E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Visualizando neuronas en Redes Neuronales Convolucionales

Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumno: David Erroz Arroyo
Director: Mikel Galar Idoate
Pamplona, 10/06/2019

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Resumen

Hoy en día, las Redes Neuronales Convolucionales son de vital importancia en Machine Learning cuando se trabaja con imágenes debido a su gran capacidad detectando objetos, reconociendo patrones en imágenes y demás.

Sin embargo, al igual que el resto de modelos de Deep Learning, tienen un gran hándicap, que es su falta de interpretabilidad para un humano. Esto es por su estructura multicapa altamente no lineal. Y esta falta de transparencia limita su aplicación práctica a pesar de su gran rendimiento en numerosos problemas reales.

Por tanto, el propósito de este trabajo es el de estudiar y analizar diferentes métodos y técnicas existentes para visualizar neuronas tanto de la última capa (las relativas a las decisiones) como de capas intermedias (para ver qué es lo que está aprendiendo la red) y llevar a cabo una comparativa de los métodos a través de diferentes imágenes y arquitecturas. De esta forma, podremos extraer conclusiones en un área de cada vez más relevancia.

Abstract

Nowadays, Convolutional Neural Networks are of paramount importance in Machine Learning when working with images due to their great performance detecting objects, recongnizing patterns in images and so on.

However, like the rest of Deep Learning models, they have a significant disadvantage which is their lack of interpretability for the human being. This is because of their multilayer nonlinear structure. And that absence of transparency, limits their application in practice although they perform impressively well in a large amount of different problems.

Thus, the purpose of this project is to study and analyze different existing methods and techniques for visualizing neurons, either the ones from the last layer (those related to decisions) or the ones from intermediate layers (to see what the network is learning) and then carry out a comparative study of the methods through different architectures and images. In this way, we can draw conclusions in an area of growing relevance.

Palabras clave: Red Neuronal Convolucional, visualización, método, neurona

Keywords: Convolutional Neural Network, visualization, method, neuron

Índice

1. Introducción

Hoy en día la influencia que tiene la Inteligencia Artificial en nuestras vidas es muy grande. Prácticamente todo el mundo tiene acceso a dispositivos que incorporan software 'inteligente' y hemos aprendido a convivir con ello. No solo eso, sino que en ámbitos muy importantes de la vida como en el de la medicina o el de las finanzas, cada vez son más y más frecuentes este tipo de sistemas 'inteligentes'. Además, todo indica que estos sistemas han llegado para quedarse y que en un futuro serán aún más habituales ya que cada vez tienen más y más presencia en nuestra sociedad.

Sin embargo, siempre nos queda la duda de si debemos confiar ciegamente o no en estos sistemas 'inteligentes'. Sabemos que nos son de gran ayuda y que en ocasiones incluso tienen un rendimiento superior al del ser humano en muchas facetas, pero no son infalibles. Así pues, el ser humano debería requerir al sistema información acerca de la decisión tomada que le hiciera confiar o no en la misma. En otras palabras, el sistema debería ser capaz de explicar su decisión, así como lo haría un ser humano para poder darla por válida o no. Pero, generalmente, esto no es así.

Resulta evidente que cuanto más complejo es el sistema 'inteligente', más complicado se vuelve comprender su funcionamiento y por tanto más difícil entender las decisiones que adopta éste.

De hecho, uno de los temas más debatidos e investigados en Machine Learning y en concreto en Deep Learning es el de la interpretabilidad de los modelos entrenados. En el ámbito del Deep Learning se suele emplear bastante el término "black box", que hace referencia a la falta de información que dispone el usuario para poder determinar el proceso de inferencia llevado a cabo por el algoritmo para predecir tal o cual cosa. Es decir, el modelo actúa como una especie de "caja negra" para el usuario que introduce unos datos y recibe unos resultados pero que a pesar de conocer el funcionamiento matemático del algoritmo (cómo procede éste para determinar unas salidas en función de las entradas), le resulta muy difícil entender e interpretar el proceso desde un punto de vista lógico/abstracto más parecido a cómo lo haría un humano. Esta dificultad va en aumento conforme los modelos son más grandes y profundos.
Es por ello que surgen varios problemas a raíz de esto como son la falta de confianza en los resultados que arroja el modelo (¿Cómo podemos confiar en algo que ni siquiera sabemos explicar cómo funciona?) o la gran dificultad para poder debuggear el modelo en busca de algún fallo o corrección que pueda hacer mejorar el comportamiento del modelo, entre otros.

Además, todo esto es particularmente interesante cuando hablamos de imágenes y es que la visión por computador es una de las ramas con más peso en la Inteligencia Artificial de hoy en día. La visión por computador se emplea en numerosos sectores como el de la automoción (para la inspección en la fabricación y el ensamblaje), alimentación (en el control de calidad de los productos alimentarios), electrónica (en la manipulación e identificación de los distintos componentes o en la comprobación de la correcta soldadura y

ensamblaje de piezas), medicina (en la medición de volúmenes de tejidos, localización de tumores o identificación de otras enfermedades), cirugía (para el diagnóstico y planificación de tratamientos), seguridad (en la identificación de rostros y reconocimiento biométrico), etc. Es por eso que sería de vital importancia poder ser capaces de entender y explicar en todo momento las decisiones de estos sistemas basados en visión artificial.

En concreto, las Redes Neuronales Convolucionales(CNNs) son el tipo de redes más usado en Deep Learning para tareas de imagen por su gran rendimiento detectando objetos, reconociendo patrones en imágenes, etc. Y al igual que el resto de modelos de Deep Learning carecen de interpretabilidad debido a su elevada cantidad de parámetros y su complejidad. A pesar de esto, existen técnicas relativamente recientes que tratan precisamente de lograr aportar algo de interpretabilidad a estos modelos permitiendo conocer en qué se fijan estas redes para predecir lo que predicen.

1.1. Motivación

La cantidad de técnicas y métodos existentes para explicar predicciones en modelos de Deep Learning va en aumento. Dichas técnicas se centran más específicamente en visualización de predicciones en Redes Neuronales Convolucionales, pues existe una creciente preocupación por entender las decisiones de estos modelos tan importantes en aplicaciones de visión artificial. Sin embargo, a pesar de tratarse de algo que podría tener gran relevancia, la información general existente acerca de estas técnicas no es abundante. Esto aunado con el ámbito de investigación al que está vinculado este campo, inspira este trabajo que busca poner algo de luz en este asunto.

1.2. Objetivos

Principal:

Estudiar y evaluar los principales métodos de visualización de Redes Neuronales Convolucionales para así poder comprobar realmente cómo de bien funcionan en diferentes situaciones y poder concluir si su uso nos podría ser de ayuda o no.

Particulares:

1. Estudiar los diferentes métodos. Analizar y entender cómo funcionan.
2. Implementar/poner en funcionamiento los métodos. Trasladar a código cada método para poder probar realmente su funcionamiento con ejemplos.
3. Comparar la teoría del método (cómo se supone que debería funcionar) con cómo realmente funciona en la práctica. Analizar la congruencia de los resultados con la descripción teórica del método.
4. Llevar a cabo una comparación entre los métodos estudiados para poder establecer cuál de ellos obtiene mejores resultados.

2. Preliminares

2.1. Machine Learning

El Machine Learning o, en castellano, aprendizaje automático, es una disciplina del ámbito de la Inteligencia Artificial que abarca todos aquellos sistemas que aprenden automáticamente (aprenden solos sin necesidad de programar explícitamente el conocimiento que son capaces de adquirir). Con 'aprender' entiéndase identificar patrones complejos en multitud de datos. Es decir, el sistema/máquina aprende realmente gracias a un algoritmo que revisando los datos que recibe es capaz de predecir comportamientos futuros. Esto por ende implica que los sistemas mejoran su aprendizaje de forma autónoma con el tiempo (sin intervención humana) en base a procesar cada vez más y más datos nuevos (mejoran con la experiencia).

Concretamente un reputado científico informático estadounidense como **Tom M. Mitchell** define el Machine Learning en uno de sus libros como *"El estudio de algoritmos de computación que mejoran automáticamente su rendimiento gracias a la experiencia. Se dice que un programa informático aprende sobre un conjunto de tareas, gracias a la experiencia y usando una medida de rendimiento, si su desempeño en estas tareas mejoran con la experiencia".* Para entender mejor la diferencia entre un algoritmo de aprendizaje autónomo y uno tradicional viualizar la *Figura 1*.

Muchos de los métodos empleados en Machine Learning existen desde hace ya varias décadas, pero el motivo por el que hoy en día se habla tanto de ellos es por el aumento de la capacidad computacional de los ordenadores, que nos permiten tratar problemas que antes eran inviables y también por la revolución de los datos motivada por la digitalización (ha supuesto un aumento ingente de datos que pueden ser procesados y modelizados para obtener conocimiento de ellos).

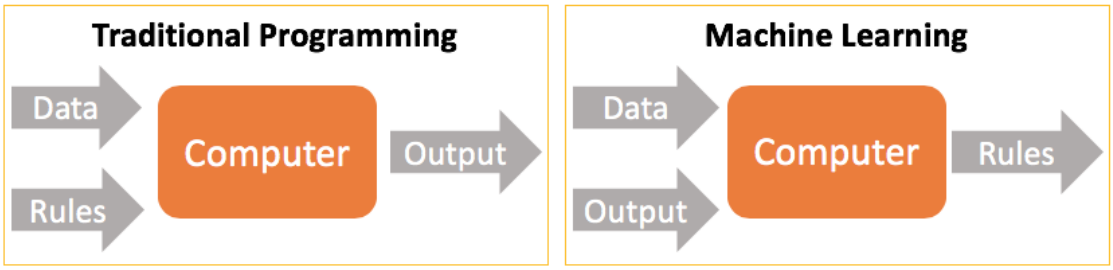


Figura 1. Programación tradicional vs Aprendizaje automático. Antes, el ordenador computaba una salida en base a unos datos de entrada y un programa que se aplicara sobre ellos. Ahora en Machine Learning el ordenador arroja como salida el propio programa, aprendido en base a los datos de entrada. Se dice que computa un modelo.

El objetivo pues de los algoritmos de Machine Learning es el de construir un modelo que capture el conocimiento aprendido sobre los datos de entrada y que gracias a este modelo posteriormente podamos inferir conocimiento sobre nuevos datos. Es así que podemos distinguir 2 posibles fases: la fase de aprendizaje y la fase de inferencia. La fase de aprendizaje está orientada a

construir dicho modelo en base a unos datos (datos de entrenamiento) y una vez confeccionado el modelo llega la fase de inferencia, en la que podemos ver lo potente que es el modelo con datos completamente nuevos y desconocidos.

Existen diferentes tipos de aprendizaje en Machine Learning. Los dos principales son el aprendizaje supervisado y el no supervisado. Los dos difieren en la forma en que aprenden el modelo.

El primero de ellos aprende en base a unos datos previamente etiquetados por un ser humano. Es decir, en el proceso de aprendizaje de un algoritmo de aprendizaje supervisado, un ser humano se encarga de indicar al algoritmo los datos de entrada, así como las predicciones o salidas que debe dar el modelo para esos mismos datos. De esta forma, el modelo (que no es otra cosa que una función que toma como entrada unos datos y devuelve un resultado o predicción sobre esos datos) tiene que adaptarse para procurar dar la salida esperada en base a los datos de entrada. Implícitamente aprende la relación existente entre los datos de entrada y salida, de modo que para nuevos datos, si ha entrenado lo suficientemente bien (con muchos datos, significativos, diferentes entre sí, etc), será capaz de aproximar bien la salida. Nótese que en el fondo de esta manera estamos intentando crear un algoritmo capaz de aproximar el razonamiento humano, pues tratamos de computar una función que prediga para unos datos de entrada lo que predeciría un humano.

Por su parte, el aprendizaje no supervisado tiene como objetivo la confección de un modelo basado exclusivamente en los datos de entrada (sin partir de una salida etiquetada a la que ajustarse).

En este trabajo nos centraremos en el aprendizaje supervisado.

Problemas principales:

Existen dos tipos principales de problemas de aprendizaje supervisado: problemas de clasificación y problemas de regresión. La principal diferencia entre ellos es el tipo de salida que arrojan. En el primero de ellos la salida es un valor discreto (clase) y en el segundo, un valor real (número).

Tanto para problemas de clasificación como para problemas de regresión, se conocen múltiples algoritmos que computan modelos para resolverlos. Uno de los modelos más simples en el caso de regresión es el de regresión lineal. En cambio, en el caso de clasificación, el primer algoritmo que se suele estudiar es el de regresión logística.

En regresión lineal, se tiene por objetivo estimar un modelo en el que la variable de salida(**y**) se expresa linealmente en términos de las variables de entrada(**X**) y de un conjunto de parámetros, que son los que hay que entrenar (adaptar sus valores) para que se obtenga la salida esperada para los datos de entrada (ver *Figura 2*). La idea es buscar aquellos parámetros tal que la salida arrojada por el modelo sea lo más parecida posible a los valores esperados de salida (los etiquetados **y**) para los ejemplos de entrenamiento (conjunto de datos de entrada junto con sus correspondientes salidas esperadas). La clave está en medir lo

'parecidas' que son las salidas deseadas y obtenidas. Para ello se introduce una medida de error que suele ser la diferencia elevada al cuadrado, es decir, $(y-d)^2$ donde **y** es la salida obtenida por el modelo y **d** la deseada. Al final, el algoritmo de aprendizaje consiste en minimizar el error global (la suma de errores para cada ejemplo).

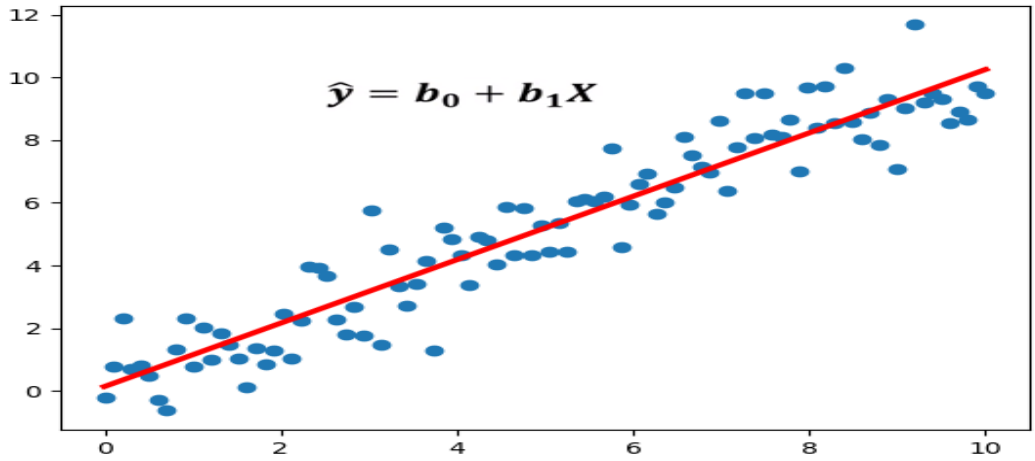


Figura 2. Ejemplo de modelo de regresión lineal.

Este proceso de minimización lo podríamos llevar a cabo de forma analítica, esto es, resolviendo directamente el sistema de ecuaciones resultante de igualar a 0 el gradiente de la función de error global con respecto a cada parámetro del modelo, pero conforme el algoritmo emplea un mayor número variables y parámetros, esto resulta impracticable computacionalmente. A cambio, la manera de proceder para resolver este sistema es iterativamente mediante el método de descenso por gradiente. Este método consiste en partir de unos valores aleatorios de los parámetros del modelo e ir actualizando dichos valores iterativamente de modo que converjan hacia el valor en el que el gradiente del error con respecto al parámetro sea 0. Esto es: $\theta = \theta - \alpha \frac{\partial J}{\partial \theta}$, donde **θ** es el parámetro del modelo a actualizar, **J** es la función del error que se pretende minimizar y $\boldsymbol{\alpha}$ es un hiperparámetro (denominado ratio de aprendizaje) que se utiliza para controlar el ritmo al que se desciende el gradiente con el fin de garantizar una convergencia eficiente. El proceso se ilustra en la *Figura 3.*

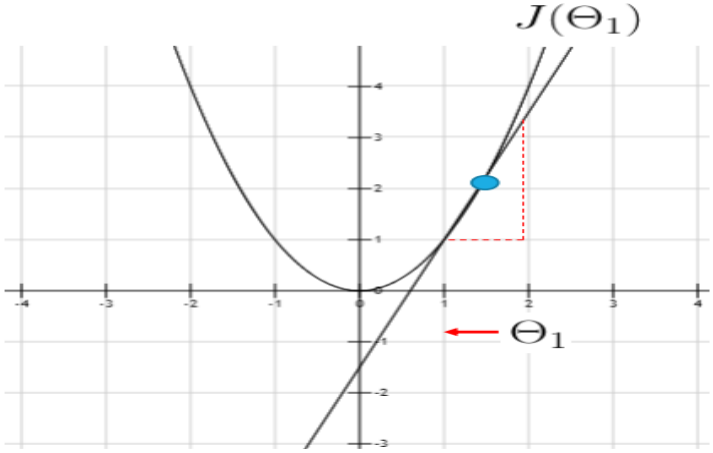


Figura 3. Una iteración del descenso por gradiente. Se muestra como θ disminuye, pues la pendiente de la recta tangente a la función de error en ese punto(derivada) es positiva (la función es creciente en ese punto), luego el valor mínimo de J se encuentra a la izquierda del punto.

En regresión logística, la salida que debemos dar ya no es un número real sino un valor discreto que representa una clase[1]. En concreto, las salidas a esperar vendrán en binario (1 si pertenece a la clase buscada o 0 si no). Así pues, la salida del modelo ha de ser un número entre 0 y 1 que represente la probabilidad de pertenencia a la clase del ejemplo considerado. Si dicha probabilidad es mayor o igual a 0.5 podemos considerar que el ejemplo pertenece a la clase, y si no, que no pertenece.

El hecho de que la salida sea un número entre 0 y 1 hace que la regresión logística sea algo diferente a la regresión lineal. Ahora, debemos introducir una función que tome como entrada el resultado de la regresión (un número real) y devuelva un valor entre 0 y 1. Esta función suele ser la sigmoide $\sigma(z)=\frac{1}{1+e^{-z}}$.
Además de este pequeño cambio con respecto al modelo de regresión lineal, un modelo de regresión logística introduce una función de error algo diferente, pues se necesita reflejar el error en un intervalo reducido pero conservando las mismas proporciones de error (si la salida esperada es 1 y la obtenida es 0, el error no ha de ser 1 sino infinito pues es lo más lejana que puede estar la salida obtenida de la esperada). El error será ahora: $-d*\log(y)-(1-d)*\log(1-y)$, donde **d** es la salida deseada e **y** la obtenida.

El resto no cambia con respecto a regresión lineal, pues la idea de minimizar el error global (la suma de errores para cada ejemplo) iterativamente mediante el algoritmo de descenso por gradiente es la misma que en regresión lineal.
En la *Figura 4* se puede apreciar la diferencia entre un modelo de regresión lineal y uno de regresión logística.

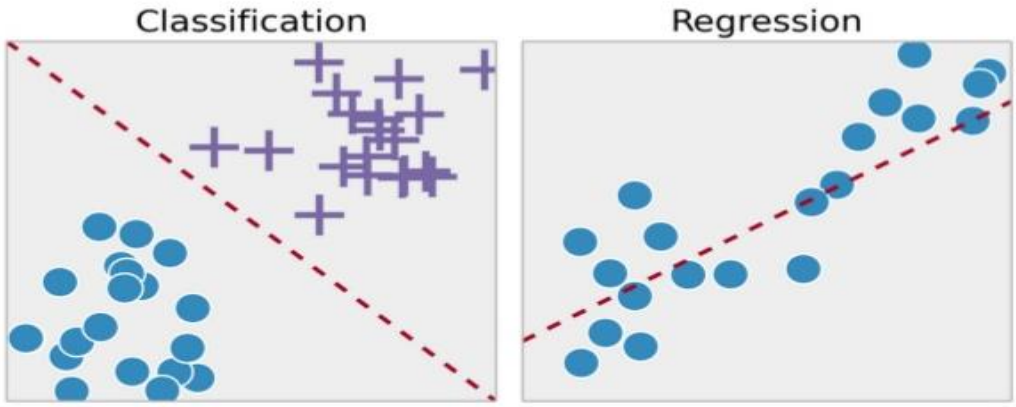


Figura 4. Los modelos vienen representados por las líneas rojas discontinuas. El de la izquierda es un modelo de clasificación y el de la derecha de regresión.

Train/Test:

Como ya se ha dicho, el objetivo de los algoritmos de aprendizaje supervisado es el de generar un modelo que aprenda lo suficientemente bien las relaciones entre las entradas y sus salidas esperadas para posteriormente poder de forma fiable predecir las salidas para nuevas entradas no antes vistas.

[1] Esta condición la cumple cualquier algoritmo de clasificación, a excepción de que existen modelos que admiten más de una salida (multiclase).

Para ello, se aprende el modelo sobre un conjunto de datos denominado *train set* o conjunto de entrenamiento en castellano. Lo que ocurre es que muchas veces este conjunto no es el ideal bien sea porque no es lo suficientemente representativo de los posibles datos futuros que pueda tomar el modelo y esto resulta preocupante. Resulta preocupante porque lo que se pretende es conseguir un modelo que generalice bien a nuevos ejemplos (para ello se crea). Se necesita pues conocer cómo de bien es capaz de clasificar nuevos ejemplos nuestro modelo para poder dar validez al mismo. Es por eso que se introduce un nuevo conjunto de datos denominado *test set* o conjunto de pruebas. De esta forma si aprendemos el modelo mediante el conjunto de *train,* lo podemos evaluar mediante el de *test,* que es importante que no contenga los mismos ejemplos con los que se ha entrenado el modelo ya que lo que persigue es obtener una medida representativa de cómo funciona el modelo ante nuevos datos desconocidos.

Además de estos conjuntos, puede entrar en escena un tercer conjunto de datos denominado *validation set* o conjunto de validación en castellano. Este conjunto sirve para ayudar a construir el modelo. Se emplea cuando nos interesa seleccionar un modelo de entre un conjunto. Esto ocurre cuando no sabemos qué hiperparámetros escoger para nuestro modelo (cuáles serán más adecuados para nuestro problema concreto) y necesitamos pues probar a configurar varios modelos cada uno con un valor diferente para sus hiperparámetros. El entrenamiento de estos modelos se lleva a cabo de forma normal mediante el conjunto de *train,* pero ahora necesitamos un nuevo conjunto distinto que nos permita evaluar qué modelo se adapta mejor a nuestro problema y así poder escoger uno. Este conjunto ha de ser distinto al de *train* ya que sirve para escoger el modelo que supuestamente mejor generalice a datos futuros de entre los modelos contemplados y también debe ser distinto al de *test* porque el de *test* tiene por objetivo aportar una medida de lo bien que funciona el modelo para datos desconocidos, y esta medida ya no sería fiable si el conjunto ha sido empleado para seleccionar el modelo (medida optimista).

Generalmente estos conjuntos se establecen a partir de un único conjunto inicial de datos en una partición similar a: 60% *train*, 20% *test* y 20% *val*.

Métricas:

Se ha hablado de evaluación de modelos, que es algo fundamental en Machine Learning, pero sin embargo no se ha hablado de cómo evaluarlos (qué medida emplear para hacerlo). La realidad es que existen varias métricas para analizar el error de los modelos. Hablaremos de algunas de ellas.

La primera es el ratio de clasificación. Esta medida es la más básica y consiste en calcular el porcentaje de aciertos del modelo para los datos contemplados. Esta métrica a pesar de ser útil en muchos problemas, no tiene en cuenta el número de ejemplos de cada clase y esto impide reflejar fielmente la calidad del clasificador.

Métricas sin ese problema son Precisión, que consiste en calcular el porcentaje de aciertos en los ejemplos predichos como una determinada clase y Recall, que

consiste en calcular el porcentaje de aciertos en los ejemplos que realmente son de una determinada clase.
Finalmente podemos calcular una medida que combine Precisión y Recall para así buscar un buen balance entre precisión y recall. Esto se consigue mediante la métrica F score: $F_{score} = 2\frac{P*R}{P+R}$, donde P=Precision y R=Recall.

2.2. Redes Neuronales y Deep Learning

Dentro del ámbito del Machine Learning, una de las técnicas más conocidas y con más renombre en la actualidad es la de las Redes Neuronales Artificiales(RNAs). Estos algoritmos, que datan de los años 40 y 50, nunca gozaron de una gran popularidad hasta ahora debido a la gran cantidad de recursos computacionales que requieren y de la enorme cantidad de datos de la que dependen para su correcto funcionamiento. Actualmente en cambio, se les considera como una de las mejores técnicas para muchas aplicaciones entre las cuales destacan la visión por computador o el procesamiento del lenguaje natural.

La idea de las Redes Neuronales es, como su propio nombre indica, tratar de imitar el cerebro humano. A pesar de que no se conoce exactamente cómo funciona el cerebro humano, sí que se saben ciertas características como la estructura que tiene e inspirándose vagamente en estas observaciones surgen estos algoritmos. Concretamente el algoritmo consiste en un conjunto de unidades denominadas neuronas conectadas entre sí para transmitirse información de unas a otras. Cuando los datos/información de entrada atraviesan esta red de neuronas, se acaba produciendo un resultado o unos valores de salida que son los que nos interesan. De este modo se puede entender una red neuronal como una gran función que produce una salida dependiente de unos datos de entrada, pero dicha función para ello se apoya en multitud de subfunciones que la componen. Es decir, se puede ver una red neuronal como una función de funciones que permiten a la función principal computar y capturar potentes relaciones en los datos. El objetivo final es que dicha gran función acabe realizando predicciones correctas para los datos de entrada (la salida obtenida por la red sea la esperada). Luego por tanto estamos ante un caso de algoritmo de aprendizaje supervisado en el que necesitamos una función que se adapte a unos datos previamente etiquetados (recibe una serie de salidas esperadas para unas entradas determinadas y necesita ajustarse para producir esas mismas salidas).

En su versión más básica, una red neuronal de una sola neurona se puede concebir como un modelo de clasificación corriente como el explicado en el apartado anterior. Sin embargo, cuando la red dispone de más de una neurona el modelo se complica algo. Ahora, la salida de la red ya no es la salida de la única neurona que teníamos, sino que, al haber más de una, necesitamos otra neurona que compute la salida en base a las salidas de las demás neuronas. Es decir, se ha añadido una neurona que está conectada al resto de neuronas de la red. Se dice que dicha neurona pertenece a otra capa de la red, pues necesita tomar como entrada lo que las demás arrojan como salida, siendo

secuencialmente posterior (pertenece a la capa siguiente). Un ejemplo de representación de red neuronal se puede observar en la *Figura 5.*

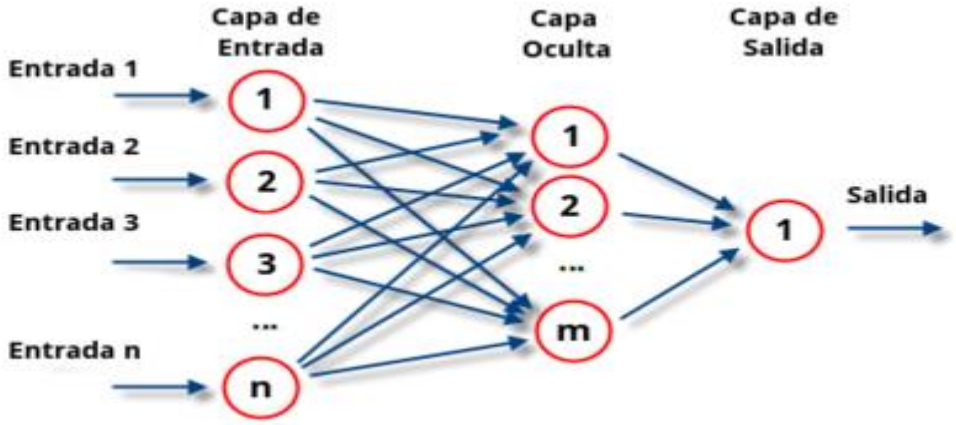


Figura 5. Ejemplo de red neuronal de una única capa oculta. Se les denomina capas ocultas a aquellas capas conectadas tanto por la entrada como por la salida con otras capas. Se les considera capas a la entrada y a la salida de la red.

En las Redes Neuronales cada neurona actúa como un clasificador con sus propios parámetros y su función de activación (en este caso la función ReLU: $ReLU(x) = max(0, x)$ es una de las más utilizadas). Lo que cambia es que al haber una interconexión de neuronas, los clasificadores ahora toman como entrada la salida de otros clasificadores (los de la capa anterior). Además, ahora debemos entrenar la red en su conjunto y no cada clasificador por separado.

De todos modos, el entrenamiento de una red neuronal no es muy diferente al de un modelo de clasificación normal, pues en el fondo ya hemos concluido que una red es en sí misma es un gran clasificador. La idea de minimizar una función objetivo es la misma. De hecho, la propia función es la misma solo que ahora la salida obtenida por el modelo es más compleja porque depende de muchos más parámetros.

Una red neuronal puede ser lo compleja que se quiera y eso dependerá del número de capas y de neuronas por capa que tenga la red. Cuantas más capas disponga la red y más neuronas tenga, mayor será supuestamente su capacidad para aprender relaciones complejas en datos, pero a cambio mayor será también la dificultad y complejidad de la red. A las redes neuronales con muchas capas intermedias (ver *Figura 6*) se les engloba en un nuevo marco del mundo del aprendizaje autónomo denominado Deep Learning (aprendizaje profundo en castellano).

Para entrenar bien estas redes tan profundas se necesitan muchos datos y técnicas específicas. El entrenamiento de estas redes es un proceso costoso tanto en tiempo como en recursos computacionales, pero se ha constatado que merece la pena en vista a sus magníficos resultados en aplicaciones complejas como el reconocimiento de voz, localización de caras e identificación de emociones faciales o incluso en la conducción autónoma. La complejidad del entrenamiento en este tipo de redes tiene que ver con que ya no se pueden

actualizar todos los parámetros[2] del modelo a la vez, sino que se recorre capa a capa actualizando poco a poco los pesos. Se sigue pretendiendo lograr que el gradiente de la función de error con respecto a cada peso sea 0(mínimo error en ese punto) y para ello se aplica la regla de la cadena, que nos permite aprovechar cómputos de gradientes anteriores para calcular los actuales si recorremos la red en sentido contrario (desde las capas más profundas a las primeras capas - *Backpropagation*).

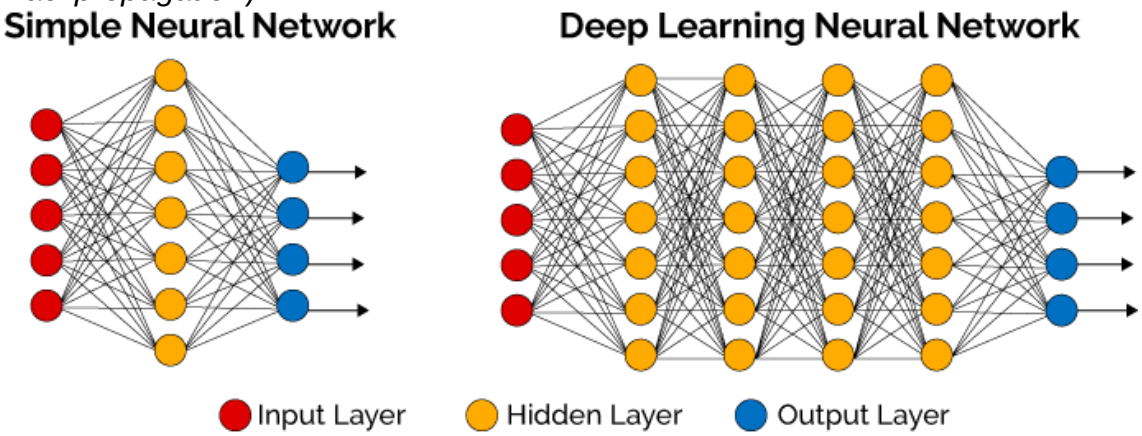


Figura 6. A la izquierda se muestra una red neuronal sencilla y a la derecha una profunda. La diferencia en el número de capas de la red es lo que introduce el término Deep Learning o Aprendizaje profundo.

Más adelante en este trabajo se hablará algo más específicamente de algunos aspectos importantes a tener en cuenta durante proceso de entrenamiento de las redes neuronales.

2.3. Redes Neuronales Convolucionales. Arquitecturas

Un tipo de red neuronal diferente al visto en el apartado anterior es la red neuronal convolucional[1]. Este tipo de redes surgen debido a la importancia de la visión por computador en el mundo del Deep Learning.

Las redes neuronales convencionales son *fully connected,* que significa que todas las neuronas de una capa oculta están conectadas a todas las neuronas de la capa que les precede y sucede. Esto implica que cada una de las características de entrada está conectada a cada neurona de la capa siguiente y en el caso de las imágenes, que cada píxel de la imagen de entrada está conectado a cada neurona de la primera capa oculta. Pero esto supone un gran problema ya que cuando se trabaja con imágenes, se suele trabajar con grandes imágenes de alta resolución (y cada vez mayor). Esto hace que el empleo de redes neuronales *fully connected* sea prácticamente inviable por la grandísima cantidad de pesos que debería manejar.

Por este motivo surgieron las redes neuronales convolucionales. Estas redes constan de diversas capas convolucionales y de *pooling*(agrupación) alternadas,

[2] En realidad, para hacer referencia a los parámetros de una red neuronal se emplea el término 'pesos', que van asociados siempre a una conexión concreta (cada uno a una).

acabando al final con una serie de capas convencionales *fully connected* (*Figura 7*). La entrada a una capa convolucional suele ser una imagen de tamaño **m** x **n** x **c** donde **m** y **n** son la anchura y altura de la imagen y **c**, la profundidad o el número de canales. Las capas convolucionales suelen constar de más de un filtro (también denominado kernel) de tamaño **f** x **f** x **c**[3] cada uno. De esta forma, cada filtro genera mediante la operación de convolución un nuevo mapa de características bidimensional. El total de mapa de características generados a la salida de la capa será igual al número de filtros de la capa, por lo que podemos ver la salida en su conjunto como una nueva imagen cuyo número de canales es igual al número de filtros de la capa. Esta nueva imagen pasará a ser la entrada de la siguiente capa después de que se le haya sumado un término independiente o bias y haber pasado por una función de no linealidad que dará lugar a lo que llamamos mapa de activaciones (mismo procedimiento que en las redes convencionales). Dicha función de activación, al igual que en las redes convencionales, suele ser la ReLU.

Adicionalmente se suele introducir una capa de *pooling* tras la convolución (bien "max pooling" o bien "mean pooling") que sirve para reducir el tamaño de las imágenes realizando la correspondiente operación de agregación (máximo o media) sobre los valores de una región contigua.

Finalmente, como la salida ha de ser una clasificación (probabilidades para cada clase), se incorporan capas *fully connected* capaces de realizar dicha labor.

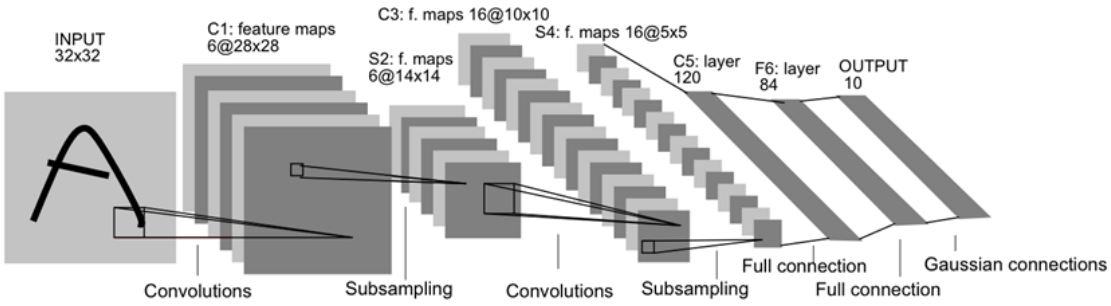


Figura 7. Esquema genérico de una red neuronal convolucional.

Con esto, la estructura de las Redes Neuronales Convolucionales parece clara. Sin embargo, falta por definir la clave del algoritmo, que es la operación de convolución.

La operación de convolución (explicada en la *Figura 8*) consiste en superponer un filtro sobre una imagen e ir desplazando/deslizando el filtro sobre la misma hasta haber recorrido así todas las regiones de la imagen (todos los píxeles de la imagen han de haber sido 'visitados' por el filtro en algún momento durante el proceso). Por cada desplazamiento del filtro sobre la imagen se calcula el valor resultante de la suma de todas las multiplicaciones de cada píxel por el valor correspondiente a la posición del filtro que superpone a ese píxel. De esta forma se construye el mapa de características: cada vez que se calcula un nuevo resultado se almacena en la siguiente posición de la matriz resultado. En una

[3] El número de canales de un filtro convolucional ha de coincidir con el número de canales de la imagen de entrada.

convolución sobre imágenes de tres dimensiones, el filtro a aplicar también se alinea a través de esa tercera dimensión con la imagen (es por eso que ha de coincidir en el número de canales).
El desplazamiento de la ventana de convolución (el filtro superpuesto sobre la imagen) por la imagen se puede configurar. Se puede establecer que sea de una unidad, dos, etc[4]. A esta variable se le llama *stride*.

Otra de las cosas que se pueden modificar es el *padding.* A veces es conveniente rellenar con 0 los bordes de las imágenes o mapas de entrada, pues esto nos permite controlar de alguna manera el tamaño que queramos que tengan las salidas. Y esto se puede conseguir estableciendo un tamaño para este *zero-padding*, esto es, configurar el número de filas o columnas de ceros a añadir alrededor de los bordes de la entrada. Esos son los principales hiperparámetros característicos de estas redes.

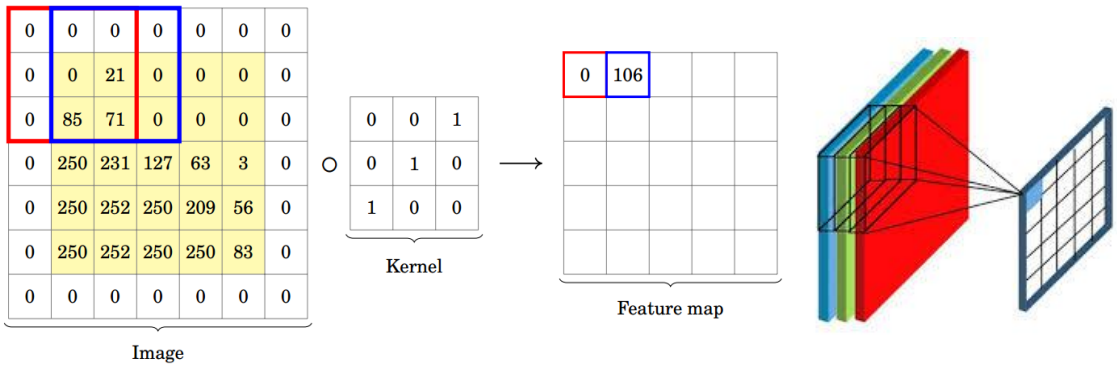


Figura 8. Ejemplo de la operación de convolución. A la izquierda se ilustra una convolución bidimensional sobre una matriz de entrada a la que se le ha aplicado *zero-padding* de una fila y una columna. A la derecha se muestra la idea de la aplicación de esta operación sobre volúmenes de dimensión 3.

