XXVII JORNADAS DE JOVENS PESQUISADORES
23 A 25 DE OUTUBRO DE 2019
A ciência e a tecnologia na produção de inovação e transformação social

17. Ciencia, Tecnología, e Innovación

Clasificación de objetos en imágenes usando redes convolucionales

Autor: Rubio, Ignacio; ignacio.rubio@itt.unnoba.edu.ar
Coautor: Fondato, Germán; german.fondato@itt.unnoba.edu.ar
Orientador: Esnaola, Leonardo; leonardo.esnaola@itt.unnoba.edu.ar
Instituto de Tecnología y Transferencia (ITT)
Universidad Nacional del Noroeste de la provincia de Buenos Aires (UNNOBA)

Resumen

La clasificación de objetos en imágenes utilizando redes convolucionales es una técnica para el análisis de imágenes que utiliza *Deep Learning* o Aprendizaje Profundo. La misma consiste determinar a qué clase, de un conjunto de clases dado, pertenece el objeto que se encuentra en la imagen. Para la tarea de clasificación, dicho elemento a clasificar tiene el foco principal de la imagen y es el único objeto en la misma.

Las redes convolucionales son un tipo especial de red neuronal, que realizan la operación matemática de convolución durante su operación. Las mismas suponen explícitamente que las entradas son imágenes, lo que permite codificar ciertas propiedades en su arquitectura. Esto permite una implementación y funcionamiento mucho más eficiente reduciendo enormemente la cantidad de parámetros de la red. [3]

En el presente documento se realiza una comparativa entre algunos modelos de renombre y ampliamente utilizados como VGG16, VGG19 y ResNet, midiendo los resultados obtenidos sobre el *dataset* CIFAR100 [9]; mediciones en función de la exactitud alcanzada por el modelo, la reducción del error lograda en el reconocimiento, y los tiempos necesarios de ejecución (siempre en un mismo ambiente de pruebas).

Los modelos utilizados para la clasificación de objetos son la base para otras tareas referentes al análisis de imágenes: detección y localización, y segmentación. La motivación del presente trabajo es comparar y seleccionar el modelo más apropiado para realizar las actividades de análisis de imágenes mencionadas. Con esto, nos referimos al más eficiente en un determinado ambiente de pruebas, con recursos computacionales definidos, tiempos de pruebas establecidos y objetivos de aplicación particulares para el uso del modelo.

Palabras clave: red convolucional, clasificación de imágenes, *Deep Learning*.

Introducción

Existen varios enfoques en el dominio del análisis de imágenes usando redes neuronales convolucionales [5] (figura 1), estos son:

- *Clasificación*: dada una imagen se debe determinar la clase particular a la que pertenece (es un perro, es una persona, es un automóvil, etc.).
- *Localización y detección*: en este caso son varios los objetos a clasificar; además hay que determinar su posición en la imagen dibujando un recuadro que contenga el objeto.
- *Segmentación*: consiste en crear máscaras individuales para cada objeto de la imagen. La segmentación puede ser semántica o de objetos.

Todas estas técnicas tienen en común que utilizan *Deep Learning* en su implementación. Como se mencionó anteriormente, el Aprendizaje Profundo utiliza redes neuronales; una buena definición de este concepto es la siguiente:

Conjunto de nodos, llamados neuronas artificiales, organizadas en capas. Cada nodo tiene pesos y sesgos que se pueden modificar mediante el entrenamiento de la red en función de obtener los resultados esperados. Cada neurona recibe algunas entradas (datos) provenientes de otras ubicadas en una capa anterior, realiza un producto punto y envía su salida a las neuronas de la siguiente capa. Existiendo en la estructura de la red una *capa de entrada* que obtiene los datos desde su origen, una o varias *capas intermedias* u *ocultas*, y una *capa de salida* que genera el resultado de procesar una determinada entrada [5][6]. La figura 2 ilustra este concepto.

