MUSHROOMS CLASIFICATION

Uri Jared Gopar Morales, ITC A01709413, Tecnologico Campus Querétaro.

***Abstracto*- En el siguiente documento se presentará el análisis de la clasificación de diferentes tipos de hongos.**

I. INTRODUCCION

En nuestro planeta existen números organismos que cumplen con diferentes funciones para el ecosistema. Uno de estos organismos son los hongos, los cuales cumplen su función como descomponedores hasta simbiontes. Sin embargo, la identificación de todas las especies de hongos conocidas es compleja, debido a que varias de estas especies presentan características muy similares, lo que hace que la clasificación de forma manual sea tardada y propensa a errores. En este documento, se utilizará un dataset de imágenes de hongos, el cual contiene varias especies clasificadas por su especie.

El objetivo principal es entrenar un modelo de redes neuronales convolucionales (CNN) para clasificar de manera automática las especies de hongos a partir de las imágenes.

II. MANEJO DE DATOS

El dataset cuenta con un total de 7,763 imágenes de hongos, divididas en 11 especies. Para cargar estas imágenes, se utilizó PyTorch, que proporciona una herramienta que asigna automáticamente etiquetas de clase basadas en las subcarpetas donde se encuentran las imágenes. Esto significa que, si las imágenes están en una carpeta con el nombre "AMANITA", todas las imágenes de esa carpeta recibirán la etiqueta "AMANITA".

Al inspeccionar el dataset, encontré que había imágenes dañadas, lo que significa que estaban vacías o en un formato incorrecto. Para identificar de manera más eficiente dichas imágenes, creé una clase personalizada que detectaba cuáles eran sospechosas.

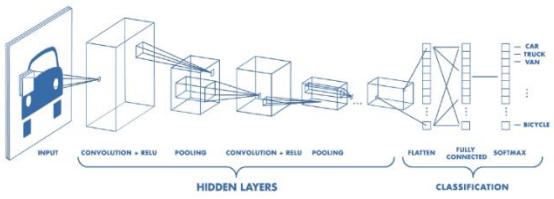
Una vez que se eliminaron las imágenes con errores, el resto fueron procesadas a través de una serie de transformaciones definidas en un pipeline.

- Se redimensionaron a un tamaño de 128x128 pixeles, esto principalmente para que todas sean de la misma dimensión.

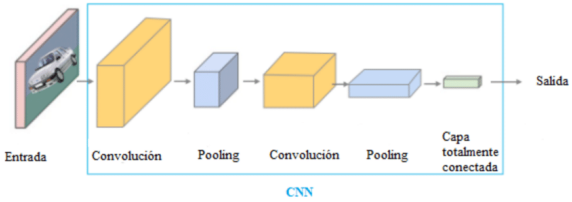
- Se convirtieron a Tensores, el cual es el formato requerido por Pytorch para alimentar los datos al modelo.
- Se normalizaron en un rango de valores entre (-1,1) utilizando los valores medio y desviaciones estándar de los canales de color RGB, para mejorar la convergencia.

III. MODELO

Dicho anteriormente el modelo que decide utilizar fue un modelo de redes neuronales convolucionales (CNN), ya que la arquitectura de este modelo es específicamente para procesar datos como imágenes. Este modelo tiene la capacidad para poder captar patrones espaciales y jerárquicos de las imágenes (bordes, texturas y diferentes formas) a través de sus capas convolucionales.



IV. ARQUITECTURA DE CNN



Capas Convolucionales: Detectan características locales de la imagen, como son los bordes, texturas o colores, estas aplican los filtros Kernels, los cuales son matrices pequeñas que realizan una operación de convolución.

Función de Activación: Introduce no linealidades, permitiendo que la red aprenda relaciones complejas.

Capa Pooling: Reduce las dimensiones espaciales **alto y ancho** esto para quedarse con las características más importantes.

Capa de Aplanamiento (Flattening): Convierte los mapas de características bidimensionales en un vector unidimensional para conectarlo con capas densas.

