UNIVERSIDAD NACIONAL
GUILLERMO BROWN

Seminario Final

Licenciatura en Ciencia de Datos

Modelo predictivo de errores en impresión 3D

Autor: Pablo MOREIRA

Profesor: Dr. Juan Domingo GONZALEZ

UNaB – 2024

Resumen

El objetivo principal de este trabajo es desarrollar un modelo predictivo para detectar errores en impresiones 3D utilizando técnicas de aprendizaje automático, específicamente redes neuronales convolucionales (CNN). Nuestro propósito es analizar imágenes de impresiones 3D para clasificarlas como defectuosas o no defectuosas, con el fin de mejorar la calidad, eficiencia y sostenibilidad en el proceso de impresión 3D.

El modelo se entrenó y validó utilizando un conjunto de datos de imágenes de impresiones 3D, y se utilizaron métricas como accuracy, precision y recall para evaluar su rendimiento. Los resultados obtenidos muestran un alto grado de precisión en la clasificación de errores, lo que indica el potencial del modelo para mejorar la detección temprana de problemas en la impresión 3D.

Además, se propone una futura expansión del proyecto para desarrollar un sistema en tiempo real que compare simulaciones de impresión con imágenes reales, utilizando un dispositivo Arduino para interactuar con la impresora y tomar decisiones basadas en la clasificación del modelo. Se espera que este sistema contribuya a la optimización de los procesos de impresión 3D y a la reducción de residuos, para mejorar así la calidad y eficiencia en la fabricación aditiva.

Palabras clave: Impresión 3D, FDM, Modelado por Deposición Fundida, errores de impresión, modelo predictivo, red neuronal convolucional (CNN), clasificación de imágenes, aprendizaje automático.

Contenido

Introducción

Muchos son los desarrollos iniciados dentro del rubro de impresión 3D. La mayoría están orientados a mejorar la experiencia de uso, en un principio las máquinas eran construidas y configuradas casi en su totalidad por los usuarios, incluyendo la parte mecánica y el firmware (proyecto RepRap[1])Ilustración 1- Proyecto RepRap.

Originalmente, la complejidad de configuración y la necesidad de ajustes personalizados limitaban el uso de las impresoras 3D a un nicho reducido. Sin embargo, los avances actuales han simplificado enormemente el proceso de armado y calibración, permitiendo un uso inmediato gracias a la autorregulación y a los preajustes para diferentes materiales.

Por otro lado, se intenta también lograr la automatización del proceso de impresión y mejorar la calidad del resultado final, evitando al máximo el post procesado de las piezas. El software que se utiliza para enviar la información a la impresora posee los datos de cada modelo y sus configuraciones particulares para facilitar esta tarea.

Los modelos más recientes incluyen la posibilidad de operar a distancia a través de conexión wifi y el uso de cámaras para supervisar remotamente el trabajo, existe la posibilidad de que ante un corte de energía se pueda reanudar el trabajo sin perder lo realizado.

En este trabajo nos vamos a centrar en la impresión 3D bajo el sistema FDM (Modelado por Deposición Fundida), un filamento termoplástico se alimenta a través de un extrusor que lo calienta, lo funde y lo extruye a través de una boquilla fina2. Este material fundido se deposita capa por capa en la impresora 3D para construir el objeto tridimensional.

Principales áreas de uso en Impresión 3D:

La impresión 3D FDM es una tecnología versátil con aplicaciones en diversos campos. En la industria, se utiliza para prototipado rápido, producción personalizada de piezas y repuestos, y fabricación de componentes ligeros para sectores como el aeroespacial y automotriz. En arquitectura, permite crear maquetas detalladas, en medicina facilita la impresión de modelos anatómicos y prototipos de dispositivos médicos. Además, tiene usos en joyería, diseño de moda, educación, arte, gastronomía y experimentación científica. También se ha convertido en una actividad utilizada como medio de ingresos para aficionados y emprendedores.

Componentes de una impresora 3D

Los componentes incluidos en cada equipo pueden variar según su marca y modelo pero todos incluyen un seteo básico detallado a continuación (Ilustración 2- Detalle de impresora:

Marco y Estructura: El marco proporciona la estructura principal y la estabilidad de la impresora. Está compuesto por perfiles de aluminio que ofrecen rigidez y durabilidad.

Plataforma de Construcción (Cama Caliente): Es la superficie donde se imprime el objeto. Se calienta para ayudar a adherir el filamento al inicio y prevenir el warping (deformación) de las capas inferiores.

[1] Reprap, Slic3r y el Futuro de la Impresión 3D

Extrusor y Hotend: El extrusor empuja el filamento hacia el hotend. El hotend se encarga de fundir el filamento y depositarlo capa por capa en la plataforma de construcción.

Ejes (X, Y, Z): El eje X se refiere al movimiento horizontal del cabezal de impresión. El eje Y se encarga del movimiento de la cama caliente hacia adelante y hacia atrás. El eje Z controla el movimiento vertical del cabezal de impresión y la cama.

Motor Paso a Paso: Cada eje está impulsado por un motor paso a paso que convierte impulsos eléctricos en movimientos precisos.

Tornillos y Varillas Roscadas: Se utilizan para mover los ejes Z y mantener la estabilidad.

Pantalla LCD y Botones de Control: Proporcionan una interfaz para controlar la impresora y ajustar parámetros como la temperatura y la velocidad de impresión.

Fuente de Alimentación: Suministra la energía necesaria para la impresora.

Ventiladores: Ayudan a mantener las temperaturas bajo control, especialmente en el hotend y el motor del extrusor.

Tarjeta Controladora (Mainboard): Es la unidad central que controla todos los componentes de la impresora.

Sensores e Interruptores: Sensores de fin de carrera y interruptores que ayudan a la impresora a determinar la posición de los ejes y la cama.

Puerto USB y Ranura para Tarjeta SD: Permite la conexión de la impresora a una computadora y la impresión desde tarjetas SD.

Los equipos que se encuentran en el mercado son perfeccionados continuamente por las distintas marcas para lograr mejores resultados y con interfaces cada vez más simples. Actualmente las investigaciones se relacionan con la detección temprana de errores, desarrollando equipos con la capacidad de cancelar un trabajo al presentarse un error grave para evitar mayor pérdida de material[2].

Principales errores en el proceso de Impresión 3D

Cuando nos referimos a errores de impresión, los principales son los detallados a continuación, que dividiremos en dos grupos principales, los que tienen solución durante el proceso y los que no pueden ser corregidos y requieren que el proceso vuelva a iniciarse.

Grupo de errores corregibles mediante modificación de parámetros en tiempo real:

- Líneas de capa visibles: puede rectificarse casi de inmediato modificando la altura de capa y la temperatura/refrigeración (Ilustración 3- Líneas de capa visibles.
- Retracción: Se produce cuando la impresora 3D invierte temporalmente la dirección en la que se extruye el material plástico, esto implica tirar hacia atrás una pequeña cantidad de filamento del extremo caliente del extrusor antes de que la impresora mueva su cabezal de impresión a una nueva ubicación. Si esa acción no está bien configurada, quedaran hilos entre las distintas partes de la pieza (Ilustración 4 - Retracción

[2] Optimización de los procesos de impresión 3D

Grupo de errores sin posibilidad de corrección durante la impresión:

- Problemas de adhesión a la cama y warping: en ambos casos la pieza no se adhiere bien a la cama, puede deformarse por diferencia de temperatura o simplemente soltarse de la cama. En ambos casos la impresión tiene que volver a comenzar (Ilustración 5 - Warping.
- Falla en el diseño de soportes: depende la inclinación de la figura es probable que necesite soportes para ser impresa, si los mismos están mal diseñados o configurados pueden afectar al resultado final y muchas veces esto puede detectarse antes de finalizar la pieza (Ilustración 6 - Diseño de soportes.
- Error en la nivelación de la altura de cama: Un proceso que en la mayoría de los casos es manual, salvo en las maquinas que cuentan con sensor de autolevel, una mala altura del pico con respecto a la cama puede generar un error al momento del inicio de la impresión (Ilustración 7 - Seteo de la altura de cama.

Metodología

Para construir nuestro modelo predictivo, utilizaremos el conjunto de datos "3d-printer-defected-dataset" disponible en Kaggle[3]. Este conjunto contiene imágenes tomadas desde el extremo del perfil del eje x de la máquina, captando fotos de impresiones 3D, clasificadas como "defected" (defectuosas) o "no_defected" (no defectuosas) Ilustración 8 - Imágenes clasificadas por clases.

El dataset comprende un total de 1557 imágenes, distribuidas de manera equitativa entre ambas clases. Es importante destacar que todas las impresiones registradas se realizaron con el mismo filamento y sin cambios de color, lo que garantiza una mayor consistencia en los datos.

Para maximizar la reducción de residuos, mejorar la sostenibilidad y optimizar costos en la impresión 3D, la detección temprana de errores debe complementarse con la capacidad de corregirlos. Sin embargo, los métodos actuales requieren un amplio conjunto de datos con múltiples impresiones del mismo objeto para ser eficientes, y no permiten correcciones en tiempo real, lo que implica que una pieza defectuosa no puede ser recuperada una vez que se detecta el error.

Primera aproximación al problema

El número de imágenes reportado (1557) resulta insuficiente para abordar el problema de clasificación utilizando redes neuronales. Para lograr buenos resultados, es necesario aumentar el banco de imágenes. El aumento de datos (data augmentation) es una técnica utilizada en el aprendizaje automático para aumentar artificialmente la cantidad y diversidad de datos de entrenamiento. Esto se logra aplicando diversas transformaciones a los datos existentes, como rotaciones, cambios de escala, recortes, ajustes de brillo y contraste, entre otros, creando nuevas muestras a partir de las imágenes originales. De esta manera, el modelo puede aprender a reconocer patrones y características relevantes en una variedad más amplia de condiciones, mejorando su capacidad de generalización y evitando el sobreajuste.