Este documento se enfoca en la primera técnica, la clasificación, los modelos de redes convolucionales existentes y su implementación en Python utilizando la

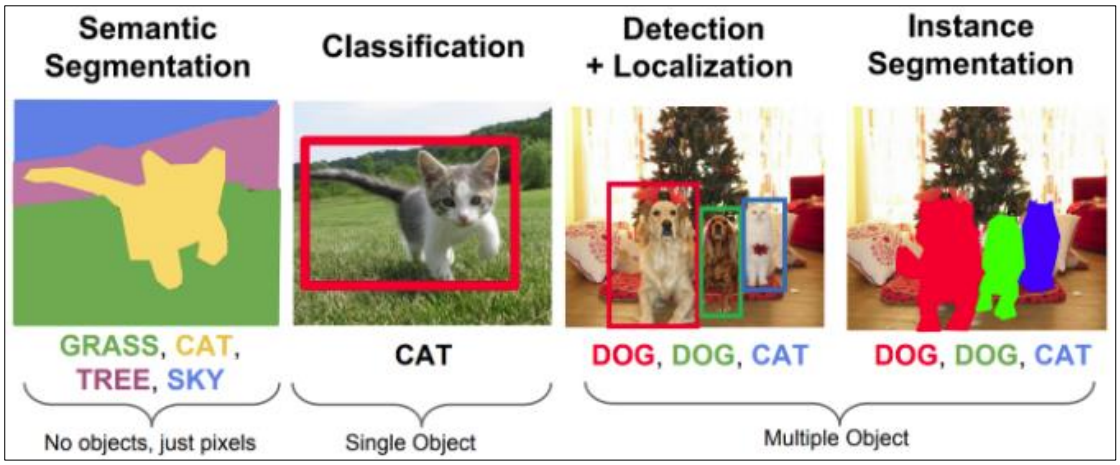

Figura 1. Técnicas de análisis de imágenes

librería de TensorFlow para posteriormente realizar una comparativa de dichos modelos funcionando bajo las mismas condiciones de operación (recursos hardware, software y *dataset*).

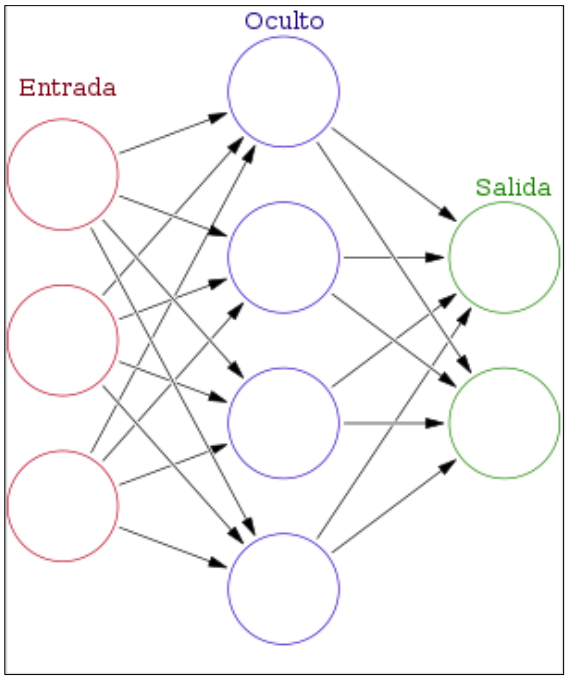

Figura 2. Estructura de una red neuronal

El último concepto que queda por definir, de manera más detallada, es el de red convolucional:

Las redes convolucionales son un tipo de redes neuronales aplicadas a matrices bidimensionales que resultan muy efectivas en tareas de visión artificial [3]. Este tipo de arquitectura transforma las imágenes de entrada en un volumen de salida mediante la utilización de un conjunto especial de capas, entre las que se deben destacar dos:

- *Capa convolucional*: es el corazón de este tipo de redes. Realiza la convolución entre la matriz de entrada y los n filtros (o *kernel*) para generar un mapa de características de la imagen.
- *Capa de reducción*: reduce la cantidad de parámetros al seleccionar las características más representativas del mapa obtenido en la capa de convolución, descartando el resto.

Objetivos

El objetivo del trabajo es realizar la comparativa entre diferentes modelos, de uso extendido y resultados comprobados, para la clasificación de objetos en imágenes, bajo las mismas condiciones de operación: *dataset* empleado, software, librerías utilizadas y hardware disponible.