Las Redes Neuronales Convolucionales están expresamente diseñadas para trabajar sobre imágenes. Una de las claves de este algoritmo es la conservación de la espacialidad durante el proceso, pues al contrario de lo que sucede en las redes *fully connected,* éstas en todo momento mantienen la información sobre la localidad de las activaciones que van calculando gracias a trabajar con matrices en lugar de con datos por separado. Esto último es esencial cuando se trata de imágenes ya que en éstas es muy importante no solo la relación existente entre píxeles sino también la localidad de los mismos, que se pierde completamente si no lo hacemos de esta forma.

Aparte de esto, una de las grandes razones del uso de este algoritmo como ya se ha mencionado es la eficiencia computacional que nos proporciona. El algoritmo se aprovecha de que en las imágenes se suele dar el caso de que características o rasgos presentes en una parte de la imagen pueden ser los mismos que otros que se encuentran en una zona totalmente distinta. Esto le

[4] Con unidad nos referimos a un píxel en el caso de estar llevándose a cabo la convolución sobre la imagen de entrada o a una activación en caso de estar realizándose sobre un mapa de activaciones proveniente de una capa anterior.

permite utilizar los mismos pesos para computar las activaciones sobre distintas partes de la imagen (deslizamiento de la misma ventana de convolución por toda la imagen). Por consiguiente, se reducen el número de conexiones y parámetros a entrenar.

Además, en estas redes cada valor de salida depende exclusivamente de un pequeño número de valores de entrada y no se establece una conexión con cada valor de entrada por cada valor de salida, ahorrándose así multitud de parámetros y ganando eficiencia. Esta idea está inspirada en cómo funciona el sistema visual biológico, en el cual las neuronas poseen una serie de campos receptivos que responden solo ante unos estímulos localizados en una región o área específica.

Explicada ya la idea que motiva las Redes Neuronales Convolucionales, puede entenderse aún mejor la razón de introducir la operación de *pooling* en las mismas. Y es que sabiendo que las redes conservan la localidad o espacialidad de la entrada y entendiendo también su gran objetivo de reducir notablemente la carga computacional, se puede ver como un proceso natural una vez computada una convolución, tratar de fijarse en qué zona de la imagen son los rasgos capturados más predominantes. De paso así se disminuye aún más el coste computacional, pues la matriz resultante de la operación de *pooling* resulta de unas dimensiones considerablemente menores que la matriz de características obtenida en la capa convolucional. Al mismo tiempo ayuda con la caracterización de la imagen obteniendo y localizando los rasgos predominantes en ella.

El gran éxito de estas redes es su gran capacidad a la hora de identificar características en las imágenes de entrada. Se ha demostrado con la experiencia que son muy buenas detectando características cada vez más y más complejas (como se muestra en la *Figura 9*). Esto se debe a que los filtros de las capas menos profundas actúan como detectores de rasgos pequeños como bordes o texturas y los filtros de las capas más profundas son capaces de detectar estructuras cada vez más y más complejas como formas u objetos a partir de los resultados de las detecciones previas.

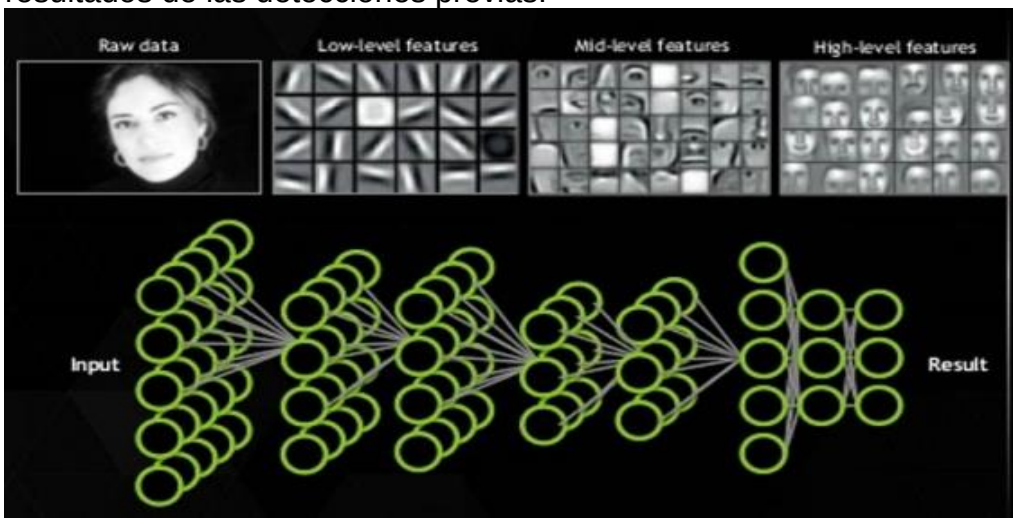


Figura 9. Esquema de detecciones de una Red Neuronal Convolucional. Conforme avanzamos por la red hacia capas más profundas, las características detectadas de la imagen de entrada son cada vez más complejas.

Arquitecturas:

En este punto sabemos cómo funcionan las Redes Neuronales Convolucionales y las diferentes partes de las que se componen. Sin embargo, sigue siendo de gran interés investigativo la búsqueda de una arquitectura que maximice el rendimiento en clasificación de imágenes. En este apartado veremos algunas de las más destacadas.

- LeNet-5[1]

  Esta arquitectura es la más antigua. Consiste en dos bloques de capas convolucionales y de *average pooling* seguidas por dos capas *fully connected*. Finalmente, en las versiones más modernas de esta arquitectura se emplea un clasificador *softmax* después de las capas *fully connected.* Básicamente un clasificador *softmax* lo que hace es normalizar la salida de la última capa de la red y así poder obtener una probabilidad para cada etiqueta de clase.

  Las imágenes que admite como entrada son de tamaño 32 x 32 x 1(escala de grises) y las pasa por una primera capa convolucional de 6 filtros de tamaño 5 x 5. Después aplica *average pooling* con un filtro tamaño 2 x 2 y *stride*=2[5] y de esta manera va reduciendo el tamaño de los mapas hasta el final. A la vez va aumentando el número de canales.

  Inicialmente, esta arquitectura fue propuesta para reconocer dígitos escritos a mano y es una de las arquitecturas más pequeñas (cuenta con aproximadamente 60.000 parámetros). En la *Figura 7* se muestra esta arquitectura.

- AlexNet[2]

  La arquitectura *AlexNet* (ver *Figura 10*) es bastante similar a la *LeNet-5* pero es más profunda. Tiene un mayor número de filtros por capa y muchísimos más parámetros (60 millones).

  La entrada a la red es una imagen de tamaño 227 x 227 x 3(RGB) y consiste en 5 capas convolucionales y 3 *fully connected*. En una misma capa convolucional suele emplear muchos filtros del mismo tamaño (por ejemplo utiliza 96 filtros de tamaño 11 x 11 x 3 en la primera capa) y además una particularidad de esta arquitectura es que emplea varias capas convolucionales consecutivas.

  También emplea *max pooling* después de ciertas capas convolucionales, siendo un *max pooling* poco convencional al utilizar ventanas de tamaño 3x3 y *stride*=2 (se solapan las regiones, algo que no es habitual en las regiones de *pooling*).

[5] Las capas de *pooling* funcionan de la misma manera que las convolucionales en el sentido en que aplican una ventana deslizante sobre la entrada. Lo que las diferencia es la operación que realizan sobre dicha ventana.

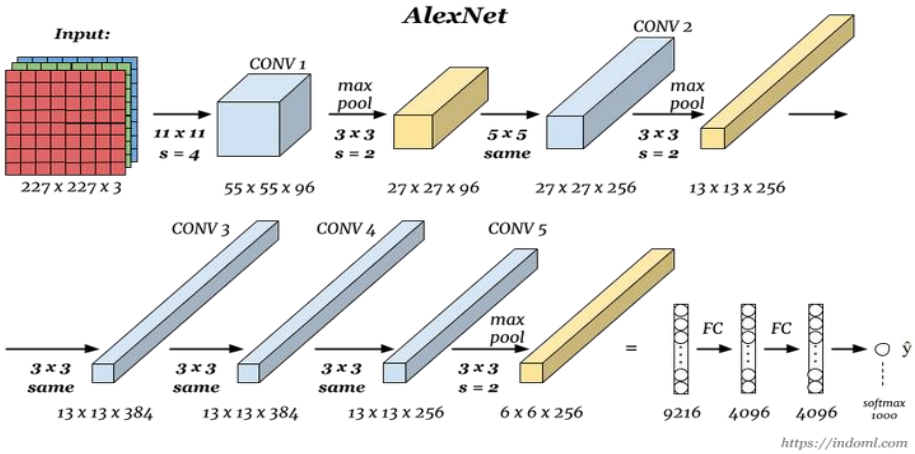


Figura 10. Esquema de la arquitectura AlexNet.

❖ VGG-16[3]

La arquitectura *VGG-16* utiliza filtros convolucionales siempre del mismo tamaño: 3 x 3 y *stride*=1(*padding* same[6]) y *max pooling* sobre ventanas 2x2, *stride*=2. El número de capas del que consta es de 16 y su arquitectura es muy uniforme. Emplea ventanas pequeñas pero muchos filtros.

Es una de las arquitecturas más utilizadas para la extracción de características en imágenes.

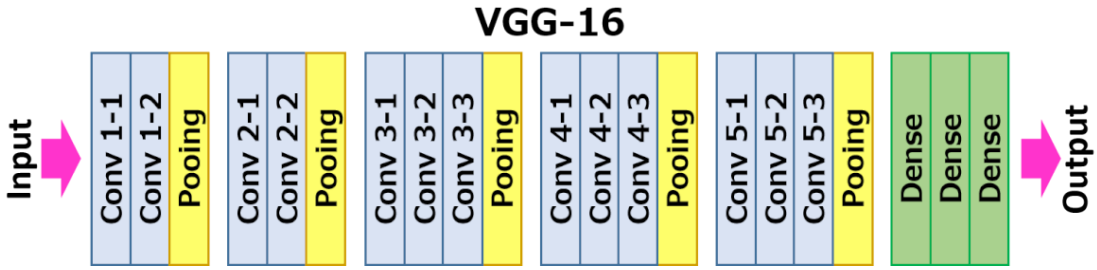


Figura 11. Esquema de la arquitectura VGG-16.

❖ ResNets[4]

La arquitectura *ResNet* es una de las más novedosas. Esta arquitectura se propuso con el objetivo de resolver el conflicto existente entre la necesidad de modelos muy profundos capaces de resolver tareas más complicadas y también a la vez incrementar o mejorar la precisión de la clasificación. Esto era un problema complicado ya que cuanto más profundo es el modelo el aprendizaje se vuelve notablemente más difícil.

Lo que propone *ResNet* es incorporar saltos en las conexiones entre capas. Es decir, permitir una conexión directa entre la entrada de una capa n y una capa $n + x$**.** En otras palabras, permitir saltos en las conexiones de la red.

[6] *Padding* 'same' se refiere a aplicar *padding* a la entrada de forma que la salida de la operación de convolución sea del mismo tamaño que la entrada original.

De hecho, se ha acabado probando que entrenar estas redes es más sencillo que entrenar las simples.

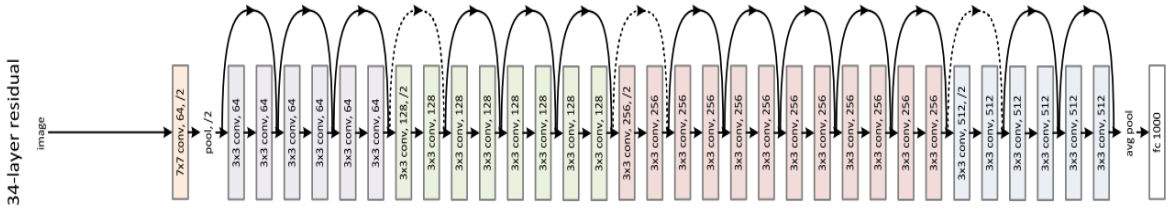


Figura 12. Esquema arquitectura *ResNets*.

- Inception[5]

La idea novedosa de esta arquitectura consiste en concatenar los resultados obtenidos de aplicar diferentes filtros de convolución (de diferentes tamaños, etc), ventanas de pooling a una misma entrada.

Esto permite al modelo beneficiarse de la extracción de características a múltiples niveles en un único paso (al mismo tiempo). De hecho, extrae características generales y locales a la vez. La filosofía de esta arquitectura es la de por qué tener que escoger entre opciones[7] diferentes si se pueden tener todas ellas.

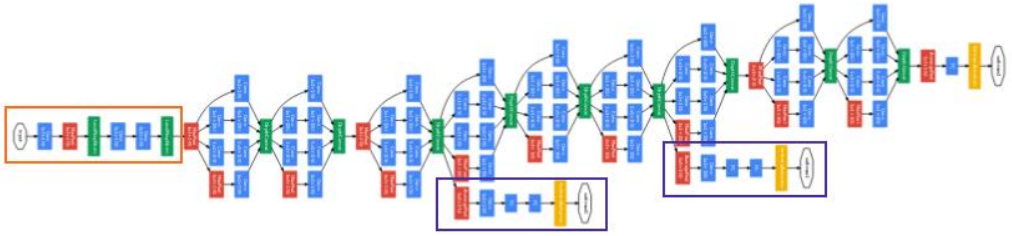


Figura 13. Esquema de arquitectura Inception Network.

- Exception[6]

Esta arquitectura es una extensión de la anterior (*Inception*). De hecho, al igual que la anterior, esta arquitectura fue presentada por Google. La principal diferencia con su antecesora es que introduce el concepto de convolución separable por profundidad. Es decir, en lugar de afrontar la convolución como un único paso en el que se obtiene un único mapa de características resultado de convolucionar un filtro con la entrada, ahora, se obtiene un mapa por cada canal. Es decir, se realiza una convolución independiente de cada canal de la entrada con cada canal del filtro.

- MobileNet[7]

MobileNet es una arquitectura también propuesta por *Google* pensada principalmente para aplicaciones de visión móviles. Por tanto, se centra

[7] Con opciones entiéndase la variedad de valores diferentes que pueden tomar los distintos hiperparámetros del modelo como el tamaño de los filtros, *stride*, etc.

principalmente en reducir lo más posible la potencia computacional necesaria para el algoritmo. Busca una arquitectura muy sencilla a pesar de que esto le haga sacrificar algo de precisión y rendimiento. Esto lo consigue en gran medida gracias a la utilización de convoluciones separables por profundidad.

2.4. Entrenamiento de Redes Neuronales

El objetivo de minimizar la función de error que nos permita ajustarnos a los ejemplos del conjunto de entrenamiento no es tarea sencilla en Deep Learning. No lo es debido a la gran cantidad de datos que se manejan y necesitan para que estos algoritmos funcionen correctamente. Eso por no hablar de todos los millones de parámetros que han de ser actualizados en cada paso del proceso. En este apartado hablaremos de algunos de los aspectos clave del entrenamiento.

Inicialización de pesos:

La inicialización de los pesos juega un papel importante en el entrenamiento de las redes.

Lo que está claro es que no se pueden inicializar a 0 los pesos de una red ya que esto no hace otra cosa que convertir el modelo en uno equivalente a un modelo lineal. Esto sucede porque el gradiente de la función de coste con respecto a cada peso de una misma capa es siempre el mismo, provocando por consiguiente que todos los pesos de esa capa acaben en cada iteración con los mismos valores. Por tanto, se suele optar por inicializar los pesos aleatoriamente a valores cercanos a 0.

Sin embargo, una de las mayores preocupaciones en las redes profundas es la tendencia que tienen los gradientes a desvanecerse (hacerse cada vez más pequeños) conforme se avanza en la propagación hacia atrás (*Backpropagation*). También puede suceder exactamente lo contrario: que los gradientes tiendan a ser cada vez más y más grandes. En cualquier caso, cualquiera de estas situaciones pone en serio peligro el correcto entrenamiento de la red, ocasionando serios problemas en la convergencia.

Una de las medidas adoptadas para luchar contra este tipo de problemas es el de la elección de la inicialización de los pesos. Se suele tratar de inicializar los pesos guiándose por el tipo de función de activación que se va a emplear en la red, pues están bastante influenciadas por los pesos que se inicializan. En el caso de *ReLU*, se sigue una distribución normal de media 1 y varianza igual a $\sqrt{\frac{2}{tamaño\ de\ la\ capa\ anterior}}$. Para el caso de tanh[8] se utiliza una inicialización similar: *Xavier initialization*[8].

[8] La función tanh es otra de las funciones de activación más usadas en Deep Learning.

Regularización:

La regularización es muy utilizada en Deep Learning. Regularizar es introducir una modificación en el modelo para restringir que el modelo se ajuste en demasía a los ejemplos de entrenamiento.

Mediante la regularización se trata de evitar crear un modelo que no generalice lo suficientemente bien debido a que haya sido demasiado condicionado por los ejemplos de entrenamiento (sobreentrenamiento). Un ejemplo de regularización es la L2 o L1, que buscan actualizar la función de coste añadiendo un nuevo término regularizador. Este término pretende que el algoritmo alcance el equilibrio entre reducir lo máximo posible el error original y que el modelo sea lo más simple posible.

Otra técnica de regularización es la Dropout[9] y es una de las más usadas (aunque no en capas convolucionales). Dropout asigna a cada neurona de la red una probabilidad de ser 'apagada' o no. Así, en cada iteración, trabaja con una serie de neuronas diferentes y por consiguiente no genera dependencias hacia ninguna de ellas (pues todas en algún momento no se usarían para entrenar).

Batch size:

Los datos que hemos de pasar a la red para entrenarla son excesivos como para pasarlos todos a la vez a través de la red. Lo que se suele hacer es dividir estos datos en paquetes de datos más pequeños (subconjuntos del conjunto original) y que sean estos paquetes los que se propaguen por la red.

En este contexto, el concepto *Época* se emplea para referirse al paso por la red de todo el conjunto de datos original. Cuando todos los paquetes o *batches* hayan pasado una vez por la red se dirá que se ha consumido una época. Además, como el proceso de minimización es iterativo, se necesitan varias épocas para entrenar la red. Es por tanto un hiperparámetro más del modelo el valor del tamaño de los *batches* a utilizar. Generalmente, cuando el conjunto de *train* no es muy grande (<=2000) conviene que el tamaño del *batch* sea igual al del conjunto de *train* y cuando no, típicamente se emplean potencias de 2 como tamaño de los *batches*. Lo primordial es que los *batches* quepan en la memoria CPU/GPU para así beneficiarse de la ganancia en eficiencia.

Learning Rate:

El *learning rate* α o ratio de aprendizaje es probablemente el aspecto más relevante en la mayoría de métodos de optimización empleados en Deep Learning.

Es un hiperparámetro que controla cómo de rápido se avanza durante el proceso de aprendizaje. El valor elegido para esta tasa marca las diferencias entre un buen algoritmo y uno no tan bueno. Lo difícil a la hora de escoger este valor es que se trata de conseguir una convergencia rápida (buscamos un método que en primer lugar converja, y en segundo lugar, que lo haga rápido).
Si escogemos un valor muy grande de α entonces el método oscilará en torno al punto objetivo (punto de error mínimo) pero no convergerá, y si por el contrario

escogemos un $\boldsymbol{\alpha}$ demasiado pequeño, el aprendizaje resultará excesivamente lento.

Lo que ocurre es que para cada problema el $\boldsymbol{\alpha}$ idóneo es diferente y no sabemos cuál es. Algunas técnicas proponen que dicho valor $\boldsymbol{\alpha}$ no sea prefijado, sino que decrezca conforme nos acercamos al mínimo (*Learning rate Decay*).

Métodos de optimización:

Anteriormente cuando hemos hablado de las Redes Neuronales, hemos mencionado el algoritmo del descenso por gradiente como el método empleado para minimizar la función de error. Sin embargo, este no es el único método de optimización que se usa en Deep Learning.

- SGD

  Stochastic Gradient Descent (Descenso estocástico por gradiente) es un caso particular del método de descenso por gradiente. *SGD* consiste en aplicar descenso por gradiente a un solo ejemplo por iteración. Estamos ante la situación de aplicar *GD* por *batches* de tamaño 1. Al empezar una nueva época lo que hacemos es remezclar aleatoriamente los ejemplos del conjunto de entrenamiento.

- Momentum

  Una modificación del descenso por gradiente es incluir inercia o momento. Básicamente lo que hace es recordar el incremento aplicado a los pesos en cada iteración y determinar la siguiente actualización como una combinación lineal entre el gradiente y esos incrementos anteriores. De esta forma se suavizan las oscilaciones en torno al mínimo durante la convergencia del algoritmo de descenso por gradiente.

- RMSprop

  RMSprop pretende, al igual que *Momentum GD*, suavizar las oscilaciones durante la convergencia del método. Lo que tiene en cuenta para ello es la necesidad de ajustar el *Learning Rate* y hacerlo automáticamente. En cada actualización de los pesos, escoge un valor diferente de este ratio para cada peso. Se trata pues de un método adaptativo ya que el proceso de aprendizaje se va adaptando sobre la marcha.

- Adam

  Por último, *Adam* es un optimizador que combina los 2 métodos anteriores: *Momentum SGD* y *RMSprop*.

Batch Normalization:

Una técnica que se emplea mucho en Deep Learning para agilizar el proceso de aprendizaje es la de *Batch Normalization*[10].

Batch Normalization consiste en estandarizar/normalizar las entradas a cada capa de la red por cada *batch.* Esta idea surge de contemplar que una de las razones por las que el proceso de aprendizaje en redes profundas se vuelve tan tan complicado es que la distribución de las entradas a las capas ocultas podría cambiar con cada *batch* tras la actualización de los pesos. Y si es frecuente preprocesar las entradas a la red normalizándolas, por qué va a ser diferente a la hora de hacerlo con las activaciones que entran a otras capas.
De hecho, este continuo cambio en la distribución de las activaciones ralentiza considerablemente el entrenamiento ya que cada capa debe aprender a adaptarse a nuevas distribuciones de datos cada vez. La solución de realizar *Batch Normalization* tiene como efecto estabilizar el proceso de aprendizaje y reducir dramáticamente el número de épocas requeridas para entrenar aceptablemente una red profunda.

2.5. Herramientas/Frameworks

Python – Numpy:

Python es sin duda uno de los lenguajes más utilizados en Machine Learning si no el que más. Es un lenguaje de programación interpretado y multiparadigma, ya que soporta orientación a objetos, programación imperativa y también programación funcional. Además, es multiplataforma.

La razón de que sea tan popular en este ámbito es la simplicidad y, sobre todo, la grandísima variedad de librerías que dispone. También ayuda por supuesto que sea de código abierto y su gran capacidad en el manejo de datos.

Una de las extensiones de Python más populares es *NumPy*, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca matemática de alto nivel para operar con esos vectores. De hecho, esta herramienta resulta de gran utilidad cuando se trabaja con Redes Neuronales, que operan con ingentes cantidades de matrices y datos.

Cabe destacar también la biblioteca *MatPlotLib*, que genera gráficos a partir de datos contenidos en arrays, por lo que su uso también está ligado a *NumPy*.

Keras:

Keras es una biblioteca de código abierto de Redes Neuronales escrita en *Python*. Está especialmente diseñada para facilitar una rápida experimentación en Deep Learning. Permite crear prototipos rápidos de redes profundas y llevar a cabo su entrenamiento de forma cómoda y rápida.

Es un framework de alto nivel y está diseñado para correr sobre otros frameworks de más bajo nivel como *TensorFlow* o *Theano*.

Tensorflow:

TensorFlow es una biblioteca desarrollada por *Google* de código abierto para aprendizaje automático. Esta librería de computación matemática ejecuta de

forma rápida y eficiente gráficos de flujo. Estos gráficos están formados por operaciones matemáticas representadas sobre nudos y cuyas entradas y salidas son vectores multidimensionales de datos(tensores).

Keras y *Tensorflow* combinan tan bien ya que *Keras* nos permite abstraernos de estas operaciones vectoriales gracias a su interfaz sencilla y sintaxis homogénea.

Google Colab:

Colaboratory de *Google* es un entorno gratuito que no requiere configuración y se ejecuta completamente en la nube. Este entorno interactivo nos permite escribir y ejecutar código desde el navegador, pudiendo combinarlo con texto, imágenes etc (*Jupyter Notebook*).

La gran ventaja de trabajar en este entorno es la posibilidad de ejecutar código en GPU potentes que ofrece *Google* (*Google* nos proporciona recursos computacionales para poder ejecutar nuestro código en ellos; el código se ejecuta en 'la nube'). La importancia de esto es brutal, pues se traduce en un ahorro más que considerable de tiempo (en Deep Learning se trabaja con grandes matrices que requieren de potentes GPU para procesarlas rápido, por lo que si no se dispone de dichas GPU, *Google Colab* es una buena opción).

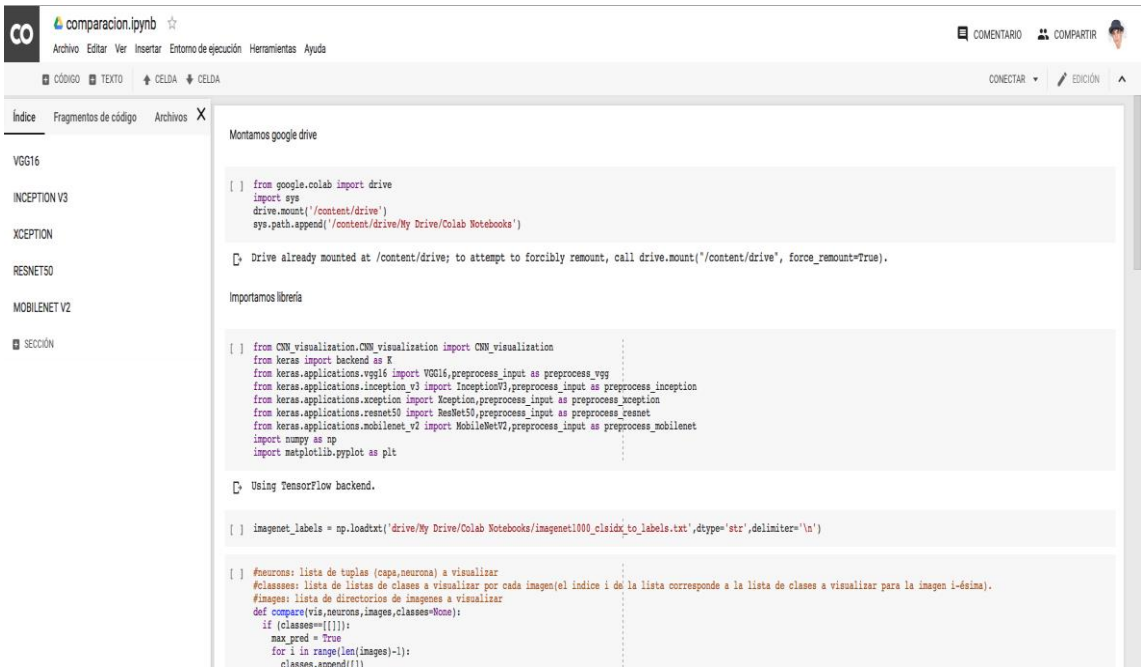


Figura 14. Vista del entorno de ejecución *Google Colab*.

3. Métodos de visualización de Redes Neuronales Convolucionales

3.1. Visualización de CNNs

El notable rendimiento de las CNNs[9] se debe principalmente a su destacada capacidad para aprender características de las imágenes de entrada. Sin embargo, estas características aprendidas son difíciles de identificar e interpretar por un humano, lo que deriva en un problema para entender el funcionamiento del mecanismo interno de las redes. Para mejorar esta falta de interpretabilidad, se utiliza el mecanismo de visualización de CNNs, que es un método cualitativo que sirve para analizar el funcionamiento interno de la red mediante la traducción de sus características internas a patrones visuales perceptibles (en otras palabras, conseguir una representación visual de las características de una imagen que está aprendiendo la red).

La visualización ayuda enormemente a interpretar las características internas de las CNNs, pues éstas se basan en el sistema córtex visual humano. El córtex visual humano procesa las características de los objetos de manera jerárquica a través de diferentes zonas neuronales[11], es decir, primero en el área neuronal de más bajo nivel visual se detectan las características visuales más básicas como bordes o líneas y posteriormente en las más altas se empiezan a detectar características más complejas como formas y objetos, así hasta llegar a detectar imágenes completas. Por su parte, las CNNs comienzan extrayendo pequeñas características como bordes en la primera capa y poco a poco van extrayendo características más complejas (formas y objetos parciales) en las capas más profundas hasta llegar a la clasificación final (FC[10] layers). Así pues, comparando las neuronas cerebrales visuales de un humano y las de las CNNs, la visualización nos permite observar las funcionalidades de cada componente en las CNNs (qué es lo que aprende cada neurona, etc).

Esto en definitiva es muy importante para saber y entender qué es lo que está haciendo nuestro modelo y cómo está tomando decisiones en las predicciones. Entender cómo trabaja y funciona nuestro modelo también nos permite averiguar fallos en el mismo e intuir el por qué fallan o explicar sus decisiones a clientes y nos ayuda a elegir los hiperparámetros más adecuados o la arquitectura ideal para el problema concreto que tenemos.

Ahora bien, existen diversos métodos de visualización de CNNs. Todos ellos tratan de resaltar en una imagen las características que estimulan en mayor medida a una neurona o conjunto de neuronas determinadas, que son la que pretendemos visualizar. Visualizar una neurona no es otra cosa pues que observar qué patrones de una imagen consiguen que ésta se active más

[9] CNNs es la abreviatura empleada para referirse a las Redes Neuronales Convolucionales (Convolutional Neural Networks en inglés).

[10] FC es la abreviatura empleada para referirse a las capas *fully connected* de una red neuronal.

intensamente, o, dicho de otra forma, observar aquellos patrones que provocan una mayor activación en la neurona. De esta forma conseguimos fijarnos en cuál es el campo receptivo de dicha neurona: ante qué estímulos es sensible. Podremos averiguar si la neurona está detectando bordes, formas, objetos, etc.

Antes de continuar, conviene especificar a qué nos referimos por 'neurona' en una red convolucional. Con neurona nos referimos al componente de la red que computa un mapa de activaciones en función de la entrada que recibe. Es decir, una neurona está asociada a un único filtro de pesos (también a un término independiente o bias) y a una función de activación o no linealidad, que le permiten construir una salida en forma de mapa o matriz de activaciones a partir de otros mapas de activaciones que toma como entrada, provenientes de las neuronas de la capa anterior. Esta estructura organizativa es la que nos permite entender por qué las neuronas más profundas computan características cada vez más complejas. Lo hacen porque operan sobre los resultados procesados por todas las neuronas de la capa anterior, con lo que el nuevo resultado que arrojan es una combinación de los resultados anteriores, siendo por consiguiente un resultado más complejo. Así sucesivamente, con lo que, si comenzamos por neuronas que detectan bordes o colores, podremos acabar en neuronas que detectan formas complejas como por ejemplo caras en base a las anteriores detecciones.

Tras esta aclaración, visualizar una neurona de una capa convolucional significa ver qué características (patrones visuales) tienen impacto en el mapa de activaciones asociado a dicha neurona.

Fijarnos en las neuronas de capas intermedias está muy bien ya que tal y como acabamos de comentar sirven para diseccionar la red e incluso descubrir aspectos nuevos, cosa que es de gran ayuda para la investigación. Sin embargo, suele ser mucho más habitual tener por objetivo visualizar neuronas de salida, es decir, las que computan la clasificación final. Esto es así porque lo que realmente nos interesa de estos algoritmos (las Redes Neuronales) son sus predicciones, aislándonos del proceso que sigan para lograrlas con tal de que dichas predicciones sean certeras.

Visualizar predicciones sirve para comprobar en qué se ha fundamentado la decisión, dicho de otra manera, observar en qué partes de la imagen de entrada se ha fijado la red para predecir tal o cual cosa. Y en vistas a que lo que nos interesa de la red son sus predicciones, esto es fundamental para cerciorarse de que la red haya aprendido correctamente (Ejemplo: si tenemos una neurona encargada de clasificar una imagen como 'imagen de pingüino' o 'no de pingüino' y la entrenamos con todo imágenes de pingüinos en el Ártico, es muy probable que la red obtenga una precisión muy buena en *test*[11] a pesar de que en lo que realmente se esté fijando no sea en el pingüino en sí mismo, sino en la nieve que le rodea, el paisaje etc).

[11] Es probable que la red obtuviese una buena precisión en *test* ya que los pingüinos generalmente se suelen encontrar en ese tipo de hábitats, luego por tanto las imágenes del conjunto de test serían similares a las de *train* y la red las clasificaría bien a pesar de que realmente no haya aprendido qué es un pingüino.

Asimismo, puede suceder que no nos convenza una decisión determinada del clasificador y queramos obtener una explicación de la misma. O también puede ocurrir que un experto se ayude de estos algoritmos para tomar decisiones y por tanto necesite siempre una prueba que soporte dicha decisión o que se necesite explicar la decisión adoptada por el clasificador a un cliente, etc. En definitiva, visualizar neuronas de salida o decisiones es lo más importante y en lo que se centran la mayoría de métodos de visualización existentes.

La diferencia entre visualizar neuronas de salida y visualizar neuronas de capas ocultas radica en que las primeras son de capas *fully connected* y por tanto su salida computada es un valor unidimensional, mientras que las segundas son neuronas convolucionales y calculan un valor multidimensional (una matriz o mapa de activaciones). A priori, ser capaces de ver qué patrones de una imagen suscitan un mayor valor de activación es más fácil en neuronas de salida que en neuronas intermedias de capas convolucionales, pues, mientras que las primeras consisten de un único valor, las segundas consisten de una matriz de valores. Definir qué es un aumento de la activación en un mapa de activaciones requiere emplear algún tipo de función de agregación (máximo, media, suma, etc).

Se puede abordar el problema de visualización desde dos puntos de vista. El primero sería generando la imagen que represente las características que más estimulen la neurona objetivo que queremos visualizar y la segunda, para una imagen concreta destacar aquellas regiones o patrones en los que se está fijando. Son dos puntos de vista distintos a la hora de afrontar el problema, aunque suele ser más común y práctico el segundo de ellos (generar una visualización para cada imagen concreta que entra a la red). La razón de que este segundo punto de vista sea el más utilizado es la que se ha expuesto justo antes: generalmente nos interesa más entender por qué la neurona predice tal o cual cosa que hallar cuál es su entrada preferida.

De ahora en adelante se exponen algunos de los métodos más destacados de visualización neuronal en Redes Neuronales Convolucionales. La mayoría de ellos son del mencionado segundo tipo/punto de vista (visualización para cada imagen concreta). Sin embargo, antes comenzaremos viendo alguno de los primeros (visualización de lo que más estimula una neurona). Por último, finalizaremos viendo algunos métodos de visualización de conjuntos de neuronas (visualizan grupos de neuronas en vez de neuronas por individual).

3.2. Activation Maximization

Uno de los primeros métodos surgidos para visualizar neuronas es el propuesto por *Erhan et al*. en 2009[12].

Este método es de los que buscan generar una imagen que maximice la activación de la neurona específica que tratamos de visualizar. Así, lo que en realidad estamos consiguiendo es la imagen preferida de la neurona, que nos indica qué características ha aprendido ésta. El objetivo es ver estas características plasmadas en la nueva imagen generada. Para ello, se parte de

una imagen aleatoria e iterativamente se actualizan los valores de cada uno de los píxeles de dicha imagen con tal de que la nueva imagen resultante produzca una mayor activación que la anterior en la neurona considerada.

Lo cierto es que aunque este algoritmo fuera propuesto por *Erhan et al.*[12], no se utilizó concretamente en Redes Neuronales Convolucionales hasta 2014, cuando *Simonyan et al.*[13] lo utilizó para maximizar la activación de neuronas en la última capa de CNNs. Además, a raíz de este algoritmo ha surgido el resto de esta familia de métodos que buscan generar la imagen que más estimule una neurona. Estos algoritmos tratan de mejorar la interpretabilidad y comprensibilidad de los patrones visuales obtenidos en la imagen generada. Algunos hacen que dichos patrones sean más reconocibles mediante distintas técnicas de regularización[14] y otros mediante distintas técnicas de optimización como algoritmos evolutivos[15] o redes generadoras de imágenes[16].

El algoritmo principal consiste en encontrar aquella imagen $\boldsymbol{x}'$ que maximice la activación de la neurona objetivo: $\boldsymbol{x}' = \underset{x}{\operatorname{argmax}}\, a_{i,l}(\theta, x)$, donde θ representa el conjunto de parámetros de la red y $a_{i,l}$ es la neurona objetivo.
El proceso se puede dividir en cuatro fases diferentes:

1) Se toma una imagen aleatoria (píxeles con valores aleatorios) $x = \boldsymbol{x_0}$ como entrada inicial a la red
2) Se calculan los gradientes de la activación $a_{i,l}$ con respecto a cada píxel de la imagen mediante *backpropagation* pero sin actualizar en ningún momento los pesos de la red (únicamente aplicando regla de la cadena)
3) Cada píxel de la imagen se actualiza según: $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha \frac{\partial a_{i,l}(\theta, \boldsymbol{x})}{\partial \boldsymbol{x}}$, donde α representa tamaño de cada paso en este ascenso por gradiente. Esta fase se repite conjuntamente con la 2) de forma iterativa.
4) Cuando el procedimiento iterativo haya convergido o casi lo haya hecho, se para y se obtiene la imagen específica $\boldsymbol{x}'$que se andaba buscando.

En resumen, el algoritmo no es otra cosa que aplicar ascenso por gradiente sobre la función de activación $a_{i,l}$ para la imagen de entrada a la red, de modo que se termine con una imagen que maximice la activación de la neurona a visualizar. Nótese que aplicar ascenso por gradiente sobre la función de activación $a_{i,l}$ es lo mismo que aplicar descenso por gradiente sobre la función $-a_{i,l}$.