Capas Densas (Capas Conectadas): Realizan la clasificación usando las características aprendidas, lo que realiza es una conexión entre todas las neuronas de la capa anterior con la capa actual. Conectan todo lo aprendido anteriormente.

Softmax (Activacio): Convierte las salidas en probabilidades para cada clase

V. IMPLEMENTACION

Capas convolucionales: Las capas aplican los filtros “Kernels” sobre la imagen de entrada para detectar características, bordes, texturas y detalles. Utilice 2 capas convolucionales:

- La primera convierte la imagen de entrada en un mapa de características con 32 filtros de 3x3.
- La segunda capa toma la salida de la primera y genera 64 filtros adicionales.

Se utiliza la función de activación ReLU, para poder introducir no linealidades en el

modelo, esto para aprender relaciones más complejas en los datos.

Capas de Max Pooling reducen el tamaño de las representaciones generadas por las capas convolucionales al seleccionar el valor máximo en una región. Ayuda a reducir la dimensionalidad, haciendo el modelo más eficiente, y a la vez ayuda para resaltar las características más importantes detectadas por las capas convulsiónales.

Capas conectadas, estas juntan todas las características extraídas por las capas convolucionales con las salidas del modelo. En mi primera capa conectada de mi modelo contiene 512 neuronas. Mi segunda capa lo que hace es que conecta al número de clases de salida, los tipos de hongo a los que corresponde.

Por terminar tenemos la función de perdida, la cual mide la diferencia entre predicciones del modelo y ajusta los parámetros para mejorar sus predicciones.

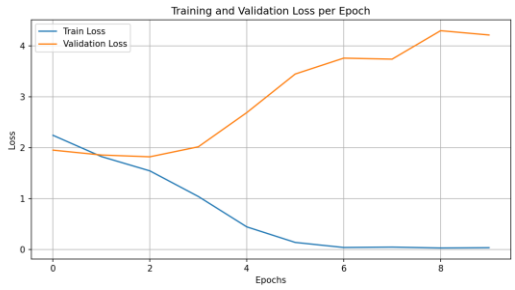
VI. ENTRENAMIENTOS

Para implementar el proceso de entrenamiento de mi red neuronal convolucional, separe mi dataset en 60% entrenamiento, 20% valores de validación y 20% prueba, con la finalidad para poder ver como es el rendimiento de mi modelo y como este a través de las épocas va aprendiendo.

- Entrenamiento 60%: Ayuda para ajustar sus pesos y parámetros, minimizando la perdida por época.
- Validación 20%: Evalúa al modelo durante su entrenamiento, este ajusta los hiperparámetros como la tasa de aprendizaje con datos que nunca ha visto.
- Prueba 20%: Se utiliza para evaluar el rendimiento final del modelo, este nos ayuda para ver qué tan acertado es nuestro modelo con las predicciones de datos que nunca ha visto.

VII. RESULTADOS

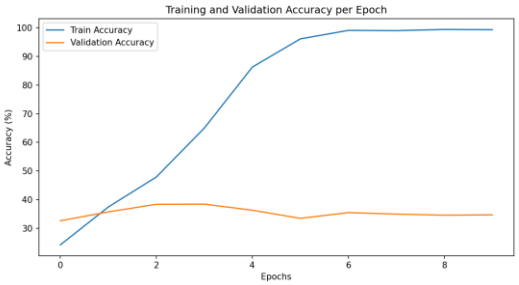
Los resultados obtenidos a través de dicho modelo fueron del 0.3610 lo que nos indica que el modelo clasifica de modelo correcta el 36.10% de las especies de hongos.



Sin embargo, al observar la gráfica anterior, que muestra el *Training Loss* y el *Validation Loss* a lo largo de las épocas, notamos una disminución en la pérdida de entrenamiento, lo que indica que el modelo se está ajustando a los datos. No obstante, al analizar la gráfica del *Validation Loss*, este no sigue el mismo patrón de pérdida. En su lugar, al principio disminuye, pero después de la segunda época aumenta drásticamente.