Se realizaron varias pruebas y mediciones para una posterior comparación del desempeño sobre un mismo dataset: CIFAR100. Dicho *dataset* está formado por 60000 imágenes a color de 32x32 pixeles agrupadas en 100 categorías, con 600 imágenes cada una [9].

Los modelos utilizados fueron VGG16, VGG19 y ResNet. Para todos ellos se buscó el mejor desempeño de cada uno en función de los recursos y limitantes del entorno de trabajo con que se cuenta. El desempeño de cada modelo se ajusta modificando diferentes parámetros de los mismos (épocas de entrenamiento, número de filtros, funciones de activación, normalización de los datos, etc.). El ajuste

de dichos parámetros se consigue haciendo varias pruebas sobre el mismo modelo y modificando uno a uno los valores de dichos parámetros en función de los resultados obtenidos en pruebas anteriores. Es en base a los mejores resultados obtenidos para cada modelo que se realiza la comparativa final, plasmada en este documento.

La elección de los modelos mencionados se realizó en función de su popularidad y uso en tareas de clasificación de imágenes, son los modelos más nuevos y con mejores resultados obtenidos en las últimas ediciones de ILSVRC, e incluso implementan mejoras sobre otras arquitecturas de renombre (por ejemplo, las redes VGG modifican el *kernel* usado por AlexNet mejorando sus resultados) [4][7].

Materiales y métodos

A continuación se detallan los recursos de hardware de la computadora utilizada en los experimentos:

- Disco rígido: HDD 500GB
- RAM: 16GB
- Placa de video: NVIDIA GeForce GTX 1060 6GB

Las herramientas de software utilizadas son las siguientes:

- SO: Windows 10 Pro
- Python 3.7 (plataforma Anaconda)
- API: TensorFlow con soporte para funcionar sobre GPU + Keras (librería de alto nivel que funciona sobre TensorFlow).
- IDE: Jupyter Notebook 6.0.0
- NVIDIA CUDA Toolkit 10.1 + NVIDIA cudNN

El método consistió en probar los diferentes modelos modificando uno a uno diferentes parámetros para conseguir el mejor rendimiento posible de cada uno. Estos parámetros son:

- Profundidad: cantidad de capas intermedias de la red. Solo en el caso de ResNet (VGG16 y VGG19 tiene una profundidad definida).
- Normalización: los valores de los vectores de datos que representan las imágenes de entrada se ajustan a un rango de valores favorables para el entrenamiento de las redes, entre 0 y 1.
- Número de filtros de entrada de la capa convolucional (solo para ResNet).
- Función de optimización: usadas para el ajuste de los pesos de las neuronas de la red.
- Función de activación: define la salida de un nodo dada una entrada o un conjunto de entradas.

El *dataset* utilizado, CIFAR100, se describió anteriormente. Ahora, es necesario mencionar que, de las 600 imágenes de cada categoría se reservan

500 para el proceso de entrenamiento de la red y las 100 restantes para evaluar el desempeño de la misma. Esta subdivisión entre entrenamiento y evaluación está definida de forma fija por el propio dataset.

A continuación se describen de manera muy breve las tres arquitecturas de redes convolucionales utilizadas:

Configuración de VGG16 (figura 3) [1]:

- Capas convolucionales (*kernel* de dimensiones 3x3)
- Capa de reducción MaxPooling (de *kerel* de dimensiones 2x2)
- Tres capas completamente conectadas al final.
- 16 capas en total (no se consideran las capas de reducción).
- Número de parámetros: 34.006.948

VGG19 (figura 4) es una variante del modelo anterior que agrega tres capas convolucionales, incrementando la profundidad de la arquitectura y el número de parámetros entrenables [1].

- Número de parámetros: 39.316.644

La idea de ResNet, también llamada *Red Neuronal Residual*, se basa en aumentar el número de capas introduciendo conexiones residuales con una capa identidad (figura 5). Esta capa pasa directamente a la siguiente, mejorando el proceso de aprendizaje [2][8].

En comparación con los dos modelos anteriores donde las salidas de una capa pasan a la siguiente capa inmediata, aquí la capa identidad rompe ese modelo secuencial.

