test/00001_00029.jpg
  inflating: pare_test/00003_00015.jpg
  inflating: pare_test/00003_00020.jpg
  inflating: pare_test/00003_00021.jpg
  inflating: pare_test/00003_00023.jpg
  inflating: pare_test/00003_00024.jpg
  inflating: pare_test/00003_00028.jpg
  inflating: pare_test/00004_00018.jpg
  inflating: pare_test/00004_00020.jpg
  inflating: pare_test/00004_00022.jpg
  inflating: pare_test/00004_00025.jpg
  inflating: pare_test/00004_00026.jpg
  inflating: pare_test/00004_00027.jpg
  inflating: pare_test/00004_00028.jpg
```

```
  inflating: pare_test/00005_00022.jpg
  inflating: pare_test/00005_00027.jpg
  inflating: pare_test/00005_00028.jpg
  inflating: pare_test/00005_00029.jpg
  inflating: pare_test/00006_00019.jpg
  inflating: pare_test/00007_00017.jpg
  inflating: pare_test/00007_00018.jpg
  inflating: pare_test/00007_00023.jpg
  inflating: pare_test/00007_00025.jpg
  inflating: pare_test/00008_00025.jpg
  inflating: pare_test/00009_00017.jpg
  inflating: pare_test/00009_00021.jpg
  inflating: pare_test/00009_00024.jpg
  inflating: pare_test/00009_00028.jpg
  inflating: pare_test/00010_00003.jpg
  inflating: pare_test/00010_00006.jpg
  inflating: pare_test/00010_00009.jpg
  inflating: pare_test/00010_00014.jpg
  inflating: pare_test/00010_00017.jpg
  inflating: pare_test/00010_00018.jpg
  inflating: pare_test/00010_00020.jpg
  inflating: pare_test/00010_00021.jpg
  inflating: pare_test/00010_00025.jpg
  inflating: pare_test/00010_00027.jpg
  inflating: pare_test/00010_00028.jpg
  inflating: pare_test/00011_00021.jpg
  inflating: pare_test/00011_00024.jpg
  inflating: pare_test/00011_00025.jpg
  inflating: pare_test/00013_00010.jpg
```

```
# Descomprimir imagenes en las carpetas
%cd cedaPaso_test
!unzip cedaPaso_test.zip

%cd ..

%cd limiteVelocidad_test
!unzip limitesVelocidad_test.zip

%cd ..

%cd pare_test
!unzip pare_test.zip

%cd ..

%cd semaforos_test
!unzip semaforos_test.zip

%cd ..
```

```
/content/cedaPaso_test
Archive:  cedaPaso_test.zip
  inflating: cedaPaso_test/00003_00023.jpg
  inflating: cedaPaso_test/00003_00025.jpg
  inflating: cedaPaso_test/00004_00027.jpg
  inflating: cedaPaso_test/00005_00027.jpg
  inflating: cedaPaso_test/00006_00025.jpg
  inflating: cedaPaso_test/00008_00010.jpg
  inflating: cedaPaso_test/00009_00024.jpg
  inflating: cedaPaso_test/00009_00027.jpg
  inflating: cedaPaso_test/00010_00027.jpg
  inflating: cedaPaso_test/00012_00028.jpg
  inflating: cedaPaso_test/00013_00026.jpg
  inflating: cedaPaso_test/00014_00019.jpg
  inflating: cedaPaso_test/00014_00024.jpg
  inflating: cedaPaso_test/00014_00028.jpg
  inflating: cedaPaso_test/00016_00025.jpg
  inflating: cedaPaso_test/00016_00026.jpg
  inflating: cedaPaso_test/00016_00028.jpg
  inflating: cedaPaso_test/00017_00016.jpg
  inflating: cedaPaso_test/00018_00021.jpg
  inflating: cedaPaso_test/00018_00025.jpg
  inflating: cedaPaso_test/00018_00026.jpg
  inflating: cedaPaso_test/00019_00002.jpg
  inflating: cedaPaso_test/00019_00015.jpg
  inflating: cedaPaso_test/00019_00023.jpg
  inflating: cedaPaso_test/00019_00024.jpg
  inflating: cedaPaso_test/00019_00026.jpg
  inflating: cedaPaso_test/00019_00028.jpg
  inflating: cedaPaso_test/00020_00018.jpg
  inflating: cedaPaso_test/00020_00020.jpg
```

```
  inflating: cedaPaso_test/00020_00021.jpg
  inflating: cedaPaso_test/00020_00025.jpg
  inflating: cedaPaso_test/00020_00029.jpg
  inflating: cedaPaso_test/00022_00021.jpg
  inflating: cedaPaso_test/00022_00029.jpg
  inflating: cedaPaso_test/00024_00020.jpg
  inflating: cedaPaso_test/00025_00024.jpg
  inflating: cedaPaso_test/00025_00025.jpg
  inflating: cedaPaso_test/00030_00019.jpg
  inflating: cedaPaso_test/00035_00017.jpg
  inflating: cedaPaso_test/00035_00020.jpg
  inflating: cedaPaso_test/00035_00023.jpg
  inflating: cedaPaso_test/00035_00025.jpg
  inflating: cedaPaso_test/00035_00026.jpg
  inflating: cedaPaso_test/00035_00027.jpg
  inflating: cedaPaso_test/00035_00029.jpg
  inflating: cedaPaso_test/00037_00027.jpg
  inflating: cedaPaso_test/00038_00027.jpg
  inflating: cedaPaso_test/00039_00022.jpg
  inflating: cedaPaso_test/00041_00012.jpg
  inflating: cedaPaso_test/00041_00022.jpg
  inflating: cedaPaso_test/00041_00026.jpg
  inflating: cedaPaso_test/00045_00020.jpg
  inflating: cedaPaso_test/00047_00015.jpg
  inflating: cedaPaso_test/00048_00021.jpg
  inflating: cedaPaso_test/00048_00022.jpg
  inflating: cedaPaso_test/00048_00023.jpg
```

```
#Eliminar archivos zip
!rm -rf /content/cedaPaso_test/cedaPaso_test.zip
!rm -rf /content/limiteVelocidad_test/limitesVelocidad_test.zip
!rm -rf /content/pare_test/pare_test.zip
!rm -rf /content/semaforos_test/semaforos_test.zip
```

Exploración de los datos

