FACULTAD DE ESTUDIOS ESTADÍSTICOS

MÁSTER EN MINERÍA DE DATOS E INTELIGENCIA DE NEGOCIOS

Curso 2018/2019

Trabajo de Fin de Máster

Título: Aplicación de Deep Learning para problemas de clasificación de imágenes

***Alumno*: Agustín Marín Castejón**

***Tutor*: Juana María Alonso Revenga**

Septiembre de 2019

UNIVERSIDAD COMPLUTENSE
MADRID

Agradecimientos

Agradecer a todas las personas que han ayudado de forma directa o indirecta en la realización de este Trabajo Fin de Master. Concretamente me gustaría agradecerle a mi tutora el tiempo y la dedicación. A Juan y a todo el equipo de PeRTICA por poner a mi disposición todos los recursos para poder llevar a cabo el trabajo. Particularmente, agradecer a Víctor los reinicios de servicios que aumentaban conforme se acercaba la fecha de entrega. Por último, agradecer a familia y a amigos todo el apoyo recibido.

Índice

Índice de tablas

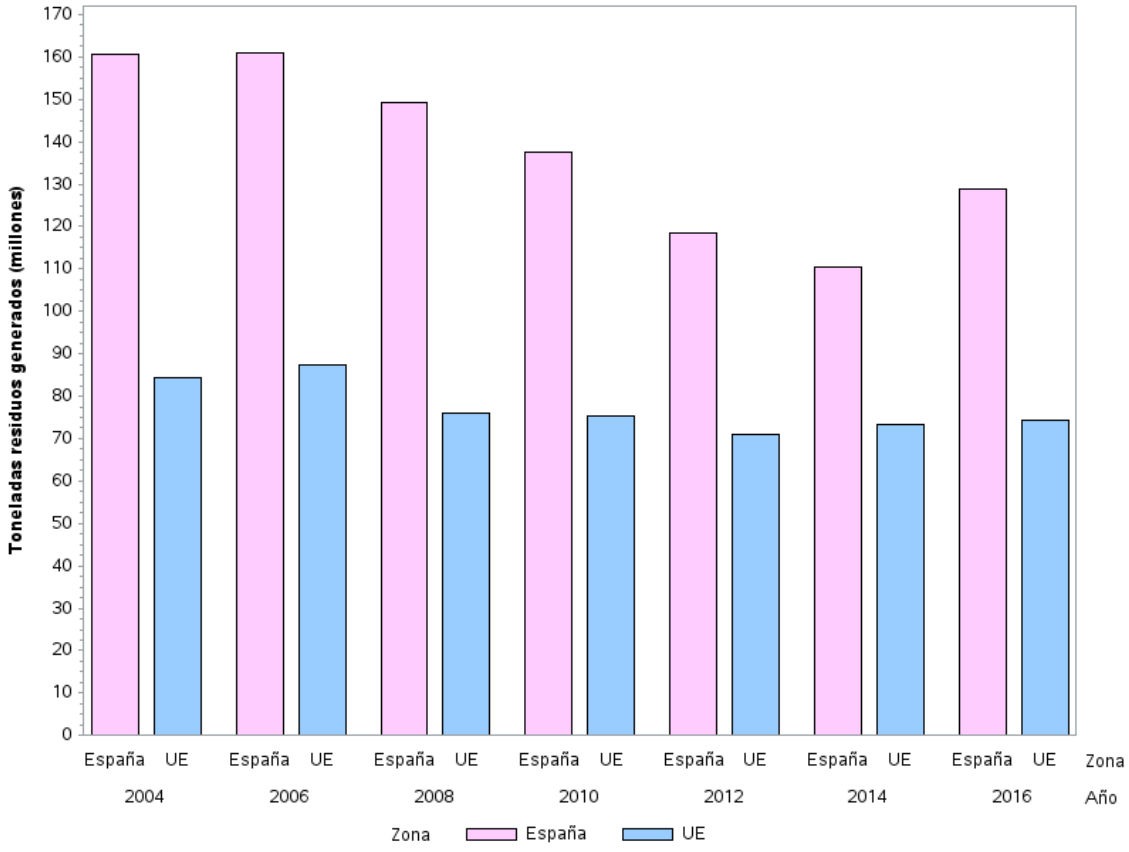
Índice de gráficos

Índice de imágenes

1 Introducción

El reciclaje según la RAE es: *"Transformar materiales de los residuos en nuevos productos, materiales o sustancias, tanto si es con la finalidad original como con cualquier otra finalidad".* El reciclaje es uno de los pilares de la ecología, es un proceso fundamental para la reutilización de materiales no biodegradables. Cada vez los ciudadanos estamos más concienciados sobre ello, es tarea de todos respetar el medio ambiente. Reducir la generación excesiva de residuos es, junto a otros, uno de los principales objetivos medioambientales del siglo XXI. El tratamiento de residuos supone una gran inversión y costes para cualquier país. En el siguiente gráfico se puede observar como ha evolucionado la generación de residuos en los últimos años. Para mayor comprensión, indicar que la barra referida a la UE recoge la media del conjunto de países que forman la Unión Europea. España genera residuos por encima de la media europea, a pesar de tener una tendencia decreciente, en el 2016 la generación de residuos se incrementó un 17% respecto al año anterior.

Gráfico 1 Residuos generados media Europa y España



Fuente: Elaboración propia a partir de datos de Eurostat https://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=env_wasgen&lang=en

El término "Las tres erres", acuñado en la cumbre del G8 2004, hace referencia a Reducir, Reutilizar y Reciclar. Los residuos son reciclables, pero con ciertos límites,

no pueden pasar por el proceso infinitas veces (Lara González, 2008). El reciclaje parece una práctica muy actual, sin embargo, numerosos historiadores han llegado a encontrar indicios de prácticas que se asemejan al actual reciclaje o reutilización en etapas paleolíticas (Amick, 2014).

El tratamiento de los residuos se realiza en plantas especializadas, se siguen una serie de procesos de forma secuencial hasta completar el ciclo. Aunque este tipo de procesos están automatizados en algunos niveles, existen tareas que se siguen realizando de forma manual, por ejemplo, la separación de determinados residuos. Surge por tanto la oportunidad de automatización de procesos con el consiguiente incremento de la productividad y reducción de costes. Por otra parte, el mismo ciudadano que deposita los residuos en los diferentes contenedores en muchas ocasiones lo hace de forma incorrecta, esto supone un verdadero problema, todos los residuos que son depositados en el contenedor equivocado pasan a formar parte de "residuos rechazados", por lo tanto, no se le podrán aplicar el proceso de reciclaje[1].

Desde el punto de vista estadístico, se pueden plantear diferentes soluciones a este tipo de problemas. Los análisis de la estadística clásica podrían servir a la hora de optimizar procesos, predicciones de cantidades de residuos, optimización de rutas. El problema que ocupa este TFM se acerca más a una cuestión de clasificación, la solución que se plantea es el reconocimiento de patrones en imágenes, para ello se utilizarán técnicas de Deep learning para buscar una solución óptima. Deep learning[2], la traducción de este término es aprendizaje profundo, consisten en algoritmos de machine learning (aprendizaje automático), se utilizan para modelar estructuras de datos en forma de vector, matriz, tensor. Machine learning, se refiere al aprendizaje automático que realiza una máquina, consiste en construir un modelo que vaya actualizando sus parámetros cada vez que se añaden nuevos datos, de esta forma el modelo se ajusta de forma automática y la máquina aprende a reconocer patrones.

Estas novedosas técnicas de análisis de datos pueden ayudar a dar solución a los problemas que se plantean en lo relativo al reciclaje de residuos. Desde el punto de vista de optimización de procesos, se puede crear una aplicación que permita la clasificación de residuos. Por otro lado, podría existir una nueva forma de reciclar, consistiría en la creación de "Smart-Containers", estos nuevos contenedores permitirán la introducción de todo tipo de residuos, internamente tendrá un sistema que aplicará un modelo de clasificación de residuos. De la misma forma, la incorporación de tecnología a los contenedores se podría aprovechar para otros fines como conocer el nivel residuos en un contenedor, optimizando así las rutas de recogida de residuos y otras ideas que puedan surgir. Todo esto hace pensar que cada vez es más real la idea que se plantea de unos años a esta parte sobre las Ciudades Inteligentes o Smart-cities, la estadística y la ciencia de datos unida a la tecnología e informática tiene un gran potencial en este ámbito.

Existen trabajos previos donde se han realizado modelos similares de clasificación de residuos, en el artículo de Yang & Thung se comparan diferentes algoritmos de clasificación de Redes Neuronales, más concretamente se utilizan arquitecturas de

[1] http://www.consumer.es/web/es/medio_ambiente/urbano/2016/10/11/224401.php
[2] https://www.sas.com/es_es/insights/analytics/machine-learning.html

Redes Neuronales Convolucionales y el algoritmo de Support Vector Machines (SVM). En este caso el algoritmo ganador es SVM obteniendo un 63% accuracy, en Redes Neuronales Convolucionales (CNN) obtienen únicamente un 22% de accuracy. Según los investigadores no era el resultado que esperaban, las Redes Neuronales Convolucionales deberían haber obtenido mejores resultados. La explicación que encuentran a esto es que SVM es un algoritmo más sencillo y las Redes Neuronales necesitan más tiempo de entrenamiento, añaden que el tamaño del conjunto de datos de entrenamiento tiene que ser mayor para obtener mejores resultados con CNN. Según Yang & Thung las CNN tienen más potencial para la extracción de patrones en las imágenes que SVM, aunque los resultados del trabajo llevado a cabo no reflejen lo mismo (Yang & Thung, 2016) .

Utilizando el mismo conjunto de imágenes, otros investigadores publicaron mejores resultados utilizando CNN, llegando a crear incluso una arquitectura de Redes Neuronales Convolucionales específica para el problema a tratar denominada 'RecycleNet'. Los resultados obtenidos por estos investigadores son realmente buenos, teniendo en cuenta que hacen uso de GPU y de máquinas con gran capacidad de procesamiento alcanzan un accuracy en test del 81% con la arquitectura RecycleNet. Finalmente establecen las limitaciones que puede tener este tipo de modelos para el caso de la aplicación real en plantas de reciclaje, ya que el procesamiento de los residuos puede deformar los materiales que se quieren clasificar, perdiendo de esta forma las propiedades que permitían a los modelos clasificar dicho residuo (Bircanoğlu, Atay, Beser, & Ayyuce, 2018).

En cuanto a la estructura del trabajo, se iniciará con el establecimiento de los objetivos, este punto es fundamental puesto que sirve de guía para el resto del trabajo. Si los objetivos no son establecidos a priori es difícil saber cómo estructurar el trabajo. El siguiente punto hará referencia a la metodología empleada, a este punto se le va a dar cierta importancia ya que debe explicar de forma clara el proceso de análisis de imágenes a través de Redes Neuronales, de esta forma se conocerán los diferentes parámetros que se pueden configurar en una Red Neuronal, conociendo esto el proceso de creación de un algoritmo que se ajuste a las imágenes será más fácil. Resulta interesante conocer cómo se produce el proceso de entrenamiento de una Red Neuronal. La parte de metodología se centrará en las denominadas Redes Neuronales Convolucionales, es una tipología de red neuronal creada con el objetivo de reconocer patrones en imágenes.

Por último, en lo referente a metodología, se hablará sobre el software utilizado y como funciona por detrás. Tras conocer cuál es la teoría relativa a modelos de clasificación de imágenes se pasará a la práctica en el siguiente punto. El punto "Construcción de modelos" consistirá en buscar que arquitectura de Redes Neuronales que se adapte mejor al problema que se plantea, se probarán diferentes arquitecturas de Redes Convolucionales y procesos de entrenamiento hasta dar con el mejor modelo. Por último, el trabajo terminará con un apartado de conclusiones donde se expondrán las principales ideas a las que se ha llegado tras la realización de la investigación, así como, futuras ampliaciones que se pueden llevar a cabo.

2 Objetivos

Podemos dividir los objetivos del presente trabajo en dos grandes grupos, por un lado, objetivos generales, que harán referencia a los objetivos que deben cumplirse al final de la investigación. Por otro lado, se nombrarán una serie de objetivos secundarios, son objetivos cuya consecución es necesaria para poder llegar a cumplir los objetivos principales.

Objetivos principales

- Crear modelos de clasificación de imágenes que permitan distinguir imágenes test y clasificarlas adecuadamente, obteniendo niveles de accuracy en consonancia con los recursos computacionales disponibles.
- Conocer en profundidad el funcionamiento de una Red Neuronal, así como, la tipología de Redes Neuronales Convolucionales.

Objetivos secundarios

- Desarrollar los modelos utilizando SAS, esto implica aprender a utilizar los paquetes DLPY y SWAT, así como, el notebook Jupyter Notebook con lenguaje de programación Python 3.
- Conocer los hiperparámetros que se pueden configurar en la construcción de Redes Neuronales, tanto en el proceso de entrenamiento como en la arquitectura de la red.
- Diferenciar entre Redes Neuronales y Redes Neuronales Convolucionales.
- Identificar que preprocesamiento de imágenes es mejor para el problema en cuestión respondiendo a preguntas: ¿Qué tamaño de imágenes es óptimo para el procesamiento de las mismas?, ¿Qué técnicas de aumento de la muestra son adecuadas?
- Comparar las diferentes arquitecturas de Redes Neuronales Disponibles.
- Buscar aplicabilidad en la vida real a los modelos desarrollados desde un punto de vista meramente teórico.

En definitiva, en este trabajo fin de master se busca aplicar los conocimientos adquiridos durante el curso y profundizar en el conocimiento de una técnica tan novedosa como es el análisis de imágenes para problemas de clasificación.

3 Metodología empleada

En este apartado se va a introducir teóricamente la metodología más adelante utilizada para la parte práctica del trabajo. Se considera esta parte del trabajo fundamental, las Redes Neuronales es una metodología compleja y se debe profundizar en su conocimiento teórico para posteriormente poder aplicarla. La estructura del apartado empezará por la parte más general, explicando que son las Redes Neuronales, como es el proceso de entrenamiento de las mismas, se explicarán las Redes Neuronales Convolucionales y por último unos apuntes sobre el software utilizado.

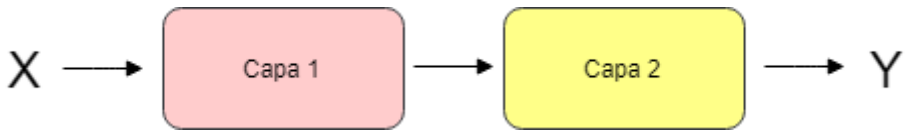
3.1 ¿Qué son las Redes Neuronales?

Un modelo estadístico de interdependencias busca relaciones entre variables, generalmente estos modelos se pueden definir como una función con una variable input y una variable output o respuesta. Esta función está formada normalmente por una serie de parámetros: el término que acompaña a las variables independientes (β_p) , un parámetro constante (β_0) y un término aleatorio ($\in$).

$$y_t = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \in$$

Las Redes Neuronales es una alternativa a los modelos clásicos que permite resolver problemas complejos de análisis de datos. Se fundamentan en la función anteriormente descrita con alguna modificación. Reciben el nombre debido a que su forma de trabajar la información se asemeja al de las neuronas biológicas. Las Redes Neuronales están formadas por diferentes capas conectadas entre sí, por estas capas se va trasmitiendo la información. La siguiente imagen ilustra como funciona un modelo de Redes Neuronales, finalmente se trata de una función compleja a la que introducimos una serie de datos input (X), está información pasa por diferentes capas donde se calculan los pesos y se buscan relaciones entre variables para finalmente obtener una predicción de la variable objetivo (Y). Las capas transmiten y transforman la información recogida de la capa anterior y la pasan a la siguiente capa, de esta forma llegan a la última capa donde se obtiene finalmente la predicción.

Imagen 1 Red Neuronal



El concepto matemático al que se debe hacer referencia es la composición de funciones, esté término se puede resumir como funciones dentro de funciones donde:

$$y_0 = capa_0\ (x)$$

$$y_1 = capa_1\ (y_0)$$

$$Y = capa_2\ (y_1)$$

$$Y = capa_2(capa_1(x))$$

Al final como se puede ver en la ecuación una red neuronal es una única función compuesta por otras funciones a su vez, el número de funciones dentro de la función principal dependerá del numero de capas con las que se configure la red.

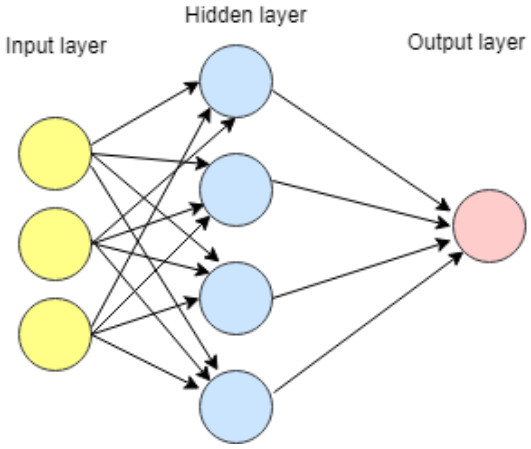
3.1.1 Estructura de las Redes Neuronales

A modo general se pueden establecer que las Redes Neuronales están compuestas por 3 tipos de capas: Input layer, Hidden layer y Output layer.

- Input layer, es la capa en la que se introduce la información con la que se quiere construir el modelo, es decir, el conjunto de datos con el que se va a realizar el entrenamiento de la red.
- Hidden layer, es la capa oculta está conectada con la capa input del modelo a través de la función de combinación. Estas capas dan grados de libertad a la red neuronal lo que le permite recoger características del entorno más fácilmente (Larrañaga, Inza, & Moujahid, n.d.).
- Output layer, es la última capa de la red neuronal que da lugar a la predicción que realiza el modelo sobre la variable objetivo.

Otro aspecto relevante de las redes neuronales son las neuronas, esto se refieren al numero de nodos que hay en cada capa. En la capa input habrá tantas neuronas como variables introduzcamos en el modelo, los nodos de las capas ocultas serán definidos por la arquitectura de la red y por último el número de neuronas en la capa output dependerá del problema de clasificación o predicción que se quiera realizar. Por ejemplo: si se quiere predecir dos categorías de una variable output, la capa output tendrá dos neuronas.

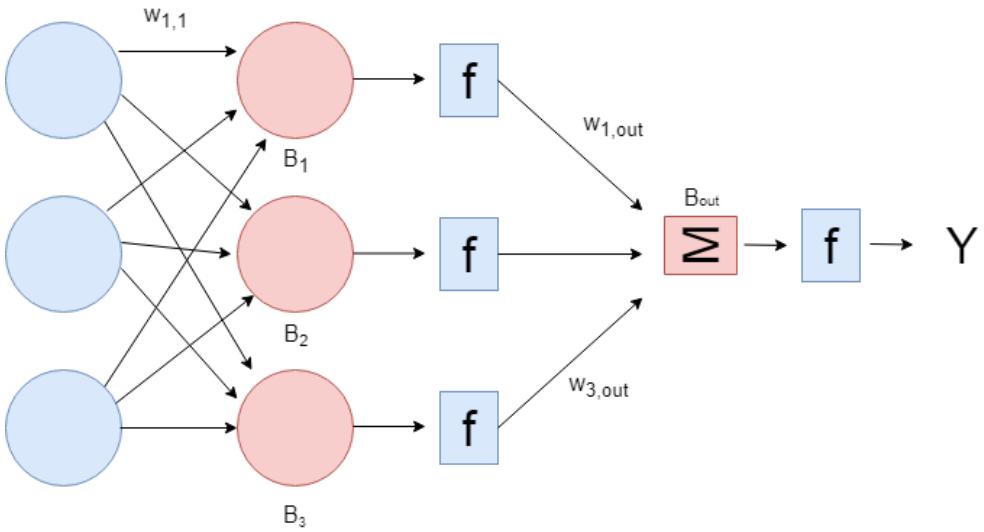
Imagen 2 Estructura Red Neuronal



El objetivo final del modelo es estimar los parámetros de la función, estos serán por un lado los pesos (W) y los bias (β), los pesos sirven para dar mayor importancia a determinadas conexiones de la red, los bias facilitan la activación de determinadas neuronas. La función de activación permite decidir si la información se va a trasladar a la siguiente neurona o no, el valor de la neurona se acotará a determinados valores según la función de activación que se utilice. La función de activación lo que permite es eliminar la linealidad de las neuronas de las capas ocultas. Finalmente, la unión de todas las capas ocultas no dará lugar a una relación lineal, la relación será no lineal y por lo tanto permitirá adaptarse a todo tipo de datos.