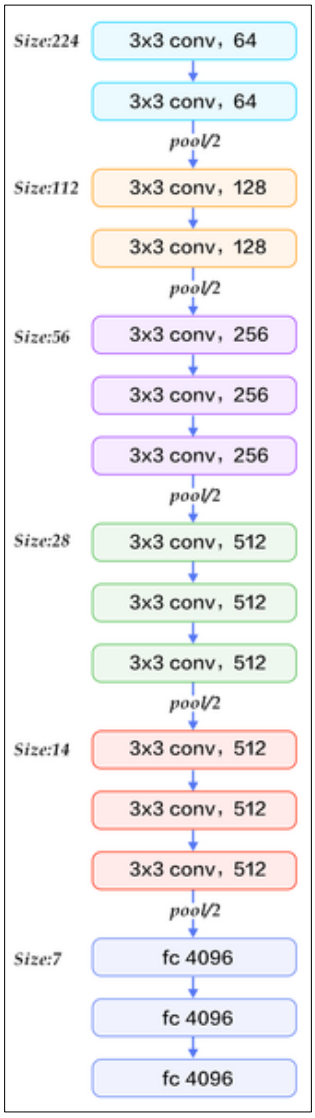

Figura 3. Arquitectura VGG16

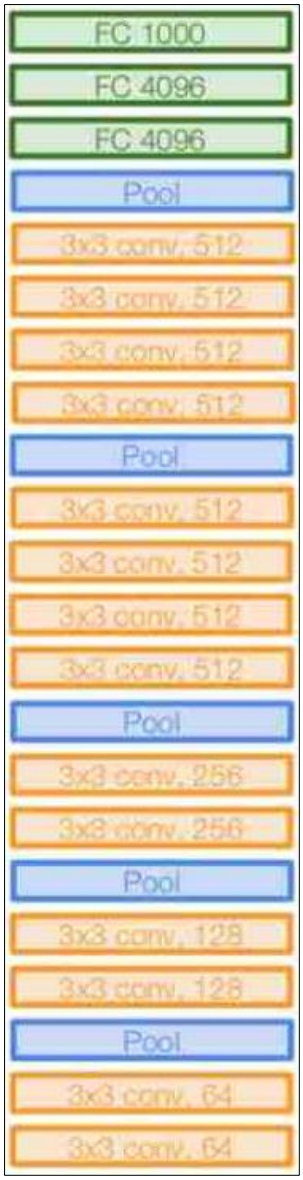

Figura 4. Arquitectura VGG19

El algoritmo para realizar las pruebas, más allá del modelo de red usado en cada momento, tiene la siguiente estructura:

1. Obtención del *dataset*: el mismo está incluido en la API de Keras. Se realiza la importación de la librería necesaria para su uso, posteriormente se dividen los datos de entrenamiento y prueba, y se normalizan dichos datos usando *z-score* como método de normalización.
2. Se define la función para ajustar la taza de entrenamiento: la misma disminuye el valor lr o taza de aprendizaje (*learning rate*) a medida que avanzan las épocas de entrenamiento.
3. Se utiliza *Data Argumentation*: conjunto de técnicas para extender las imágenes de entrenamiento. Se genera a partir de las imágenes dadas, otras nuevas, rotadas n cantidad de grados, invertidas en alguno de sus ejes, con cambios de color, etc.

