Modelo de clasificación de residuos

Autora: Andrea Medina Rico **Matrícula:** A01705541

Abstracto. Este reporte detalla la creación e implementación de un modelo para la clasificación de residuos de acuerdo a seis categorías. La herramienta principal utilizada es Tensorflow para el entrenamiento. **Explicar con qué fue entrenado y métricas finales.**

I. INTRODUCCIÓN

La práctica de separar correctamente la basura tiene una gran área de oportunidad en nuestro país. A partir del 1° de enero de 2026, será obligatoria en la Ciudad de México. Para facilitar y fomentar el hábito en los ciudadanos, se propone un modelo capaz de identificar el tipo de residuo a través de una imagen. Se busca demostrar que se puede identificar un tipo de residuo con alto nivel de confianza a partir de una imagen del mismo.

II. PREPROCESAMIENTO DE LOS DATOS

a. Descripción del conjunto de datos

Las imágenes utilizadas para el entrenamiento provienen del conjunto de datos Garbage Dataset Classification de Kaggle. Contiene 6 carpetas, cada una siendo una categoría de residuos:

- *cardboard:* cajas de cartón
  - 2214 imágenes (15.8%)
- *glass:* envases de vidrio
  - 2500 imágenes (18%)
- *metal:* envases metálicos (latas)
  - 2084 imágenes (15%)
- *paper:* diversas presentaciones de papel (periódicos, revistas, sobres, cartas, entre otros)
  - 2315 imágenes (16.6%)
- *plastic:* envases de plástico
  - 2288 imágenes (16.6%)
- *trash:* residuos de todo tipo que no encajan en ninguna categoría previa (médicos, electrónicos, ropa, envolturas, entre otros)
  - 2500 imágenes (18%)

b. Separación de datos

El conjunto de datos fue dividido para entrenamiento y pruebas en *train, validation* y *test* con un porcentaje de 70-10-10%, respectivamente. La cantidad final de imágenes es de 11,121 para entrenamiento, 1390 de validación y 1390 de pruebas.

c. Normalización

Tras la separación, los datos fueron normalizados para tenerlos en una escala similar. De un rango de 1 a 255 en cada pixel, pasaron a un rango entre 0 y 1. Esta práctica ayuda al modelo a converger más rápido y encontrar los pesos adecuados del modelo sin sesgo hacia los valores más elevados.

III. REDES NEURONALES CONVOLUCIONALES (CNN)

Las CNN son redes neuronales con la capacidad de extraer características de su entrada, ideales para problemas de clasificación de imágenes. Cuentan con los siguientes elementos:

- Entrada
  La entrada, en este caso, es el conjunto de imágenes. Cada imagen tiene tres dimensiones (altura, ancho y profundidad), siendo la

última los 3 canales del formato RGB.

- Capa convolucional
  Extrae características a partir de un filtro, una matriz (de 3x3 en este caso) que recorre la imagen completa.

- Capa de activación
  Agrega no linealidad a la red, mejorando su *accuracy*, pues la mayoría de las relaciones no son lineales.

- Capa de agrupamiento
  Reduce el tamaño de la imagen, disminuyendo la posibilidad de *overfitting* y el costo del entrenamiento. En este proyecto, se utiliza *Average pooling*.

- Capa de aplanamiento
  Transforma la matriz de la imagen en un vector de una dimensión. Esto con el objetivo de que sus valores sean al entrada a una capa densa.

- Capa densa
  Capa densa *fully connected* que realiza la clasificación de la entrada.

a. Funciones utilizadas

Como optimizador, se utiliza ADAM (*Adaptive Moment Estimation*). Éste ajusta la tasa de aprendizaje durante el entrenamiento.

Como función de activación se utiliza ReLU. ReLU convierte todo valor menor a 0 en un 0 y mantiene los valores mayores a 0 con su magnitud tal cual. Resulta ser una función sencilla de implementar y de bajo costo.

Como función de pérdida se utiliza *sparse categorical cross entropy,* útil para clasificación multiclase.

Como función de activación final se utiliza *softmax*.

b. Métricas de evaluación

La métrica principal de evaluación es *accuracy* (exactitud), la cual mide la proporción de clasificaciones adecuadas de todas las clasificaciones hechas. Se considera adecuada debido a que la proporción entre las clases está equilibrada, siendo la menor de 15% de los datos y la mayor de 18%. Además, por la naturaleza del problema a resolver, no existen casos o errores más costosos que otros.