Para ilustrar mejor se va a representar la función de una red neuronal con tres variables input, una capa oculta con 4 neuronas y una capa output. En el diagrama se puede observar los parámetros y pesos que se deben calcular en una red, las f representan las funciones de activación por las cuales transmiten información las diferentes neuronas. La M representa la función de combinación que agrupa todos los anteriores parámetros y valores en una única función.

Imagen 3 Funcionamiento de una Red Neuronal

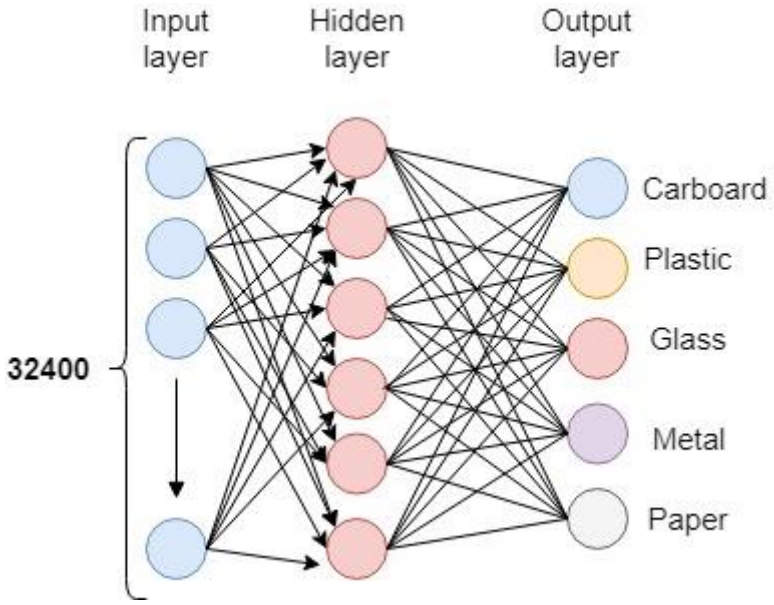


$$Y = \tanh\left(w_{1,out}\left(\tanh\left(w_{1,1}x_1 + w_{2,1}x_2 + w_{3,1}x_3 + \beta_1\right)\right)\right.$$
$$+ w_{2,out}\left(\tanh\left(w_{1,2}x_1 + w_{2,2}x_2 + w_{3,2}x_3 + \beta_2\right)\right)$$
$$\left. + w_{3,out}\left(\tanh\left(w_{1,3}x_1 + w_{2,3}x_2 + w_{3,3}x_3 + \beta_3\right)\right) + \beta_0\right)$$

En este ejemplo el número de parámetros a calcular son 16 de los cuales: 9 pertenecen a las conexiones existentes entre la capa input y la capa oculta, 3 bias de cada nodo de la capa oculta, 3 pesos de la conexión entre la capa oculta y la capa output y el último parámetro correspondiente a el bias de la capa output.

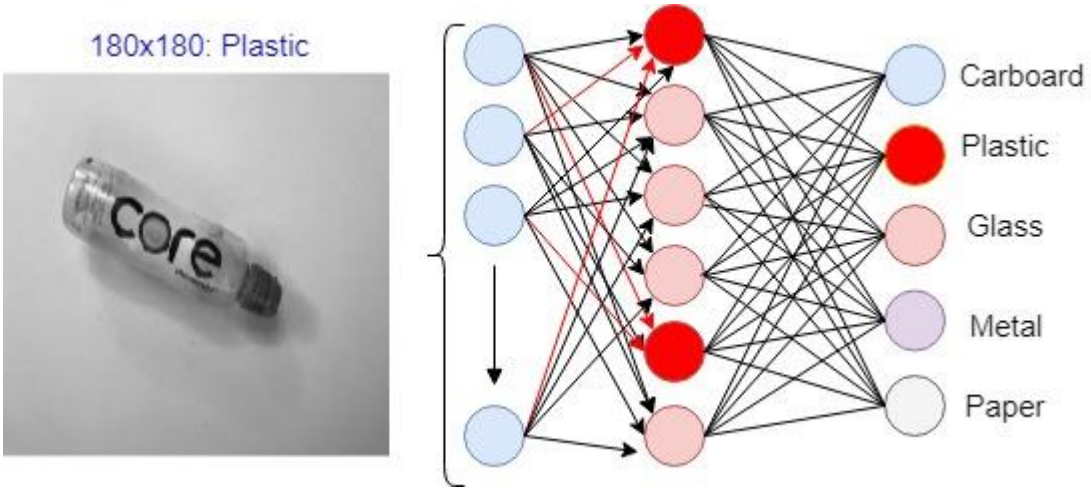
Para conocer mejor como las imágenes se modelan con redes neuronales se plantea el siguiente ejemplo: Las imágenes utilizadas tendrán una dimensión 180x180 píxeles y estarán pre procesadas en escala de grises. La capa input de esta red neuronal tendrá 32400 neuronas, cada neurona recoge información sobre la imagen en términos de color blanco o negro. Tomará valores cercanos a 0 cuando el tono del pixel sea blanco y valores cercanos a 1 cuando el tono sea negro, a este número correspondiente a cada pixel se le llama Activación. La capa de input en este modelo será un vector con 32400 valores. La capa output tendrá tantas neuronas como categorías queremos clasificar, en este caso 6 categorías. Las capas ocultas serán las encargadas de encontrar características en las imágenes, estas características permitirán encontrar patrones de activación de neuronas para cada categoría a predecir. En la siguiente imagen se puede observar un ejemplo de cómo sería la red neuronal resultante con una capa oculta, cada nodo de la capa oculta reconocerá una característica de la imagen y se activará en cada imagen que pasemos al modelo, esto hará que tras entrenar nuestra red con distintas imágenes podamos obtener una clasificación de cada imagen input.

Imagen 4 Funcionamiento de una Red Neuronal

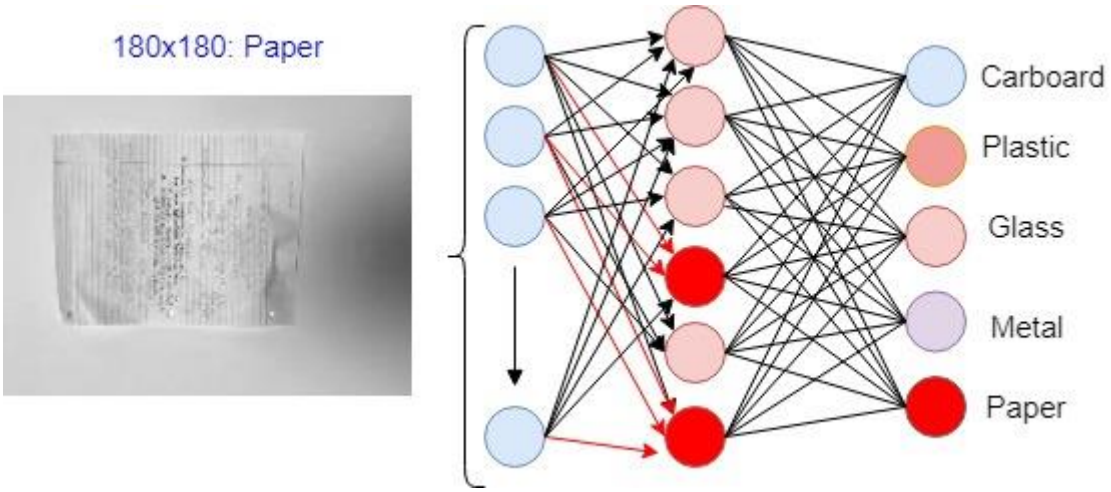


Tras el proceso de entrenamiento, la categoría 'plastic' tendrá un patrón de activación de neuronas común en la red neuronal, esto permitirá clasificar nuevas imágenes a través de la red.

Imagen 5 Patrones de activación



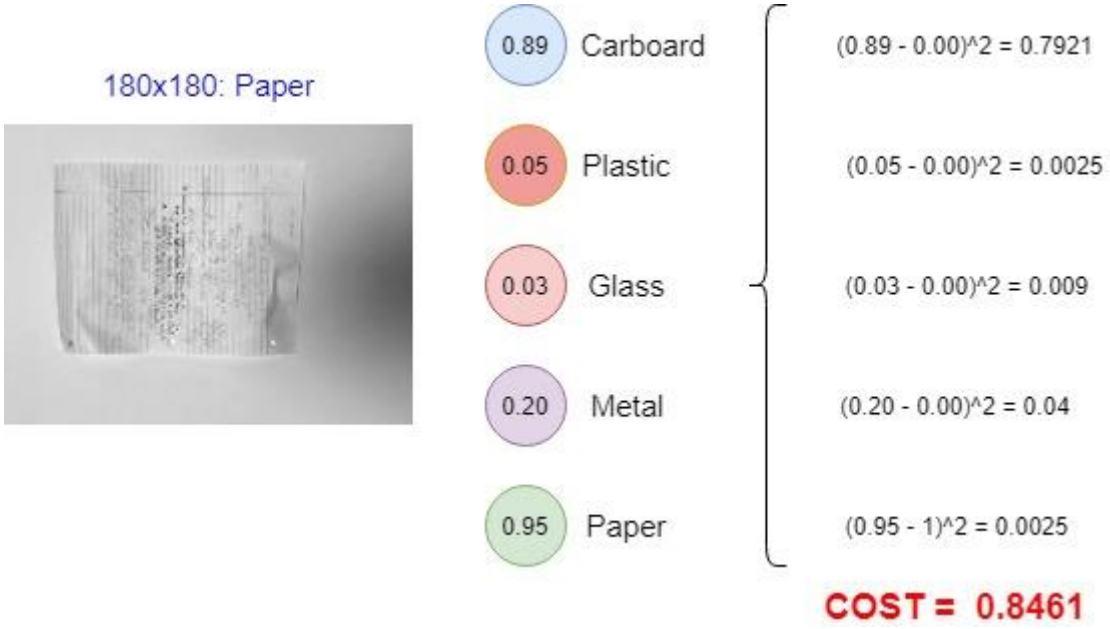
s



3.1.2 Entrenamiento de las redes

El entrenamiento de una red neuronal es el proceso por el cual se obtiene una serie de parámetros que más adelante servirán para predecir nuevas observaciones. Cada imagen del conjunto de entrenamiento puede asociarse con un coste, este coste indica como de bien clasifica la red una determinada imagen. Se utilizan los valores obtenidos en la capa output para cada categoría que queremos clasificar, conforme más cercano a 1 significa que la red considera que la imagen pertenece a dicha categoría. El cálculo del coste se lleva a cabo obteniendo el cuadrado de la diferencia entre el valor obtenido (por la red) y el valor que debería haber obtenido. En la imagen que se muestra a continuación, la observación debería ser clasificada como 'Paper' habría que restar 1 al valor obtenido por la red, en los otros casos se tomará 0 como valor a restar. Cuando la red clasifique correctamente el valor del coste será bajo y viceversa. Sobre este cálculo de coste de la red neuronal para la capa output, se aplica un algoritmo denominado "Backpropagation", consiste en una vez tenemos el coste cometido ir hacia atrás en la red para poder recalcular unos parámetros óptimos que minimicen el valor del coste.

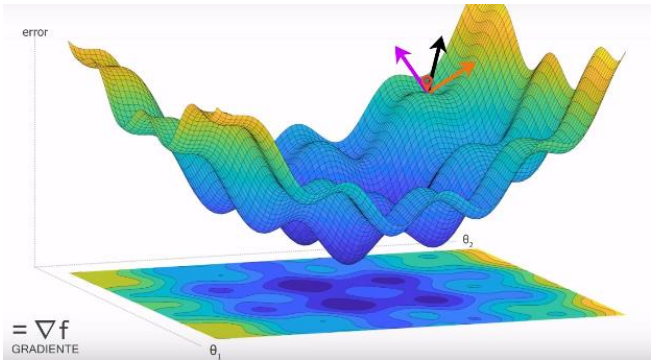
Imagen 6 Cálculo del coste



La función de coste tiene como variables input todos los parámetros de la red por lo tanto habrá que buscar aquellos parámetros de la red que minimicen el valor de la función de coste. El objeto del entrenamiento consistirá en minimizar esta función de coste. El algoritmo de optimización más utilizado en el entrenamiento de Redes Neuronales es el descenso del gradiente. Lo que se busca es encontrar el mínimo de la función, este dará los valores que deben tomar los parámetros de nuestra red para obtener buenos resultados de predicción. El descenso del gradiente es una generalización de la derivada, es un vector con tantas dimensiones como la función y cada dimensión contiene la derivada parcial en dicha dimensión. El gradiente es el vector que contienen la información de cuanto crece la función en un punto especifico por cada dimensión de la función[3].

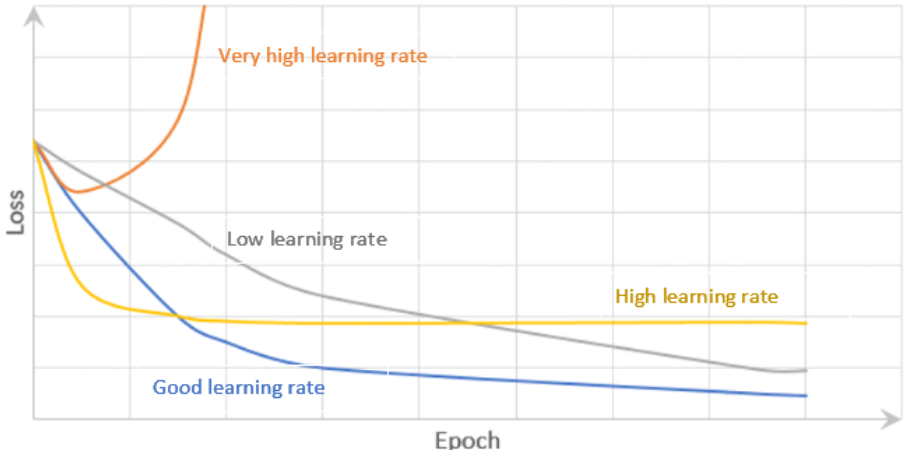
[3] https://turing.iimas.unam.mx/~ivanvladimir/posts/gradient_descent/

Imagen 7 Descenso del gradiente



En el entrenamiento es importante conocer el parámetro denominado *Learning rate*, este indica el paso que damos con cada iteración que se realiza en el descenso del gradiente, conforme más pequeño sea más lento será el proceso de entrenamiento, pero más posibilidades existirán de hallar el verdadero mínimo global. Para poder evaluar cómo se comporta el *Learning rate* durante el entrenamiento de la red el grafico que recoge información sobre la loss function puede ser util. Esta función permite observar si el *Learning rate* ha sido seleccionado correctamente.

Gráfico 2 Learning rate



[4]

Otros dos parámetros que se deben conocer del proceso de entrenamiento de una Red Neuronal son el *Bathc size* y *Epoch*. El *Batch size* controla el número de observaciones del conjunto de entrenamiento que se utilizan para calcular los pesos de forma iterativa. Es decir, se toman tantas observaciones como el Batch size indique, se calcula el error o coste cometido con esos parámetros y se actualizan los pesos. El Epcoh hace referencia a las veces que pasa el conjunto entero de training por el modelo. Para ilustrar todo mejor se propone el siguiente ejemplo: Con un conjunto de entrenamiento de 50 datos, se establece un Batch size de 10 , esto es igual a 5 batches con 10 datos cada uno y por cada Batch se actualizarán los pesos.

[4] https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10

Por cada *Epoch* entonces se harán 5 actualizaciones a los pesos del modelo, si se establecen para este ejemplo 100 *Epochs* será igual a 500 actualizaciones de los pesos del modelo.

Los problemas derivados del proceso de análisis de imágenes con Redes Neuronales tradicionales son:

- La gran cantidad de información tratada por la red y la complejidad de las conexiones, lo cual supone un coste computacional elevado.
- En el ejemplo mostrado la información de la imagen se resume en un vector, esto supone que la información de la relación entre pixeles que se encuentran cercanos se pierde, la interacción espacial entre pixeles no se contempla al introducir la información de forma vectorial. Este problema fue estudiado en el paper Gradient-based learning applied to document recognition (Lecun, Bottou, Bengio, & Haffner, 1998), en el mismo se plantean las Redes Neuronales Convolucionales como solución a este problema . Esta tipología de redes tiene en cuenta la información de pixeles adyacentes para tratar la información.

3.2 Convolutional neural networks

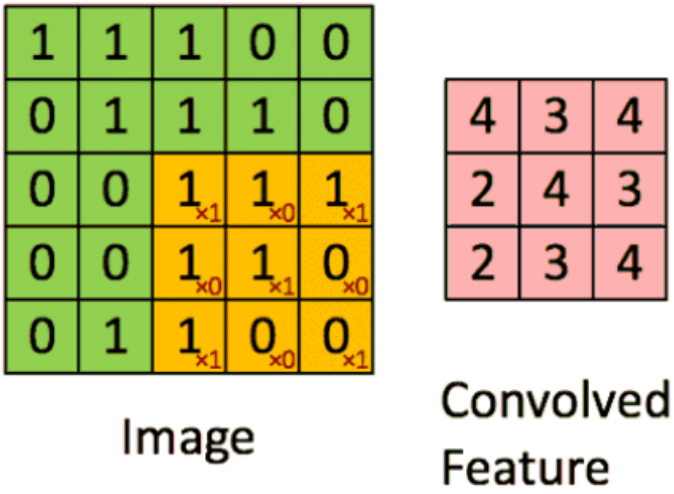
Las Redes Neuronales Convolucionales, forman parte de la familia de redes neuronal, surgen como solución para el procesamiento de imágenes. La estructura de este tipo de redes es similar a la de una red neuronal clásica, tendrá las capas input, hidden y output. Dentro de las capas ocultas podremos una nueva tipología de capas especifica de este tipo de redes, capas pooling y capas de convolución. En la siguiente ilustración podemos ver como se sigue manteniendo la estructura típica de las redes neuronales, internamente las capas convolucionales y de pooling funcionan de forma diferente a lo se ha visto hasta ahora. El principal objetivo con el que surgen las redes neuronales convolucionales es el tratamiento de imágenes, aunque se utilizan en otro tipo de problemas como text mining.