Analizando el problema puntual planteado hay ciertas técnicas que no podemos utilizar, y esto se debe a que en el campo de la impresión 3D las imágenes nunca van a girar (la cama siempre va a permanecer en la misma posición), tampoco trasladar el centro de la imagen ni cambiar su tamaño. El desplazamiento horizontal puede utilizarse, pero no así el vertical por el mismo motivo que la rotación.

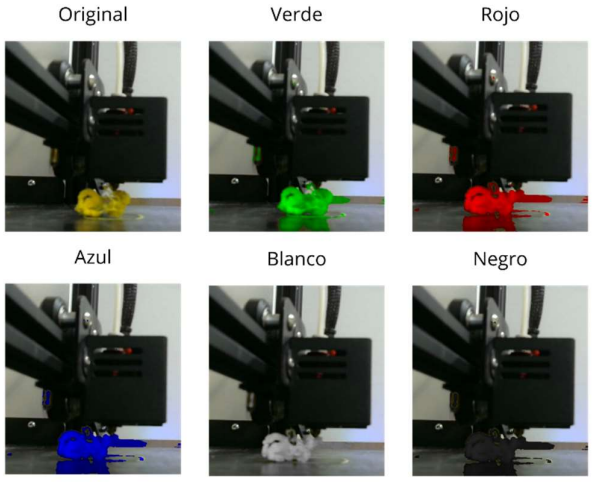
Tenemos entonces el objetivo de aumentar la base sin utilizar los métodos más comunes para hacerlo.

Dentro del ámbito de la impresión 3D no solo se utilizan muchos materiales sino también infinidad de colores es por eso que orientamos el aumento de la base en ese sentido desarrollando un script de Python que tome las imágenes originales que solo contienen impresiones en color amarillo y las duplique en rojo, verde, azul, negro y blanco.

El algoritmo toma la imagen original, detecta un rango predeterminado de color codificado en RGB y lo reemplaza por uno nuevo, aplicando a la imagen resultante el mismo filtro de

[3] 3D-Printer Defected Dataset

luces y sombras para que la nueva imagen sea comparable con una real (Ilustración 9- Imágenes de la base ampliada clasificadas por clases).



De esta manera quintuplicamos la base sin modificar la ubicación.

Luego tenemos que decidir cómo se realizará la división de la base.

La metodología de separación de la base de datos en conjuntos de entrenamiento, validación y test es fundamental en el desarrollo de modelos de aprendizaje automático. El objetivo es garantizar que el modelo tenga un buen rendimiento tanto en los datos conocidos (entrenamiento) como en datos nuevos (test).

En el aprendizaje automático, los datos se dividen en tres conjuntos: entrenamiento (80%), validación (10%) y test (10% restante). El conjunto de entrenamiento se utiliza para enseñar al modelo a reconocer patrones, el conjunto de validación para evaluar el rendimiento y ajustar los hiperparámetros durante el entrenamiento, y el conjunto test, que es completamente independiente, para evaluar el rendimiento final del modelo y asegurar que generalice bien a datos nuevos. Esta división ayuda a prevenir el sobreajuste y proporciona una estimación realista de cómo el modelo se desempeñará en el mundo real.

Tomando la base ampliada los datos quedan distribuidos de la siguiente manera:

Set	Total Images	Defected	Not Defected
Training Set	5604	2732	2872
Validation Set	1869	911	958
Test Set	1869	911	958

Etapas del procedimiento:

1. Mezclar aleatoriamente: Se mezclan los datos para asegurar que la distribución de las clases o características sea similares en todos los conjuntos.

2. Dividir: Se divide la base de datos en las proporciones mencionadas anteriormente.

3. Entrenar: Se utiliza el conjunto de entrenamiento para ajustar el modelo y el conjunto de validación para seleccionar los mejores hiperparámetros.

4. Evaluar: Se utiliza el conjunto test para obtener una estimación final del rendimiento del modelo.

Es muy importante que la separación siempre sea aleatoria, estratificada y manteniendo la misma proporción de clases en cada conjunto para evitar sesgos. Tampoco debe utilizarse el conjunto test para ajustar el modelo, ya que esto invalidaría su capacidad para estimar el rendimiento en datos nuevos.

Análisis exploratorio de la base ampliada

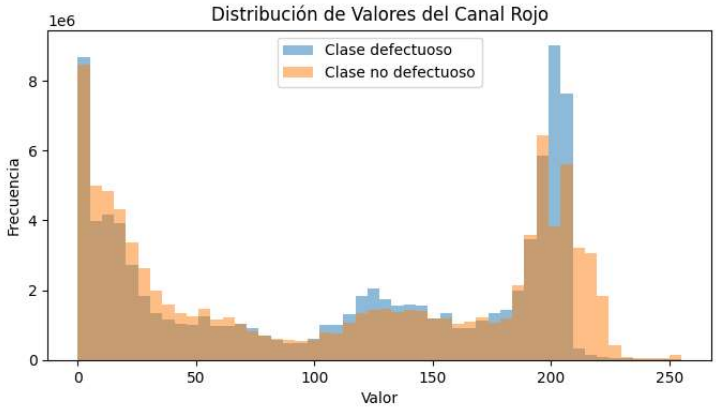
Una vez que la base se encuentra ampliada tenemos que ver la forma en que se distribuyen los parámetros. En el caso de imágenes lo más importante es el color identificado por los tres canales que forman el código RGB. Si estos parámetros varían mucho entre las dos clases que tenemos definidas la forma de tratar el modelo cambiaría y es un dato que tenemos que saber a priori.

En nuestro trabajo, hay una superposición significativa entre las distribuciones de ambas clases en los tres canales de color, especialmente en los valores de intensidad altos. Esto indica que la intensidad de los colores por sí sola no es suficiente para distinguir claramente entre las clases(Ilustración 10 - Distribución de valores en los distintos canales de color.

Canal Rojo:

Clase defectuoso: Media=111.68, Desviación Estándar=78.73

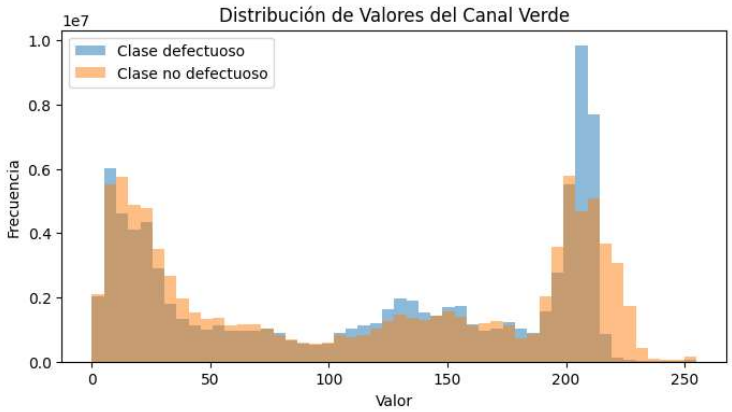
Clase no defectuoso: Media=109.24, Desviación Estándar=81.92



Canal Verde:

Clase defectuoso: Media=116.22, Desviación Estándar=79.05

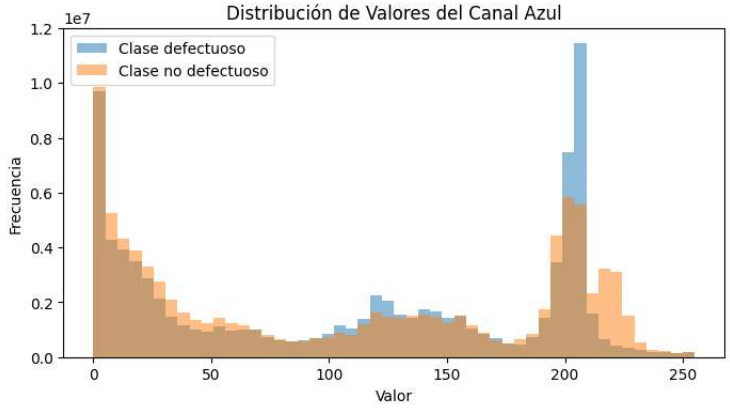
Clase no defectuoso: Media=113.59, Desviación Estándar=81.99



Canal Azul:

Clase defectuoso: Media=110.40, Desviación Estándar=80.78

Clase no defectuoso: Media=108.08, Desviación Estándar=83.54



Aunque hay superposición, existen algunas diferencias sutiles en las distribuciones. Por ejemplo, la clase "no defectuoso" tiende a tener más píxeles oscuros en el canal azul y menos píxeles con baja intensidad de verde.

Dado que las diferencias en los canales de color individuales no son muy marcadas, es probable que el modelo necesite aprender características más complejas (texturas, patrones, combinaciones de colores) para lograr una buena clasificación.

Consideraciones a tener en cuenta en la creación del modelo

La construcción de un modelo de red neuronal efectivo para la clasificación de imágenes requiere una consideración cuidadosa de varios aspectos clave: la elección de la arquitectura adecuada que sea indicada para extraer características relevantes de las imágenes; encontrar el equilibrio justo entre profundidad y complejidad de la red para evitar el sobreajuste y asegurar una buena capacidad de generalización; aplicar técnicas de regularización para prevenir el sobreajuste y mejorar el rendimiento del modelo; y elegir un optimizador adecuado con sus hiperparámetros óptimos, ya que esto puede afectar significativamente la eficiencia del entrenamiento y la calidad del modelo resultante.

