Clasificador De Imágenes De Basura Utilizando Pre Entrenamiento VGG-16: Mejora En Precisión De 75% A 86%-90%

Resumen

El estado del arte de un clasificador de imágenes actual, suelen no utilizar arquitecturas que sean construidas de "scratch", sino que suelen utilizar modelos pre entrenados. Aunado a lo anterior, la clasificación de imágenes tiene demasiadas variantes, consecuencia de los distintos formatos que puede llegar a tener una imagen, como por ejemplo: número de píxeles, calidad de la imágen, color de la imagen tonos RGB o simplemente en escala de grises y además RGBA (donde se añade transparencia). Agregando, es de suma importancia llegar a responder varias preguntas antes de tomar la decisión de si, es bueno o no tomar un modelo pre-entrenado. Algunas de las preguntas son: ¿Cuento con datos suficientes, cómo para que mi el modelo entrenado con "x" arquitectura construida sea capaz de generalizar?, ¿Es un problema de clasificación de dos clases?, ¿Es un problema que involucra varias clases?, ¿Cuánto tiempo se tiene para generar los datos?, ¿Requiere datos adicionales a los inicialmente seleccionados? Generalmente, si se cuenta con poca cantidad de datos, es mucho mejor utilizar un modelo pre-entrenado, puesto que ya son modelos de Inteligencia Artificial, que han sido estudiados y entrenados con grandes cantidades de datos. [3]

Todas las preguntas anteriores, fueron contestadas y fueron la base para iniciar con el proceso de construcción del modelo. En este problema, se contaba con un pequeño set de datos un total de 4752 imágenes, distribuidas de manera no uniforme en 9 clases [1], extraído de Kaggle, además de esto, como se comenta en [3] y en [4][5][6][7] es bastante usual utilizar un modelo pre-entrenado para un clasificador de imágenes, ejemplos de estos modelos es: VGG16. Dados los resultados obtenidos en el artículo de investigación siguiente [4] como clasificador de imágenes de basura, donde únicamente se ha utilizado el modelo pre-entrenado VGG16, se ha obtenido una precisión del 75%.

En un clasificador de imágenes, suelen aparecer varios "problemas", y que, dada la experimentación de esta investigación, es de suma importancia resolver esos problemas antes de entrenar un modelo. La estandarización del tamaño de imágenes, uso de algoritmos que hacen la clasificación "más sencilla", como por ejemplo: reducción de ruido o aumento de ruido, algoritmo de detección Canny (señalar en un cuadro el objeto a analizar), etc [6][7]. Un desbalance de datos/imágenes por categorías, generando sesgos al momento de predecir hacia ciertas clases o incluso, que el modelo aprenda patrones, pero no llegue a generalizar debido a que aprendió únicamente los de una clase (overfitting). En el presente, con base en buenas prácticas y experimentaciones, el mejor resultado ha sido un modelo capaz de llegar a una precisión de validación del 87%

Introducción

El uso y mejora de inteligencia artificial, está en una era en la que, claramente está de escalada exponencial, encaminando el destino de la siguiente revolución industrial en un par de años. La

inteligencia artificial, no ha surgido de un año para otro, ha sido la construcción constante de arduo trabajo de investigaciones de grandes científicos, que desde el Siglo XVII hasta el actual Siglo XXI han dedicado su vida hacia la construcción de lo que hoy en día conocemos como “Inteligencia Artificial”. Desde el primer prototipo de computadora, que fue en 1804 por Joseph Marie Jacquard, combinación de dos primeras invenciones en los años 1725, 1728 y 1740 por Basile Bouchon, Jean Baptiste Falcon y Jacques Vaucanson respectivamente. Algunos creen que la primera invención de una computadora fue cuando Charles Babbage “padre de la computadora” en 1822, desarrolló su primer diseño de una computadora funcional, demostrando un motor analítico. No fue desarrollada físicamente por él, pero sí tuvo ayuda de otra gran persona, Ada Lovelace quien se dice ha creado el primer algoritmo. Después, el “padre de la computación” Alan Mathison Turing a inicios del siglo XX, quien llegó a revolucionar las ciencias de la computación, aportando la “Máquina de Turing”, también diseñó la “bomba de Turing”, que sirvió para romper con un código Enigma durante la Segunda Guerra Mundial, salvando miles de vidas, no por menos importante la “prueba de Turing”, para medir el intelecto de una máquina. Después de su probable suidio en el año 1954, no fue hasta el año 2019, que la Reina Elizabeth II, reconoció a Turing como la figura más importante del siglo XX.

Un libro publicado entre los años 1909-1913 por Bertrand Russel y Alfred North Whitehead, llamado Principios matemáticos, donde se muestra que todos los elementos matemáticos podrían ser resueltos con razonamiento mecánico a través del uso de lógica formal. Posteriormente, en 1914 Leonardo Torres Quevedo, descrito después como “El primer pionero de la inteligencia artificial”, introdujo la aritmética punto-flotante (de suma importancia en la Inteligencia Artificial y su optimización). En 1935, Alonzo Church introdujo el “cálculo lambda”, fundamental para la teoría de lenguajes de computación. En 1943, las primeras redes neuronales artificiales fueron descritas por Warren Sturgis McCulloch y Walter Pitts. El primer programa de Inteligencia Artificial, probablemente fue en 1951, jugando “checkers” con “ajedrez”. En 1955, ya se había escrito un programa que podía mejorar por sí mismo, jugando “checkers”. También, en 1956, se probaron 38 de 52 teoremas en el libro de los “principios matemáticos” por Rusell y Whitehead, donde genuinamente, se realizó ingeniería, dando así nacimiento a la inteligencia artificial. En 1965, el primer algoritmo para aprendizaje automático profundo (DNN) para percepciones multi-capa, fue desarrollado por Alexey Gringorevevich Ivakhnenko y Valintin Lapo en Ucrania. En 1980, Kunihiko Fukushima, conocido como “quien le dió la vista a las computadoras”, siguiendo el trabajo de Hubel y Wiesel, publicaron la neurocognición, que fue la original profunda “red convolucional neuronal” (arquitectura CNN). A su vez, en 1969, Fukushima introdujo el ReLU (Rectifying Linear Unit), que es una función de activación en el contexto de extracción de características en redes neuronales jerarquizadas. Hasta el año 2017, ReLU fue la función de activación más utilizada en redes neuronales profundas. Después, en los 1980s, los “padrinos de la inteligencia artificial”, Yann André LeCun y sus colegas, Yoshua Bengie y Geoffrey Hinton, introdujeron el concepto de “redes neuronales convolutivas”. [2]

Aunado a lo anterior, en el año 2024, John J. Hopfield y Geoffrey E. Hinton fueron galardonados con el Premio Nobel de Física por su trabajo en redes neuronales artificiales, suponiendo grandes avances no únicamente en la rama de ciencias de la computación, sino en ramas como la física y la medicina.