```
# Previsualización de imagenes

import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

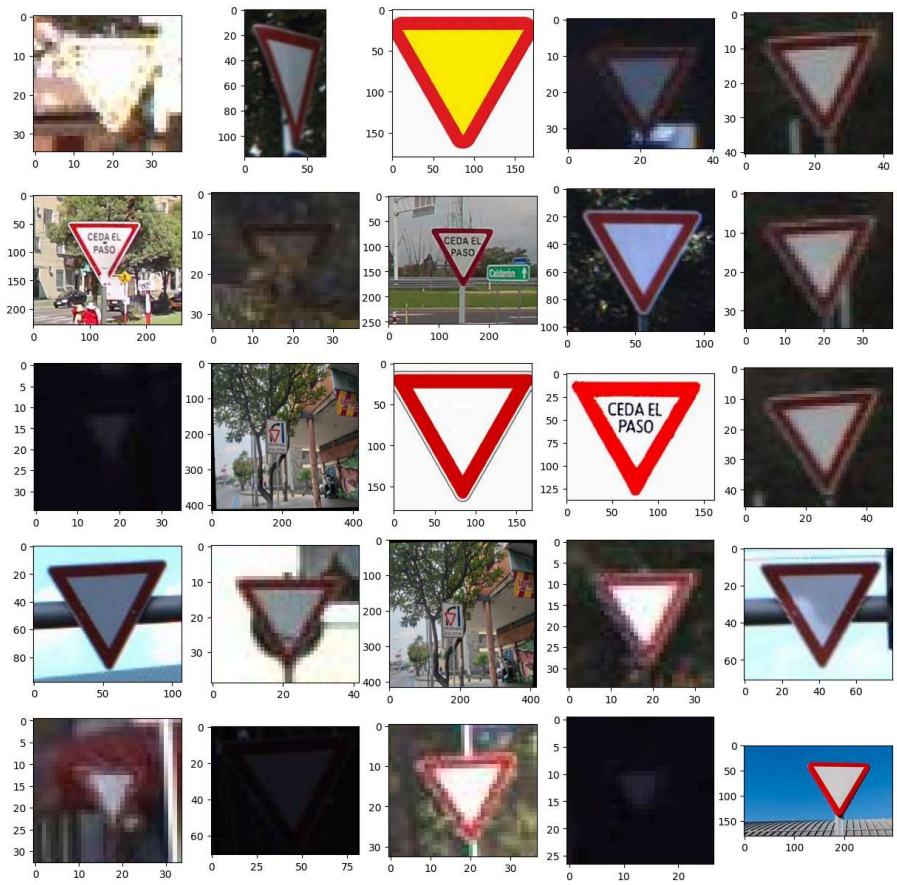
plt.figure(figsize=(15,15))
carpeta = '/content/limiteVelocidad/limiteVelocidad'
imagenes = os.listdir(carpeta)
for i, nombreimg in enumerate(imagenes[:25]):
  plt.subplot(5,5,i+1)
  imagen = mpimg.imread(carpeta + '/' + nombreimg)
  plt.imshow( imagen)
```



```
# Previsualización de imagenes

import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

plt.figure(figsize=(15,15))
carpeta = '/content/cedaPaso/cedaPaso'
imagenes = os.listdir(carpeta)
for i, nombreimg in enumerate(imagenes[:25]):
  plt.subplot(5,5,i+1)
  imagen = mpimg.imread(carpeta + '/' + nombreimg)
  plt.imshow( imagen)
```




```
# Previsualización de imagenes

import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

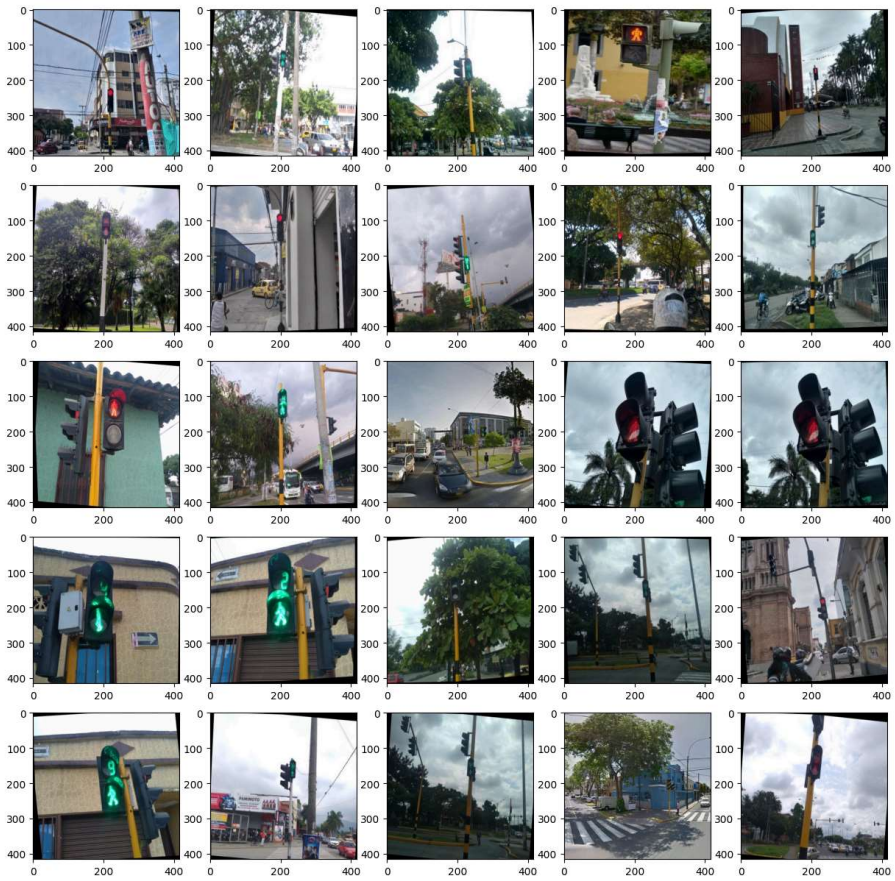
plt.figure(figsize=(15,15))
carpeta = '/content/pare/Pare'
imagenes = os.listdir(carpeta)
for i, nombreimg in enumerate(imagenes[:25]):
  plt.subplot(5,5,i+1)
  imagen = mpimg.imread(carpeta + '/' + nombreimg)
  plt.imshow( imagen)
```


PARE
STOP

```
# Previsualización de imagenes

import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

plt.figure(figsize=(15,15))
carpeta = '/content/semaforos/Semaforos'
imagenes = os.listdir(carpeta)
for i, nombreimg in enumerate(imagenes[:25]):
  plt.subplot(5,5,i+1)
  imagen = mpimg.imread(carpeta + '/' + nombreimg)
  plt.imshow( imagen)
```



Fase 3. Preparación de los datos

A continuación se realiza la prepración de los datos para poder entrenar el modelo. En el cuaderno no se evidencia la limpieza ni la selección de datos debido a que esto fue realizado de manera manual antes de cargar los datos