Características de la red neuronal usada en el modelo base

Una red neuronal secuencial es una función compuesta que mapea un vector de entrada a un vector de salida a través de una secuencia de capas, donde cada una aplica transformación lineal seguida de una función de activación no lineal. Las principales características de este modelo son el orden secuencial de las capas y la conexión directa entre ellas, lo que facilita su creación y modificación. Es adecuado para redes neuronales simples y para experimentar con diferentes arquitecturas, especialmente cuando se requiere una pila lineal de capas sin ramificaciones ni conexiones complejas. Las funciones de activación introducen la no linealidad necesaria para el aprendizaje de relaciones complejas, mientras que los pesos y sesgos son los parámetros ajustables que se aprenden durante el entrenamiento. La arquitectura de la red se elige en función del problema específico a resolver.

A continuación, explicaremos el modelo nro 1 (inicial) utilizado que no incluye el dataset aumentado y con una configuración base que luego será modificada (ver sección Modelo nro 2 – Final).

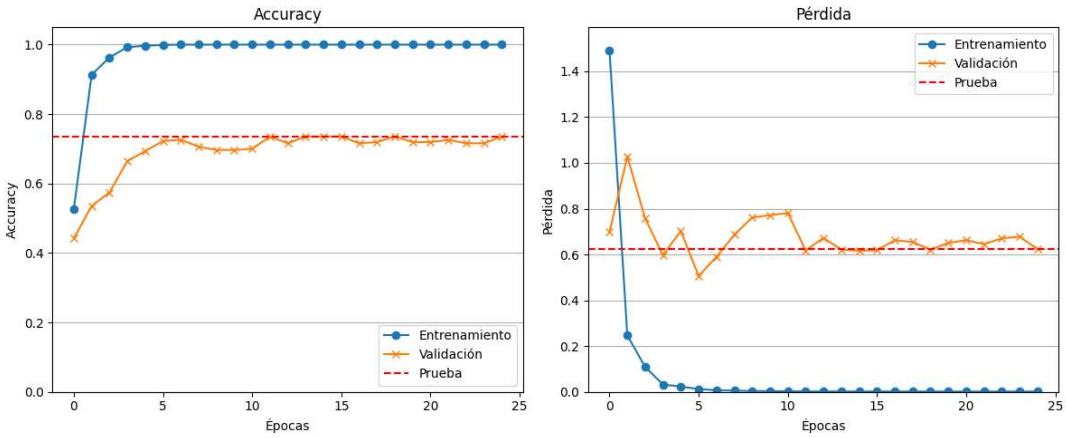
Capa (tipo)	Forma de Salida	Núm. Parámetros
conv2d_4 (Conv2D)	(Ninguno, 180, 180, 4)	304
max_pooling2d_4 (MaxPooling2D)	(Ninguno, 90, 90, 4)	0
flatten_4 (Flatten)	(Ninguno, 32400)	0
dense_8 (Dense)	(Ninguno, 32)	1.036.832
dense_9 (Dense)	(Ninguno, 2)	66

El modelo es una red neuronal secuencial básica para la clasificación de imágenes. En primer lugar, aplica una capa convolucional (`Conv2D`) con 4 filtros de tamaño 5x5 y padding 'same', que extrae características locales de la imagen de entrada mediante la aplicación de filtros que detectan patrones como bordes y texturas. La función de activación ReLU introduce no linealidad, permitiendo a la red aprender patrones más complejos. A continuación, una capa de Max Pooling reduce la dimensionalidad de la salida de la capa convolucional, seleccionando el valor máximo dentro de regiones locales y ayudando a prevenir el sobreajuste. Después, una capa de aplanamiento (`Flatten`) transforma la salida en un vector unidimensional, preparando los datos para las capas densamente conectadas.

Finalmente, dos capas densamente conectadas (`Dense`) realizan la clasificación. La primera capa, con 32 neuronas, aprende representaciones más abstractas de las características extraídas, mientras que la segunda capa, con un número de neuronas igual al número de clases (`num_classes`), produce logits (puntuaciones sin normalizar) para cada clase, indicando la probabilidad de que la imagen pertenezca a cada una de ellas. El modelo se compila utilizando el optimizador 'adam', que adapta automáticamente la tasa de aprendizaje, la función de pérdida `SparseCategoricalCrossentropy` con `from_logits=True`, que maneja la clasificación multiclase con logits crudos, y la métrica 'accuracy', que mide la proporción de predicciones correctas.

Resultados del modelo inicial

Esta primera aproximación en teoría supone un funcionamiento correcto pero los resultados sobre la base (sin aumento de datos previo) fueron los siguientes:

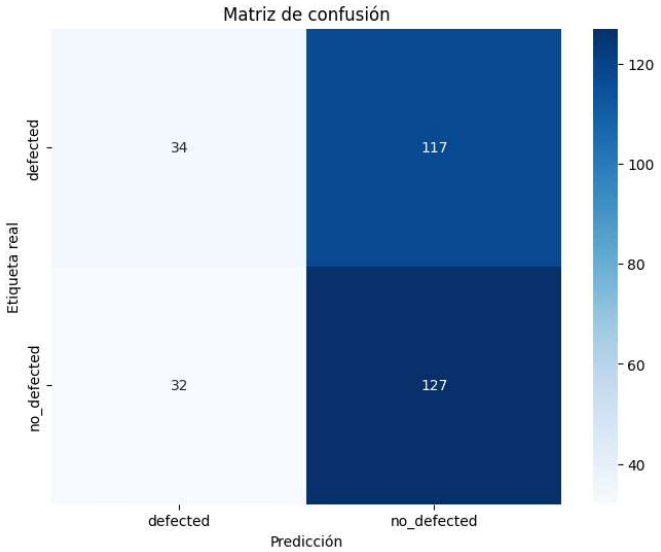


El modelo de red neuronal secuencial muestra un rendimiento mixto en las métricas evaluadas. Durante el entrenamiento, la precisión mejora rápidamente y se estabiliza cerca del 100%, lo que indica una buena capacidad de aprendizaje en los datos de entrenamiento. Sin embargo, la precisión en el conjunto de validación se estanca alrededor del 75% y presenta fluctuaciones, sugiriendo un posible sobreajuste, donde el modelo se adapta demasiado a los datos de entrenamiento y pierde capacidad de generalización a los nuevos.

Esta hipótesis de sobreajuste se refuerza al observar la pérdida. La pérdida en el conjunto de entrenamiento disminuye rápidamente y se estabiliza, lo que es deseable. Sin embargo, en el conjunto de validación, la pérdida disminuye inicialmente pero luego comienza a fluctuar y aumentar ligeramente, lo que es un claro indicio de sobreajuste.

Al evaluar el modelo en el conjunto de prueba, se observa un rendimiento inferior al esperado. La precisión es de 0.5316, el recall de 0.5219 y el F1-score de 0.4857. Estos valores indican que el modelo no generaliza bien a datos no vistos, cometiendo errores significativos en sus predicciones y mostrando dificultades para identificar correctamente la clase positiva.

Este problema se refleja claramente en la matriz de confusión donde vemos como clasifica las dos categorías comparadas con sus etiquetas reales



Observamos que el modelo tiene un rendimiento notablemente mejor en la clasificación de imágenes "no_defected", con 124 aciertos, en comparación con solo 35 errores. Esto sugiere que el modelo es bastante hábil para identificar correctamente productos que no presentan

defectos. Sin embargo, el modelo muestra dificultades en la clasificación de imágenes "defected". Aunque logra 117 aciertos, también comete 34 errores, lo que indica que un número significativo de productos defectuosos se clasifican erróneamente como no defectuosos. En general, la matriz de confusión confirma que el modelo tiene un sesgo hacia la clasificación de productos como "no_defected", lo que puede ser problemático en aplicaciones donde la identificación precisa de defectos es crucial.

En conclusión, el modelo presenta un buen rendimiento en los datos de entrenamiento, pero sufre de sobreajuste, lo que resulta en un rendimiento deficiente en los datos de validación y prueba. Para mejorar el modelo, debemos aplicar técnicas de regularización, aumentar la cantidad y diversidad de datos de entrenamiento, explorar arquitecturas de modelo más complejas, ajustar los hiperparámetros y realizar una evaluación más exhaustiva para comprender mejor los tipos de errores que comete el modelo.

Red neuronal final utilizada en el modelo

La elección final del modelo es un proceso de prueba y error donde se busca encontrar el mejor resultado final basándose en las herramientas computacionales y matemáticas que ayuden a detectar los errores y realizar las correcciones.

En el caso puntual que estamos tratando, el modelo final sufrió muchas modificaciones durante varias etapas para llegar a una versión más ajustada, aunque perfectible, lo interesante es analizar los cambios y su efecto en el resultado final.

Uno de los primeros puntos considerados fue, como se detalla al principio, el aumento de la base. Esto es determinante para mejorar el entrenamiento. En primera instancia se desarrolló el script para poder automatizar el cambio de color y luego se aplicó directamente sobre el modelo base

Cambio en el learning rate

Seguido a eso se intentó reducir la tasa de aprendizaje (learning rate), es un hiperparámetro crucial en el entrenamiento de modelos de aprendizaje automático. Determina cuánto se ajustan los pesos del modelo en cada iteración del proceso de optimización, en respuesta al error estimado.

Durante el entrenamiento, el objetivo es minimizar una función de pérdida (loss function) que mide qué tan bien el modelo predice los datos de entrenamiento. La tasa de aprendizaje controla la magnitud de los pasos que se toman en la dirección opuesta al gradiente de la función de pérdida.

