Visión por Computador
Práctica Final

Guillermo Gómez Trenado | 77820354-S
guillermogotre@correo.ugr.es
José Antonio Ruiz Millán | 26503402-L
jantonioruiz@correo.ugr.es

18 de enero de 2019

Índice

1. Introducción

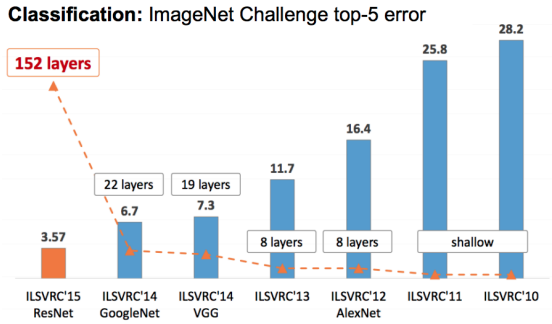
Las redes convolucionales profundas han experimentado un auge en la popularidad para resolver infinitos problemas de visión por computador desde que en 2012 *AlexNet*[1] diera un golpe en la mesa en la competición sobre el dataset *Imagenet*, con ocho etapas de convoluciones, que sentaba una nueva tendencia en la resolución de este tipo de problemas y que no ha hecho más que ganar en popularidad durante estos años, debido al uso de ReLU como función de activación no lineal en vez de la sigmoidal o funciones parecidas que permite la propagación hacia atrás sin degradarse durante más capas, y a la explosión de capacidad computacional que introducen las GPU frente al procesamiento clásico en CPU.

Sin embargo, las CNN funcionan a modo de caja negra, se entrenan durante suficiente tiempo con muchos ejemplos, se almacenan los pesos de las conexiones y se utilizan estos para el cálculo posterior de nuevas instancias. Comprender el funcionamiento y el razonamiento de estos modelos es tremendamente complicado, especialmente porque las CNN unen las dos etapas de extracción de características y clasificación en un único modelo —CNN + *Fully connected layers*— frente a los modelos clásicos donde la extracción de características se hacía a mano y podíamos ir evaluando de forma alternativa el impacto de éstas en el resultado. Por esto, desde la explosión en popularidad de las CNN en el 2012 se han abierto dos vías paralelas de investigación, una que intenta explotar el potencial de esta arquitectura, como *Resnet*[2] con la introducción de capas residuales —luego analizaremos—, y otra orientada a la visualización del contenido de las redes y su funcionamiento en cada instancia; es sobre esta última sobre la que realizamos nuestro trabajo. Nos proponemos implementar la estrategia para visualización de *guided backpropagation* publicada por Springenberg[3] además de otras más sencillas como la estimación de áreas críticas por medio de ventana deslizante y una adaptación de Class Activation Maps[4] utilizando un valor extraido a partir de *guided backpropagation* como peso.

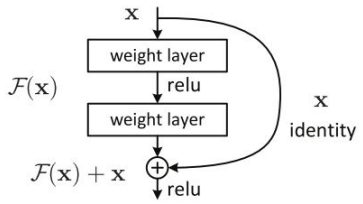
[1] Krizhevsky et al (2012)
[2] He et al.(2016)
[3] Springenberg et al.(2014)
[4] Zhou et al.(2016)

2. Arquitectura escogida

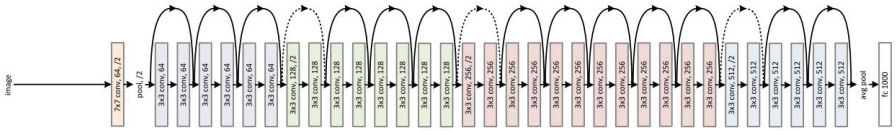
2.1. Modelo escogido



Todos los *papers* antes descritos utilizan AlexNet como arquitectura de soporte, sin embargo actualmente este diseño ha dejado de ser competitivo y el uso de capas residuales ha definido un estándar en el diseño de nuevas redes, pues facilita la propagación hacia atrás permitiendo así modelos mucho más profundos.



Vamos a adaptar por lo tanto los algoritmos definidos en las publicaciones antes descritas a ResNet, en su versión de 34 capas de convoluciones



La red se caracteriza por utilizar Batch Normalization después de cada convolución, y agrupa la estructura en 5 grupos, el primer grupo aplica una única capa de convoluciones de tamaño 7x7, stride igual a 2 y padding de 3 y finalmente maxpooling, esta es la reducción más agresiva de tamaño transformando las imágenes originales de 224 píxeles en 64 mapas de 56x56; las siguientes capas son estructuralmente similares, la primera capa residual —en la imagen las vemos con la línea punteada— reduce el tamaño del volumen de entrada (X_{in}) no mediante maxpooling sino con stride igual a 1 en la primera capa de convolución y para sumar el volumen anterior X_{in} de dimensiones distintas, aplica una capa de convolución de tamaño 1 y stride 1 —tomar sólo los valores impares— sobre X_{in} con dimensión de salida igual a la dimensión de salida de la última convolución de la capa residual, el resto de capas residuales del grupo aplican el mismo procedimiento sin reducción de la dimensionalidad.