```
# Crear carpeta para el dataset de entrenamiento y validación
!mkdir dataset
!mkdir dataset/pare
!mkdir dataset/cedaPaso
!mkdir dataset/limiteVelocidad
!mkdir dataset/semaforos


#Copiar imagenes de Pare
import shutil

fuente = '/content/pare/Pare'
destino = '/content/dataset/pare'

imagenes = os.listdir(fuente)
for i, nombreimg in enumerate(imagenes):
    shutil.copy(fuente + '/' + nombreimg, destino + '/' + nombreimg)


#Copiar imagenes ceda el paso
import shutil

fuente = '/content/cedaPaso/cedaPaso'
destino = '/content/dataset/cedaPaso'

imagenes = os.listdir(fuente)
for i, nombreimg in enumerate(imagenes):
    shutil.copy(fuente + '/' + nombreimg, destino + '/' + nombreimg)


#Copiar imagenes límites de velocidad
import shutil

fuente = '/content/limiteVelocidad/limiteVelocidad'
destino = '/content/dataset/limiteVelocidad'

imagenes = os.listdir(fuente)
for i, nombreimg in enumerate(imagenes):
    shutil.copy(fuente + '/' + nombreimg, destino + '/' + nombreimg)


#Copiar imagenes semáforos
import shutil

fuente = '/content/semaforos/Semaforos'
destino = '/content/dataset/semaforos'

imagenes = os.listdir(fuente)
for i, nombreimg in enumerate(imagenes):
    shutil.copy(fuente + '/' + nombreimg, destino + '/' + nombreimg)
```

Datos de prueba

```
# Crear carpeta para el dataset de entrenamiento y validación
!mkdir test
!mkdir test/pare
!mkdir test/cedaPaso
!mkdir test/limiteVelocidad
!mkdir test/semaforos


#Copiar imagenes de Pare
import shutil

fuente = '/content/pare_test/pare_test'
destino = '/content/test/pare'

imagenes = os.listdir(fuente)
for i, nombreimg in enumerate(imagenes):
    shutil.copy(fuente + '/' + nombreimg, destino + '/' + nombreimg)
```

```
#Copiar imagenes ceda el paso
import shutil

fuente = '/content/cedaPaso_test/cedaPaso_test'
destino = '/content/test/cedaPaso'

imagenes = os.listdir(fuente)
for i, nombreimg in enumerate(imagenes):
    shutil.copy(fuente + '/' + nombreimg, destino + '/' + nombreimg)


#Copiar imagenes límites de velocidad
import shutil

fuente = '/content/limiteVelocidad_test/limitesVelocidad_test'
destino = '/content/test/limiteVelocidad'

imagenes = os.listdir(fuente)
for i, nombreimg in enumerate(imagenes):
    shutil.copy(fuente + '/' + nombreimg, destino + '/' + nombreimg)


#Copiar imagenes semáforos
import shutil

fuente = '/content/semaforos_test/semaforos_test'
destino = '/content/test/semaforos'

imagenes = os.listdir(fuente)
for i, nombreimg in enumerate(imagenes):
    shutil.copy(fuente + '/' + nombreimg, destino + '/' + nombreimg)


pip install --upgrade tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0)
Collecting tensorflow
  Downloading tensorflow-2.16.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (589.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 589.8/589.8 MB 1.3 MB/s eta 0:00:00
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Collecting ml-dtypes~=0.3.1 (from tensorflow)
  Downloading ml_dtypes-0.3.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 852.4 kB/s eta 0:00:00
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/pyt
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.31.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.11.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.0)
Collecting tensorboard<2.17,>=2.16 (from tensorflow)
  Downloading tensorboard-2.16.2-py3-none-any.whl (5.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.5/5.5 MB 7.2 MB/s eta 0:00:00
Collecting keras>=3.0.0 (from tensorflow)
  Downloading keras-3.3.3-py3-none-any.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 7.2 MB/s eta 0:00:00
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.3
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.0.0->tensorflow) (13.7.1)
Collecting namex (from keras>=3.0.0->tensorflow)
  Downloading namex-0.0.8-py3-none-any.whl (5.8 kB)
Collecting optree (from keras>=3.0.0->tensorflow)
  Downloading optree-0.11.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (311 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 311.2/311.2 kB 6.6 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorf
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.17,>=2.16->tensorflow)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.1
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.17,>=2.16->tensorflow)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.17,>
```

```
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.0.0->tensorflow)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.0.0->tensorflo
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.0.0-
Installing collected packages: namex, optree, ml-dtypes, tensorboard, keras, tensorflow
  Attempting uninstall: ml-dtypes
    Found existing installation: ml-dtypes 0.2.0
    Uninstalling ml-dtypes-0.2.0:
      Successfully uninstalled ml-dtypes-0.2.0
  Attempting uninstall: tensorboard
```

Construcción de nuevos datos

Para crear los conjuntos de entrenamiento, validación y prueba se utilizará la bilbioteca de flow_from_directory la cual nos permite crear los conjuntos de datos con las etiquetas de la carpeta a la que pertenecen. Dando así la salida que necesitaría el modelo predecir

De igual manera se aplica el aumento de datos con la intención de darle al modelo mayor resistencia al over fitting

```
#Aumento de datos con ImageDataGenerator
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

