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. El modelo final, generado con MobileNet V3 Large, tiene un 61.75% de accuracy.

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 la posibilidad de 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)
  - 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 80-10-10%, respectivamente. La cantidad final de imágenes es de 11,121 para entrenamiento, 1390 de validación y 1390 de pruebas. En las gráficas, se visualiza la comparación entre entrenamiento y validación únicamente, pues la parte de pruebas está destinada a usarse al final.

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 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.

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

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

a. Funciones utilizadas

- Optimizador: se usa ADAM (*Adaptive Moment Estimation*). Éste ajusta la tasa de aprendizaje durante el entrenamiento.
- Agrupamiento: *Average pooling*, la cual mantiene el promedio de todas las celdas seleccionadas.
- Activación: ReLU, la cual 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.
- Pérdida: *sparse categorical cross entropy,* útil para clasificación multiclase.
- Activación final: se usa *softmax*. Convierte el resultado final perteneciente a cada clase en un número entre 0 y 1, es decir, una probabilidad. La suma de todas las clases da un resultado de 1. La clase con mayor valor es la que se selecciona como la clase predicha, pues es de la que el modelo está más seguro que pertenece la imagen.

b. Métricas de evaluación

La métrica principal de evaluación es *accuracy*, la cual mide la proporción de clasificaciones adecuadas de todas las clasificaciones hechas. Se utiliza 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, buscando mejorarlo posteriormente 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).

1. La primera capa corresponde a una Convolución 2D de 16 filtros, cada uno de tamaño (3, 3), junto con la activación ReLU.
2. Posteriormente, se realiza un *MaxPooling* para reducir tamaño.
3. Se aplica una segunda convolución con 8 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.

4. Se aplana la salida en un solo vector.
5. Se aplica una capa densa de 128 neuronas con activación ReLU.
6. Se aplica la última capa densa con 6 neuronas, una por cada clase, y se activa con *softmax* para obtener la probabilidad por 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.

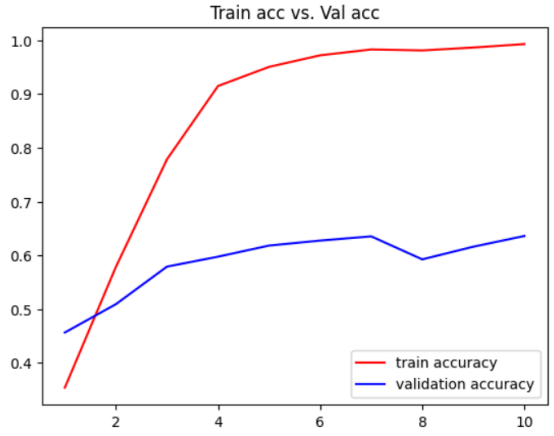

Figura 1. Gráfica de *accuracy* entre *train* y *validation* del modelo línea base.

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

Los tiempos

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

a. MobileNet V3 Large

MobileNet V3 Large es una CNN ligera que combina las mejores características de las versiones anteriores de la familia MobileNet. Tiene dos características principales:

1. *Depthwise convolutions:* Procesa cada canal de la imagen por separado y luego los junta con una convolución 1 x 1.
2. *Bloques invertidos y cuellos de botella:* reduce el tamaño de la entrada, la procesa y la expande a su tamaño original.

Esta arquitectura, por sus características, resulta adecuada para relativamente pocos datos.

El modelo se extrae con cuatro especificaciones.

1. La entrada es del mismo tamaño de las imágenes y mantiene los tres canales (256, 256, 3).
2. Se específica que no incluya la capa densa que viene con este, pues fue creada específicamente para clasificar las imágenes de ImageNet.
3. 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.
4. Las capas que componen al modelo permanecen congeladas para que no se continúen entrenando, pues esto implicaría perder 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

Efficient Net B0 es un modelo de aprendizaje profundo con la característica principal de *compound scaling,* que aumenta de forma equilibrada la profundidad, el ancho y el tamaño de entrada, logrando alta precisión. Resulta una buena opción en conjuntos de datos grandes y con mucho ruido. Si bien este dataset no tiene gran tamaño, se prueba por ser un modelo muy preciso.