IV. MODELO LÍNEA BASE

Se implementa un modelo inicial como línea base a mejorarse con arquitecturas y modelos más trabajados. El modelo línea base de este proyecto es una CNN que recibe la imagen en su tamaño original (256, 256, 3). La primera capa corresponde a una Convolución 2D de dieciséis filtros, cada uno de tamaño (3, 3). En esta, la función de aplicación aplicada es ReLU. Posteriormente, se realiza un *MaxPooling* para reducir tamaño. Se aplica una segunda convolución con ocho filtros con el mismo tamaño, activación y *pooling* que el la anterior. Ambas capas tienen un número reducido de filtros al ser únicamente planteado como base para observar el comportamiento del modelo.

Una vez que la salida de las convoluciones se aplana en un solo vector, se aplica una capa densa de 128 neuronas con activación ReLU. La última capa densa contiene 6 neuronas, una por cada clase, y se activa con *softmax* para obtener la probabilidad de que pertenezca a cada clase.

c. Resultados

El modelo obtuvo un *accuracy* de 0.9936 en entrenamiento y 0.636 en validación,

teniendo una diferencia de 0.3576 entre ambas. Se diagnostica con *overfitting,* siendo incapaz de replicar sus buenos resultados del entrenamiento al momento de la validación.

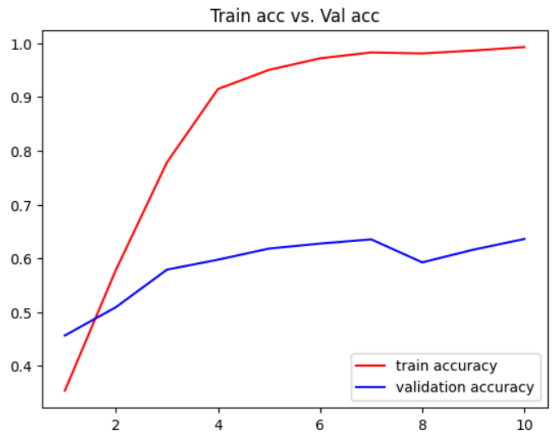

Figura 1. Gráfica de *accuracy* entre *train* y *validation* del modelo línea base.

Esto puede significar una tendencia del modelo a tener un buen desempeño por la naturaleza de los datos, pero un gran *overfitting* si no se implementa adecuadamente.

V. ELECCIÓN DE MODELO

Se comparan dos modelos inicialmente, evaluando su comportamiento y eligiendo uno de ellos para su mejora posterior.

a. MobileNet V3 Large

Se utiliza con tres parámetros específicos. La entrada es del mismo tamaño de las imágenes y mantiene los tres canales (256, 256, 3). Se específica que no incluya la capa densa que viene con este, pues fue creada específicamente para clasificar las imágenes de ImageNet. Lo que sí se usa sobre ImageNet son los pesos que el modelo obtuvo de su entrenamiento con ese conjunto de datos. Si bien no es el mismo conjunto de datos, está entrenado en el reconocimiento de diversos objetos. Las capas que componen al modelo permanecen congeladas para que no se continúen entrenando, pues poco a poco se perderían las características del modelo que se buscan aprovechar.

Al no incluir un clasificador, se agrega una capa de *Average Pooling* y una capa densa con 6 neuronas (por las 6 categorías a clasificar). La activación final se realiza con *softmax*.

b. Efficient Net B0

La aplicación del modelo al proyecto sigue los mismos pasos que se realizaron para MobileNet V3 Large. La forma de la entrada, pesos y capas de clasificación son las mismas. Asimismo, los pesos utilizados son de ImageNet y no se utiliza la capa clasificadora que viene con el modelo.

c. Comparación de resultados

El modelo de MobileNet V3 tuvo un *accuracy* de 0.3587 en entrenamiento y 0.334 en validación. Se observa una ligera tendencia hacia el sobreajuste. Sin embargo, a pesar de no tener un *accuracy* muy alto, se observa un desempeño óptimo.

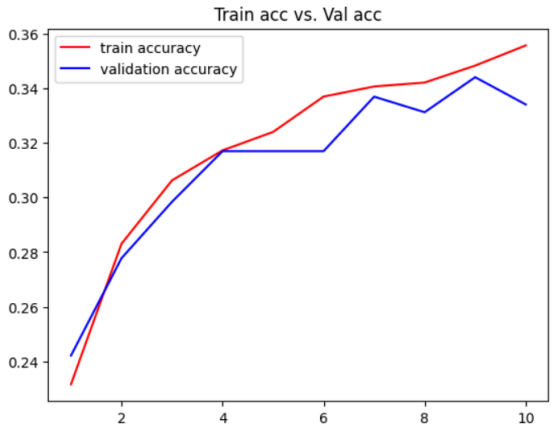

Figura 2. Gráfica de *accuracy* entre *train* y *validation* del modelo MobileNet V3.