La actualización de los pesos se puede expresar como:

$$\mathrm{w}_{(\mathrm{t}+1)} = \mathrm{w}_{\mathrm{t}} - \alpha\nabla\mathrm{L}(\mathrm{w}_{\mathrm{t}})$$

$\mathrm{w}_{(\mathrm{t}+1)}$ son los pesos actualizados en el siguiente paso de tiempo.

w_t son los pesos actuales.

α es la tasa de aprendizaje (un hiperparámetro que controla la magnitud del cambio en los pesos).

$\nabla L(w_t)$ es el gradiente de la función de pérdida con respecto a los pesos actuales.

El gradiente indica la dirección en la que la función de pérdida aumenta más rápidamente, por lo que, al restarlo de los pesos, nos movemos en la dirección opuesta, buscando el

mínimo de la función. La tasa de aprendizaje escala este movimiento, determinando qué tan grande es el paso que damos en cada iteración.

Tasa de aprendizaje alta: Puede llevar a una convergencia más rápida, pero también puede hacer que el modelo sobrepase el mínimo óptimo y oscile alrededor de él, dificultando la convergencia.

Tasa de aprendizaje baja: Puede llevar a una convergencia más lenta, pero también puede hacer que el modelo quede atrapado en mínimos locales, impidiendo que alcance el mínimo global.

Elección de la tasa de aprendizaje:

La elección de la tasa de aprendizaje adecuada es crucial para el éxito del entrenamiento. Un valor demasiado alto puede llevar a una inestabilidad en el entrenamiento, mientras que un valor demasiado bajo puede ralentizarlo considerablemente. En la práctica, se suelen utilizar técnicas como la disminución gradual de la tasa de aprendizaje (learning rate decay) para ajustar dinámicamente la tasa durante el entrenamiento y lograr un equilibrio entre velocidad y estabilidad.

También existen métodos más avanzados, como los optimizadores adaptativos (Adam, RMSprop, etc.), que ajustan automáticamente la tasa de aprendizaje para cada parámetro del modelo, lo que puede mejorar aún más el rendimiento del entrenamiento. En nuestro caso el mejor resultado lo obtuvo el optimizador RMSprop que luego desarrollaremos.

El modelo al que arribamos tiene las siguientes características:

Capa (tipo)	Forma de Salida	Núm. Parámetros
conv2d_6 (Conv2D)	(None, 180, 180, 32)	896
batch_normalization (BatchNormalization)	(None, 180, 180, 32)	128
max_pooling2d_3 (MaxPooling2D)	(None, 90, 90, 32)	0
dropout_4 (Dropout)	(None, 90, 90, 32)	0
conv2d_7 (Conv2D)	(None, 90, 90, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 90, 90, 64)	256
max_pooling2d_4 (MaxPooling2D)	(None, 45, 45, 64)	0
dropout_5 (Dropout)	(None, 45, 45, 64)	0
conv2d_8 (Conv2D)	(None, 45, 45, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 45, 45, 128)	512
max_pooling2d_5 (MaxPooling2D)	(None, 22, 22, 128)	0
dropout_6 (Dropout)	(None, 22, 22, 128)	0
flatten_1 (Flatten)	(None, 61952)	0
dense_2 (Dense)	(None, 128)	7,929,984
dropout_7 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 1)	129

Arquitectura del Modelo:

El modelo es una Red Neuronal Convolucional (CNN)[4] secuencial diseñada para tareas de clasificación binaria (dos clases). La arquitectura consta de los siguientes bloques principales:

Bloques Convolucionales:

Conv2D (32 filtros, kernel 3x3, activación ReLU):

La capa Conv2D aplica una serie de filtros (también llamados kernels) a la imagen de entrada. Cada filtro es una pequeña matriz de números (en este caso, 3x3). El proceso de convolución consiste en deslizar cada filtro sobre la imagen de entrada, realizando una operación matemática. En cada posición, se multiplican elemento por elemento los valores del filtro con los valores de la imagen que se encuentran debajo del filtro, y luego se suman los resultados. Esto produce un único valor de salida para esa posición.

Mapa de Características (Feature Map): Al deslizar el filtro sobre toda la imagen, se obtiene un mapa de características, que es una nueva imagen donde cada píxel representa el resultado de la convolución en la posición correspondiente.

Múltiples Filtros: En una capa Conv2D, se aplican múltiples filtros (32 en este caso) a la imagen de entrada, lo que resulta en múltiples mapas de características. Cada filtro aprende a detectar diferentes patrones en la imagen (bordes, texturas, etc.).

Función de Activación ReLU: Después de la convolución, se aplica una función de activación ReLU (Rectified Linear Unit) a cada elemento del mapa de características. La función ReLU introduce no linealidad en el modelo y ayuda a aprender patrones más complejos.

Sea X la imagen de entrada de tamaño (H, W, C), donde H es la altura, W es el ancho y C es el número de canales de color. Sea F un filtro de tamaño (k, k), donde k es el tamaño del kernel (3 en este caso).

La convolución en una posición (i, j) se calcula como:

$$Y(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} X(i+m, j+n) * F(m,n)$$

$Y(i,j)$ es el valor del mapa de características en la posición (i,j).

$X(i+m, j+n)$ es el valor de la imagen de entrada en la posición $(i+m, j+n)$.

$F(m,n)$ es el valor del filtro en la posición (m,n).

Después de la convolución, se aplica la función de activación ReLU:

$$Z(i,j) = \max\big(0, Y(i,j)\big)$$

$Z(i,j)$ es el valor final del mapa de características en la posición (i,j) después de aplicar ReLU.

kernel_regularizer=l2(0.005):

Al agregar el término de penalización L2 a la función de pérdida, el modelo se ve obligado a encontrar un equilibrio entre minimizar la pérdida en los datos de entrenamiento y mantener los pesos del kernel pequeños. Esto ayuda a prevenir que el modelo se

[4] Torres, Jordi (2020) Python Deep Learning. Introducción práctica con Keras y TensorFlow 2

sobreajuste a los datos de entrenamiento y mejora su capacidad de generalización a nuevos datos.

La norma L2 de un vector es la raíz cuadrada de la suma de los cuadrados de sus elementos. En el caso de los pesos del kernel, la norma L2 es:

$$||W||^2 = \sqrt{\sum_i \sum_j w_{ij}^2}$$

W es la matriz de pesos del kernel.

w_{ij} es el peso en la fila i y columna j de la matriz W.

BatchNormalization:

La capa BatchNormalization (Normalización por Lotes) se utiliza para normalizar las activaciones de una capa anterior en una red neuronal, lo que ayuda a estabilizar y acelerar el entrenamiento. Este proceso se aplica a cada mini-batch, que es un pequeño subconjunto de muestras extraídas aleatoriamente del conjunto de datos completo, durante el entrenamiento. La normalización consiste en centrar y escalar las activaciones para que tengan media cero y desviación estándar uno, lo que mejora la convergencia y el rendimiento del modelo.

Los pasos principales de BatchNormalization son:

Cálculo de la Media y Varianza: Para cada mini-batch, se calcula la media y la varianza de las activaciones a lo largo del eje de los canales.

Normalización: Las activaciones se normalizan restando la media y dividiendo por la desviación estándar (con una pequeña constante para evitar la división por cero).

Escalado y Desplazamiento: Se aplican dos parámetros entrenables (gamma y beta) para escalar y desplazar las activaciones normalizadas. Esto permite a la red aprender la distribución óptima de las activaciones.

Sea X un mini-batch de activaciones de tamaño (N, C, H, W), donde N es el tamaño del mini-batch, C es el número de canales, H es la altura y W es el ancho.

Cálculo de la Media y Varianza:

$$\mu_B = \left(\frac{1}{N}\right) * \sum_{i=1}^{N} x_i$$

$$\sigma_B^2 = \left(\frac{1}{N}\right) * \sum_{i=1}^{N} (x_i - \mu_B)^2$$

μ_B es la media del mini-batch.

σ_B^2 es la varianza del mini-batch.

x_i es la activación i-ésima del mini-batch.

Normalización:

$$\hat{x}_i = \frac{(x_i - \mu_B)}{\sqrt{\sigma_B^2 + \varepsilon}}$$

$\hat{x}_i$ es la activación normalizada i-ésima.

ε es una pequeña constante (por ejemplo, 1e-5) para evitar la división por cero.

Escalado y Desplazamiento:

$$y_i = \gamma * \hat{x}_i + \beta$$

y_i es la activación final i-ésima.

γ y β son parámetros entrenables.

MaxPooling2D (2x2):

La capa MaxPooling2D divide la imagen de entrada en regiones no superpuestas utilizando una ventana de pooling (en este caso, 2x2). Para cada región, se selecciona el valor máximo y se descarta el resto. Este valor máximo se convierte en el valor de salida para esa región. Al seleccionar solo el valor máximo de cada región, la capa MaxPooling2D reduce las dimensiones espaciales (altura y ancho) de la imagen de entrada, mientras que el número de canales se mantiene igual.

Sea X la imagen de entrada de tamaño (H, W, C), donde H es la altura, W es el ancho y C es el número de canales de color. Sea p el tamaño de la ventana de pooling (2 en este caso).

El valor de salida $Y(i,j)$ en la posición (i,j) del mapa de características de salida se calcula como:

$$Y(i,j) = \max_{m=0}^{p-1} \max_{n=0}^{p-1} X(p * i + m, p * j + n)$$

$Y(i,j)$ es el valor del mapa de características de salida en la posición (i, j).

$X(p * i + m, p * j + n)$ es el valor de la imagen de entrada en la posición $(p * i + m, p * j + n)$.