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. A pesar de no tener un *accuracy* muy alto, se observa un desempeño óptimo.

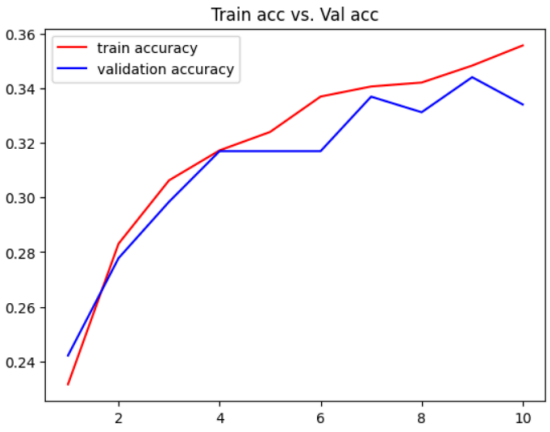

Figura 2. Gráfica de *accuracy* entre *train* y *validation* del modelo MobileNet V3.

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.

Por estas razones, se decide continuar con la mejora del modelo de MobileNet V3 Large.

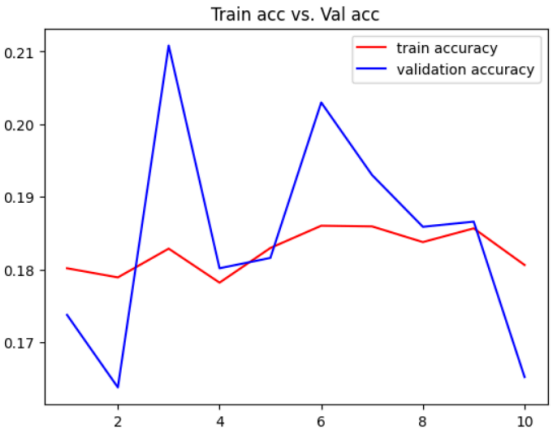

Figura 3. Gráfica de *accuracy* entre *train* y *validation* del modelo EfficientNet B0.

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. En esta iteración, se agregan dos capas de regularización.

1. Después de la arquitectura, se agrega un *average pooling.*
2. Se aplica *Batch Normalization*, la cual normaliza las activaciones de cada capa para dejarlas en un rango entre 0 y 1.
3. Se aplica *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.
4. Se aplica una capa densa de 256 neuronas activada con ReLU
5. Finaliza con la capa densa clasificadora (6 neuronas y *softmax)*

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

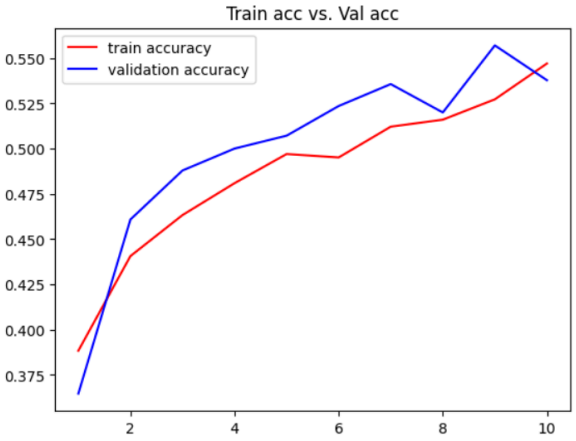

Figura 4. Gráfica de *accuracy* entre *train* y *validation* de la primera mejora del modelo.

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.

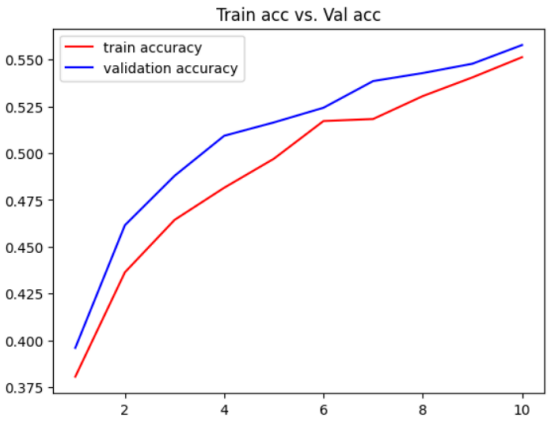

Figura 5. Gráfica de *accuracy* entre *train* y *validation* de la segunda mejora del modelo.