Luego profundizaremos un poco más en las capas residuales y en las funciones de activación porque serán clave para la implementación de *guided backpropagation*.

2.2. Software escogido

Para este trabajo hemos utilizado PyTorch como plataforma de desarrollo, por dos motivos, el primero es que al contrario de TensorFlow, Pytorch no se basa en la compilación del grafo definido por la CNN para calcular la derivada parcial en cada punto de forma estática y llamarla en cada *back-propagation*, sino que cada nodo es un *tensor* que almacena su historial de operaciones y calcula el gradiente de forma dinámica, la primera implicación de esto es que para llevar el recorrido a través de MaxPool de la entrada de la capa anterior no tenemos que almacenar una máscara de correspondencia complementaria —que haga corresponder cada pixel tras MaxPool con el pixel antes de éste—, sino que cada valor en los mapas de activación de salida *sabe* si ha sido él el que ha pasado por la etapa de *pooling*, esto es tremendamente útil para la implementación de *guided backpropagation* como veremos más adelante. La segunda razón es que incluye en la propia libreria de PyTorch modelos preentrenados sobre ImageNet, lo cual no es una tarea trivial, y que permiten modificar los elementos individualmente y añadir puntos de anclaje para modificar los valores en vivo. Pytorch está implementado íntegramente en Python, al contrario que Tensorflow que expone una API para su implementación en C, lo cual hace que sea significativamente más lento, aunque el uso de CUDA agiliza mucho la ejecución, pero desarrollar en este framework es mucho más ágil e intuitivo.

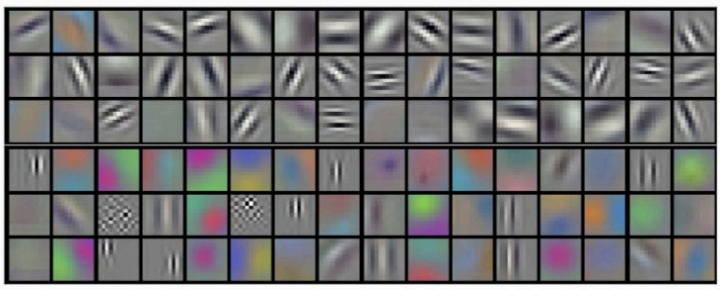
3. Visualización de redes

3.1. Primeros intentos de visualización

Históricamente, los primeros intentos de visualización del funcionamiento de estas redes, seguían tratando la CNN como una caja negra y actuaban antes y después modificando los valores de entrada y viendo cómo se modifica la salida, o sencillamente analizando los mapas de activación tras la aplicación de la función $ReLU$ e intuyendo lo que están evaluando.

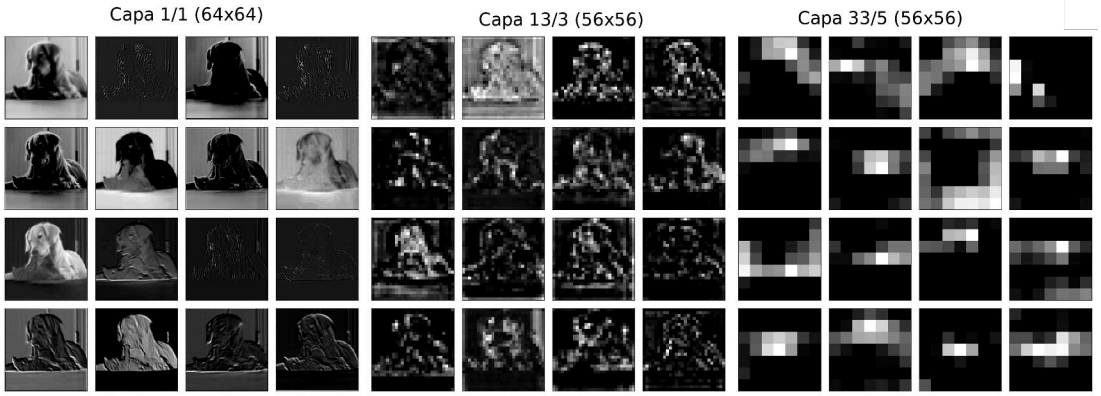
Una primera estrategia frecuente es visualizar los kernels de la primera capa ($k \times k \times 3$) como una imagen de tres canales. Aquí un ejemplo [5]

[5] Kasukurthi (2018)



3.1.1. Mapas de activación

Esta es la primera estrategia que hemos implementado, el procedimiento es sencillo, añadimos un *hook* a todas las capas ReLU de la red, y copiamos la salida, para analizarla posteriormente.



El problema es evidente, en las primeras capas la función de cada filtro es relativamente estimable, vemos kernels que buscan bordes, otros que reaccionan a algún color, otros que parecen ser derivadas parciales en alguna coordenada; sin embargo, a medida que subimos de nivel, los filtros son cada vez más crípticos y las propiedades que describen cada vez más abstractas, hasta que llega un punto en el que son indescriptibles.

3.1.2. Mapas de calor mediante ventana deslizante

Otra estrategia que se utilizó para tener un conocimiento más informado sobre cómo se está resolviendo el problema fue el uso de una ventana deslizante que va ocluyendo la imagen y vemos el impacto en la salida para la clase que queremos predecir [6].