Dropout (0.25):

Durante el entrenamiento, la capa Dropout desactiva aleatoriamente un porcentaje de las neuronas de la capa anterior. En este caso, con un valor de 0.25, el 25% de las neuronas serán desactivadas en cada iteración del entrenamiento.

Para compensar la desactivación de neuronas, las salidas de las neuronas activas se escalan por un factor de $\frac{1}{(1-p)}$, donde p es la probabilidad de desactivar una neurona (0.25 en este caso). Esto asegura que la magnitud esperada de la salida de la capa se mantenga constante durante el entrenamiento.

Durante la inferencia (cuando se utiliza el modelo para hacer predicciones), todas las neuronas están activas, y no se aplica ningún escalado.

Sea x el vector de entrada a la capa Dropout y y el vector de salida. Sea p la probabilidad de desactivar una neurona (0.25 en este caso).

La salida de la capa Dropout se calcula como:

$$y = \frac{(x * m)}{(1 - p)}$$

x es el vector de entrada a la capa Dropout y y el vector de salida

p la probabilidad de desactivar una neurona

m es un vector de máscara generado de la misma dimensión que x, donde cada elemento de m es 1 con probabilidad $(1 - p)$ y 0 con probabilidad p

Bloque de Clasificación:

Flatten: Esta capa transforma la salida de la capa anterior (que podría ser una capa convolucional o de agrupación) en un vector unidimensional. Esencialmente, toma todos los elementos de la matriz de entrada y los coloca en una sola fila.

Dense (128 neuronas, activación ReLU):

Capa completamente conectada con 128 neuronas y activación ReLU para aprender representaciones más abstractas de las características extraídas por las capas convolucionales.

Dropout (0.5): Desactiva aleatoriamente el 50% de las neuronas para prevenir el sobreajuste en la capa densa.

Dense (1 neurona, activación Sigmoid):

Transformación Lineal (Dense): Aplica una transformación lineal a la entrada. En este caso, como hay una sola neurona de salida (1), la transformación lineal se reduce a una combinación lineal ponderada de las entradas, más un sesgo:

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

z es el resultado de la transformación lineal.

$w^1, w^2, \dots, w_{\mathrm{n}}$ son los pesos de la neurona.

$x^1, x^2, \dots, x_{\mathrm{n}}$ son las entradas a la neurona.

b es el sesgo de la neurona.

Aplica la función de activación sigmoid al resultado de la transformación lineal. La función sigmoid comprime el valor de z en un rango entre 0 y 1, donde la salida representa la probabilidad de pertenecer a una clase:

$$a = \frac{1}{1 + \mathrm{e}^{-\mathrm{z}}}$$

a es la salida de la neurona después de aplicar la función sigmoid.

e es la base del logaritmo natural

Optimizador y Compilación:

RMSprop (Root Mean Square Propagation) es un algoritmo de optimización utilizado en el entrenamiento de redes neuronales profundas. Su objetivo principal es acelerar la convergencia del proceso de aprendizaje ajustando adaptativamente la tasa de aprendizaje para cada parámetro individual.

Cálculo de la Media Móvil Exponencial del Cuadrado de los Gradientes:

RMSprop mantiene una media móvil exponencial (EMA) del cuadrado de los gradientes para cada parámetro. Esta EMA se calcula de la siguiente manera:

$$E[g^2]_t = \beta * E[g^2]_{t-1} + (1 - \beta) * g_t^2$$

$E[g^2]_t$ es la EMA del cuadrado del gradiente en el paso de tiempo t.

β es un factor de decaimiento (hiperparámetro) que controla la importancia de los gradientes pasados (típicamente entre 0.9 y 0.99).

g_t^2 es el cuadrado del gradiente en el paso de tiempo t.

Actualización de los Parámetros:

Los parámetros de la red neuronal se actualizan utilizando la siguiente fórmula:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\left(\sqrt{E[g^2]_t} + \varepsilon\right)} * g_t$$

θ_t es el valor del parámetro en el paso de tiempo t.

η es la tasa de aprendizaje (hiperparámetro).

ε es una pequeña constante (por ejemplo, 1e-8) para evitar la división por cero.

El término $\sqrt{E[g^2]_t} + \varepsilon$ en el denominador escala la tasa de aprendizaje para cada parámetro. Si la EMA del cuadrado del gradiente es grande, la tasa de aprendizaje se reduce, y viceversa.

La constante ε asegura que la actualización sea numéricamente estable.

Loss 'binary_crossentropy': Función de pérdida adecuada para clasificación binaria, que mide la diferencia entre las probabilidades predichas y las etiquetas verdaderas. Se define como el promedio de las entropías cruzadas individuales para cada muestra en el conjunto de datos. La entropía cruzada para una sola muestra se calcula de la siguiente manera:

$$L = -(y * \log(p) + (1 - y) * \log(1 - p))$$

L es la pérdida para una sola muestra.

y es la etiqueta verdadera (0 o 1).

p es la probabilidad predicha de que la muestra pertenezca a la clase positiva (clase 1).

La entropía cruzada mide la diferencia entre la distribución de probabilidad verdadera (representada por la etiqueta y) y la distribución de probabilidad predicha (representada por p).

El primer término $y * \log(p)$ se activa solo cuando la etiqueta verdadera es 1. Penaliza las predicciones incorrectas cuando la muestra pertenece a la clase positiva.

El segundo término $(1 - y) * \log(1 - p)$ se activa solo cuando la etiqueta verdadera es 0. Penaliza las predicciones incorrectas cuando la muestra pertenece a la clase negativa.

Metrics: Se utilizan métricas de accuracy, precision y recall para evaluar el rendimiento del modelo durante el entrenamiento y la validación.

Sabiendo que:

VP corresponde a los verdaderos positivos

VN representa los verdaderos negativos

FP indica los falsos positivos

FN es el valor que muestra los falsos negativos

La accuracy mide la proporción de predicciones correctas sobre el total de predicciones.

$$Accuracy = \frac{(VP + VN)}{(VP + VN + FP + FN)}$$

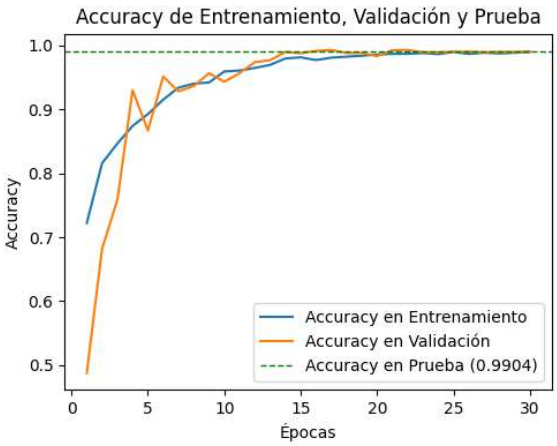
La precision mide la proporción de predicciones positivas correctas sobre el total de predicciones positivas.

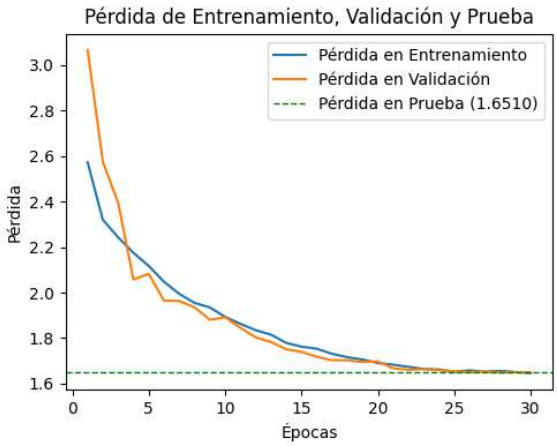
$$Precision = \frac{VP}{(VP + FP)}$$

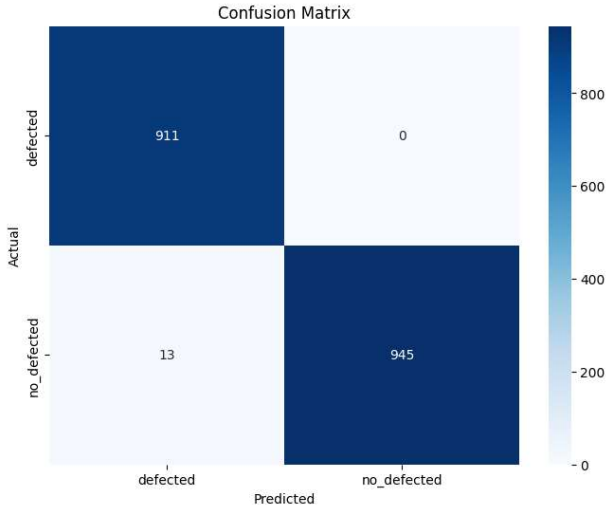
El recall mide la proporción de predicciones positivas correctas sobre el total de muestras positivas reales.

$$Recall = \frac{VP}{(VP + FN)}$$

Resultados del modelo final







	precision	recall	f1-score	support
defected	0.99	1.00	0.99	911
no_defected	1.00	0.99	0.99	958
accuracy			0.99	1869

macro avg	0.99	0.99	0.99	1869
weighted avg	0.99	0.99	0.99	1869

Análisis de Resultados y Matriz de Confusión:

Entrenamiento:

Accuracy (Precisión): A lo largo de las épocas, la precisión en el conjunto de entrenamiento aumenta progresivamente, llegando a valores cercanos al 99%. Esto indica que el modelo está aprendiendo a clasificar correctamente las muestras de entrenamiento.