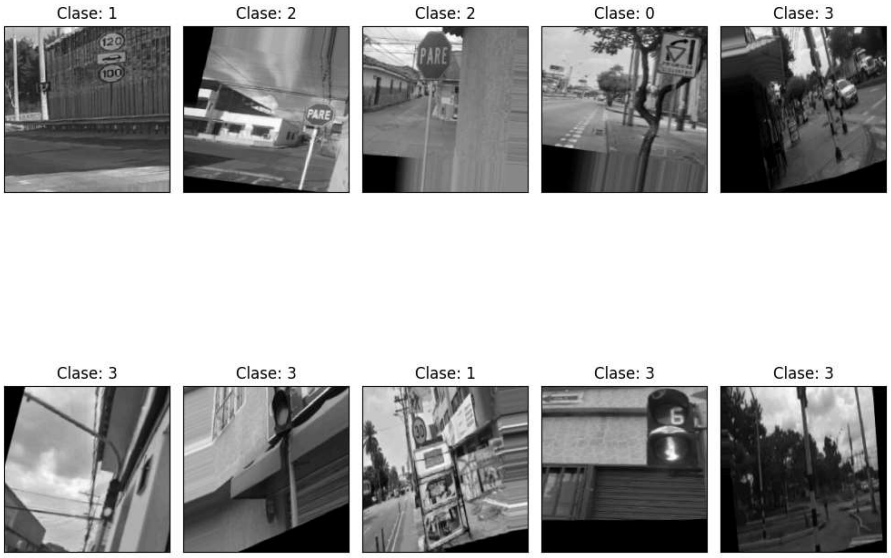
# Crear una el dataset de entrenamiento
datagen = ImageDataGenerator(
    rescale=1. / 255,           # Escalar los valores de los píxeles a [0, 1]
    rotation_range=15,          # Rotar las imágenes al azar en un rango de 15 grados
    width_shift_range=0.25,     # Desplazar las imágenes horizontalmente hasta un 25% del ancho
    height_shift_range=0.25,    # Desplazar las imágenes verticalmente hasta un 25% de la altura
    shear_range=10,             # Aplicar transformaciones de cizalladura a las imágenes
    zoom_range=[0.5, 1.25],     # Aplicar zoom aleatorio a las imágenes en el rango de 50% a 125%
    validation_split=0.2        # Reservar el 20% de los datos para la validación
)

#Generadores para sets de entrenamiento y validación
data_gen_entrenamiento=datagen.flow_from_directory('/content/dataset',target_size=(224,224),batch_size=32,color_mode='grayscale',class_mode=
data_gen_pruebas=datagen.flow_from_directory('/content/dataset',target_size=(224,224),batch_size=32,color_mode='grayscale',class_mode='categ

# Imprimir 10 imágenes de entrenamiento
fig, axes = plt.subplots(2, 5, figsize=(10, 10))  # Ajustar el tamaño de la figura

for imagen, etiqueta in data_gen_entrenamiento:
    for i in range(10):
        plt.subplot(2, 5, i + 1)  # Crear una cuadrícula de 2x5 para las imágenes
        plt.xticks([])
        plt.yticks([])
        plt.imshow(imagen[i].squeeze(), cmap='gray')
        plt.title(f'Clase: {np.argmax(etiqueta[i])}')
    break
plt.tight_layout()
plt.show()
```

```
Found 4255 images belonging to 4 classes.
Found 1063 images belonging to 4 classes.
```



Preparación datos de prueba

En este paso se realiza pasos similares que para los datos de validación que los de entrenamiento. Con la diferencia que no es necesario realizar aumento de datos, solo el era necesario el reescalar los pixeles y el tamaño de las imágenes

```
test_path = '/content/test'

# Crear un ImageDataGenerator para el conjunto de prueba
test_datagen = ImageDataGenerator(rescale=1. / 255)

# Crear un generador de datos para el conjunto de prueba
test_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(224, 224),
    batch_size=32,
    color_mode='grayscale',
    class_mode='categorical',
    shuffle=False
)
```

```
Found 400 images belonging to 4 classes.
```

Fase 4. Modelado

Generación de los modelos

Se realizaron 3 modelos para la comparación. El primero una red neuronal tradicional. El segundo una red convolucional sin regularización. Y el tercero una red convolucional con regularización

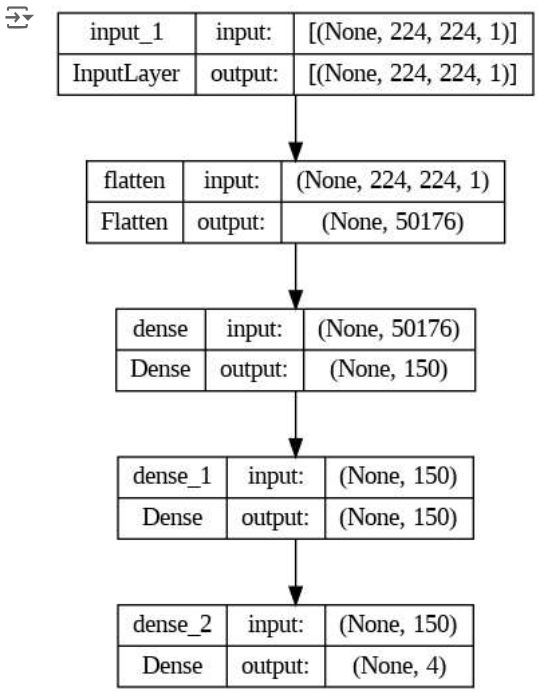
```
import tensorflow as tf

modeloDenso = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(224,224,1)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(150, activation='relu'),
    tf.keras.layers.Dense(150, activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax')
])

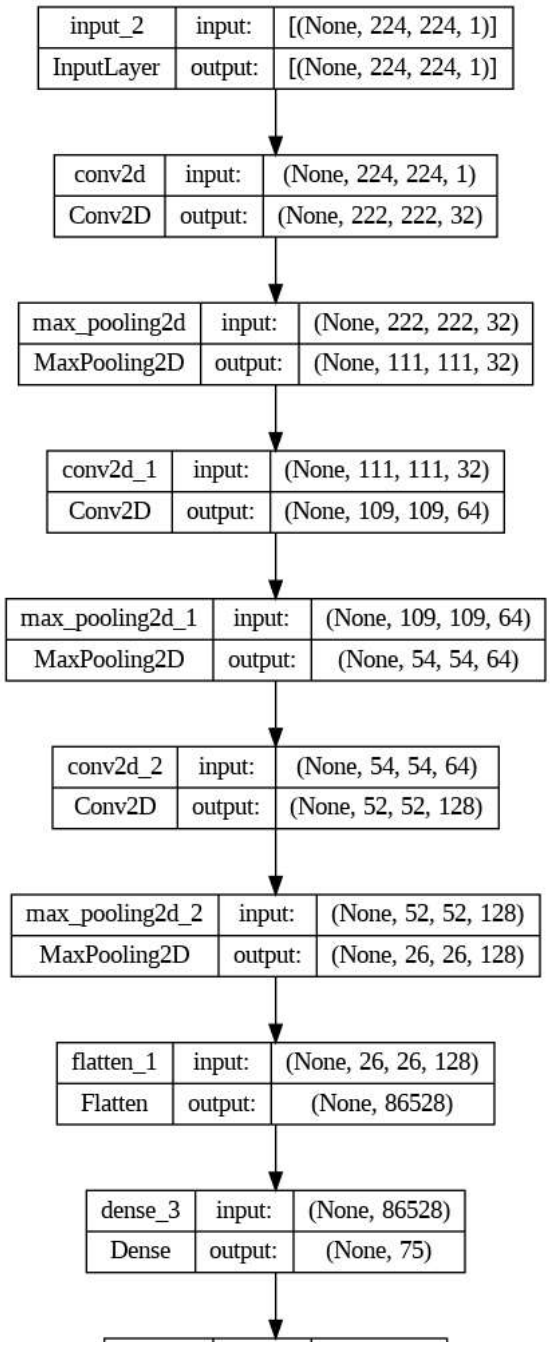
modeloCNN = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(224,224,1)),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(75, activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax'),
])

modeloCNN2 = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(224,224,1)),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(75, activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax'),
])
```