El modelo obtuvo un *accuracy* de 0.5464 en entrenamiento y 0.5577 en validación. A comparación de la primera iteración, esta 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 150 épocas. Este ya es un número de épocas aproximado a la realidad de lo que necesita un modelo para entrenarse.

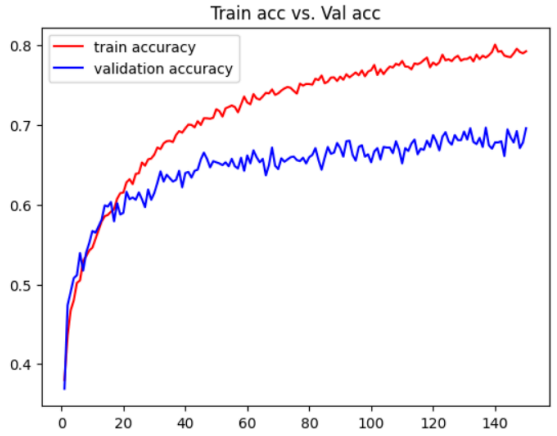

Figura 6. Gráfica de *accuracy* entre *train* y *validation* de la tercera mejora del modelo.

El modelo obtuvo un *accuracy* de 0.79 en entrenamiento y 0.6959 en validación. Se observa una diferencia de 0.1 entre ambas líneas. Si bien no es una diferencia demasiado grande, podría convertirse en sobreajuste 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

Se toma el modelo de la primera iteración con un número de 100 épocas. Se redujo el número de épocas a comparación del pasado por la observación de que se estancaba en cierto punto.

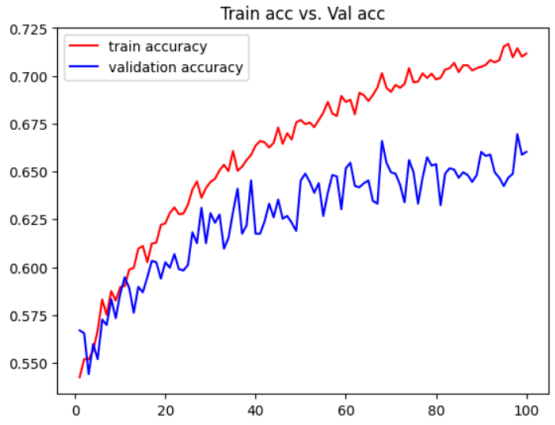

Figura 7. Gráfica de *accuracy* entre *train* y *validation* de la cuarta mejora del modelo.

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. 5° Iteración

En este modelo, se aplica una sola capa densa de 512 neuronas después del dropout, para finalizar con la capa clasificadora de 6 neuronas y softmax.

El modelo obtuvo un *accuracy* de 0.698 en entrenamiento y 0.656 en validación. Si bien no ha sido el de accuracy más alto, es el que menor diferencia ha tenido entre entrenamiento y validación con un valor de 0.042, por lo que es modelo que se mantiene para las mejoras futuras.

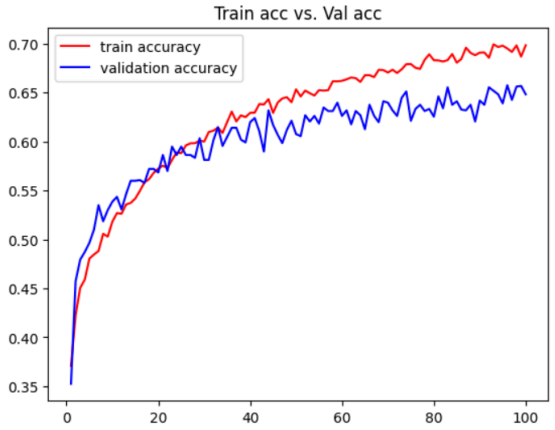

Figura 7. Gráfica de *accuracy* entre *train* y *validation* de la quinta mejora del modelo.