Por otro lado, el modelo de Efficient Net B0 obtuvo un *accuracy* de 0.1761 en entrenamiento y 0.1652 en validación. Se

observa un comportamiento inestable en la parte de validación y, en general, obtuvo un *accuracy* menor que el modelo anterior.

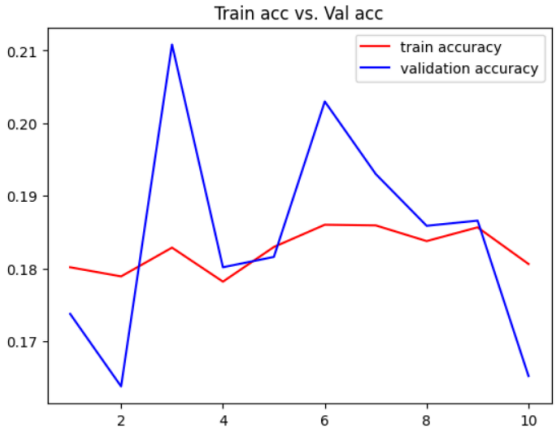

Figura 3. Gráfica de *accuracy* entre *train* y *validation* del modelo EfficientNet B0.

Se decide continuar con la mejora del modelo de MobileNet V3 Large.

VI. MEJORA DEL MODELO

a. 1° Iteración

La forma de extraer el modelo MobileNet V3 Large permanece igual. La diferencia radica en las capas que se agregan para la clasificación.

Después de la arquitectura y el *average pooling,* se agregan dos capas de regularización. La primera es *Batch Normalization*, la cual normaliza las activaciones de cada capa para dejarlas en un rango entre 0 y 1. La segunda es *Dropout,* la cual hace que cada pasada, solo se utilicen algunas neuronas y otras se desactiven. Así, se reduce la dependencia de ciertas neuronas y la posibilidad de sobreajuste. Se aplicó *Dropout* con un valor de 0.4.

Posterior a eso, se aplicó una capa densa de 256 neuronas activada con ReLU y se finalizó con la capa densa clasificadora (seis neuronas y *softmax)*.

Se probó con 10 épocas para pre-visualizar su comportamiento antes de ser entrenado a fondo.

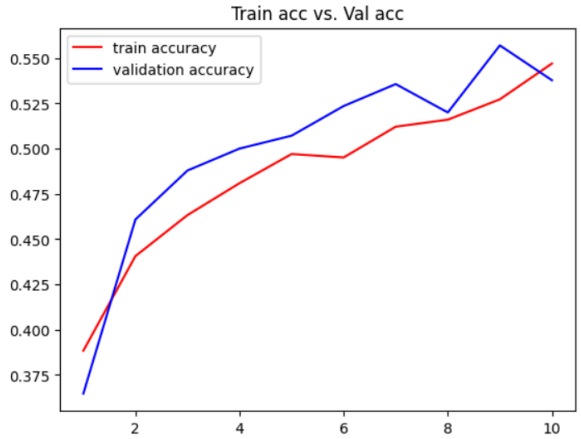

Figura 4. Gráfica de *accuracy* entre *train* y *validation* de la primera mejora del modelo.

El modelo obtuvo 0.5435 en entrenamiento y 0.5377 en validación. A pesar de ser pocas épocas, se observa un buen desempeño al grado de que, en ocasiones, la validación supera los resultados del entrenamiento.

b. 2° Iteración

En la segunda iteración, se agregó una capa densa después de la de 256 neuronas, con un total de 128. Esto puede ocasionar un sobre ajuste porque vuelve al modelo demasiado complejo para la solución. Sin embargo, se probó para evaluar su comportamiento.

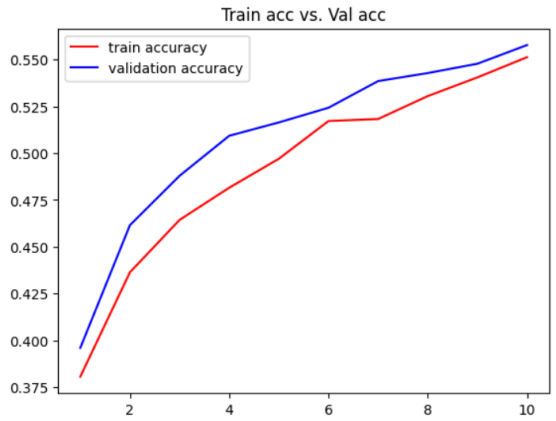

Figura 5. Gráfica de *accuracy* entre *train* y *validation* de la segunda mejora del modelo.

El modelo obtuvo un *accuracy* de 0.5464 en entrenamiento y 0.5577 en validación. A comparación del primer modelo, este parece tener un cambio más estable con el mismo número de épocas.

c. 3° Iteración

Al parecer haber un comportamiento más estable en la segunda mejora, se implementó este mismo con la diferencia de que se aplicaron 150 épocas. Se aumentó el número de épocas considerablemente puesto que este ya es un número de épocas aproximado a la realidad de lo que necesita un modelo para entrenarse.