El gran problema de este método es que conforme queremos visualizar neuronas de capas más profundas, los patrones visuales generados tienden a volverse irrealistas y no interpretables. Como ya se ha mencionado, una solución a esto es la regularización. Se ha experimentado con muchos métodos regularizadores que han demostrado mejorar la interpretabilidad de las imágenes generadas. El nuevo esquema del método *Activation Maximization* quedaría como: $\boldsymbol{x}' = \underset{x}{\operatorname{argmax}}(\, a_{i,l}(\theta, x) - \lambda(x)\,)$, donde λ es el parámetro regularizador y al que podemos otorgar la importancia que consideremos con respecto a la maximización. λ generalmente actúa evitando que existan píxeles en la imagen de entrada con valores extremos, haciendo siempre que se mantengan bajo un rango determinado. Esto consigue que se alcance un equilibrio entre lo realista

que parezca la imagen generada a ojos de un humano y lo mucho que maximice la activación de la neurona visualizada. Podemos ver algún ejemplo de resultado en la *Figura 15*.

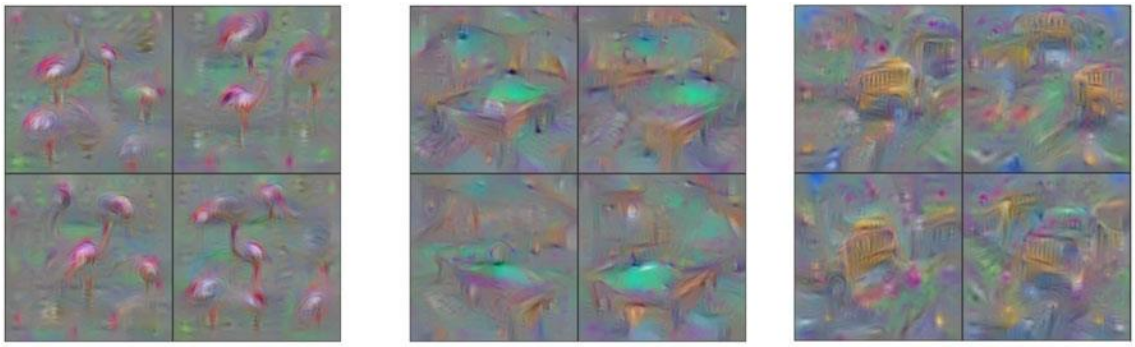


Figura 15. Ejemplos de resultados de la técnica para las clases*: Flamenco, Mesa de billar y Autobús escolar.*

3.3. Redes Neuronales Deconvolucionales: DeConvNets

La idea de este método es encontrar los patrones de la imagen de entrada que activan una determinada neurona en una capa de una CNN, o, dicho de otra forma, revelar los estímulos entrantes que más excitan a cada mapa de activaciones en particular. Para ello lo que hace es mapear esas características internas de la red al espacio de entrada (proyectar el mapa de activaciones de las neuronas de vuelta al espacio dimensional de la imagen) y así poder mostrar qué patrones de la entrada son los que originalmente causaron esa activación en el mapa de características. Para lograr este propósito se emplean Redes Neuronales Deconvolucionales (*DeconvNets*--DNs).

Las DNs fueron por primera vez introducidas por *Zeiler et al.* en el año 2011[17]. Estas redes actúan exactamente igual que las CNNs solo que en orden inverso. Así pues, los componentes más característicos de las CNNs como las *pooling layers* (capas de agrupación), las convoluciones o las funciones de no linealidad están presentes también en las DNs, pero dispuestas en inverso orden (y las operaciones son las "inversas"[12]). El objetivo es deshacer el proceso realizado por las CNNs, así que si éstas lo que hacen es mapear píxeles a características, las DNs hacen justo lo contrario. De hecho inicialmente fueron propuestas como una forma de aprendizaje no supervisado[17].

El proceso para examinar una CNN es el que sigue:

1) Primero se pasa una imagen a la CNN y se calculan todos sus mapas de activaciones
2) Se selecciona el mapa de activaciones de la neurona que se quiere visualizar y el resto de mapas se establecen a 0.

[12] En el sentido más riguroso, las operaciones que realiza la *DeconvNet* no son las inversas a las de la red original, pero sí que simulan serlo.

3) Finalmente se pasan estos nuevos mapas a la DN, que tratará de reconstruir la actividad en las capas más bajas que hayan dado lugar a la activación analizada. Este proceso termina cuando se alcanza el espacio de entrada, es decir, cuando la activación ha sido proyectada al espacio dimensional de la imagen.

Ya solo quedan por definir los componentes de cada capa de la DN:

- Unpooling: la operación *max pooling* no es invertible pero sí que podemos llegar a obtener una aproximación si recordamos las posiciones de los valores máximos de cada región de *pooling* (*switches*) y posteriormente en la DN usamos estos *switches* para reconstruir la capa anterior situando los valores correspondientes a cada región *pooling* en el lugar que ocupaban exactamente antes de la operación de *pooling*. El resto de valores de la región se suelen completar con 0 (no se recuperan).

- Rectification: La función no lineal que se usa es la *ReLU*, que siempre nos asegura que nuestros mapas de características sean positivos, ya que la imagen de esta función es $>= 0$. Así pues, para obtener reconstrucciones válidas de cada capa (que, como hemos mencionado, han de ser siempre positivas) pasamos la señal reconstruida a una *ReLU*.

- Deconvolution/Filtering: Los mismos filtros aprendidos y utilizados por la CNN para convolver los mapas de características de la capa anterior son también utilizados por la DN para revertir la convolución pero de diferente manera. En primer lugar, ya no se aplican sobre las salidas de la capa anterior sino sobre los mapas rectificados (después de aplicar *ReLU*) y, en segundo lugar, se aplican traspuestos (volcados horizontal y verticalmente). Matemáticamente se puede demostrar que el resultado de deshacer una convolución (deconvolución) tiene la misma dimensión que la entrada a esa convolución (ver *Figura 16*).

De acuerdo a su modo de proceder, no es poca la gente que opina que el nombre escogido para este método es poco acertado. Opinan que en lugar de “Redes Neuronales Deconvolucionales” se deberían llamar de otra manera ya que en ningún momento realizan la operación matemática de deconvolución (que consistiría en recuperar la entrada original de una convolución partiendo de su resultado). En su lugar lo que realizan es una convolución traspuesta, que sí les permite recuperar el tamaño de la entrada original.

El resumen del proceso seguido durante el método de acuerdo al documento original[18] se puede apreciar en la *Figura 17*. En la figura podemos ver en detalle la operación de *Unpooling*, que es uno de los puntos más remarcados del documento.

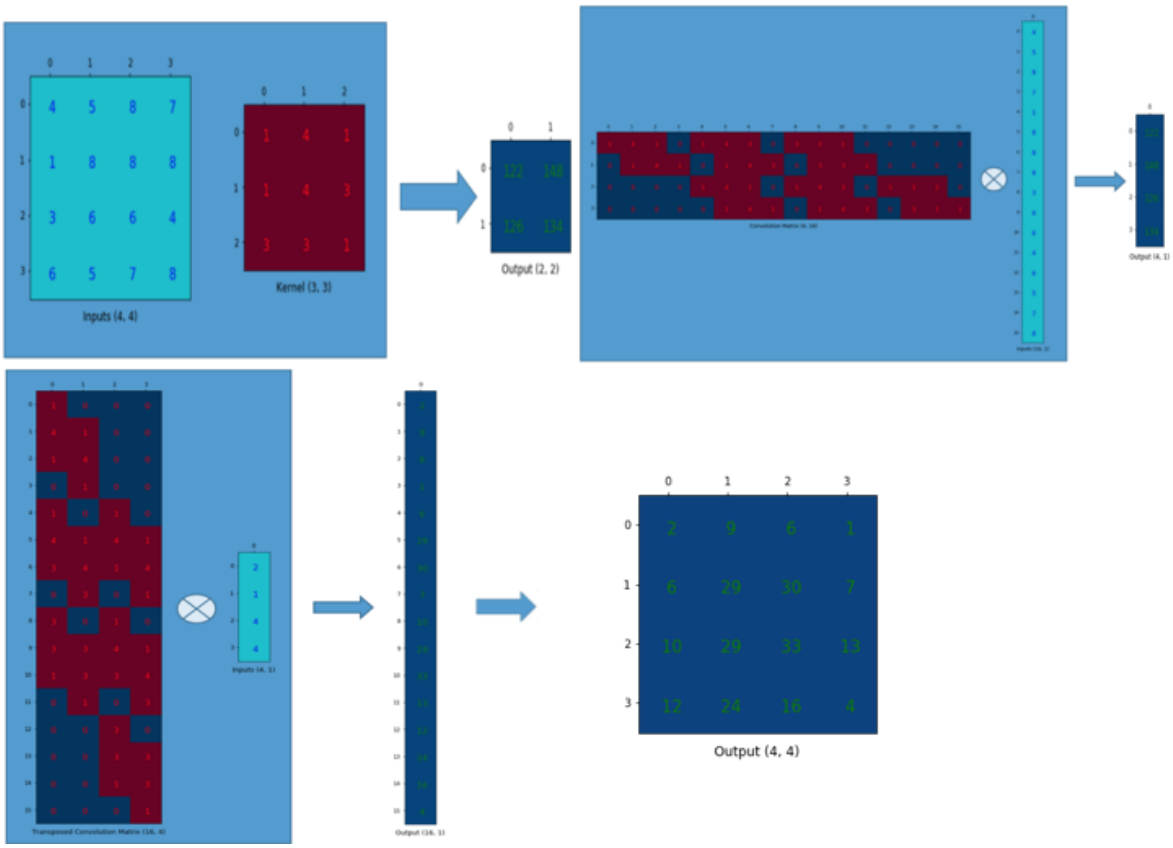


Figura 16. Se muestra cómo la operación de convolución puede ser representada como una operación producto matricial. De esta forma, aplicar el producto de la matriz traspuesta por el resultado de la operación original da como resultado una matriz de dimensiones iguales a la entrada original. Arriba a la izquierda, podemos ver la operación de convolución original y, a su derecha, la misma operación pero dispuesta en forma de producto matricial. Tomando la traspuesta de la matriz que representa al filtro y aplicando el producto matricial de esa traspuesta por el resultado obtenido, obtenemos una matriz unidimensional que, si la redimensionamos, da lugar a una matriz del mismo tamaño que la de la entrada original (abajo en la figura).

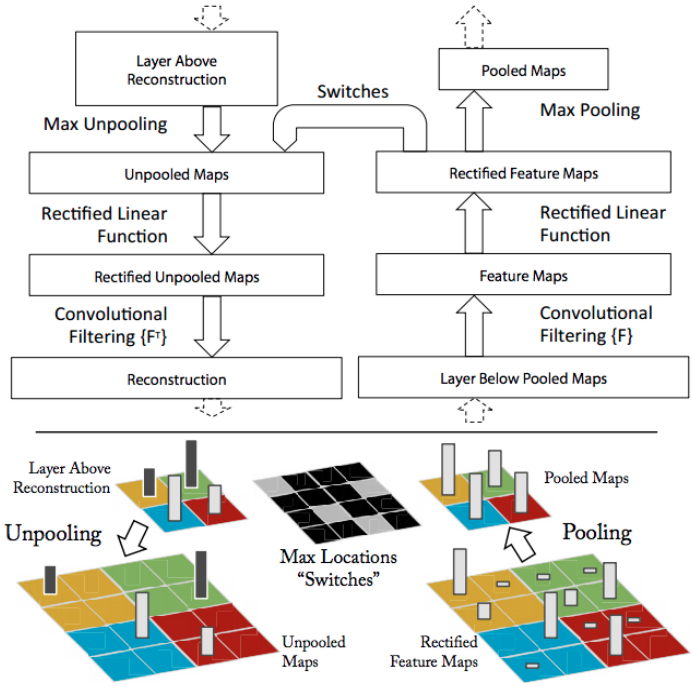


Figura 17. Una capa de la red original y su correspondiente en la deconvolucional. La deconvolucional reconstruye una versión aproximada de los mapas de activación de la capa inferior.

3.4. Saliency Maps

Esta técnica, al igual que la anterior, computa los patrones de entrada que maximizan la activación de una neurona para una imagen en particular. La idea es resaltar la importancia de cada píxel en la decisión mediante un mapa de prominencia (*saliency map*). El método fue propuesto por *Simonyan at al*. en 2014[13] y se centra en visualizar neuronas de salida, de clasificación.

Básicamente el método consiste en el cálculo del gradiente de la activación de la neurona a considerar con respecto a la imagen de entrada. Esto intuitivamente calcula cuánta es la variación en la activación de dicha neurona cuando la entrada cambia, aunque muy muy poco. Así podemos saber lo sensible que es esa neurona con respecto a cada píxel de la imagen de entrada (qué píxeles de la imagen necesitan cambiar lo menos posible para afectar a la activación lo máximo posible) y podremos visualizar los patrones que representan esos píxeles que más afectan a la neurona.

Si nos centrásemos en las neuronas de la primera capa del modelo o si el modelo fuese lineal[13] sería fácil ver que los pesos w con los que se combina linealmente

[13] En el documento original[13] se trata todo desde la perspectiva de visualizar neuronas de clase(de salida) y por tanto cualquier explicación o supuesto se da en base a un cambio en el enfoque de la estructura del modelo y no de la visualización. Así para visualizar una neurona de la capa l se ha de asumir que dicha neurona es de salida (el modelo tiene **l** capas). Haciendo esta asunción, es fácil ver que se puede generalizar el método para visualizar neuronas intermedias.

la imagen de entrada I_0: $a = wI_0 + b$, son los que definen la importancia que le corresponde a cada píxel de la imagen para la activación. Sin embargo, para modelos con más de una capa esto no sirve para visualizar sus salidas, ya que la función que da lugar a una activación de una capa profunda de una CNN es no lineal y muy compleja. En estos casos, directamente se calcula el gradiente de la activación con respecto a la imagen. Este gradiente se puede demostrar que hace las veces de w en el ejemplo anterior (aproximadamente). Esta demostración se puede realizar gracias a la expansión de Taylor de primer orden: $a \approx wI_0 + b$, donde w es la derivada de a con respecto a la imagen I en el punto I_0: $w = \frac{\partial a}{\partial I}\Big|_{I_0}$.

Finalmente, el mapa de prominencia se genera calculando el valor absoluto del resultado de los gradientes si la imagen está en escala de grises (únicamente tiene un canal/dimensión) o como el máximo a través del canal de color de los valores absolutos de los gradientes si la imagen es RGB.

En la misma publicación[13], se pone de manifiesto la estrecha conexión entre la visualización basada en el gradiente y la arquitectura *DeconvNet*. Se demuestra que el método *Saliency Maps* y *DeconvNets* son equivalentes.
Esto significa que una *DeconvNet* se corresponde con el *backpropagation* a través de la red original asociada a esa *DeconvNet*. Para demostrar que ambos métodos realizan los mismos cálculos, se procede para cada uno de los componentes de cada capa de una DN:

- Para la capa convolucional, el gradiente lo calcularíamos como $\frac{\partial a}{\partial X_n} = \frac{\partial a}{\partial X_{n+1}} * f$, mientras que la reconstrucción de la capa anterior en una DN la calcularíamos como $R_n = R_{n+1} * f$.

- Para la capa *ReLU*, la expresión para el gradiente vendría dada por: $\frac{\partial a}{\partial X_n} = \frac{\partial a}{\partial X_{n+1}} \mathbf{1}(X_n > 0)$, mientras que la reconstrucción *ReLU* en una DN sería: $R_n = R_{n+1} 1(R_{n+1} > 0)$.
  Este paso es un poco diferente ya que mientras que *Saliency Maps* aplica la operación booleana sobre los valores de la entrada a la capa X_n, *DeconvNets* lo hace sobre la reconstrucción R_{n+1}, que es la reconstrucción de la salida de la capa.

- Por último, para la capa de *Pooling* (*max-pooling*), en *Saliency Maps* únicamente se propagan los gradientes que pertenecen a las posiciones donde el valor de las activaciones era el máximo en la región de *pooling*. En *DeconvNets*, se almacena dicho valor durante el *forward-pass* (*switches*) y luego se emplea de igual modo durante la reconstrucción.

Se puede concluir que, salvo por la capa de activación *ReLU*, la reconstrucción aproximada de un mapa de activaciones R_n es equivalente a $\frac{\partial a}{\partial X_n}$. Por tanto, el método de visualización basado en gradiente (*Saliency Maps)* se puede ver como una generalización del de DNs ya que abarca más capas que éste

(también trata las capas de clasificación que son *fully-connected* y no convolucionales), aunque realmente ambos métodos no son idénticos, pues difieren en el *backpropagation* de los gradientes por la capa de activación *ReLU*.

3.5. Guided Backpropagation

Este método presentado por *Springenberg et al*. en 2015[19] se basa también en realizar *backpropagation* para visualizar las neuronas, pero difiere en un detalle con el anterior de *Saliency Maps:* en el *backpropagation* de la capa *ReLU*, adicionalmente se suprimen los gradientes negativos(se establecen a 0).

Una vez discutida en el apartado anterior la diferencia entre los dos métodos *DeconvNets* y *Saliency Maps,* resulta evidente que este método *Guided Backprop* no es sino que una combinación de los dos. Y es que lo que propone es, en vez de dejar de propagar (establecer a 0) exclusivamente los valores correspondientes a entradas negativas de los gradientes provenientes de capas más altas como hace *DeconvNets* o dejar de propagar exclusivamente los valores correspondientes a entradas de activaciones negativas como hace *Saliency Maps*[14], suprimir las entradas para los que al menos uno de estos valores es negativo. Es decir, deja de propagar tanto los gradientes negativos como los gradientes que ocupan la posición de activaciones negativas.
Esto lo que consigue es prevenir el flujo de gradientes negativos, que se corresponden con neuronas que decrementan la activación de la neurona de más alto nivel (de una capa más alta) que pretendemos visualizar. Además, también se evita que se auto-anulen caminos con influencia positiva con caminos con influencia negativa durante el *backpropagation,* dando así lugar a imágenes no tan ruidosas.

Los tres métodos de visualización neuronal para imágenes concretas que hemos visto hasta ahora difieren en su modo de tratar la capa de no linealidad *ReLU* durante la etapa de propagación de gradientes. Esta diferencia queda plasmada en la *Figura 18*. Ahí podemos ver cómo este método *Guided Backpropagation* es una mezcla de los dos anteriores.

[14] Cuando empleamos el término Saliency Maps para referirnos al funcionamiento del algoritmo, en realidad estamos pensando en el procedimiento habitual de *backpropagation.* De hecho, se suele referenciar así a este método (Con el término *Backpropagation*).

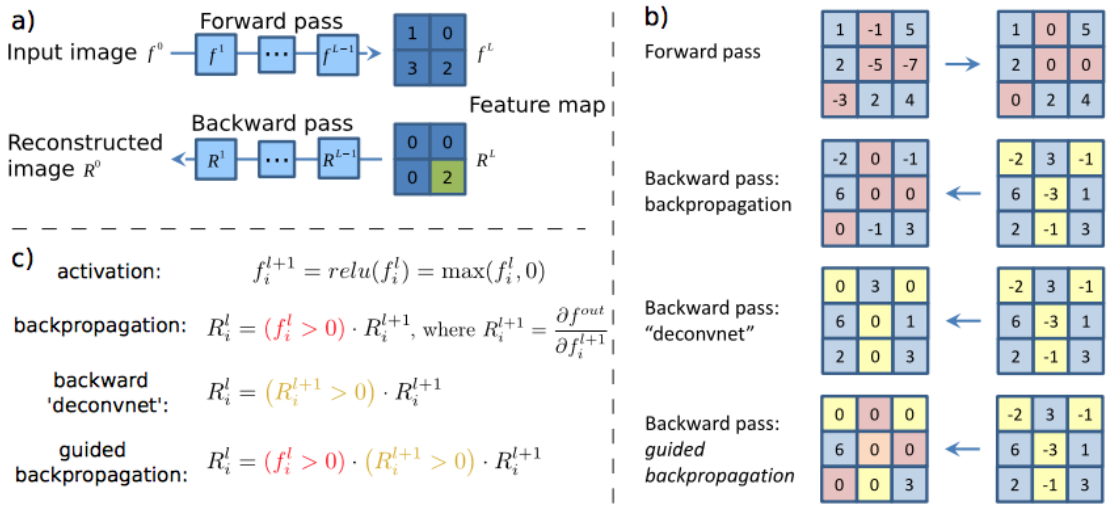


Figura 18. Esquema de la diferencia de los tres métodos: *DeconvNets*, *Saliency Maps* o *Backpropagation* y *Guided Backpropagation*. Se representa el paso por la capa de activación ReLU durante la propagación de gradientes. La imagen pertenece al artículo original[19].

3.6. Deep Taylor Decomposition

Para explicar las predicciones en términos de las variables de entrada (píxeles), existe otro método propuesto por *Montavon et al.* a finales de 2015[20] que consiste en descomponer la salida de la red neuronal en las variables/características de entrada.

Este método, al igual que los anteriores, pretende determinar qué características de entrada han contribuido a la clasificación (qué píxeles de la imagen de entrada son más importantes para la predicción) y mostrar estos resultados mediante un mapa de calor (*heatmap*). Sin embargo, difiere en la forma de hacerlo: mientras que las anteriores técnicas abordan el problema buscando las características de entrada que maximizan la salida para esa clase (qué píxeles hacen que pertenezca más/menos a la clase), esta técnica busca aquellas características que hacen que la imagen pertenezca a la clase. El matiz es lo que hace que en lugar de explicar la decisión de la red midiendo efectos diferenciales, podamos explicarla en su conjunto. De ahora en adelante, denominaremos estos dos tipos de técnicas como técnicas sensitivas y técnicas descomposicionales o de descomposición.

En realidad, técnicamente los métodos que hemos estudiado hasta ahora (sensitivos) no explican la decisión del clasificador, sino que la intuyen mediante un análisis sensitivo que realizan (cálculo de gradientes). Las técnicas de descomposición como esta que vamos a ver sí que analizan esta predicción directamente, y lo hacen a partir de la redistribución de la salida predicha al espacio de píxeles de entrada.

Deep Taylor Decomposition se inspira en el proceso de *backpropagation* de la red neuronal. Lo que se pretende es redistribuir la salida de la red en las variables de entrada de manera que estas contribuciones de las variables de entrada al valor de la salida se correspondan en cantidad con el propio valor de salida (que se cumpla la propiedad de conservación: $\sum contribuciones = valor\ de\ salida$). Estas contribuciones pueden ser vistas como un *heatmap* que muestra la importancia de cada píxel en la decisión. Para realizar la redistribución, se parte de la salida de la red y se avanza en dirección a la entrada capa a capa emulando así el proceso de *backpropagation*. Cuando alcancemos la entrada, ya tendremos el *heatmap* que buscábamos.

Queda por definir pues en qué consiste un paso del algoritmo. El algoritmo se basa en la serie de Taylor de primer orden de la función f de clasificación evaluada en una raíz de la función para poder explicar la clasificación $f(\boldsymbol{x})$ de un cierto punto $\boldsymbol{x}$. Para explicarlo, primero partiremos de un clasificador sencillo (modelo simple) que nos permita trabajar así con una función f sencilla. Un modelo que nos permite obtener directamente la salida de la clasificación a partir de la entrada (en un solo paso) es el de regresión logística.
La expansión de Taylor de primer orden para la función sería:

$$f(\boldsymbol{x}) = f(\tilde{\boldsymbol{x}}) + \left(\frac{\partial f}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}}\right)^T . (\boldsymbol{x} - \tilde{\boldsymbol{x}}) + \varepsilon =$$
$$= 0 + \sum_p \underbrace{\frac{\partial f}{\partial x_p}\Big|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}} . \left(x_p - \widetilde{x_p}\right)}_{R_p(\boldsymbol{x})} + \varepsilon$$

En dicha expresión, $\tilde{\boldsymbol{x}}$ es una raíz de la función $f(\boldsymbol{x})$ donde se evalúa Taylor y se llega a que, identificando $R_p(\boldsymbol{x})$ como las relevancias asignadas a cada píxel p de la imagen, se obtiene que $f(\boldsymbol{x})$ es igual a la suma de las relevancias, con lo que se cumple la propiedad conservativa que pretendíamos. De esta forma llegamos a la conclusión de que podemos descomponer modelos sencillos mediante la expansión de primer orden de Taylor.

Pero en Deep Learning, los modelos son bastante más complejos y se pueden ver como funciones de varias funciones. Así pues, nos podemos aprovechar de esta descomposición para aplicar el método anterior y lograr un método mejorado cuando trabajamos con redes profundas. El método *Deep Taylor Decomposition* se inspira en el paradigma de divide y vencerás. Ahora, en lugar de considerar la red en su totalidad como una única función f, consideramos el mapeo de una serie de neuronas i de una cierta capa a la relevancia asignada a una neurona j de la capa siguiente. Es decir, vamos capa a capa como en backpropagation y en cada paso aplicamos Taylor Decomposition como hemos explicado antes pero ahora a esta subfunción.
De esta manera pretendemos redistribuir la relevancia de la neurona j a las neuronas i. Sin embargo, en contra del ejemplo de la regresión logística, ahora la neurona j no tiene por qué ser la única de esa capa (puede haber más de una

neurona en la capa siguiente) y por tanto la propiedad conservativa del método se explica así:

$$\sum_j R_j = \left(\frac{\partial \left(\sum_j R_j \right)}{\partial \{x_i\}} \bigg|_{\{\tilde{x}_i\}} \right)^T . (\{x_i\} - \{\tilde{x}_i\}) + \varepsilon$$

$$\underbrace{\sum_i \sum_j \frac{\partial R_j}{\partial x_i} \bigg|_{\{\tilde{x}_i\}} . (x_i - \tilde{x}_i)}_{R_i} + \varepsilon$$

, lo que quiere decir que durante este proceso de *backpropagation* siempre se conservan las relevancias hasta alcanzar el espacio de entrada.

En la *Figura 19* se puede apreciar un resumen visual del algoritmo.

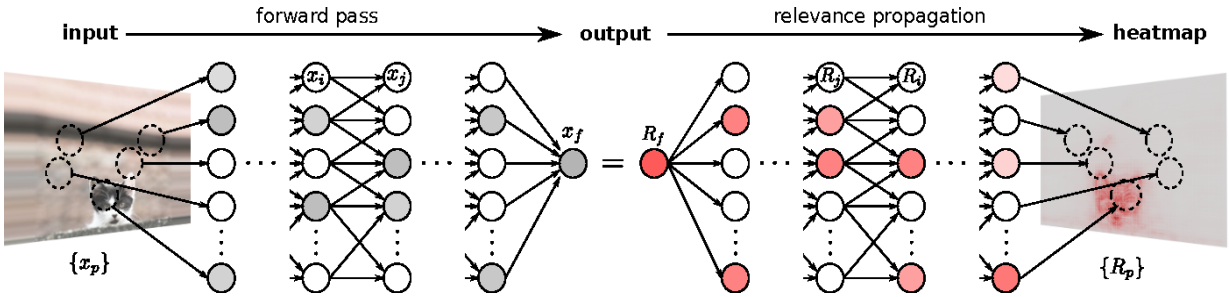


Figura 19. Esquema del método *Deep Taylor Decomposition.*

3.7. Class Activation Mapping (CAM)

Class Activation Mapping (*CAM*) es un método propuesto por *Zhou et al*.[21] para visualizar clases, neuronas de salida.

Este método se basa en generar mapas de activaciones de clase (*class activation maps*) que indican las regiones discriminativas de la imagen de entrada usadas por la CNN para identificar esa clase. Para generar estos mapas, se parte de una arquitectura sencilla consistente en capas convolucionales y en la que justo antes de la capa de salida, se realiza un *average pooling* global en los últimos mapas de activaciones. Esto se hace para así quedarnos con un único valor representante de cada mapa y pasar estos valores por una capa FC (*fully connected*) que produzca la salida deseada. Con esta arquitectura simple, podemos identificar la importancia de las regiones de la imagen proyectando hacia atrás los pesos de esta última capa y es que, si los pesos determinan la importancia que tiene cada valor resultante del *pooling* en la predicción final, también determinan entonces la importancia de sus mapas de activaciones correspondientes.

En definitiva, el mapa de activaciones de clase (*class activation map*) es la combinación lineal de la presencia de diferentes patrones visuales en diferentes

sitios de la imagen (combinación lineal de los mapas de activaciones). Esta combinación lineal se puede llevar a cabo porque conocemos unos pesos que relacionan cada mapa de activaciones con su relevancia en la predicción final de la clase contemplada. Esto es posible gracias a la arquitectura de la que parte el algoritmo. No obstante, el resultado obtenido de la combinación lineal será de las mismas dimensiones que los mapas de activaciones con los que opera. Por eso queda un último paso que es el de convertir el resultado al tamaño de la imagen de entrada (redimensionar).

Matemáticamente, si consideramos que $f_k(x, y)$ representa la activación de la unidad/neurona k en la última capa convolucional para la localización espacial (x, y), entonces el resultado de aplicar *average pooling* sobre el resultado de dicha neurona viene dado por $F^k = \sum_{x,y} f_k(x, y)$[15]. Así, para cierta clase c, la salida de la red es $\sum_k w_k^c F^k$, donde w_k^c es el peso correspondiente a la clase c para la neurona *k*. Esto es, la importancia de F^k para la clase c. Por sustitución de F^k, la salida de la red para la clase c es $\sum_k w_k^c \sum_{x,y} f_k(x, y) = \sum_{x,y} \sum_k w_k^c f_k(x, y)$. En este punto se define $M_c(x, y) = \sum_k w_k^c f_k(x, y)$ de modo que la salida para la clase c es $\sum_{x,y} M_c(x, y)$, por lo que $M_c(x, y)$ indica directamente la importancia de la activación en el lugar (x, y) en la clasificación de c.

Intuitivamente, se espera que cada neurona sea activada por algún patrón visual dentro de su campo receptivo/visual. Es por eso que f_k (mapa de activaciones) es el mapa de la presencia de ese patrón. El mapa de activaciones de clase (*class activation map)* $M_c(x, y)$ es simplemente una suma ponderada de la presencia de dichos patrones visuales en diferentes partes/localizaciones espaciales. Quizá gracias a la *Figura 20* se puedan entender aún mejor estos conceptos.

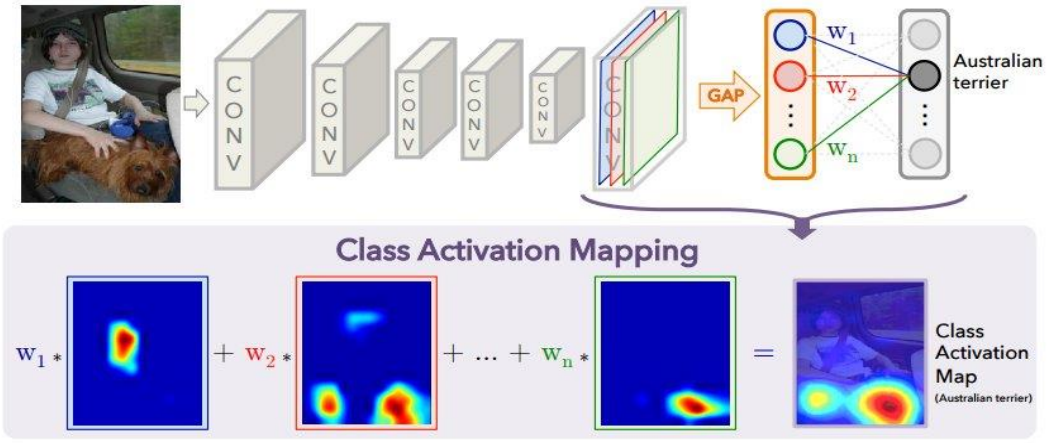


Figura 20. CAM consiste en la combinación lineal de los mapas de activaciones de la última capa convolucional. Los pesos de dicha combinación vienen dados por su relevancia de cara a la predicción final. Esta relevancia la conocemos gracias a la arquitectura de la que parte y es dependiente el método.

[15] En realidad, para ser *average pooling* quedaría por dividir F^k entre el número N de elementos de f_k, pero a efectos de lo que se pretende explicar dicha constante es irrelevante.

3.8. Gradient-weighted Class Activation Mapping (Grad-CAM)

El método anterior (*CAM*) tiene un gran inconveniente, que es la dependencia de una arquitectura muy concreta que no siempre va a ser la que nos convenga para nuestro propósito. Una posible solución a esto sería la de adaptar la arquitectura para poder emplear el método de visualización, pero esto requeriría de un reentrenamiento, que se traduce en un alto coste de recursos temporales y computacionales. En definitiva, parece inviable poder utilizar dicho método en la gran mayoría de ocasiones. Es por eso que en 2017, *Selvaraju et al*.[22] proponen *Grad-CAM,* que es la generalización de CAM y que permite combinar los mapas de activaciones usando gradientes de forma que no se requiera ninguna modificación en la arquitectura.

Grad-CAM consiste en el cálculo del gradiente de la salida correspondiente a la clase a visualizar (*class score*) con respecto a las neuronas de una cierta capa de la CNN (las de la última capa convolucional en un principio). Esto lo que pretende es visualizar la importancia de cada neurona en la decisión. Una vez calculados estos pesos, podemos realizar la combinación lineal de las neuronas como en el método *CAM*. Es decir, *Grad-CAM* difiere de *CAM* principalmente en que no necesita de una arquitectura de la cual aprovecharse para obtener directamente unos pesos que ponderen la importancia de cada neurona en la predicción de una clase, sino que el mismo método se encarga de calcular estos pesos.

Más concretamente, estos pesos los calcula realizando la media aritmética de los gradientes de la salida para la clase analizada c (y^c) con respecto a cada mapa de activaciones A^k de una capa concreta k. Es decir, promedia los valores de los gradientes de cada mapa de activaciones para obtener un único valor que represente la importancia de la neurona en la decisión final. Estos valores, que son los pesos correspondientes a cada neurona, se utilizan para calcular el mapa de activaciones de clase (*class activation map*). Además, al resultado de esta combinación lineal se le aplica la función no lineal ReLU. Así tendríamos que $\alpha_k^c = \frac{1}{Z}\sum_i \sum_j \frac{\partial y^c}{\partial A_{i,j}^k}$ captura la importancia del mapa de activaciones k para la clase c. Seguidamente realizamos la combinación lineal de los mapas de activaciones con los pesos calculados α_k^c y aplicamos ReLU al resultado: $L_{Grad-CAM}^c = ReLU(\sum_k \alpha_k^c A^k)$.

La justificación de aplicar ReLU de acuerdo al *paper* original[22] es que estamos interesados únicamente en las activaciones con influencia positiva en la clase de interés, esto es, píxeles cuya intensidad debemos incrementar para incrementar la probabilidad/*score* de la clase. *Grad-CAM* se puede ver como la generalización de *CAM* siendo α_k^c el que juega el papel de w_k^c.

La razón de querer propagar los gradientes hacia las capas convolucionales es que, como ya se ha dicho, éstas mantienen la espacialidad de la información, que se pierde en las capas *fully-connected.* Esta característica suya junto con la de ser capaces de detectar estructuras de alto nivel visual hacen que visualizar neuronas de capas no convolucionales pase por visualizar neuronas de capas que sí lo son.

Como último apunte a este método, hay que decir que es muy genérico y por tanto puede ser usado para visualizar cualquier neurona a lo largo de la red (no solo las de salida), aunque casi siempre se centre la atención en visualizar las salidas. Lo que sí que en cualquier caso se debería hacer es tomar la última capa convolucional anterior a la neurona que queramos visualizar para generar el mapa de activaciones de clase (en el caso de estar visualizando una predicción, tomaríamos la última capa convolucional del modelo).

3.9. Guided Grad-CAM

Un aspecto negativo del método *Grad-CAM* es que no muestra los resultados con una resolución muy precisa (*heatmap* de baja resolución). Esto es sin duda un problema ya que uno de los pilares de una buena explicación visual es el nivel de detalle que nos ofrezca.

Sin embargo, esto se puede solucionar combinando los mejores aspectos de *Grad-CAM* y otro método con buena resolución como puede ser *Guided Backpropagation*. El resultado de la combinación de estos dos métodos da lugar a uno nuevo: *Guided Grad-CAM*.

Guided Grad-CAM fue idea de los mismos autores que *Grad-CAM* y de hecho fue presentado en el mismo *paper*[22].

La forma en la que *Guided Grad-CAM* combina los dos métodos es mediante la multiplicación punto a punto (elemento a elemento). Es decir, se calculan los resultados para *Guided Backpropagation* y *Grad-CAM* por separado y posteriormente se multiplican elemento a elemento[16](ver *Figura 21*).
En resumen, este método busca beneficiarse de lo mejor de cada uno.

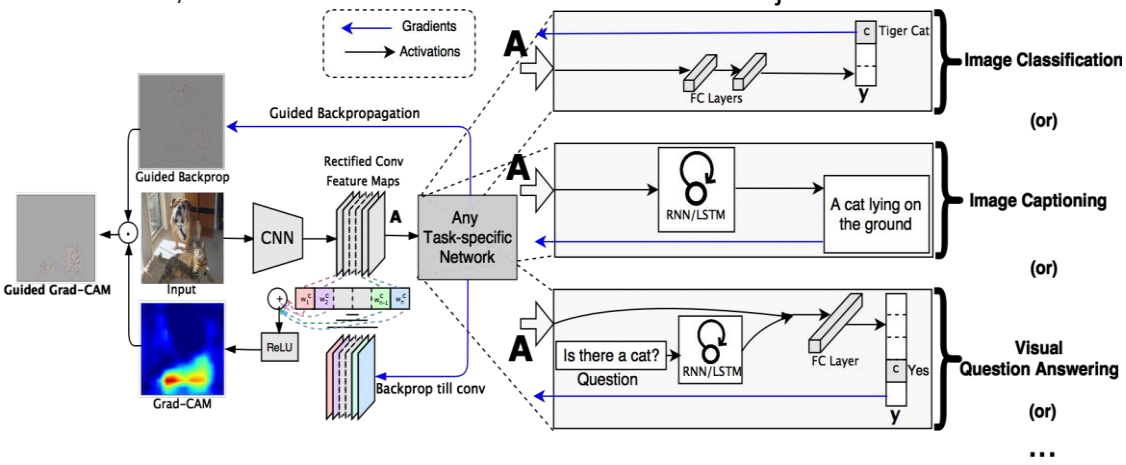


Figura 21. Esquema del método *Guided Grad-CAM,* que no es otra cosa que la combinación de los métodos *Guided Backprop* y *Grad-CAM.*

[16] Los resultados de los métodos por separado tienen la misma dimensión (pues reflejan un mapa de la importancia con respecto a cada píxel de la imagen de entrada), así que el producto punto a punto de estos resultados da lugar a un nuevo resultado de la misma dimensión que refleja la importancia "consensuada" por ambos métodos.