Entrando en contexto con el presente reporte, el aprendizaje profundo es una ramificación del aprendizaje de máquina (Machine Learning), el cual básicamente son representaciones computacionales, que dada una entrada como conjunto de datos, una salida esperada y la métrica para saber que tan bien se ha realizado la salida real con la esperada. Ahora, en términos de Deep Learning, esto hace referencia a la representación de los datos, como en el caso de clasificador de imágenes, sería buscar la mejor forma de representar a los datos, su representación variaría, si una imagen es RGB o RGBA. Con el uso de Deep Learning, se puede llegar a representar de una mejor manera a los datos para que el aprendizaje sea más “ameno”. El aprendizaje en el contexto de Machine Learning se describiría como el proceso automático de búsqueda para la transformación de datos para producir representaciones utilizables, guiadas por una señal de retroalimentación. En este caso, los algoritmos de Machine Learning no son buenos para encontrar las transformaciones de representaciones de datos, pero sí para encontrar un espacio hipotético. Por lo cual, Deep Learning, hace referencia a la “profundidad” de un modelo, el número de capas sucesivas representativas. Se puede comprender que un modelo de Deep Learning como un proceso multi-capa de destilación de información, donde la información se va filtrando cada vez más y se convierte purificada en la salida. [3]

La especificación de lo que una capa hace a la información que viene de entrada se almacena en los pesos. En otros términos, la transformación implementada por una capa es parametrizada por los pesos. Así, el aprendizaje se refiere a encontrar un conjunto de valores para los pesos de todas las capas de una red, para que la red pueda encontrar de buena forma sus objetivos. En el caso de redes neuronales profundas, los parámetros pueden llegar a ser billones. Como control/medida sobre el como se espera que has “atinado” o no a un objetivo, se usan las funciones de pérdida o funciones objetivo o funciones de costo, midiendo así que tan cerca o alejado estás del objetivo. Dicha función, toma las predicciones de la red y el objetivo real (lo que se espera de salida) y se calcula la distancia como puntaje (mide la calidad de la salida de la red). Como es de saberse, los algoritmos de Deep Learning utilizan Backpropagation, por lo cual, se utiliza dicho puntaje como señal de mejoramiento y ajuste de los pesos cada vez, encaminado hacia una dirección con una salida de mayor calidad. Dicho ajustador es llamado el optimizador. Se utilizan funciones de propagación de gradiente para el ajuste de los pesos y encontrar los mínimos en un plano, acercándose más a un objetivo (mínimo local). Completando así el ciclo de entrenamiento una vez que se repite el proceso descrito hasta el optimizador miles de veces. Una red con una pérdida mínima es aquella en que sus salidas han sido lo más cercanas posibles al objetivo (fitting). [3]

Lo que hace Deep Learning especial, es su forma automática de representar la información, lo que de manera manual sería la ingeniería de características. Otra característica, es que es supervisado,

puesto que un cambio en una señal se propaga por la red, funcionando así de manera conjunta. [3]

“Si la nueva revolución industrial es la Inteligencia Artificial, si el Deep Learning es el vapor del motor de esta revolución, entonces los datos son su carbón”. [3]
Dado esto, es que se usan modelos pre-entrenados que utilizan grandes cantidades de datos. En el caso del repositorio de ImageNet, que cuenta con 1.4 millones de imágenes que han sido categorizadas en 1,000 categorías, que como en esta investigación, se tomó el modelo pre entrenado VGG16 alimentado de datos de ImageNet. [3] [4] [5]

Uno de los primeros problemas con las primeras redes profundas y sus algoritmos de optimización empleados, fue que a medida que una arquitectura contaba con más capas, se difuminaba el cálculo de la gradiente, y por ende la pérdida iba siendo mayor. Es por esto que se han desarrollado mejores optimizaciones a este problema, lidiando así con el clásico problema de overfitting. Dentro de estas soluciones, está el desarrollo de mejores funciones de activación y el uso de mejores esquemas de optimización como RMSProp y Adam. [3]
Hacia los años 2014-2016, se desarrollaron maneras más avanzadas de lidiar con el problema de la gradiente desvaneciente, como el uso de capas “Batch Normalization”, “Residual Connections” y “Convoluciones separables en profundidad”.

Otra problemática, es que, todo proceso requiere de recursos, si bien sabemos que el principal recurso de Machine Learning son los datos, el segundo recurso importante es el consumo de energía, el tercero el consumo de recursos computacionales requeridos para computarizar los distintos procesos y arquitecturas de inteligencia artificial. Es por esto, que hay restricciones con las arquitecturas de cada computadora, para correr ciertas arquitecturas. En la actualidad, se utilizan procesadores de tipo GPU o TPU para correr arquitecturas de Inteligencia Artificial, procesadores especializados en procesamiento acelerado de gráficos y operaciones de tensores, respectivamente. Las operaciones de tensores son necesarias, puesto que, un modelo de Redes Profundas de Aprendizaje (Deep Learning), como se comentó anteriormente, generan un hiper-espacio, imposible de imaginar como humano, con K dimensiones, donde K es representada por el número de “variables” de la ecuación, o en otras palabras, representada por el número de características que contiene una función de entrada. Dado esto, matemáticamente, se utilizan los tensores como representación y operaciones de los mismos espacios.

Metodología

Se ha empleado un proceso clásico para la solución de problemas clasificatorios de imágenes con el uso de inteligencia artificial. Proceso experimental que consistió en los siguientes pasos: Recolección de datos y verificación de los mismos, recolección de más datos y verificación de los mismos en distintos sitios (leer README y ver sitios), unión y estandarización de datos en batch con scripts bash de manera local, pre-procesamiento de datos, partición de datos, pre-procesado de datos, incremento de datos en disco, extracción de aproximación de características con modelo pre-entrenado VGG16, construcción experimental de arquitectura convolucional (CNN), análisis e interpretación de resultados del modelo,

finalmente una iteración desde la construcción experimental de arquitecturas y otra desde aumento de dataset manual.