Loss (Pérdida): La pérdida en el conjunto de entrenamiento disminuye de forma constante, lo que sugiere que el modelo está mejorando su capacidad para ajustar los datos.

Precision y Recall: Ambos, precision y recall, también muestran una tendencia ascendente, lo que significa que el modelo está mejorando tanto en la identificación correcta de muestras positivas (precision) como en la detección de todas las muestras positivas (recall).

Validación:

Accuracy (Precisión): La precisión en el conjunto de validación también mejora, alcanzando un valor del 98.98%. Esto indica que el modelo está generalizando bien a nuevos datos.

Loss (Pérdida): La pérdida en el conjunto de validación disminuye, aunque no tan rápido como en el conjunto de entrenamiento. Es normal que la pérdida de validación sea ligeramente mayor, ya que el modelo no ha visto estos datos durante el entrenamiento.

Precision y Recall: En la validación, la precision se mantiene en 1.0000 (perfecta) a lo largo de las épocas, lo que significa que todas las muestras clasificadas como positivas son realmente positivas. El recall también aumenta, pero no llega a ser perfecto, indicando que el modelo todavía clasifica mal algunas muestras positivas.

Matriz de Confusión:

La matriz de confusión proporciona información valiosa sobre el rendimiento del modelo:

Verdaderos Positivos (911): El modelo clasificó correctamente 911 muestras como "defected" (defectuosas).

Verdaderos Negativos (945): El modelo clasificó correctamente 945 muestras como "no_defected" (no defectuosas).

Falsos Positivos (0): El modelo no clasificó erróneamente ninguna muestra "no_defected" como "defected".

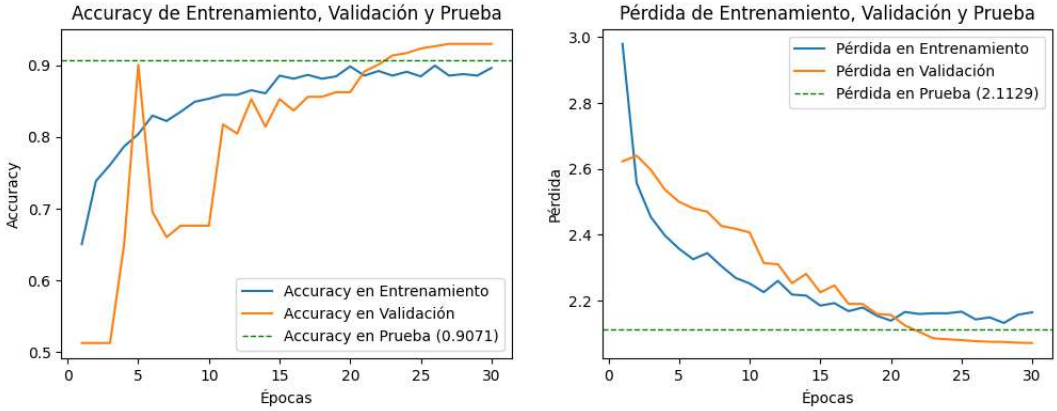
Falsos Negativos (13): El modelo clasificó erróneamente 13 muestras "defected" como "no_defected".

A pesar de que todos los modelos son perfectibles estos resultados muestran un muy buen desempeño del modelo final, siendo el mismo altamente recomendado para su utilización en proyectos futuros.

Importancia del aumento de datos para entrenamiento

Una vez obtenido el resultado esperado podemos hacer un chequeo para definir la importancia fundamental de haber aumentado los datos.

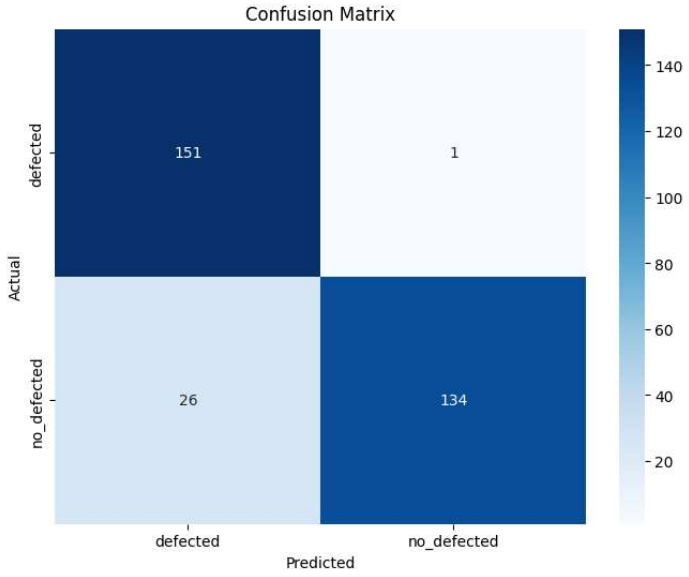
Al correr el modelo final con la base original obtenemos los siguientes resultados:



El modelo sin aumento de datos muestra un rendimiento inicial prometedor, pero enfrenta desafíos significativos en términos de generalización. Aunque alcanza una alta precisión en el conjunto de entrenamiento, su desempeño en los conjuntos de validación y prueba es notablemente inferior y más variable. Esto indica un problema de sobreajuste, donde el modelo se adapta demasiado a los datos de entrenamiento y pierde la capacidad de predecir con precisión datos nuevos.

El análisis de la pérdida confirma esta observación, mostrando una disminución inicial en los conjuntos de validación y prueba, seguida de un aumento, un claro indicador de sobreajuste. La falta de aumento de datos probablemente contribuye a este problema, ya que el modelo no está expuesto a suficientes variaciones durante el entrenamiento.

Los problemas previamente detallados influyen en la clasificación que empeora con respecto al modelo entrenado con una base más amplia:



El sobreajuste, un problema común en el aprendizaje automático, tiene un impacto significativo en la matriz de confusión del modelo sin aumento de datos. Aunque la precisión general puede parecer alta debido a los valores elevados en la diagonal de la matriz, un análisis más detallado revela problemas subyacentes.

El modelo muestra una tendencia a clasificar erróneamente un número considerable de imágenes defectuosas como no defectuosas (falsos negativos). Esto se debe a que el sobreajuste lleva al modelo a priorizar la clase mayoritaria ("no_defected") para minimizar el error de entrenamiento, descuidando la capacidad de reconocer adecuadamente la clase minoritaria ("defected").

Esta incapacidad para generalizar a datos nuevos se refleja en la matriz de confusión, donde los errores se hacen evidentes. A pesar de la alta precisión en la diagonal, el modelo no aprende las características subyacentes que distinguen a las clases, sino que memoriza los datos de entrenamiento.

Comparación de resultados con y sin base ampliada:

	Modelo final				Modelo final sin base ampliada			
	precision	recall	f1-score	support	precision	recall	f1-score	support
defected	0.99	1.00	0.99	911	0.85	0.99	0.92	152
no_defected	1.00	0.99	0.99	958	0.99	0.84	0.91	160
accuracy			0.99	1869			0.91	312
macro avg	0.99	0.99	0.99	1869	0.92	0.92	0.91	312
weighted avg	0.99	0.99	0.99	1869	0.92	0.91	0.91	312

La comparación entre el Modelo final y el Modelo final sin base ampliada revela una clara superioridad del primero en todas las métricas de evaluación. La diferencia más notable se observa en la precisión de la clase "defected", donde el Modelo final demuestra una capacidad superior para identificar correctamente los elementos defectuosos. Además, la diferencia en el número de muestras de cada clase sugiere que el Modelo final se benefició de un conjunto de datos de entrenamiento más amplio y equilibrado. Aunque ambos modelos muestran un buen rendimiento general, el Modelo final se destaca como la opción preferible, especialmente en aplicaciones donde la detección precisa de defectos es crucial.

Prueba del modelo final con datos originales

La prueba del modelo final con datos originales no modificados es fundamental para determinar su aplicabilidad en situaciones reales. Para ello, empleamos una metodología rigurosa que consiste en dos etapas clave:

Entrenamiento con datos aumentados: En esta fase, el modelo aprende a reconocer patrones y características relevantes utilizando un conjunto de datos enriquecido artificialmente mediante una metodología de data augmentation desarrollada previamente en este trabajo.. Esto implica modificar las imágenes originales (por ejemplo, cambiando su color o brillo) para generar variaciones que amplíen la diversidad de ejemplos de entrenamiento.

Evaluación del modelo final con datos originales: Una vez alcanzado el modelo final tras el entrenamiento, se somete a una prueba crucial: se le presentan imágenes originales, sin ninguna modificación, que no fueron utilizadas durante el entrenamiento. Esto simula el escenario de utilizar el modelo en un entorno real, donde se enfrentará a datos nuevos y desconocidos.

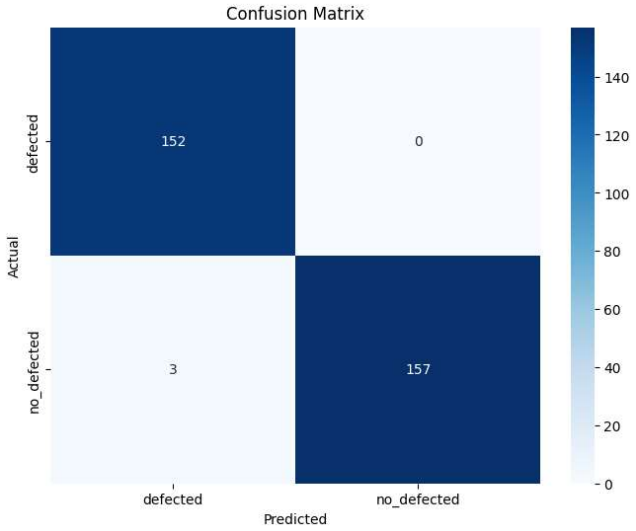
Al entrenar con datos aumentados y evaluar el modelo final con datos originales, recreamos las condiciones de uso práctico del modelo. Esta metodología rigurosa nos permite confiar en que el modelo final no solo se desempeña bien en condiciones de laboratorio con datos controlados, sino que también es capaz de generalizar su conocimiento y clasificar correctamente imágenes en situaciones reales con nuevas bases de datos de impresiones.