El procedimiento que hemos definido es sencillo

[6] Zeiler & Fergus (2014)

1. Creamos un generador, que vaya desplazando una ventana (msk) por la imagen con varios tamaños incrementales, y en el area de la ventana cambiamos el contenido por ruido gaussiano ($\mu = 0{,}5, \sigma = 0{,}1$) acotado entre 0 y 1 —los mismos límites que en la imagen—.

2. Definimos una matriz (H) a cero, de altura y anchura igual a la imagen original.

3. Evaluamos el modelo en la imagen y obtenemos el valor de salida en la última capa para la clase deseada s.

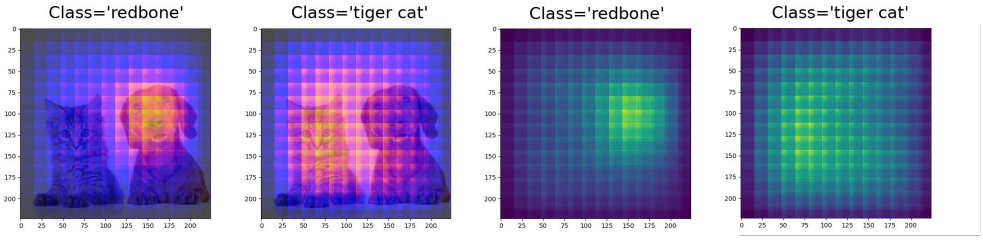
4. Sumamos a H en la misma región de la máscara de oclusión msk.

$$H[msk] += \frac{1}{s}$$

5. Finalmente normalizamos entre 0 y 1 e imprimimos la imagen resultante.

En definitiva, estamos sumando la caída en rendimiento debido a la oclusión de un área sobre la imagen original, esto nos da una idea de qué zonas son más relevantes para la predicción de la clase correcta.

Dependiendo de la clase que midamos en la salida del modelo, el mapa parece discriminar con relativo acierto la posición, aquí un ejemplo donde en la misma imagen aparecen un perro y un gato, con etiquetas en ImageNet *redbone (168)* y *tiger cat (282)* respectivamente.



Sin embargo esta estrategia presenta dos problemas inmediatos, el primero es que la granularidad de la solución, si bien es un primer paso frente a la ceguera absoluta que teníamos en primer lugar, es demasiado gruesa, y no sabemos qué aspectos concretos de la imagen propician la respuesta, únicamente el área en el que se ubican; y el segundo problema, es que computacionalmente es muy costoso, pues el tiempo de ejecución aumenta polinómicamente con la granularidad más fina del resultado —menor ventana o menor desplazamiento—.

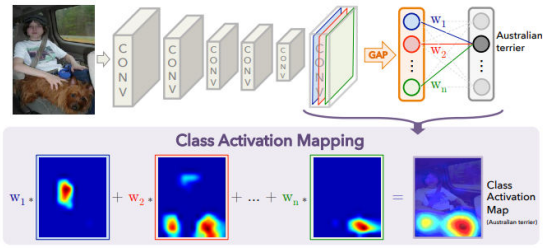
3.2. Utilizando la información del gradiente

3.2.1. Class Activation Maps

Esta técnica es una adaptación de la técnica original[7]. Hay dos diferencias esenciales, la primera es que su modelo es un modelo basado en LeNet definido

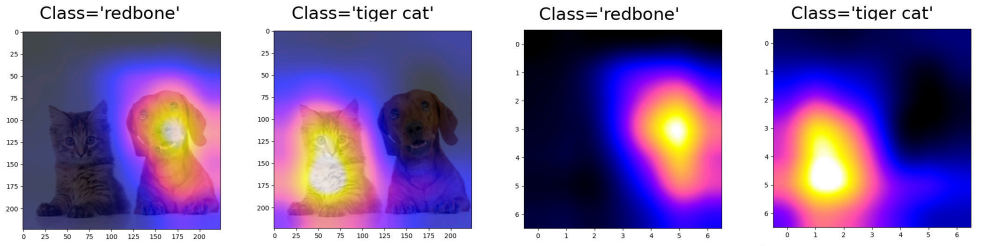
[7] Zou et al. (2016)

ex profeso para esa aplicación que utiliza *Global Average Pooling*:[8]



Donde tras el último bloque de convoluciones se calcula la sumatoria de todos los valores de la matriz de activación para cada capa y se utilizan unos pesos asignados para clasificar y son estos mismos pesos los que utiliza para realizar el *Class Activation Mapping* o *CAM* sumando cada mapa de activación resultado de la última capa de convoluciones ponderado por este peso para determinar el área de interés del objeto de la clase definida, mientras que nosotros —y esta es la segunda diferencia— utilizamos los valores del gradiente para estimar la importancia relativa de cada mapa de activación.

Estas decisiones de diseño nos permiten adaptar esta estrategia a cualquier modelo ya preentrenado y se ejecuta sin sobrecoste computacional, al contrario que el algoritmo de ventana deslizante que describíamos en el apartado anterior. Posteriormente cuando implementemos *guided backpropagation* veremos cómo nos podemos beneficiar de esto para ajustar aún mejor el resultado. A continuación los resultados en las mismas condiciones que para el apartado anterior.