Recolección de Datos y Verificación I

Durante esta etapa, se realizó primeramente la búsqueda del dataset, dentro de Kaggle. Dicho repositorio de Kaggle, contaba ya con el Dataset clasificado con las clases mencionadas anteriormente. En este caso, se hizo una verificación de manera manual, revisando que la separación fuese correcta. La cantidad de datos dentro de este sitio fue el mencionado en la introducción. La estandarización de tamaño de imágenes de este Dataset son de 524 x 524 (píxeles). Un total de 4752 imágenes, lo cual no son suficientes, ya que si fuera una equilibrada división de los datos, serían 528 imágenes por clase.

Recolección de Datos y Verificación II

Este paso, fue realizado posteriormente a realizar una iteración del modelo con únicamente los datos brindados por el repositorio del paso anterior. En este caso, se buscaron imágenes y fueron clasificadas de manera manual en cada una de las clases. El tamaño final de las clases de imágenes fue: Vegetation con 980 imágenes, Textile Trash con 756 imágenes, Plastic con 1,404 imágenes, Paper con 1,095 imágenes, Metal con 1,201 imágenes, Glass con 922 imágenes, Food Organics con 912 imágenes, Cardboard con 865 imágenes y Miscellaneous Trash con 677 imágenes. Dando un total de 8,812 imágenes.

Unión y Estandarización de Datos en Batch

Se utilizaron dos scripts bash, con el objetivo de generar nombres de imágenes de manera estandarizada. El nombre de la clase perteneciente a la imágen con un índice i junto con la extensión .jpg. Se unieron todas las imágenes pertenecientes a su clase como directorio.

Partición de Datos (Data Splitting)

La partición seleccionada fue de 70%, 15% y 15% de datos de entrenamiento, validación y de pruebas. Realmente no hay ciencia cierta de cuál es la cantidad de datos que hay que seleccionar, como regla, el mayor porcentaje de datos son seleccionados para la categoría de entrenamiento, las otras dos categorías se dividen equitativamente. Generalmente se suelen tomar particiones de 70%, 15% y 15% o 80% 10% 10%, dependiendo de la cantidad de datos disponible.

Preprocesado de Datos

Dada la ardua investigación en 2 reportes de investigación, como para detección de objetos y mejorar su reconocimiento. Dado que los datos del proyecto son en su mayoría un solo objeto, es de gran ventaja e importancia hacer el esfuerzo por realizar el proceso de pre-procesado. Se han encontrado diferentes tipos de ruido que pueden ser utilizados. Ruido amplificador (Ruido Gaussiano). Forma idealizada del ruido blanco causado por fluctuaciones de señal. Puede haber más ruido en el canal azul que en el rojo y verde. Ruido impulsivo (Ruido de sal y pimienta): Cuando hay "puntos" que pueden ser "pixeles muertos" encontrados en la imagen. Ruido de tiro. Distribución Poisson, no tan diferente a la Gaussiana. Se identifica cuando el ruido es causado por los puntos con mayor luz "photon shot noise" y en áreas oscuras "dark shot noise" o "dark-current shot noise". Ruido de cuantización (ruido uniforme) Causado al cuantificar los píxeles de una imagen

percibida hacia un número discreto de niveles. Film grain: el grano de una "película" fotográfica es una señal de ruido dependiente, relacionada con el ruido de tiro. Ruido on-isotrópico. Ruido de mancha (Ruido Multiplicativo). Ruido granular, que inherentemente existe dentro y degrada la calidad del radar activo y la apertura sintética de las de imágenes (SAR). Puede ser modelado con valores aleatorios multiplicados por cada valor de píxel, por eso el nombre. Ruido periódico. [5][6]

Además, segmentación, la tarea de suma importancia para un problema de visión de computación, que consiste en particionar la imagen en múltiples segmentos o regiones basadas en ciertas características como su color, intensidad, textura o moción.
La meta de la segmentación es simplificar la representación de la imagen al agrupar los píxeles y regiones que comparten propiedades similares, haciendo más eficiente y sencillo la extracción de "features" de una imagen (datos importantes
de una imagen). Al aplicar esta técnica, se conseguirán regiones segmentadas que habilitan la eficiencia en el análisis e interpretación de los datos de una imagen. Puntos que pueden ser la iluminación de la imagen, en veces llamado bordes
de la imagen. Técnica ampliamente utilizada para procesado de imágenes digitales, modelo de reconocimiento, morfológico de imágenes, extracción de recursos [6].
En este caso, se realizó una combinación de los procedimientos encontrados en cada uno de los papeles de investigación [5]. Se utilizó la librería de cv2, que aporta funciones ya establecidas [5]. Las funciones implementadas con el fin de incrementar la precisión de reconocimiento de objetos fueron: conversión a escala de gris, aplicación de ruido Gaussiano [6], aplicación del algoritmo de visión de computación de detección de bordes Canny [7]. Aplicación de dilatación, incrementando los bordes después de haber detectado los bordes con el algoritmo de Canny. Se utilizó erosión, decrementando el tamaño de los bordes retornados por el algoritmo de Canny. Se utilizó la función de encontrar bordes o “contours”. Finalmente, al aplicar todas estas funciones, se obtuvo el máximo borde, se obtuvieron las coordenadas del rectángulo dibujado por el reconocimiento del objeto (si es que se identificó), se “dibujó” un rectángulo con las coordenadas encontradas en un cuadro de 255 píxeles, para finalmente, volver a la imagen a un tamaño de 224 píxeles (el default como entrada para el modelo pre-entrenado VGG16). [5][7]

Incremento de Datos en Disco (Data Augmentation)

Como se mencionó anteriormente, el carbón de la inteligencia artificial son los datos, más que un problema clasificatorio de imágenes. Otro de los beneficios de realizar este procedimiento es que decrementa la posibilidad de overfitting [3][4][5][9]. Dicho esto, se ha realizado un proceso de incremento de imágenes en disco (Data Augmentation) [3][5][8]. El aumento de datos depende de la arquitectura que se tiene planeada como modelo de inteligencia artificial. Puede haber un aumento de datos en tiempo de ejecución/entrenamiento o antes, también llamado aumento de datos en disco. Hay implicaciones para la toma de decisiones en cuanto al tipo de aumento de datos seleccionado. La mayor implicación es el uso de recurso de cómputo, que generalmente, para realizar un aumento de datos en momento de ejecución (dependiendo del problema) pero para

procesamiento de imágenes requiere de un procesador de GPU, en lugar de un CPU. Dadas las implicaciones del proyecto y las condiciones de las mismas, se optó por un aumento de datos en disco.