Al realizar esta prueba, los resultados obtenidos equiparan los del modelo final testeado con datos de la base aumentada:

	Modelo final testeado con datos nuevos			
	precision	recall	f1-score	support
defected	0.99	1.00	0.99	911
no_defected	1.00	0.99	0.99	958
accuracy			0.99	1869
macro avg	0.99	0.99	0.99	1869
weighted avg	0.99	0.99	0.99	1869

Esto significa que el modelo final obtiene los mismos indicadores de performance que había obtenido con la base aumentada. O sea que se desempeña igual aunque los datos recibidos sean completamente nuevos.

Arroja solo 3 falsos negativos del total de la base de testeo según lo demuestra la matriz de confusión:



El modelo final demuestra un rendimiento excepcional en la tarea de clasificación, con una capacidad muy alta para identificar impresiones defectuosas y prácticamente ningún error al clasificar no defectuosas. Esto sugiere que el modelo está listo para ser implementado en un entorno real y puede ser una herramienta valiosa para asegurar la calidad del proceso de impresión.

Trabajos futuros

Tomando como punto de partida el modelo previamente obtenido, el objetivo es desarrollar un sistema capaz de comparar los resultados de una simulación de impresión 3D con imágenes de la impresión real. Para lograr esto, se ha creado un script en Python que interpreta las instrucciones enviadas a la impresora y genera una representación visual de cada capa del objeto impreso (Ilustración 11 - Imagen generada en Python de la primera capa de impresión de un G-Code.

El modelo desarrollado en el presente trabajo será entrenado utilizando la misma metodología, pero con una nueva base de datos que incluya tanto las simulaciones como las imágenes reales obtenidas de las impresiones realizadas con el mismo g-code utilizado para renderizar las imágenes. Una vez que el modelo haya sido entrenado adecuadamente, se integrará en un dispositivo Arduino. Este dispositivo interactuará con el sistema operativo de la impresora, proporcionándole el resultado de la clasificación realizada por el modelo. Con esta información, la impresora podrá tomar decisiones inteligentes, como continuar o detener el proceso de producción del objeto, basándose en si la impresión real coincide con la simulación esperada (Ilustración 12 - Diagrama del proceso final de predicción de errores en tiempo real.

Este desarrollo busca mejorar la calidad y precisión de las impresiones 3D mediante la comparación automatizada entre simulaciones y resultados reales, utilizando un modelo de aprendizaje automático.

Conclusión

Este estudio demuestra el potencial del aprendizaje automático, específicamente de las redes neuronales convolucionales (CNN), en la detección temprana de errores en impresión 3D. Se ha desarrollado un modelo predictivo eficaz que, tras un exhaustivo proceso de entrenamiento y validación con un conjunto de datos ampliado y diversificado, ha demostrado una alta precisión en la clasificación de impresiones defectuosas y no defectuosas, incluso al testearlo con datos totalmente nuevos.

El modelo no solo identifica correctamente los errores, sino que también sienta las bases para futuros desarrollos, como la implementación de un sistema en tiempo real que permita la corrección de errores durante el proceso de impresión. Este sistema, basado en la comparación de simulaciones con imágenes reales y utilizando un dispositivo Arduino para interactuar con la impresora, tiene el potencial de impulsar la eficiencia y la sostenibilidad en la industria de la impresión 3D, al optimizar los procesos de fabricación aditiva y reducir los residuos.

En última instancia, este trabajo no solo contribuye al avance de la tecnología de impresión 3D, sino que también allana el camino para una producción más eficiente y respetuosa con el medio ambiente en este campo en constante evolución.

El código correspondiente al modelo final se encuentra publicado en el siguiente repositorio: https://github.com/Pemoreira74/Trabajo_final