Vemos cómo el resultado es capaz de definir el área de mayor interés para la clasficación y nos da información sobre qué elementos son críticos en la clasficación de la imagen, en este caso, Resnet34 arroja un valor de 9.474 para el perro –la clase más valorada— mientras que el valor de salida para la clase gato es de 6.253. Parece que la cara del perro es un elemento más dominante de la clasificación correcta mientras que el pelo del pecho del gato es el elemento más definitorio para su clase, esto puede ser debido a que el gato es un gato joven y la morfología de la cara no responde suficientemente bien a los ejemplos de gatos adultos en el entrenamiento, o sencillamente que la variabilidad de la cara es mayor o es una característica menos crítica a la hora de diferencias esta raza de gatos del resto de clases; sea como sea empieza a darnos información semánticamente más rica que la que obteníamos por los métodos anteriores.

[8] Lin et al. (2013)

El procedimiento es relativamente sencillo

1. Definimos la función *forwardHook* que engancharemos a la capa ReLU en la pasada hacia delante y tomará la salida de esta y la almacena en una lista

2. Definimos la función *backwardHook* que engancharemos a la capa ReLU en la propagación hacia detrás. Esta función se encarga

   a) Calcula el percentil 50 —o mediana— del gradiente para cada salida de la convolución —almacenado en forwardHook en orden inverso—, esta será la medida que utilizaremos para definir la relevancia de un mapa de activación.

   b) Almacena la lista de *importancias* junto a la lista de mapas de activación

3. Enganchamos ambas funciones a la última función ReLU justo antes de las capas *fully-connected*

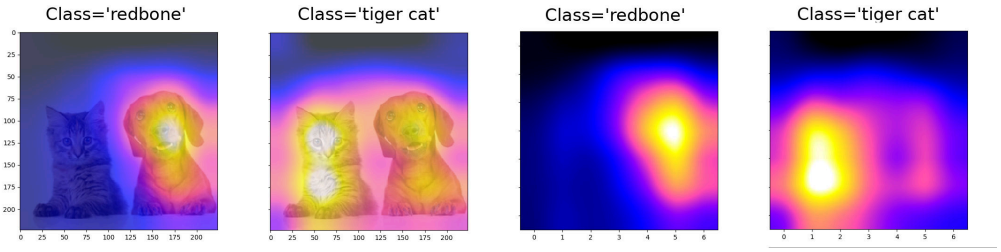
4. Utilizamos el modelo para clasificar nuestra imagen

5. Definimos el gradiente en los nodos de salida —salida de la *fully-connected*— como un vector con todos los valores a cero menos en la posición correspondiente a la clase objetivo que escribimos un uno.

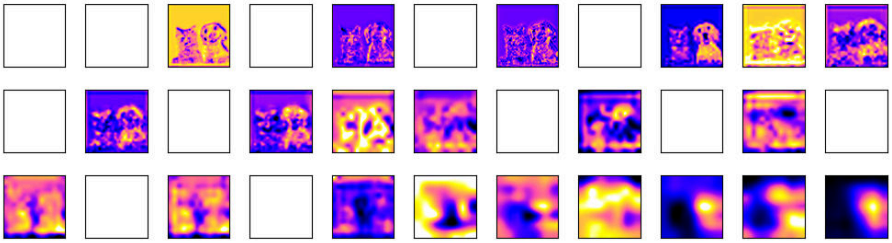
6. Calculamos la propagación hacia atrás

7. Finalmente calculamos la suma ponderada de los mapas de activación, normalizamos entre cero y uno y devolvemos esta matriz como la imagen.

Los resultados en principio resultan satisfactorios. La lógica es sencilla, y es el mismo razonamiento descrito en el artículo sobre guided backpropagation, aquellas características que ayuden a clasificar con más probabilidad una imagen como la clase objetivo tendrán un gradiente positivo, y tendrán mayor capacidad discriminante, al ponderar las convoluciones según el percentil 50 de su gradiente estamos definiendo una métrica, de cómo de generalizadamente contribuye esa convolución a la clasificación final. Desplazando el valor del percentil podemos buscar características más o menos exigentes, este ejemplo es con el percentil 90, siendo una métrica menos exigente.

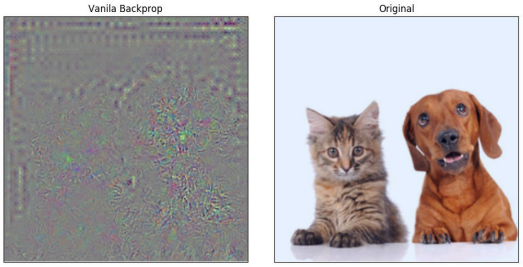


Podemos visualizar también este *heatmap* en distintos niveles del modelo, sin embargo, el valor informativo decrece pues debido a las transformaciones intermedias la influencia en el resultado final puede cambiar de signo, por lo tanto si observamos el heatmap de una capa distinta a la última el valor informativo disminuye enormemente.