El framework de keras construido encima de Tensorflow, la clase de generación de instancias de nuevas imágenes "ImageDataGenerator". Dicha función es bastante configurable, permite normalización, rotación en cierta cantidad de grados, reducción del ancho de la imágen, reducción del alto de la imágen, aplicación de zoom entre otras. En este caso, y meramente experimental (se podrían realizar varias experimentaciones con base en diferentes configuraciones, el proceso experimental es costoso, sin embargo puede haber mejoras en este sentido) se ha optado por la siguiente configuración: normalizar las imágenes del rango [0, 1] en lugar de [0, 255], rotación en un rango máximo de 20% en ángulo, cambio del ancho y alto de la imágen en un rango máximo de 20%, alargamiento o "shear" de la imágen en máximo 20%, cantidad de zoom aplicado a la imágen en máximo un 20% y rotación espejo de manera horizontal.

Con el fin de encontrar un número de clases balanceadas, se ha realizado el aumento de imágenes en aquellas clases que contaran con menos de 1,000 imágenes después de haber realizado la separación de los datos [3]. Siguiendo el proceso, en aquellas categorías donde su cantidad de imágenes en la partición de entrenamiento fueron menor a 1,000, se añadieron dividió entre 5 la cantidad total de imágenes, y después, de manera aleatoria se tomó un rango de las imágenes disponibles con máximo número de la diferencia entre 1,000 y la longitud de las imágenes dividido entre 5 y se aplicó la función de aumento de datos antes mencionada a este rango de imágenes, en un rango de 5, es decir, se incrementaron en 5 las imágenes por cada rango de la diferencia entre 1,000 y la longitud de las imágenes dividido entre 5. Finalmente, en disco se guardaron las imágenes pertinentes. De esta manera, se "equilibraron" las clases (únicamente en la parte de entrenamiento). En las particiones de validación y de prueba, no es buena práctica aplicar el proceso descrito, puesto que se estaría distorsionando el resultado de una función o ecuación, y por tanto, se podría generalizar erróneamente el modelo. Otra idea de mejora y que, por incremento en el costo, es realizar experimentaciones con un mayor balanceo en el número de imágenes por clase, en este caso, el proceso a realizar sería de "decremento de imágenes" puesto que la categoría de "Miscellaneous Trash" cuenta con pocos datos. Sería balancear a todas las categorías con base a la clase que cuenta con el menor número de datos. Sin embargo, dicho procedimiento, estaría dentro del proceso de "partición de datos", siendo un procedimiento costoso.

Extracción de Aproximación de Características (Feature Extraction)

La extracción de características es de suma importancia, esa es la razón por la cual se utilizan los modelos pre-entrenados, puesto que a través de una entrada (las imágenes en un espacio tensorial) de las siguientes dimensiones (None, 224, 244, 3) el primer parámetro indica el número de "batch", el segundo y tercero los píxeles de la imágen (alto y ancho) y el tercer parámetro indica los canales que tiene la imágen (RGB en este caso). Se utiliza el pre entrenado para que resulte en un tensor de menor espacio y con las adecuaciones pertinentes para llegar a un mejor resultado con mejor

precisión con base a la predicción en cuestión. Dicho esto, un modelo pre-entrenado toma tus datos, y con base a las arquitecturas estudiadas y con un amplio tamaño de datos disponibles, generan lo que se llama “feature extraction”, de esta manera, ahorras costos y solamente te ocupas de mejorar la precisión realizando con “hypertuning” [3].

Se pueden emplear dos técnicas dentro de este rubro, uno es el “feature extraction” antes de realizar el entrenamiento como tal, el otro consiste en realizar el “feature extraction” al momento de entrenar el modelo y con aumento de datos. La última opción solo es viable en computadores con procesadores GPU [3], en resultados veremos cómo es que se pudo mejorar el modelo con CPU comparado con GPU.
Se puede obtener una mejor precisión de validación y de pruebas (mejora de modelo) mediante el congelamiento de algunas capas para realizar “tuning” de manera atrasada. Las buenas prácticas suelen congelar las primeras capas de la arquitectura, mientras más cerca con respecto a la capa densa final, se aplica “tuning”, generando una mejor configuración de la arquitectura [3].

Construcción Experimental de Arquitectura Convolucional (CNN)

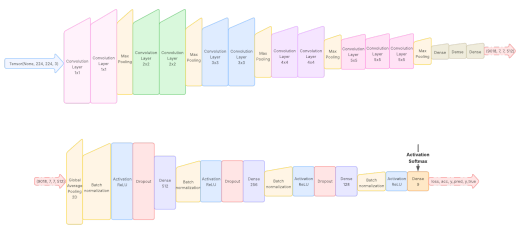


Figura 1 Arquitectura Final

Del Modelo de Reconocimiento de Imágenes

La arquitectura seguida, fue una combinación del uso del modelo pre-entrando VGG16 junto con un modelo manualmente construido con base en buenas prácticas. Como se comentó anteriormente, el modelo pre-entrenado VGG16 se utilizó para la extracción de “features” o de variables que, realizando una mejor aproximación a una mejor predicción. Del input inicial (None, 224, 244, 3), donde la primera posición indica el número de batch, el segundo y tercer elemento significan los píxeles de la imágen, y el tercer elemento los canales de la imágen (color RGB). Entrenar un modelo manualmente con una red de tipo CNN, requiere de demasiados recursos, puesto que se estarían realizando un Flatten o GlobalAvergaPooling, para realizar una red con capas densas en su intermedio (una arquitectura de CNN “Scratch”). En el caso de la arquitectura VGG16, se realizan capas convolucionales (generalmente son 5 conjuntos de capas convolucionales) el número de parámetros puede alcanzar los 14,000,000 fácilmente, lo cual es muy ineficiente y poco probable realizar un entrenamiento simulando esas capas.

Con “feature extraction” incluido en Keras, utiliza el dataset de imágenes de ImageNet, donde personas, se han dedicado a realizar las apropiadas configuraciones de hiper parámetros como: capas convolucionales, ratio de aprendizaje, filtros, tamaños de kernel, padding en capas de MaxPooling, así como el número de neuronas de la última capa densa que suele caracterizar a un modelo VGG16.