Índice de ilustraciones

Apéndice de Ilustraciones

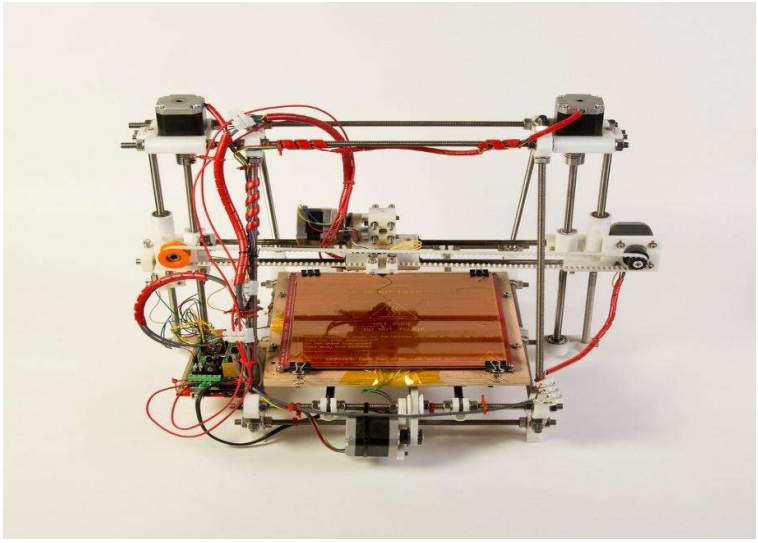


Ilustración 1- Proyecto RepRap

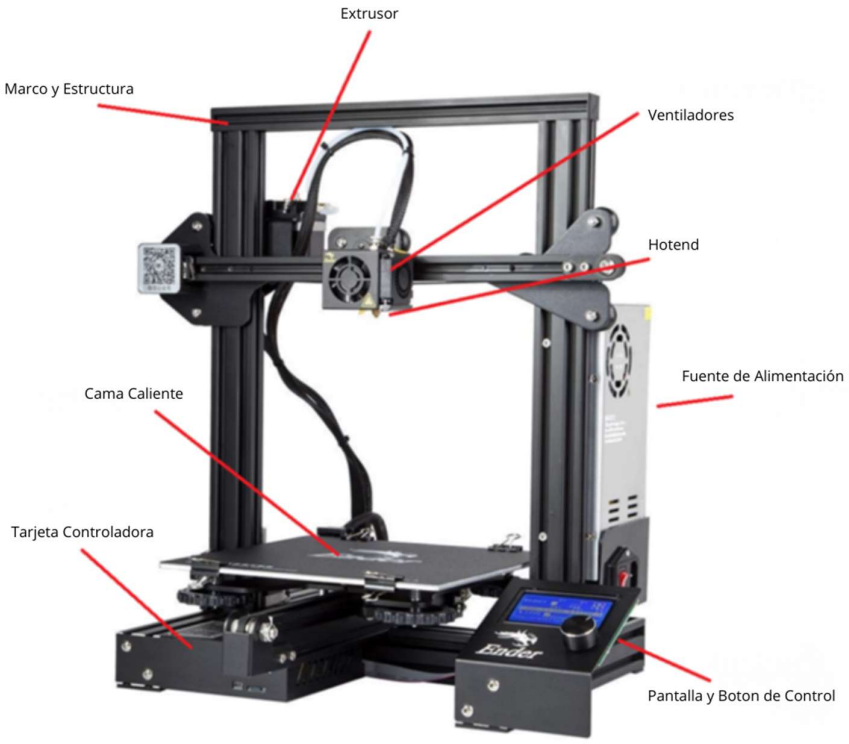


Ilustración 2- Detalle de impresora

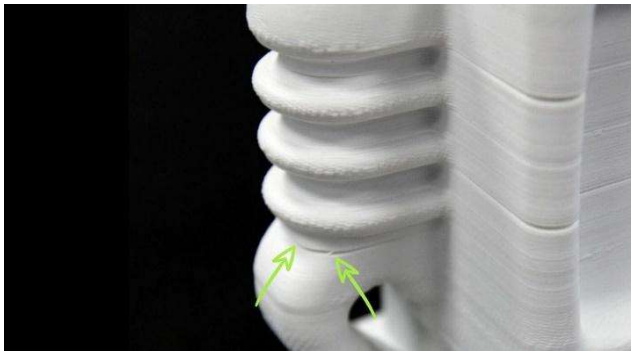


Ilustración 3- Líneas de capa visibles

Ilustración 4 - Retracción

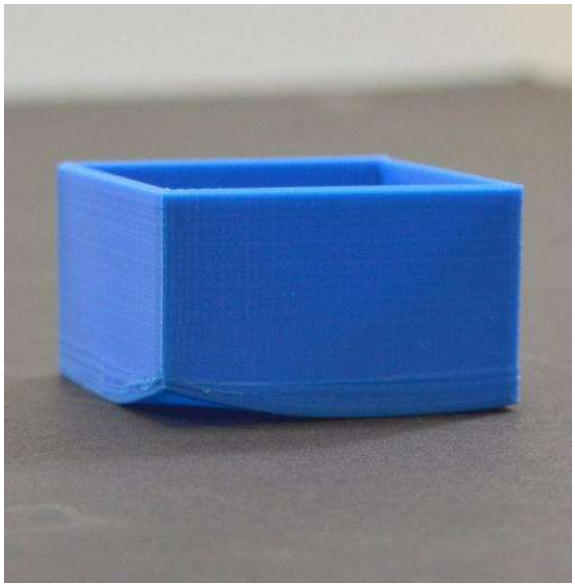


Ilustración 5 - Warping

Ilustración 6 - Diseño de soportes

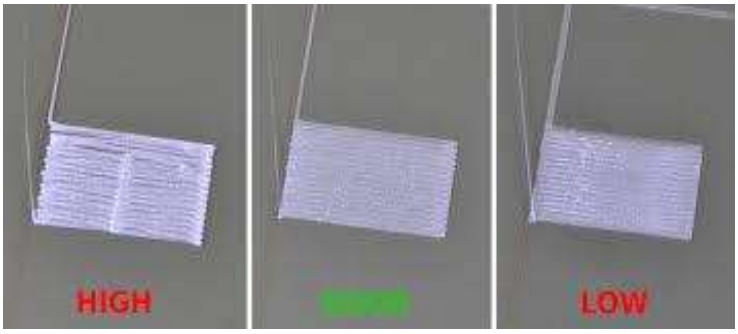


Ilustración 7 - Seteo de la altura de cama

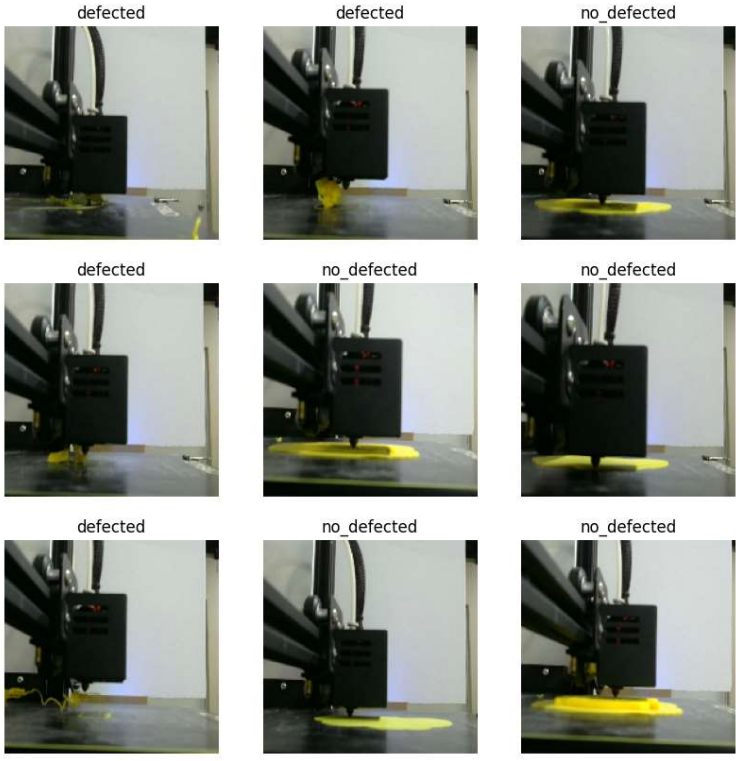


Ilustración 8 - Imágenes clasificadas por clases

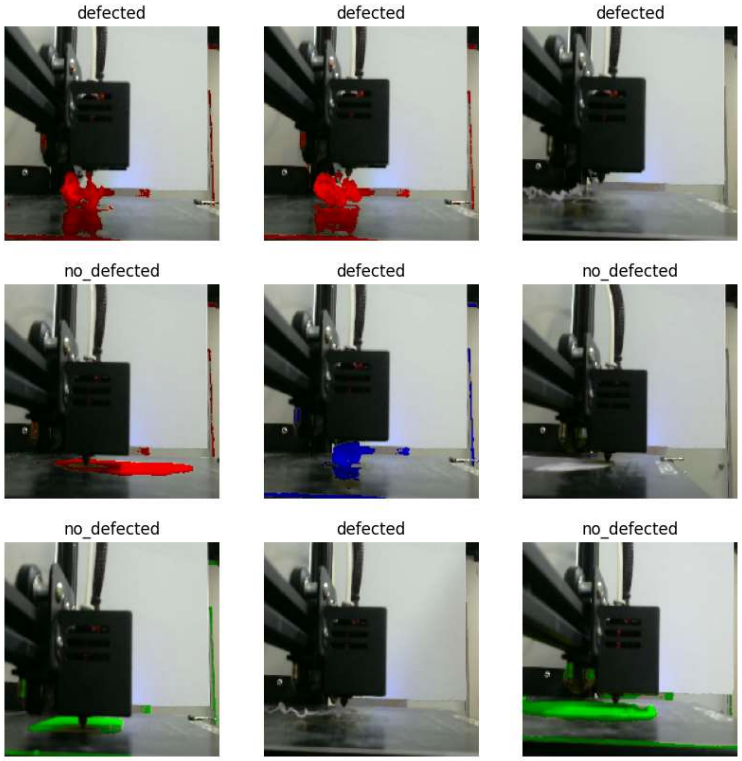


Ilustración 9- Imágenes de la base ampliada clasificadas por clases

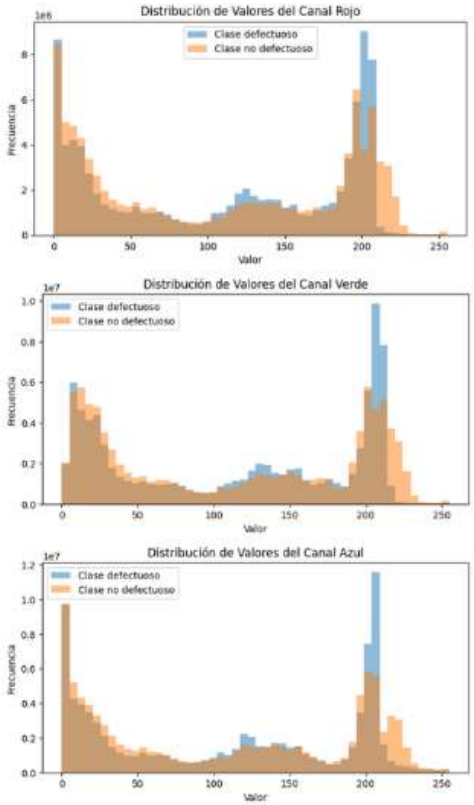


Ilustración 10 - Distribución de valores en los distintos canales de color

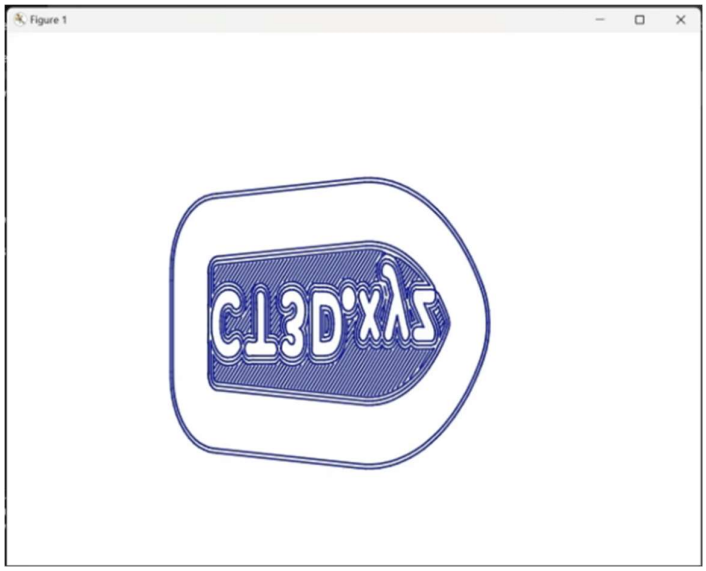


Ilustración 11 - Imagen generada en Python de la primera capa de impresión de un G-Code

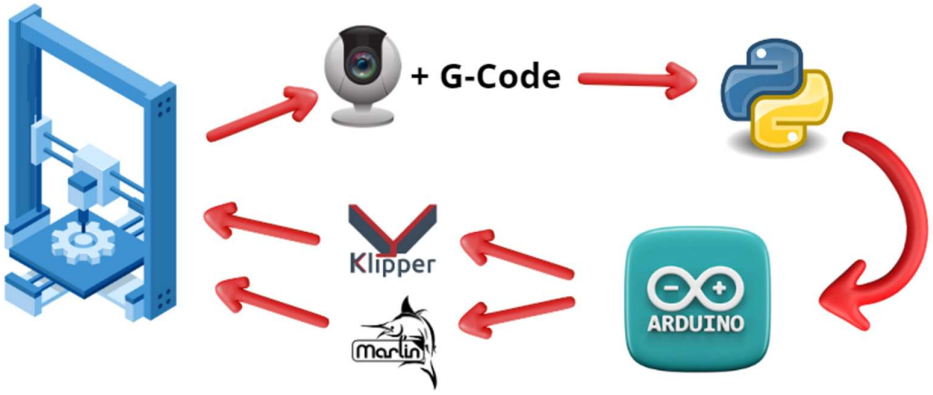


Ilustración 12 - Diagrama del proceso final de predicción de errores en tiempo real

Bibliografía

Prado, Andrea & Arellano, Bernardo & López, Marco & Bandala, Saul. (2016). Metodología de la impresión tridimensional: Modelado de Deposición Fundida (FDM).

Alessandro Ranellucci(2014). Reprap, Slic3r y el Futuro de la Impresión 3D

Izaurieta, F., & Saavedra, C. (2000). Redes neuronales artificiales. Departamento de Física, Universidad de Concepción Chile.

Martín Morlanes, Juan Luis (2020). Optimización de los procesos de impresión 3D

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019.

Torres, Jordi (2020) Python Deep Learning. Introducción práctica con Keras y TensorFlow 2