3.2.2. Guided Backpropagation

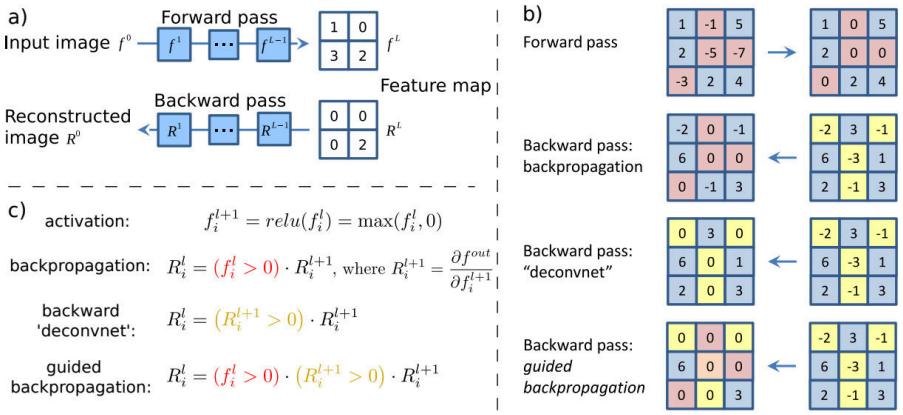
Una idea interesante para visualizar el funcionamiento de las redes es el uso de la propagación hacia atrás hasta la capa de la imagen, como si esta fuera también un volumen de pesos minimizable respecto a alguna función, sin embargo, si intentamos visualizar esto el resultado que obtenemos no es en absoluto alentador.



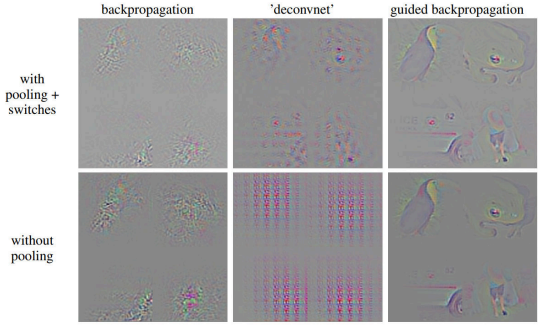
Conociendo la imagen de salida podríamos aventurarnos a estimar que en el area del perro se concentran puntos de mayor activación, pero es prácticamente un salto de fe y la información es inutilizable a nivel práctico.

Springenberg et al. (2014)[9] proponen en su artículo *Striving for Simplicity: The All Convolutional Net* un método novedoso para la visualización del funcionamiento de CNN, basado en la idea de *deconvnet*[10].

[9] Springenberg et al. (2014)
[10] Zeiler & Fergus (2014)



En ambos métodos se intercepta y se modifica el gradiente en cada capa de activación y la diferencia radica en la fórmula, en *deconvnet* el gradiente en la capa ReLU es igual al gradiente original excepto en aquellas posiciones donde éste sea menor que 0, que pasa a ser 0, así nos garantizamos no propagar un gradiente negativo, citando al artista "*a fin de visualizar las partes de una imagen que son más discriminantes para una unidad dada en la red*"; *guided backpropagation* lleva aún más lejos esta idea anulando —poniendo a 0— en el gradiente no sólo aquellos valores que sean 0 en éste, sino también los valores que fueron cero en la pasada hacia delante. Aquí una comparación de los resultados extraído del artículo.



"El procedimiento se puede resumir de la siguiente manera, dado un mapa de caraterísticas de alto nivel, tanto *deconvnet* como *guided backpropagation* invierten el flujo de datos de la CNN, yendo desde la activación de neuran en una capa dada hasta la imagen. Típicamente una sola neurona se deja a un valor mayor distinto de cero en el mapa de características. Finalmente, la imagen reconstruida final muestra las partes de la imagen de entrada que activan a esta neurona con mayor intensidad, y por lo tanto tienen mayor capacidad discriminante."[11]

En nuestro caso vamos a aplicarlo sobre las 1000 neuronas de salida que se utilizan para la clasificación, pero si tuvieramos interés en mapear el area

[11] Springenberg et al. (2014)

de la imagen original que ha estimulado una neurona en la salida de una capa de activación, o queremos ver qué tipos de características se recogen en una convolución concreta, podemos definir el gradiente distinto de cero en nuestra neurona objetivo y propagar el error hacia atrás. El artículo en cuestión profundiza más en la justificación teórica del funcionamiento y en la pérdida de información que se produce en la propagación hacia atrás manipulando el gradiente de la forma descrita, y refiero a su lectura para dar paso al análisis de nuestra implementación concreta.

Uno de los problemas a los que se enfrenta este algoritmo, y que ya adelantábamos en la imagen anterior al diferenciar entre *pooling + switches* o *no pooling* es el problema de recuperar la neurona concreta que pasó el filtro de *max pooling*, para esto en el paper sugieren una estructura de datos adicional que almacena esta correspondencia, sin embargo, gracias al uso de *autograd*[12], la tecnología subyacente en la propagación hacia atrás implementada en el módulo de diferenciación automática en PyTorch, no tenemos que implementar esta estructura, pues los propios sensores llevan el historial de estos fenómenos y la implementación es mucho más inmediata.