3.10. Local Interpretable Model-Agnostic Explanations (LIME)

Otro método diferente para explicar predicciones (visualizar neuronas de clasificación) es el *Local Interpretable Model-Agnostic Explanations* (*LIME*) propuesto en 2016 por *Marco Tulio Ribeiro et al.*[23].

LIME consiste en generar un modelo interpretable a partir del original que sea el que nos permita explicar la predicción. Con interpretable nos referimos a que sea fácilmente entendible por el usuario, con lo que podemos utilizar por ejemplo un modelo lineal. Además, en el proceso de generación de este nuevo modelo no podemos hacer ninguna suposición acerca del modelo original (*Model-Agnostic*). En resumen, la idea es encontrar un modelo lineal (también puede ser cualquier otro modelo interpretable como árboles de decisión, etc) que aproxime bien al original (que no es interpretable porque por ejemplo es altamente no lineal como lo son las Redes Neuronales) para valores cercanos a la instancia contemplada cuya predicción queremos explicar. Es decir, como intentar aproximar el modelo globalmente (mediante uno simple) es prácticamente imposible y lo que nos interesa realmente es explicar una decisión concreta del modelo (la correspondiente a una instancia concreta), lo que buscamos es aproximar el modelo de forma local, esto es encontrar un modelo simple que aproxime bien al original en la vecindad de la instancia. La *Figura 22* puede ayudarnos a tener una mejor intuición del método.

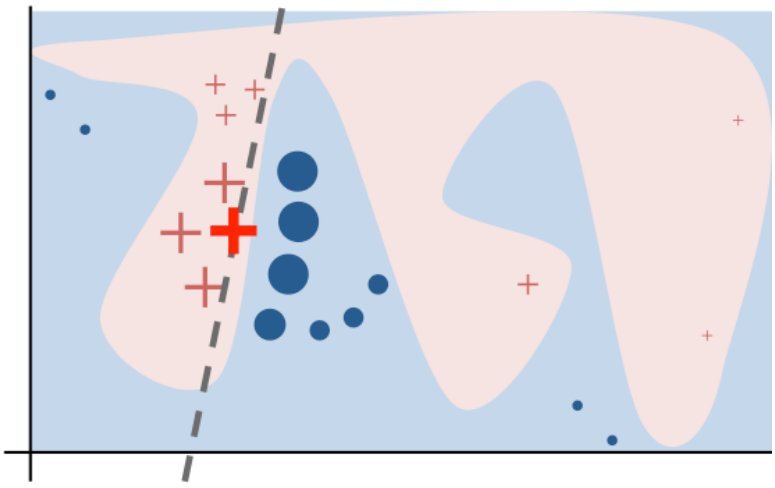


Figura 22. Idea del método *LIME*. La función compleja que queremos visualizar está representada por el fondo azul/rosa, que claramente no puede ser bien aproximada por un modelo lineal. Para ello se busca una aproximación de dicha función solo en el entorno de la instancia (que se representa mediante la cruz destacada en rojo). Lo que hace LIME es generar muestras (diferentes instancias), obtener sus predicciones usando la función original y ponderarlas en base a su proximidad a la instancia siendo explicada. Al final se aprende una explicación local que se representa en la imagen por la línea discontinua.

Formalmente, se define una explicación como un modelo $g \in G$, donde G es una clase de modelos potencialmente interpretables como árboles de decisión o modelos lineales. Lo que tratamos es de encontrar un modelo g lo más simple posible que minimice el error entre sus predicciones y las del modelo original que trata de explicar para instancias alrededor de la original. Esto es:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} L(f, g, \pi_x) + \Omega(g),$$

donde $\Omega(g)$ mide la complejidad del modelo g (en un modelo lineal podría ser el número de pesos distintos de cero) y $L(f, g, \pi_x) = \sum_{z,z' \in Z} \pi_x(\mathrm{z})\,(\mathrm{f(z)} - \mathrm{g(z')})^2$ es la función que mide cuánto de bien aproxima el nuevo modelo g al modelo original f en la vecindad de x. El tercer parámetro (π_x) de la función L, se utiliza como medida de la cercanía/proximidad de las instancias contempladas en la vecindad de x al propio x ($\pi_x(z)$ mide la proximidad de la instancia z a la instancia x). Con esto se pretende ponderar los errores en las predicciones de cada instancia z según la cercanía de dicha instancia a x, dando así más relevancia a los errores cometidos en las instancias más parecidas/similares a x (que es la instancia original sobre la cual se quiere llevar a cabo la explicación). De esta forma, no necesitamos realizar ninguna suposición acerca del modelo original para aprender un nuevo modelo interpretable que aproxime al original localmente (*Model-Agnostic*).

Otro aspecto a tener en cuenta en este método es que, si nuestro nuevo modelo queremos que sea interpretable, ha de ser expresado en términos de componentes interpretables. Lo que significa esto último es que el dominio de este nuevo modelo ha de ser $\{0, 1\}^{d'}$, que indica la presencia/ausencia de los componentes interpretables. Así cuando en la fórmula anterior $L(f, g, \pi_x) = \sum_{z,z' \in Z} \pi_x(\mathrm{z})\,(\mathrm{f(z)} - \mathrm{g(z')})^2$ nos referimos a z', nos estamos refiriendo a z pero representada en términos de componentes interpretables (que es la forma en que podemos evaluar z en el nuevo modelo g).

En el caso concreto que nos ocupa, que es el de clasificación de imágenes, los componentes interpretables con los que trabajamos son superpíxeles (agrupaciones de píxeles con unas mismas características en común; por ejemplo, color). Por tanto, lo que hacemos es, a partir de la imagen original, generar otras imágenes apagando (poniendo en color gris) superpíxeles de la original (técnica de oclusión) y aprender el nuevo modelo lineal que minimice el error existente entre sus predicciones y las del original para las imágenes perturbadas (que son imágenes en la vecindad de la original[17]), ponderando más los errores correspondientes a las imágenes más parecidas a la original (menos alteradas).

Al final, los superpíxeles asociados a los mayores pesos en el nuevo modelo aprendido son los que más importan/contribuyen a la decisión, por lo que para explicar la decisión podemos devolver una imagen en la que solo mostremos los superpíxeles con los mayores pesos en el nuevo modelo y el resto, los tapemos (poniéndolos en color gris).

Por concluir, *LIME* se puede ver como un método derivado de la técnica de oclusión. Esta técnica (oclusión) también se emplea en visualización de CNNs apareciendo por primera vez en [18] y consiste simplemente en ocluir o "tapar"

[17] El muestreo de instancias en la vecindad de la original se hace de acuerdo al dominio de g. Es decir, se generan instancias z'alrededor de x'. El matiz está en que el proceso de muestreo se lleva a cabo desde el enfoque de g y no de f.

zonas o píxeles de la imagen de entrada para ver cómo esta perturbación altera la predicción que se pretende visualizar. Esto puede servir para saber si el modelo se está fijando donde creemos que lo está haciendo (si la predicción apenas varía al ocluir una porción de la imagen, podemos concluir que dicha porción no es importante para la predicción). Como decimos, *LIME* es una manera de automatizar la técnica de oclusión.

3.11. VisualBackProp

VisualBackProp es una técnica surgida en 2017 gracias a *Bojarski et al*.[24] para visualizar el conjunto de píxeles que más contribuyen a las predicciones realizadas por una CNN. Este método fue originalmente desarrollado para sistemas de conducción autónoma basados en Redes Neuronales Convolucionales, pero al ser un método genérico se puede emplear para cualquier otro tipo de aplicaciones.

El método se basa en la intuición de que conforme nos movemos por capas más profundas de una red, los mapas de características contienen cada vez menos información irrelevante de cara a la predicción. Siendo así, los mapas de activaciones de la última capa convolucional deberían contener la información más relevante para determinar la salida de la red. Sin embargo, al mismo tiempo, también es cierto que los mapas de activaciones de las capas más altas tienen más baja resolución. La idea entonces es combinar los mapas de activaciones que contienen solo información relevante (los de las capas altas) con aquellos con más alta resolución (los de capas más bajas). Para lograrlo, se parte de los mapas de activaciones de la última capa convolucional y se propaga la información sobre las regiones de importancia (*backprop*) mientras que simultáneamente se va incrementando la resolución. Este *backpropagation* no es basado en gradientes (*gradient-based*) sino en valores (*value-based*).

En la *Figura 23* se muestra un diagrama del algoritmo. El algoritmo se puede dividir en dos fases diferentes. La primera consiste en realizar la media aritmética de los mapas de activaciones obtenidos en cada capa tras la operación de *ReLU* para así obtener un único mapa de activaciones por capa. Esta fase en realidad no supone un coste computacional adicional ya que se aprovecha del *forward pass* que sí o sí hace falta realizar para poder obtener la predicción a visualizar. Después, en la segunda fase, se parte del mapa de activaciones promedio obtenido en la última capa convolucional y se redimensiona al tamaño del mapa de activaciones promedio de la capa anterior. Este nuevo mapa redimensionado se multiplica por el mapa promedio correspondiente a esa capa. Con el nuevo resultado se vuelve a aplicar el mismo proceso. Así hasta haber alcanzado el espacio de entrada y por consiguiente haber acabado con un mapa de la misma dimensión que la imagen de entrada.

Queda por definir la operación de redimensionado que se realiza en el algoritmo. Esta operación es la de deconvolución[18]. El tamaño del filtro y el *stride* empleados en dicha deconvolución son los mismos que los empleados en la operación de convolución homóloga, pero los pesos son todos unos y los términos independientes o biases, ceros.

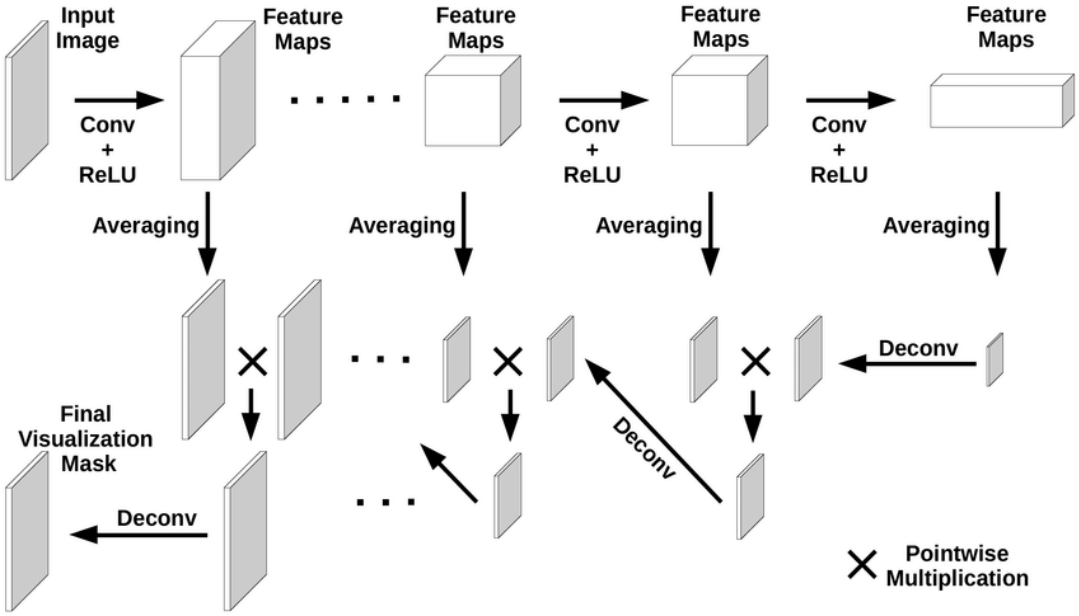


Figura 23. Diagrama del método *VisualBackProp.*

Tal y como está planteado este método (*VisualBackProp*), no sirve para visualizar neuronas específicas o predicciones concretas, sino que más bien se centra en la construcción de patrones visuales que la red en su conjunto es capaz de conseguir.

Así pues, esta técnica ha sido pensada con la intención de verificar que las predicciones generadas por la red, estén basadas en patrones visuales razonables y reconocibles de la imagen de entrada. Esto no viene mal como herramienta para *debuggear* un modelo y sobre todo para ayudar en el proceso de entrenamiento de la red (si la red no es capaz de aprender los patrones visuales esperados se puede optar por cambiar de estrategia, etc). De hecho, en el *paper* original[24] se hace especial hincapié en la complejidad computacional del algoritmo, poniendo en alza el hecho de que sea capaz de correr en tiempo real y que requiera muy poca computación[19]. Sin embargo, la naturaleza del método nos impide emplearlo para explicar decisiones concretas.

[18] Como ya hemos especificado en el caso de las *DeconvNets,* cuando hablamos de "deconvolución" nos referimos a la operación que transforma una salida a la dimensión de la entrada a través de una operación de convolución. Pero no es matemáticamente una deconvolución ya que no revierte la operación original pudiendo recuperar la entrada original.

[19] Ciertamente la ventaja computacional a la que hace referencia [24] es importante, pues no debemos perder de vista que en el origen el algoritmo fue diseñado para sistemas de conducción autónoma, donde es fundamental una respuesta rápida del método de visualización para poder *debuggear* el modelo.

3.12. Network Inversion

Existen métodos de visualización basados en la inversión de la red. La idea básica de estos métodos es la de reconstruir una imagen a partir de los mapas de activaciones de una cierta capa de la red para así destacar aquellas características que está aprendiendo esa capa concreta de la red para una cierta imagen de entrada (qué características de la imagen de entrada son las responsables de dichas activaciones). Algo que distingue a estos métodos de los anteriores (a excepción de VisualBackprop), es que pueden utilizarse para analizar las activaciones desde la perspectiva de una capa completa y no solo de una neurona individual[20] (se puede visualizar una capa en conjunto sin necesitar visualizar cada neurona por separado).

Principalmente existen 2 variantes de *Network Inversion* propuestas para visualizar CNNs. Una fue propuesta en 2014 por *Mahendran et al*.[25] y consiste en reconstruir la imagen aplicando descenso por gradiente sobre la propia imagen de forma que se minimice el error existente entre las activaciones de la capa contemplada de la imagen original y las de la imagen reconstruida(*Regularizer based Network Inversion*); y la otra fue propuesta en 2016 por *Dosovitskiy et al*.[26] y consiste en entrenar una red neuronal que realice los pasos en sentido contrario a la original (que vaya de arriba abajo, es decir, desde las capas más profundas hasta la imagen de entrada), de forma que la salida de esta nueva red sea la imagen reconstruida (el proceso es similar al método de *DeconvNets*, solo que en este caso se aplica descenso por gradiente (o cualquier otra técnica de optimización) sobre los filtros del modelo para minimizar el error entre la imagen reconstruida y la original, mientras que en *DeconvNets* directamente se usan las traspuestas de los mismos filtros. A esta segunda técnica se le denomina *UpConvNets.*

Así pues, mientras que *Regularizer based network Inversion* trata de encontrar una imagen que de unas activaciones lo más similares a las de la imagen original, *UpConvNets* trata de encontrar un modelo que dadas las activaciones de la imagen original arroje una imagen lo más parecida a la original. De este modo, *UpConvNets* nos supone un coste computacional más bajo ya que una vez entrenado el modelo, se puede reconstruir cualquier imagen sin necesidad de computar ningún gradiente (en contra del primero que necesita calcular para cada imagen).

El resumen del proceso seguido por ambos métodos se muestra en la *Figura 24*. Podemos ver que la diferencia está en el enfoque, pues ambos tienen el objetivo de reconstruir la imagen que ha provocado las activaciones de una capa concreta.

[20] *VisualBackProp* también analiza las activaciones desde una perspectiva global de capa, pero en concreto de la última capa de la red. Esto le permite visualizar las estructuras más complejas que es capaz de detectar la red y que son las que sirven para las predicciones.

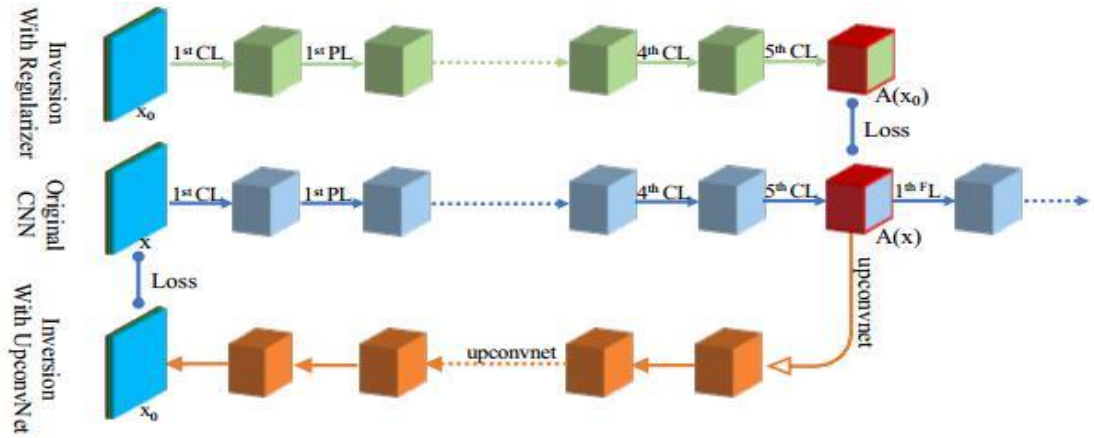


Figura 24. Diagrama comparativo de ambos métodos*: Regularizer based network Inversion* y *UpConvNets*

3.13. Comparativa teórica

Ahora que ya hemos visto los principales métodos de visualización de Redes Neuronales Convolucionales, estamos en disposición de poder compararlos entre sí. Lo haremos destacando las propiedades/características de cada uno de ellos.

La *Tabla 1* resume la comparativa de las propiedades de los métodos estudiados. En la tabla se puede ver que todos los métodos salvo uno realizan la visualización neuronal desde la perspectiva de la imagen de entrada (generando los patrones visuales que producen la activación de la neurona para la imagen de entrada) y no de la propia neurona en sí (generando los patrones visuales que ha aprendido la neurona en general). También se puede ver que la mayoría actúa sobre una única neurona y no sobre un conjunto de neuronas. Los únicos dos métodos que visualizan un conjunto de neuronas y no neuronas por individual, son *VisualBackprop* y *Network Inversion*[21]. Estos métodos visualizan las neuronas de una misma capa en su conjunto (visualizan una capa), y en concreto *VisualBackprop* visualiza la última capa convolucional de la red.

Además, en la tabla se puede apreciar en qué neuronas se centra cada método, es decir, qué tipo de neuronas visualiza cada uno (si las de salida/clasificación o las de no salida/capas intermedias). Esta propiedad pretende clasificar los métodos en dos grupos: los que explican decisiones y los que centran más su atención en explicar el funcionamiento interno de la red. Aquí hace falta aclarar que la clasificación se ha realizado en base a la idea original del método y no en base a las interpretaciones y modificaciones posteriores que se puedan hacer del mismo para que el método sea capaz de abarcar más neuronas a visualizar. Un ejemplo de esto es el caso del método *DeconvNets*, que originariamente[18] fue planteado para visualizar exclusivamente neuronas de capas convolucionales, pero que posteriormente, al demostrarse que era equivalente a

[21] En *Network Inversion* englobamos los dos métodos: *UpConvNets* y *Regularizer based Network Inversion* ya que a efectos prácticos hacen lo mismo; son equivalentes.

	Perspectiva[22]	Actúan sobre:	Centrados en neuronas de:	Basados en gradientes	Resolución	Requieren entrenamiento	Dependientes del modelo
Activation Maximization	neuronal	una neurona	salida y capas intermedias	X	Alta	X	
DeconvNets	imagen	una neurona	capas intermedias		Alta		
Saliency Maps	imagen	una neurona	salida y capas intermedias	X	Alta		
Guided Backprop	imagen	una neurona	salida y capas intermedias	X	Alta		
Deep Taylor Decomposition	imagen	una neurona	salida		Alta		
CAM	imagen	una neurona	salida		Baja		X
Grad-CAM	imagen	una neurona	salida y capas intermedias	X	Baja		
Guided Grad-CAM	imagen	una neurona	salida y capas intermedias	X	Alta		
LIME	imagen	una neurona	salida		Baja	X	
VisualBackprop	imagen	la última capa	la última capa convolucional		Alta		
Network Inversion	imagen	una capa	salida y capas intermedias	X	Alta	X	

Tabla 1. Tabla de características de los métodos de visualización estudiados. Las filas se corresponden con los métodos y las columnas, con las propiedades que se les atribuyen.

[22] Con perspectiva nos referimos a si el método afronta la visualización desde el punto de vista de la neurona (visualizando los patrones aprendidos por la neurona) o desde el punto de vista de la imagen de entrada (visualizando los patrones de la misma que la activan).

un *backward pass* por la red[13], se pudo ver modificado para así poder visualizar también neuronas de capas f*ully connected.* Otros métodos como *Saliency Maps* o *Grad-CAM*, aunque fundamentan su explicación en visualizar neuronas de clasificación, sí que mencionan que la generalidad del método permite visualizar también otro tipo de neuronas de la red.

En general, los métodos que visualizan salidas de la red también deberían servir para visualizar neuronas de capas intermedias, pues como ya se ha mencionado con anterioridad, las activaciones de las neuronas convolucionales difieren de las de salida en la dimensión, pero que, mediante algún tipo de función de agregación, podemos conseguir obtener un único valor representante de la activación que, ahora sí, tenga la misma dimensión que una activación de salida. Y, tomando dicho valor agregado como si uno de salida, ya podríamos aplicar el método para visualizarlo.

El problema de esto sería que a priori perderíamos información al realizar la agregación (además la espacialidad es un aspecto muy importante contenido en los mapas de activaciones, que también perderíamos) y no sabríamos cuán representativo es el valor agregado del mapa de activaciones original. Sin embargo, este es un problema que sucede siempre con cualquier método (ya sea de visualización de neuronas convolucionales, etc) ya que de una forma u otra al final siempre vamos a necesitar realizar una agregación del mapa de activaciones para poder visualizarlo. Por ejemplo, en el método *Saliency Maps* o en cualquier otro que emplee *backpropagation*, implícitamente se utiliza la suma como función de agregación, pues por defecto cuando hacemos $\frac{\partial Z}{\partial A}$ siendo Z un array bidimensional, los gradientes se agregan utilizando la operación suma. Es decir, en el fondo matemáticamente lo que estamos haciendo es calcular $\frac{\partial \sum_{i,j} Z_{i,j}}{\partial \mathrm{A}}$.
En este apartado, mención especial merece *VisualBackprop*, ya que, aunque ya hemos dicho que se centra en la última capa convolucional del modelo, su objetivo es visualizar predicciones (todas ellas en conjunto) valiéndose para ello de esta capa que le aporta información muy útil para hacerlo (indirectamente). Así pues, en realidad visualiza la capa pero también podríamos afirmar que se centra en las predicciones (pues en realidad es su objetivo).

Otra clasificación que se puede hacer de los métodos es si son basados en gradientes (*gradient-based*) o no lo son. Que sean basados en gradientes implica que sean sensitivos y basen su análisis en las variaciones que se produzcan en las activaciones cuando se alteren algo los datos de entrada o no. Vemos en la tabla como la mayoría de ellos sí que son basados en gradientes y, por tanto, sensitivos. LIME sí que de alguna manera podríamos decir que es un método sensitivo, porque calcula la importancia de cada superpíxel de la imagen de entrada alterando la misma (“encendiendo” y “apagando” superpíxeles). Sin embargo, no es basado en gradientes porque no realiza el proceso de *backpropagation* por la red para llevar a cabo ese análisis sensitivo.

Por otra parte, una propiedad muy importante también que distingue a los métodos es la de la resolución de los resultados visuales que arrojan. Y es que, no es la primera vez que decimos en este trabajo que la clave de una buena visualización no solamente se encuentra en la capacidad para resaltar las

regiones espaciales en las que se fija una neurona, sino también en hacerlo con cierto nivel de detalle. En este apartado, los únicos métodos que no muestran una resolución alta son los que no "reconstruyen" una salida. Entiéndase por "reconstruir" el proceso de llevar el resultado de la activación a visualizar de vuelta al espacio original de la entrada. Es decir, los métodos que no realizan ningún tipo de *backpropagation* por la red son los que peor resolución obtienen. En vista a esta observación, podemos ratificar la hipótesis de la que parte el método VisualBackprop, que es que en las capas más bajas de la red los mapas de activaciones tienen más alta resolución[23].

Por último, en la tabla también se hace distinción entre los métodos que requieren entrenamiento y los que no. Con "entrenamiento" naturalmente nos referimos al procedimiento iterativo para ajustar/adaptar parámetros (que no tienen por qué ser propios de la red sino también pueden ser ajenos; por ejemplo entrenar la imagen de entrada). Esto es un aspecto fundamental a tener en cuenta, pues un entrenamiento se traduce en coste computacional y temporal. Así pues, las técnicas que requieren entrenamiento son más costosas en tiempo y otros recursos que las que no lo requieren. Lógicamente a nosotros nos interesa obtener las explicaciones en un margen de tiempo razonable.

Los métodos que sí requieren de algún tipo de entrenamiento son *Activation Maximization*, *Network Inversion* y *LIME*. *LIME* además requiere entrenar un nuevo modelo desde cero, y, por muy simple que sea dicho modelo, esto implica bastantes iteraciones. También dentro de la técnica *Network Inversion*, el método *UpConvNets* necesita entrenar una nueva red, pero a diferencia de *LIME*, este entrenamiento solo se produce una vez (al principio) y no cada vez que necesitamos realizar una explicación sobre una nueva imagen. El que sí que requiere entrenamiento cada vez que se pretende realizar una nueva explicación al igual que LIME es el método de inversión *Regularizer based Network Inversion.* Sin embargo, este método realiza el entrenamiento de la imagen de entrada (ajustando los valores de los píxeles de la imagen para que la red produzca una cierta activación) y no de la red. En este caso, solo se necesita minimizar la función de coste adaptando los valores de una única instancia y no adaptando los pesos de toda una red para minimizar una función coste global que involucra a muchas instancias. *Activation Maximization* (*AM*) consiste en hacer esto mismo también solo que la función de coste es diferente. En definitiva, el método que probablemente más tiempo consuma durante su ejecución es *LIME*.

En este punto se podría considerar también que *CAM* podría requerir un entrenamiento cuando se pretenden visualizar redes con otras arquitecturas diferentes a la propuesta por el método. Pues recordemos que *CAM* es el único método estrictamente dependiente de la arquitectura de la red a visualizar. En la tabla no se presenta a *CAM* como un método que requiera entrenamiento ya que si éste fuera preciso, en sí no formaría parte del método.

[23] Esto en cierto modo es lógico ya que los componentes de los mapas de activaciones de las capas más altas, abarcan más píxeles de la imagen de entrada que los de capas más bajas (son capaces de fijarse en regiones más amplias, perdiendo así resolución).

4. Estudio Experimental

4.1. Marco Experimental

Ha llegado el momento de observar los resultados que se obtienen con los diferentes métodos de visualización. Hasta ahora, hemos analizado los aspectos teóricos de los mismos, pero, sin duda, lo más importante es ver cómo funcionan realmente en la práctica. Para ello, llevaremos a cabo un estudio experimental.

El estudio consistirá principalmente en realizar una comparativa entre los métodos para así poder ser capaces de determinar cuál de ellos obtiene mejores resultados[24]. También será realmente interesante comprobar la coherencia existente entre ellos, pues, suponiendo que todos funcionaran correctamente, entonces los resultados deberían ser razonablemente similares. De hecho, este es un aspecto clave a tener en cuenta durante la evaluación de los métodos, ya que si fuera el caso de que todos ellos arrojasen resultados nada satisfactorios a ojos de un humano, pero sin embargo entre ellos sí que mantuvieran una cierta coherencia, entonces lo que podría estar fallando en esa situación no es tanto los métodos de visualización como el modelo en sí.

Por otra parte, como ya hemos explicado en el estudio teórico de los métodos, no todos ellos tienen el mismo enfoque, es decir, algunos se centran en visualizar neuronas para una imagen concreta, otros lo hacen desde el punto de vista de la propia neurona generando la imagen que más la estimula en términos absolutos, otros visualizan capas de neuronas en vez de neuronas por individual, etc. Esto lo que nos indica es que no podemos realizar una comparativa entre todos ellos, sino, más bien, como mínimo una comparativa por cada 'tipo' de método. Y, dado que hemos repetido en más de una ocasión a lo largo de este trabajo que el 'tipo' más frecuente de método es el que realiza una visualización relativa a una imagen de entrada (más aún, casi todos los métodos que hemos estudiado son de este tipo), lo que haremos será una comparativa entre los métodos de este tipo.

Así, nuestro estudio se basará en una comparación entre los diferentes métodos estudiados de visualización de neuronas relativos a una imagen de entrada. No obstante, también trataremos de mostrar algún resultado de algún método de los otros tipos. En cualquier caso, lo primero que haremos será centrarnos en la comparativa, que ha de ser la parte principal de este apartado.

Antes de nada, debemos poner en situación al lector. Debemos especificar el proceso llevado a cabo para realizar el estudio experimental. Debemos especificar el código utilizado para la implementación de los métodos, las imágenes empleadas y como no, explicar a más nivel de detalle la comparativa que vamos a efectuar.

[24] Con "mejores resultados" nos referimos a los resultados más lógicos/coherentes a los que nos proporcionaría un ser humano.

Código:

La experimentación se ha realizado en el entorno *Google Colab* utilizando los frameworks *Keras* y *TensorFlow* (*Keras* como librería de alto nivel corriendo sobre *TensorFlow* como librería de más bajo nivel). De esta forma, se ha tratado de buscar siempre en la medida de lo posible implementaciones de los métodos que utilizasen *Keras*.

La manera de proceder ha sido la de adquirir el código del repositorio *github* correspondiente al método y adaptarlo para funcionar en nuestro entorno y bajo unas condiciones de igualdad con respecto al resto de métodos (todos ellos produzcan el mismo tipo de salidas y entradas para luego poder así automatizar la comparativa). Para ello, aparte de tratar individualmente el código de cada método, se ha implementado una librería que los reuniese a todos para así poder abstraerse de las diferentes implementaciones. Dicha librería permite realizar la visualización de cualquier método de los implementados de forma cómoda, tan solo especificando qué métodos se pretenden utilizar para visualizar qué neuronas de qué modelo. Utilizar esta librería nos simplifica la comparativa, pues ahora ya solo nos debemos preocupar por iterar por los diferentes modelos, neuronas y métodos que vamos a emplear en la comparativa y llamar a la librería con cada configuración.

Para la mayoría de métodos, existen varias implementaciones en la nube. Además, como ya hemos mencionado, cada una tiene sus peculiaridades (algunas solo trabajan con un modelo determinado, otras en cambio solo están pensadas para visualizar neuronas de una determinada capa, unas toman como entrada imágenes en color y otras en blanco y negro, preprocesado/postprocesado diferentes, etc) y resulta difícil encontrar implementaciones lo suficientemente genéricas para nuestro propósito. Esto hace que irremediablemente tengamos que intervenir en el código. Por eso, resulta más que conveniente cerciorarnos de que lo que estamos implementando sea correcto.

Para asegurarnos de que la implementación que hagamos sea correcta, lo que hacemos es, en el caso de un método que tenga varias implementaciones diferentes subidas en la red, emplear más de una y comparar sus resultados. Cuando veamos que los resultados son los mismos nos quedamos con cualquiera de ellas. Finalmente, como prueba definitiva de la corrección, vamos al *paper* original correspondiente al método implementado y comparamos los resultados que expone dicho *paper* con los obtenidos. De esta forma nos aseguramos de hacer una comparación justa entre métodos.

Dicho esto, podemos destacar las fuentes utilizadas para la extracción de código durante este proceso. Algunas librerías incluyen la implementación de varios métodos. Una de ellas es por ejemplo [27], que incluye la implementación de métodos como *Activation Maximization*, *Saliency Maps* o *Grad-CAM.* Métodos como *Saliency Maps* o *Guided Backprop* también están implementados en [28]. En [29] podemos encontrar la implementación de *Grad-CAM* y de *Guided Backprop*. *Grad-CAM* también se encuentra implementado en [30] y [31]. Para implementar *DeconvNets* nos hemos inspirado en varias fuentes como [32][33][34]. La implementación para *LIME* la hemos obtenido de [35]. Otros

frameworks con implementaciones de varios métodos como [36] o [37] nos han servido de gran ayuda para implementar *Deep Taylor Decomposition* y *Visualbackprop* respectivamente. También nos hemos apoyado en otros frameworks como [38] o [39] durante el proceso.

Dataset/Imágenes:

Las imágenes que hemos empleado para realizar este experimento provienen de dos fuentes diferentes. Queríamos comprobar cómo se comportan los métodos ante distintas situaciones y por tanto hemos optado por utilizar imágenes del conocido dataset *ImageNet*[40] pero también imágenes aleatorias de internet (sacadas de *Google*).

ImageNet es un proyecto que proporciona una gran base de datos de imágenes con sus correspondientes anotaciones indicando el contenido de las imágenes. Este enorme dataset cuenta con imágenes de 1000 clases diferentes y debido a su gran tamaño es fundamental en el ámbito de investigación en visión artificial. De hecho, cada año, el equipo de *ImageNet* organiza una competición en la que diferentes equipos de investigación prueban sus algoritmos de reconocimiento visual. En dicha competición se usan más de un millón de imágenes de las 1000 clases que tiene el dataset (es un subconjunto del dataset el que se usa, ya que el dataset contiene aún más imágenes). Gracias a esta competición podemos saber qué redes son las mejores clasificando imágenes. Y es que, muchas de las arquitecturas conocidas de Redes Neuronales Convolucionales se hicieron famosas a raíz de la competición.

Tan importante es el dataset *ImageNet* que la mayoría de frameworks en Deep Learning almacenan los pesos obtenidos de haber entrenado cada modelo con el dataset (al menos lo hacen para los modelos principales, esto es, las principales arquitecturas). De este modo, podemos en cualquier momento cargar esos pesos y ya tendríamos nuestro modelo entrenado sin necesidad de haber realizado nosotros el entrenamiento (modelos preentrenados). Esto es de grandísima ayuda ya que recordemos que en el caso de las Redes Neuronales Convolucionales, éstas detectan características de la imagen de entrada que luego le sirven para realizar la predicción. Por tanto, a pesar de que queramos realizar nuestra propia clasificación, partir del modelo ya entrenado en *ImageNet* nos facilita enormemente las cosas, pues sobre todo las capas más bajas, que son las que detectan las características más simples, no necesitarán apenas ser modificadas.

Con todo lo dicho, con emplear imágenes de *ImageNet* debería bastar para nuestro análisis. Sin embargo, dado que casi todos los modelos y métodos de visualización son evaluados siempre con imágenes de *ImageNet*, nosotros queremos ver también lo bien o no que generalizan a otras imágenes de distribuciones diferentes.

Arquitecturas y métodos y comparativa:

Anteriormente ya se ha comentado que los métodos de visualización de CNNs sirven como medida para evaluar el comportamiento de un modelo determinado, es decir, sirven para ver si una red está bien entrenada o no. En cambio, en este trabajo lo que pretendemos es comparar los distintos métodos de visualización. Es por eso que, si los resultados obtenidos no son todo lo buenos que esperamos, no podemos llegar a saber a ciencia cierta si se trata de un problema del método de visualización o de la red. Claro que como vamos a evaluar más de un método, comparando sus resultados podremos llegar a intuirlo. Parece lógico pensar que, si todos los métodos arrojasen resultados similares, entonces el posible problema se encontraría en la red y no en los métodos, mientras que si los métodos arrojaran resultados distintos, entonces alguno de ellos estaría fallando. En el peor de los escenarios, si los resultados obtenidos por los métodos fueran completamente diferentes entre sí y además ninguno mostrara un resultado aceptable, entonces no sabríamos si estaría fallando la red y realmente alguno de los métodos sí que arrojara resultados correctos o si por el contrario la red no fallase y lo hicieran todos los métodos.

Para solventar esta paradoja, lo mejor que podemos hacer es trabajar con modelos para los cuales tengamos una confianza lo más alta posible de que estén bien entrenados. Y, dado que *ImageNet* es uno de los datasets más grandes para tareas de clasificación, parece más que apropiado trabajar con modelos preentrenados en *ImageNet*. De esta forma también nos aseguramos de que el modelo sea capaz de generalizar bien a nuevos datos, imágenes distintas a las utilizadas durante el entrenamiento.

También, para que la comparativa entre métodos sea lo más fiable posible, debemos trabajar con más de un modelo diferente (arquitecturas diferentes) y con diferentes imágenes. En concreto, la comparativa constará de varios métodos, varias imágenes y varios modelos. La idea es visualizar por cada método, imagen y modelo más de una neurona del mismo.

En cuanto a las arquitecturas involucradas en la comparativa, diremos que son las principales: *Vgg16*, *Inception V3, Xception, ResNet50, MobileNet V2.* Como decimos, de cada una de ellas, tomaremos el modelo preentrenado en *ImageNet*.

Respecto a las imágenes, tomaremos unas cuatro de ImageNet y otras seis de otra fuente diferente (no de *ImageNet*). Insistimos, esto lo hacemos para asegurarnos de que los métodos no sean más sensibles a un tipo de imágenes que a otras (pues según hemos visto en el estudio teórico no deberían ser afectados por la distribución de las imágenes). En teoría si el modelo está bien entrenado en *ImageNet*, debería generalizar bien a imágenes distintas a las de *train*, con lo que los métodos de visualización no deberían verse afectados. En cualquier caso, las imágenes que escojamos han de pertenecer a una de las clases de *ImageNet*, pues vamos a trabajar con modelos entrenados en *ImageNet* cuya salida es un vector de 1000 neuronas, cada cual indicando la probabilidad de pertenencia de la imagen de entrada a cada una de las clases de *ImageNet*.

Como ya venimos diciendo, la comparativa se va a realizar entre los diferentes métodos estudiados de visualización neuronal relativa a una imagen de entrada. Esto comprende los métodos: *DeconvNets, Saliency Maps, Guided backprop, Grad-CAM, Deep-Taylor Decomposition, LIME* y *Guided Grad-CAM.* Nótese que el método *CAM* no va a entrar en la comparativa ya que el método *Grad-CAM* es su generalización y nos permite visualizar modelos de varias arquitecturas diferentes.

En la comparativa, no solo visualizaremos las neuronas de salida (predicciones) sino que también visualizaremos algunas neuronas de capas intermedias. Ya sabemos que no todos los métodos permiten hacer esto: solamente lo haremos con aquellos que sí lo permitan. Como excepción conviene destacar el caso del método *DeconvNets*, que ya hemos señalado que es un método creado en su origen para visualizar exclusivamente neuronas de capas intermedias y no de salida. Pero, dado que la visualización de las predicciones es de lo más relevante y la interpretación que se hizo posteriormente del método lo permite, en esta ocasión sí que visualizaremos la capa de salida con este método.