```
from tensorflow.keras.utils import plot_model
# Generar la representación gráfica del modelo
plot_model(modeloDenso, to_file='modeloDenso.png', show_shapes=True, show_layer_names=True)
```




```
# Generar la representación gráfica del modelo
plot_model(modeloCNN, to_file='modeloCNN.png', show_shapes=True, show_layer_names=True)
```




```
# Compilar el modelo denso
modeloDenso.compile(
    optimizer='adam',                  # Usar el optimizador Adam para ajustar los pesos del modelo
    loss='categorical_crossentropy',   # Usar la función de pérdida de entropía cruzada categórica
    metrics=['accuracy']               # Evaluar el modelo usando accuracy
)

# Compilar el modelo convolucional sin regularización
modeloCNN.compile(
    optimizer='adam',                  # Usar el optimizador Adam para ajustar los pesos del modelo
    loss='categorical_crossentropy',   # Usar la función de pérdida de entropía cruzada categórica
    metrics=['accuracy']               # Evaluar el modelo usando accuracy
)

# Compilar el el modelo convolucional con regularización
modeloCNN2.compile(
    optimizer='adam',                  # Usar el optimizador Adam para ajustar los pesos del modelo
    loss='categorical_crossentropy',   # Usar la función de pérdida de entropía cruzada categórica
    metrics=['accuracy']               # Evaluar el modelo usando accuracy
)


from tensorflow.keras.callbacks import TensorBoard
```

Entrenamiento modelo denso

Para el modelo tradicional o denso se usarán 100 épocas con los conjuntos de entrenamiento y validación

```
tensorboardDenso = TensorBoard(log_dir='logs/denso')
modeloDenso.fit(
    data_gen_entrenamiento,
    validation_data=data_gen_pruebas,
    epochs=100,
    callbacks=[tensorboardDenso]
)
```

```
Epoch 1/100
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cla
  self._warn_if_super_not_called()
133/133 ━━━━━━━━━━━━━━━━━━━━ 30s 207ms/step - accuracy: 0.3328 - loss: 3.8135 - val_accuracy: 0.1994 - val_loss: 1.5942
Epoch 2/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.3808 - loss: 1.5315 - val_accuracy: 0.4779 - val_loss: 1.2785
Epoch 3/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 202ms/step - accuracy: 0.4185 - loss: 1.2865 - val_accuracy: 0.4901 - val_loss: 1.2172
Epoch 4/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 204ms/step - accuracy: 0.4214 - loss: 1.2972 - val_accuracy: 0.4130 - val_loss: 1.2777
Epoch 5/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4380 - loss: 1.2571 - val_accuracy: 0.4346 - val_loss: 1.2229
Epoch 6/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 204ms/step - accuracy: 0.4487 - loss: 1.2492 - val_accuracy: 0.4365 - val_loss: 1.2595
Epoch 7/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4294 - loss: 1.2378 - val_accuracy: 0.4271 - val_loss: 1.3763
Epoch 8/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 204ms/step - accuracy: 0.4561 - loss: 1.2165 - val_accuracy: 0.4196 - val_loss: 1.2504
Epoch 9/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4466 - loss: 1.2096 - val_accuracy: 0.4186 - val_loss: 1.3083
Epoch 10/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4566 - loss: 1.2260 - val_accuracy: 0.4102 - val_loss: 1.2847
Epoch 11/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4139 - loss: 1.3045 - val_accuracy: 0.3547 - val_loss: 1.2827
Epoch 12/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 204ms/step - accuracy: 0.4488 - loss: 1.2274 - val_accuracy: 0.4337 - val_loss: 1.2576
Epoch 13/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 204ms/step - accuracy: 0.4685 - loss: 1.1865 - val_accuracy: 0.4704 - val_loss: 1.2276
Epoch 14/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 198ms/step - accuracy: 0.4611 - loss: 1.2069 - val_accuracy: 0.4186 - val_loss: 1.2877
Epoch 15/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 201ms/step - accuracy: 0.4421 - loss: 1.1944 - val_accuracy: 0.3913 - val_loss: 1.2886
Epoch 16/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 204ms/step - accuracy: 0.4605 - loss: 1.1896 - val_accuracy: 0.3885 - val_loss: 1.2997
Epoch 17/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 202ms/step - accuracy: 0.4738 - loss: 1.1732 - val_accuracy: 0.4516 - val_loss: 1.2268
Epoch 18/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4702 - loss: 1.1880 - val_accuracy: 0.4403 - val_loss: 1.2613
Epoch 19/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4650 - loss: 1.1788 - val_accuracy: 0.3650 - val_loss: 1.2902
Epoch 20/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 205ms/step - accuracy: 0.4495 - loss: 1.2075 - val_accuracy: 0.4064 - val_loss: 1.3257
Epoch 21/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 204ms/step - accuracy: 0.4547 - loss: 1.2092 - val_accuracy: 0.3659 - val_loss: 1.2865
Epoch 22/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4304 - loss: 1.2191 - val_accuracy: 0.4901 - val_loss: 1.2372
Epoch 23/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4686 - loss: 1.1810 - val_accuracy: 0.5033 - val_loss: 1.2016
Epoch 24/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4424 - loss: 1.1904 - val_accuracy: 0.5165 - val_loss: 1.1679
Epoch 25/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4507 - loss: 1.2118 - val_accuracy: 0.5183 - val_loss: 1.1755
Epoch 26/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 204ms/step - accuracy: 0.4407 - loss: 1.2058 - val_accuracy: 0.5174 - val_loss: 1.2276
Epoch 27/100
133/133 ━━━━━━━━━━━━━━━━━━━━ 28s 203ms/step - accuracy: 0.4556 - loss: 1.2074 - val_accuracy: 0.4976 - val_loss: 1.1766
Epoch 28/100
```