El funcionamiento del algoritmo entonces, extendiendo el anterior, sólo necesita modificar la función *backwardHook* modificando el gradiente de la siguiente manera —utilizando los mismos nombres de variables que en la imagen sobre la fórmula—, tomamos f_i^l de la pila que fuimos creando en *forwardHook*, capturamos el gradiente de salida R_i^{l+1} que se nos pasa por parámetro y devolvemos lo siguiente.

$$R_i^l = (f_i^l > 0) \cdot (R_i^{l+1} > 0) \cdot R_i^{l+1}$$

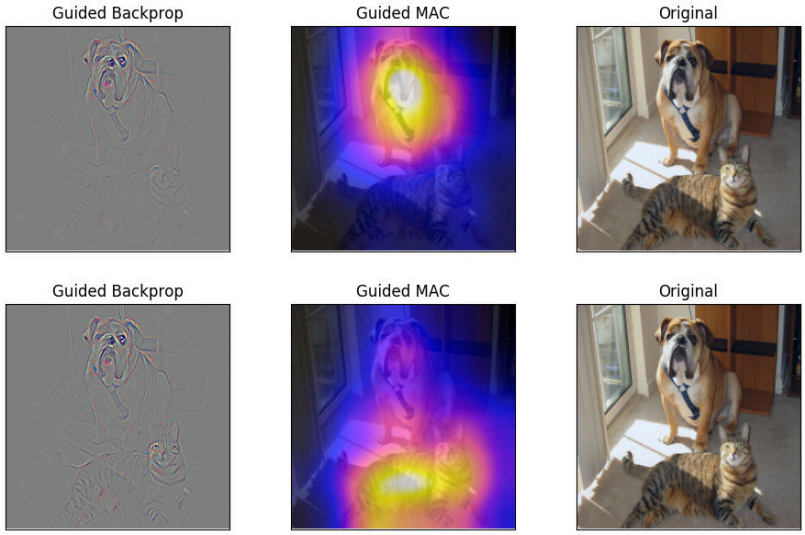
Continuando con el ejemplo anterior aquí tenemos la salida definiendo como no cero (1) la clase correspondiente a *redbone* o a *tiger cat*.

Se aprecia perfectamente cómo la clase dominante —*redbone* con valor en la neurona de salida 9.474— tiene más elementos discriminantes que la del gato —salida de 6.253—, por eso mientras que la clase perro sí es capaz de ocultar con mayor éxito las características del gato.

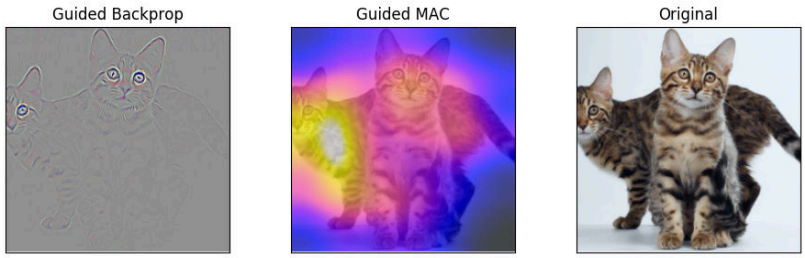
[12]Dougal (2016)

3.2.3. Algunos ejemplos más y análisis

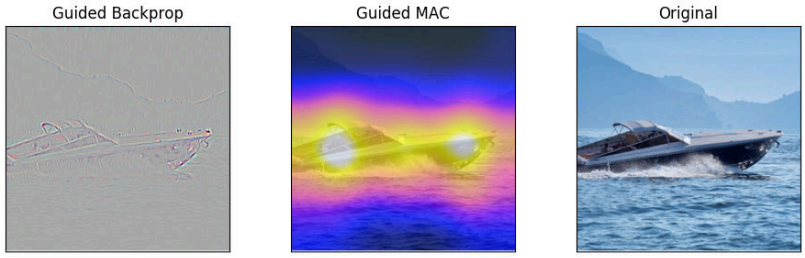
Todos los ejemplos que muestro a continuación han sido correctamente clasificados por Resnet34 y es en la propia etiqueta de salida en la que pongo el valor no cero para inicializar el gradiente salvo en este primer caso.