Las redes neuronales convolucionales consiguen superar dos problemas que hasta ahora las redes neuronales convencionales presentaban. Reducen la información que debe procesar las redes, esto reduce las necesidades computacionales y permite un mejor procesamiento de las imágenes. Para el caso concreto de las imágenes, las redes neuronales convolucionales permiten tener en cuenta la espacialidad de los pixeles, esto quiere decir que la capacidad de los modelos para reconocer imágenes aumenta y los resultados obtenidos por el mismo serán mejores. A continuación, se explica el funcionamiento de las capas ocultas específicas de este tipo de redes neuronales:

- Las capas convolucionales son las que se encargan de extraer patrones de las imágenes, para poder extraer los mismos aplica diferentes filtros sobre la imagen original. Estos filtros recorren toda la imagen original y extraen la información a parches. El filtro permite reducir el número de conexiones entre capas y hace más sencilla computacionalmente la red. En la imagen que se muestra a continuación se puede entender cómo funciona una capa convolucional, la imagen original es el recuadro verde, donde se dispone de la información original de la imagen. El recuadro amarillo representa el filtro

que se está aplicando a dicha imagen en este caso es un filtro de 3x3. Dicho filtro tras recorrer toda la imagen da lugar a él recuadro rosa que habrá extraído información sobre la imagen original.

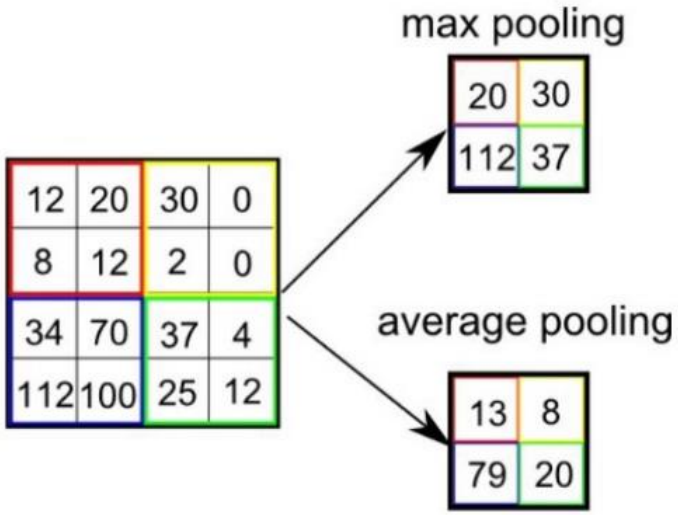
Imagen 8 Capas convolucionales



La cantidad de filtros que se aplican en una capa convolucional pueden ser más de uno, cada filtro estará especializado en la extracción de determinada información de la imagen. El resultado final de la capa de convolución serán tantas matrices convolucionales como filtros se apliquen, estas capas convolucionales se pasarán por una función no lineal, generalmente Relu. De este proceso de convolución se obtiene un resultado de matrices que pasaran a la siguiente capa con unas menores dimensiones que la imagen inicial, pero con mayor profundidad, la profundidad dependerá del número de filtros que se apliquen.

- Tras una capa de convolución siempre hay una capa de pooling, esta capa aplica dos cálculos principales para la reducción de la dimensión de las imágenes. Por lo tanto, estas capas no se encargan de la detección de patrones en las imágenes, sus tareas fundamentales son la reducción de la información disponible y evitar el sobre ajuste del modelo. Existen dos tipos de pooling el max pooling y el average pooling. En la siguiente imagen se puede observar cómo funcionan los cálculos de reducción de la dimensión llevados a cabo por una capa pooling. El average pooling tomará como valor la media del conjunto de pixeles del parche establecido y el max pooling tomará el valor máximo del conjunto de pixeles.

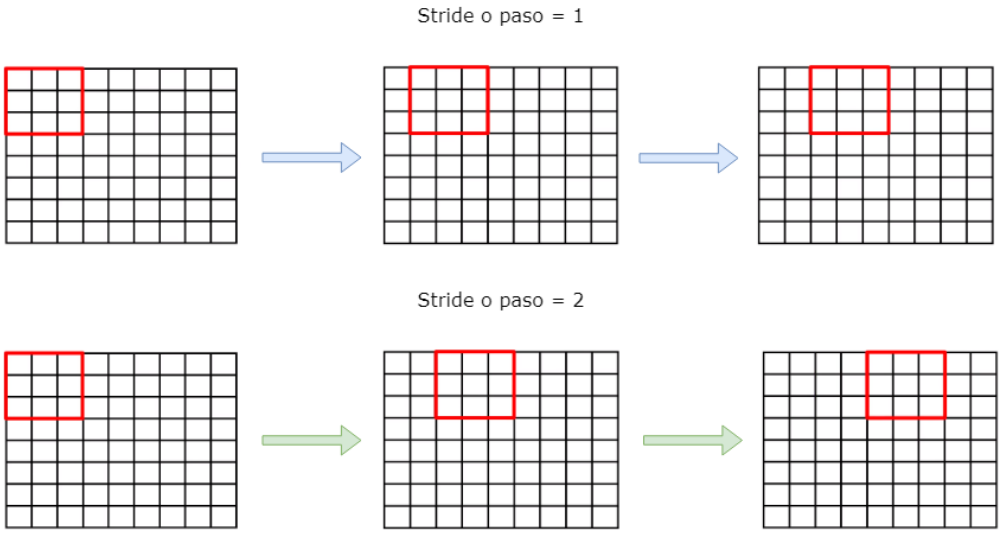
Imagen 9 Tipos de pooling



Types of Pooling

El ultimo parámetro que es interesante conocer en este tipo de redes es el stride o paso. Este se establece en la arquitectura de la red e indica cuantos pixeles se dejan cada vez que un filtro toma información. En la ilustración se puede observar cómo funciona el stride y como trabaja el filtro según el valor que demos a este parámetro. Según el funcionamiento del mismo cuanto mayor sea el stride más se reducirá la información obtenida, la matriz resultante de la aplicación del filtro menores dimensiones tendrá.

Imagen 10 Stride o paso



En definitiva, las redes neuronales convolucionales son una tipología especifica de redes neuronales que superan los problemas que tenían las redes neuronales tradicionales para el procesamiento de imágenes. En cuanto al proceso de entrenamiento de la red es similar al explicado anteriormente. En este tipo de redes una parte fundamental es la definición de la arquitectura de la red, existe mucha bibliografía y trabajos sobre este aspecto.

3.2.1 Modelos pre-entrenados y arquitecturas predefinidas

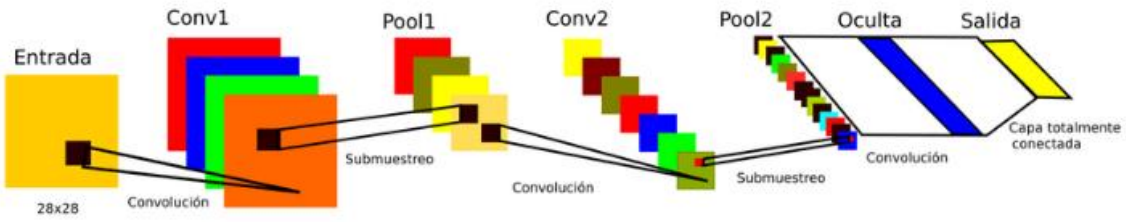
Los modelos pre-entrenados es un concepto muy importante dentro de Redes Neuronales Convolucionales ya que permiten reducir las necesidades computacionales y mejorar los resultados de los modelos. La idea es la siguiente, existe una serie de arquitecturas predefinidas que se han entrenado sobre el conjunto de datos ImageNet[5]. ImagenNet es un proyecto que proporciona un conjunto de imágenes con su información correspondiente. Este conjunto de imágenes está compuesto por 1,2 millones de imágenes con 1000 clases diferentes, es utilizado en las competiciones de ILSVRC (ImageNet Large Scale Visual, Recognition Challenge), se trata de una competición mundial sobre clasificación de imágenes al que acuden grupos de investigación para aplicar diferentes arquitecturas y encontrar la mejor.

Al final este tipo de aportaciones quedan recogidas en artículos de investigación donde se explica la arquitectura de la red y pone a disposición los pesos obtenidos en el entrenamiento de estas redes, dando la posibilidad de usar dichos desarrollos para la aplicación de otros problemas de clasificación más específicos. Se pueden utilizar los pesos obtenidos en el desarrollo de redes neuronales, se conocen a estos modelos como redes pre-entrenadas. La idea principal es poder reducir tiempos de ejecución en los modelos ya que los pesos no se inician de forma aleatoria, sino que ya tienen un sentido desde el inicio del proceso de entrenamiento. El proceso de utilización de pesos pre-entrenados para un problema de clasificación concreto se denomina en inglés "Fine-tuning".

Las arquitecturas de redes convolucionales más comunes son:

- LeNet- se conoce a esta clasificación como la primera aplicación de las redes neuronales convolucionales exitosa , elaborada por Yann LeCun , consistió originalmente en un problema de clasificación de dígitos (Lecun et al., 1998).

Imagen 11 Arquitectura LeNet



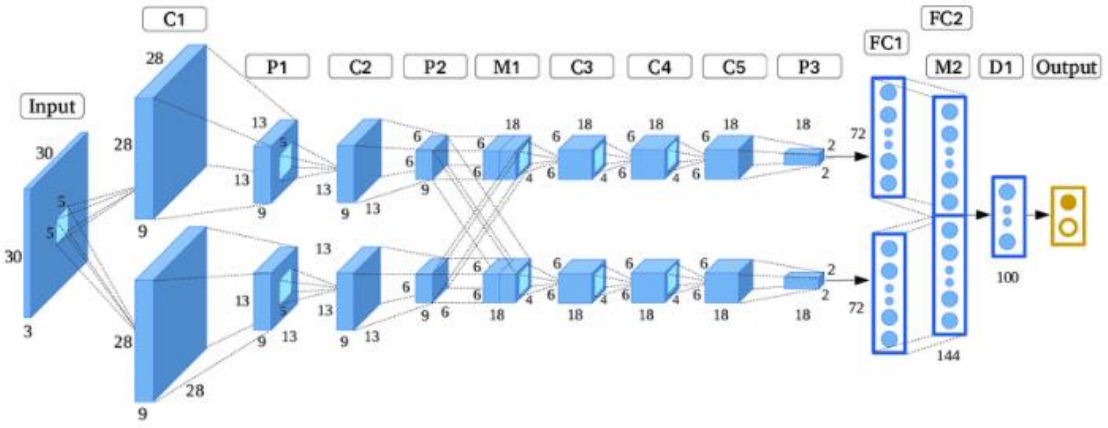
Fuente: https://www.researchgate.net/figure/Figura-46-La-arquitectura-LeNet-consiste-en-dos-capas-convolucionales-capas-de_fig6_329453166

[5] http://www.image-net.org/

Imagen 12 MINIST base de datos

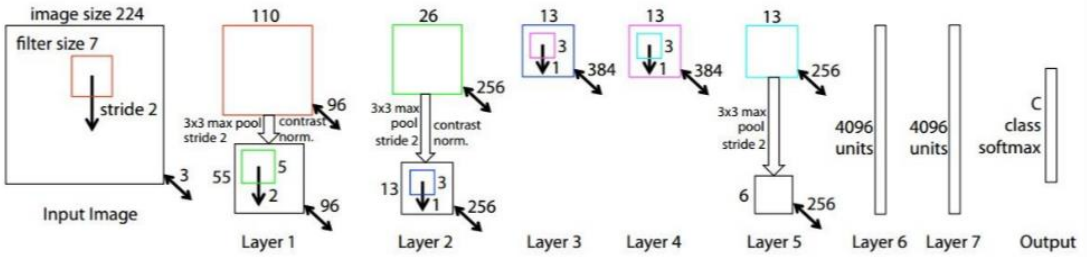
- AlexNet, es la primera arquitectura de redes neuronales que se estandariza tras la participación de la misma en el ILSVRC. AlexNet tuvo un resultado muy por encima de sus competidores, obtuvo un top 5 error de 16%, diez puntos porcentuales por encima del segundo clasificado. Este error quiere decir que el 16% de las imágenes de test no tienen entre las 5 probabilidades predichas la categoría correcta de la imagen. En cuanto a la arquitectura de esta red, es similar a la LeNet añadiendo complejidad, mayor profundidad de las capas convolucionales, se apilan capas convolucionales unas encima de otras. Esta arquitectura supone una innovación, hasta el momento las capas ocultas estaban formadas por una única capa convolucional(Monien, Preis, & Schamberger, 2007).

Imagen 13 Arquitectura AlexNet



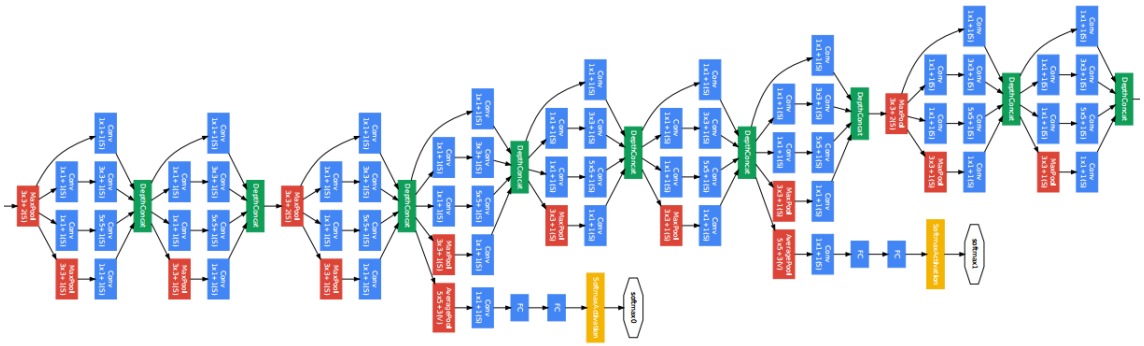
- ZF Net, gano la edición 2013 de ILSVRC, supuso una mejora de la arquitectura planteada AlexNet, para ello cambio hiper parámetros de la red. Se amplió el tamaño de las capas convolucionales medias utilizando un menor paso o stride en la primera capa (Zeiler & Fergus, 2014).

Imagen 14 Arquitectura ZFNet



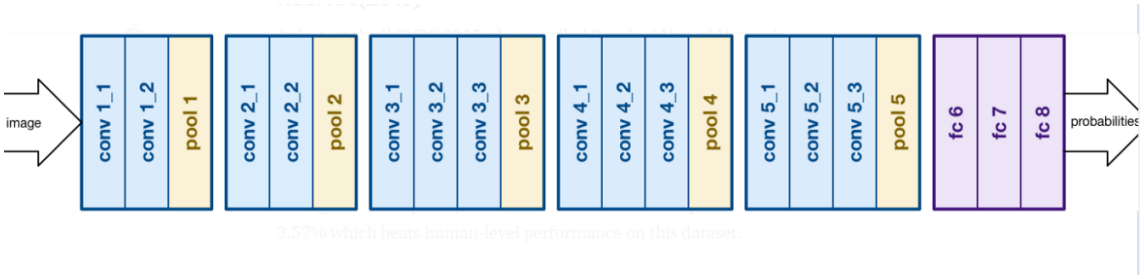
- GoogleNet, esta red neuronal creada por Szegedy et al supuso una simplificación sobre los modelos anteriores, consiguió una reducción del número de parámetros de la red. Esta mejora se basó en la modificación de la arquitectura de la red, en vez de terminar la red con una fully conected layers se utilizó una capa de pooling utilizando el método de average. Esta nueva red tiene 4 millones de parámetros, frente a la AlexNet que tiene 60 millones de parámetros supone una notable mejora, el proceso de entrenamiento será mucho menos costoso ya que no se tendrán que calcular tantos parámetros(Szegedy et al., 2015).

Imagen 15 Arquitectura GoogleNet



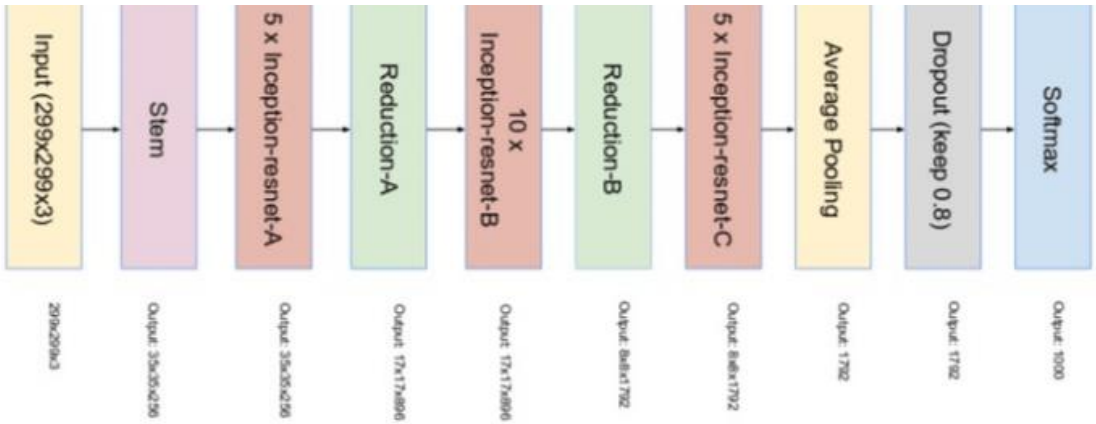
- VGG16, fue el modelo subcampeón en 2014 en ILSVRC, con este modelo se demostró que la profundidad de la red es una parte fundamental para obtener mejores resultados, esto supone un mayor número de parámetros, quizás fue la razón por la cual este modelo no fue el primer clasificado. Obtuvo unos resultados de top 5 error del 7,4% sobre el conjunto de datos test (Simonyan & Zisserman, 2014).

Imagen 16 Arquitectura GoogleNet



- ResNet, ganadora del ILSVRC en la edición del 2015, destaca como el modelo GoogleNet por no terminar con una capa fully conected. Esta arquitectura es considerada por muchos investigadores como la red que mejor funciona en problemas de clasificación (He, Zhang, Ren, & Sun, 2016).

Imagen 17 Arquitectura ResNet



En resumen, estas son las principales arquitecturas aplicadas en el mundo de las Redes Neuronales Convolucionales, que los autores de las mismas pongan a disposición del público los pesos obtenidos facilita mucho aplicar nuevos modelos. Gracias a ello se optimiza el proceso de entrenamiento y se facilita la obtención de mejores resultados para cualquier problema de clasificación de imágenes.

3.3 Software utilizado

En este trabajo se ha utilizado SAS como software principal para el desarrollo del mismo. En la última versión de SAS (SAS Viya) entre otras nuevas funcionalidades se ha incorporado la posibilidad de ejecutar códigos de software libre, más concretamente R y Python. Esto a aumentado las posibilidades y recursos en la plataforma de forma exponencial. Esto surge como solución a la incipiente entrada de software libre en el mundo del análisis de datos, con esta nueva posibilidad el potencial de SAS como herramienta analítica se incrementa exponencialmente. En los siguientes párrafos se va a explicar el funcionamiento de la plataforma, concretamente para las tareas que interesan, en este caso, análisis de imágenes.

Lo primero que se debe conocer de esta nueva plataforma es que ofrece procesamiento de datos en la nube. Esta plataforma se llama Cloud Analytics Services (CAS), es un servidor que proporciona el entorno de tiempo de ejecución basado en la nube para la gestión de datos y análisis con SAS. CAS utiliza una combinación de hardware y software donde la gestión de datos y el análisis tienen lugar en una sola máquina o en un servidor distribuido en varias máquinas[6]. En otro lado tenemos el paquete SAS SWAT es una interfaz de Python para SAS Cloud Analytic Services (CAS) Con este paquete, puede cargar y analizar conjuntos de datos de cualquier tamaño en su escritorio o en la nube. Dado que CAS se puede utilizar en un escritorio local o en un entorno de nube alojado, puede analizar conjuntos de datos extremadamente grandes utilizando la potencia de procesamiento que necesite, al tiempo que conserva la facilidad de uso de Python y las posibilidades que esto da para el procesamiento de datos[7].