```
%load_ext tensorboard
```

```
%tensorboard --logdir logs/
```

Como es posible observar después de las 100 épocas el modelo no representa un alto porcentaje de precisión. En sus mejores iteraciones llega a un 0.5 lo cual no puede ser considerado como válido

Cambio en los datos de entrenamiento.

Para la redes convolucionales se tiene un mayor número de capas que capas, por lo que un tamaño de batch muy pequeño puede resultar un en problema para los tiempos de procesamiento, por lo cual se llevó a elegir un tamaño mayor

```
#Generadores para sets de entrenamiento y validación
data_gen_entrenamiento=datagen.flow_from_directory('/content/dataset',target_size=(224,224),batch_size=96,color_mode='grayscale',class_mode=
data_gen_pruebas=datagen.flow_from_directory('/content/dataset',target_size=(224,224),batch_size=96,color_mode='grayscale',class_mode='categ
```

```
Found 4255 images belonging to 4 classes.
Found 1063 images belonging to 4 classes.
```

Entrenamiento modelo convolucional sin regularización

A diferencia del anterior modelo una red convolucional necesita una mayor cantidad de épocas para que pueda mostrar resultados. De este modo se destinaron 300 épocas de entrenamiento con los mismos conjuntos de dato de entrenamiento y validación

```
tensorboardCNN=TensorBoard(log_dir='logs/CNN')
modeloCNN.fit(
    data_gen_entrenamiento,
    validation_data=data_gen_pruebas,
    epochs=300,
    callbacks=[tensorboardCNN]
)
```

```
Epoch 1/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.6699 - loss: 0.8287 - val_accuracy: 0.5230 - val_loss: 1.2141
Epoch 2/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.6753 - loss: 0.8045 - val_accuracy: 0.4986 - val_loss: 1.2020
Epoch 3/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.6815 - loss: 0.8041 - val_accuracy: 0.5118 - val_loss: 1.1902
Epoch 4/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.7090 - loss: 0.7469 - val_accuracy: 0.5437 - val_loss: 1.1608
Epoch 5/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.7046 - loss: 0.7532 - val_accuracy: 0.5466 - val_loss: 1.1114
Epoch 6/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7173 - loss: 0.7098 - val_accuracy: 0.4835 - val_loss: 1.2048
Epoch 7/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7208 - loss: 0.7201 - val_accuracy: 0.5861 - val_loss: 1.0592
Epoch 8/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7257 - loss: 0.6867 - val_accuracy: 0.4995 - val_loss: 1.2597
Epoch 9/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7321 - loss: 0.6857 - val_accuracy: 0.5061 - val_loss: 1.1877
Epoch 10/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.7457 - loss: 0.6578 - val_accuracy: 0.5089 - val_loss: 1.1447
Epoch 11/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7508 - loss: 0.6588 - val_accuracy: 0.5710 - val_loss: 1.1846
Epoch 12/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7685 - loss: 0.6140 - val_accuracy: 0.5466 - val_loss: 1.1205
Epoch 13/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7707 - loss: 0.6069 - val_accuracy: 0.5230 - val_loss: 1.1368
Epoch 14/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7698 - loss: 0.6028 - val_accuracy: 0.6049 - val_loss: 1.0006
Epoch 15/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.7726 - loss: 0.5957 - val_accuracy: 0.5400 - val_loss: 1.1000
Epoch 16/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7815 - loss: 0.5824 - val_accuracy: 0.5720 - val_loss: 1.0606
Epoch 17/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7821 - loss: 0.5744 - val_accuracy: 0.6237 - val_loss: 0.9654
Epoch 18/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.7939 - loss: 0.5442 - val_accuracy: 0.5917 - val_loss: 1.0935
Epoch 19/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.8060 - loss: 0.5100 - val_accuracy: 0.6002 - val_loss: 1.0722
Epoch 20/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.8009 - loss: 0.5233 - val_accuracy: 0.5992 - val_loss: 0.9991
Epoch 21/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.8094 - loss: 0.5015 - val_accuracy: 0.5663 - val_loss: 1.1267
Epoch 22/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.8100 - loss: 0.4910 - val_accuracy: 0.5908 - val_loss: 1.0780
Epoch 23/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.8219 - loss: 0.4772 - val_accuracy: 0.5569 - val_loss: 1.1665
Epoch 24/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.8312 - loss: 0.4682 - val_accuracy: 0.6388 - val_loss: 0.9806
Epoch 25/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.8256 - loss: 0.4744 - val_accuracy: 0.6754 - val_loss: 0.8957
Epoch 26/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.8316 - loss: 0.4569 - val_accuracy: 0.6632 - val_loss: 0.9580
Epoch 27/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.8406 - loss: 0.4301 - val_accuracy: 0.6623 - val_loss: 0.9755
Epoch 28/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.8270 - loss: 0.4691 - val_accuracy: 0.6736 - val_loss: 0.8648
Epoch 29/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 59s 1s/step - accuracy: 0.8433 - loss: 0.4269 - val_accuracy: 0.5936 - val_loss: 1.0939
```

```
%tensorboard --logdir logs/
```



TensorBoard

En este modelo si se evidencia una una mejora significativa con respecto a la red tradicional. Como se puede evidenciar este modelo ya llega a una precisión de 0.80 con los de validación. Sin embargo, parece ser que el modelo no puede superar ese margen debido a que en las últimas épocas ya se mantiene entre los mismos valores

Modelo de red convolucional con regularización

Este modelo ya es el más robusto destinado para este estudio. Se coloca medidas de regularización como el dropout para ayudar al modelo a no tener neuronas muertas y obtener resultados superiores al resto de modelos

```
tensorboardCNN2=TensorBoard(log_dir='logs/CNN2')
modeloCNN2.fit(
    data_gen_entrenamiento,
    validation_data=data_gen_pruebas,
    epochs=300,
    callbacks=[tensorboardCNN2]
)
```