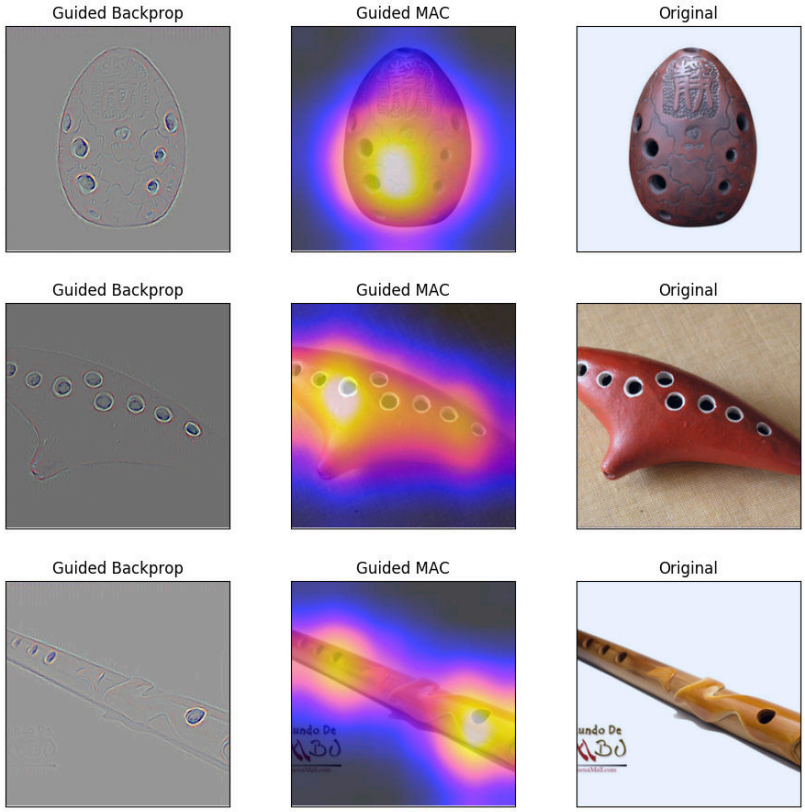
En este caso observamos el mismo fenómeno que en el ejercicio anterior, el modelo es capaz de discriminar al gato del perro en un sentido pero no en el otro, de nuevo la clase más botada *bull mastiff (243)* con 12.187 frente a *tiger cat* con 11.625, mucho más cerca esta vez.



La forma en la que aparece MAC parece indicar que tiene mayor facilidd clasificando gatos en una vista de perfil —al igual que en el ejemplo anterior— que de frente.

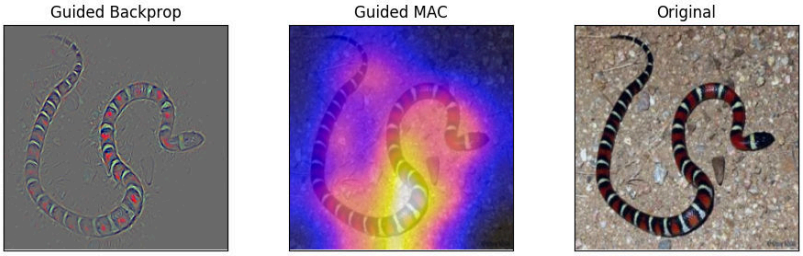


Esta imagen es interesante porque se aprecia cómo no sólo utiliza el area del barco para predecir la clase, sino que el perfil de la costa también aparece activo, seguramente se deba a que esta información contextual es habitual en fotos de lanchas rápidas.



Pese a que los elementos más llamativos tanto en las ocarinas como en la flauta sean los agujeros, Resnet es capaz de discriminarlos con éxito, es interesante ver cómo en las ocarinas le presta mucha más atencion a la textura barrosa que a las vetas de la madera, es posible que el barro sea el material habitual de la ocarina.

En este caso vuelve a ser llamativo la capacidad para separar en la *MAC* al perro del gato tumbado, bordeando a éste segundo la zona de mayor intensidad, seguramente por la diferencia de texturas de pelo y color.



De este último caso cabe destacar la calidad con la que *guided backpropagation* es capaz de definir a la serpiente frente al suelo.

4. Valoración de los resultados

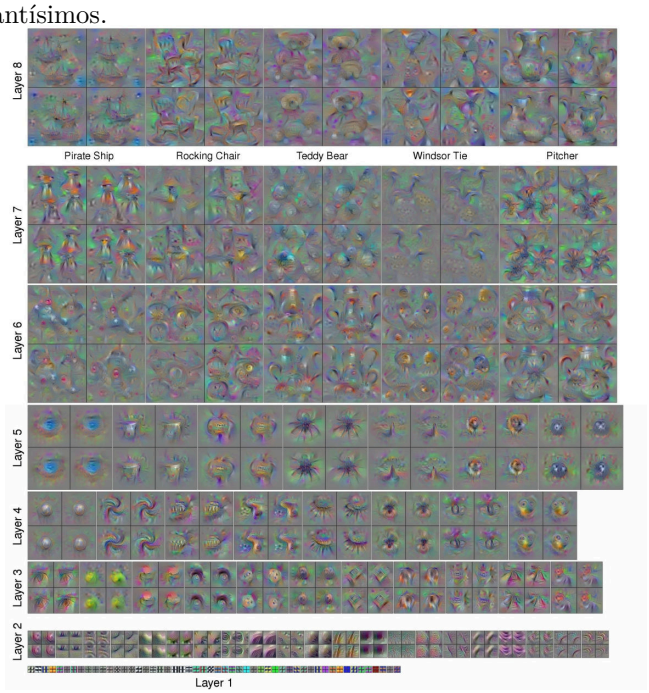
Ambas estrategias de visualización nos han resultado gratamente sorprendentes, los resultados efectivamente ayudan a la comprensión del funcionamiento de las redes, y un análisis más concienzudo, donde se extraiga información más diversa redundará seguro en la capacidad para generar mejores modelos, es evidente que la visualización de los problemas y las soluciones son una guía indispensable para la solución de cualquier problema ingieneril, y este tipo de estrategias son un salto cualitativo en esa dirección.