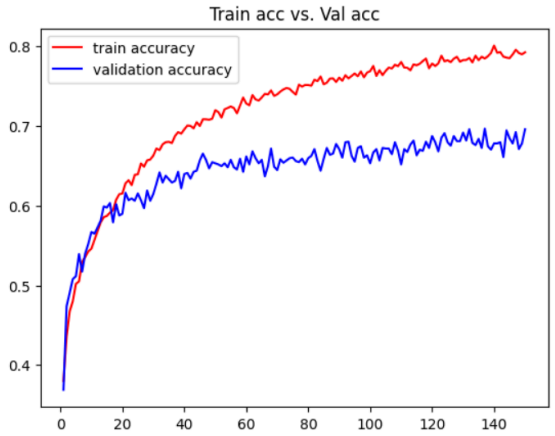

Figura 6. Gráfica de *accuracy* entre *train* y *validation* de la tercera mejora del modelo.

El modelo obtuvo un *accuracy* de 0.79 en entrenamiento y 0.6959 en validación. Se observa claramente el sobreajuste que existe entre ambas, teniendo una diferencia de 0.1. Si bien no es una diferencia demasiado grande, es un claro sobreajuste que iría aumentando. Al agregar épocas, *train* seguiría mejorando pero *validation* se quedaría igual de estancado como hasta el momento. Aproximadamente desde la capa 50, no hay una mejora notable de la validación. Con la siguiente mejora, se busca reducir las épocas y la complejidad del modelo, buscando reducir el sobreajuste. Esto debido a que ambos suelen ser razones de *overfitting*.

d. 4° Iteración

Para esta iteración, se toma el primer modelo mejorado con la diferencia de que se aplican 100 épocas al mismo. Se redujo el número de épocas a comparación del pasado por la observación de que se estancaba en cierto punto y solo aumentaba el sobreajuste.

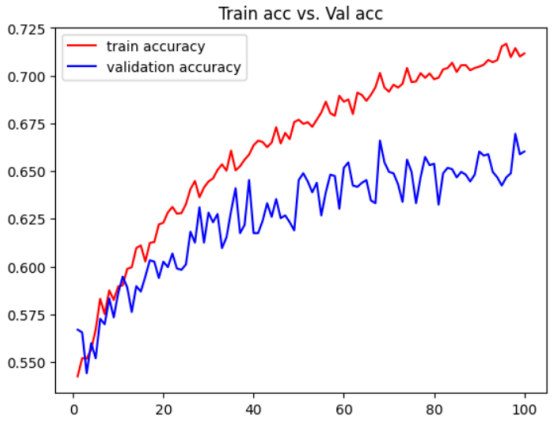

Figura 7. Gráfica de *accuracy* entre *train* y *validation* de la cuarta mejora del modelo.

El modelo obtuvo un *accuracy* de 0.7064 en entrenamiento y 0.6603 en validación. Aunque visualmente parece una gran diferencia entre ambas líneas, la escala de valores es más pequeña. La diferencia entre entrenamiento y validación es de menos del 0.05.

e. Elección final

La elección final es el último modelo mejorado debido a que tiene un *accuracy* alto con el sobreajuste más bajo al momento de ser entrando con el número de épocas adecuado

VII. CONCLUSIÓN

El modelo obtenido logra tener un desempeño adecuado para lo que la problemática requiere. En una futura iteración, se comparará el resultado con el conjunto de datos destinado a pruebas para valorar mejor su desempeño.

VIII. REFERENCIAS

https://www.ibm.com/mx-es/think/topics/convolutional-neural-networks

GeeksforGeeks. (2025a, julio 11). Introduction to Convolution Neural Network. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/introduction-convolution-neural-network/

Datos numéricos: Normalización. (s. f.). Google For Developers. https://developers.google.com/machine-learning/crash-course/numerical-data/normalization?hl=es-419

Clasificación: Exactitud, recuperación, precisión y métricas relacionadas. (s. f.). Google For Developers. https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall?hl=es-419

GeeksforGeeks. (2025d, octubre 4). What is Adam Optimizer? GeeksforGeeks. https://www.geeksforgeeks.org/deep-learning/adam-optimizer/

https://medium.com/@RobuRishabh/understanding-and-implementing-mobilenetv3-422bd0bdfb5a

GeeksforGeeks. (2025c, julio 26). What is Sparse Categorical Crossentropy. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/what-is-sparse-categorical-crossentropy/

https://medium.com/@datasciencejourney100_83560/regularization-techniques-in-deep-learning-3de958b14fba