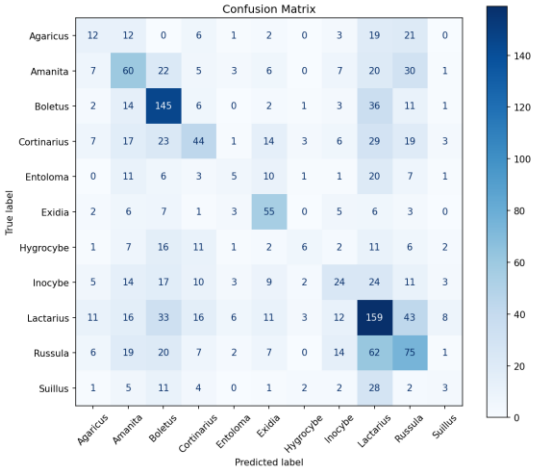
Esto es un indicio de que el modelo está presentando *overfitting*, ya que se ajusta demasiado a los datos de entrenamiento y pierde la capacidad de generalizar correctamente a datos no vistos.

En la gráfica siguiente podemos observar la precisión del modelo tanto en entrenamiento como en validación. Se nota que, en los datos de entrenamiento, el modelo se ajusta rápidamente, mientras que en los datos de validación la precisión se mantiene baja. Esto se refleja en una precisión de validación (*Validation Accuracy*) del 34.72% y una precisión en el conjunto de prueba (*Test Accuracy*) del 36.10%.




```
Entrenamiento Epoch 6/10: 100%|██████████| 146/146 [00:26<00:00,  5.41batch/s, loss=0.0967]
Epoch 6/10, Train Loss: 0.1411, Val Loss: 3.4456, Train Acc: 96.09%, Val Acc: 33.51%
Entrenamiento Epoch 7/10: 100%|██████████| 146/146 [00:27<00:00,  5.38batch/s, loss=0.121]
Epoch 7/10, Train Loss: 0.0415, Val Loss: 3.7574, Train Acc: 99.06%, Val Acc: 35.50%
Entrenamiento Epoch 8/10: 100%|██████████| 146/146 [00:26<00:00,  5.46batch/s, loss=0.00948]
Epoch 8/10, Train Loss: 0.0487, Val Loss: 3.7375, Train Acc: 98.97%, Val Acc: 34.99%
Entrenamiento Epoch 9/10: 100%|██████████| 146/146 [00:26<00:00,  5.49batch/s, loss=0.0949]
Epoch 9/10, Train Loss: 0.0322, Val Loss: 4.2960, Train Acc: 99.38%, Val Acc: 34.60%
Entrenamiento Epoch 10/10: 100%|██████████| 146/146 [00:26<00:00,  5.46batch/s, loss=0.18]
Epoch 10/10, Train Loss: 0.0374, Val Loss: 4.2117, Train Acc: 99.29%, Val Acc: 34.73%
Training complete.
Test Accuracy: 36.10%
```

Para analizar mejor cómo el modelo está clasificando, generé una matriz de confusión. Esto me permitió identificar cuáles clases tienen mejor rendimiento y en cuáles se confunde nuestra CNN. Como resumen, se puede observar que el modelo tiene un buen rendimiento para la clasificación de *Boletus* y *Lactarius*. Por otro lado, las clases con mayores confusiones son *Agaricus* y *Amanita*. Esto puede deberse a que estas clases comparten características muy similares, lo que dificulta su predicción.



VIII. MEJORAS A LOS DATOS

Como podemos observar, mi modelo no lograba predecir correctamente las diferentes especies de hongos contenidas en el dataset. Dado que no mostraba mejoras significativas incluso después de añadir 7 capas convolucionales y 4,000 neuronas, decidí explorar las imágenes a fondo. Durante esta actividad, noté que varias de las imágenes no aportaban valor alguno al modelo, llegando incluso a perjudicar su rendimiento.