Las capas intermedias 'elegidas' para llevar a cabo la visualización serán aleatorias siempre y cuando estén bien distribuidas. En un principio, iban a ser cuatro las neuronas de capas intermedias repartidas por la red a visualizar, pero hemos probado que, debido a que las neuronas de capas más bajas detectan características tan básicas y poco perceptibles (texturas, bordes…), sus resultados carecen de interpretabilidad y no nos aportan información de cara a la comparativa. En la *Figura 25* podemos contemplar un ejemplo de lo que estamos comentando. En ella se muestra la visualización de una neurona de la tercera capa de una red *Vgg16* realizada por los distintos métodos analizados. Podemos ver que, sea lo que fuere lo que detecta, no podemos realmente apreciarlo en una imagen, aunque presumiblemente intuimos que se trata de bordes o texturas. En resumen, podemos reducir la comparativa quedándonos con dos neuronas de capas intermedias, que sean de capas suficientemente altas. Una vez que hayamos elegido las dos capas, seleccionaremos una neurona por capa, que será la que visualizaremos. La neurona se seleccionará de acuerdo a su activación. Lo que haremos será analizar las activaciones de todas las neuronas de la capa y la que mayor activación tenga, será la que visualicemos. Con mayor activación nos referimos a la que sume un mayor valor de activación (la función de agregación para los mapas de activaciones será la media).

Tiene sentido que visualicemos las neuronas con mayor activación de entre un conjunto (capa) y que no visualicemos cualquiera al azar, porque las neuronas con baja activación indican que no están siendo estimuladas por la imagen de entrada, lo cual quiere decir que realmente no están detectando nada en especial. Al no estar detectando nada, los resultados que pudiéramos obtener de la visualización no serían interesantes. Esto no haría más que confundirnos a la hora de evaluar la correcta actuación del método.

Seguimos el mismo criterio para visualizar neuronas de salida. Visualizaremos únicamente las clases de las imágenes. Quizá en este caso se vea aún más clara la necesidad de hacerlo de esta manera (no tiene sentido ponernos a

visualizar qué características de la imagen hacen que la red prediga 'perro' cuando realmente la red no está viendo un perro en la imagen), aunque sí es cierto que podría suceder que la red otorgase probabilidades muy altas a más de una clase diferente. En esa situación sí podría convenir visualizar más de una clase, pero nunca en cualquier caso visualizar una clase a la que la red no da casi probabilidad.

Figura 25. Visualización de la neurona más activada de la tercera capa de una red *Vgg16*. La visualización se ha llevado a cabo con cinco métodos diferentes: los cinco estudiados que nos permiten visualizar neuronas de capas intermedias. De izquierda a derecha: *DeconvNets*, *Saliency Maps*, *Guided backprop*, *Grad-CAM*, *Guided Grad-CAM*. Vemos a simple vista que la comparativa de los métodos en esta situación es estéril, pues ni siquiera podemos saber qué es lo que está detectando la neurona y mucho menos, apreciarlo en el resultado de los métodos. A una mala, en neuronas de capas más altas, a pesar de que tampoco conozcamos lo que debería detectar la neurona, como lo que sí sabemos es que debe ser algo más complejo y destacable en una imagen (objetos, etc), los métodos nos pueden ser de gran ayuda para descifrarlo.

Con todo, ya estamos listos para presentar los resultados de la comparativa.

4.2. Comparativa: Resultados

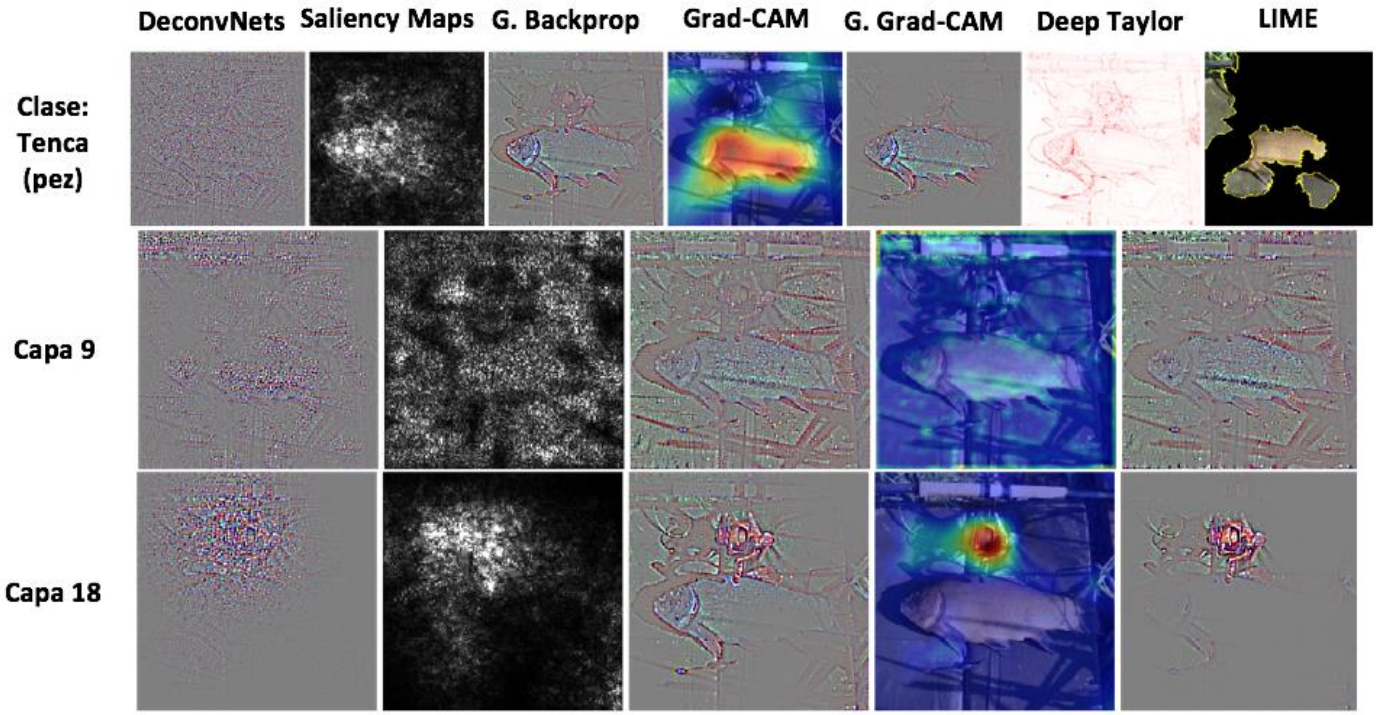
A continuación, se presentan los resultados. El esquema de presentación que se sigue es: primero se muestra la imagen original de entrada que se está empleando en la visualización[25]. Para esa imagen se indica si es de las sacadas de *ImageNet* o no. Luego se muestra para cada arquitectura los resultados de la visualización con esa imagen. Es importante recordar que las neuronas a visualizar se escogen de acuerdo a su activación, por tanto, en el caso de querer visualizar una neurona de la capa de salida, ésta será la más activada de entre todas las neuronas de salida. Esto quiere decir que no tienen por qué coincidir las clases a visualizar para las diferentes redes (puede ocurrir perfectamente que para la misma imagen la predicción de una red sea distinta a la de otra). Se indicará en cada caso qué clase es la que se está visualizando.

Este proceso se repetirá para las diez imágenes utilizadas en la comparativa.

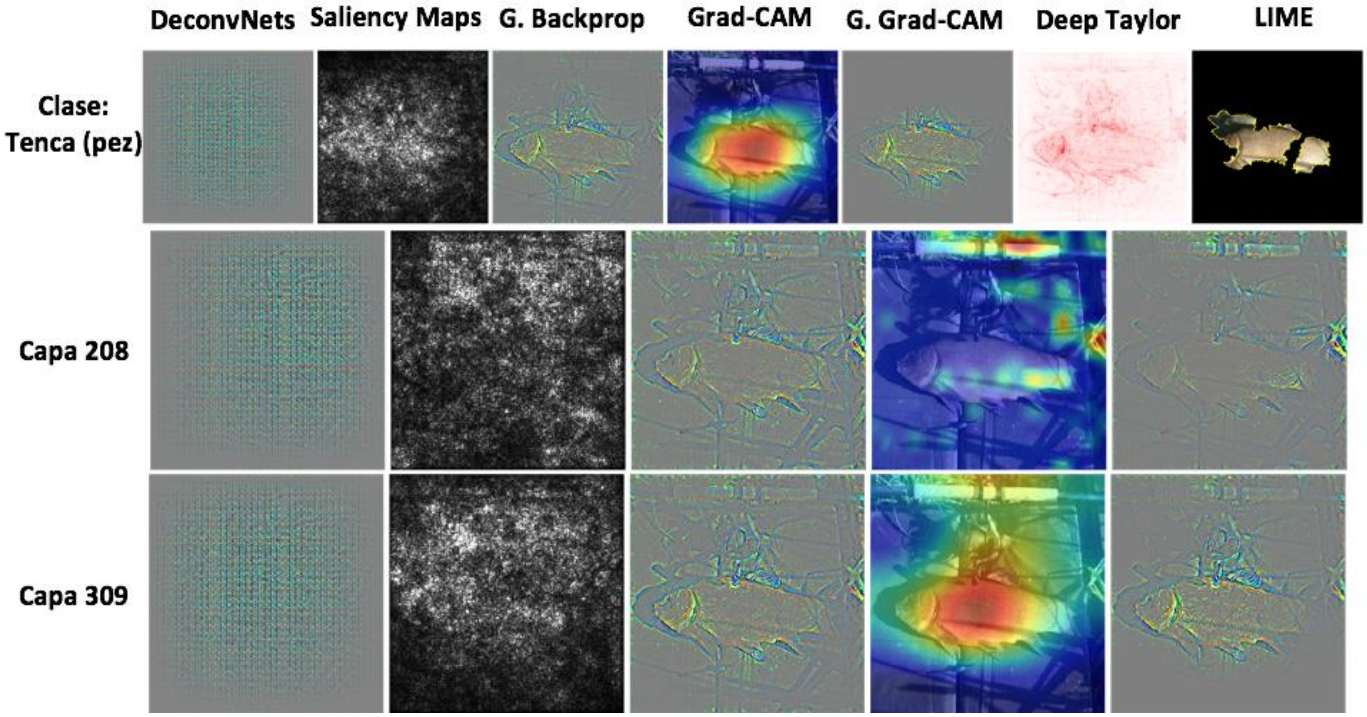
[25] En realidad, la imagen original nunca se emplea como entrada directamente, sino que se preprocesa de acuerdo a una función específica de cada tipo de red.

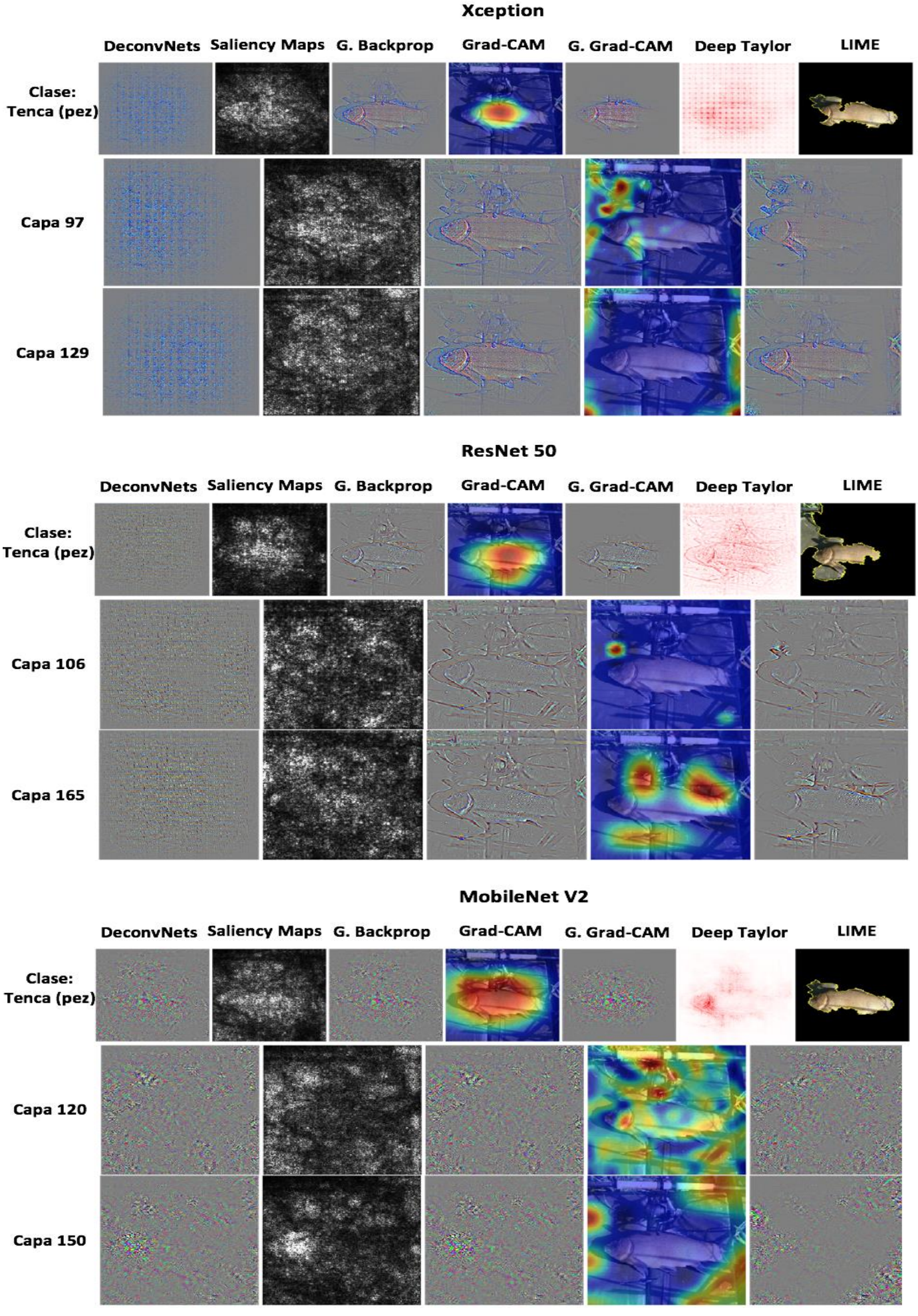
From ImageNet

VGG 16



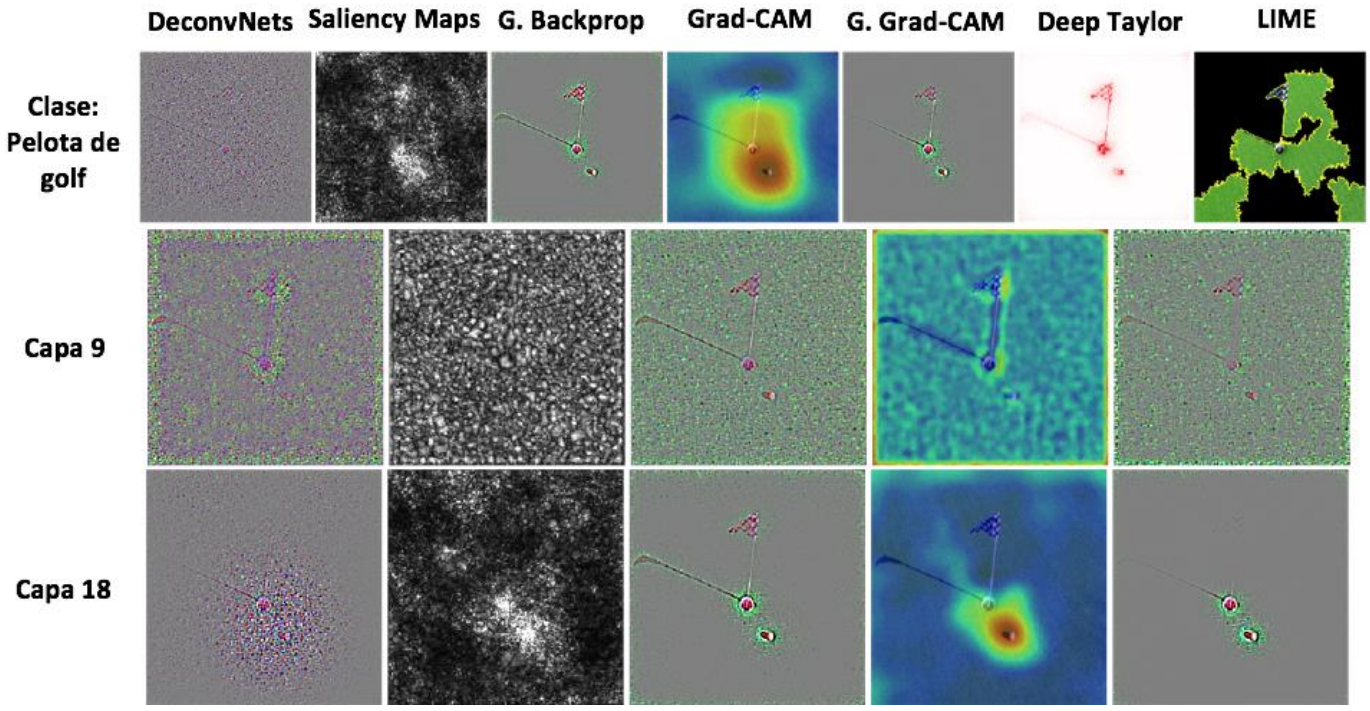
Inception V3



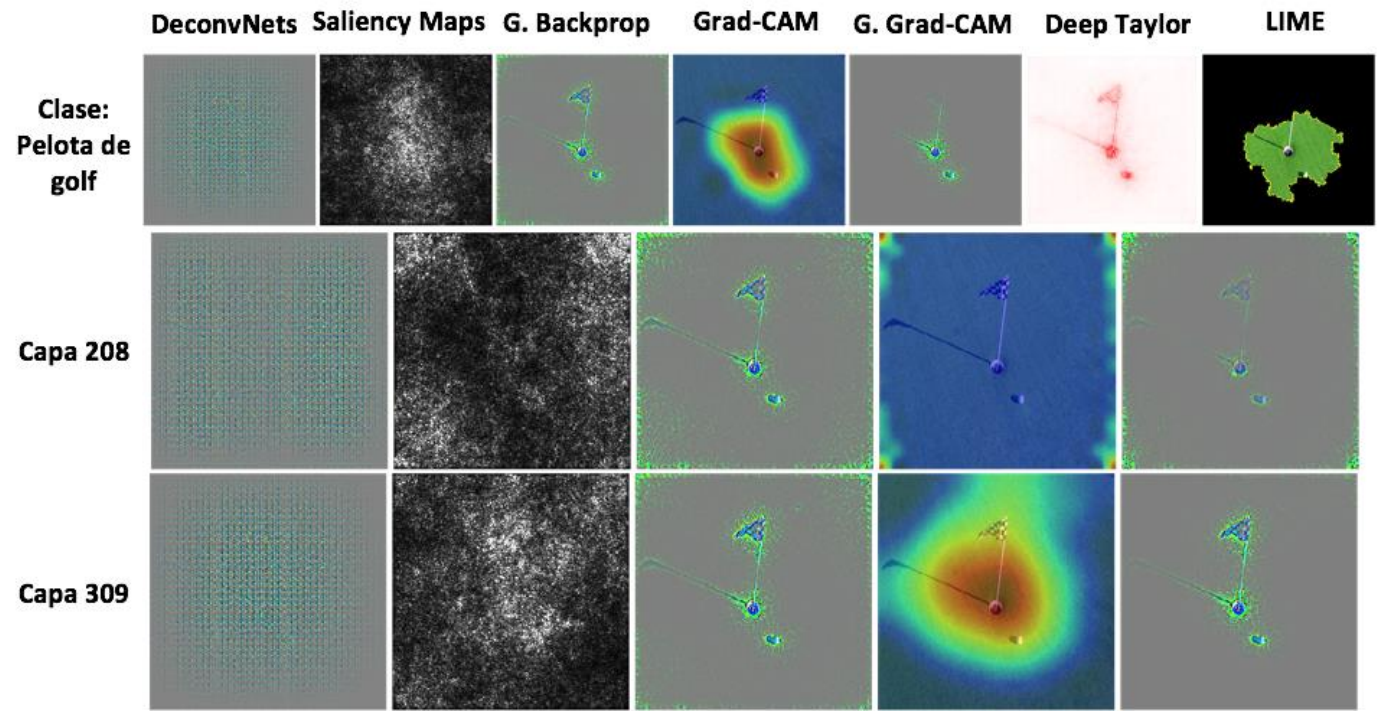

Xception
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Tenca (pez)
Capa 97
Capa 129
ResNet 50
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Tenca (pez)
Capa 106
Capa 165
MobileNet V2
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Tenca (pez)
Capa 120
Capa 150

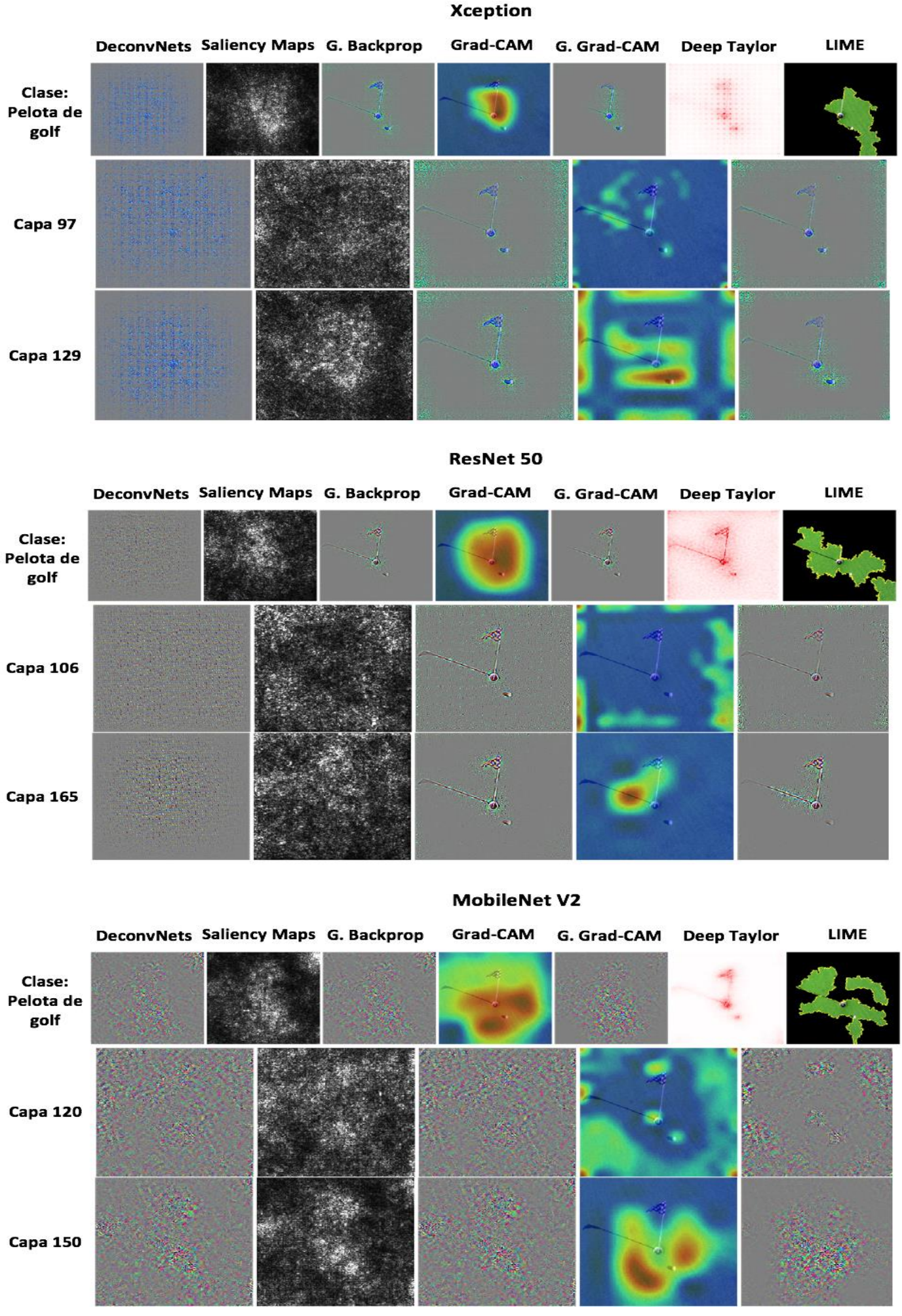
From ImageNet

VGG 16



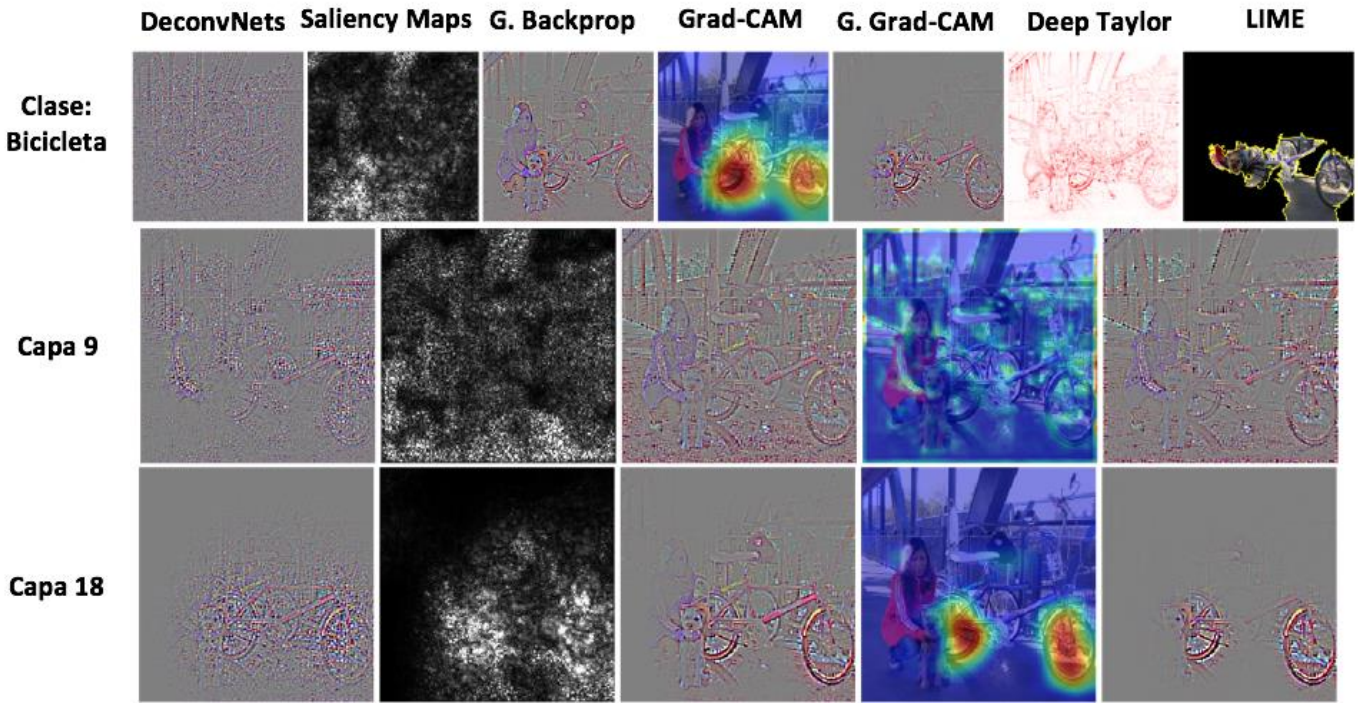
Inception V3



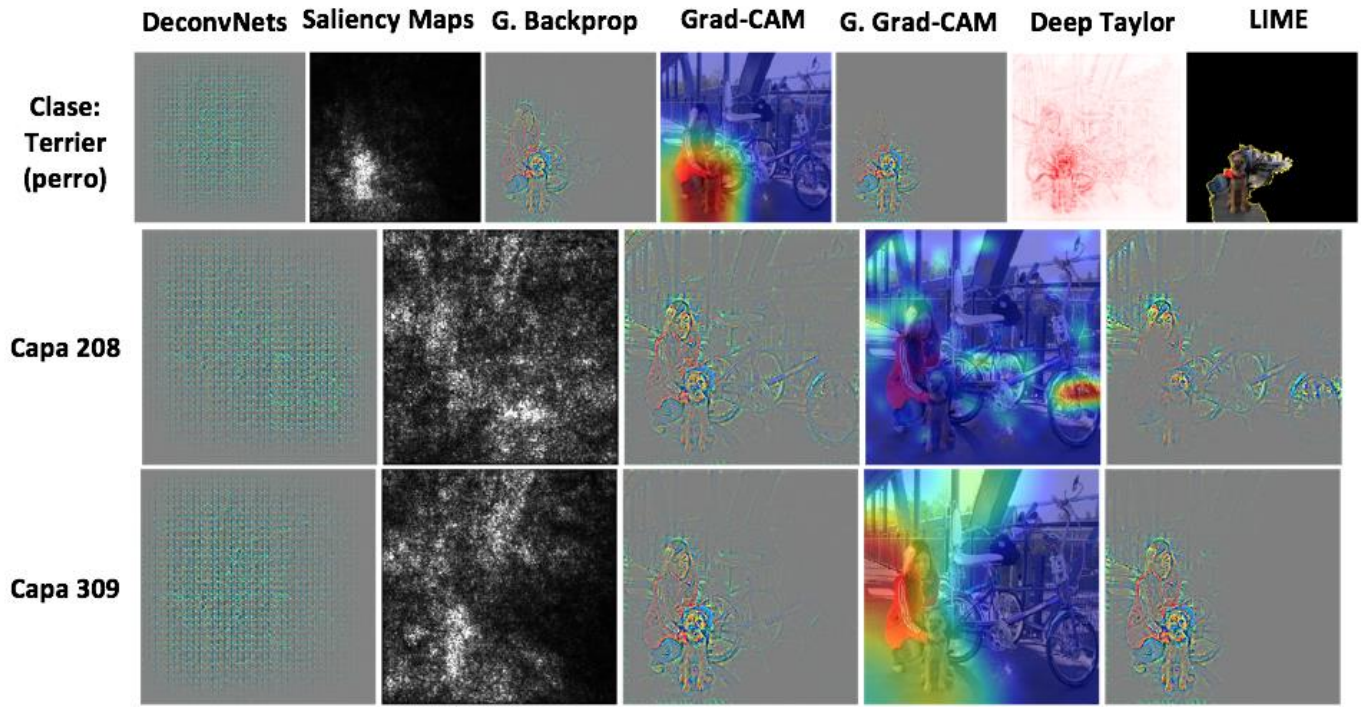

Xception
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Pelota de golf
Capa 97
Capa 129
ResNet 50
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Pelota de golf
Capa 106
Capa 165
MobileNet V2
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Pelota de golf
Capa 120
Capa 150

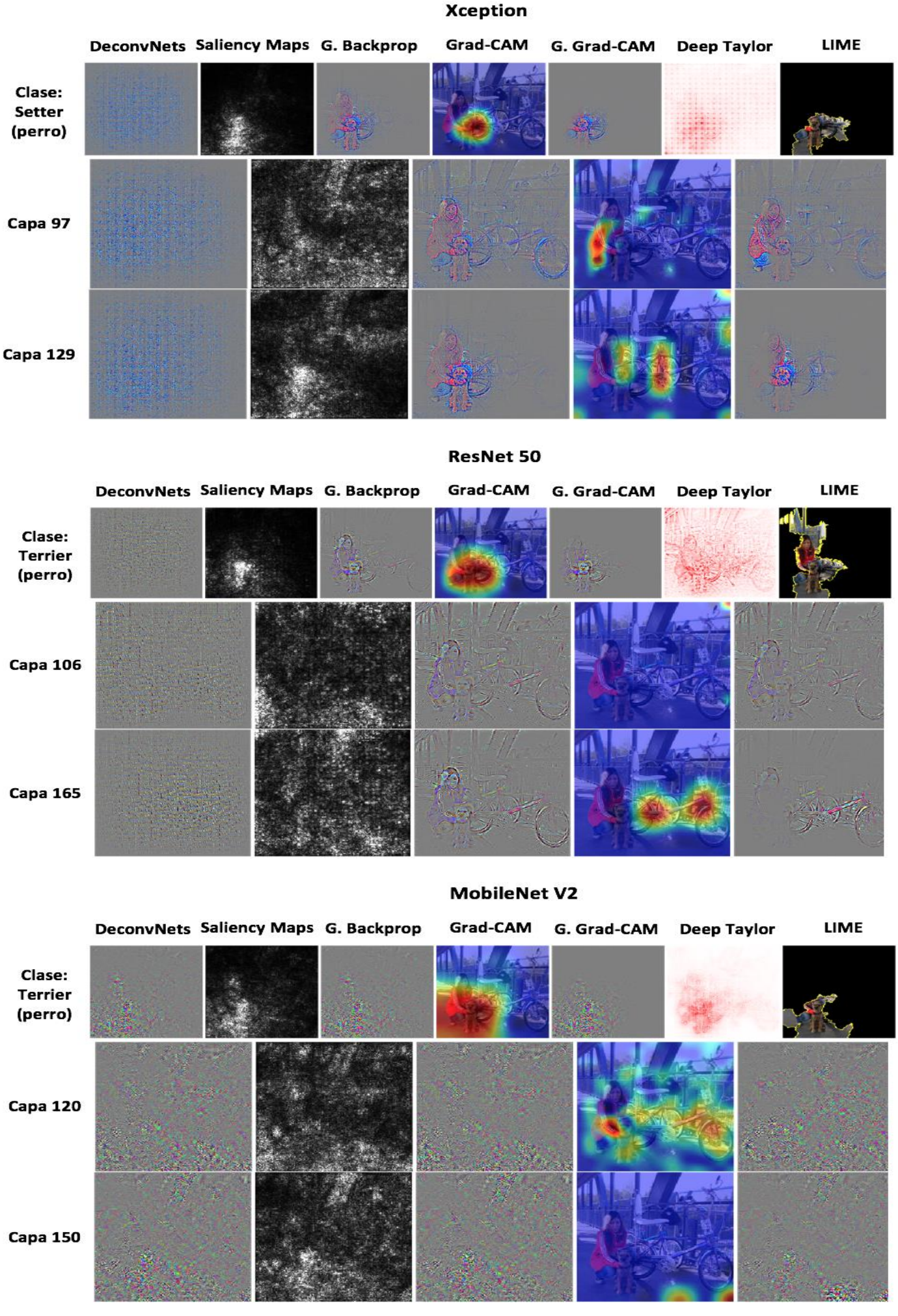
From ImageNet

VGG 16



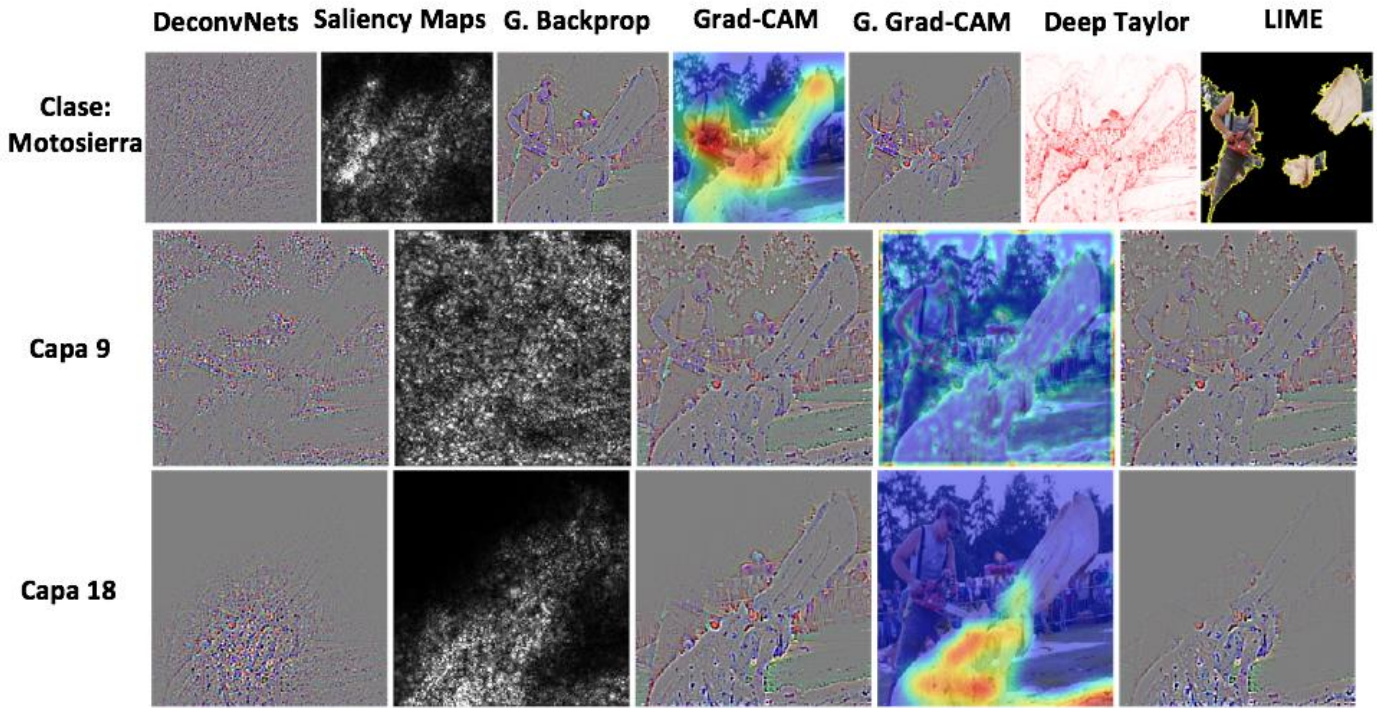
Inception V3



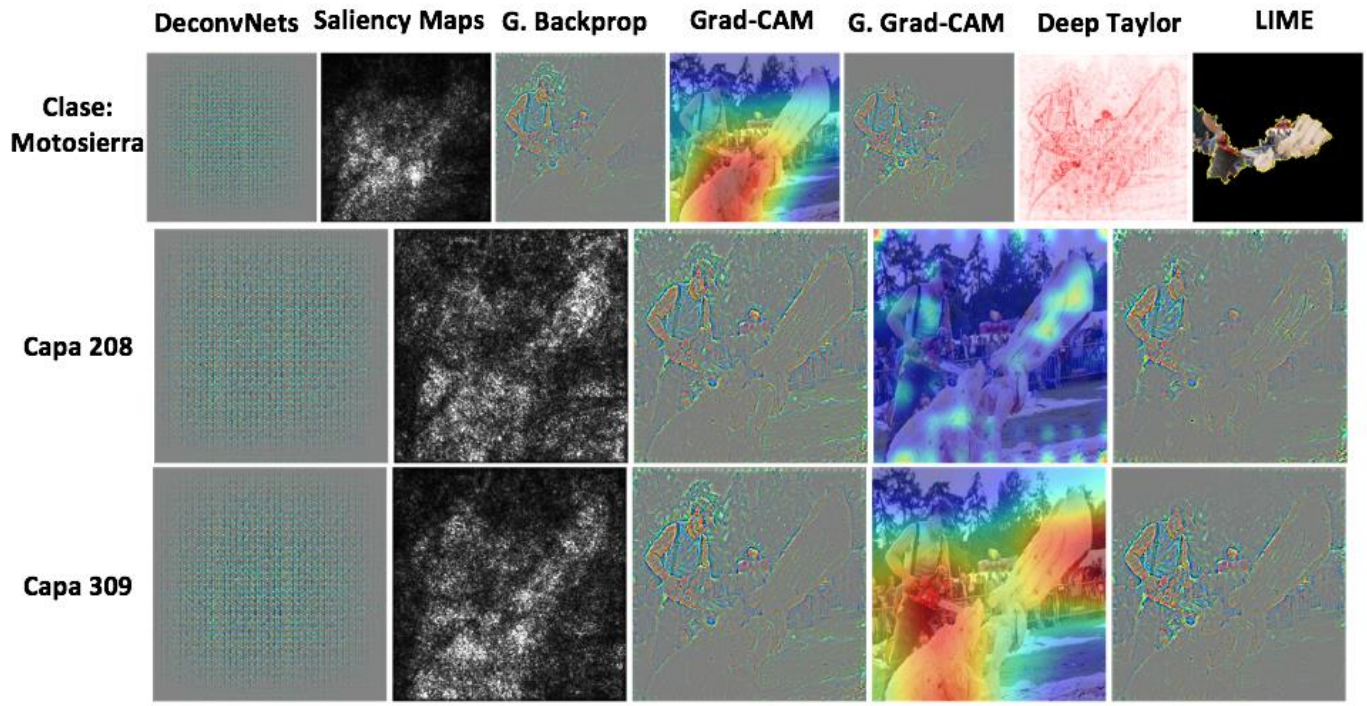

Xception
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Setter (perro)
Capa 97
Capa 129
ResNet 50
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Terrier (perro)
Capa 106
Capa 165
MobileNet V2
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Terrier (perro)
Capa 120
Capa 150