En este caso, se intentó realizar una prueba con una arquitectura simulando al

modelo pre-entrenado VGG16 en Keras, siguiendo la siguiente figura:

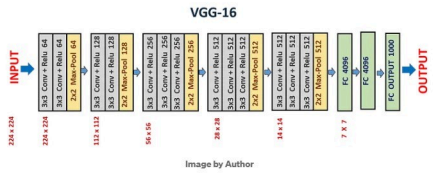


Figura 2 Arquitectura de modelo pre-entrenado VGG16

Con el recurso de cómputo actual, se realizó una iteración de entrenamiento, el proceso no se pudo completar tras una noche (además de que el consumo de luz es demasiado alto). Dadas estas condiciones, se tomó la decisión de seguir los procesos de las referencias utilizadas, donde utilizan el modelo VGG16 como feature extractor.

Dado esto, se entrenó por dos fases el primer input descrito (None, 224, 224, 3), y que por medio del proceso descrito anteriormente, el modelo pre-entrenado de salida genera inputs de forma: (9018, 7, 7, 512), donde el primer elemento simboliza la cantidad de datos que conforma mi dataset, el segundo (altura), tercer(ancho) y cuarto elemento (canales) simbolizan los features que han sido extraídos por el modelo, y que, realmente podrían no tener un significado para humanos, más que variables que mejoran la precisión de predicciones. Después de esto, se hizo un diccionario de varias arquitecturas con distintas configuraciones de hiper-parámetros, obteniendo resultados como gráficas y DataFrames donde se muestran las métricas más relevantes para un análisis de resultados profundo.

La arquitectura propuesta de la Figura 1 (sub-arquitectura parte de abajo), donde se muestran cada una de las capas utilizadas. Primero, el input descrito del párrafo anterior, input de forma (9018,7,7,512), después una capa llamada GlobalAveragePooling2D, encargada de realizar una baja de algunos de los parámetros “downsampling” al hacer un cómputo de la media de la altura y el ancho de las dimensiones del input. Esto con el fin de reducir la cantidad de parámetros y como tal, los recursos utilizados. Después, secuencias de capas similares, donde se pusieron 3 bloques similares en secuencia: BatchNormalization, ActivationReLU, Dropout, Dense. [3] En general, el objetivo de una buena arquitectura sería: “conseguir las mejores métricas de precisión en cálculos que demore lo menos posible”. Las secuencias de dicha arquitectura han sido entonces realizadas con dicho fin, además de reducir el overfitting mediante técnicas de regularización comunes. Tomando en cuenta el objetivo pasado, una capa de BatchNormalization no mejora las métricas, sin embargo, reduce el tiempo de ejecución del modelo, no está verificado y validado que tenga un efecto positivo en cuanto a disminuir overfitting. Es una forma de normalización, donde durante el entrenamiento, utiliza la media y la varianza del batch actual de datos para normalizar sus muestras, y durante la inferencia (cuando no hay un batch suficiente para la representar los datos), utiliza movimientos de promedio exponenciales de la media y varianza del batch de los datos vistos en entrenamiento [3]. La siguiente capa Activation ReLU se quita a la última capa Densa, ConvNet vista, y se agrega después de la capa de BatchNormalization, esto ya que como la capa BatchNormalization centraliza todos sus entradas en cero, mientras que la activación ReLU utiliza esos ceros como pivote para mantener o soltar los canales

activados. [3] La siguiente capa, Dropout que realmente, son capas que ayudan a reducir el overfitting, donde básicamente se da un porcentaje, que simboliza la cantidad de datos que quieres que se conviertan en cero (eliminando neuronas) para la siguiente capa (generalmente se utiliza 0.5).

La última capa Densa, realmente en estas se han obtenido distintos resultados con base a las distintas arquitecturas (por el número de neuronas), y también con base en diferentes hiper-parámetros comentados anteriormente, por lo que la secuencia de capas de bloques descritos anteriormente, depende de varios factores. Las capas densas, contienen regularizadores de tipo L2, técnica que ayuda a generalizar un modelo o reducir el overfitting, ya que apacigua o aumenta las activaciones de operaciones entre capas (rango en porcentaje), generalmente se utilizan valores de 0.02.
En el caso de la Figura 1, se tomó en cuenta una arquitectura con varias secuencias de los bloques descritos anteriormente. Y que, realmente no hubo demasiado cambio entre esa arquitectura descrita y con aumento o disminución del número de neuronas, como buena práctica el número de neuronas es en exponenciales de 2, en orden descendiente.
La última capa del modelo, siempre será Densa con 9 neuronas, que simboliza las clases de categorización del problema, y que finalmente, te regresa el historial del modelo y las métricas que ayudan a analizar los resultados).

Resultados

Dados los resultados de otros reportes de investigación, las distintas métricas han sido muy diferentes con respecto a la obtenida en este proyecto experimental de investigación. Los resultados dependen de los recursos disponibles del proyecto. En este caso se cuenta con un dispositivo Macbook Pro 2020 con procesador 2.3 GHz Quad-Core Intel Core i7.
De primera instancia, se realizó un modelo inicial, el cual no ha generado resultado alguno, puesto que el poder de cómputo no fue suficiente para correr el modelo. Se utilizó la arquitectura de la figura 2 de manera manual.

Como segunda iteración de modelos, se realizó la construcción y configuración de modelos con distintos hiper parámetros, no sin antes, haber realizado el proceso de feature extraction, con el modelo pre-entrenado VGG16, no se contempló un aumento de imágenes con la función de DataAugmentator, dicha precisión de validación alcanzó en un máximo 76% así como un máximo de precisión de entrenamiento de 86%. Se realizaron varias iteraciones de modelos con distintos hiper parámetros, sin embargo, no fue posible incrementar las precisiones dichas anteriormente. Únicamente se han tomado algunos de los resultados obtenidos, no de todos los resultados (el de mejor precisión). Figura 3 y Figura 4.