```
Epoch 1/300
/usr/local/lib/python3.10/dist-packages/PIL/Image.py:1000: UserWarning: Palette images w
  warnings.warn(
45/45 ━━━━━━━━━━━━━━━━━━━━ 64s 1s/step - accuracy: 0.3659 - loss: 1.4722 - val_accuracy:
Epoch 2/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.5103 - loss: 1.1283 - val_accuracy:
Epoch 3/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.5367 - loss: 1.0763 - val_accuracy:
Epoch 4/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.5830 - loss: 1.0157 - val_accuracy:
Epoch 5/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.5939 - loss: 0.9844 - val_accuracy:
Epoch 6/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.6147 - loss: 0.9578 - val_accuracy:
Epoch 7/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.6356 - loss: 0.9137 - val_accuracy:
Epoch 8/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.6439 - loss: 0.8831 - val_accuracy:
Epoch 9/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.6612 - loss: 0.8434 - val_accuracy:
Epoch 10/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.6956 - loss: 0.7945 - val_accuracy:
Epoch 11/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.6967 - loss: 0.8042 - val_accuracy:
Epoch 12/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.7032 - loss: 0.7575 - val_accuracy:
Epoch 13/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.7046 - loss: 0.7519 - val_accuracy:
Epoch 14/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.7386 - loss: 0.6739 - val_accuracy:
Epoch 15/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.7463 - loss: 0.6578 - val_accuracy:
Epoch 16/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.7492 - loss: 0.6503 - val_accuracy:
Epoch 17/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.7587 - loss: 0.6298 - val_accuracy:
Epoch 18/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.7674 - loss: 0.6161 - val_accuracy:
Epoch 19/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.7688 - loss: 0.6040 - val_accuracy:
Epoch 20/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 63s 1s/step - accuracy: 0.7706 - loss: 0.6072 - val_accuracy:
Epoch 21/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 63s 1s/step - accuracy: 0.7938 - loss: 0.5461 - val_accuracy:
Epoch 22/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.8061 - loss: 0.5111 - val_accuracy:
Epoch 23/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 63s 1s/step - accuracy: 0.7987 - loss: 0.5452 - val_accuracy:
Epoch 24/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.8035 - loss: 0.5261 - val_accuracy:
Epoch 25/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.8112 - loss: 0.5098 - val_accuracy:
Epoch 26/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.8115 - loss: 0.5052 - val_accuracy:
Epoch 27/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 63s 1s/step - accuracy: 0.8141 - loss: 0.4884 - val_accuracy:
Epoch 28/300
```

Se interrumpe el modelo con regularización ya que sus tasa de aprendizaje son muy similares a las del modelo sin regularización y parece quedar estancado en el rango d 74%-77% de precisión por lo que se lo replantea otra vez a continuación

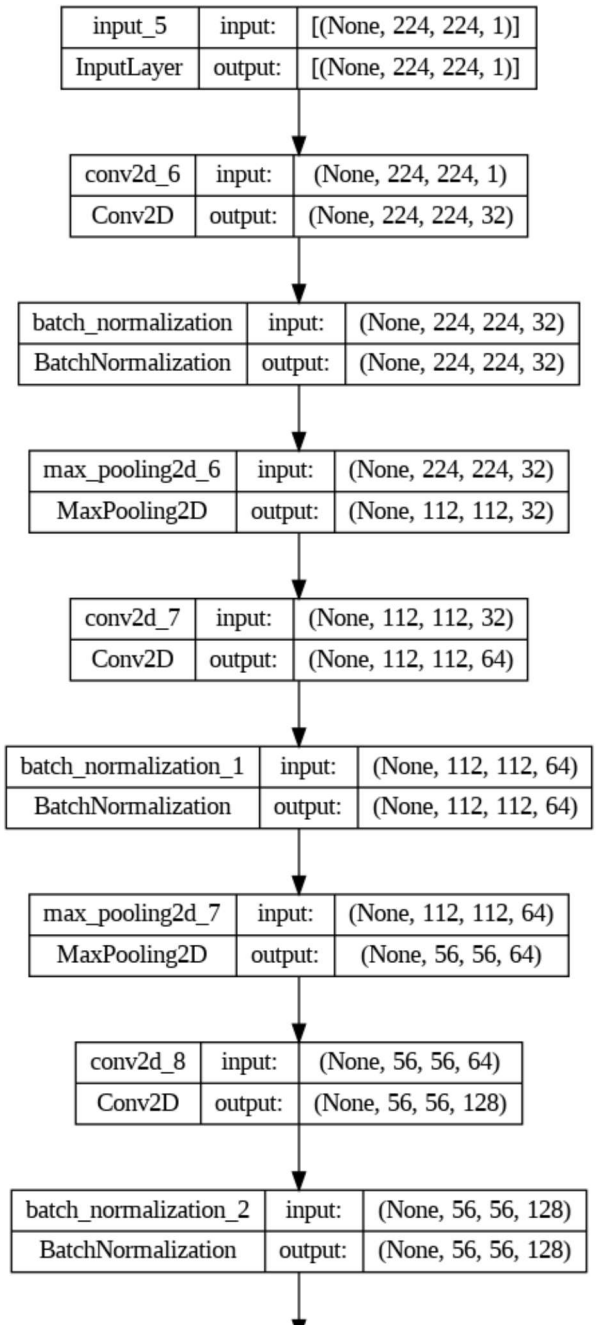
```
45/45 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.8416 - loss: 0.4443 - val accuracy:
```

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dropout, Flatten, Dense, BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam

modeloCNN2 = Sequential([
    Input(shape=(224, 224, 1)),
    Conv2D(32, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Conv2D(256, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Dropout(0.5),
    Flatten(),
    Dense(128, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(4, activation='softmax'),
])

     Epoch 37/300

# Generar la representación gráfica del modelo
plot_model(modeloCNN2, to_file='modeloCNN2.png', show_shapes=True, show_layer_names=True)
```




```
# Configuración del optimizador con un learning rate scheduler
modeloCNN2.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

tensorboardCNN2=TensorBoard(log_dir='logs/CNN2')
modeloCNN2.fit(
    data_gen_entrenamiento,
    validation_data=data_gen_pruebas,
    epochs=300,
    callbacks=[tensorboardCNN2]
)

# Resumen del modelo
modeloCNN2.summary()
```

```
Epoch 1/300
 7/45 ━━━━━━━━━━━━━━━━━━━━ 1:49 3s/step - accuracy: 0.3274 - loss: 14.3576/usr/local/lib
  warnings.warn(
45/45 ━━━━━━━━━━━━━━━━━━━━ 141s 3s/step - accuracy: 0.4039 - loss: 9.2399 - val_accuracy
Epoch 2/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.4873 - loss: 4.8872 - val_accuracy
Epoch 3/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.5178 - loss: 3.8222 - val_accuracy
Epoch 4/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 137s 3s/step - accuracy: 0.5375 - loss: 3.0949 - val_accuracy
Epoch 5/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 139s 3s/step - accuracy: 0.5493 - loss: 2.4427 - val_accuracy
Epoch 6/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 139s 3s/step - accuracy: 0.5656 - loss: 1.9936 - val_accuracy
Epoch 7/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 138s 3s/step - accuracy: 0.5677 - loss: 1.7868 - val_accuracy
Epoch 8/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 137s 3s/step - accuracy: 0.5958 - loss: 1.5975 - val_accuracy
Epoch 9/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 138s 3s/step - accuracy: 0.5847 - loss: 1.5392 - val_accuracy
Epoch 10/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 138s 3s/step - accuracy: 0.6048 - loss: 1.4310 - val_accuracy
Epoch 11/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.5872 - loss: 1.4075 - val_accuracy
Epoch 12/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.5968 - loss: 2.0909 - val_accuracy
Epoch 13/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.5841 - loss: 2.0340 - val_accuracy
Epoch 14/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.5937 - loss: 1.6924 - val_accuracy
Epoch 15/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.6099 - loss: 1.4235 - val_accuracy
Epoch 16/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.6016 - loss: 1.2699 - val_accuracy
Epoch 17/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.6430 - loss: 1.1690 - val_accuracy
Epoch 18/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 135s 3s/step - accuracy: 0.6331 - loss: 1.1322 - val_accuracy
Epoch 19/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.6507 - loss: 1.2376 - val_accuracy
Epoch 20/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.6318 - loss: 1.5525 - val_accuracy
Epoch 21/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 135s 3s/step - accuracy: 0.6365 - loss: 1.9591 - val_accuracy
Epoch 22/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 135s 3s/step - accuracy: 0.6692 - loss: 1.5140 - val_accuracy
Epoch 23/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.6913 - loss: 1.2508 - val_accuracy
Epoch 24/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.7028 - loss: 1.2129 - val_accuracy
Epoch 25/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.7405 - loss: 1.1601 - val_accuracy
Epoch 26/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.7286 - loss: 1.2417 - val_accuracy
Epoch 27/300
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.7357 - loss: 1.2995 - val_accuracy
Epoch 28/300
```

```
%tensorboard --logdir logs/
```