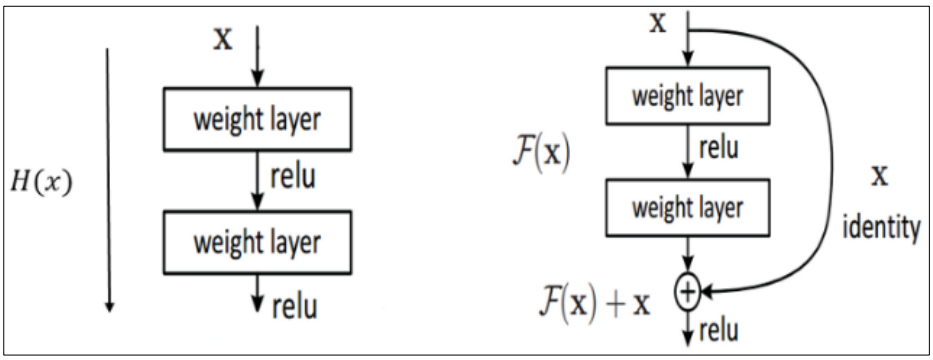

Figura 5. Capa residual del modelo ResNet

4. Se define el modelo a utilizar y se lo compila junto con las funciones de activación y pérdida elegidas.
5. Se entrena dicho modelo durante un número definido épocas.
6. Se realiza la evaluación de la red obtenida midiendo la exactitud de nuevas predicciones.

A continuación se resumen los valores de parámetros y funciones utilizadas en cada uno de los modelos finales:

VGG16:

- Épocas de entrenamiento: 250
- Función de activación: ReLU
- Función de optimización: SGD
- Taza de entrenamiento: 0.1

VGG19:

- Épocas de entrenamiento: 250
- Función de activación: ReLU
- Función de optimización: SGD + Nesterov
- Taza de entrenamiento: 0.1

ResNet:

- Épocas de entrenamiento: 225
- Función de activación: ReLU
- Función de optimización: Adam
- Taza de entrenamiento: 0.001 + lr_scheduler (función que disminuye este parámetro con el paso del tiempo).

Para los tres modelos se usaron las siguientes estrategias de *Data Argumentation*:

- *Rotation Range*: 15
- *Width Shift Range*: 0.1
- *Height Shift Range*: 0.1
- *Horizontal Flip*

Resultados

La siguiente tabla (tabla 1) muestra las mediciones obtenidas en cuanto a *exactitud*, *error* obtenido y *tiempo de entrenamiento* (para lograr el mejor resultado) para cada una de las redes probadas:

Modelo	Exactitud	Error	Tiempo de ejecución
VGG16	36,77%	7,06	2,78 hs.
VGG19	28,47%	7,84	3 hs.
ResNet	73,3%	1,78	10,6 hs.

Tabla 1. Comparativa entre VGG16, VGG19 y ResNet

Se observa claramente que los mejores resultados los obtiene la arquitectura ResNet. Si bien el tiempo de entrenamiento requerido es mayor que en los otros dos casos, también lo es la exactitud alcanzada y, el objetivo del entrenamiento es llevar esa medición lo más cercano de 100%, valor utópico que significaría futuras predicciones sin ningún error.

Los siguientes gráficos muestran la evolución de la exactitud de cada uno de los modelos:

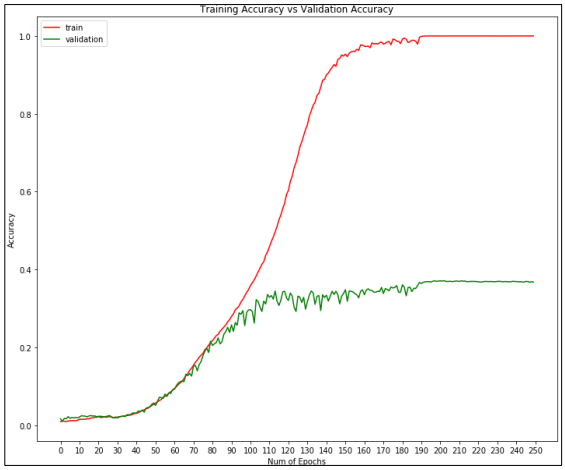

Figura 6. VGG16: Precisión del entrenamiento (rojo) vs. Precisión en la evaluación (verde)