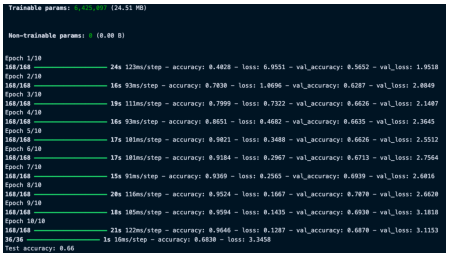


Figura 3
Resultados de modelo segunda iteración

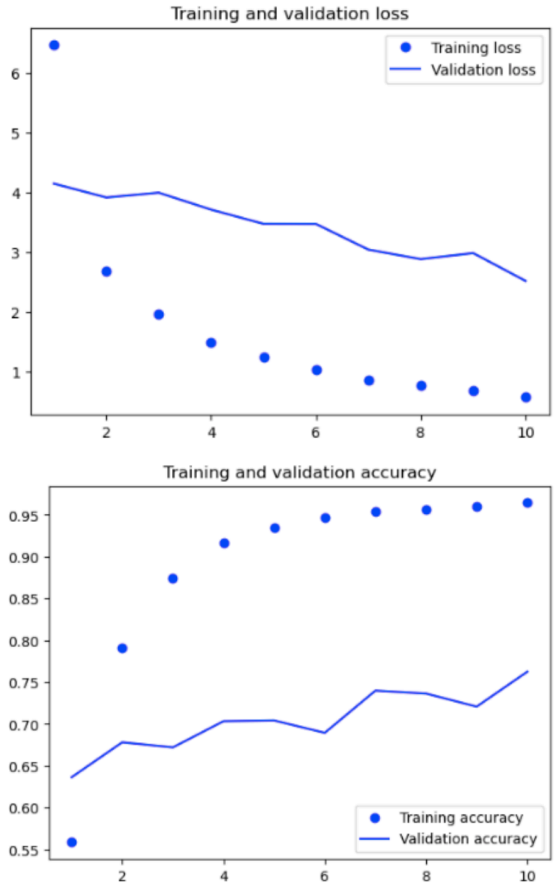


Figura 4 y 5
Gráficos de modelo segunda iteración

Se decidió realizar más iteraciones, puesto que claramente, en la gráfica de abajo, hay un “underfitting”, problema que puede ser solucionado de varias maneras, como por ejemplo aumentando el número de capas ocultas. Se realizaron pruebas y experimentaciones con distintos hiper parámetros, sin embargo, se llegó a la conclusión de que era necesario aumentar el número de datos, la razón por esto, es que el modelo no ha alcanzado un overfitting (la línea de validation accuracy está por debajo de training accuracy) [3].

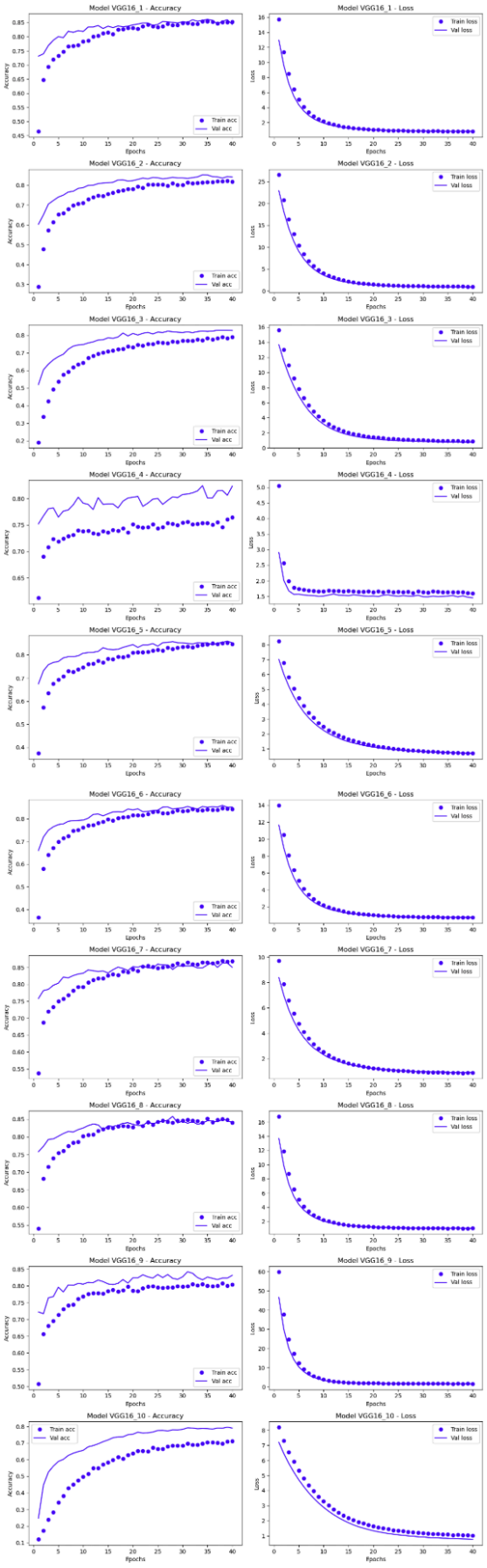
Tercera iteración, de igual manera, se utilizó el proceso de extracción de features con el modelo pre-entrenado VGG16, posteriormente, de manera iterativa, cambiando hiper parámetros: número de capas densas, neuronas, tasa de aprendizaje, porcentajes de regularizadores L2, porcentaje de las capas de Dropout, épocas [3]. En esta ocasión, los resultados fueron los siguientes:

Model Performance Summary

	Model	Train accuracy	Validation accuracy	Test accuracy	Loss	Learning rate	Epochs
0	VGG16_1	0.85	0.84	0.84	0.85	0.000100	40
1	VGG16_2	0.82	0.84	0.84	0.92	0.000100	40
2	VGG16_3	0.79	0.83	0.83	0.76	0.000100	40
3	VGG16_4	0.76	0.82	0.80	1.45	0.001000	40
4	VGG16_5	0.85	0.85	0.85	0.70	0.000100	40
5	VGG16_6	0.84	0.85	0.84	0.74	0.000100	40
6	VGG16_7	0.87	0.85	0.84	0.97	0.000100	40
7	VGG16_8	0.84	0.84	0.85	1.03	0.000100	40
8	VGG16_9	0.80	0.83	0.82	1.63	0.000100	40
9	VGG16_10	0.71	0.79	0.79	0.79	0.000100	40
10	VGG16_11	0.77	0.81	0.80	0.69	0.000100	40
11	VGG16_12	0.88	0.84	0.83	0.55	0.000100	40
12	VGG16_13	0.92	0.85	0.85	0.55	0.000100	40
13	VGG16_14	0.88	0.85	0.84	0.93	0.000300	50

Figura 6
DataFrame de rendimiento de modelos

Se tomó el mejor modelo VGG16_13, donde se tiene un mayor porcentaje de precisión de entrenamiento, una precisión de validación igual a otras dos arquitecturas, la precisión de prueba es la más alta y la pérdida es la menor. A continuación las figuras que han sido de apoyo para poder realizar distintas configuraciones, según underfitting o overfitting.