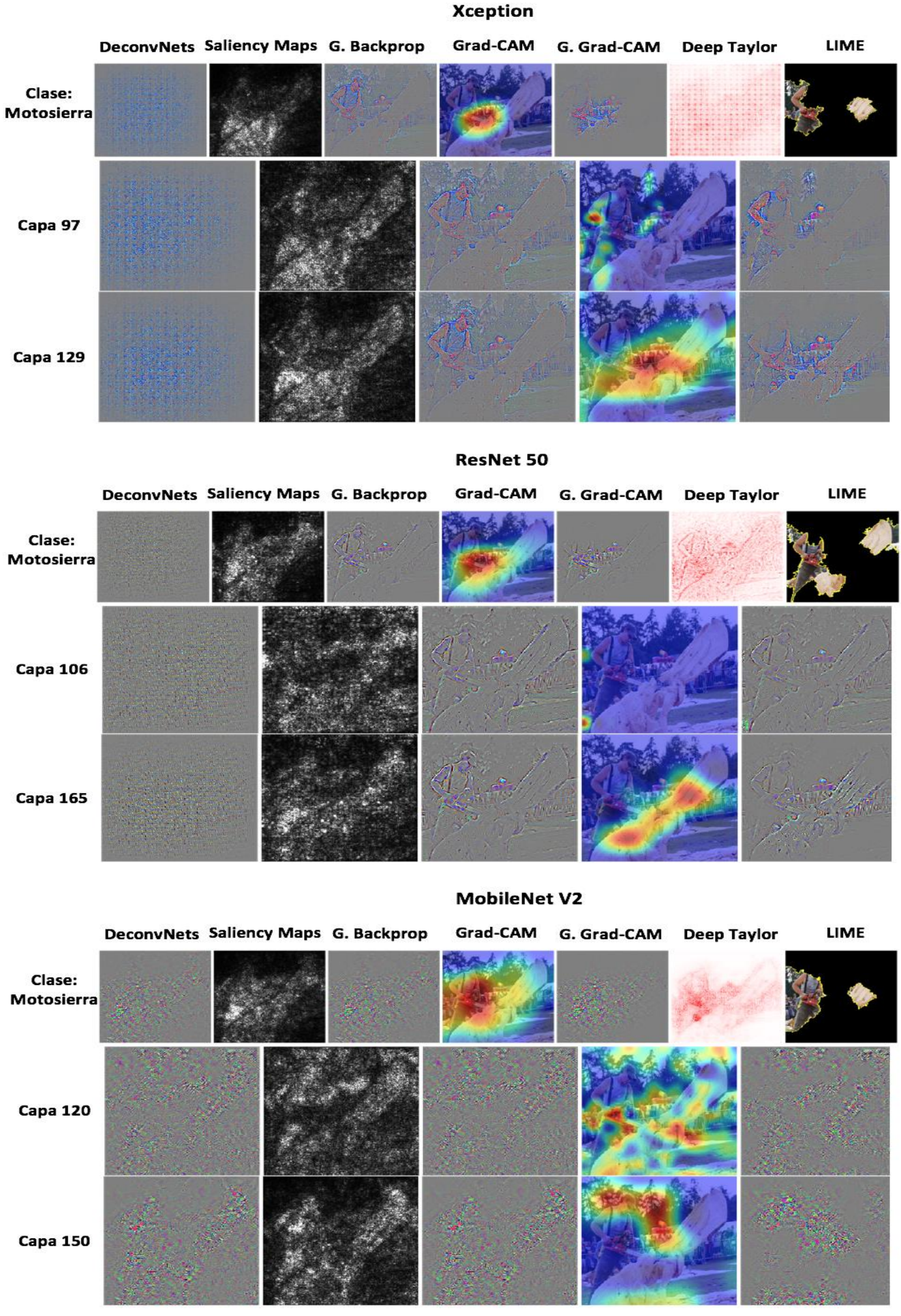
From ImageNet

VGG 16



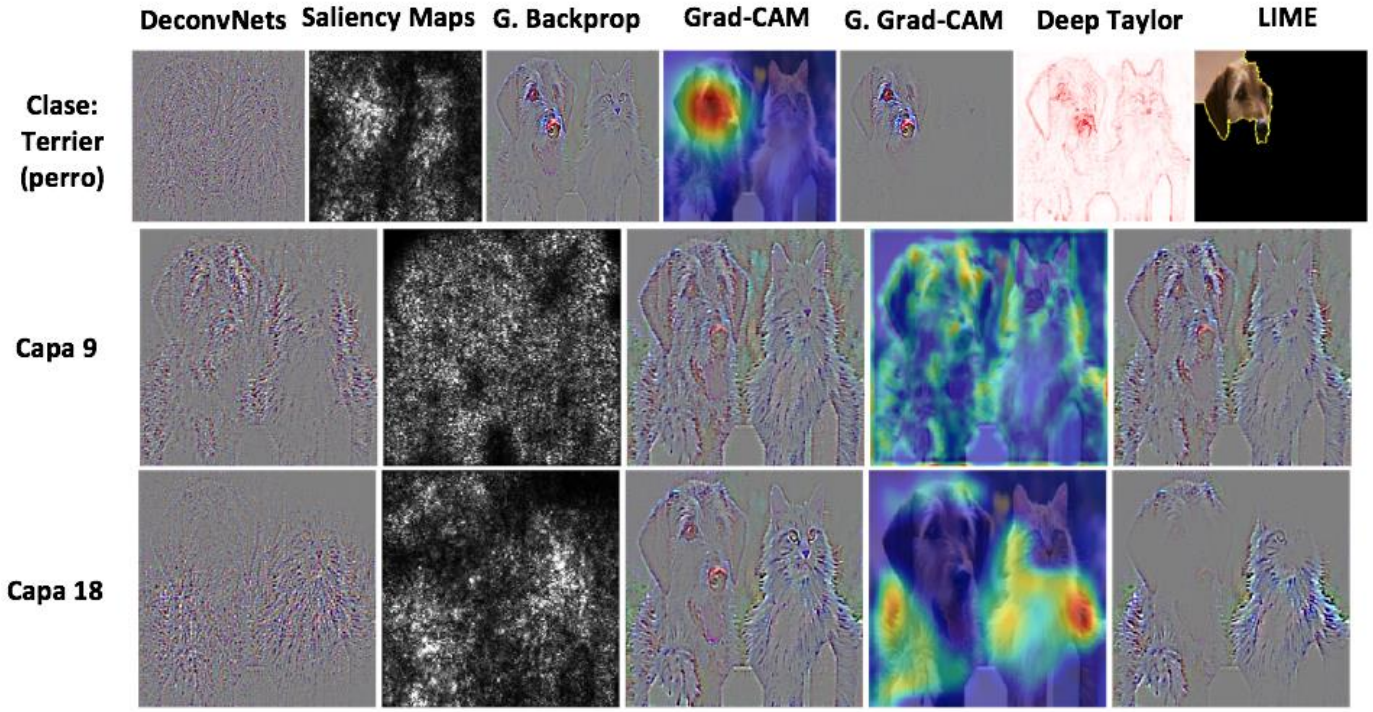
Inception V3



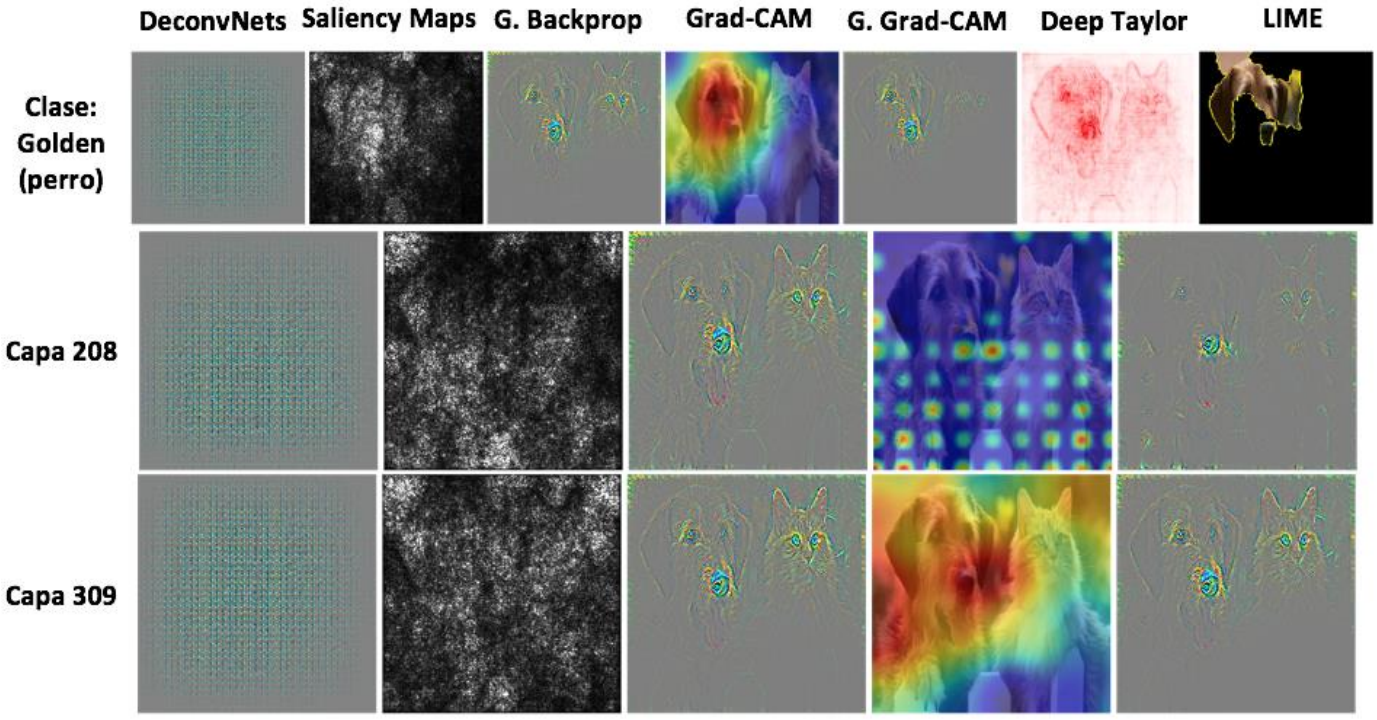

Xception
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Motosierra
Capa 97
Capa 129
ResNet 50
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Motosierra
Capa 106
Capa 165
MobileNet V2
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Motosierra
Capa 120
Capa 150

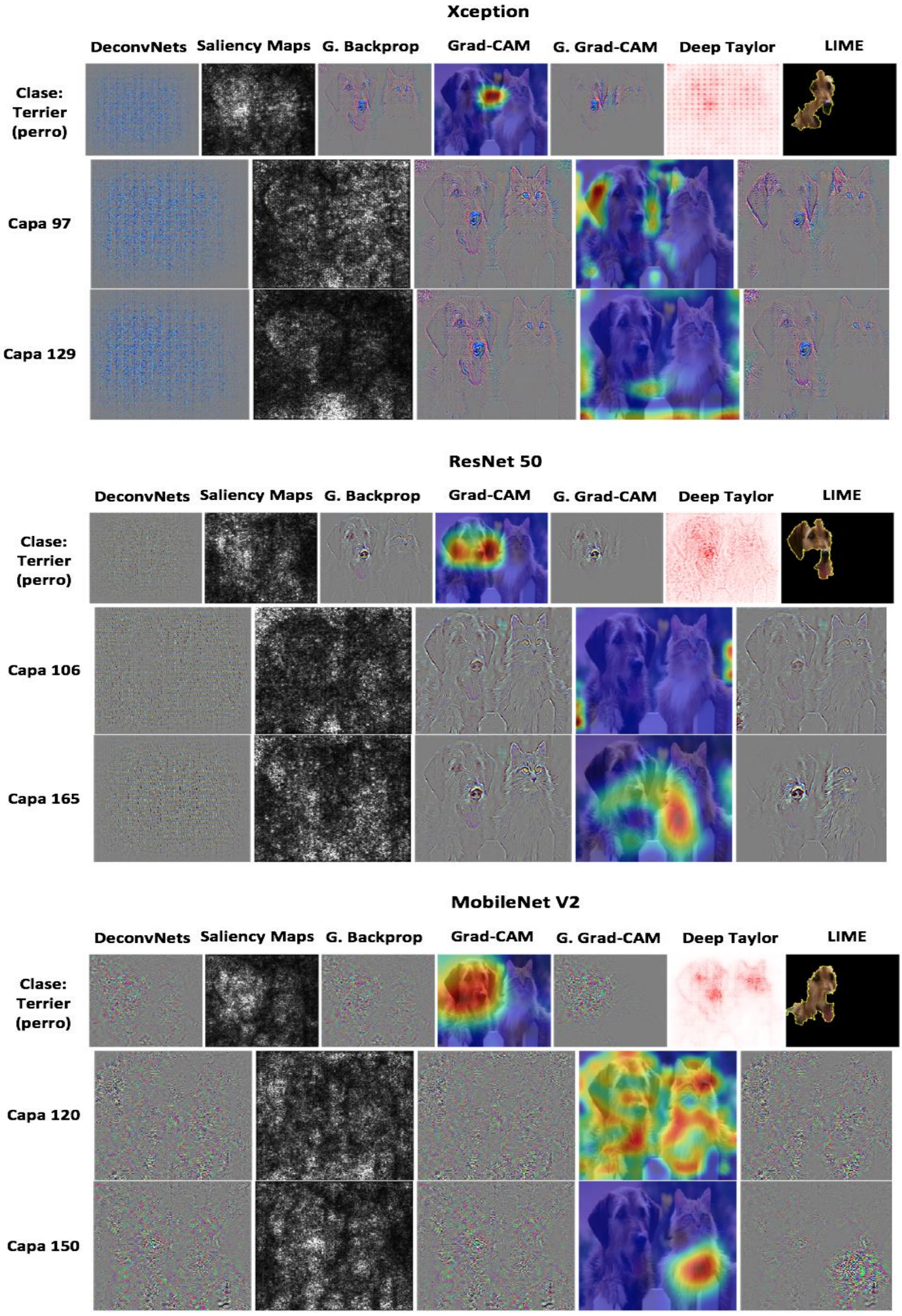
Not from ImageNet

VGG 16



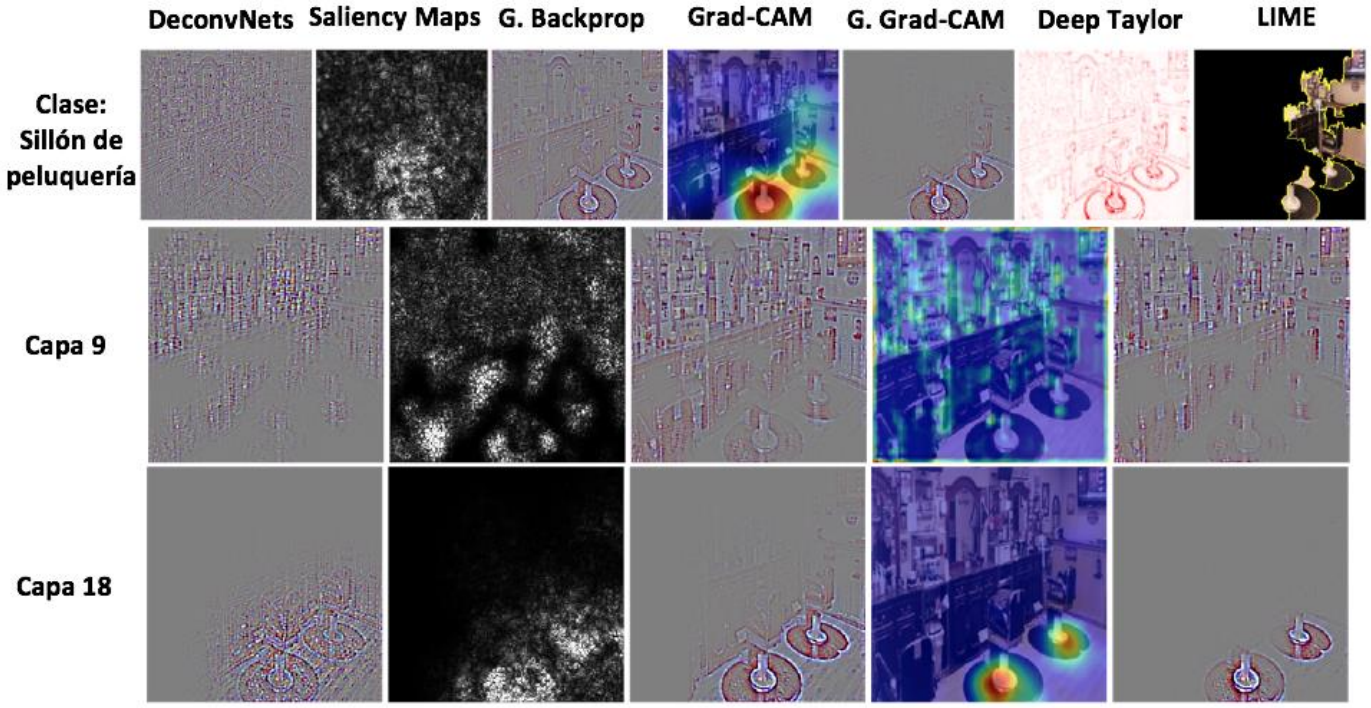
Inception V3



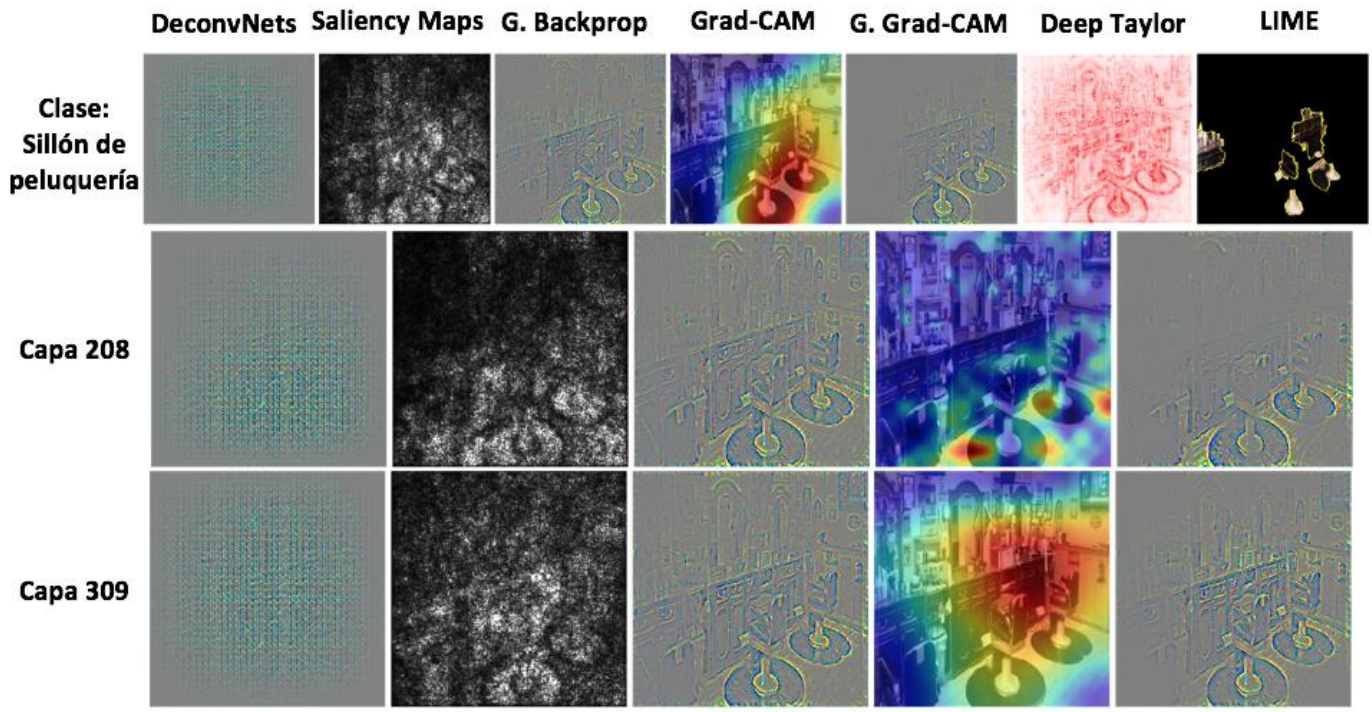

Xception
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Terrier (perro)
Capa 97
Capa 129
ResNet 50
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Terrier (perro)
Capa 106
Capa 165
MobileNet V2
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Terrier (perro)
Capa 120
Capa 150

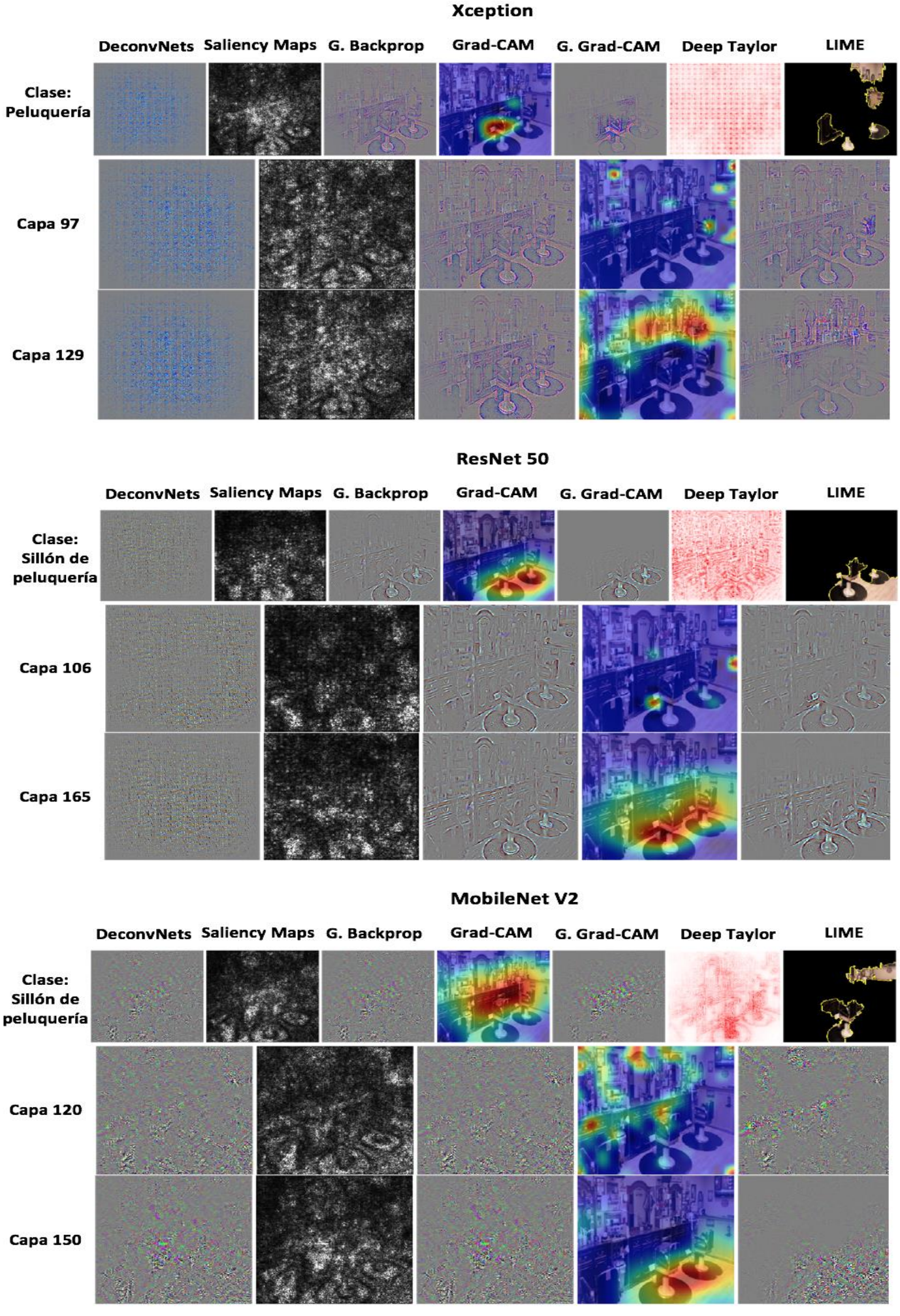
Not from ImageNet

VGG 16



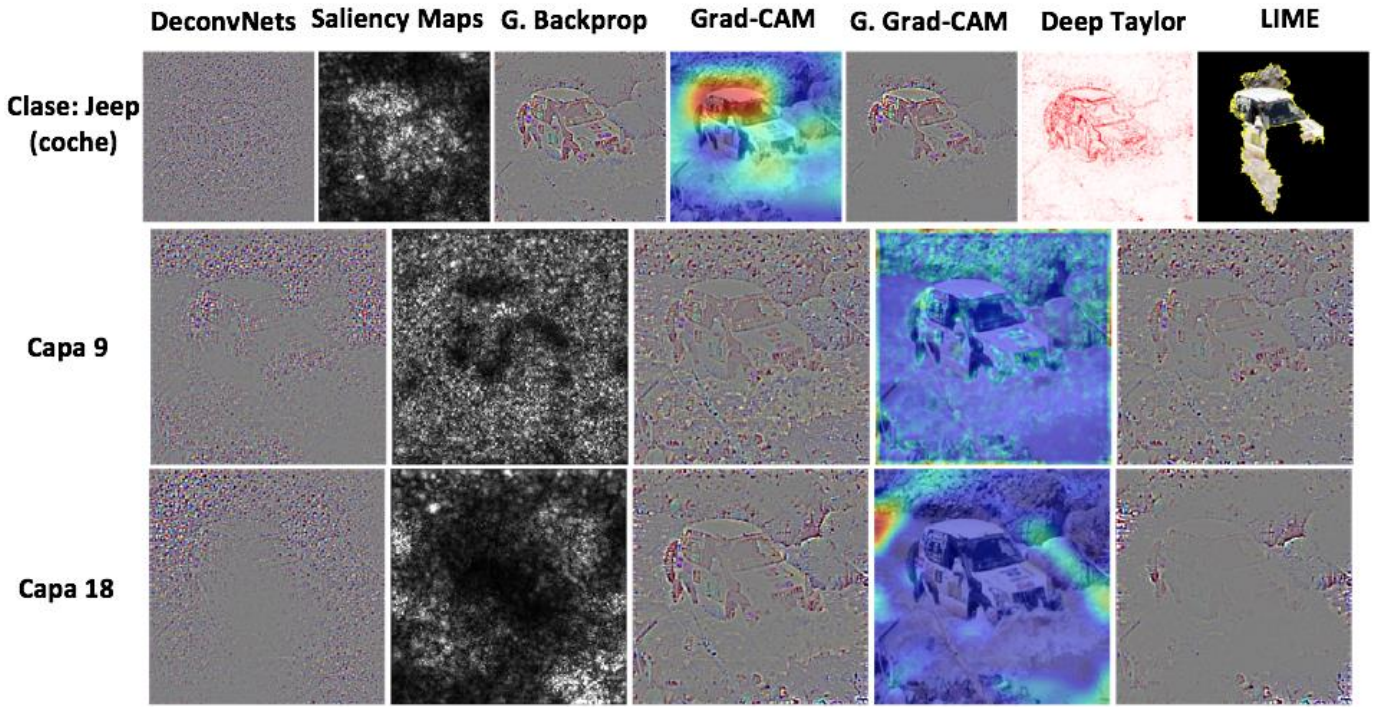
Inception V3



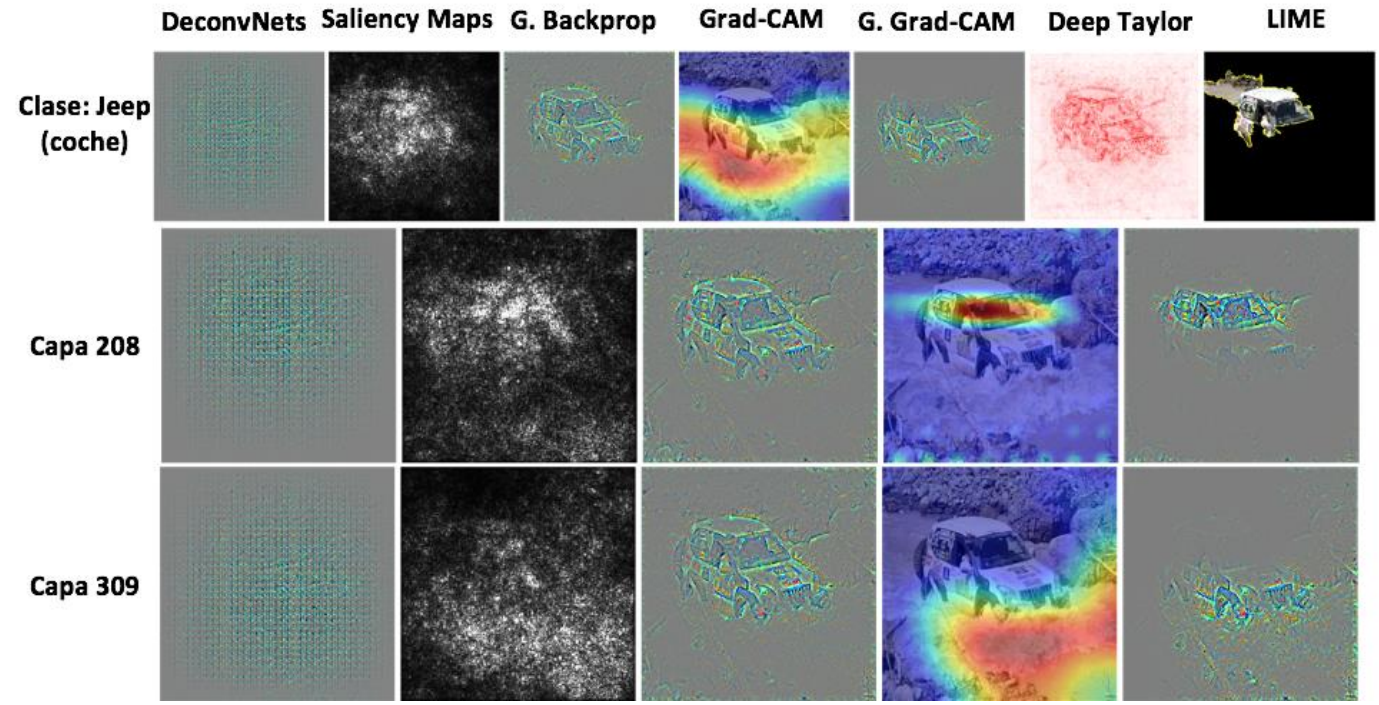

Xception
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Peluquería
Capa 97
Capa 129
ResNet 50
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Sillón de peluquería
Capa 106
Capa 165
MobileNet V2
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Sillón de peluquería
Capa 120
Capa 150

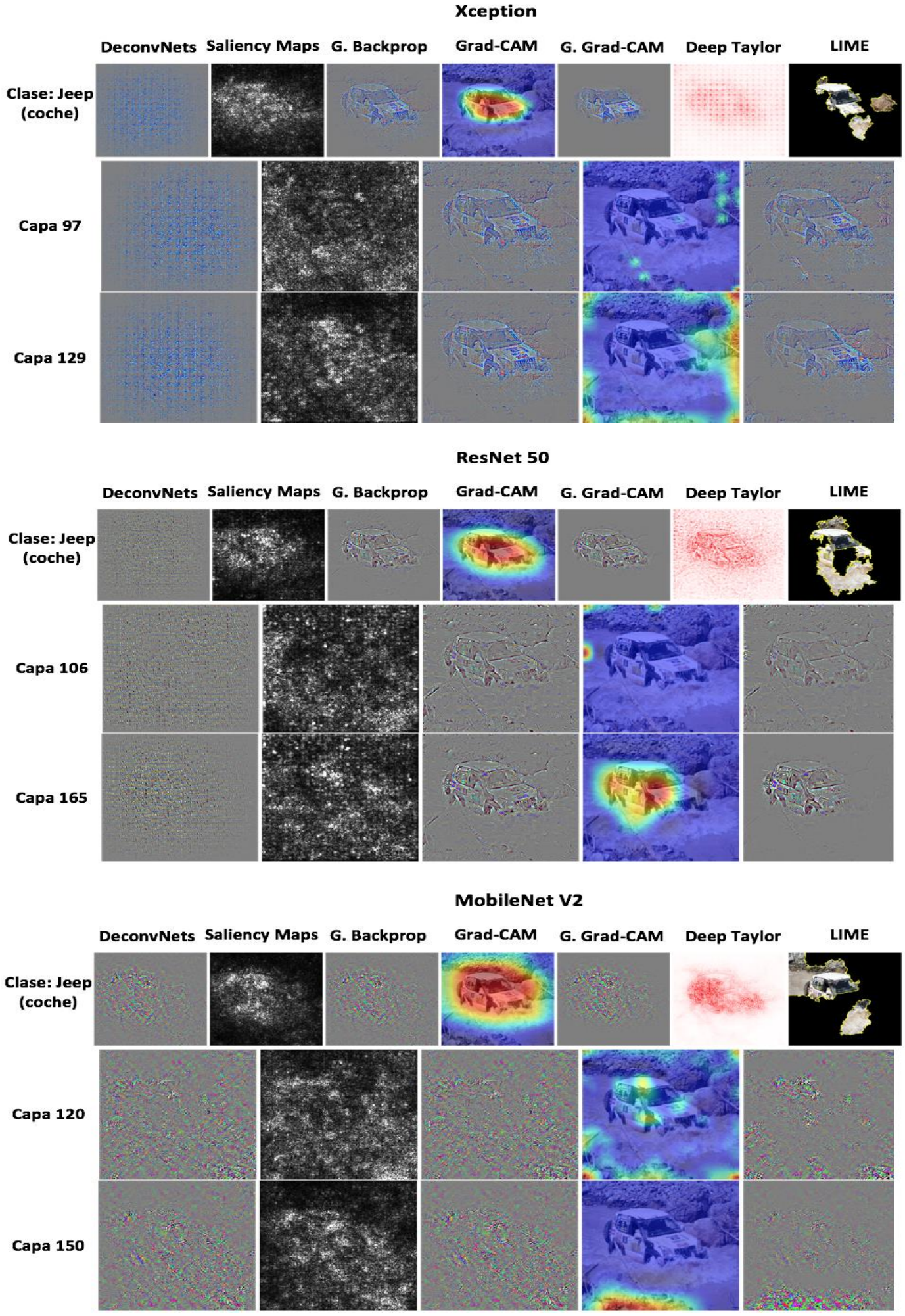
Not from ImageNet

VGG 16



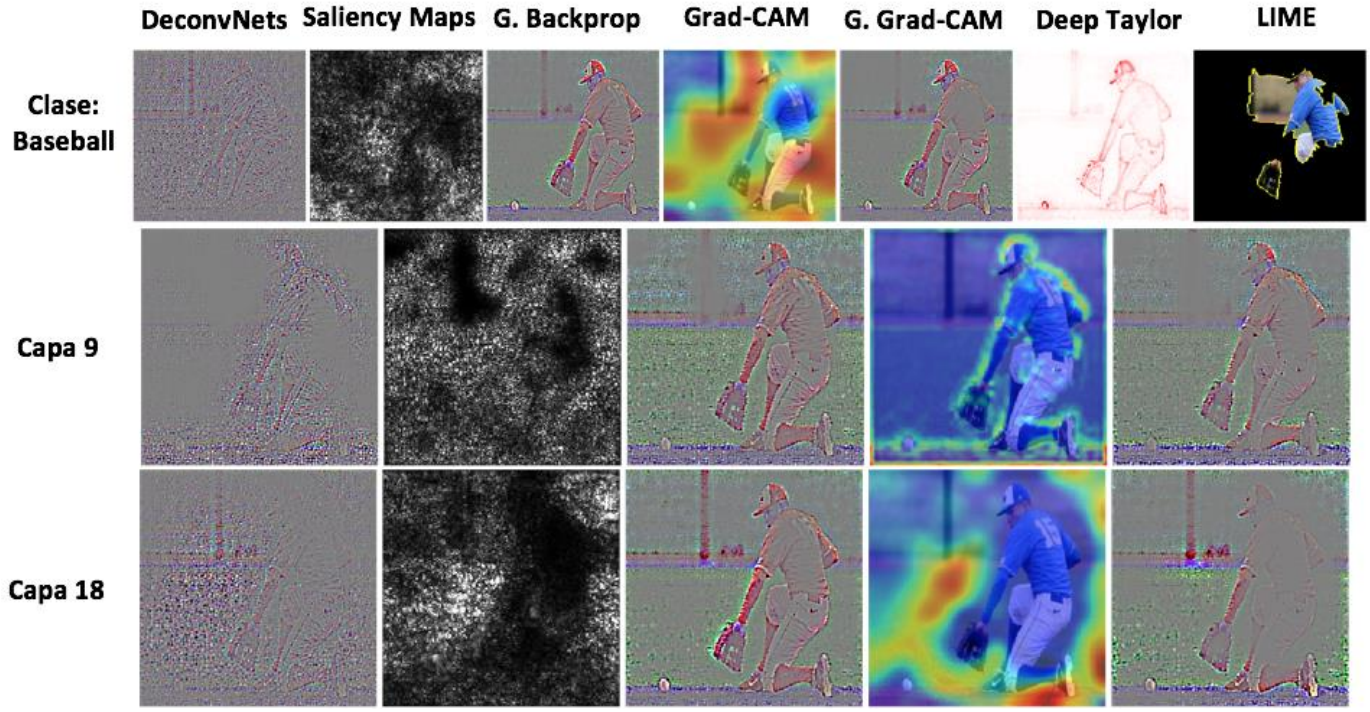
Inception V3



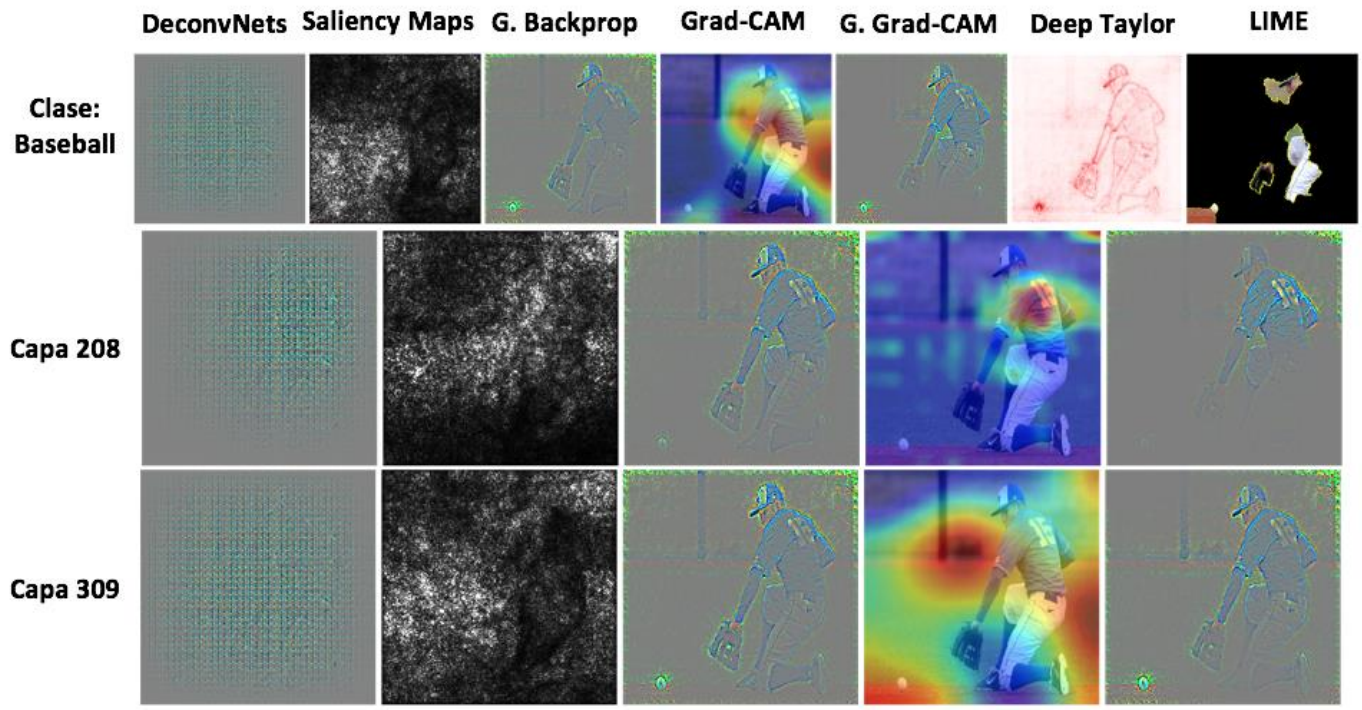

Xception
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Jeep (coche)
Capa 97
Capa 129
ResNet 50
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Jeep (coche)
Capa 106
Capa 165
MobileNet V2
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Jeep (coche)
Capa 120
Capa 150