TensorBoard

Este modelo, ya con las modificaciones muestra un desempeño mayor que los otros dos modelos. Llegando a un 0.91 de precisión con los datos de validación. De igual manera, este modelo todavía tiene capacidad de crecimiento ya que con las diferentes regularizaciones este presenta fluctuaciones. Sin embargo, la tendencia del modelo seguí en crecimeinto, lo que indica que se podría obtener mejores resultados con más épocas de entrenamiento

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.9206 - loss: 0.9414 - val_accuracy
```

Evaluación del modelo

Una vez se observó el desempeño de los 3 modelos se escoge el modelo con regularización ya que posee el mejor desempeño. A continuación se lo va a someter a los datos de prueba para obtener una precisión más real

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.9411 - loss: 0.8222 - val_accuracy
```

```
# Obtener las predicciones del modelo para el conjunto de prueba
predictions = modeloCNN2.predict(test_generator)

# Convertir las predicciones a etiquetas de clase
predicted_classes = np.argmax(predictions, axis=1)

# Obtener las etiquetas verdaderas
true_classes = test_generator.classes

# Obtener los nombres de las clases
class_labels = list(test_generator.class_indices.keys())

# Evaluar el modelo y mostrar precisión en el conjunto de prueba
test_loss, test_accuracy = modeloCNN2.evaluate(test_generator)
print(f'Loss en el conjunto de prueba: {test_loss}')
print(f'Accuracy en el conjunto de prueba: {test_accuracy}')
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 115ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 114ms/step - accuracy: 0.8800 - loss: 1.1384
Loss en el conjunto de prueba: 1.7527399063110352
Accuracy en el conjunto de prueba: 0.8149999976158142

Epoch 87/300
```

Se realizaron dos pruebas de precisión para el modelo. La primera corresponde a realizar directamente las predicciones sobre el conjunto de prueba y comparar con los etiquetas reales. En ese caso el modelo obtuvo un 0.88 de precisión lo que indica que si no es excelente si es apto para las predicciones.

En la segunda se usaron las métricas propias de la librería para evaluar el desempeño en términos de precisión. Este devolvión un valor mucho menor de 0.81. A fines del estudio y al superar el 0.80 se considera como válido.

Además, en ambos casos la pérdida tiene valores bajos y en comparación con los obtenidos en los valores de validación se observan datos similares. Lo que indica que el modelo no ha caído en over-fitting

A continuación, se mostrará el resultado de algunas de las predicciones y las etiquetas reales que debían presentar

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.9437 - loss: 0.8156 - val_accuracy
```

```
# Obtener todas las imágenes del generador
test_images = []
test_labels = []
for i in range(len(test_generator)):
    images, labels = next(test_generator)
    test_images.extend(images)
    test_labels.extend(labels)

test_images = np.array(test_images)
test_labels = np.array(test_labels)

# Seleccionar aleatoriamente algunas imágenes para visualizar
num_images = len(test_images)
random_indices = random.sample(range(num_images), 9)

# Visualizar las imágenes con sus etiquetas verdaderas y predicciones
plt.figure(figsize=(12, 12))
for i, idx in enumerate(random_indices):
    plt.subplot(3, 3, i + 1)
    plt.imshow(test_images[idx].squeeze(), cmap='gray')
    true_label = class_labels[true_classes[idx]]
    predicted_label = class_labels[predicted_classes[idx]]
    plt.title(f'True: {true_label}\nPred: {predicted_label}')
    plt.axis('off')
plt.show()
```



45/45 ━━━━━━━━━━━━━━━━━━━━ 137s 3s/step - accuracy: 0.9573 - loss: 0.6624 - val_accuracy

```
# Obtener todas las imágenes del generador
test_images = []
test_labels = []
for i in range(len(test_generator)):
    images, labels = next(test_generator)
    test_images.extend(images)
    test_labels.extend(labels)

test_images = np.array(test_images)
test_labels = np.array(test_labels)

# Seleccionar aleatoriamente algunas imágenes para visualizar
num_images = len(test_images)
random_indices = random.sample(range(num_images), 9)

# Visualizar las imágenes con sus etiquetas verdaderas y predicciones
plt.figure(figsize=(12, 12))
for i, idx in enumerate(random_indices):
    plt.subplot(3, 3, i + 1)
    plt.imshow(test_images[idx].squeeze(), cmap='gray')
    true_label = class_labels[true_classes[idx]]
    predicted_label = class_labels[predicted_classes[idx]]
    plt.title(f'True: {true_label}\nPred: {predicted_label}')
    plt.axis('off')
plt.show()
```