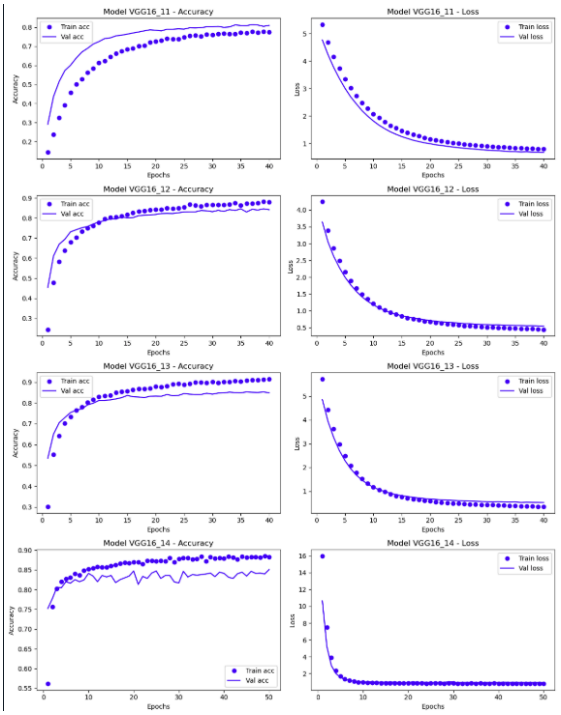


Figura 7, 8 y 9
Gráficos de arquitecturas

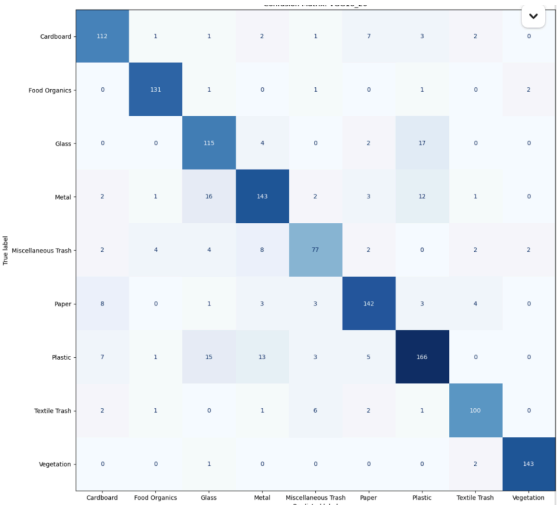


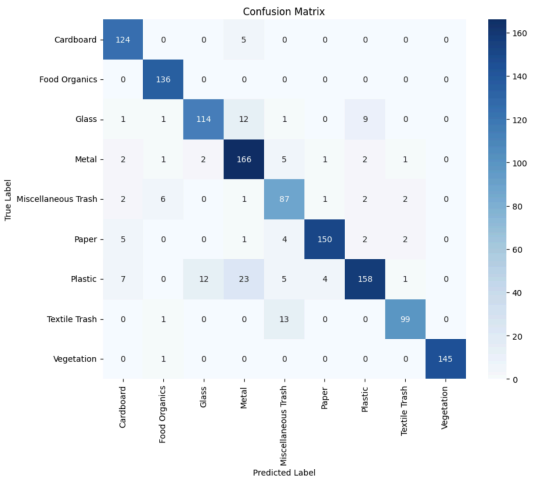
Figura 10
Mapa de Calor de VGG16_13

Claramente, el modelo ya no tenía tantos falsos positivos la diagonal superior izquierda a inferior derecha los colores son azul no tan claro, por lo que, las predicciones realizadas han sido en su mayoría verdaderas.

La última iteración, fue con el seguimiento de un proceso de fine-tuning, el cual

utiliza feature extraction, pero además consiste en descongelar una cierta cantidad de las capas superiores, y junto con el entrenamiento de ambas partes añadidas del modelo y las capas superiores. Se le llama fine-tuning porque se ajustan de manera abstracta las representaciones del modelo que están siendo re-usadas para hacerlos más relevantes para el problema. Solamente es posible hacer fine-tune en las capas superiores de la base convolucional una vez que el clasificador en la parte superior ha sido entrenado [3]. Además un proceso de aumento de datos “on the fly”.

Se siguió el proceso descrito en el libro de Deep Learning, dando como resultado el mejor modelo de esta investigación experimental. Distintas arquitecturas e iteraciones no fueron posibles, debido a que el proceso requiere del uso de GPU y el recurso es limitado en Google Collaboratory.




```
# Evaluate the model after unfreeze
# 8. Evaluate the model with new
loss, accuracy = model_vgg16.evaluate(test_dataset)
print(f"Test accuracy: {accuracy:.2f}")

best_val_acc = max(history_vgg16.history['val_accuracy'])
print(f"Best validation accuracy (initial training): {best_val_acc * 100:.2f}%")

best_val_acc_ft = max(history_finetune.history['val_accuracy'])
print(f"Best validation accuracy (fine-tuning): {best_val_acc_ft * 100:.2f}%")
```

```
42/42 ━━━━━━━━━━━━━━━━━━━━ 6s 131ms/step - accuracy: 0.8995 - loss: 0.4677
Test accuracy: 0.90
Best validation accuracy (initial training): 84.86%
Best validation accuracy (fine-tuning): 91.64%
```

```
# Evaluate the model after unfreeze
# 8. Evaluate the model with new
loss, accuracy = model_vgg16.evaluate(test_dataset)
print(f"Test accuracy: {accuracy:.2f}")
```

```
42/42 ━━━━━━━━━━━━━━━━━━━━ 6s 132ms/step - accuracy: 0.8984 - loss: 0.4812
Test accuracy: 0.90
```

```
282/282 ━━━━━━━━━━━━━━━━━━━━ 52s 183ms/step - accuracy: 0.8552 - loss: 0.5804 - val_accuracy: 0.8426 - val_loss: 0.6093
Epoch 39/40
282/282 ━━━━━━━━━━━━━━━━━━━━ 77s 165ms/step - accuracy: 0.8611 - loss: 0.5651 - val_accuracy: 0.8411 - val_loss: 0.6070
Epoch 40/40
282/282 ━━━━━━━━━━━━━━━━━━━━ 82s 166ms/step - accuracy: 0.8598 - loss: 0.5513 - val_accuracy: 0.8464 - val_loss: 0.5893
42/42 ━━━━━━━━━━━━━━━━━━━━ 489s 12s/step - accuracy: 0.8319 - loss: 0.6296
Test accuracy: 0.85
```

```
Classification Report:
                    precision    recall  f1-score   support

         Cardboard       0.88      0.96      0.92       129
     Food Organics       0.93      1.00      0.96       136
             Glass       0.89      0.83      0.86       138
             Metal       0.80      0.92      0.86       180
Miscellaneous Trash       0.76      0.86      0.81       101
             Paper       0.96      0.91      0.94       164
           Plastic       0.91      0.75      0.83       210
      Textile Trash       0.94      0.88      0.91       113
         Vegetation       1.00      0.99      1.00       146

          accuracy                           0.90      1317
         macro avg       0.90      0.90      0.90      1317
      weighted avg       0.90      0.90      0.89      1317
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92       129
           1       0.93      1.00      0.96       136
           2       0.89      0.83      0.86       138
           3       0.80      0.92      0.86       180
           4       0.76      0.86      0.81       101
           5       0.96      0.91      0.94       164
           6       0.91      0.75      0.83       210
           7       0.94      0.88      0.91       113
           8       1.00      0.99      1.00       146

    accuracy                           0.90      1317
   macro avg       0.90      0.90      0.90      1317
weighted avg       0.90      0.90      0.89      1317
```