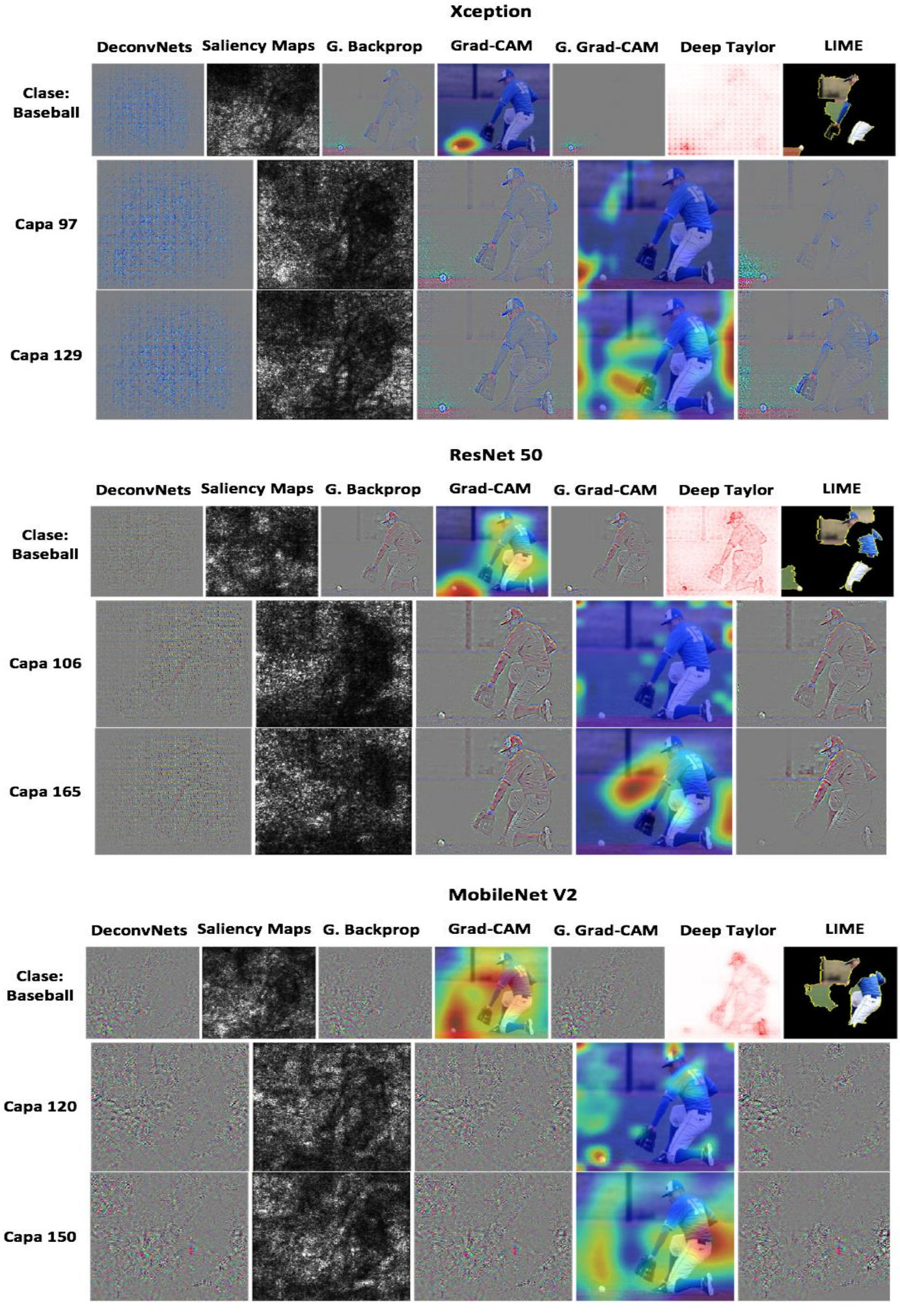
Not from ImageNet

VGG 16



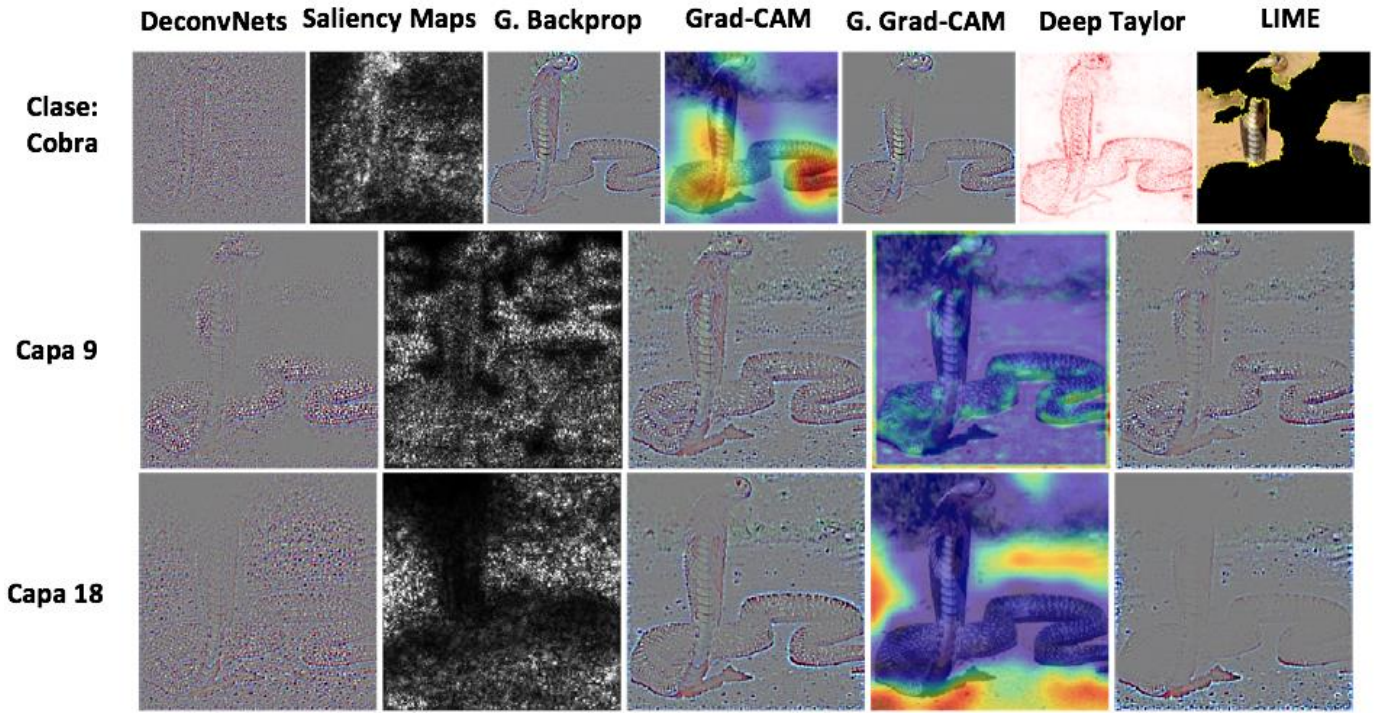
Inception V3



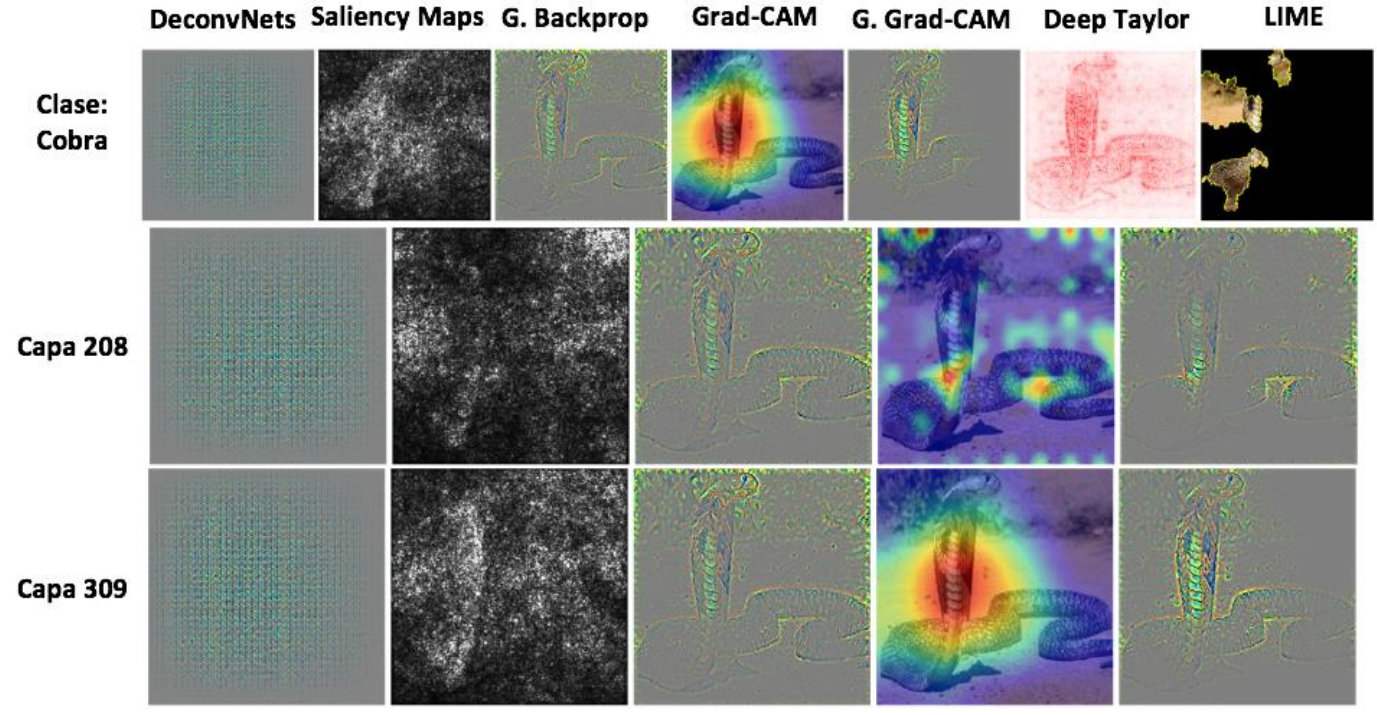

Xception
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Baseball
Capa 97
Capa 129
ResNet 50
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Baseball
Capa 106
Capa 165
MobileNet V2
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Baseball
Capa 120
Capa 150

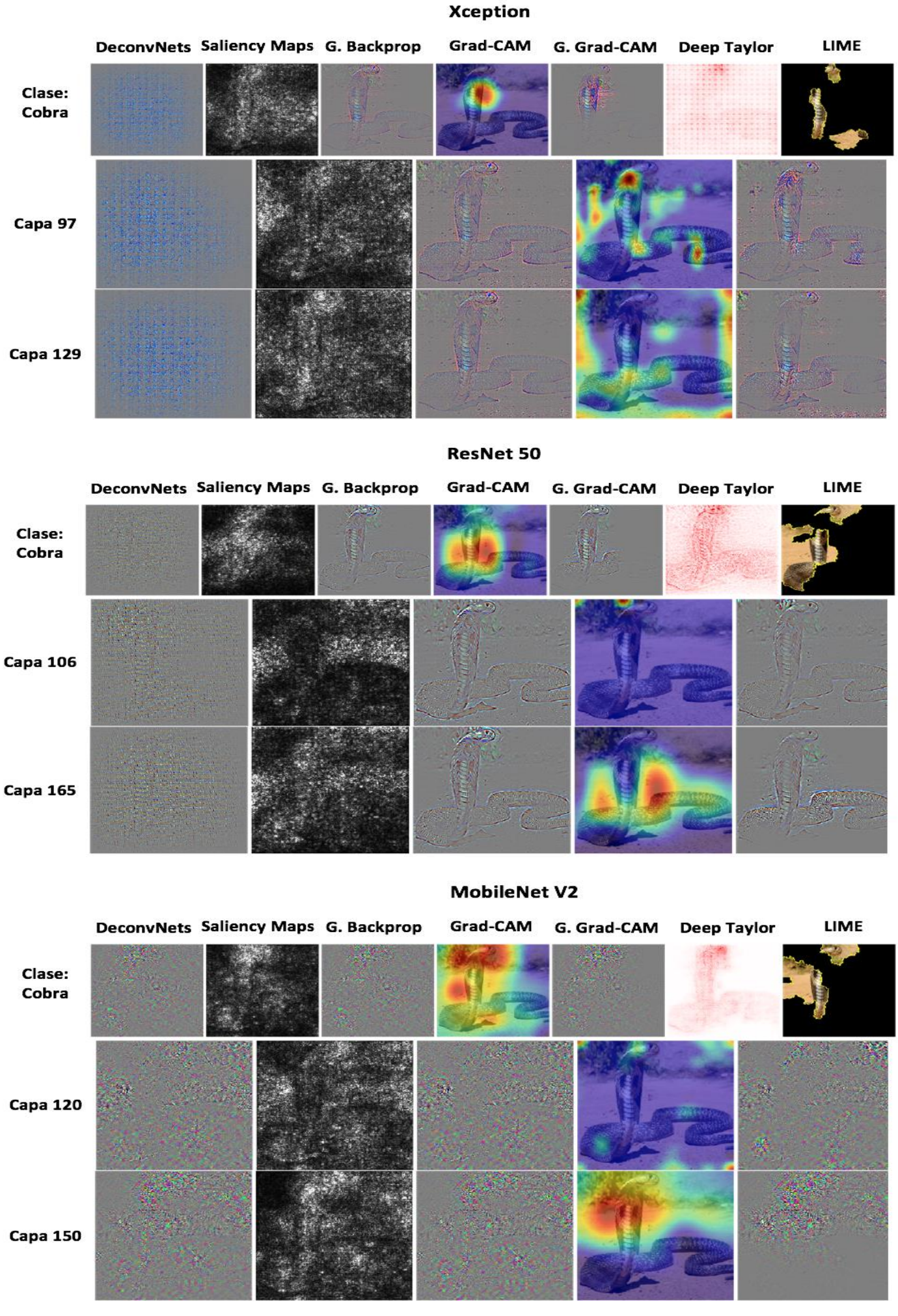
Not from ImageNet

VGG 16



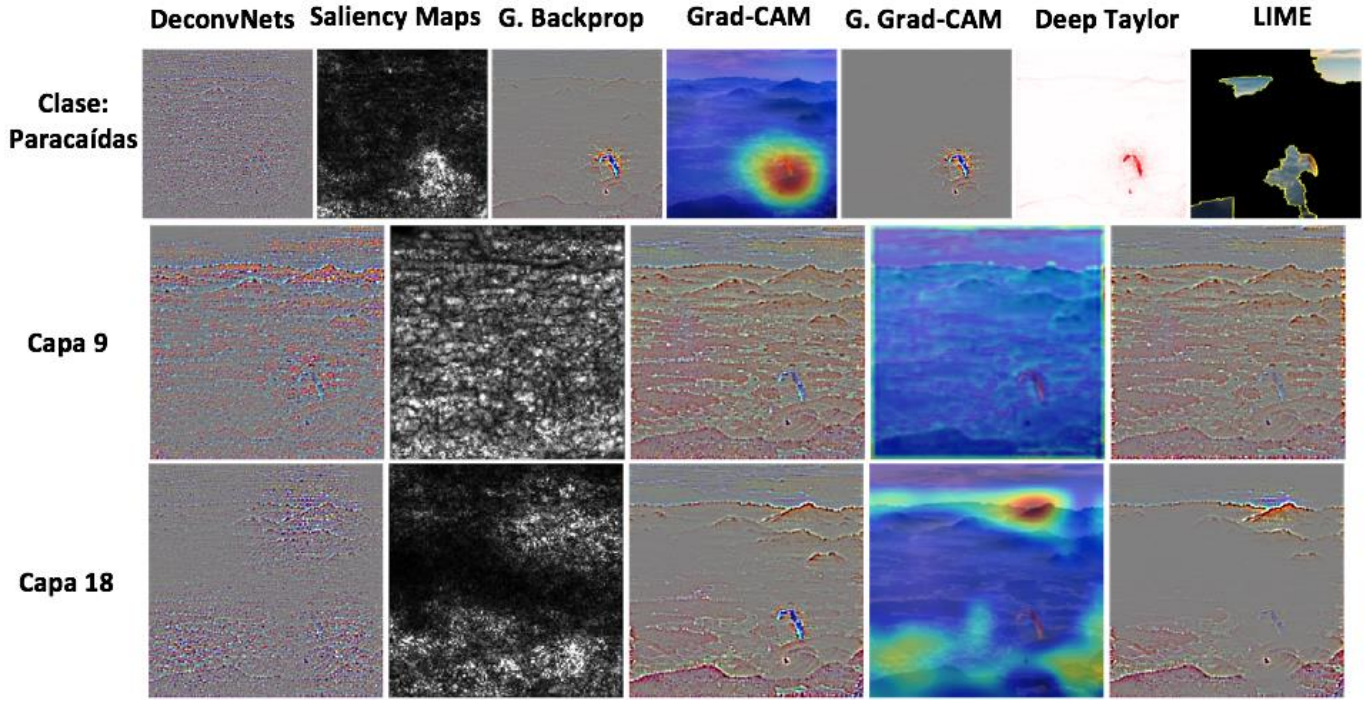
Inception V3



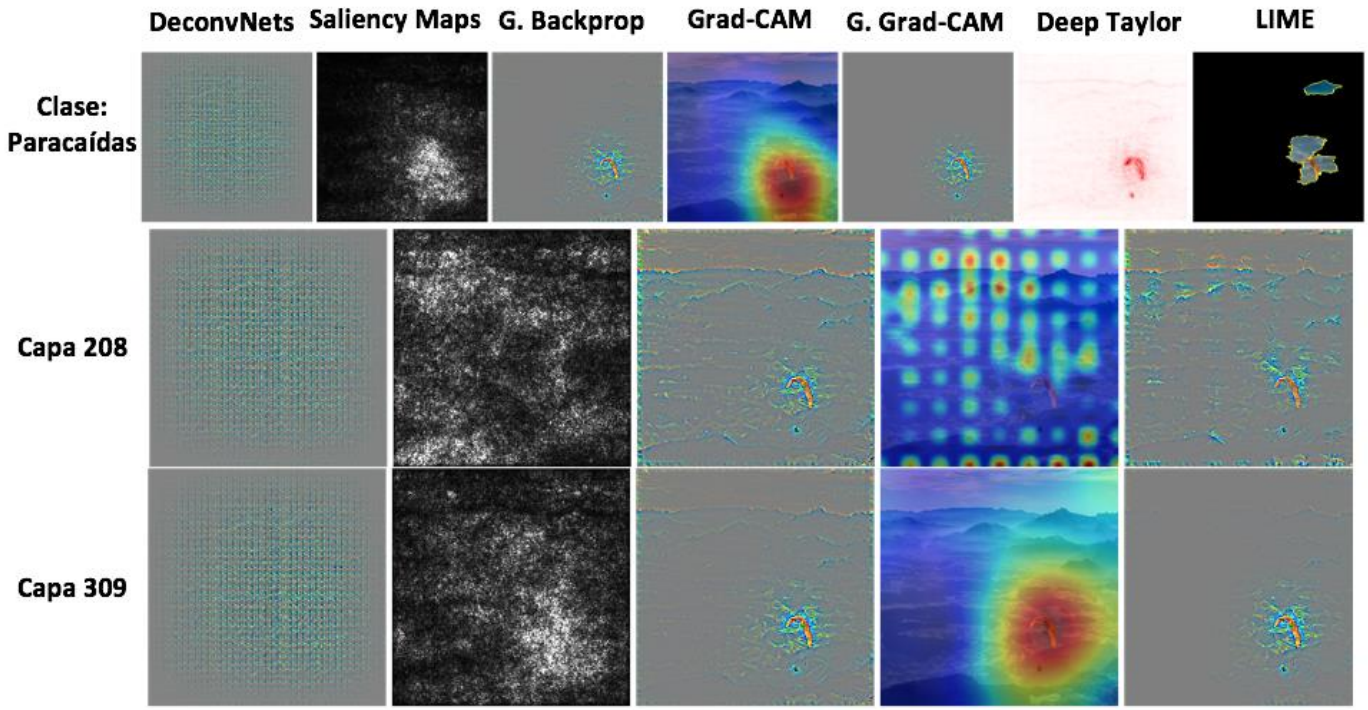

Xception
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Cobra
Capa 97
Capa 129
ResNet 50
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Cobra
Capa 106
Capa 165
MobileNet V2
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Cobra
Capa 120
Capa 150

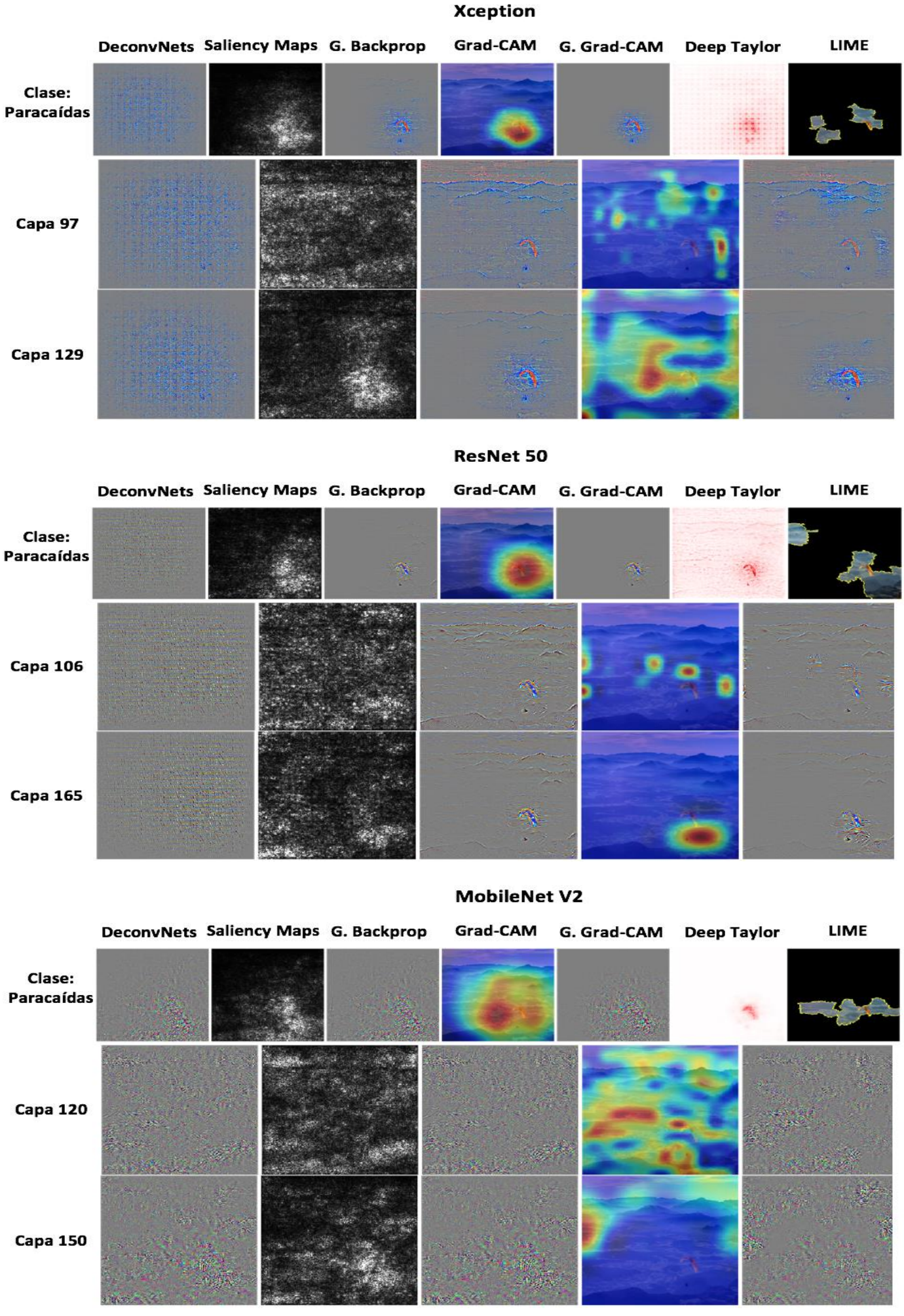
Not from ImageNet

VGG 16



Inception V3




Xception
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Paracaídas
Capa 97
Capa 129
ResNet 50
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Paracaídas
Capa 106
Capa 165
MobileNet V2
DeconvNets
Saliency Maps
G. Backprop
Grad-CAM
G. Grad-CAM
Deep Taylor
LIME
Clase: Paracaídas
Capa 120
Capa 150

Ya tenemos todos los resultados. Como podemos observar, por lo general, los resultados no son nada buenos. No es fácil encontrarles una lógica que explique su comportamiento (el de los métodos). Además, vemos que no es tanto una cuestión del modelo (tipo de arquitectura) o de las imágenes empleadas (si son o no de *ImageNet*) ya que los resultados son muy parecidos en todas las situaciones. Esto de todos modos refuerza nuestra idea de que los métodos son en cierto modo independientes de todo el entorno que les rodea: solo se centran en visualizar lo que detecta una neurona, sin importarles el contexto en el que se haya producido esta activación que han de visualizar.

Vemos como, a pesar de que no hemos mostrado los resultados de visualizaciones de neuronas de capas bajas de las redes, los que sí hemos mostrado de capas intermedias dejan mucho que desear. Bien es cierto que desconocemos el resultado ideal (la visualización real de lo que está detectando la neurona). Tampoco disponemos de ninguna referencia en la que apoyarnos (pues por ejemplo en el caso de las neuronas de salida, tampoco hay forma de saber lo que está detectando realmente la neurona, pero sí tenemos la referencia de lo que debería detectar). En cualquier caso, las visualizaciones obtenidas de estas neuronas no son muy interpretables. Podría darse que estas neuronas estuviesen detectando formas que carecen de sentido para el ser humano, ya que, en el fondo, las neuronas han aprendido solas sin haber sido el objetivo del entrenamiento (no es como las neuronas de salida, donde se 'fuerza' a que detecten las clases que queramos). De todos modos, sí que se debería cumplir por la lógica estructural de una CNN, que las formas detectadas en las últimas capas de la red fueran complejas y por tanto perceptibles/diferenciables en la imagen. En este sentido sí que se puede apreciar que los resultados relativos a las neuronas de las capas intermedias más avanzadas reflejan/destacan zonas más localizadas y no tan dispersas, lo que quiere significar un indicio del correcto funcionamiento de los métodos que proporcionan estos resultados (no todos lo cumplen).

De los resultados correspondientes a la visualización de neuronas de salida sí podemos extraer más conclusiones. Podemos hacerlo precisamente porque tenemos una referencia (el concepto de la clase a visualizar) y trabajamos con un modelo muy bien entrenado (esto implica que en la inmensa mayoría de ocasiones aquello que detecta realmente la neurona se corresponde con la referencia que nosotros tenemos). Así, comparando directamente los resultados obtenidos con nuestra noción de lo que debería ser, podemos evaluar estos resultados. En este aspecto se puede decir que por lo general siempre hay algún método cuyos resultados se adecúan a lo esperado.

Conviene mencionar que las redes en su gran mayoría utilizan una capa final de normalización tras la capa de activaciones de salida (cuando es una red de clases múltiples, claro está), que es la que permite una representación en forma de probabilidades (valores entre cero y uno de manera que todas las probabilidades asignadas a cada clase sumen uno). A esta capa se le denomina *softmax* ya que emplea la función *softmax* para realizar la normalización. Y esta mención viene a cuento ya que, a la hora de realizar la visualización de las neuronas de salida, si visualizamos las salidas tras el *softmax* (las neuronas de

la capa *softmax*), el resultado que obtenemos es realmente malo. Esto se debe a que las salidas de la capa *softmax* no solamente se ven influidas por las activaciones de la propia clase a visualizar, sino también del resto de clases (debido al proceso de normalización). Esto se traduce en que maximizar una salida tras el *softmax* puede lograrse minimizando activaciones que nada tienen que ver con la activación correspondiente a la clase. Es evidente que esto afecta negativamente a los métodos de visualización y en especial, a los sensitivos. Por tanto, los resultados mostrados son todos previos a la operación *softmax* (se visualizan las neuronas de la última capa previa a la *softmax*).

Ahora bien, con todo lo dicho y explicado hasta el momento en este trabajo, lo más reseñable de estos resultados es la ausencia manifiesta de coherencia entre los métodos. Venimos avisando desde la presentación de este estudio experimental que seguramente lo más certero que podríamos sacar en claro de esta comparativa era precisamente esto: la existencia o no de patrones de comportamiento comunes a todos los métodos que hiciese indicar la fiabilidad o no de los mismos. Sin embargo, a la vista está que más bien nos encontramos con todo lo contrario, pues no parece haber muchos casos en los que dos métodos coincidan ni se asemejen siquiera en el resultado. Sabemos por consiguiente que alguno de ellos si no todos están fallando. Pero, retomando la observación de que siempre por lo general hay algún método cuyos resultados se adaptan a lo esperado, podemos determinar con una alta probabilidad de acierto cuáles de ellos son los que fallan.

En concreto, por empezar a poner nombres propios, el método *DeconvNets* no parece dar casi nunca buenos resultados. Y la única arquitectura para la que da resultados descifrables es la *Vgg16*. Incluso para la *Vgg16* las imágenes que devuelve son ruidosas y en ocasiones cuesta distinguir los patrones visuales detectados. Esto se acentúa aún más cuando se utiliza el método para visualizar neuronas de salida. En este caso sí que ya el método actúa meramente como un detector de bordes, sin centrar su atención en ninguna zona específica sino abarcando toda la imagen. Claramente estos resultados no pueden ser válidos. A continuación, se presenta una explicación al respecto.

En un inicio, el método surgió para reconstruir la imagen que provocaba la activación de la neurona a visualizar. Con ese fin, los autores[18] decidieron que la reconstrucción de la capa de activación *ReLU* debía consistir en la propia aplicación de la función *ReLU* para así mantener el signo positivo que toda activación ha de tener. Sin ellos saberlo, lo que estaban haciendo en realidad era un *backpropagation* por la red, en el que lo único que cambiaba era el gradiente de la función *ReLU* (cambiaba el *backward pass* por la capa *ReLU*: ya no se propagaban los gradientes multiplicándolos por la derivada de la función *ReLU* con respecto a la salida de la convolución anterior, sino que directamente se propagaban pasándolos por la función *ReLU*[26]). De esta forma calculaban una especie de "gradiente" de la activación de la neurona a visualizar con respecto de la imagen de entrada, que además obtenía unos resultados que parecían más que aceptables. Y parecían más que aceptables por dos motivos. El primero, por

[26] Nótese que en realidad no es correcto hablar de "gradientes" en el método *DeconvNets*. Abusamos del término para simplificar la explicación.

la resolución que mostraba (ya decimos que, por ejemplo, en el caso de las redes *Vgg16*, visualmente se puede apreciar como es capaz de reconstruir los patrones de la imagen de entrada, aunque no con demasiada calidad). Esta resolución la conseguía precisamente por este peculiar *backward pass* de la operación *ReLU*, que solo permite propagar valores positivos. En breve hablaremos de esto, pero, siguiendo con la explicación, el método parecía más que aceptable también por su capacidad para centrar su atención en zonas específicas de la imagen (discriminativo). Sin embargo, a la vista está que esto último no es realmente así.

La intuición que hemos podido desarrollar en este trabajo nos lleva a pensar que lo que realmente está sucediendo es que, al partir el método de un mapa de activaciones con varios ceros y propagar dichas activaciones hacia atrás hasta la imagen de entrada, sí que podríamos terminar con una imagen que conservase varios de esos ceros (gracias a la espacialidad de las CNNs). Esto suele suceder en capas avanzadas donde las características a detectar son más complejas y por tanto están menos repartidas por la imagen. Pero, cuando la neurona que queremos visualizar pertenece a una capa ya demasiado avanzada de la red, entonces se hace prácticamente imposible lograrlo. Esto explica el porqué de los malos resultados del método visualizando capas muy altas como las evaluadas en las redes de *Inception* o *ResNet*. También por supuesto explica los malos resultados visualizando las neuronas de salida (en ellas se pierde la espacialidad y ya no tiene modo el método de saber qué regiones no activan la neurona). Hay que matizar que sobre todo en redes como *Xception*, *Inception* o *MobileNet* juega un papel aún más importante el tipo de convoluciones que se emplean, que no son contempladas por el método original (convoluciones espacialmente separables, etc), y que a la vista está que producen un mucho peor resultado.

El problema de la incapacidad de centrar la atención en regiones concretas de la imagen también lo sufren el resto de métodos que emplean algún tipo de *backpropagation*. De hecho, se puede ver en los resultados que ninguno de ellos es discriminativo (no se suelen fijar en regiones específicas, sino que tienden a abarcar toda la imagen). Sin embargo, para las neuronas de salida, los resultados de *DeconvNets* aún así son peores (menos discriminativos). Sin duda es porque no utiliza en ningún momento el cálculo de los gradientes que le puedan guiar hacia la solución. Esto provoca que visualice regiones que realmente no influyen en la activación de la neurona objetivo (visualiza igual las regiones que producen ceros en las activaciones que las que no). Esto termina por explicar por qué actúa el método como un simple reconstructor de imágenes, que, ciertamente, en el caso de visualizar neuronas de capas intermedias, algunas veces puede funcionar si parte de un mapa de activaciones con bastantes ceros gracias a la espacialidad de las CNNs. Sin embargo, en otro trabajo como [19] se desliza que la causa de los peores resultados de *DeconvNets* en capas altas, podría deberse a la manera en que se trata la reconstrucción de la capa de *pooling*. Pero esto no resulta muy convincente ya que según hemos visto en el análisis teórico, la reconstrucción de la capa es equivalente a un *backward pass* por la misma y la experimentación realizada no parece respaldar dicha hipótesis (entre otras cosas no recoge el cambio de comportamiento a la hora de visualizar neuronas de salida). En definitiva, el

método *DeconvNets* no parece tener unas bases lógicas que lo sustenten y esto se ve directamente reflejado en los resultados.