Con SWAT, puede ejecutar flujos de trabajo de acciones analíticas CAS, luego extraer los datos resumidos para procesarlos más en el lado del cliente en Python, o fusionarse con datos de otras fuentes utilizando estructuras de datos Pandas familiares. De hecho, el paquete SWAT imita gran parte de la API del paquete Pandas para que el uso de CAS sea familiar para los usuarios actuales de Pandas. El interpretador de código utilizado en viya para ejecutar código de R y Python es Jupyter Notebook. Jupyter Notebook es una aplicación web de código abierto que le permite crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo. Los usos incluyen: limpieza y transformación de datos, simulación numérica, modelado estadístico, visualización de datos, aprendizaje automático y mucho más[8].

SAS ha creado un paquete especial para el procesamiento de imágenes y la creación de redes neuronales. DLPy es un paquete de alto nivel para las API de Python creadas para el back-end de SAS Viya 3.3 (y más reciente) Deep Learning. Proporciona una manera conveniente de aplicar funcionalidades de aprendizaje profundo para resolver los problemas de visión por computadora, PNL, pronóstico y procesamiento del habla. Este paquete facilita en gran medida tareas de clasificación de imágenes, tiene opciones de preprocesamiento, aumento del conjunto de datos, arquitecturas de redes típicas como RESNET y VGG16 con sus pesos pre-entrenados correspondientes.

[6] https://documentation.sas.com/?docsetId=calcld&docsetTarget=n03001viyaservers000000admin.htm&docsetVersion=3.2&locale=en

[7] https://developer.sas.com/apis/swat/python/v1.1.0/

[8] https://developer.sas.com/apis/swat/python/v1.1.0/

4 Construcción del algoritmo

En este punto empieza la parte práctica de la investigación, implicará la presentación del conjunto de datos, el preprocesamiento de los mismos y la construcción de modelos. El proceso de creación de un algoritmo de este tipo es un proceso gradual, los primeros modelos pueden que no tengan resultados muy buenos en términos de error, estos sin embargo marcarán el camino hacia el modelo final. La idea de este apartado es empezar por arquitecturas de Redes Neuronales Convolucionales sencillas y terminar aplicando modelos pre-entrenados, este proceso permitirá conocer mejor el funcionamiento de las redes desde una óptica más practica que en la primera parte del presente trabajo.

4.1 Conjunto de datos

El conjunto de datos es público[9], se permite el uso del mismo para cualquier fin divulgativo-científico. En total el conjunto de datos tiene información sobre 2522 imágenes clasificadas en 6 categorías de residuos. Un resumen del número de imágenes que se tienen etiquetadas en cada una de las categorías se muestra en la tabla 1. La categoría con mayor representación el 'Paper', acumula un 24% de la muestra. En el lado opuesto se encuentra la categoría 'Trash' la cual solo representa un 5% de la muestra, esto, puede suponer un problema a la hora de obtener modelos que clasifiquen bien. El desequilibrio entre clases provoca que se pierda capacidad para clasificar nuevas instancias (Espinar, 2018). Se podrían proponer soluciones como el aumento del conjunto de datos en esa categoría, por criterios propios se decide desechar dicha categoría para los futuros modelos por dos motivos fundamentales. Primero, no es una categoría determinante ya que se refiere a la basura orgánica, este tipo de residuos ya tienen un proceso independiente al que siguen otros residuos reciclables. Así como, quitar una categoría facilitará la creación de modelos más óptimos y que se acerquen a los resultados esperados.

Tabla 1 Frecuencias de las imágenes

	Frecuencia	**%**
Cardboard	403	15.98
Glass	501	19.87
Metal	410	16.26
Paper	594	23.55
Plastic	482	19.11
Trash	132	5.23

Las imágenes se han recogido realizando fotografías a diferentes residuos sobre un fondo blanco, estas se han realizado con un foco de iluminación para poder ver bien los detalles de cada residuo.

[9] https://github.com/garythung/trashnet

Imagen 18 Muestra imágenes

4.2 Preprocesamiento de datos

En los modelos estadísticos clásicos se lleva a cabo una depuración de los datos, limpieza de errores, transformaciones… Pues bien, en el análisis de imágenes esto se traduce al preprocesamiento de imágenes. Una imagen está formada por x pixeles de altura y x pixeles de anchura, los pixeles a su vez se forman por la combinación de colores primarios. Cuando la imagen tiene un único canal es que los pixeles solo están formados por uno de los colores primarios y es cuando se dice que la imagen está en escala de grises. Esto tiene cierta importancia ya que incide directamente sobre la dimensión que tendrán las capas input de las Redes Neuronales creadas. En el caso de las imágenes disponibles tienen los tres canales de color, por lo tanto, cada pixel es una combinación de los tres colores primarios. Los principales estadísticos nos indican que las imágenes tienen una anchura y altura heterogénea, esto representa un problema para la aplicación de redes neuronales y por lo tanto habrá que aplicar técnicas de redimensionado para homogeneizar las imágenes.

Tabla 2 Resumen de las imágenes

MinWidth	2448
MaxWidth	4032
MinHeight	2448
MaxHeight	4032
Mean1stChannel	154.529
Mean2ndChannel	162.978
Mean3rdChannel	171.224

Una parte importante del preprocesamiento de imágenes es redimensionar las imágenes (*resize*), lo que se consigue es tener un conjunto de imágenes con las mismas dimensiones en términos de anchura por altura, es necesario para poder aplicar modelos posteriormente. Con esto se consigue reducir el tiempo de entrenamiento ya que aplicaremos un *resize* que reduzca la información a procesar por la Red Neuronal. Como podemos ver en la siguiente ilustración, el proceso de redimensionamiento de las imágenes permite homogeneizar las mismas en términos de anchura y altura. En este caso la anchura y altura definida ha sido 384x512.

Imagen 19 Resize de imágenes

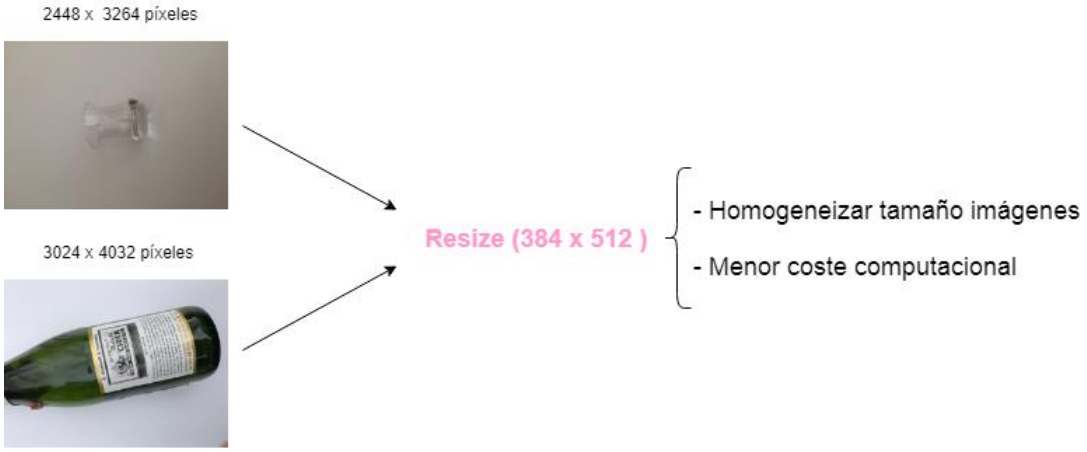


Tabla 3 Estadísticos después del resize

MinWidth	384
MaxWidth	384
MinHeight	512
MaxHeight	512

Como anteriormente se indicó nuestra muestra cuenta con 2522 observaciones, de las cuales se han excluido las pertenecientes a la categoría 'Trash', el nuevo tamaño muestral será de 2390 imágenes. Es un tamaño razonable, aun así, existe la posibilidad de aumentar el tamaño muestral con el objetivo de mejorar los resultados. A estas técnicas de aumento del conjunto de datos se les denomina 'Augmenting data' (aumento de la muestra), a través del procesamiento de las imágenes con determinados parámetros se puede conseguir ampliar la muestra más que notablemente. Cabe señalar que el aumento de la muestra se suele hacer sobre el conjunto de entrenamiento, ya que es el que se va a utilizar para entrenar la red. Existen numerosas técnicas de aumento de la muestra; *image shifts*, *image crop*, *image flips*, *image zoom*, *image rotation*, *image patches* y muchas más, en este caso nos vamos a centrar en explicar las siguientes:

- *Image patches*, esta forma de aumentar el conjunto de datos se lleva a cabo a través de una ventana de recorte con una determinada anchura y altura, realiza recortes sobre la imagen original para obtener nuevas imágenes. En la siguiente imagen se muestra el conjunto de imágenes obtenido de única imagen tras aplicarle esta técnica de aumento de la muestra.

Imagen 20 Aumento de la muestra con patches

```
imagen.as_patches(width=200, height=200, step_size=15, output_width=224, output_height=224)
```

```
imagen.show()
```

- *Image rotation*, en este caso obtenemos conjunto de datos con las imágenes orientadas en diferentes direcciones. De la misma forma se pueden aplicar distintas técnicas de aumento del conjunto de datos, obteniendo de esta forma un conjunto de datos de entrenamiento bastante amplio. En el ejemplo que se muestra a continuación se ha utilizado una función que permite aplicar varias técnicas a la vez; *rotation, flip y dark* (marco negro alrededor de la imagen).

Imagen 21 Aumento de la muestra con random mutations

```
imagen_crop.random_mutations(color_jitter=True, color_shift=True, darken=True, horizontal_flip=True, invert_pixels=False, lighte
```

```
imagen_crop.show()
```

En resumen, el conjunto de datos de entrenamiento se puede incrementar con facilidad utilizando este tipo de técnicas, sin embargo, esto no asegura que los resultados del modelo vayan a mejorar. Se deberá valorar por lo tanto como influyen este tipo de procesamiento de datos sobre los resultados del modelo, así como, si los tiempos de ejecución compensan para los resultados que se obtengan.

4.3 Modelos creados de forma secuencial

En este apartado se empezará por los modelos más básicos aplicados al conjunto de datos, de esta forma se conseguirá una aproximación a los datos y a las Redes Neuronales Convolucionales. En primer lugar, se debe decidir el número de capas que va a tener el modelo, es decir la arquitectura, esta primera arquitectura será muy sencilla, únicamente tendrá dos capas ocultas, este primer modelo se llamará "Modelo secuencial 1". La arquitectura es la siguiente, se adjuntan el código utilizado para crear esta primera red, en las siguientes se omitirá.

- Una capa input con 3 canales, las imágenes input tendrá información de 224 x 224 pixeles.

```
model_secuencial1.add(InputLayer(3, 224, 224, offsets=tr_img.channel_means))
```

- Una capa convolucional seguida de una capa de pooling. La capa convolucional estará formada por 8 filtros con un tamaño de 7x7, la capa de pooling tendrá un tamaño de 4x4.

```
model_secuencial1.add(Conv2d(8, 7))
model_secuencial1.add(Pooling(4))
```

- Una fully-conected layer con 16 neuronas.

```
model2.add(Dense(16))
```

- Por último, una capa oculta con función de activación softmax, y el número de neuronas igual a 5 que son las categorías que se quieren clasificar.

```
model1.add(OutputLayer(act='softmax', n=5)
```

Un resumen de las diferentes capas se puede observar en la siguiente tabla, en total se dispone de 5 capas. En esta tabla es interesante observar como la primera capa convolucional provoca un aumento de las dimensiones de la red hasta 8, las funciones de activación para las capas convolucionales son 'Relu' y para la capa output 'Softmax'. El número de parámetros va aumentando en cada capa hasta llegar a un total de 201941, es prácticamente imposible entender cómo se relacionan las diferentes neuronas dentro de a la red, sin embargo, lo que interesa se obtiene en la capa output, las cinco neuronas que componen esta capa nos darán la predicción de la red para una determinada imagen. El proceso de entrenamiento se configura con los mismos parámetros para poder comparar la eficiencia de las arquitecturas. El entrenamiento tendrá un *Batch size* de 64, *Epoch* o épocas igual a 50 y una tasa de aprendizaje o *Learning rate* de 0,01.

Tabla 4 Resumen modelo secuencial 1

	Layer Id	Layer	Type	Kernel Size	Stride	Activation	Output Size	Number of Parameters
0	0	Input1	input			None	(224, 224, 3)	(0, 0)
1	1	Convo.1	convo	(7, 7)	1	Relu	(224, 224, 8)	(1176, 8)
2	2	Pool1	pool	(4, 4)	4	Max	(56, 56, 8)	(0, 0)
3	3	F.C.1	fc	(25088, 8)		Relu	8	(200704, 0)
4	4	Output1	output	(8, 5)		Softmax	5	(40, 5)
5								201933

	Valor
Learning rate	0,01
Bathc size	64
Epochs	50
Número de capas	5
Capas input	1
Capas output	1
Capas convolucionales	1
Capas de pooling	1
Número de pesos (weights)	201920
Números de Bias	21
Número total de parámetros	201941
Coste de memoria del entrenamiento (MB)	4540

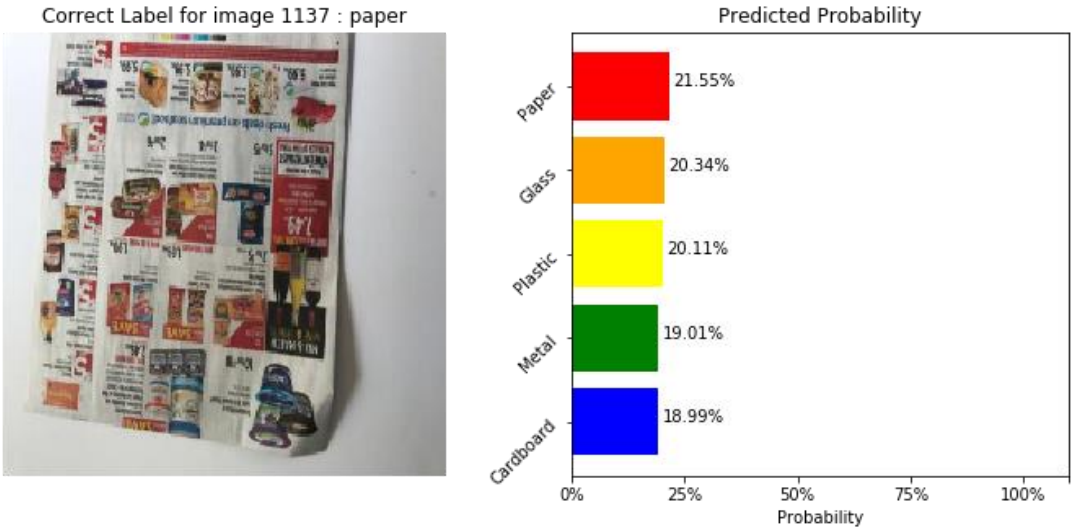
Los resultados obtenidos por esta red neuronal sobre el conjunto de datos test es de un 75%. Esto quiere decir que del conjunto de observaciones que pertenecen a test, el modelo solo ha clasificado bien un 25% de ellas. Estos resultados son normales, para poder detectar patrones la Red Neuronal debe contar con una mayor complejidad en términos de arquitectura, así como, un proceso de entrenamiento. Durante el proceso de entrenamiento la red no ha sido capaz de reducir el error.

Tabla 5 Resultados modelo secuencial 1

	Conjunto test	**Conjunto entrenamiento**
Número de observaciones evaluadas	478	1912
Misclassification error (%)	75,11	75,16

En el siguiente gráfico se pueden observar cómo predice una imagen concreta del conjunto de datos test, como se puede observar, aunque la predicción es correcta ya que el modelo decide que la fotografía pertenece a la categoría 'Paper', asignando a esta categoría una probabilidad de 21,55%, la diferencia entre las probabilidades predichas es mínima. Esto indica que el modelo no segmenta correctamente las categorías a clasificar. Se deberán buscar otro tipo de arquitecturas o procesos de entrenamiento que mejoren estos modelos.

Gráfico 3 Predicción modelo secuencial 1



Vistos lo resultado obtenidos se procede a cambiar la arquitectura de la red neuronal con el objetivo de reconozca más patrones de las imágenes y mejoren los resultados del modelo. El siguiente modelo "Modelo secuencial 2", contará con un mayor número de capas se ha aumentado el número de capas ocultas, se han añadido una capa convolucional y una capa de pooling.

Tabla 6 Resumen modelo secuencial 2

	Layer Id	Layer	Type	Kernel Size	Stride	Activation	Output Size	Number of Parameters
0	0	Input1	input			None	(512, 384, 3)	(0, 0)
1	1	Convo.1	convo	(7, 7)	1	Relu	(512, 384, 8)	(1176, 8)
2	2	Pool1	pool	(4, 4)	4	Max	(128, 96, 8)	(0, 0)
3	3	Convo.2	convo	(7, 7)	1	Relu	(128, 96, 8)	(3136, 8)
4	4	Pool2	pool	(4, 4)	4	Max	(32, 24, 8)	(0, 0)
5	5	F.C.1	fc	(6144, 16)		Relu	16	(98304, 0)
6	6	Output1	output	(16, 5)		Softmax	5	(80, 5)
7								102717

	Valor
Learning rate	0,01
Bathc size	64
Epochs	50
Número de capas	7
Capas input	1
Capas output	1
Capas convolucionales	2
Capas de pooling	2
Número de pesos (weights)	29480
Números de Bias	37

Número total de parámetros	29517
Coste de memoria del entrenamiento (MB)	4753

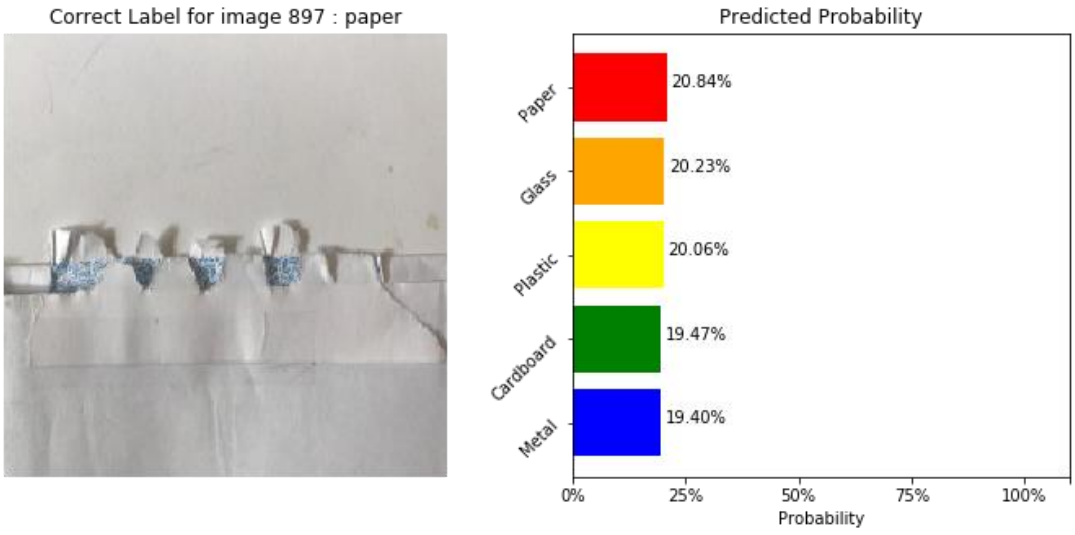
Los resultados obtenidos por este modelo secuencial 2 son de casi un 80% de error para el conjunto test.