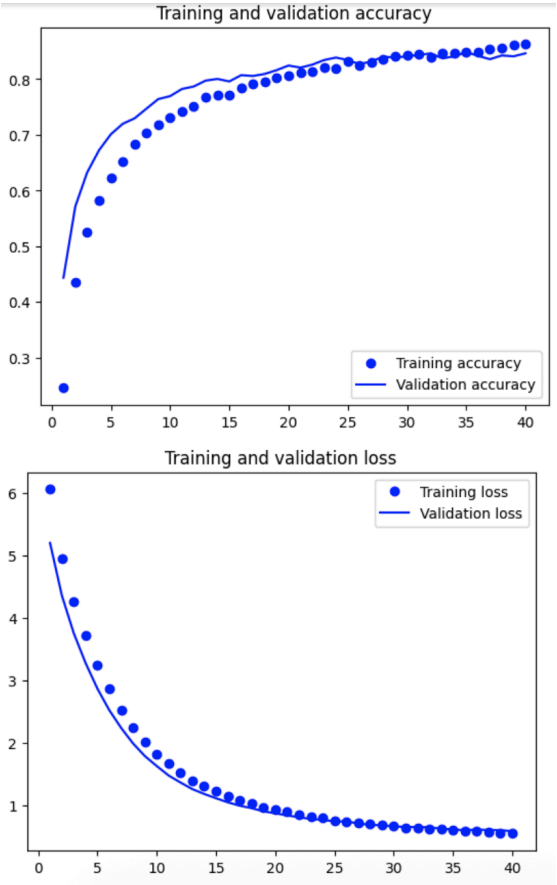


Figura 11-18
Resultados modelo con Fine-Tuning

En este último modelo, se ha alcanzado una precisión de entrenamiento de 90%, precisión de validación de 91.64% y una precisión de entrenamiento de 90%.
Además, obtuvimos un 90% de precisión en todas las métricas que son arrojadas por el reporte de clasificación: puntaje F1, precisión general y recall.

Conclusión

El uso de modelos pre-entrenados en la actualidad es de suma importancia para lograr un buen desempeño de los mismos y por ende un buen producto de inteligencia artificial. Los modelos de las dos últimas iteraciones fueron los mejores en cuanto a métricas, han superado la precisión del modelo propuesto en la referencia [3] con un porcentaje de precisión de 75.6%. Se ha cumplido el objetivo de sobrepasar el porcentaje de dicho modelo. El modelo, aunque suene que es bastante bueno, tiene áreas de mejora, como en el preprocesado de los datos, además en el aumento de datos, siempre manteniendo un balance en el número de clases. Además de esto, sería interesante utilizar herramientas como Keras Tuner o incluso algún modelo pre-entrenado que brinde una mejor precisión como lo son ResNet50, EfficientNet, Vision Transformers o incluso VGG19.

Referencias

[1] “UCI Machine Learning Repository,” Uci.edu, 2023. https://archive.ics.uci.edu/dataset/908/realwaste (accessed May 22, 2025).

[2] A. Grzybowski, K. Pawlikowska – Łagód, and W. C. Lambert, “A History of Artificial Intelligence,” Clinics in Dermatology, vol. 42, no. 3, Jan. 2024, doi: https://doi.org/10.1016/j.clindermatol.2023.12.016.

[3] F. Chollet, “Deep Learning with Python, Second Edition,” O’Reilly Online Learning, Dec. 07, 2021. https://learning.oreilly.com/library/view/deep-learning-with/9781617296864/Text/08.htm#heading_id_14 (accessed May 15, 2025).

[4] T. Kaur and T. K. Gandhi, “Automated Brain Image Classification Based on VGG-16 and Transfer Learning,” 2019 International Conference on Information Technology (ICIT), Dec. 2019, doi: https://doi.org/10.1109/icit48102.2019.00023.

[5] H. Wang, “Garbage Recognition and Classification System Based on

Convolutional Neural Network VGG16," 2020 3rd International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE), Apr. 2020, doi: https://doi.org/10.1109/aemcse50948.2020.00061.

[6] A. Hambal, Z. Pei, and F. Libent, "Image Noise Reduction and Filtering Techniques," International Journal of Science and Research (IJSR) ISSN, vol. 6, no. 3, pp. 2319–7064, Mar. 2017, doi: https://doi.org/10.21275/25031706.

[7] A. Demelash and E. Derb, "Habesha Cultural Cloth Classification Using Deep Learning," Scientific Reports, vol. 15, no. 1, Apr. 2025, doi: https://doi.org/10.1038/s41598-025-98269-5.

[8] S. Tammina, "Transfer learning using VGG-16 with Deep Convolutional Neural Network for Classifying Images," International Journal of Scientific and Research Publications (IJSRP), vol. 9, no. 10, p. p9420, Oct. 2019, doi: https://doi.org/10.29322/ijsrp.9.10.2019.p9420.

[9] A. Patil, A. Tatke, N. Vachhani, M. Patil, and P. Gulhane, "Garbage Classifying Application Using Deep Learning Techniques," IEEE Xplore (Scopus), Aug. 01, 2021. https://ieeexplore.ieee.org/document/9573599 (accessed Oct. 20, 2022).