Por esta razón, en mi primera iteración, el modelo fue capaz de memorizar gran parte del conjunto de entrenamiento, pero al evaluar con el conjunto de validación, no encontraba similitudes significativas, excepto en algunas imágenes, como en el caso de las especies *Lactarius* y *Boletus*. Estas especies fueron las que el modelo clasificó con mayor precisión, como se mostró anteriormente en la matriz de confusión.

Las imágenes que perjudicaban eran:

- Tallos de los hongos
- Hongos partidos a la mitad
- Hongos vistos por medio de un microscopio
- Diferente posición de los hongos
- Resolución de imágenes de diferente calidad
- Fotos que contenían mucho ruido (plantas,pastos,arboles,etc.)

Para limpiar el dataset, utilicé otro conjunto de imágenes que clasificaba mejor las especies de hongos. Este nuevo conjunto incluía imágenes más claras y con mejor resolución, permitió mejorar la calidad de los datos. Como resultado, mi nuevo conjunto de especies quedó compuesto por: *Amanita, Boletus, Calocera, Calycina, Leccinum versipelle, Nectrina, Parmelia, Phallus impudicus* y *Trametes versicolor*.

IX. MEJORAS AL MODELO.

Con el dataset más limpio, procedí a mejorar mi modelo de CNN. Implementé un total de 6 capas convolucionales, donde la última capa cuenta con 1,024 mapas de características. Estas capas fueron diseñadas para mejorar el aprendizaje de patrones más complejos. Dado que los hongos tienen formas muy similares, el modelo necesita enfocarse en características como colores, texturas y pequeñas variaciones en las formas que distinguen a las diferentes especies.

Utilicé la técnica de *Batch Normalization*, que estabiliza y acelera el proceso de entrenamiento al ajustar las distribuciones de datos durante este proceso. Además, ayuda a reducir el *overfitting*. Incrementé el número de neuronas, alcanzando un total de 4,096, para permitir que el modelo aprenda relaciones más complejas entre las características extraídas.

Para evitar el *overfitting*, también apliqué la técnica de regularización *dropout* con un valor de 0.6. Esto significa que un porcentaje de las neuronas se desactiva aleatoriamente durante el entrenamiento, promoviendo un aprendizaje distribuido y evitando que la red dependa excesivamente de un conjunto específico de neuronas.

A primera vista, estas mejoras pueden parecer demasiado robustas. Sin embargo, es importante recordar que estamos tratando de predecir un total de 9 especies de hongos, cada una con diferentes tipos de información. Además, el dataset es considerablemente grande, con un total de 1,600 imágenes que ocupan un espacio de 1.2 GB. Estas características justifican el enfoque más complejo en el diseño del modelo.

X. HIPERPARAMETROS

Utilicé una tasa de aprendizaje relativamente baja de 0.0005, lo que permite realizar ajustes graduales en los pesos del modelo. Esto evita pasos demasiado grandes que podrían causar oscilaciones en la pérdida, produciendo pesos más estables y mejorando la capacidad del modelo para generalizar en los datos de prueba. Esta configuración ayuda a combatir el ruido presente en las imágenes.

Además, implementé la técnica de *Weight Decay* (decaimiento de peso), que utiliza regularización L2 para penalizar pesos

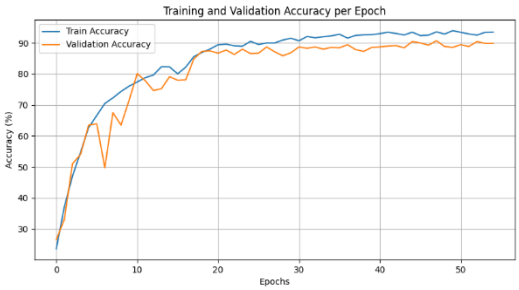
grandes en el modelo y así evitar el *overfitting*. Esto es especialmente útil para reducir el riesgo de que las capas densas, con muchos parámetros, memoricen los datos de entrenamiento.