Tabla 7 Resultados modelo secuencial 2

	Conjunto test	**Conjunto entrenamiento**
Número de observaciones evaluadas	478	1912
Misclassification error (%)	75,15	75,16

Los resultados obtenidos en términos son similares a una red más sencilla y las probabilidades predichas siguen siendo muy parecidas para todas las categorías.

Gráfico 4 Predicción modelo secuencial 2



En el siguiente modelo "Modelo secuencial 3" se añaden otras dos capas ocultas, alcanzando un total de 9 capas y 9621 parámetros.

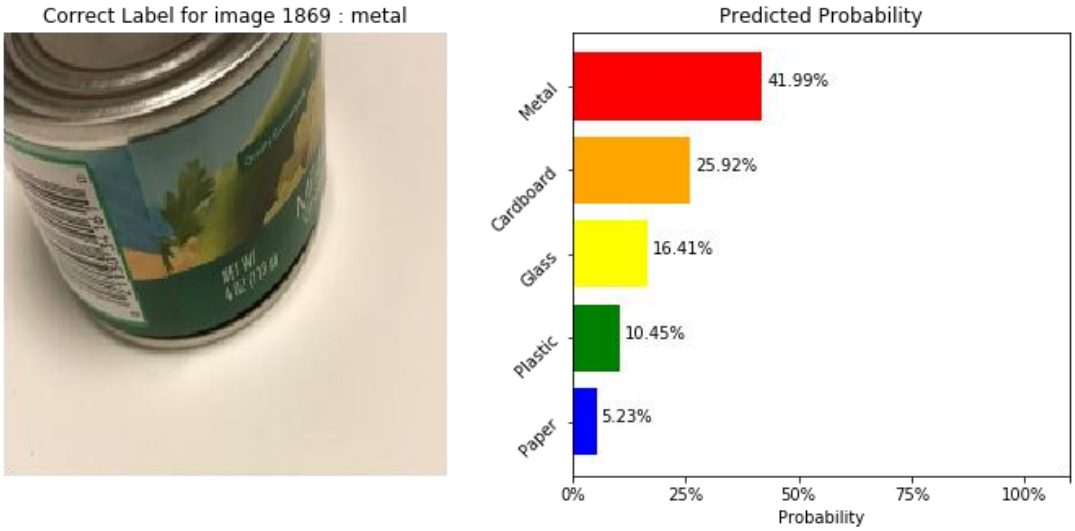
Tabla 7 Resumen modelo secuencial 3

	Layer Id	Layer	Type	Kernel Size	Stride	Activation	Output Size	Number of Parameters
0	0	Input1	input			None	(224, 224, 3)	(0, 0)
1	1	Convo.1	convo	(7, 7)	1	Relu	(224, 224, 8)	(1176, 8)
2	2	Pool1	pool	(4, 4)	4	Max	(56, 56, 8)	(0, 0)
3	3	Convo.2	convo	(7, 7)	1	Relu	(56, 56, 8)	(3136, 8)
4	4	Pool2	pool	(4, 4)	4	Max	(14, 14, 8)	(0, 0)
5	5	Convo.3	convo	(7, 7)	1	Relu	(14, 14, 8)	(3136, 8)
6	6	Pool3	pool	(4, 4)	4	Max	(4, 4, 8)	(0, 0)
7	7	F.C.1	fc	(128, 16)		Relu	16	(2048, 0)
8	8	Output1	output	(16, 5)		Softmax	5	(80, 5)
9								9605

	Valor
Learning rate	0,01
Bathc size	64
Epochs	50
Número de capas	9
Capas input	1
Capas output	1
Capas convolucionales	3
Capas de pooling	3
Número de pesos (weights)	9576
Números de Bias	45
Número total de parámetros	9621
Coste de memoria del entrenamiento (MB)	8481

En términos de error el modelo no mejora, empeora incluso, alcanzando un misclassification error del 80%, sin embargo, a la hora de obtener probabilidades sobre las imágenes de test se observa una mejora. Los porcentajes asignados a cada categoría ya son más distantes, esto quiere decir que la red es capaz de distinguir algunas características de las imágenes y por lo tanto las probabilidades que da a cada categoría muestran diferencias. En la siguiente imagen se puede observar como para una imagen etiquetada como metal la red neuronal construida da una probabilidad de esta misma categoría de un 41,99%.

Gráfico 5 Predicción modelo secuencial 3



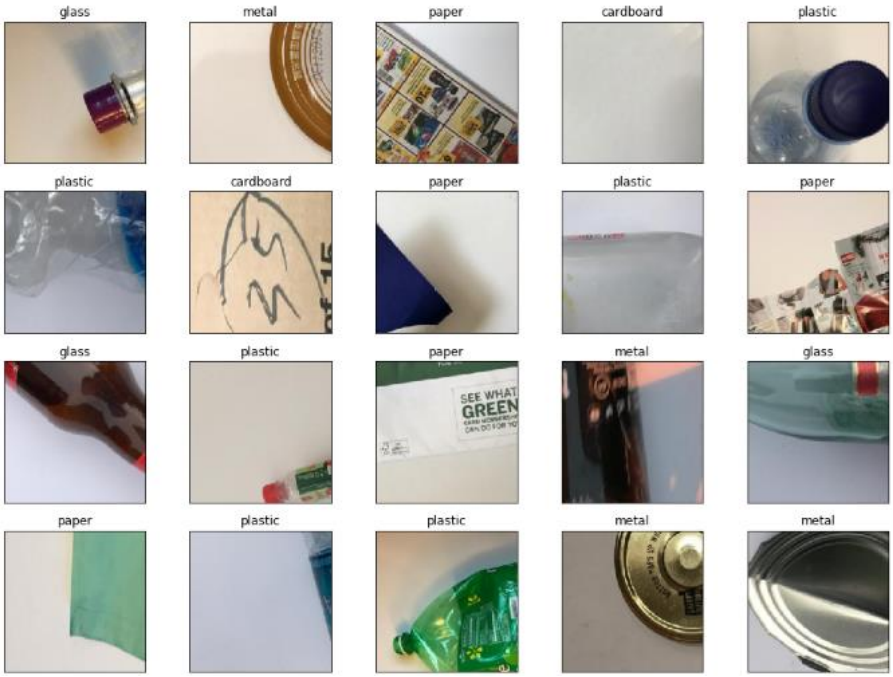
En este primer conjunto de modelos el último modelo que se va a probar tendrá la misma arquitectura que el modelo secuencial 3, la diferencia estará en el número de *Epochs* aplicadas en el entrenamiento. De esta forma el modelo tendrá opciones a obtener un mayor aprendizaje. El entrenamiento del modelo mejora notablemente hasta alcanzar un misclassification error de 45,82%. Por lo tanto, el proceso de entrenamiento incide directamente sobre los resultados obtenidos por la red, esto tienen sentido, un mayor número de *Epochs* proporciona a la red un mayor aprendizaje sobre las imágenes. Cuantas más *Epochs* se establezca, mejor serán los resultados obtenidos, sin embargo, esto supondrá mayores capacidades computacionales y mayores tiempos de ejecución.

4.3.1 Modelos secuenciales con aumento de la muestra

En este caso se ha procedido a aumentar la muestra disponible para el entrenamiento de la red, esto permitirá que la red se entrene con más imágenes, en teoría esto da más capacidad predictiva a las redes, sin embargo, no tiene por qué mejorar sustancialmente los resultados. Lo que si provoca seguro es un aumento de los recursos computacionales necesarios, deberá estudiarse por tanto si compensa el aumento de las imágenes en los resultados del modelo.

Para el primer aumento de la muestra se ha utilizado la función '*as_patches'* de la librería DLPY, esta función da como salida recortes sobre la imagen original. La función toma imágenes de 224 x 224 pixeles y aplica recortes cada 24 pixeles dando lugar a unas imágenes de salida con 200 x 200 pixeles. Las imágenes obtenidas tras el aumento de la muestra serán como las mostradas a continuación.

Imagen 22 Aumento de la muestra con patches



El aumento del conjunto de datos nos deja un total de 298272 imágenes para poder entrenar la red.

Tabla 8 Frecuencias aumento muestra as_patches

	Frecuencia	%
Cardboard	50232	16,84
Glass	62556	20,97
Metal	51168	17,15
Paper	74100	24,84
Plastic	60216	20,19

La gran cantidad de imágenes ha hecho impracticable el entrenamiento de las redes, el conjunto de entrenamiento era demasiado grande, se han cambiado los parámetros de la función para obtener un conjunto de entrenamiento más manejable, obteniendo 28230 imágenes para el entrenamiento de la red.

Tabla 9 Frecuencias aumento muestra as_patches

	Frecuencia	%
Cardboard	4830	16,84

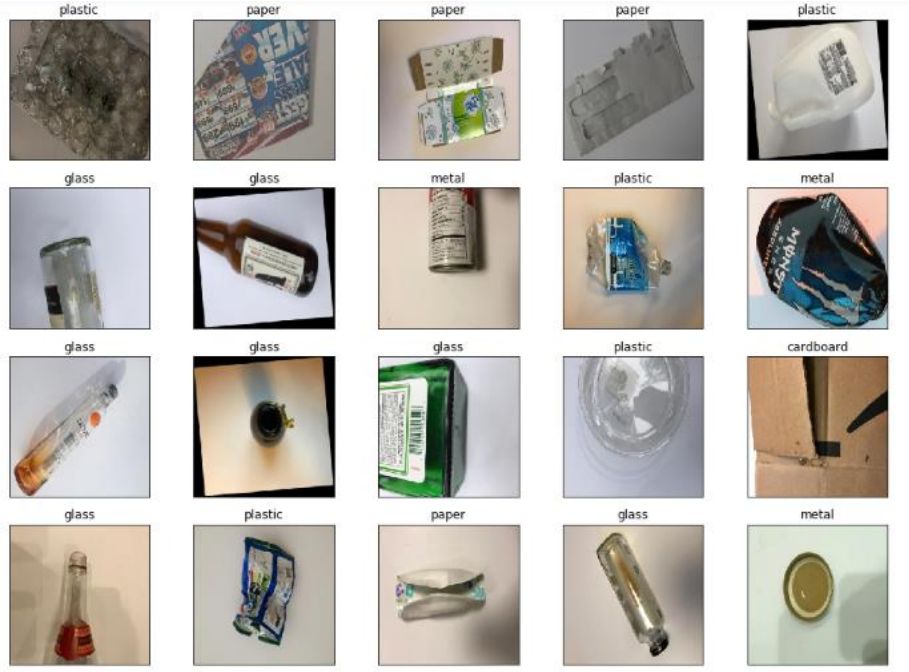
Glass	6015	20,97
Metal	4920	17,15
Paper	7125	24,84
Plastic	5790	20,19

Utilizando este conjunto de datos de entrenamiento y aplicando los modelos secuenciales creados anteriormente se obtienen resultados muy parecidos, más adelante se compararán con los modelos sin aumento de la muestra. El aumento de las imágenes por esta metodología no influye directamente sobre obtener mejores resultados. Otra opción que da el paquete DLPY es la función image_table.random_mutations, esta permite obtener un aumento del conjunto de datos de entrenamiento a través de diversas transformaciones sobre las imágenes originales. En la tabla se muestra las frecuencias de imágenes del conjunto de datos de entrenamiento para las diferentes categorías. El número de imágenes generadas, como se puede observar, es menor al resultante del otro método, lo cual facilitará el proceso de entrenamiento. Por este método se amplía el tamaño muestral del conjunto de entrenamiento hasta un total de 6840 imágenes.

Tabla 10 Frecuencias aumento muestra random mutations

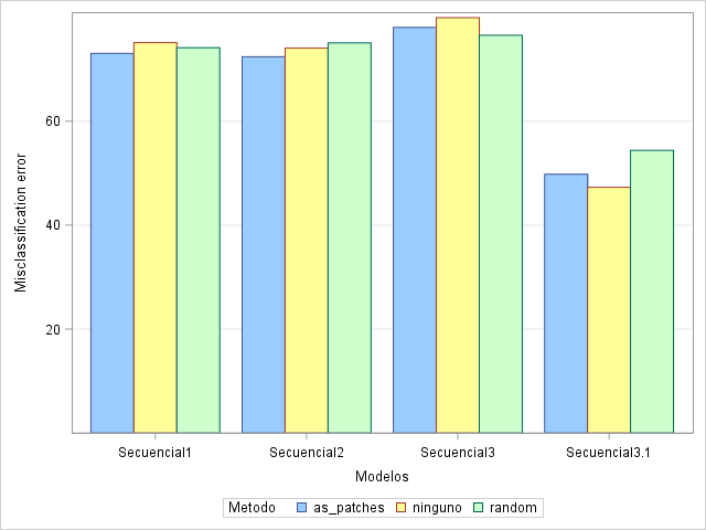
	Frecuencia	**%**
Cardboard	1169	17,09
Glass	1417	20,72
Metal	1197	17,50
Paper	1699	24,84
Plastic	1358	19,85

Imagen 23 Aumento de muestra random mutations



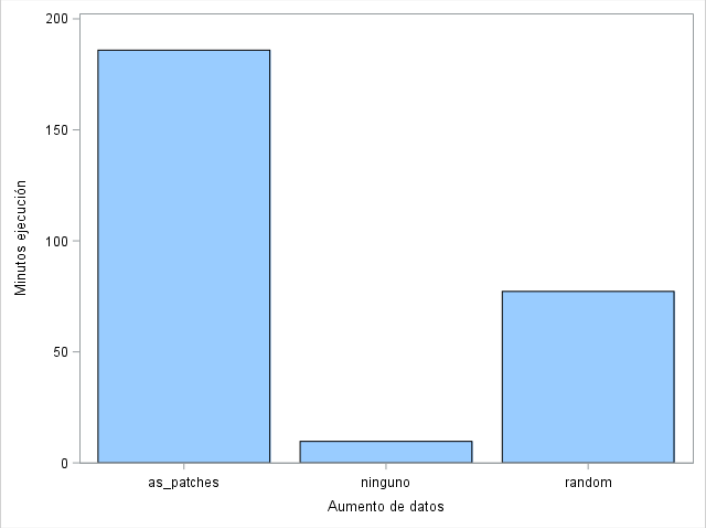
Para poder comprobar si realmente un aumento de la muestra supone ventajas en el resultado de los modelos se van a comparar todos los modelos secuenciales teniendo en cuenta si se ha aplicado aumento de la muestra y el método de aumento que se ha aplicado. En este primer gráfico observamos el misclassification error para los modelos, la aplicación de aumento de datos del conjunto de entrenamiento no ha resultado significativa sobre los resultados. Los mejores resultados se dan en el modelo secuencia 3.1, como vimos anteriormente en estos modelos se aumentaron las *Epochs* sobre la red neuronal, debe ser un parámetro a tener en cuenta para futuros modelos. En el modelo secuencial 3.1 si se pueden observar algunas diferencias, pero son mínimas, no se confirman por tanto que un aumento de la muestra en el conjunto de entrenamiento mejore los resultados.

Gráfico 6 Misclassification error en modelos secuenciales



En cuanto a los tiempos de entrenamiento existe una clara diferencia entre las diferentes clases de modelos según el aumento o no de la muestra de entrenamiento. En el siguiente grafico se representa la media de tiempo (minutos) que tardan los modelos en completar el proceso de entrenamiento según si se ha aumentado la muestra y el método de aumento de la misma. Se puede observar que la media más alta la alcanzan aquellos modelos que han llevado a cabo el aumento con la función as.patches , estos modelos en media tardan en realizar el entrenamiento un total de 186 minutos. Los modelos que aplican 'random mutations' como método de aumento del conjunto de datos tardan 77 minutos. Por último, los modelos sobre los que no se ha aplicado ningún tipo de aumento de datos la media de entrenamiento e 9 minutos.

Gráfico 7 Minutos de ejecución



Se deberá considerar que el aumento de datos por lo tanto como una opción poco aplicable a este tipo de modelos, aumenta los costes computacionales y tiempos de ejecución sin obtener grandes resultados.

4.4 Modelos con arquitecturas predefinidas

En vista de que los resultados no son buenos, se busca una alternativa que permita obtener mejores resultados. En la librería DLPY se permite el uso de arquitecturas predefinidas como son ResNet50, VGG16, InceptionV3, LeNet y muchas otras para diferentes problemas de redes neuronales, tanto clasificación como localización de objetos en imágenes.

La primera que se va a utilizar es la **arquitectura LeNet**, esta arquitectura está compuesta por 8 capas, una capa de input y de output respectivamente, dos capas de pooling, dos capas convolucionales y dos capas fully conected. El proceso de creación de un modelo con una arquitectura predefinida en términos de código es diferente al caso de los modelos secuenciales, se va a adjuntar y explicar para el caso del LeNet, para el resto de arquitecturas sería un proceso similar.

Se define el modelo con la función dlpy.applications.LeNet , tiene como parámetros , la conexión al cas server , la tabla en la que se va a guardar el modelo, el número de clases que se quieren predecir, el número de canales de la imagen, en este caso son 3 , y la altura y anchura de las imágenes.

```
modelo_lenet5=dlpy.applications.LeNet5(conn,model_table='LENET5',n_classes=5,
n_channels=3, width=224, height=224)
```

Como se trata de arquitecturas predefinidas no hace falta establecer las capas una a una como en el caso secuencial, se pasa directamente al entrenamiento de la red. Utilizando la función model.fit de la librería DLPY, se establecen los parámetros: conjunto de datos para el proceso de entrenamiento (data), *Batch size*, *Epochs* y *Learning rate* o tasa de aprendizaje.

```
modelo_lenet5.fit(data=tr_img,mini_batch_size=32,max_epochs=50,lr=0.001)
```

La medición de los resultados del modelo para el conjunto de datos test se lleva a cabo con la función model.evaluate , dará como salida una tabla con los resultados del modelo. Únicamente hay que indicar a esta función cual es el conjunto de datos que queremos que evalué.

```
modelo_lenet5.evaluate(data=te_img)
```

Para obtener las probabilidades estimadas para cada imagen en particular del conjunto test. Se utiliza la función model.plot_evaluate_res , el parámetro más interesante de esta función es img_type en el que se puede establecer las imágenes a mostrar 'M' para las imágenes mal clasificadas , 'C' para las bien clasificadas y 'A' para mezclar las dos anteriores.

```
modelo_lenet5.plot_evaluate_res(img_type='C', randomize=True, n_images=20)
```

En este caso también se hará uso de otras funciones del paquete DLPY que permiten evaluar los resultados obtenidos por la red. En modelos de clasificación interesa saber que categorías es capaz de clasificar con mayor éxito, para ello existe la matriz de confusión. El paquete DLPY permite obtener esta información de los modelos con la función model.valid_conf_mat , permitirá caracterizar los modelos y facilitará la comparación de los mismos.