VII. DATA AUGMENTATION

Buscando una mejora en las predicciones, se aplicó *data augmentation*. Consiste en crear nuevas imágenes con alteraciones como rotaciones, recortes, vueltas, y aplicarlas al conjunto de entrenamiento. Se busca que mejore la generalización del modelo y reduzca el sobreajuste. Así, el modelo aprende a identificar la figura a pesar de las transformaciones.

En cada época, se decidió aplicar las transformaciones únicamente en el 30% de los datos. Funciona para aplicarlas sin ralentizar tanto el entrenamiento. Se aplicaron los siguientes cambios:

1. Vuelta a la imagen horizontalmente
2. Rotación de 40 grados
3. Zoom del 20%
4. Desplazamiento del 20%
5. Recorte del 20%

a. 1° Iteración

Se tomó la estructura del último modelo (5° iteración) y se aplicaron las primeras 3 transformaciones. Obtuvo un accuracy de 0.5764 en entrenamiento y 0.6175 en validación. Tuvo menores valores que modelos anteriores, por lo que se agregaron las últimas dos transformaciones.

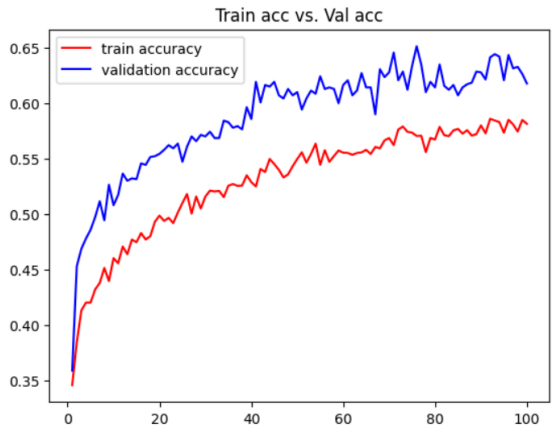

Figura 7. Gráfica de *accuracy* entre *train* y *validation* de la primera mejora con data augmentation.

b. 2° Iteración

Aplicando todas las transformaciones, se obtuvo un accuracy de 0.597 en entrenamiento y 0.6496 en validación, con un total de 190 épocas. Comparándolo con el modelo anterior, a las 100 épocas obtuvo un accuracy de 0.5865 y 0.6296, respectivamente.

VIII. COMPARACIÓN FINAL

Modelo	Train acc	Val acc	Diff. acc
Base	0.9855	0.6717	0.3138
MobV3	0.3547	0.3440	0.0107
EffB0	0.1761	0.1652	0.0109
Iter. 1	0.5435	0.5377	0.0058
Iter. 2	0.5464	0.5577	0.0113
Iter. 3	0.7900	0.6959	0.0941
Iter. 4	0.7064	0.6603	0.0461
Iter. 5	0.6980	0.6560	0.0420
DA 1	0.5764	0.6175	0.0411
DA 2	0.5970	0.6496	0.0526

Figura 8. Tabla de comparación de resultados entre modelos generados.

Se eligió como modelo final el primero con data augmentation (subrayado en amarillo) por las siguientes razones:

1. El aumento de datos mejora las predicciones en basura real obtenidas de internet.
2. Si bien el segundo modelo de data augmentation presenta un accuracy mayor, la diferencia es de 3%. Para el tiempo que necesita el segundo modelo para entrenarse, no es una mejora significativa.
3. Es un modelo que generaliza adecuadamente, pues no tiene overfitting.

IX. CONCLUSIÓN

El modelo final tiene un desempeño adecuado para lo que la problemática requiere. En el peor de los casos, al ser incapaz de clasificar un objeto, lo manda a la basura general. Una posible mejora sería eliminar esta categoría para evaluar el comportamiento del modelo y su certeza por clase.

X. REFERENCIAS

Dataset utilizado: https://www.kaggle.com/datasets/zlatan599/garbage-dataset-classification

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

GeeksforGeeks. (2025, 17 noviembre). Softmax Activation Function in Neural Networks. GeeksforGeeks. https://www.geeksforgeeks.org/deep-learning/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/

GeeksforGeeks. (2025a, julio 23). EfficientNet Architecture. GeeksforGeeks. https://www.geeksforgeeks.org/computer-vision/efficientnet-architecture/#efficientnetb0-architecture-overview