Es muy interesante ver cómo efectivamente el uso de la información local supone una ventaja tremenda para las CNN frente a los algoritmos clásicos que intentaban resolver este tipo de problemas, y aquí, especialmente en los *MAC*s se aprecia cómo no sólo está utilizando la información que proporciona la localidad de los datos sino que de alguna manera esa información permanece oculta en la red. Este tipo de acercamientos puede ser una primera etapa para el entrenamiento de sistemas de segmentación no supervisados —sin las coordenadas de la segmentación—, y estos resultados, combinados quizás con algoritmos de detección de bordes pueden ayudar a generar instancias de imágenes con atributos de segmentación para mejorar el rendimiento de modelos que se enfrentan a este tipo de problemas donde los datos de entrenamiento son tan costosos por lo tedioso de su etiquetado.

En definitiva son unos resultados interesantes y ejemplos del tipo de investigaciones que se están desarrollando en esta materia a fin de arrojar algo de luz dentro de estos modelos de caja negra.

5. Trabajos futuros

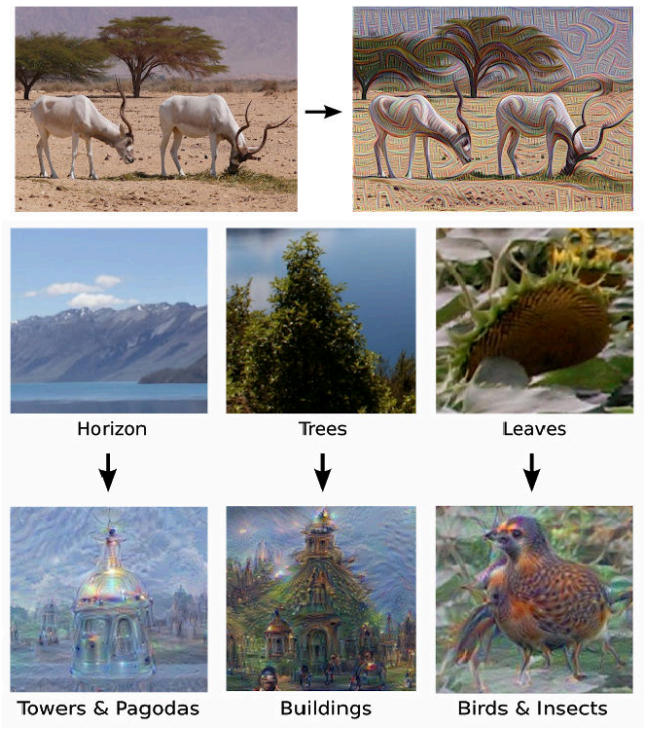
Guided backpropagation es un algoritmo interesante y que aún no ha agotado su utilidad. Un ejemplo de aplicación de este algoritmo es la visualización de kernels concretos de capas de niveles altos que no permiten una comprensión clara de su función debido a la alta abstracción de las características que codifica. Yosinki (2015)[13] utiliza este algoritmo para dada una semilla inicial con ruido gaussiano va obteniendo el gradiente en la imagen de entrada sumándolo a la entrada, aplicando un suavizado gaussiano y volviendo a repetir, después de suficientes iteraciones elicita características en las imágenes que describen de alguna manera las características que esas convoluciones codifican. Esto es aplicado tanto en las capas final, correspondiendo a clases concretas, como a convoluciones intermedias que codifican patrones más sencillos. Los resultados son interesantísimos.



Por otro lado, con una estrategia prácticamente idéntica y en una línea más creativa, Google DeepDream[14] utiliza como semilla imágenes con contenido en vez de ruido gaussiano va aplicado el mismo procedimiento de forma iterativa

[13] Yosinki et al. (2015)
[14] Mordvintsev (2015)

hasta que la imagen se va transformando, aunque sea una aplicación trivial nos da información sobre el contenido de las redes.



6. Bibliografía

1. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105). https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

2. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778). https://arxiv.org/abs/1512.03385

3. Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806. https://arxiv.org/abs/1412.6806

4. Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2921-2929). http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf

5. Kasukurthi, N. (2018). Visualising CNN Models Using PyTorch. Intel Academy. https://software.intel.com/en-us/articles/visualising-cnn-models-using-pytorch

6. Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham. https://arxiv.org/abs/1311.2901

7. Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400. https://arxiv.org/pdf/1312.4400.pdf

8. Dougal, M. (2016) Modeling, Inference and Optimization with Composable Differentiable Procedures (Tesis doctoral). Capitulo 4 : Autograd. https://dougalmaclaurin.com/phd-thesis.pdf

9. Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., & Lipson, H. (2015). Understanding neural networks through deep visualization. arXiv preprint arXiv:1506.06579. http://yosinski.com/media/papers/Yosinski__2015__ICML_DL__Understanding_Neural_Networks_Through_Deep_Visualization__.pdf

10. Mordvintsev, A. (2015). Inceptionism: Going Deeper into Neural Networks. Google AI Blog. https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html