Otra función interesante es la que permite observar mapas de calor del modelo sobre las imágenes de test. Los mapas de calor muestran en que parte de la imagen se está fijando la red neuronal para decidir cuál va a ser la clasificación de una imagen concreta. Este tipo de mapas son muy útiles ya que permiten observar que está haciendo la red neuronal.

```
modelo_lenet5.heat_map_analysis(data=None, mask_width=None,
mask_height=None, step_size=None, display=True, img_type='A', image_id=None,
filename=None, inputs='_image_', target='_label_', max_display=10)
```

El resumen del **modelo LeNet** se puede observar en la siguiente tabla, el total de parámetros a calcular por la red es de 6034470. El proceso de entrenamiento cuenta con un *Learning rate* de 0,001, un *Bathc size* de 32 y 50 *Epochs*, estos parámetros de entrenamiento serán comunes a todas las arquitecturas que se muestran a continuación para que la comparación de modelos sea consistente.

Tabla 11 Modelo LeNet

	Valor
Learning rate	0,001
Bathc size	32
Epochs	50
Número de capas	8
Capas input	1
Capas output	1
Capas convolucionales	2
Capas de pooling	2
Capas fully conected	2
Número de pesos (weights)	6034470
Números de Bias	231

Número total de parámetros	6034701
Coste de memoria del entrenamiento (MB)	3680
Número de observaciones evaluadas	478
Misclassification error (%)	63,39

El misclassification error alcanzado por este modelo es de un 63,39%, un error bastante elevado. Uno de los resultados que interesa en los modelos de clasificación y a los que aún no se ha hecho ninguna mención es la matriz de confusión de los modelos, en esta se puede observar que clase, de las cinco que quieren clasificar, es mejor clasificada por el modelo. En este caso la matriz de confusión del modelo LeNet5 es la que se muestra en la siguiente tabla. La información recogida en la tabla hace referencia a las imágenes del conjunto de imágenes test, en las columnas se recoge la información de las predicciones realizadas por el modelo frente a las filas que recogen la etiqueta real de la imagen en cuestión. Si se interpreta la primera celda, por ejemplo, esta quiere decir que existe únicamente una imagen que tenía la etiqueta 'Cardboard' y que ha sido clasificada como tal por el modelo. En este caso el modelo clasifica la mayoría de imágenes que tienen una etiqueta 'Cardboard' como 'Paper'.

Tabla 12 Matriz confusión modelo LeNet5

	Predicciones				
Reales	Cardboard	Glass	Metal	Paper	Plastic
Cardboard	1	4	0	70	6
Glass	0	15	0	52	33
Metal	0	13	2	50	17
Paper	0	3	1	101	14
Plastic	0	5	0	35	56

Al tener 5 categorías puede resultar un poco engorroso la interpretación de matrices de confusión por ello lo que se ha buscado con el siguiente tipo de tablas es facilitar esta tarea. En esta tabla se han calculado los porcentajes sobre el total de imágenes de cada categoría en el conjunto entrenamiento. El 86,42% de imágenes con etiqueta 'Carboard' han sido clasificadas como 'Paper' por este modelo. Existe un claro sesgo por este modelo hacia la categoría 'Paper', clasifica la mayoría de imágenes como 'Paper'. El modelo hace una buena clasificación de la categoría 'Plastic' con un 58% de acierto sobre dicha categoría.

Tabla 13 Matriz de confusión porcentajes LeNet5

	Predicciones				
Reales	Cardboard	Glass	Metal	Paper	Plastic
Cardboard	1,23%	4,94%	0,00%	86,42%	7,41%

Glass	0,00%	15,00%	0,00%	52,00%	33,00%
Metal	0,00%	15,85%	2,44%	60,98%	20,73%
Paper	0,00%	2,52%	0,84%	84,87%	11,76%
Plastic	0,00%	5,21%	0,00%	36,46%	58,33%

La siguiente arquitectura que se va a modelar es la **ResNet**, está compuesta por un total de 46 capas, incorpora nuevas tipologías de capas como son las "Batch Normalization Layers" y las capas residuales. Al tener tantas capas en este tipo de arquitectura el número de parámetros se disparan hasta alcanzar la cifra de 10997440.

Tabla 14 Modelo ResNet

	Valor
Learning rate	0,001
Bathc size	32
Epochs	50
Número de capas	46
Capas input	1
Capas output	1
Capas convolucionales	17
Capas de pooling	2
Capas fully conected	0
Capas batch normalization	17
Capas residuales	8
Número de pesos (weights)	10997440
Números de Bias	6917
Número total de parámetros	11004357
Coste de memoria del entrenamiento (MB)	24119
Número de observaciones evaluadas	478
Misclassification error (%)	43,10

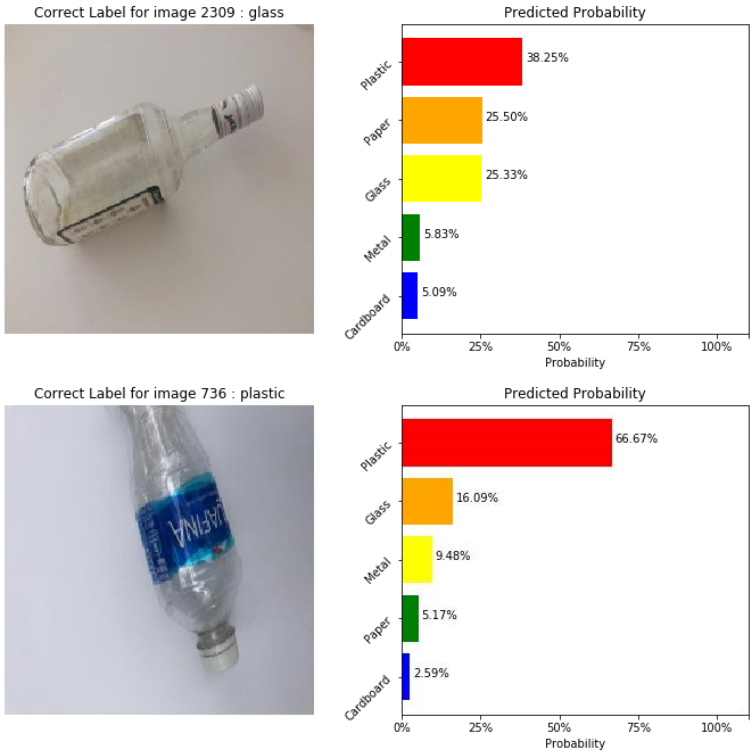
En esta red se alcanza un misclassification error de 43,10% lo cual significa una mejora, pasando el umbral del 50%, la red neuronal ya empieza a distinguir patrones en las imágenes. Esto se ve reflejado además en la matriz de confusión, siguiendo el esquema presentado anteriormente, la tabla muestra la diagonal en color verde. En todas las categorías, menos en 'Metal', más del 50% de las imágenes del conjunto de entrenamiento son clasificadas por la red con la etiqueta correcta. Cabe destacar que un 25% de las imágenes pertenecientes a la categoría 'Glass' se clasifican por el modelo como 'Plastic', esto tiene sentido debido a la similitud existente entre botellas de plástico y botellas de vidrio. Este error no se comete en la dirección contraria, los residuos de plástico son mal clasificados por la red con mayor frecuencia hacia la categoría 'Paper'.

Tabla 15 Matriz confusión porcentajes ResNet

	Predicciones				
Reales	Cardboard	Glass	Metal	Paper	Plastic
Cardboard	70,37%	6,17%	4,94%	12,35%	6,17%
Glass	2,00%	57,00%	5,00%	11,00%	25,00%
Metal	3,66%	18,29%	42,68%	28,05%	7,32%
Paper	1,68%	21,85%	9,24%	52,10%	15,13%
Plastic	7,95%	5,68%	3,41%	13,64%	69,32%

El modelo ResNet tiene cierto sesgo a la hora de clasificar las imágenes etiquetadas como "Glass", la razón la podemos observar en la siguiente imagen. Son dos imágenes del conjunto de entrenamiento, los problemas que encuentra la red a la hora de clasificar se deben al parecido que guardan las botellas de plástico. Al final el modelo busca patrones en las imágenes y por lo tanto distinguir dos formas tan parecidas, como pueden ser botellas, es complicado.

Gráfico 8 ResNet Glass vs Plastic



El siguiente modelo utilizará **la arquitectura VGG16**, el total de parámetros a calcular en esta red es de 134281029, esto conllevará un gran coste computacional. En total el modelo cuenta con 22 capas, 13 de ellas son capas convolucionales de ahí que exista un número tan elevado de parámetros a calcular.

Tabla 16 Modelo VGG16

	Valor
Learning rate	0,001
Bathc size	32
Epochs	50
Número de capas	22
Capas input	1
Capas output	1
Capas convolucionales	13
Capas de pooling	5
Capas fully conected	2
Número de pesos (weights)	134268608
Números de Bias	12421
Número total de parámetros	134281029
Coste de memoria del entrenamiento (MB)	73833
Número de observaciones evaluadas	478
Misclassification error (%)	54,18

Desde el punto de vista de resultados se obtiene un misclassification rate del 54,18% en el conjunto de datos de entrenamiento. En la arquitectura VGG16 pasa exactamente lo contrario que en el modelo ResNet, en este caso las imágenes pertenecientes a plásticos son mal calcificadas por la red en su mayoría debido a que las confunde con cristal.

Tabla 17 Matriz de confusión porcentajes VGG16

	Predicciones				
Reales	Cardboard	Glass	Metal	Paper	Plastic
Cardboard	71,60%	4,94%	2,47%	14,81%	6,17%
Glass	16,00%	41,00%	14,00%	20,00%	9,00%
Metal	14,63%	21,95%	30,49%	31,71%	1,22%
Paper	6,72%	7,56%	2,52%	55,46%	27,73%
Plastic	11,46%	42,71%	4,17%	11,46%	30,21%

El siguiente modelo utilizará la arquitectura **InceptionV3**, esta arquitectura se caracteriza por tener un número elevado de capas, un total de 219 las cuales generan 21795813 parámetros que debe calcular la red.

Tabla 18 Modelo InceptionV3

	Valor
Learning rate	0,001
Bathc size	32

Epochs	50
Número de capas	22
Capas input	1
Capas output	1
Capas convolucionales	13
Capas de pooling	5
Capas fully conected	2
Número de pesos (weights)	134268608
Números de Bias	12421
Número total de parámetros	134281029
Coste de memoria del entrenamiento (MB)	73833
Número de observaciones evaluadas	478
Misclassification error (%)	46,65

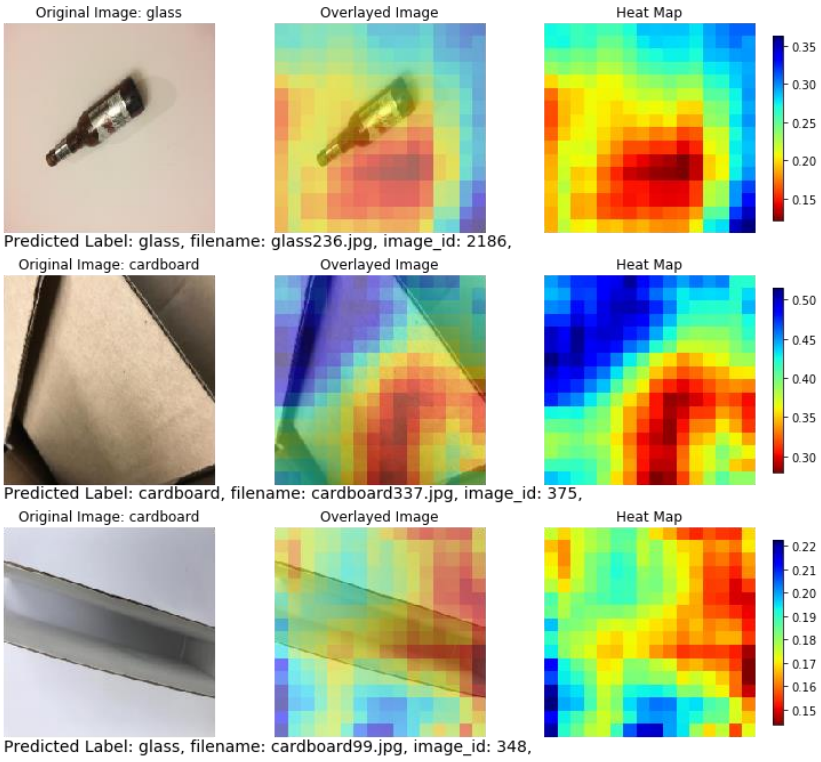
El misclassification alcanzado por este modelo es de 46,65%, supera por lo tanto el 50% de accuracy. Se puede observar la siguiente tabla que el mayor problema que encuentra esta red a la hora de clasificar el conjunto de entrenamiento son las imágenes pertenecientes a la categoría 'Meta'. Clasifica un 59,68% de las imágenes pertenecientes a residuos metálicos como 'Glass'.

Tabla 19 Matrix de confusión porcentajes InceptionV3

	Predicciones				
Reales	Cardboard	Glass	Metal	Paper	Plastic
Cardboard	66,67%	12,35%	6,17%	7,41%	7,41%
Glass	2,00%	64,00%	6,00%	7,00%	21,00%
Metal	1,61%	59,68%	3,23%	25,81%	9,68%
Paper	1,68%	18,49%	10,08%	46,22%	23,53%
Plastic	3,13%	21,88%	4,17%	13,54%	57,29%

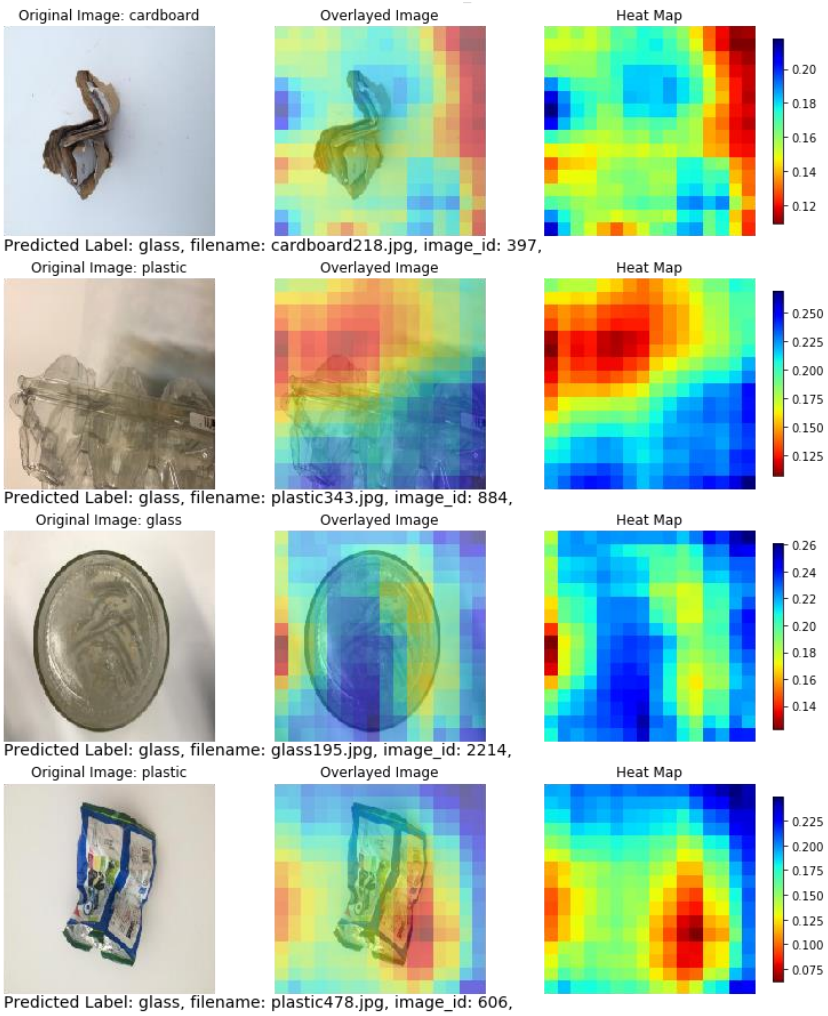
Para poder evaluar los resultados del modelo se pueden utilizar mapas de calor sobre las imágenes clasificadas, los mapas vienen a decir en que pixeles de la imagen se ha fijado la Red Neuronal para clasificar una imagen concreta. En la siguiente imagen se puede observar unos ejemplos de este típico de gráficos para imágenes del conjunto de imágenes que han sido bien clasificadas. La red neuronal se está fijando en la parte sombreada con el color rojo, con la transición de gráficos se puede ver claramente la parte de la imagen en la que se está centrando la red.

Imagen 24 Mapas de calor correctas IncepcionV3



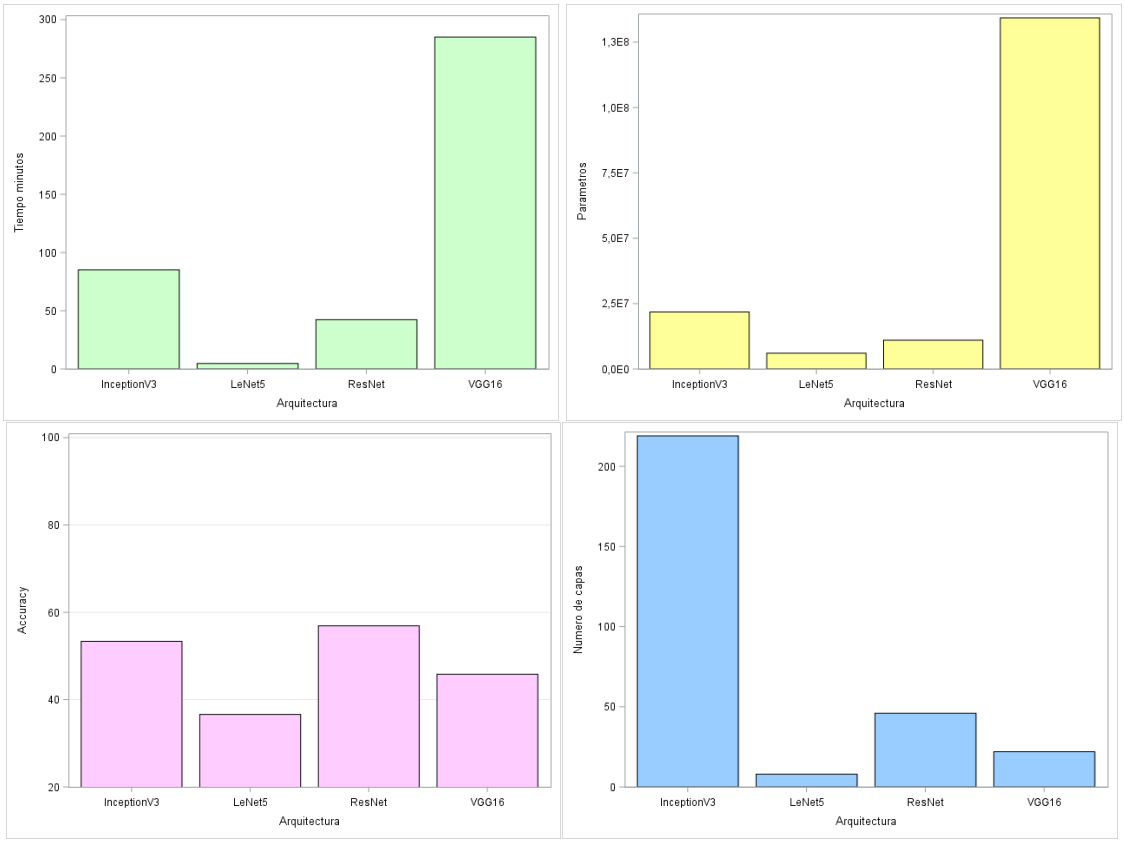
Fijándose en los mapas de calor de las imágenes mal clasificadas se puede llegar a alguna conclusión sobre la razón del fallo de la Red. La Red Neuronal se está fijando incorrectamente en las imágenes, esquinas, partes parciales del residuo o está reconociendo patrones que le hacen confundirse. Por ejemplo, en el caso del plástico mal clasificado puede que la transparencia le esté haciendo confundir el residuo de plástico con un cristal (fila 2 de imágenes).