El número de épocas para entrenar el modelo se estableció en 40, lo que permite que el modelo aprenda lo máximo posible los patrones presentes en los datos sin sobreentrenarse.

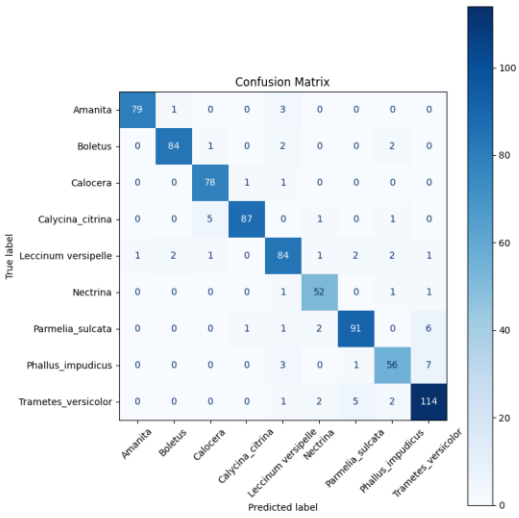
XI. RESULTADOS

Al aplicar estas mejoras y volver a entrenar el modelo de la clasificación, nos proporciona un acurrency del 92%, esto indica que nuestro modelo ya es capaz de reconocer 9 especies de hongos, mostrando una diferencia de aprendizaje mucho mayor que en la primera iteración.

En la primera imagen podemos observar la pérdida en el entrenamiento y en la validación, en la cual observamos como disminuye constantemente, lo que indica el aprendizaje del modelo. Mientras que en la curva de validación disminuye, sin embargo, presenta oscilaciones, indicando que el modelo está alcanzando su capacidad máxima para generalizar las imágenes. Al tener las dos perdidas en un mismo grafico podemos ver como la perdida de validación no diverge de la curva de entrenamiento, lo que indica que nuestro nuevo modelo no sufre de overfitting.



También realice la gráfica de precisión, la cual nos ayuda a ver el porcentaje de imágenes que fueron correctamente clasificados en el conjunto de validación y entrenamiento. En esta podemos apreciar como el conjunto de validación es similar al de entrenamiento, lo que dice que el modelo esta generalizando bien datos no vistos anteriormente. En esta mayormente observamos como en las primeras épocas el modelo va aprendiendo, hasta llegar al 93-95% que es donde se estabiliza.



Por último, podemos observar cómo nuestra matriz de confusión mejoro significativamente a la primera iteración, y como es que ya no se confunde significativamente con las especies conocidas.

XII. CONCLUSION

Al reflexionar sobre la transformación del primer modelo al segundo, entendí la importancia de trabajar de manera iterativa en el desarrollo de modelos. Desde la limpieza del dataset hasta la optimización del modelo, cada mejora ayudó a convertir un modelo básico en una solución más eficiente y precisa. Este proceso me enseñó lo fundamental que es analizar y depurar los datos, así como ajustar la estructura del modelo y sus configuraciones para obtener mejores resultados.

REFERENCIAS:

- Androbomb. (2019, 11 diciembre). *Using CNN to classify images w/PyTorch*. Kaggle. https://www.kaggle.com/code/androbomb/using-cnn-to-classify-images-w-pytorch
- GeeksforGeeks. (2024, 21 mayo). *Image Classification using CNN*. GeeksforGeeks. https://www.geeksforgeeks.org/image-classifier-using-cnn/
- *Mushroom Classification Dataset*. (2023, 26 mayo). Kaggle. https://www.kaggle.com/datasets/lizhecheng/mushroom-classification?select=archive
- *Mushroom species*. (2023, 6 septiembre). Kaggle. https://www.kaggle.com/datasets/thehir0/mushroom-species
- *Training a Classifier — PyTorch Tutorials 2.5.0+cu124 documentation*. (s. f.). https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html