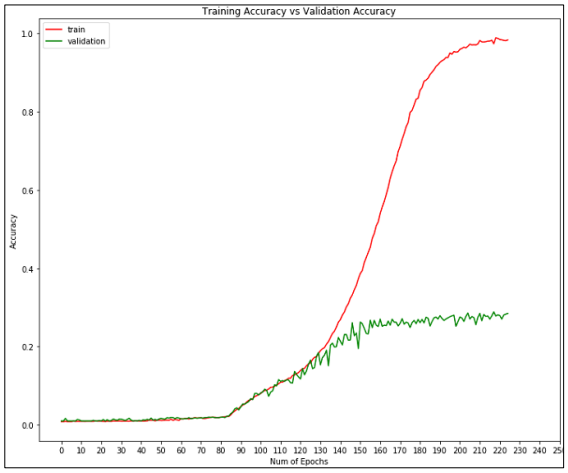

Figura 7. VGG19: Precisión del entrenamiento vs. Precisión en la evaluación

Figura 8. ResNet: Precisión del entrenamiento vs. Precisión en la evaluación

Se observa de los gráficos que el principal problema es la capacidad de generalización de los modelos, es decir, la exactitud en la clasificación de nuevas imágenes. Esto está representado por la función de color verde, mientras que la función de color rojo refiere a la exactitud con los datos de entrenamiento. Que ambas funciones se alejen tanto no es un buen síntoma, significa que si bien la exactitud es muy buena durante el entrenamiento, la eficiencia de la red no lo será para su uso futuro, con datos no vistos previamente. Ambas funciones deberían ser lo más parecidas posibles. Note que en los tres gráficos la función roja alcanza aproximadamente el valor ideal 1 (100%), pero en el gráfico de ResNet la función verde, que mide la evaluación de la red, es la más cercana, alcanzando el 73,3%. Esto es la exactitud real alcanzada por ese modelo con esa arquitectura.

Conclusiones

Las conclusiones son las observadas de los resultados obtenidos. Claramente ResNet es la red que da mejores resultados, pero es necesario hacer una serie de consideraciones. Primero, ResNet obtiene mejores indicadores porque es una arquitectura que se desarrolló en función de los problemas detectados en otras ya existentes, entre ellas VGG.

Segundo, los resultados obtenidos para las diferentes redes podrían tener variaciones sobre otros *datasets*, distinto hardware y, probablemente, usando otras librerías de aprendizaje de máquina. Sin embargo, los resultados alcanzados siguen siendo representativos para la comparativa propuesta.

Y tercero, es probable que con más datos de entrenamiento los modelos (principalmente VGG16 y VGG19) mejoren su potencia de generalización. CIFAR100 tiene para cada categoría tan solo 500 imágenes de entrenamiento, que pueden no ser suficientes para todos los modelos de redes convolucionales (incluso utilizando *Data Argumentation*). Otros *datasets* poseen una cantidad mucho mayor de datos de entrenamiento; CIFAR10, por ejemplo, para cada una de sus diez categorías posee 5000 imágenes [9], diez veces más que para el *dataset* utilizado en estas pruebas.

Por último, si bien estas mediciones no son prometedoras usando los modelos por defecto, principalmente para las dos redes VGG, se podrían implementar mejoras utilizando dichos modelos como base, por ejemplo, manteniendo la estructura de capas de convolución y reducción e intercalando capas *Dropout* y *Batch Normalization*, para obtener mejores generalizaciones.

Bibliografía

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun (2015). Deep Residual Learning for Image Recognition.

[2] Simonyan Karen y Zisserman Andrew (2015). Very Deep Convolutional Networks for large-scale image recognition

[3] CS231N Staff. Convolutional Neural Networks for Visual Recognition. Recuperado de http://cs231n.github.io/convolutional-networks/

[4] Deshpande Adit. The 9 Deep Learning Papers You Need To Know About. Recuperado de: https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

[5] Agarwal Rahul (2018). Object Detection: An End to End Theoretical Perspective. Recuperado de https://towardsdatascience.com/object-detection-using-deep-learning-approaches-an-end-to-end-theoretical-perspective-4ca27eee8a9a

[6] Wikipedia (2019). Red neuronal artificial. Recuperado de https://es.wikipedia.org/wiki/Red_neuronal_artificial

[7] CV-Tricks. ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks Recuperado de https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

[8] Burgal Jesús (2018). Deep Learning básico con Keras (Parte 4): ResNet. Recuperado de: https://enmilocalfunciona.io/deep-learning-basico-con-keras-parte-4-resnet/

[9] Krizhevsky Alex (2017). CIFAR-10 and CIFAR-100 Datasets. Recuperado de https://www.cs.toronto.edu/~kriz/cifar.html