Imagen 25 Mapas de calor incorrectas IncepcionV3



Un resumen de los modelos aplicados con arquitecturas predefinidas se puede observar en el siguiente conjunto de gráficos. La arquitectura VGG16 es la que toma mayores tiempos de ejecución, está claro que esto se debe al gran número de parámetros que debe calcular dicha red, sin embargo, no alcanza los mejores niveles de accuracy frente a las otras arquitecturas. La arquitectura LeNet5 presenta el menor tiempo de ejecución, esto se debe a que es la arquitectura con menos capas y parámetros a calcular, esto se ve reflejado en los resultados con el menor accuracy alcanzado. La arquitectura ResNet completa el proceso de entrenamiento relativamente rápido, tiene un menor número de parámetros a calcular y obtiene los mejores resultados en términos de accuracy. Por último, la red InceptionV3 obtiene resultados adecuados y los parámetros a calcular no son muy elevados pese al gran número de capas por las que está compuesta.

Gráfico 9 Comparación de diferentes arquitecturas



Los resultados obtenidos por estas arquitecturas predefinidas mejoran a los modelos creados de forma manual, de este conjunto de arquitecturas si se tuviera que elegir una ganadora sería las ResNet, aunque la inceptionV3 se encuentra cerca en términos de accuracy, tanto los tiempos de ejecución como el número de parámetros hacen ganadora a la ResNet.

4.5 Modelos con pesos pre entrenados.

Hasta ahora todos los modelos aplicados partían de unos pesos aleatorios, los modelos con pesos pre entrenados surgen para optimizar los modelos. La idea es que el modelo no inicie los pesos de forma aleatoria, se importan unos pesos ya entrenados en otro problema de clasificación, el proceso de entrenamiento se lleva con el conjunto de datos a clasificar para que la red tome las características de las imágenes. El resultado son modelos con mejores resultados y menores tiempos de ejecución. El proceso de creación de un modelo pre entrenado a través de código es el siguiente. La función VGG19 tiene como parámetros importantes '*pre_trained_weights'*, establece que en la Red Neuronal se van a utilizar pesos pre entrenados, '*pre_trained_weights_file'* que apunta hacia la ruta donde están guardados los pesos pre entrenados en un archivo tipo .h5 y por último '*include_top'* especifica si se incluirán los pesos pre entrenados de las capas superiores de la red, es decir de las fully conected layers.

```
model_vgg19 = VGG19(conn, model_table='VGG19_notop', n_classes=5, n_channels=3,
width=224,height=224,scale=1,offsets=tr_img.channel_means,pre_trained_weights=True,
pre_trained_weights_file="/data/imageAnalitics/recycledvgg19/VGG_ILSVRC_19_layers.caffe
model.h5" include_top=False)
```

En la siguiente tabla se presenta un modelo **VGG16** con pesos pre entrenados, en este caso el proceso de entrenamiento se ha configurado con 0,01 *Learning rate*, un 32 *Bathc size* y un total de 15 *Epochs*. En este caso el misclassification alcanzado es del 32,85% lo cual mejora bastante los resultados obtenidos hasta el momento.

Tabla 20 VGG16 con pesos

	Valor
Learning rate	0,01
Bathc size	32
Epochs	15
Número total de parámetros	134281029
Coste de memoria del entrenamiento (MB)	73833
Número de observaciones evaluadas	478
Misclassification error (%)	32,85

Este modelo clasifica muy bien tanto la categoría 'Cardboard' como 'Paper', superando en ambas el 90% de acierto, la categoría 'Paper' alcanza una buena clasificación del 84,87%. En las categorías 'Glass' y 'Plastic' es donde la red presenta mayores problemas para clasificar correctamente las imágenes. En ambos casos la mayoría de imágenes son clasificadas como 'Metal' en vez de su verdadera clase.

Tabla 21 Matriz de confusión porcentajes pre entrenada VGG16

	Predicciones				
Reales	Cardboard	Glass	Metal	Paper	Plastic
Cardboard	93,83%	0,00%	0,00%	6,17%	0,00%
Glass	8,00%	44,00%	45,00%	3,00%	0,00%
Metal	3,66%	0,00%	92,68%	3,66%	0,00%
Paper	14,29%	0,00%	0,84%	84,87%	0,00%
Plastic	8,33%	7,29%	46,88%	12,50%	25,00%

Siguiendo los conocimientos desarrollados durante el presente trabajo, se opta por aumentar el número de *Epochs* aplicados al modelo **VGG16**, estableciendo 25. El modelo mejora sustancialmente ante este único cambio alcanzando un misclassification error de 18,41% esto quiere decir que el modelo clasifica bien un 82% del conjunto de entrenamiento.

Tabla 22 VGG16 con pesos y 25 epochs

	Valor
Learning rate	0,01
Bathc size	32
Epochs	25
Número total de parámetros	134281029
Coste de memoria del entrenamiento (MB)	73833
Número de observaciones evaluadas	478
Misclassification error (%)	18,41

La tabla de acierto sobre la matriz de confusión nos muestra una diagonal verde, esto es lo que se buscaba, un modelo que fuera capaz de obtener altos porcentajes de acierto sobre el conjunto de imágenes test. En este caso los porcentajes de acierto según categoría son: 'Cardboard' 90,12%, 'Glass' 54%, 'Metal' 74,39%, 'Paper' 96,64% y 'Plastic' 90,63%.

Tabla 23 Matriz de confusión porcentajes pre entrenados VGG16

	Predicciones				
Reales	Cardboard	Glass	Metal	Paper	Plastic
Cardboard	90,12%	0,00%	0,00%	9,88%	0,00%
Glass	3,00%	54,00%	7,00%	1,00%	35,00%
Metal	6,10%	2,44%	74,39%	1,22%	15,85%
Paper	2,52%	0,00%	0,00%	96,64%	0,84%
Plastic	0,00%	1,04%	1,04%	7,29%	90,63%

El siguiente modelo pre entrenado a probar es el **VGG19**, este modelo ha obtenido un misclassification error del 21,13%, es un resultado muy bueno. El número de parámetros a calcular es mayor que en el caso de la arquitectura VGG16.

Tabla 24 VGG19 con pesos

	Valor
Learning rate	0,01
Bathc size	32
Epochs	15
Número total de parámetros	139590725
Coste de memoria del entrenamiento (MB)	79384
Número de observaciones evaluadas	478
Misclassification error (%)	21,13

En la matriz de confusión se puede observar que este modelo presenta problemas a la hora de clasificar la categoría 'Plastic' y la categoría 'Metal'.

Tabla 25 Matiz de confusión porcentajes VGG19

	Predicciones				
Reales	Cardboard	Glass	Metal	Paper	Plastic
Cardboard	88,89%	0,00%	0,00%	7,41%	3,70%
Glass	1,00%	77,00%	8,00%	0,00%	14,00%
Metal	1,22%	26,83%	59,76%	1,22%	10,98%
Paper	4,20%	0,00%	0,84%	94,96%	0,00%
Plastic	0,00%	18,75%	4,17%	8,33%	68,75%

La arquitectura **ResNet** es considerada por los investigadores como las más novedosa para problemas de clasificación de imágenes. El primer modelo a probar será ResNet 50. En este modelo se obtienen los mejores resultados hasta ahora, alcanza un misclassification del 9,62%.

Tabla 26 ResNet50 con pesos

	Valor
Learning rate	0,01
Bathc size	1
Epochs	5
Número total de parámetros	23518341
Coste de memoria del entrenamiento (MB)	5212
Número de observaciones evaluadas	478
Misclassification error (%)	9,62

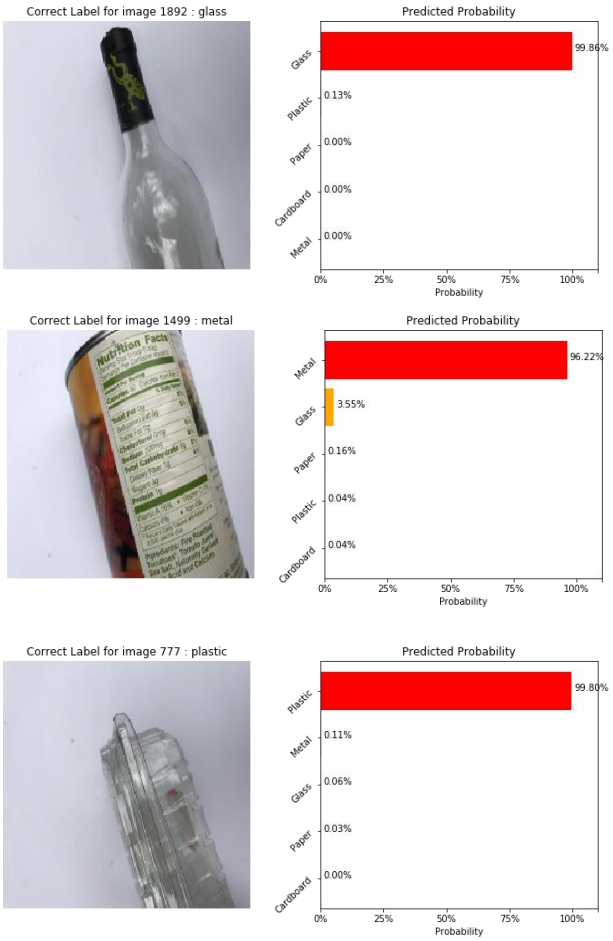
Los porcentajes de buena clasificación de este modelo son muy buenos, destacando la clase 'Metal' que es la que peor clasifica con un 85,37% y la clase 'Paper' es la mejor clasificada con un 93,28%.

Tabla 27 Matriz de confusión con porcentajes ResNet50

	Predicciones				
Reales	Cardboard	Glass	Metal	Paper	Plastic
Cardboard	91,36%	0,00%	1,23%	7,41%	0,00%
Glass	0,00%	91,00%	4,00%	1,00%	4,00%
Metal	1,22%	8,54%	85,37%	0,00%	4,88%
Paper	4,20%	0,00%	1,68%	93,28%	0,84%
Plastic	0,00%	4,17%	2,08%	4,17%	89,58%

En la siguiente imagen se puede observar la capacidad de la red para clasificar las diferentes imágenes. Los porcentajes estimados por la red para cada imagen tienen un nivel de acierto muy elevado.

Gráfico 10 Modelo ResNet 50 pre entrenado



Por último, se probará el modelo **ResNet101**, esta red obtiene unos resultados de misclassification realmente buenos alcanzando un 6,28%. La red clasifica mal únicamente un 6,28% de las imágenes del conjunto de test. Estos resultados son realmente buenos teniendo en cuenta que son 5 categorías a clasificar.

Tabla 28 ResNet101 con pesos

	Valor
Learning rate	0,01
Bathc size	1
Epochs	5
Número total de parámetros	42510469

Coste de memoria del entrenamiento (MB)	8287
Número de observaciones evaluadas	478
Misclassification error (%)	6,28

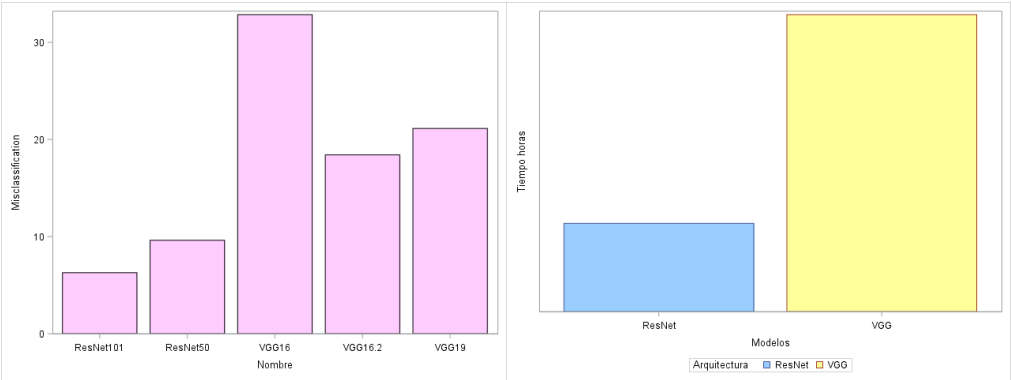
En la matriz de confusión se pude observar como la diagonal supera en todas las categorías un acierto del 90%.

Tabla 29 Matiz de confusión porcentajes ResNet101

	Predicciones				
Reales	Cardboard	Glass	Metal	Paper	Plastic
Cardboard	100,00%	0,00%	0,00%	0,00%	0,00%
Glass	0,00%	91,92%	4,04%	0,00%	4,04%
Metal	1,22%	1,22%	93,90%	0,00%	3,66%
Paper	2,52%	0,00%	1,68%	95,80%	0,00%
Plastic	0,00%	3,13%	1,04%	5,21%	90,63%

Si comparamos los resultados de estas dos arquitecturas con pesos pre entrenados, la arquitectura ResNet obtiene errores más bajos. Así como, los tiempos de ejecución de las arquitecturas ResNet son muy bajos. Se puede concluir por lo tanto que la arquitectura ResNet es la que mejor se ajusta al problema de clasificación planteado.

Gráfico 11 Comparando modelos pre entrenados



5 Principales resultados y conclusiones

Las Redes Neuronales surgen como alternativa a los modelos estadísticos clásicos, pese a que estas cuentan con mucho más parámetros y cierta complejidad, tienen ciertas ventajas para determinadas tareas que no se pueden llevar a cabo con modelos clásicos. Los problemas relacionados con las imágenes se pueden abordar con este tipo de algoritmos obteniendo muy buenos resultados. En el caso del presente trabajo se ha planteado un problema de clasificación de imágenes, para ello se han utilizado una tipología de Redes Neuronales llamadas Redes Neuronales Convolucionales. Las Redes Neuronales Convolucionales presenta la ventaja de mantener la espacialidad de los pixeles, es decir que tienen en cuenta la proximidad entre pixeles, esto no era posible en Redes Neuronales normales.

El procesamiento de imágenes en este tipo de modelos requiere de una gran capacidad computacional, en este caso se ha trabajado sobre CPU, aunque la opción más preferible es trabajar sobre GPU. La GPU da una mayor capacidad de procesamiento de la información y hace más ágil y facilita los cálculos matemáticos. Los tiempos de entrenamiento dependerán en gran medida, a parte de lo relacionado con la parte de hardware, de las características del modelo que se quiere entrenar. El número de parámetros incidirá directamente en los tiempos de entrenamiento, así como, los propios parámetros que se establezcan en el proceso de entrenamiento *Batch size, Epochs, Learning rate*…

En el análisis de imágenes tiene una gran importancia el conjunto de datos elegidos, la calidad de las imágenes es importante, pues facilitará el proceso de aprendizaje de la red. Pre procesar las imágenes y transformarlas dará la oportunidad de construir una Red más robusta ante nuevas observaciones. En el caso concreto de este trabajo se ha observado que un aumento del conjunto de datos de entrenamiento no incide positivamente sobre los resultados obtenidos. Es importante también tratar las imágenes aplicando *resize* y limpiando errores para facilitar el trabajo de la Red.

En cuanto a los modelos aplicados, destacar que los mejores resultados han sido obtenidos por las Redes Neuronales con pesos pre entrenados. La existencia de arquitecturas predefinidas en la literatura científica, así como, los pesos pre entrenados son fundamental para el desarrollo de modelos de este tipo. Sin este tipo de investigaciones que son públicas y disponibles sería muy difícil llegar a resultados tan buenos en problemas de este tipo.

Los principales resultados de los modelos se representan en la siguiente tabla, el avance hasta encontrar una red que se ajustase al problema de clasificación ha sido progresivo. Un aumento del conjunto de entrenamiento no supone mejoras significativas sobre los resultados, en cambio, el aumento de las *Epochs* en el proceso de entrenamiento sí que influye más que significativamente sobre la mejora de los resultados. El modelo con mejores resultados ha sido el que utiliza pesos pre entrenados, alcanza un accuracy del 93,72%. En definitiva, la arquitectura que mejor se adapta al problema planteado es la ResNet, tanto en el apartado de arquitecturas como en la comparación con otros modelos pre entrenados, esta arquitectura resulta ganadora en cuanto a exactitud en sus predicciones como en coste computacional.

Tabla 30 Resumen de modelos

Nombre	Aumento_de_datos	Pesos pre entrenados	Epochs	Accuracy
Secuencial3	No	No	50	20,08
Secuencial3	Si	No	30	21,95
Secuencial3	Si	No	30	23,47
Secuencial1	No	No	50	24,89
Secuencial2	Si	No	30	24,94
Secuencial1	Si	No	30	25,86
Secuencial2	No	No	50	25,95
Secuencial1	Si	No	30	26,95
Secuencial2	Si	No	30	27,60
Secuencial3.1	Si	No	200	45,61
Secuencial3.1	Si	No	30	50,20
Secuencial3.1	No	No	200	52,72
LeNet5	No	No	50	36,61
VGG16	No	No	50	45,82
InceptionV3	No	No	50	53,35
ResNet	No	No	50	56,9
VGG16	No	Si	15	67,15
VGG19	No	Si	15	78,87
VGG16.2	No	Si	25	81,59
ResNet50	No	Si	5	90,38
ResNet101	No	Si	5	93,72

El trabajo presenta diversas limitaciones y ampliaciones que se podrán llevar a cabo en futuros proyectos. Comprobar que pasaría si ejecutamos los mismos algoritmos sobre GPU en vez de CPU y observar si existen diferencias. Sería interesante intentar diferenciar las clases más conflictivas en modelos independientes, se ha observado que por lo general a la red le cuesta distinguir entre la clase 'Plastic' y 'Glass', se podría estudiar este problema por separado y ver que soluciones se pueden plantear.

Estudiar la posible aplicación de este tipo de modelos en la vida real puede ser una idea interesante. Para ello se debe conocer como es el proceso completo de una cadena de reciclaje e integrar los modelos para facilitar el procesado de residuos. Poder llegar a poner en producción un modelo de este tipo de modelos se necesitaría una parte *Backend* que permitiera ejecutar el modelo y guardar los pesos del modelo, esta parte debería ser de tomar una foto cualquiera, aplicarle el preprocesamiento necesario y obtener una predicción de la misma. Esto en el *frontend* se traduce en una aplicación web que te permita subir una imagen y obtener un gráfico tipo a los presentados en el trabajo con las probabilidades estimadas para dicha imagen.

En definitiva, las Redes Neuronales Convolucionales han servido para resolver el problema de clasificación planteado. Es posible clasificar residuos obteniendo niveles de error muy bajos, este tipo de modelos se pueden aplicar a la vida real y obtener grandes beneficios de ellos.

Blibliografía

Amick, D. S. (2014). Reflection on the Origins of Recycling: a Paleolithic Perspective. *Lithic Technology*, *39*(1), 64–69. https://doi.org/10.1179/0197726113z.00000000025

Bircanoğlu, C., Atay, M., Beser, F., & Ayyuce, M. (2018). RecycleNet: Intelligent Waste Sorting Using Deep Nerual Networks. *Innovations in Intelligent Systems and Applications (INISTA)*, 1–7.

Espinar, R. L. (2018). *Modelos de Clasificacion con datos no balanceados*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Lara González, J. D. (2008). Reducir,Reutilizar,Reciclar. *Elementos 69*, 45–48.