A pesar de todo, *DeconvNets* es capaz de reconstruir patrones visuales (aunque solo sea en la red *Vgg16*), cosa que no sucede con *Saliency Maps*, por ejemplo. *Saliency Maps*, que es el *Backpropagation* habitual, obtiene unos resultados realmente pobres. Sus imágenes son excesivamente ruidosas y la reconstrucción de formas/patrones es casi nula (no se pueden ver con nada de nitidez las detecciones). Pese a ser algo que todavía es motivo de investigación ([41][42]), en este trabajo se ha intentado dar una razón a esto.

Lo que se ha deducido es que todo se debe a los tipos de influencia que existen. Cuando calculamos el gradiente de la salida con respecto a la imagen de entrada, lo que estamos haciendo es calcular la influencia que tiene cada característica/píxel de la imagen de entrada en la activación de salida (esa es la motivación del método). Sin embargo, lo que no estamos teniendo en cuenta es que en realidad una red convolucional está tratando todo el rato de detectar características repartidas por toda la imagen. Aunque en una cierta región de la imagen no aparezca absolutamente nada de lo que una neurona está encomendada a detectar, ésta va a intentarlo ya que es su función. Así, los resultados siempre van a venir influenciados por todo tipo de píxeles: tanto los que contienen las formas a detectar como los que no. Esto es un verdadero problema porque durante el proceso estas influencias se mezclan y se hace imposible al final determinar qué regiones son las que verdaderamente contienen las formas detectadas. En otros trabajos, en cambio, la razón que se da a este comportamiento es que la existencia de influencias positivas e influencias negativas (gradientes de distinto signo que representan incidencias positivas y negativas respectivamente en la salida a visualizar) provoca un patrón de interferencia entre ellas que perturba el resultado final.

Sea como fuere, *DeconvNets* evita cualquiera de estos problemas propagando exclusivamente valores positivos. Es por eso que evita interferencias entre distintos tipos de influencias y el resultado que obtiene no es tan caótico, sino que sigue un cierto orden. *Guided Backprop*, que es, recordemos, la combinación de estos dos, obtiene sin duda los mejores resultados de entre los tres. La justificación de sus autores es que el método realiza un *backpropagation* en el que solo se propagan gradientes positivos con el fin de detectar únicamente las influencias positivas en las características de entrada. De esta forma se está evitando también cualquier tipo de interferencia, que nos permite obtener unos resultados claros, nítidos. Sin embargo, el método tampoco es discriminativo, lo que le hace ser poco útil (no es capaz en la mayoría de casos de señalar las zonas/regiones concretas de la imagen que provocan la activación de la neurona a visualizar).

Un método que vemos que sí es discriminativo es *Grad-CAM*. A simple vista podemos ver que es uno de los métodos que mejores resultados obtiene. De nuevo, preguntándonos el por qué, obtenemos la respuesta en el funcionamiento del mismo. Y es que es de los analizados hasta ahora, el único que no realiza *backpropagation* por la red. Se basa directamente en el mapa de activaciones a visualizar y por tanto funciona bien en las capas muy altas en las que hay muchos

ceros en los mapas de activaciones (según acabamos de explicar para el caso de las *DeconvNets*, esto es clave para unos resultados discriminativos). Al redimensionar directamente ese mapa al espacio de entrada, su información espacial se conserva y no se ve alterada por el proceso de *backpropagation*. Como contrapartida tiene que se pierde mucho detalle en las visualizaciones (falta resolución), y los *heatmaps* devueltos no tienen por qué ser del todo precisos (no se está analizando la incidencia de cada píxel individualmente).

Los buenos resultados que se obtienen con *Grad-CAM* se ven reflejados en el método *Guided Grad-CAM*, pues es la combinación de los métodos *Grad-CAM* y *Guided Backprop*. Pero, destacamos que el mayor culpable del buen rendimiento de *Guided Grad-CAM* es *Grad-CAM* más que *Guided Backprop*.

En definitiva, parece claro que los métodos que realizan algún tipo de *backpropagation* por la red, sufren ser no discriminativos. Esta circunstancia también se puede apreciar en los resultados del método *Deep Taylor Decomposition* (que también emula un procedimiento de propagación de valores por la red).

Por último, podemos destacar en cierta medida los resultados de *LIME*. En general vemos como *LIME* casi siempre devuelve resultados bastante aceptables señalando regiones esperadas. Lo que ocurre, es que también muchas veces peca de mostrar regiones de más que no debería (no discriminativo). Pero, al contrario que en el caso de los métodos que realizan *backpropagation*, la culpa de esto no la tiene el funcionamiento interno del método sino su propia naturaleza (es un problema teórico que tiene que ver con el planteamiento y la descripción del método). En concreto, todo viene a la hora de seleccionar los superpíxeles asociados a los pesos más grandes del modelo creado por *LIME*. Tenemos que elegir la cantidad de superpíxeles que queremos visualizar y no tenemos forma de saber qué número es el más adecuado para cada situación. Tampoco podemos establecer un umbral y seleccionar todos los superpíxeles asociados a los pesos que se encuentren por encima de ese umbral, ya que no conocemos de antemano el rango de valores que van a tomar los pesos en ese nuevo modelo. Esto nos lleva a la situación de tener que prefijar el número de superpíxeles a ser visualizados. Por defecto, dicho valor es 5, y de hecho, en los resultados mostrados se ha empleado ese mismo valor. Por tanto, quizá en ocasiones ocurre que esté mostrando menos de lo que debería, pero en la gran mayoría de veces pasa lo contrario: muestra de más.

Otra cosa que debemos de tener en cuenta de este método (*LIME*) es que sus resultados pueden variar a cada iteración, aunque esta variación no suele ser ni mucho menos drástica.

4.3. Comparativa: Resumen

Los resultados que hemos recogido nos permiten evaluar los métodos, y esto es clave en nuestro afán por descubrir cuál de ellos funciona mejor en la práctica.

Para evaluarlos, hemos construido un sistema de puntuaciones. Este sistema otorga un punto a cada método que devuelva un resultado aceptable o que nosotros consideremos bueno. Exploraremos todas las combinaciones [modelo, imagen, neurona] y por cada una de ellas puntuaremos cada método con un 1 si nos parece que su resultado es correcto y un 0 si no. Los resultados de las puntuaciones los agruparemos por arquitecturas, para así poder tener constancia también del comportamiento de cada método con respecto a cada arquitectura, pues este es un factor bien relevante.

El criterio que seguiremos para determinar la corrección de un método es el que hemos ido comentando a lo largo de todo el estudio. Es decir, para las neuronas de salida, compararemos la visualización del método con nuestra propia noción de la clase que representa esa neurona. Y podemos hacerlo porque tenemos la certeza de que los modelos con los que trabajamos están muy bien entrenados y en caso de fallo (esto es, que se estuviera fijando realmente en algo que no debiera para realizar las predicciones, o, equivalentemente, tuviera un concepto/noción equivocada de la clase), éste (el fallo) sería raro, ocasional por lo que no tendría mucha influencia en el análisis. Por otro lado, para las neuronas de capas intermedias, tiraremos de cierta lógica e intuición. Por ejemplo, sabemos que las neuronas de capas altas deberían detectar formas complejas por lo que en el momento en que un método no detecte patrones diferenciables, podemos intuir que no funciona bien. En la gran mayoría de casos se ve claro cuál de los métodos es el que es más que probable que esté funcionando correctamente y cuál no. Obviamente también hay casos en los que resulta muy difícil si no imposible determinar si es que hay algún método que está funcionando bien o no (sobre todo cuando las neuronas detectan patrones visuales sin significado aparente para un humano, o cuando dichos patrones son de tan bajo nivel que resulta complicado entenderlos). En esas situaciones ningún método sumará puntos. En la *Figura 26* se muestra un ejemplo del criterio seguido durante esta evaluación.

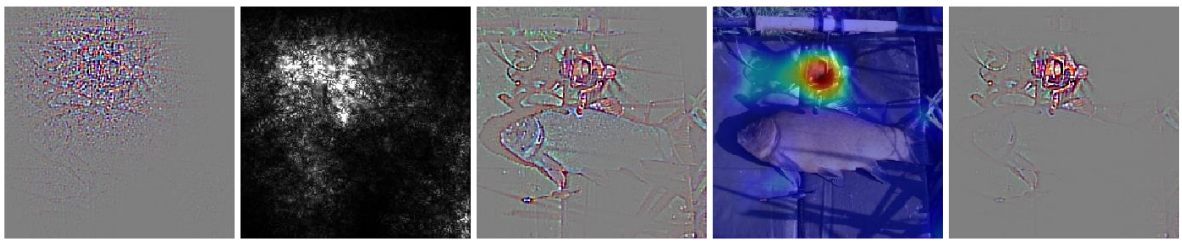


Figura 26. Se muestran los resultados de alguna combinación [modelo,neurona,imagen] que debemos evaluar. Debemos otorgar un punto a los métodos que creamos que lo estén haciendo bien, pero no disponemos de más información que la mostrada en la figura (es una neurona intermedia y por tanto no podemos basarnos en ninguna referencia extra). Parece intuitivo en este caso que los métodos que se llevarían el punto serían los dos de la derecha. Se lo llevarían porque están destacando una región específica de la imagen que parece contener un objeto identificable y por tanto el resultado de los métodos parece tener sentido. En cambio, el resto de métodos no se llevarían ningún punto porque no queda en ningún momento claro aquello que estén detectando: algunos porque no son nada concretos (no discriminativos) y otros porque simplemente es imposible descubrirlo (sus resultados no son interpretables).

Una vez expuesto el sistema de evaluación, podemos pasar por fin a comentar sus resultados. Estos resultados se exponen en la Tabla 2. En la tabla se puede ver por cada método y arquitectura, cuántos aciertos del método hay en los resultados relativos a esa arquitectura. También se puede ver el recuento final de los aciertos totales de cada método en la comparativa.

En la tabla podemos ver reflejado lo que ya hemos hablado en su momento en el apartado de los resultados. Hemos de decir que no hemos sido especialmente estrictos puntuando. Podíamos haberlo sido más y únicamente haber puntuado en aquellos casos en los que estuviera clarísimo que el método estuviera haciéndolo muy bien, pero hemos optado por una mayor flexibilidad porque vemos que es demasiado remoto que esto suceda (que un resultado sea muy bueno). Además, para muchas imágenes de entrada no podemos saber realmente en algunas situaciones si los métodos lo están haciendo bien de verdad o si tan solo están reconstruyendo la imagen y parece que sus resultados son más buenos de lo que en realidad son.

De todos modos, se ve muy claro aquellos métodos que funcionan mejor y peor (en términos relativos con respecto al resto de métodos). Como ya habíamos comentando en el apartado de resultados y queda más que evidente en la tabla, el método *DeconvNets* es el que peores resultados obtiene. Tan solo obtiene puntos de los resultados para el modelo *Vgg16* y no son demasiados. Los pocos que obtiene se deben a su capacidad de reconstrucción gracias al carácter espacial de las convoluciones (ya lo hemos explicado en el apartado anterior), capacidad que se ve mermada cuando se visualizan capas muy altas como en el caso de *ResNet-50*. Para las otras arquitecturas como *Xception*, *Inception* o *MobileNet*, el método tampoco puntúa. Pero, en esos casos ya no es más bien por estar visualizando neuronas de capas muy altas (que también), sino sobre todo por el tipo de convoluciones especiales que realizan estos tres métodos, que dificultan enormemente el proceso de reconstrucción.

También es un hecho lo que comentábamos de que *Guided Backprop* obtiene unos resultados más satisfactorios que *DeconvNets* y *Saliency Maps*. No obstante, en la tabla podemos ver que sale ganador pero no por tanto. Esto sin duda es porque, a pesar de mostrar muy nítidos resultados, no es nada discriminativo y casi siempre sus resultados abarcan toda la imagen, con lo que no podemos saber qué es lo que detecta en realidad. En cambio, los resultados de *Saliency Maps* no tienen una buena resolución pero en ocasiones son más discriminativos. Además, *Guided Backprop* falla con estrépito en la arquitectura *MobileNet*.

Con Deep-Taylor ocurre algo similar. El método presenta unos resultados por lo general nítidos, pero siempre nos queda la duda de si realmente está detectando algo o si está actuando como un mero reconstructor (también es no discriminativo).

La técnica ganadora de la comparativa resulta ser *Grad-CAM*. Es la que más puntos obtiene en la tabla y esto es gracias a algo que la distingue del resto de métodos como es su carácter discriminativo.

	DeconvNets	Saliency Maps	Guided Backprop.	Grad-CAM	Guided Grad-CAM	Deep Taylor Decomp.	LIME
Vgg16	4	6	3	12	13	5	7
Inception V3	0	4	7	9	11	4	8
Xception	0	3	6	9	10	6	7
ResNet 50	0	2	6	13	12	4	7
MobileNet V2	0	2	0	7	3	6	7
Total Aciertos	**4**	**17**	**22**	**50**	**49**	**25**	**36**

Tabla 2. Tabla de evaluación de los distintos métodos de visualización empleados en la comparativa. Las columnas representan los métodos y las filas, las arquitecturas empleadas en la comparativa. Por cada método y arquitectura se muestra la puntuación obtenida por ese método evaluando únicamente resultados obtenidos con esa arquitectura. Al final se hace un recuento para saber los aciertos totales de cada método.

Es pues una evidencia que a nuestro juicio lo más importante que debe cumplir una visualización es ser discriminativa y limitarse a mostrar exclusivamente aquello que importa a la neurona concreta a visualizar. Luego ya en un segundo escalón nos importaría la nitidez y resolución de los resultados. Pero, muy pocos son los métodos que logran esa primera condición y por eso se le da aún más valor a este método.

Grad-CAM obtiene una puntuación muy buena en comparación con el resto de métodos, y, por ende, *Guided Grad-CAM* también. Sin embargo, la puntuación de *Guided Grad-CAM* no tiene por qué ser igual o mejor a la de *Grad-CAM* (por el hecho de ser una combinación de métodos que busca recoger lo mejor de cada uno). Esto se ve en la tabla y es que *Guided Grad-CAM* obtiene menor puntuación total que *Grad-CAM*. Y no es mejor la puntuación porque en muchos casos la información del mapa de calor (*heatmap*) obtenido con *Grad-CAM* se distorsiona al pasarla al método *Guided Grad-CAM*. Es decir, las regiones a las que *Grad-CAM* atribuye una mayor importancia (mayor intensidad en el *heatmap*), no se destacan de la misma manera en *Guided Grad-CAM*. Esto hace al método *Guided Grad-CAM* menos discriminativo.

Por último, *LIME* también puede salir de la comparativa como ganador junto con *Grad-CAM*, ya que su puntuación, teniendo en cuenta que es un método que solo se emplea para visualizar neuronas de salida y no de capas intermedias, es considerablemente buena. En proporción de hecho, obtiene una puntuación muy parecida si no mejor a la que obtendría *Grad-CAM* siendo evaluado exclusivamente en visualizaciones de clase.

Lo que vuelve a quedar claro de todo esto es que los métodos que no realizan ningún tipo de *backpropagation* por la red son los que mejor puntuación obtienen, porque son los más discriminativos.

Otra de las observaciones que nos deja la tabla es que, en general, la arquitectura que peores resultados tiene es la *MobileNet*. Esto puede que se deba simplemente a la propia naturaleza de la red, que como bien sabemos, es bastante más simple que las otras (menos compleja), lo que sacrifica un aprendizaje más potente.

Con todo, no parece por lo común existir una dependencia clara de los métodos de visualización a las arquitecturas usadas (*DeconvNets* es una excepción en este sentido, aunque más bien si hubiésemos optado por visualizar neuronas de capas más bajas y con redes cuyas convoluciones sean las normales[27], presumiblemente hubiéramos obtenido unos resultados bastante similares entre sí).

[27] Con normales nos referimos a que no sean convoluciones separables por profundidad, etc. sino las explicadas en su momento en el apartado de Redes Neuronales Convolucionales en preliminares.

4.4. Otros resultados

En esta sección se presentan muy brevemente resultados obtenidos con otros métodos de visualización de CNNs no incluidos en la comparativa.

Ya decíamos cuando hemos presentado el estudio experimental que existen diversos tipos de métodos de visualización y no podíamos compararlos entre ellos. Por tanto, en los resultados de la comparativa no hemos podido observar cómo se comportan estos otros métodos que no se centran en visualizar neuronas para una imagen concreta de entrada. Es una pena que ni siquiera veamos el tipo de resultados que generan. Con el fin de arrojar algo de luz sobre el asunto, presentaremos al menos un método de los estudiados tanto de visualización de neuronas en términos absolutos (los que generan la imagen que más activa la neurona), como de visualización de conjuntos de neuronas (capas).

Respecto a la visualización de neuronas en términos absolutos, probaremos el método *Activation Maximization*. Dicho método, tal y como hemos comentado en su estudio teórico, devuelve la imagen que más activa la neurona a visualizar, pero esa imagen que devuelve no es demasiado interpretable a ojos de un humano (no es una imagen realista/natural). Es por eso que existían variantes del mismo centradas en mejorar esta faceta. Nosotros en este caso optaremos por emplear algún regularizador. Lo haremos de acuerdo a [43] usando dos regularizadores. Por un lado, uno que mantenga las intensidades de los píxeles en unos márgenes adecuados (que aplique una norma L^p), y por otro, uno que sea el que le aporte realismo (TV- *Total Variation*). Utilizaremos los valores recomendados para los parámetros de estos regularizadores.

Trabajaremos con una red *Vgg16* preentrenada en *ImageNet* e intentaremos ver una neurona de salida para así poder ver la noción de clase que tiene de acuerdo al método de visualización. Antes, nos aseguraremos de que efectivamente esa neurona sea más sensible a imágenes de esa clase que a otras imágenes de otras clases distintas. Y no solo eso, sino que nos cercioraremos de que lo que detecte sea realmente lo que esperamos (la noción que tenemos nosotros de la clase). Esto lo llevaremos a cabo empleando varias imágenes tanto de la clase contemplada como no y el método *Grad-CAM* que aplicaremos sobre las imágenes detectadas como esa clase, ya que, en vista de los resultados de la comparativa, es el más fiable. De esta forma, podremos constatar si existe o no coherencia entre el resultado de la noción de clase según *Activation Maximization* y el resultado de lo que se fija la neurona para una imagen de esa clase. Se supone que *Grad-CAM* se debería fijar en regiones de las imágenes de entrada donde los patrones visuales sean muy similares a los mostrados por la imagen generada por *Activation Maximization*.

El resultado de este proceso se muestra en la *Figura 27*. La clase elegida a visualizar es un pez. Efectivamente vemos como, aunque tampoco de forma muy evidente, sí que se intuye la forma del pez en la imagen generada por el método. También como era de esperar el ratio de aciertos es alto (del 90%). Este ratio refleja en este caso la proporción de imágenes de peces que se encuentran entre las imágenes que el modelo ha clasificado con una mayor confianza (mayor

probabilidad). También los resultados obtenidos por *Grad-CAM* respaldan nuestra hipótesis de congruencia entre los métodos.

Sin embargo, no en todas las situaciones el método se comporta igual de bien digamos. En la *Figura 28* se puede ver un ejemplo de un resultado del método en donde no se intuye la clase visualizada (paracaídas).

Por otra parte, nuestra intención también era abordar un problema de visualización grupal de neuronas. El método que hemos decidido para hacerlo es *VisualBackprop*. Lo hemos elegido ya que es un método utilizado en tareas importantes de visión artificial como la conducción autónoma y porque intenta visualizar predicciones (recordemos que visualiza la última capa convolucional previa a las activaciones de salida con el objetivo de mostrar toda forma visual contemplada en cualquier predicción). Al visualizar predicciones, los resultados obtenidos con el método deberían ser interpretables y las características detectadas diferenciables, distinguibles en la imagen. Además, el funcionamiento interno del método parece más interesante que el de otros como los de *Network Inversion*, que emplean entrenamiento para reconstruir la imagen que produce las activaciones de una capa de la red (al realizar un entrenamiento es como que en cierto modo están interviniendo y guiando demasiado la solución, con lo que lo normal es que terminen con una imagen muy parecida a la original que tampoco nos proporcione excesiva información).

Los resultados de este método se pueden ver en la *Figura 29*. En ella se presentan diez imágenes y para cada una de ellas, el resultado obtenido con el método *VisualBackprop*. Para ser sinceros, los resultados no son muy esclarecedores. No devuelve imágenes con estructuras lógicas detectadas, sino que parece centrarse en zonas de la imagen de entrada de más bien poco interés. Sí es justo decir que para algunas imágenes devuelve resultados con sentido, aunque la resolución de los mismos siempre es bastante pobre. De hecho, es curioso porque peca de ser demasiado discriminativo (y esto según venimos viendo a lo largo de todo el estudio es muy atípico) señalando muy pocas regiones de la imagen que provoquen las activaciones. Que sea tan selectivo en este aspecto hace que se deje partes importantes por señalar, quedándose corto en la explicación (a veces falta información para terminar de entender bien lo que el método está visualizando).

Lo que desde luego sí podemos hacer es confirmar como cierto lo que se dice en la presentación original del método[24] y que ya hemos comentando en su momento en el estudio teórico: la ejecución del método es asombrosamente rápida. Se puede comprender perfectamente por qué es utilizado en aplicaciones como la de la conducción autónoma, donde se require de procesamiento en tiempo real.

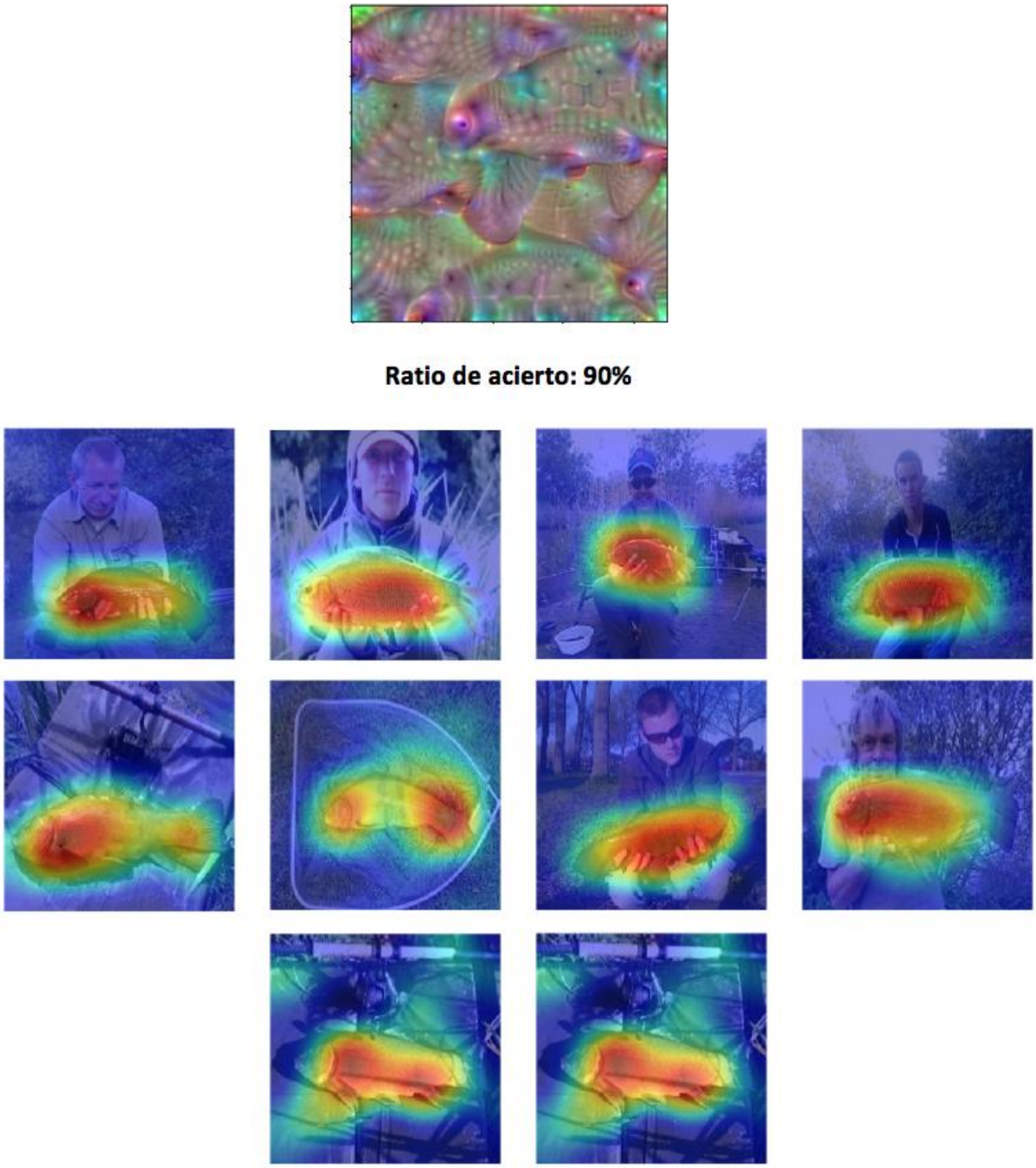


*Figura 27. (*Arriba) Resultado del método *Activation Maximization* visualizando la clase 'Tenca (pez)' en una red *Vgg16* entrenada en *ImageNet*. (Abajo) También se muestra, de entre las imágenes que más deberían estimular la neurona visualizada, cuántas realmente así lo hacen (en porcentaje). Para las que efectivamente lo hacen, se muestra el resultado de la visualización del método *Grad-CAM*. Así podemos comparar los resultados de los métodos *Grad-CAM* y *Activation Maximization* y ver que son congruentes (los patrones que más estimulan una neurona son los que luego se resaltan cuando se visualiza una imagen concreta).

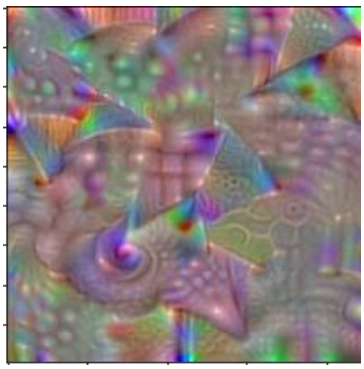


Figura 28. Resultado del método *Activation* Maximization visualizando la clase 'paracaídas' en una red *Vgg16* entrenada en *ImageNet.*

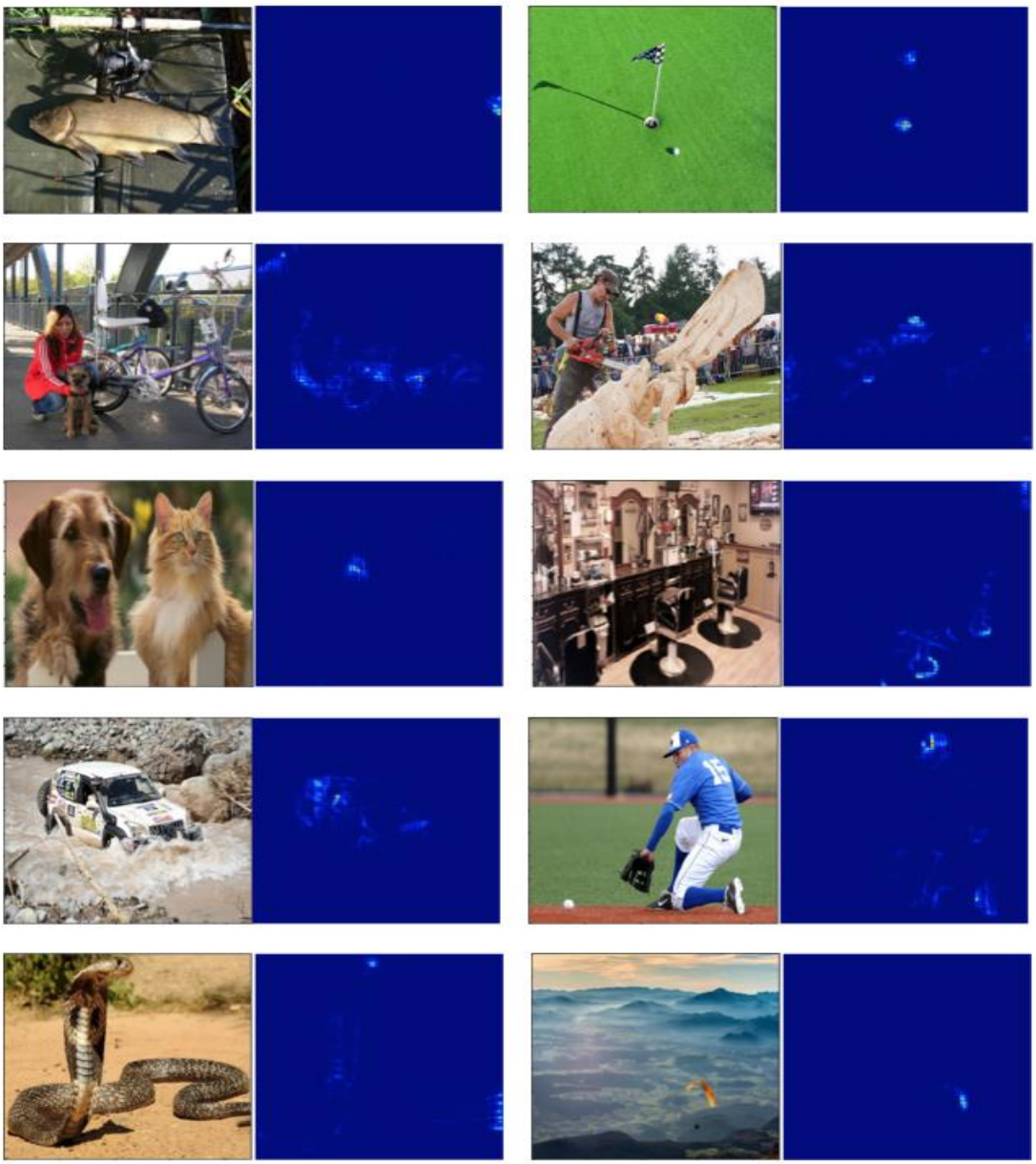


Figura 29. Resultados obtenidos con el método VisualBackprop en un modelo Vgg16 entrenado en ImageNet. Por cada imagen se muestra el resultado obtenido con el método.

5. Conclusiones y líneas futuras

Hemos alcanzado el final de este trabajo. Empezamos el trabajo hablando de la cada vez más influencia y relevancia de la Inteligencia Artificial en nuestras vidas. En concreto, del Deep Learning y sus múltiples aplicaciones en cualquier ámbito de la vida, y, sobre todo, en aplicaciones que requieren de algún tipo de visión artificial. Para estas aplicaciones se suelen emplear las Redes Neuronales Convolucionales. Se hace necesario entonces algún mecanismo que nos permita controlar y comprender en todo momento el funcionamiento de estas redes. Con este fin surgen los métodos de visualización de Redes Neuronales Convolucionales.

El siguiente paso que hemos dado ha sido estudiar estos métodos, comprender la teoría de los mismos y comparar sus propiedades. Posteriormente ya los hemos sometido a un estudio experimental. Todo para lograr sacar algo en claro acerca de la potencial utilidad de todas o alguna de estas técnicas.

Pues bien, lo que hemos concluido ha sido que en general su uso todavía no parece ser muy ventajoso. Los resultados que se obtienen con la mayoría de métodos no nos ayudan demasiado, pues por lo común son bastante menos precisos de lo esperado. Nuestro criterio nos dice que no son muy buenos ya sea porque no son discriminativos (no se fijan en regiones concretas que nos permitan distinguir los patrones detectados de los no detectados) o porque su resolución es muy baja (los resultados no son lo suficientemente claros, nítidos como para que un humano pueda interpretarlos sin problema). También son malos cuando no se ajustan a nuestras expectativas. Pero lo realmente más preocupante es la ausencia de congruencia entre los resultados de los distintos métodos, es decir, que no exista un consenso entre los métodos. Esto no hace sino indicar que están fallando.

Y en este trabajo hemos procurado justificar y razonar el porqué de semejantes resultados. Hemos intentado analizar los resultados particulares de cada método basándonos en su funcionamiento y tratando de encajar todas las piezas del puzzle (las explicaciones de los resultados de un método no entren en contradicción con las de otro, sino al contrario, se complementen). Sin embargo, es inevitable que queden en el aire preguntas por responder. Por ejemplo, ¿Por qué el método *Guided Backprop* es tan poco discriminativo si sus autores justifican su buen funcionamiento con respecto a otros como *DeconvNets* o *Saliency Maps* alegando que solo visualiza lo que influye positivamente en la neurona? Muchas de estas cuestiones son las que hacen que continuamente se estén presentando nuevos métodos y que éste sea un tema muy sujeto a la investigación.

Es verdad que no todo es negativo. Se puede rescatar que algún que otro método funciona aceptablemente bien (*Grad-CAM* o *LIME*) y podría ser utilizado con cautela (siempre siendo críticos con los resultados y no fiándonos ciegamente de los mismos). Aún y todo, estos métodos “rescatables” también tienen alguna pega inherente a sus bases/principios teóricos (baja resolución en los resultados, etc).

En resumidas cuentas, lo que resulta claro es que todavía queda mucho trabajo por hacer en este tema. Un asunto tan sumamente importante como es el de la visión artificial en nuestras vidas, merece dedicación por tratar de conseguir romper la barrera de la interpretabilidad que tienen los modelos de Deep Learning en ella empleados, mediante algún método de visualización que, de momento, no existe.

6. Bibliografía

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “A B7CEDGF HIB7PRQTSUDGQICWVYX HIB edCdSISIXvg5r ` CdQTw XvefCdS,” *proc. IEEE*, 1998.

[2] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolutional neural networks,” 2012.

[3] C. Chung *et al.*, “Very Deep Convolutional Networks for Large-Scale Image Recognization,” *[Vgg]*, vol. 75, no. 6, pp. 398–406, 2018.

[4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016.

[5] C. Szegedy *et al.*, “Going deeper with convolutions,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 1–9, 2015.

[6] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 1800–1807, 2017.

[7] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017.

[8] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” *Proc. Thirteen. Int. Conf. Artif. Intell. Stat.*, vol. 9, pp. 249–256, 2010.

[9] A. Agarwal, S. Negahban, and M. J. Wainwright, “A Simple Way to Prevent Neural Networks from Overfittin,” *Ann. Stat.*, vol. 40, no. 2, pp. 1171–1197, 2012.

[10] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” 2015.

[11] K. Grill-Spector and R. Malach, “The human visual cortex.,” *Annu. Rev. Neurosci.*, vol. 27, pp. 649–77, 2004.

[12] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, “Visualizing Higher-Layer Features of a Deep Network Technical Report 1341 The models,” *Bernoulli*, no. January, pp. 1–13, 2009.

[13] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps,” pp. 1–8, 2013.

[14] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, “Understanding Neural Networks Through Deep Visualization,” 2015.

[15] A. Nguyen, J. Yosinski, and J. Clune, “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 427–436, 2015.

[16] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune, “Synthesizing the preferred inputs for neurons in neural networks via deep generator networks,” no. Nips, pp. 1–29, 2016.

[17] M. D. Zeiler, G. W. Taylor, and R. Fergus, “Adaptive deconvolutional networks for mid and high level feature learning,” *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 2018–2025, 2011.

[18] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional

networks," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8689 LNCS, no. PART 1, pp. 818–833, 2014.
[19] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity," pp. 1–14, 2015.
[20] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K. R. Müller, "Explaining nonlinear classification decisions with deep Taylor decomposition," *Pattern Recognit.*, vol. 65, pp. 211–222, 2017.
[21] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," pp. 1–9, 2018.
[22] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-Octob, pp. 618–626, 2017.
[23] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?': Explaining the Predictions of Any Classifier," 2016.
[24] M. Bojarski *et al.*, "VisualBackProp: efficient visualization of CNNs," 2016.
[25] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 5188–5196, 2015.
[26] A. Dosovitskiy and T. Brox, "Inverting visual representations with convolutional networks," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 4829–4837, 2016.
[27] "https://github.com/raghakot/keras-vis." [Online]. Available: https://github.com/raghakot/keras-vis.
[28] "https://github.com/Lasagne/Recipes/blob/master/examples/Saliency%20Maps%20and%20Guided%20Backpropagation.ipynb." [Online]. Available: https://github.com/Lasagne/Recipes/blob/master/examples/Saliency Maps and Guided Backpropagation.ipynb.
[29] "https://github.com/eclique/keras-gradcam/blob/master/gradcam_vgg.ipynb." [Online]. Available: https://github.com/eclique/keras-gradcam/blob/master/gradcam_vgg.ipynb.
[30] "https://github.com/ramprs/grad-cam." [Online]. Available: https://github.com/ramprs/grad-cam.
[31] "https://github.com/jacobgil/keras-grad-cam." [Online]. Available: https://github.com/jacobgil/keras-grad-cam.
[32] "https://github.com/InFoCusp/tf_cnnvis." [Online]. Available: https://github.com/InFoCusp/tf_cnnvis.
[33] "https://github.com/piergiaj/caffe-deconvnet." [Online]. Available: https://github.com/piergiaj/caffe-deconvnet.
[34] "https://github.com/saketd403/Visualizing-and-Understanding-Convolutional-neural-networks." [Online]. Available: https://github.com/saketd403/Visualizing-and-Understanding-Convolutional-neural-networks.
[35] "https://github.com/marcotcr/lime." [Online]. Available: https://github.com/marcotcr/lime.
[36] "https://github.com/albermax/innvestigate." [Online]. Available: https://github.com/albermax/innvestigate.

[37] “https://github.com/experiencor/deep-viz-keras.” [Online]. Available: https://github.com/experiencor/deep-viz-keras.
[38] “https://github.com/sar-gupta/convisualize_nb.” [Online]. Available: https://github.com/sar-gupta/convisualize_nb.
[39] “https://github.com/utkuozbulak/pytorch-cnn-visualizations.” [Online]. Available: https://github.com/utkuozbulak/pytorch-cnn-visualizations.
[40] “Imagenet.” [Online]. Available: http://www.image-net.org.
[41] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic Attribution for Deep Networks,” 2017.
[42] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, “SmoothGrad: removing noise by adding noise,” 2017.
[43] A. Mahendran and A. Vedaldi, “Visualizing Deep Convolutional Neural Networks Using Natural Pre-images,” *Int. J. Comput. Vis.*, vol. 120, no. 3, pp. 233–255, 2016.