Larrañaga, P., Inza, I., & Moujahid, A. (n.d.). *Tema 8 Redes Neuronales*.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proccedings of the IEEE*, *86*, 2278–2324. Retrieved from http://ieeexplore.ieee.org/document/726791/#full-text-section

Monien, B., Preis, R., & Schamberger, S. (2007). Approximation algorithms for multilevel graph partitioning. *Handbook of Approximation Algorithms and Metaheuristics*, 60-1-60–16. https://doi.org/10.1201/9781420010749

Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 1–14. Retrieved from http://arxiv.org/abs/1409.1556

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07-12-June-2015*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Yang, M., & Thung, G. (2016). *Classification of Trash for Recyclability Status*. 1–6. https://doi.org/10.1145/2971648.2971731

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8689 LNCS*(PART 1), 818–833. https://doi.org/10.1007/978-3-319-10590-1_53

Webgrafía

https://dej.rae.es/lema/reciclaje

https://es.wikipedia.org/wiki/Regla_de_las_tres_erres

https://www.youtube.com/watch?v=uwbHOpp9xkc

https://www.youtube.com/watch?v=MRIv2IwFTPg

https://www.youtube.com/watch?v=aircAruvnKk&t=698s

https://www.youtube.com/watch?v=IHZwWFHWa-w&t=1009s

https://medium.com/neuron4/redes-pre-entrenadas-en-imagenet-30d858c37b1f

https://documentation.sas.com/?docsetId=casdlpg&docsetTarget=p0994iow7i7oqwn1vsk7xv2w4dz1.htm&docsetVersion=8.3&locale=en#p1f92rrri2bbpwn1j2trik2k1lu8

https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

https://www.researchgate.net/figure/AlexNet-like-architecture_fig4_320723863

https://www.researchgate.net/figure/Figura-46-La-arquitectura-LeNet-consiste-en-dos-capas-convolucionales-capas-de_fig6_329453166

Anexos

Anexo 1 Códigos jupyter notebook

```
/****************** Modelos secuenciales ******************/

#!/usr/bin/env python

# coding: utf-8

# In[2]:

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

import swat

# In[3]:

# Create CAS Connection

# In[4]:

from dlpy.images import ImageTable

# In[5]

my_images = ImageTable.load_files(conn, path=img_path)

my_images.head()

my_images.show(nimages=5, ncol=8, randomize=False, figsize=None)

my_images.image_summary

my_images.resize(width = 224, height=224)

from dlpy.splitting import two_way_split

tr_img, te_img = two_way_split(my_images, test_rate=20, seed=123)

my_images.label_freq

from dlpy import Sequential

from dlpy.model import *

from dlpy.layers import *

from dlpy.applications import *

model_secuencial1 = Sequential(conn, model_table='Simple_CNN')

model_secuencial1.add(InputLayer(3, 224, 224, offsets=tr_img.channel_means))

model_secuencial1.add(Conv2d(8, 7))

model_secuencial1.add(Pooling(4))

model_secuencial1.add(Dense(8))
```

```
model_secuencial1.add(OutputLayer(act='softmax', n=5))

model_secuencial1.print_summary()

model_secuencial1.fit(data=tr_img,

        mini_batch_size=64,

        max_epochs=50,

        lr=0.01)



model_secuencial1.plot_training_history(fig_size=(15, 6))

model_secuencial1.evaluate(tr_img)

model_secuencial1.evaluate(te_img)

model_secuencial1.plot_evaluate_res(img_type = 'C', randomize = True , n_images = 20)

test_result = model_secuencial1.predict(te_img)

test_result_table = model_secuencial1.valid_res_tbl

test_result_table.head()

display(confusion_matrix(test_result_table['target_class'], test_result_table['I_target_class']))

import numpy as np

import matplotlib.pylab as plt

from dlpy.metrics import (accuracy_score, confusion_matrix, plot_roc, plot_precision_recall,
roc_auc_score, f1_score)

get_ipython().run_line_magic('matplotlib', 'inline')

plot_roc(test_result_table['target_class'], test_result_table['P_target_class'], pos_label=1,
figsize=(6,6), liewidth=2)

model_secuencial2 = Sequential(conn, model_table='Simple_CNN')

model_secuencial2.add(InputLayer(3, 224, 224, offsets=tr_img.channel_means))

model_secuencial2.add(Conv2d(8, 7))

model_secuencial2.add(Pooling(4))

model_secuencial2.add(Conv2d(8, 7))

model_secuencial2.add(Pooling(4))

model_secuencial2.add(Dense(16))

model_secuencial2.add(OutputLayer(act='softmax', n=5))

model_secuencial2.print_summary()
```

```
model_secuencial2.fit(data=tr_img,
        mini_batch_size=64,
        max_epochs=50,
        lr=0.01)


model_secuencial2.plot_training_history(fig_size=(15, 6))
model_secuencial2.evaluate(tr_img)
model_secuencial2.evaluate(te_img)
model_secuencial2.plot_evaluate_res(img_type = 'C', randomize = True , n_images = 20)
model_secuencial3 = Sequential(conn, model_table='Simple')
model_secuencial3.add(InputLayer(3, 224, 224, offsets=tr_img.channel_means))
model_secuencial3.add(Conv2d(8, 7))
model_secuencial3.add(Pooling(4))
model_secuencial3.add(Conv2d(8, 7))
model_secuencial3.add(Pooling(4))
model_secuencial3.add(Conv2d(8, 7))
model_secuencial3.add(Pooling(4))
model_secuencial3.add(Dense(16))
model_secuencial3.add(OutputLayer(act='softmax', n=5))
model_secuencial3.print_summary()
model_secuencial3.fit(data=tr_img,
        mini_batch_size=64,
        max_epochs=50,
        lr=0.01)


model_secuencial3.plot_training_history(fig_size=(15, 6))
model_secuencial3.evaluate(tr_img)
model_secuencial3.evaluate(te_img)
model_secuencial3.plot_evaluate_res(img_type = 'C', randomize = True , n_images = 20)
```

```
model_secuencial3_entre = Sequential(conn, model_table='Simple')
model_secuencial3_entre.add(InputLayer(3, 224, 224, offsets=tr_img.channel_means))
model_secuencial3_entre.add(Conv2d(8, 7))
model_secuencial3_entre.add(Pooling(4))
model_secuencial3_entre.add(Conv2d(8, 7))
model_secuencial3_entre.add(Pooling(4))
model_secuencial3_entre.add(Conv2d(8, 7))
model_secuencial3_entre.add(Pooling(4))
model_secuencial3_entre.add(Dense(16))
model_secuencial3_entre.add(OutputLayer(act='softmax', n=5))
model_secuencial3_entre.print_summary()
model_secuencial3_entre.fit(data=tr_img,
        mini_batch_size=32,
        max_epochs=200,
        lr=0.001)


model_secuencial3_entre.plot_training_history(fig_size=(15, 6))
model_secuencial3_entre.evaluate(tr_img)
model_secuencial3_entre.evaluate(te_img)
model_secuencial3_entre.plot_evaluate_res(img_type = 'C', randomize = True , n_images = 20)
model_secuencial3_entre.plot_evaluate_res(img_type = 'M', randomize = True , n_images = 20)
model_secuencial3_entre.valid_conf_mat

/****************** Modelos con aqruitecturas ******************/

# importamos los paquetes necesarios #
import swat
import dlpy
from dlpy.images import ImageTable
from dlpy.splitting import two_way_split
```

```
from dlpy.applications import *

from dlpy.utils import add_caslib

import warnings

warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt

get_ipython().run_line_magic('matplotlib', 'inline')

modelo_lenet5=dlpy.applications.LeNet5(conn, model_table='LENET5', n_classes=5, n_channels=3, width=224, height=224)

modelo_lenet5.fit(data=tr_img,

        mini_batch_size=32,

        max_epochs=50,

        lr=0.001)

modelo_lenet5.evaluate(data=te_img)

modelo_lenet5.plot_evaluate_res(img_type='C', randomize=True, n_images=20)

modelo_lenet5.plot_evaluate_res(img_type='A', randomize=True, n_images=20)

modelo_lenet5.plot_evaluate_res(img_type='M', randomize=True, n_images=20)

modelo_lenet5.heat_map_analysis(data=None, mask_width=None, mask_height=None, step_size=None, display=True, img_type='A', image_id=None, filename=None, inputs='_image_', target='_label_', max_display=10)

modelo_lenet5.count_params()

modelo_lenet5.valid_conf_mat

import numpy as np

import matplotlib.pylab as plt

from dlpy.metrics import (accuracy_score, confusion_matrix, plot_roc, plot_precision_recall, roc_auc_score, f1_score)

get_ipython().run_line_magic('matplotlib', 'inline')

t_result = modelo_lenet5.predict(te_img)

test_result_table = modelo_lenet5.valid_res_tbl

test_result_table.head()

display(confusion_matrix(test_result_table['_label_'], test_result_table['I__label_']))

acc_score = accuracy_score(test_result_table['_label_'], test_result_table['I__label_'])

print ('the accuracy score is {:.6f}'.format(acc_score))
```

```
plot_roc(test_result_table['_label_'], test_result_table['_DL_PredLevel_'], pos_label=1, figsize=(6,6), linewidth=2)

dlpy.metrics.plot_roc( test_result_table['_label_'], test_result_table['_DL_PredLevel_'], pos_label=1, castable=test_result_table, cutstep=0.001, figsize=(8, 8), fontsize_spec=None, linewidth=1, id_vars=None)

modelo_resnet=dlpy.applications.ResNet18_SAS(conn, model_table='RESNET18_SAS', n_classes=5, n_channels=3, width=224, height=224, scale=1, random_flip='none', random_crop='none')

modelo_resnet.fit(data=tr_img,

        mini_batch_size=32,

        max_epochs=50,

        lr=0.001)

modelo_resnet.evaluate(data=te_img)

modelo_resnet.plot_evaluate_res(img_type='A', randomize=True , n_images=20 )

modelo_resnet.plot_evaluate_res(img_type='M', randomize=True , n_images=20 )

modelo_resnet.plot_evaluate_res(img_type='C', randomize=True , n_images=20 )

modelo_resnet.heat_map_analysis(data=None, mask_width=None, mask_height=None, step_size=None, display=True, img_type='A', image_id=None, filename=None, inputs='_image_', target='_label_', max_display=10)

modelo_resnet.count_params()

modelo_resnet.valid_conf_mat

modelovgg = dlpy.applications.VGG16(conn, model_table='VGG16', n_classes=5, n_channels=2, width=224, height=224)

modelovgg.fit(data=tr_img,

        mini_batch_size=32,

        max_epochs=50,

        lr=0.001)

modelovgg.evaluate(data=te_img)

modelovgg.plot_evaluate_res(img_type='M', randomize=True, n_images=20)

modelovgg.plot_evaluate_res(img_type='C', randomize=True, n_images=20)

#METODO 2: Gráficos de evaluación #

modelovgg.plot_evaluate_res(img_type='A', randomize=True, n_images=20)

modelovgg.heat_map_analysis(data=None, mask_width=None, mask_height=None, step_size=None, display=True, img_type='M', image_id=None, filename=None, inputs='_image_', target='_label_', max_display=10)
```

```
modelovgg.count_params()

modelovgg.valid_conf_mat

modelovgg19 = dlpy.applications.VGG19(conn, model_table='VGG19', n_classes=5, n_channels=2, width=224, height=224)

modelovgg19.fit(data=tr_img,

        mini_batch_size=32,

        max_epochs=50,

        lr=0.001)

modelovgg19.evaluate(data=te_img)

modelovgg19.plot_evaluate_res(img_type='M', randomize=True, n_images=20)

modelovgg19.plot_evaluate_res(img_type='C', randomize=True, n_images=20)

modelovgg19.plot_evaluate_res(img_type='A', randomize=True, n_images=20)

modelovgg19.heat_map_analysis(data=None, mask_width=None, mask_height=None, step_size=None, display=True, img_type='M', image_id=None, filename=None, inputs='_image_', target='_label_', max_display=10)

modelovgg19.valid_conf_mat

modelo_InceptionV3 =dlpy.applications.InceptionV3(conn, model_table='InceptionV3', n_classes=5, n_channels=3, width=224, height=224)

modelo_InceptionV3.fit(data=tr_img,

        mini_batch_size=32,

        max_epochs=50,

        lr=0.001)

modelo_InceptionV3.evaluate(data=te_img)

modelo_InceptionV3.plot_evaluate_res(img_type='A', randomize=True, n_images=20)

modelo_InceptionV3.plot_evaluate_res(img_type='C', randomize=True, n_images=20)

modelo_InceptionV3.plot_evaluate_res(img_type='M', randomize=True, n_images=20)

modelo_InceptionV3.heat_map_analysis(data=None, mask_width=None, mask_height=None, step_size=None, display=True, img_type='A', image_id=None, filename=None, inputs='_image_', target='_label_', max_display=10)

modelo_InceptionV3.valid_conf_mat



# In[ ]:
```

```
/****************** Modelos con pre entrenados ******************/
model_vgg16 = VGG16(
   conn,
   model_table='VGG16_notop',
   n_classes=5,
   n_channels=3,
   width=224,
   height=224,
   scale=1,
   offsets=tr_img.channel_means,
   pre_trained_weights=True,
   pre_trained_weights_file='',
   include_top=False)
# Set Optimization and Algorithm Paramters
model_vgg16.fit(data=tr_img,
            mini_batch_size=32,
            max_epochs=15,
            lr=0.01,
            log_level=2);
model_vgg16.evaluate(te_img)
conn.save (table='VGG16_notop_weights',
        name='recycled_VGG16_weights',
        replace=True);
model_vgg16.plot_training_history(fig_size=(15, 6))
model_vgg16.plot_evaluate_res(img_type = 'C', randomize = True , n_images = 20)
model_vgg16.plot_evaluate_res(img_type = 'M', randomize = True , n_images = 20)
model_vgg16.heat_map_analysis(mask_width=56, mask_height=56, step_size=8, display=3)
model_vgg16.get_feature_maps(data = te_img, label='plastic', idx=2)
model_vgg16.feature_maps.display(layer_id=0)
model_vgg16.feature_maps.display(layer_id=2)
model_vgg16.feature_maps.display(layer_id=3)
```

model_vgg16.valid_conf_mat

model_vgg16.training_histor

Anexo 2 Gráficos realizados en SAS 9.4

/********** Macro para poder usar RGB en gráficos **********/

```
%macro hex2(n);
  %local digits n1 n2;
  %let digits = 0123456789ABCDEF;
  %let n1 = %substr(&digits, &n / 16 + 1, 1);
  %let n2 = %substr(&digits, &n - &n / 16 * 16 + 1, 1);
  &n1&n2
%mend hex2;

/* convert RGB triplet (r,g,b) to SAS color in hexadecimal.
   The r, g, and b parameters are integers in the range 0--
255 */
%macro RGB(r,g,b);
  %cmpres(CX%hex2(&r)%hex2(&g)%hex2(&b))
%mend RGB;
```

/********** Gráficos comparador secuencial **********/

```
PROC IMPORT OUT= WORK.a
            DATAFILE= "**********************************
ompararsecuencial.xls"
            DBMS=EXCEL REPLACE;
     RANGE="Hoja1$";
     GETNAMES=YES;
     MIXED=NO;
     SCANTEXT=YES;
     USEDATE=YES;
     SCANTIME=YES;
RUN;

proc means data=a;
var Missclassification_rate;
by Aumento_de_datos;
run;

proc means data=a;
var Tiempo_minutos;
by Aumento_de_datos;
run;

proc sort data=a;
by Metodo;
run;
```

```
proc means data=a;
var Tiempo_minutos;
by Metodo;
run;
PROC SGPLOT  data= a ;
styleattrs DATACOLORS=(%RGB(153, 204, 255) %RGB(255, 255,
153) %RGB(204, 255, 204));
vbar Nombre /response=Missclassification_rate group= metodo
groupdisplay=cluster stat=mean ;
yaxis label = 'Misclassification error'  grid values=(20 40
60 80 100) valueshint;
xaxis label = 'Modelos';
run;

PROC SGPLOT  data= a ;
vbar metodo /response=Tiempo_minutos
fillattrs=(color=%RGB(153, 204, 255))stat=mean ;
yaxis label = 'Minutos ejecución';
xaxis label = 'Aumento de datos';
run;

DATA aa;
set a;
where Metodo='ninguno' and Nombre ^= 'Secuencial3.1';
run;

proc gplot data=aa ;
plot Capas*Accuracy;
run;


/******************** Gráficos residuos generados ********************/


data graph.waste;
set graph.graphwaste;
rename Pais=Zona;
run;

pattern1 color = %RGB(255, 204, 255);
pattern2 color = %RGB(153, 204, 255);
axis1 label=(a=90 f="Arial/Bold" "Toneladas residuos
generados (millones)");
proc gchart data=graph.waste;
   vbar Zona / subgroup=Zona patternID = subgroup group=Año
sumvar=Residuos_M
   raxis=axis1 ;
run;
quit;
```

```
/******************** Gráficos comparador arquitecturas ********************/

PROC IMPORT OUT= WORK.a2
            DATAFILE= "
"
            DBMS=EXCEL REPLACE;
     RANGE="Hoja2$";
     GETNAMES=YES;
     MIXED=NO;
     SCANTEXT=YES;
     USEDATE=YES;
     SCANTIME=YES;
RUN;

PROC SGPLOT  data= WORK.a2;
vbar Nombre /response=Accuracy fillattrs=(color=%RGB(255,
204, 255));
yaxis label = 'Accuracy' grid values=(20 40 60 80 100)
/*valueshint*/;
xaxis label = 'Arquitectura';
run;

PROC SGPLOT  data= WORK.a2;
vbar Nombre /response=Tiempo_minutos
fillattrs=(color=%RGB(204, 255, 204));
yaxis label = 'Tiempo minutos' ;
xaxis label = 'Arquitectura';
run;

PROC SGPLOT  data= WORK.a2 ;
vbar Nombre /response=Capas fillattrs=(color=%RGB(153, 204,
255));
yaxis label = 'Numero de capas';
xaxis label = 'Arquitectura';
run;


PROC SGPLOT  data= WORK.a2 ;
vbar Nombre /response=Parametros fillattrs=(color=%RGB(255,
255, 153));
yaxis label = 'Parametros';
xaxis label = 'Arquitectura';
run;
```

```
/******************** Gráficos comparador pre entrenados ********************/

PROC IMPORT OUT= WORK.a3
            DATAFILE= "
"
            DBMS=EXCEL REPLACE;
     RANGE="Hoja3$";
     GETNAMES=YES;
     MIXED=NO;
     SCANTEXT=YES;
     USEDATE=YES;
     SCANTIME=YES;
RUN;

data nuevo;
     set work.a3;
     if Batch_size ^= 32 then Arquitectura = 'ResNet';
     Else Arquitectura = 'VGG';
run;


PROC SGPLOT  data= WORK.a3;
vbar Nombre /response=Missclassification_rate
fillattrs=(color=%RGB(255, 204, 255));
yaxis label = 'Misclassification'/* grid values=(20 40 )
/*valueshint*/;
xaxis label = 'Nombre';
run;


PROC SGPLOT  data=nuevo ;
styleattrs DATACOLORS=(%RGB(153, 204, 255) %RGB(255, 255,
153) %RGB(204, 255, 204));
vbar Arquitectura /response=Tiempo_horas group=
Arquitectura  groupdisplay=cluster stat=mean ;
yaxis label = 'Tiempo horas'  grid values=(20 40 60 80 100)
valueshint;
xaxis label = 'Modelos';
run;
```